

The Use of Games and Crowdsourcing for the Fabrication-aware Design of Residential Buildings

at the Digital Design Unit
at the Faculty of Architecture
of Technische Universität Darmstadt

submitted in fulfillment of the requirements for the
degree of Doktoringenieur
(Dr.-Ing.)

**Doctoral thesis
by Anton Savov**

First assessor: Prof. Dr.-Ing. Oliver Tessmann
Second assessor: Prof. Dr.-Ing. Martin Knöll



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Darmstadt 2021 (year of the viva voce)

Savov, Anton: The Use of Games and Crowdsourcing for the Fabrication-aware Design of Residential Buildings

Darmstadt, Technische Universität Darmstadt,

Year thesis published in TUpriints 2022

URN: urn:nbn:de:tuda-tuprints-206970

URI: <https://tuprints.ulb.tu-darmstadt.de/id/eprint/20697>

Day of the viva voce: 29.11.2021

Published under CC BY-SA 4.0 International

<https://creativecommons.org/licenses/>

ABSTRACT

The Use of Games and Crowdsourcing for the Fabrication-aware Design of Residential Buildings

Anton Savov

State-of-the-art participatory design acknowledges the true, ill-defined nature of design problems, taking into account stakeholders' values and preferences. However, it overburdens the architect, who has to synthesize far more constraints into a one-of-a-kind design. Generative Design promises to equip architects with great power to standardize and systemize the design process. However, the common trap of generative design is trying to treat architecture simply as a tame problem. In this work, I investigate the use of games and crowdsourcing in architecture through two sets of explorative questions. First, if everyone can participate in the network-enabled creation of the built environment, what role will they play? And what tools will they need to enable them? And second, if anyone can use digital fabrication to build any building, how will we design it? What design paradigms will govern this process? I present a map of design paradigms that lie at the intersections of Participatory Design, Generative Design, Game Design, and Crowd Wisdom. In four case studies, I explore techniques to employ the practices from the four fields in the service of architecture. Generative Design can lower the difficulty of the challenge to design by automating a large portion of the work. A newly formulated, unified taxonomy of generative design across the disciplines of architecture, computer science, and computer games builds the base for the use of algorithms in the case studies. The work introduces Playable Voxel-Shape Grammars, a new type of generative technique. It enables Game Design to guide participants through a series of challenges, effectively increasing their skills by helping them understand the underlying principles of the design task at hand. The use of crowdsourcing in architecture can mean thousands of architects creating content for a generative design system, to expand and open up its design space. Crowdsourcing can also be about millions of people online creating designs that an architect or a homeowner can refer to increase their understanding of the complex issues at hand in a given design project and for better decision making. At the same time, game design in architecture helps find the balance between algorithmically exploring pre-defined design alternatives and open-ended, free creativity. The research reveals a layered structure of entry points for crowd-contributed content as well as the granular nature of authorship among four different roles: non-expert stakeholders, architects, the crowd, and the tool-makers.

Keywords: crowdsourcing, games, residential buildings, fabrication-aware, participation, generative design, voxel shape grammars, Minecraft

To my parents Blaguina and Matey.

Declaration

Declaration of Originality

I hereby certify that this thesis, to the best of my knowledge, is my own work. All sources and assistance with the preparation of this thesis have been acknowledged.

Author Attribution Statement

This thesis includes published material from the following peer-reviewed list. Where the material is used directly, it is cited, adapted, and integrated according to the thesis chapters.

Savov, A., Tessmann, O. and Nielsen, S. A. (2016), “Sensitive Assembly: Gamifying the Design and Assembly of Facade Wall Prototypes”, in: *International Journal of Architectural Computing* 14.1, pp. 30–48

Savov, A., Buckton, B. and Tessmann, O. (2016), “20,000 Blocks: Can Gameplay Be Used to Guide Non-Expert Groups in Creating Architecture?”, in: *ACADIA 2016: POSTHUMAN FRONTIERS: Data, Designers, and Cognitive Machines [Proceedings of the 36th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)]*, Ann Arbor, pp. 24–33

Savov, A. and Tessmann, O. (2017), “Introduction to Playable Voxel-Shape Grammars”, in: *Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, Cambridge, pp. 534–543

Savov, A., Winkler, R. and Tessmann, O. (2020), “Encoding Architectural Designs as Iso-surface Tilesets for Participatory Sculpting of Massing Models”, in: *Impact: Design With All Senses*, ed. by Gengnagel, C. et al., Cham: Springer International Publishing, pp. 199–213

Acknowledgements

Writing this acknowledgment note was one of the most humbling experiences I have ever had. I am so honored and touched to have had the chance, luck, and pleasure to have everyone named below in my life and to have been part of all these amazingly inspiring environments.

First of all, I would like to thank Prof. Oliver Tessmann for the great environment at the Digital Design Unit at TU Darmstadt and the valuable advice that allowed me to develop my work. I certainly could not have done this work anywhere else. You were a great Ph.D. supervisor and a mentor, supporting me through all ideas, dead ends, and successes. You gave me the right input at the right time and helped me see my work as part of a larger whole, with its connections in a network of people, concepts, and projects.

Thank you to my second supervisor Prof. Martin Knöll for the fresh perspective and very insightful assessment of the research that helped sharpen the narrative, frame the discussion and clean up the methodology. I appreciate the extra push to look at the details and bring this dissertation to a version I am proud of.

Ben, dear friend, thank you for the many inspiring conversations where we discussed and played through ideas never tried before. I am thankful for your generosity in coding the first versions of 20.000 BLOCKS in Minecraft just because it was fun. Thank you for opening my eyes to the elegance of good game design. And for taking the time to sit with me for the interview found in the Appendix.

A special thank you goes to Roger Winkler, who, from all my students, was the one to pick up the ideas of this research and develop his own understanding and projects around them. The case study Project Reptiles I present in this work is based on his master thesis. More than that, I would like to thank you for working together to wrap these ideas into a product for the AEC market. This brought a whole new understanding of the research and its relevance for the discipline of architecture and the construction industry.

I was lucky to be at the same time in DDU as Andrea Rossi since we exchanged ideas for the democratization and decentralization of architectural design and construction. The newly emerged field of Discrete Design in the architectural discourse, driven by Andrea's development of the toolbox WASP, was an inspiring reference when exploring the ideas around playable voxel shape grammars and the tileset approach in my own work.

I want to acknowledge all my colleagues from the DDU and the Faculty of Architecture at TU Darmstadt, most of all Prof. Anna-Maria Meister, Samim Mehdizadeh, Norwina Wölfel, Nadja Gaudilliere-Jami, Manuel Pfänder, and Sebastian Frell, for many inspiring discussions, which sparked many ideas and pushed this dissertation that one step further. Thank you, Dr. Shayani Fernando, for proof-reading drafts of this work. Thank you, Yvonne Machleid, for your patience and

swiftness in supporting me with all matters of organization and administration and for the positive, smile-inducing attitude. It would not have been possible to navigate the environment of TU Darmstadt as successfully without you.

Huge thanks to everyone I collaborated with during the last years in developing the projects that built the blocks of this dissertation and to my students at the TU Darmstadt.

More specifically, in the case of Sensitive Assembly, I would like to thank Stig Anton Nielsen for my first academic cross-institutional collaboration. Many thanks for the support with the project go to Peter Maier and Andreas Benz at the laser-cutting workshop of TU Darmstadt, Tim Wetzel for assistance with the structural model, the DDU student assistants Farnaz Oveisi and Philipp Vehrenberg, the NODE organizers, in particular David Brüll and Nils Weger as well as Kalle for the help with on-site set-up, Christian Leicher for photographs and to the many visitors of the festival who played a game at the Sensitive Assembly installation.

20.000 BLOCKS holds a special place in this work and so do the people who helped it become the inspiring case study it is. Special thank you to Alban Voss, Alexander Gösta, Alexander Stefan, Andrea Quartara, Ashris Choudhury, Ben Buckton, Christian Schwamborn, Jörg Hartmann, Lorena Müller, Marc Emmanuelli, Marios Messios, Max Rudolph, Oskar Gerspach, Rui Nong, Samuel Eliasson, Sebastian Kotterer, Steffen Bisswanger, Theo Gruner, Thomas Valentin Klink, and Ulrik Montnemery. And to the students who took a course or even did their thesis using 20.000 BLOCKS. Thank you for being open to new paths in your education to Alexander Kay Mayer, Annabell Koenen-Rindfrey, Joern Rettweiler, Julia Sajak, Julia Schäfer, Lukas Stöckli, Mehmet Erkan Eker, Mingquan Ding, Moritz Markgraf, Robin Find, Shilla Anjadini, Tadeusz Diduch, Timotheus Krzywik, Viola Abu-Salha, Yadi Wang, and Yingbo Sun. Many thanks to the organizers of Smart-Geometry 2016 for letting us host a cluster where we played with 20.000 BLOCKS and developed it further. Special thank you to IBA Heidelberg for having trust in our team of young researchers to develop a game, the IBA_GAME, for the engagement of the citizens in the design process of the new Patrick Henry Village in Heidelberg. In particular Prof. Michael Braum and Carl Zillich. The IBA_GAME version of 20.000 BLOCKS was made possible through the sponsorship by Eternit GmbH. And, of course, many thanks to all the Players of 20,000 Blocks throughout the years.

Project Reptiles is the case study I had the pleasure to guide but at the same time also watch grow and develop in the work of Roger Winkler. And I am lucky to have had this opportunity. Many thanks to the participants in the workshops Aleksandra Buchalik, Alexander Kay Mayer, Anna Mytcul, Felix Dannecker, Jan Keller, Julia Kühn, Leonhard Stark, Sun Wie, and Viola Abu-Salha. I appreciate the time and generosity of the people from the offices of ACMS, SWECO, nkbak, and Bollinger + Grohmann, who sat with Roger and me for an interview. Thank you for your honest and inspiring reading of the practical relevance of this research.

The Rechteck2BIM case study was the most challenging to frame, conduct and discuss as it was the closest to the architectural practice of all case studies in this work. This is why I am grateful for all the support and input from everyone involved. Most importantly, Roger Winkler for working with me on the most recent prototypes. I want to thank Snezana Heidebrecht, Mehran Mojtahedzadeh, Manuel Pfänder, Sebastian Frell, Nassim Delkash for seeing and trying many prototypes and

opening up to share the workflows from their respective architectural practices to make this research relevant. Many thanks to all my students from TU Darmstadt: Ana Baraibar Jiménez, Ana Sophie Sánchez Wurm, Anjuscha Helbig, Dina El Gindi, Gerrit Walser, Jörg Hartmann, Kevin Henkel, Louise Hamot, Luisa Ruffertshöfer, Marc Ritz, Mariona Carrion, Max Sand, Maximilian Pfaff, Meizi Ren, Mengxue Wang, Morgane Hamel, Natascha Damaske, Nicole Klumb, Philipp Vehrenberg, Rui Zhi, Stefanie Joachim, and Zhiyin Lu. A special thank you goes to the Computer Science bachelor students from TU Darmstadt, who implemented a prototype of the rechteck game. Their fresh view and diligent user testing helped crystalize the initial ideas into robust principles. Special thank you to HIGHEST the Innovation and Start-up Center of TU Darmstadt and the Hessen Ideen Stipendium, specifically Katja Walther, Sabine Remmert, Melanie Schöyen, and Bartosz Kajdas, for providing support and guidance in the endeavors to make this research relevant for the AEC market.

The practical and academic experience I gained before my time at DDU was essential in formulating my research agenda and developing the skills needed to pursue it. I want to thank all my colleagues and students at the Städelschule Architecture Class, specifically Prof. Johan Bettum. I would also like to thank my colleagues at the office of Bollinger + Grohmann. A special thank you goes to Anatoli Skatchkov and Ira Kublin from the gallery Platform Sarai for providing the opportunity to exhibit the participatory installations Project Avocado and Feedscape. I am grateful and honored to have been selected for the scholarship at MAK Senter for Art and Architecture in Los Angeles, where the seeds of this research were planted. Thank you to MAK Center director Kimberly Meyer for the support and exciting discussions.

Last but not least I would like to thank my family. Thank you to my parents, Blaguina and Matey, architects themselves, for their forward-thinking guidance on where the profession is heading. And for having raised me to be open to exploring ideas and inspiring me to see Architecture not as a profession but as my way of life. Thanks to my brother, Vladimir, who, through all those years, was among the first to listen to my countless raw and unpolished ideas and kept providing feedback that brought clarity to the work. I am deeply grateful to Diana for always being there when things didn't work out as planned, encouraging me to try again from a new direction or with more force. And thank you to my wonderful daughter Lina and son Matthey for indirectly motivating me to be more effective at my work so I can play more with you.

Thank you all.

Contents

Abstract	i
Acknowledgements	iv
Preface	xi
1 Introduction	1
1.1 Background and Motivation	1
1.2 Thesis Statement	3
1.3 Methodology	7
1.4 Thesis Contributions	14
1.5 Thesis Outline	14
I FOUNDATION	17
2 Participatory Design	19
2.1 Wicked Problems	19
2.2 Participation	23
2.3 Participatory Design	24
2.4 Participatory Design and Wicked Problems	25
2.5 Examples of Participatory Design	25
2.6 Conclusion	33
3 Generative Design	35
3.1 Design Systems in Architecture	36
3.2 Taxonomy of Generative Design	43
3.3 Human-in-the-loop Potential	83
3.4 Potential for Automation in Construction	102
3.5 Conclusion	107
4 Crowd Wisdom	109
4.1 Definitions	110
4.2 Architectural Competitions as Crowdsourcing	124
4.3 Conclusion	126
5 Game Design	130
5.1 What is a Game?	131
5.2 Why Games?	131
5.3 Conclusion	137

6	Four Fields and Eleven Intersections	138
6.1	The Four Fields	140
6.2	Six Double Intersections	140
6.3	Four Triple Intersections	142
6.4	The Core Intersection of All Four Fields	143
II	EXPLORATION	145
7	Sensitive Assembly	147
7.1	Design Paradigm	147
7.2	Implementation and Setup	149
7.3	Machine Agency	154
7.4	Human Agency	163
7.5	Takeaways	166
8	20.000 BLOCKS	171
8.1	Design Paradigm	172
8.2	Implementation and Setup	173
8.3	Machine Agency — Playable Voxel-Shape Grammars	189
8.4	Human Agency	206
8.5	Take-aways	262
9	Project Reptiles	267
9.1	Design Paradigm	268
9.2	Implementation and Setup	269
9.3	Machine Agency	280
9.4	Human Agency	284
9.5	Take-aways	294
10	Rechteck2BIM	295
10.1	Design Paradigm	295
10.2	Implementation and Setup	296
10.3	Machine Agency	297
10.4	Human Agency	318
10.5	Take-aways	321
11	Discussion	322
11.1	Design paradigms	323
11.2	Tools, tasks and roles	325
11.3	Hierarchy of human-machine interaction	326
11.4	Three scenarios for the use of crowdsourcing in architecture	338
11.5	Granular authorship	343
12	Conclusion	346
12.1	Summary	346
12.2	Relevance for the architectural discipline	347
12.3	Thesis contributions	348
12.4	Future Work	349

12.5 Outlook	350
III APPENDICES	353
A Interview with the Game Designer Ben Buckton	354
B Getting Started with 20.000 BLOCKS in Minecraft	363
List of Tables	400
List of Figures	401
Bibliography	413

Preface: Personal statement and experience in the field

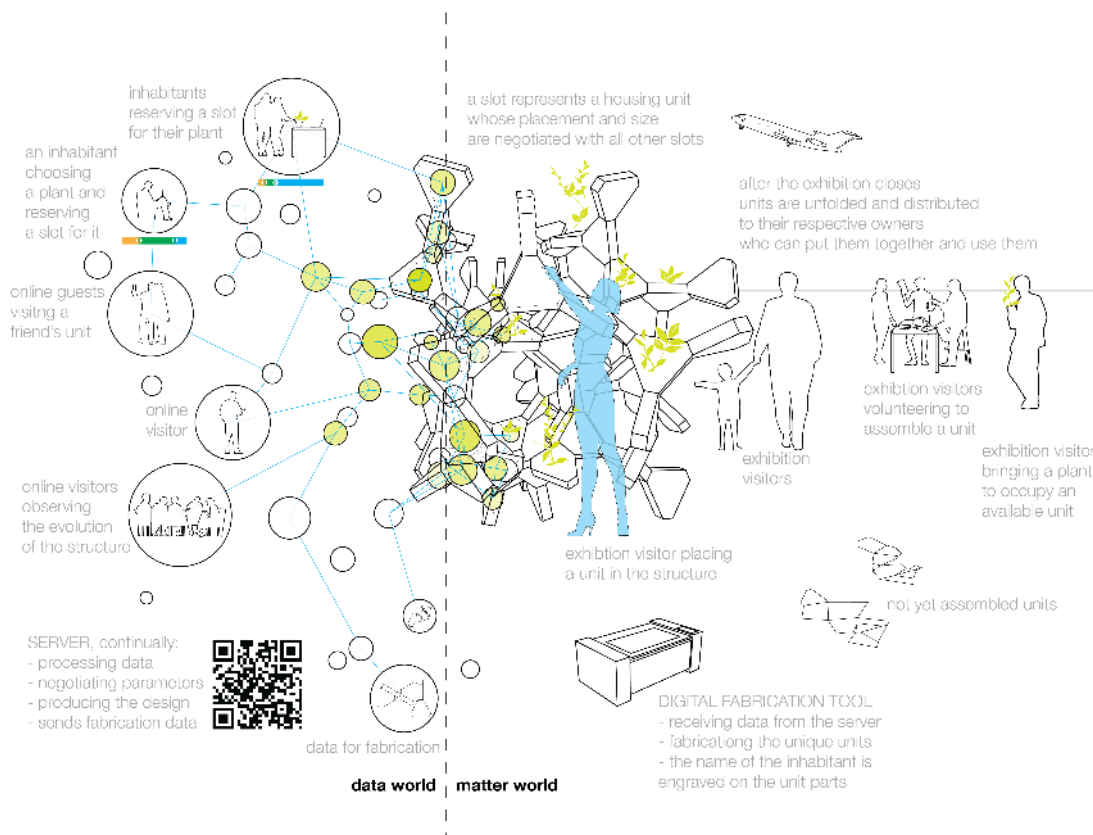


Figure 1 – Project Avocado. Image credits: the author.

I grew up in Bulgaria in a family of architects, and I am educated as an architect. This very focused professional environment exposed me to the production of architecture from early on. I was hand-drafting as early as ten years old and took on CAD and BIM during high school. I had a passion for creating virtual representations of fictional realities. Another strong interest of mine was computer programming, mainly focusing on graphic interfaces and 3D simulated worlds.

During my architectural education in Sofia, I found the applications of programming in architecture fascinating. Architects were trying to design buildings with Genetic Algorithms and Cellular Automata in the 2000s. Architects could make their own tools to simulate natural processes. Algorithmic and systems thinking influenced my architectural projects from early on. I realized that it could give the architect a means to manage the complexity of diverse inhabitants' preferences. I

felt urged to use these algorithmic methods to design alternatives to homogeneous, repetitive buildings. This was probably triggered by the socialistic “same for all” panel housing shaping the cities of Bulgaria before and during my childhood.

During my master’s studies at the Städelshule Architecture Class in Frankfurt, I explored the application of scripting in design. I linked it in real-time to online databases such as Flickr, Twitter, and Google Search. The Web 2.0 phenomena of user-generated content influenced my goals and ideals for democratizing architecture.

After I completed my master’s studies at the Städelshule, I had the opportunity to work in the inspiring, hacker-like environment at the engineering office of Bollinger+Grohmann. There, my computational design skills could be utilized for the design process in practice. Being a part of an engineering office, interfacing between disciplines and between tools was a daily activity. Simultaneously, through my teaching at Städelshule Architecture Class, I played with new digital tools and design techniques. However, in all the optimization algorithms and simulations I ran in that period, I felt something was missing — namely, the direct input of the end-user of the product we were creating. The inhabitants, although many of them with access to technology, in the form of smartphones and laptops, and skills to creatively use it, equal to or more advanced than mine, were rarely part of the design process.

This led me to try concepts at a small scale, producing interactive digital-physical installations such as Project Avocado (2012) (Figure 1) and Box in a Cloud (2013). Both projects focused on the spatial organization of residential units following user-defined parameters. The feedback I got from participants in those projects reassured me that they could understand the basic underlying principles of architecture beyond colors and decorations. Furthermore, I encountered a motivating and inspiring desire for participation and co-creation — for being a part of something larger.

Encouraged by this feedback, I began this Ph.D. thesis in 2015. I felt confident that the architectural concepts from the 1960s-70s to develop technology and scientific methods to achieve a more explicit and more democratic design process are realizable today. I believed these ideas had not become a reality simply because the needed technology did not exist back then. The insights I gained from developing the case studies for this dissertation challenged my belief. I realized that it’s precisely the advances in technology today that make it easy to see how those concepts were flawed. However, I also realized that there is a potential for architectural innovation not solely in the algorithmic nature of computers but in them offering new media for communication.

At the core of my research presented here has been how to apply technology to boost participation and creative collaboration to the extreme. How architects, inhabitants, thousands of third-party contributors, and algorithms can use the interactive medium of computer models to work together on a design problem. Everyone from the position and skills of their role in the process.

Anton Savov, Frankfurt, 15 October 2021

Chapter 1

Introduction

“Technology is the answer, but what was the question?” (Cedric Price)

1.1 Background and Motivation

By the year 2050, we need to adapt our built environment to house billions of additional people in cities worldwide, respond to the effects of climate change, as well as meet the goals to contain them (Newswatch Times 2013; Opoku 2016; C. Smith and Levermore 2008; Woetzel et al. 2014). But there are two other — technological — pillars that by 2050 will define how the built environment will be created — that my research steps on. Namely, *network-enabled participation* and *ubiquitous digital fabrication*.

The first pillar supports the narrative that technology lowers the barrier to participation and fosters collaboration (Rifkin 2015). The information revolution causes ideological shifts that encourage decentralization and weaken the traditional authority, creating a powerful force to reclaim the commons (Rifkin 2015; Roberts 2001). A newly gained societal and economic awareness highlights value differences and thus promotes conflict rather than harmony (Roberts 2001). At the same time, participation’s power has dwindled over the last decades as it has been contained within the algorithmically calculated consensus of the online platforms, turning them into “a source of exploitation, cooptation, or enhanced neoliberal domination” (Kelty 2019, p.2). *Open Design* tries to counter this with its three dimensions of openness — participation without permission, a low threshold of participation, and openly licensed output (de Mul 2016). Architecture, although late to the network game, is now also caught between using the online crowd as an unregulated resource of benefit for the few or fostering its creative power for the common good (Parvin 2013; Sanchez 2021). If everyone can participate in the *network-enabled creation* of the built environment, what role will they play?

The second technological pillar amplifies our ability to shape our environment. Our digital design models and BIM models are the fabrication instructions, the structural simulation, the bill of quantities and materials, etc. This enables us to bring about new buildings into the world faster, easier, and more precise than ever before. And with *robots and digital fabrication* entering the discipline of architecture, construction will accelerate even further (Figure 1.1 right) (Gramazio, Kohler, Willmann, et al. 2014; Leder et al. 2019; Open Systems Lab 2019; Picon 2014; Wagner, Alvarez, Kyjanek, et al. 2020). It is estimated that by 2050, facilities for

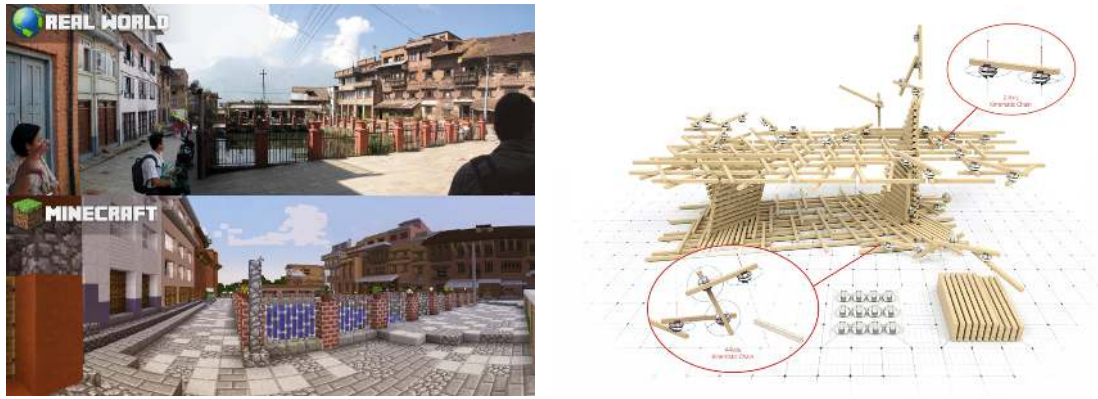


Figure 1.1 – Block by Block and Robotic Construction. Left: Block by Block a project by UN-Habitat using Minecraft to engage local communities in the renovation of their public urban spaces. Right: Entirely automated digital fabrication of a building using distributed robotics. Image credits: Block by Block, Leder et al. 2019.

digital fabrication (FabLabs), doubling in number every year and a half, will reach a billion, covering every part of the world and enabling “anybody [to] make (almost) anything locally, while using knowledge that is shared globally” (N. Gershenfeld, A. Gershenfeld and Cutcher-Gershenfeld 2017, 2018). The *democratization of digital fabrication* reverses the approach for technological innovation from top-down to bottom-up (de Mul 2016; N. Gershenfeld 2006, 2012). With such an empowered crowd and ever-accelerating construction, new buildings become a shaping force of the Anthropocene. They have a much more significant impact on society, ecosystems, and even geology than ever before (Waters et al. 2016). Yet, buildings are just printouts of a temporary state of an idea that keeps evolving in society’s collective mind, in the crowd. I regard the collective formulation of the idea for a building as more important than what gets built at the end. If anyone can use *digital fabrication* to build any building, how will we design it? Who takes part, what tools are at their disposal, and who are they in dialog with.

The questions of what roles network-enabled participants play in architecture and how buildings are designed if anyone can build anything lead to much bigger questions beyond this dissertation’s scope, yet worth listing to frame the work. In 2050, will most architects still design ‘one-of-a-kind’ buildings in the hope of winning a commission or a competition? Or will they be creating systems enabling others to design? Will inhabitants immerse themselves in Virtual Reality only to pick the floor finishes of their new homes? Or will they interactively explore qualitatively different home designs until aware of how their home fits in their life, the city, and nature? Will the networked crowd do design gigs in a race to the bottom? Or will it collaboratively invent novel housing typologies and business models? Will Architecture continue to ignore the middle of the bell curve, finding purpose in working either only for the rich or the deprived (Habraken and Teicher 2000; Parvin 2013; Sanchez 2021)? Hopefully, in 2050, the billions of people who recently arrived in cities worldwide will occupy the middle of the curve and not the extremes. Will Architecture worsen the *tragedy of the commons* (Hardin 1968) or contribute to their reconstruction?

1.2 Thesis Statement

Architects are skilled at discovering and codifying a project’s value system and turning it into a physical artifact. Architecture helps figure out *what* to build and also coordinates *how* to build it. It did that in Antiquity. It did that even after Alberti drove a wedge between the two questions in the Renaissance (Carpo 2011). It does this today. And it will still do it in 2050.

As such, Architecture sits at the interface between the realm of *wicked* problems and the realm of *tame* problems (Lange 2016; Rittel and Webber 1973). Engineering and construction deal predominantly with tame problems. They are well-defined, solutions can be rated and optimized. A pitfall to avoid is considering the whole design process as a tame problem. Architecture deals predominately with wicked problems. They are ill-defined, there is no apparent problem definition, solutions are not optimizable, nor objectively rate-able (Elezkurtaĵ and Franck 2002; Rittel and Webber 1973). When dealing with wicked problems, the approach is to confront stakeholders with potential solutions to get them to agree on a problem definition and the criteria for acceptable solutions (Elezkurtaĵ and Franck 2002; Lange 2016).

Technological, cultural, economic, and social changes are causing wider recognition and rise of wicked problems (Roberts 2001). Many processes are aiming to become more inclusive of stakeholders’ values, opinions, and preferences (Hoskins et al. 2012). Two main approaches to Architecture have emerged as a result of this — *participatory design practices* and mass-customization in the form of *building configurators*.

Current *participatory design practices* acknowledge the true nature of wicked problems, taking into account stakeholders’ values and preferences. They incorporate this input by sandwiching the traditional design phases between a *Phase Zero* and a *Phase n+1* (Granath 2001; Hofmann 2018, 2019; Poplin 2012; Sanoff 1990, 2011; Simonsen and Robertson 2013). Block by Block (Figure 1.1 left) is a good example. It was started in 2012 by the United Nations Human Settlements Programme (UN-Habitat) in collaboration with Mojang, the company making the popular game Minecraft. The project aims to “empower communities to turn neglected urban spaces into vibrant places that improve quality of life for all” (UN-Habitat 2015, 2016). Today it lists 30 projects across dozens of countries on its website¹. Out of its 12-step method, step 10 represents the entire traditional design process with all its phases and challenges (Block by Block team 2021). The additional 11 steps, nine before and two after the architect’s job is usually done, ensure participation². Because of its open-ended nature, participatory design overburdens the architect, who now has to synthesize far more constraints into a one-of-a-kind design and produce its construction drawings, following no system but a trial-and-error process. Participatory design is not scalable because it lacks explicit methods and means to encode expert knowledge. Furthermore, it often suffers from not being able to reach a diverse group of representatives of the target group of stakeholders — with participatory design workshops, especially on urban planning issues, often attracting people of similar age and background (Knöll 2018).

Building configurators, on the other hand, continue a long tradition of encoding expert knowledge into a design system, such as Durand’s *parallel plates* from the

¹<https://www.blockbyblock.org>

²See *chapter 2: Participatory Design* for details.

1800s or Gropius’ *Baukasten im Grossen* from the 1920s (Durand 2000; Gropius 1965). The novelty with a building configurator is that the design system is used not by the architect but by the inhabitant directly. A building configurator can be as simple as a cleverly made physical model, like the design toolbox by BARArchitekten given to future inhabitants of Spreefeld Berlin to define the walls of their flats from a set of predefined options (Figure 1.2 left) (Becker 2015). For decades, the prefabricated single-family house industry has operated based on design catalogs with thousands of standardized options. Parametric design has given it the ability to morph countless in-between states between any two-floor plans as illustrated by *Mr+Mrs Homes*³, a German startup for mass-customized houses (Figure 1.2 middle). Techniques for *Generative Design*⁴ promise to equip architects with an even bigger power to standardize and systemize (Benros and Duarte 2009; Bianconi, Filippucci and Buffi 2019; Dillenburger et al. 2009; Duarte 2005; A. S. Howe, Ishii and Yoshida 2017; Jensen, Olofsson, et al. 2014; Jensen, Lidelöw and Olofsson 2015; Malmgren, Jensen and Olofsson 2011). Building configurators, and generative design in general, integrate very well with digital fabrication. However, the common trap of generative design is trying to treat architecture simply as a *tame* problem. They cope with design complexity by reducing it to a finite number of floor plan alternatives and predefining aesthetics by limiting the choice of material options and geometrical expressions. This approach is not truly inclusive as it takes away the power of stakeholders to meaningfully influence the outcome.



Figure 1.2 – Two types of tools for the non-experts. Left and middle: an architecture tool — all outcomes predefined, single-user. Right: a science tool — explorative, collaborative. The number of people and the amount of expertise encoded in the tool are very low on the left but super high on the right. How to bring that to architecture? Image credits: BARArchitekten, Mr.&Mrs. Homes, Foldit.

Research gap

Participatory design practices and mass-customization in the form of *building configurators* frame the identified research gap in my work (Figure 1.3). Both approaches are the two sides of the same coin, i.e., of solving wicked problems *authoritatively*. Both are built around the central role of the experts: the one architect who will process all stakeholders’ input into a design or who will come up with a standardized design catalog. However, when dealing with wicked problems, interaction and dialog between everyone involved are beneficial (Lange 2016; Rittel and Webber 1973; Roberts 2001). According to Majchrzak and Malhotra 2013, crowds cancel out personal biases through the sheer interaction among the members of the crowd.

³<https://www.mrmrshomes.de/>

⁴See chapter 3: *Generative Design* for details.

Therefore other disciplines have started exploring the two alternative strategies to solving wicked problems — *collaborative* and *competitive* (Figure 11.1) — which emerge, respectively, when power is dispersed and contested (Roberts 2001).

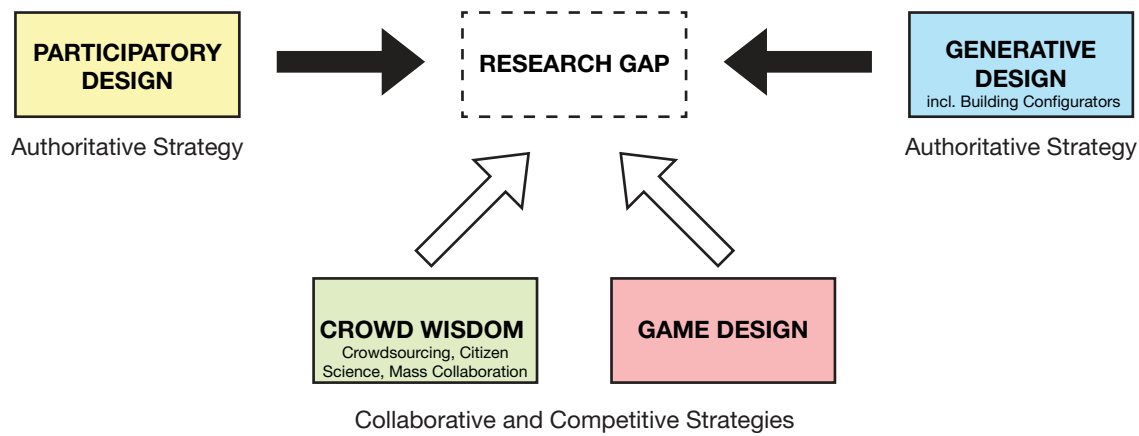


Figure 1.3 – The research gap. Image credits: the author.

The collaborative/competitive approach to solving open problems has found robust application in Citizen Science⁵. In 2008, after years of failed attempts to develop an algorithm that can simulate the folding of protein molecules, a group of scientists at the University of Washington developed the video game *Foldit* (S. Cooper, Khatib, et al. 2010). The player is presented with a protein molecule, and their mission is to reshape it by connecting couples of atoms together (Figure 1.2 right). The player’s score increases as the energy required to keep the molecule in its current shape drops, meaning it is closer to nature’s solution. Nature decides who wins and who loses the game. Thousands of online players, driven by their desire to win and simultaneously help find proteins that could potentially cure deadly diseases, started to engage and understand the research aspects behind the game. They read scientific papers and shared their progress on the protein puzzles with each other. *Foldit* shows the potential of games to encode expert knowledge, provide real-time feedback and automate or outsource onboarding to the needed skill level so that anyone can contribute.

Reddit’s *r/Place* is another example of how tens of thousands of people can collaborate and compete to produce a collective work of art (Figure 1.4) (Rappaz et al. 2018). *Reddit* is an American social news aggregation, web content rating, and discussion website. Registered members submit content to the site, such as links, text posts, images, and videos. These are then voted up or down by other members. On 1 April 2016, *Reddit* put online a white canvas of 1000 by 1000 pixels and gave people 72 hours to paint something on it. Altogether, more than 250.000 people took part. Every participant had the same 16 colors to choose from and could paint only one pixel every five minutes. If you wanted to draw anything on the canvas, you had to convince a large enough group of people to join your cause. Groups formed around nation flags, famous works of art, and symbols of internet culture but also more abstract ones such as *The Corner of Blue* or *The Green Lattice*⁶. People started *Reddit* communities (subreddits) to communicate and coordinate what they

⁵I use the term Citizen Science in its meaning of projects where nonscientists contribute and process scientific data. See chapter 4 for details.

⁶A descriptive video with time-lapse of *Reddit*’s *r/Place*: [YouTube link](#).



Figure 1.4 – Reddit r/Place. 250.000 participants, 72 hours, the underlying system is predefined, the content evolved within it. Image credits: Reddit.

wanted to draw and how to defend it from being painted over by the other groups (Vachher et al. 2020).

Thesis

How do *Participatory Design* and *Generative Design* change if we venture beyond the paradigm of the authoritative approach? The two fields that *r/Place* and *Foldit* draw from, *Crowd Wisdom* and *Game Design*, can provide the level of content creation, options exploration, and automation of guidance and education that can break participation and design computation out of their respective confines and integrate them. Altogether these four fields build the landscape for my thesis, which is given below:

Thesis: *Architectural design knowledge can be encoded into generative game worlds where every role — architects, stakeholders, and third-party participants — can contribute, respective to their skills and interests, to creating schematic architectural designs.*

Research questions

Given the thesis presented above, I have formulated the following research questions:

1. What design paradigms lie at the intersection of participatory design, generative design, crowdsourcing, and games?
2. What tools, tasks, and roles can be offered at this intersection for the creative involvement of the various groups — architects, stakeholders, third-party contributors?

An adequate approach at investigating the research questions would:

- employ computational techniques for the encoding of expert knowledge that have a low threshold of participation and can be set up and filled with content from any architectural design expert;
- allow anyone to design while assisted by encoded expert knowledge and fabrication constraints, learning from what has been designed by others before and resorting for help to experts (professionally trained or self-taught) when needed;
- offer the opportunity to adjust the balance of power between architects, stakeholders, and players so that the spectrum from authoritative towards collaborative and competitive problem solving can be explored.

1.3 Methodology

The nature of the research work presented here is explorative. I begin with research questions and aim to arrive at a set of take-aways, claims, and new hypotheses for future work. I investigate the thesis by exploring the architectural design potential at the intersection of *Participatory Design*, *Generative Design*, *Game Design*, and *Crowd Wisdom*.

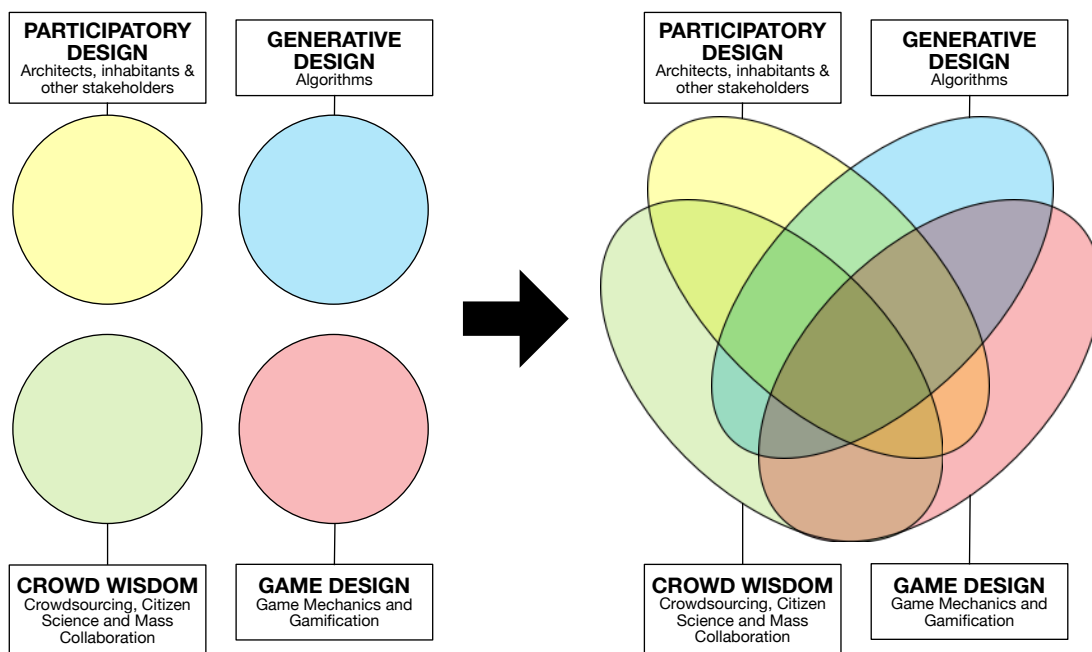


Figure 1.5 – A map of the fields investigated in this thesis. The four fields — on their own (left) and as a Venn diagram (right). Image credits: the author.

When we intersect each of the four fields with the other three, we get the Venn Diagram on Figure 1.5. I use it as a guiding map for critically assessing state-of-the-art precedents and my projects.

Answering the research questions does not lend itself to simulation, logical deduction, or quantitative methods. It relies on the participation of many people in various roles and therefore calls, mainly, for hands-on, project-based case studies. I am using the case study method for qualitative research as its purpose is exploring,

describing, and explaining phenomena embedded in their context and raising new questions that cannot be answered affront (Creswell and Poth 2018; Groat and D. Wang 2013; Wibranek 2021; R. K. Yin 2014). The case study method is a qualitative research method that:

“...follows a strategic procedure for collecting, organizing, and interpreting contextual information and generate insights into phenomena that cannot be measured quantitatively. Its goals are developing a deep understanding of a phenomenon and generating research questions and hypotheses that can be further tested in quantitative research.” (Wibranek 2021)

The case studies focus on prototyping participatory design processes and fabricating the results on a small scale. The case studies are set up either as research projects or as teaching courses offered to architecture students at the Technical University of Darmstadt.

1.3.1 Assumptions

Naturally, the question emerges: *Is it possible to automate the construction of user-generated designs using robots?* In the spirit of working in ideal types, I will make the strongest assumption possible: all construction is entirely automated, and it is possible to digitally fabricate any structurally sound building design. This is not logically possible, but I assume it to be true to avoid entangling myself in a discussion about feasibility or efficiency. This allows me to focus on the research questions concerning the balance of design power between the different roles and the range of their tools and tasks. The applicability of my research findings beyond this strong assumption can be insured by selecting generative design techniques that are compatible with robotic construction and digital fabrication (See section 3.4).

1.3.2 Case Studies Structure

With automated construction as a premise, my method primarily consists of choosing a suitable generative design technique and making it interactive. Yet, the work is not just about technical infrastructure or tooling. It is about understanding the *human agency* of the different stakeholders, their behavior, and how it can be modified, motivated, or challenged.

As various participant groups bring multiple skill levels and take on tasks with varying difficulty, my case study projects provide progressively impactful design tools for the various roles. The influence of game rules, mechanics, and the internal economics of those tools is more important than the technical realization.

I have chosen the *Flow Framework* by the psychologist Mihaly Csikszentmihalyi (Csikszentmihalyi 2008) to guide the design and assessment of the case studies. It posits the existence of three states — boredom, anxiety, and flow — where a participant might find themselves, depending on how their skills intersect with the challenge of the task assigned to them (Figure 1.6).

It is more likely that non-experts participating in the design process will lack the skills required to make design decisions and create designs. Similarly, architects aiming to provide non-expert participants with a project-specific, computational or

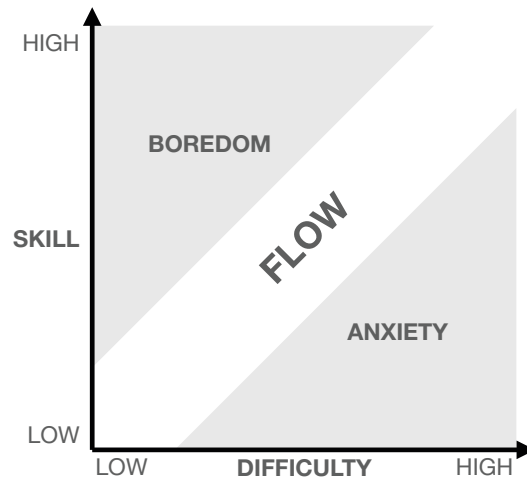


Figure 1.6 – The concept of Flow. Adapted from Csikszentmihalyi 2008. It posits the existence of three states — boredom, anxiety and flow — where a participant might find themselves, depending on how their skills intersect with the challenge of the task assigned to them. Image credits: the author.

parametric model to explore the design space systematically will rather lack the needed computer skills. Both of these groups would fall in the Anxiety zone. The case studies aim to research tools, roles, and design paradigms that bring them out of there into the Flow Zone.

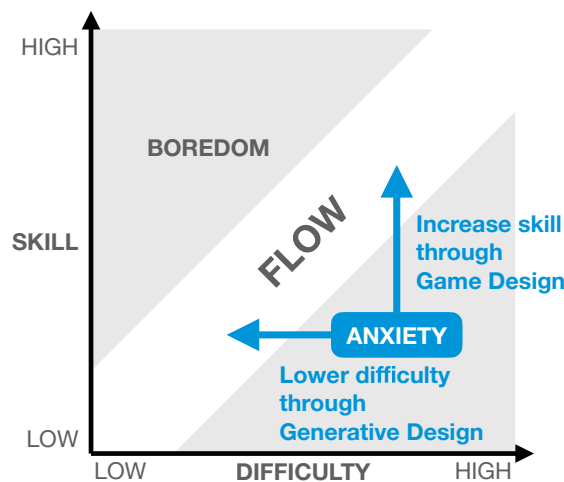


Figure 1.7 – From Anxiety into Flow. Using the machine agency of Generative Design and Game Design to respectively lower the difficulty of the challenge and increasing the skills of the participants. Image credits: the author.

The Flow framework allows me to consider the main characteristic of my cases studies, namely interactive, participatory modeling as two separate vectors. *Generative Design* can lower the difficulty of the challenge by automating a large portion of the work (Figure 1.7). And *Game Design* can guide participants through a series of challenges, effectively increasing their skills by helping them understand the underlying principles of the design task at hand. The first vector aims to lower the difficulty by employing assistive generative algorithms. The second vector seeks to increase the skill by using game design strategies to automate information and task flow to the participants.

As implied by the two vectors, generative design techniques provide architec-

turally specific *machine agency* that forms the basis for interactivity, participation, and game mechanics in this work. Therefore in both the literature review and the case studies, the focus is on the field of *Generative Design*.

In response to the concepts of *human agency* and *machine agency*, the method I am using within a project is to take a generative design technique and make it interactive. More specifically, a project typically involves the following:

1. **Machine Agency** — Pick a procedural technique that lends itself well to encode architectural design. Make the procedural system interactive by exploring suitable game designs and their integration into a 3D interactive multiuser software environment. Turn participation in finding a solution to the non-expert tasks into an entertaining and rewarding experience. Adapt and implement automated analysis, focusing on structural performance, environmental performance, etc., to evaluate and post-process the player-generated designs.
2. **Human Agency** — Create catalogs or tile-sets of design elements to inform the generative design system. Ideally, the participating experts perform this step. It involves analyzing precedents of low-rise, high-density housing to extract programmatic elements, rules, and geometric shapes that make them up. Ask people to create schematic architectural designs with the developed design system. Observe how they use it and what they make with it.

1.3.3 Scope of Work

The research aims to produce new knowledge about tools, interactions, and roles at the intersection of architecture, crowdsourcing, and game design. The aim is not to arrive at a specific object, design system, or design for a specific building.

The method I propose concerns itself with new residential buildings in an urban environment. Other architectural typologies are not considered. Although the approach could probably be used on renovation projects, this research focuses on new designs and new constructions only.

The research encompasses case studies in digital environments involving actual participants. I aim to use an existing multiuser, 3D game-like environment instead of a custom software development from scratch.

The robotic production of small-scale models of the designs (scale 1:200) is used to demonstrate the validity of findings beyond the assumption for ubiquitous digital fabrication.

Subject of research

Architectural design as a service solves many problems and has many aspects. The aspect of design explored in this dissertation is that of organizing the spaces of a building, usually called *spatial configuration* or *the layout problem*. It is the task performed in the very early design phases and consists of laying out the rough proportions of the building and the relative position and orientation of rooms, zones, and openings for connectivity.

This task entails design decisions that affect all stakeholders (problem owners) instead of affecting only one of them. For example, the choice someone makes

for which light switches they use at home is irrelevant for their neighbors or co-citizens, the choice of which window detail they use is also of no concern to the other stakeholder. On the other hand, the position, size, and proportion of windows matter to their neighbors and co-citizens because of the views this opens or closes and the influence this has on the privacy and intimacy of the inhabitants.

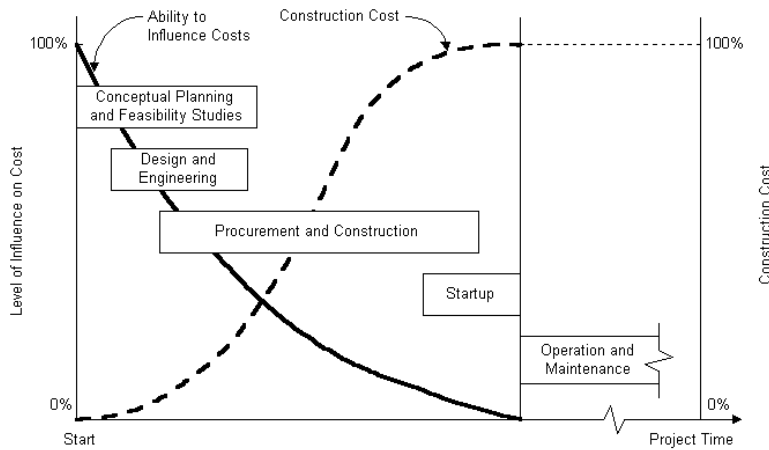


Figure 1.8 – The Paulson curve. The operation stage is taken into account. The curve shows the diminishing ability to influence costs as project stages progress and the increasing cost of changes. The curve is better known as the MacLeamy curve as Patrick Macleamy appropriated it in the early 2000s (Davis 2011). Boyd Paulson is a Professor in Civil Engineering at Stanford University. Patrick MacLeamy is an American architect and the chairman of buildingSMART International. Previously, he served as Chairman and CEO of HOK, a global architecture, engineering, and planning firm. Image credits: Hendrickson 2008.

The design phases in focus in this work are *feasibility study* and *schematic design*. *Feasibility study* and *schematic design* are the terms used in USA as defined by The American Institute Of Architects (Fontan 2019). In the UK the terms are: 0. *Strategic Definition*, 1. *Preparation and Brief* and 2. *Concept Design* (Sinclair et al. 2013). In Germany the terms are formalized by the *Ordinance on Architects' and Engineers' Fees* (HOAI) as *LP1 Grundlagenermittlung* and *LP2 Vorplanung*. These design phases have the most substantial influence over the design cost, performance, impact, etc. So the aim is to engage the stakeholders at that stage, when their input will be practical and not costly (Figure 1.8).

The subject of research in this dissertation is the design process for creating schematic designs, focusing on *spatial configuration* (See Figure 1.9) and *massing*, i.e. the building's envelope.

According to Michalek, Choudhary and Papalambros 2002, “spatial configuration is concerned with finding feasible locations and dimensions for a set of interrelated objects that meet all design requirements and maximize design quality in terms of design preferences”. The problem of spatial configuration spans various disciplines with areas of application that include component packing, route path planning, process and facilities layout, VLSI integrated-circuit design, and architectural layout (Michalek, Choudhary and Papalambros 2002).

Akin and Moustapha 2004 define *architectural massing* as “the act of composing and manipulating three-dimensional forms into a unified, coherent architectural configuration”. Architectural massing is the phase of design where the architect defines the building's identity (Akin and Moustapha 2004). As such, architectural

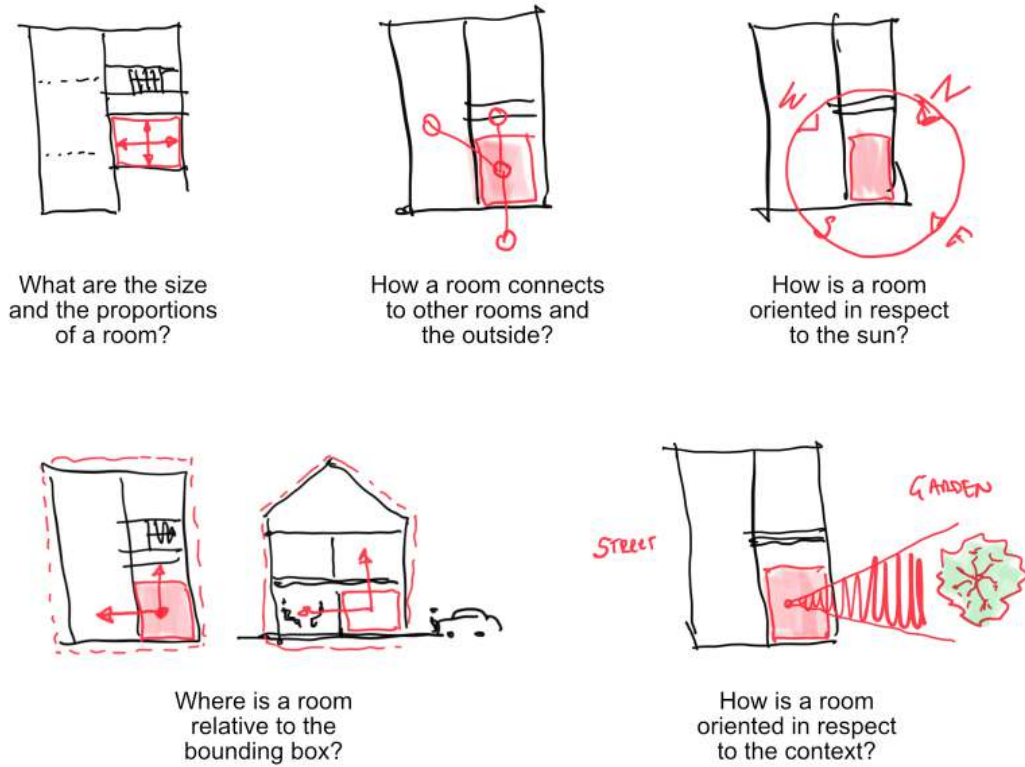


Figure 1.9 – Spatial configuration parameters. Image credits: the author.

massing, unlike spatial configuration, goes beyond the mere fulfilling of customer needs into a field of artistic expression.

Object of research

To give me and the participants specific architectural references, the object of the research is *new residential buildings*. More specifically, low-rise, high-density housing, and single-family houses. Low-rise, high-density buildings are up to 8 floors high, of diverse typologies, and often have mixed-use, share-friendly programs (ArchDaily 2015; Schramm 2008; Tomoko 2010). Several reasons make residential buildings particularly relevant for research into methods for participatory and crowdsourced designs.

First, residential buildings have high social and demographic relevance to the growing population of cities and high real estate prices. This is especially true today when we face a shortage of a billion homes worldwide (Newswatch Times 2013). Germany alone has a need of 1.5 million homes as of 2018 (Tichelmann et al. 2019).

Second, focusing on residential buildings ensures familiarity with the designed building throughout all participating groups. A house or an apartment is something that everyone is familiar with as they lived in at least one of them and had access to the whole of it. On the other hand, a school or a hospital is not familiar in its entirety to the general public because even if a person has been for prolonged times in one such building, only a few (teachers, doctors) have experienced it in its entirety.

Third, there is an inherent conflict of interest with residential buildings that an algorithm cannot solve. Namely, people want to live in independent houses but often must live in higher-density environments for economic, social, and climate sustainability reasons.

Fourth, there is a high degree of self-similarity between the spaces that make up the whole residential building. An apartment is very similar to another apartment. Therefore it is easier to make rules for structured problem modeling that are concise.

Fifth, one apartment (cluster of spaces that houses a person or family) is relatively small compared to a whole building housing hundreds of people. Therefore it offers an ample design space with many possible combinations in which these units can be oriented and placed next to each other.

Finally, residential buildings have high organizational complexity due to their large number of stakeholders (mostly inhabitants but also neighbors, municipality, developers, etc.) This creates good conditions to explore the benefits and drawbacks of a broader involvement.

Role types

The case studies consider various roles of participants. They are represented on Figure 1.10 and defined below.

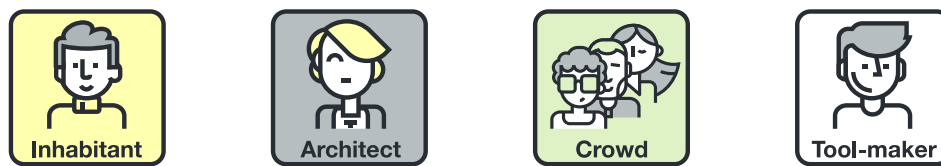


Figure 1.10 – Participant roles. Image credits: the author.

Inhabitants, i.e., Non-expert stakeholders The owners and the users of the buildings. The *Non-expert stakeholders* are characterized mainly by the fact that they will be living in the building. Compared to the crowd (of non-experts), they have different interests. The development of communication technology and democratic processes is redefining the role of the non-expert stakeholder. From simply a client who makes an order to the architect and provides a list of preferences, the non-expert is becoming a *produser* (from producer and user), a content creator, a maker (Kelty 2019).

Architects, i.e., Expert stakeholders Architects and engineers employed in the project. They define how the problem and the search for solutions will be structured. Over the last 50-70 years, the expert role of one who makes plans and designs is being challenged, and a new definition of that role, as someone who enables others to design, is being put forth (de Mul 2016; Lange 2016). According to Rittel, the experts cannot maintain a position of neutrality (Lange 2016), just like the very

definition of stakeholder means the pursuit of interest through the entity in which the *stake* is being *held*.

The crowd Third-party contributors who help explore the design space. In the case of games, those are players. The self-organizing crowd of enabled non-experts and experts emerges then as a separate category of participants, acting not in their capacity of a stakeholder but somewhat out of a desire for belonging to a cause or community (Kelty 2019; Rifkin 2015).

Tool-makers The developers of the generative design tool and the framework for structured problem definition and exploration. There is no distinction between the facilitator and the expert stakeholder in the traditional design process and current participatory design practices. Rittel defines the expert as a facilitator of argumentative processes (Lange 2016). In my case studies, the toolmakers are me and, sometimes, my collaborators and students.

1.4 Thesis Contributions

There is no practical precedence of how many non-experts enter architectural design. Investigating the thesis and the hypothesis will increase the relevance of architectural methodology to recent technological developments.

This dissertation’s contributions are:

1. A proposed Map of Design Paradigms at the intersection of *Participatory Design*, *Generative Design*, *Game Design* and *Crowd Wisdom* (See Figure 6.2) which reveals a significant research gap in architecture, as well as across all disciplines (See Figure 6.3);
2. The introduction of a unified taxonomy of generative design across the disciplines of architecture, computer science, and computer games (See section 3.2);
3. The introduction of a new type of Shape Grammars, namely *Playable Voxel Shape Grammars* (See section 8.3);
4. The identification of three use cases for games and crowdsourcing for schematic architectural design (See Figure 11.20);
5. Systematizing various game mechanics according to the balance of control between experts and non-experts over the design outcome of a crowdsourcing design environment (See Figure 11.16).

1.5 Thesis Outline

Part 1: FOUNDATION presents a review of the state-of-the-art in the four relevant fields: *Participatory Design*, *Generative Design*, *Game Design* and *Crowdsourcing/Citizen Science*. Four chapters present each field in detail, while the last chapter in this part discusses the intersections between them. The emphasis is on the systemic exploration of the variety of possible overlaps between the four fields

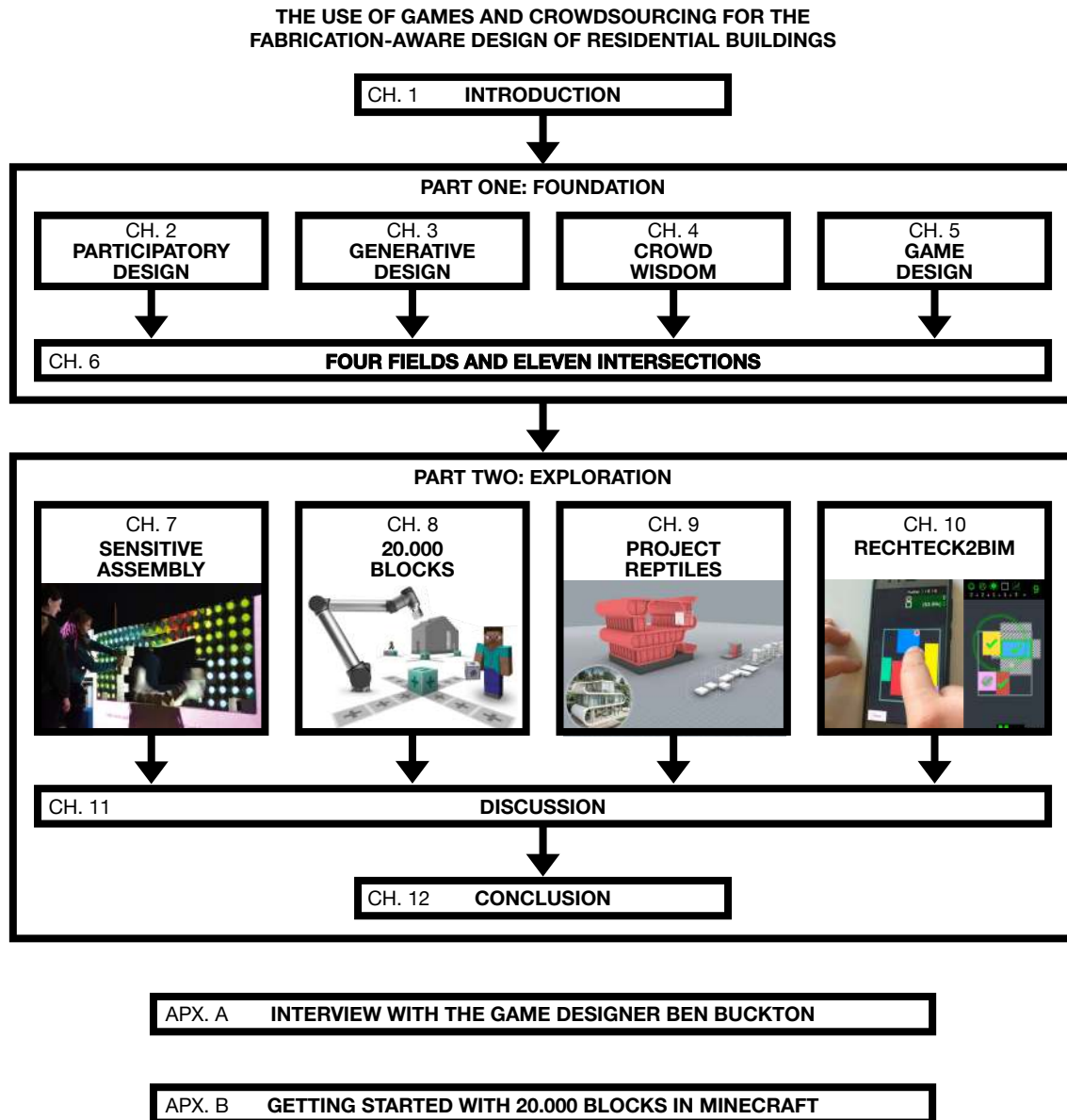


Figure 1.11 – Thesis outline. The dissertation is organized in two parts. Part one reviews the state of the art of the four fields of interest. Part two presents and discusses the case studies. Image credits: the author.

to identify the research gap. *chapter 3: Generative Design* contains a unified taxonomy of generative design techniques across all domains. The chapter identifies four generative design techniques suitable for creating crowdsourcing frameworks. These are: (i) iso-surfacing algorithms; (ii) physically-based models; (iii) set grammars; and (iv) case-based design.

Part 2: EXPLORATION presents the case study projects one after the other and ends with a discussion from the perspective of the research questions. The case studies are *Sensitive Assembly*, *20.000 BLOCKS*, *Project Reptiles*, and *Rechteck2BIM*. section 8.3 introduces a novel extension to the class of generative techniques called shape grammars. *Playable voxel shape grammars* evolved to form the basis of the qualitative research projects that I have conducted in Minecraft using *20.000*

BLOCKS from 2015 to 2018.

Appendix A includes an interview with my collaborator, the game designer Ben Buckton, illuminating in a conversational form some of the more abstract concepts from the field of games such as game mechanics, gamification, and the subject of game design.

Appendix B is, in essence, a guide on how to create a project using the Minecraft environment of *20.000 BLOCKS* that was given to participants in this case study.

Part I

FOUNDATION

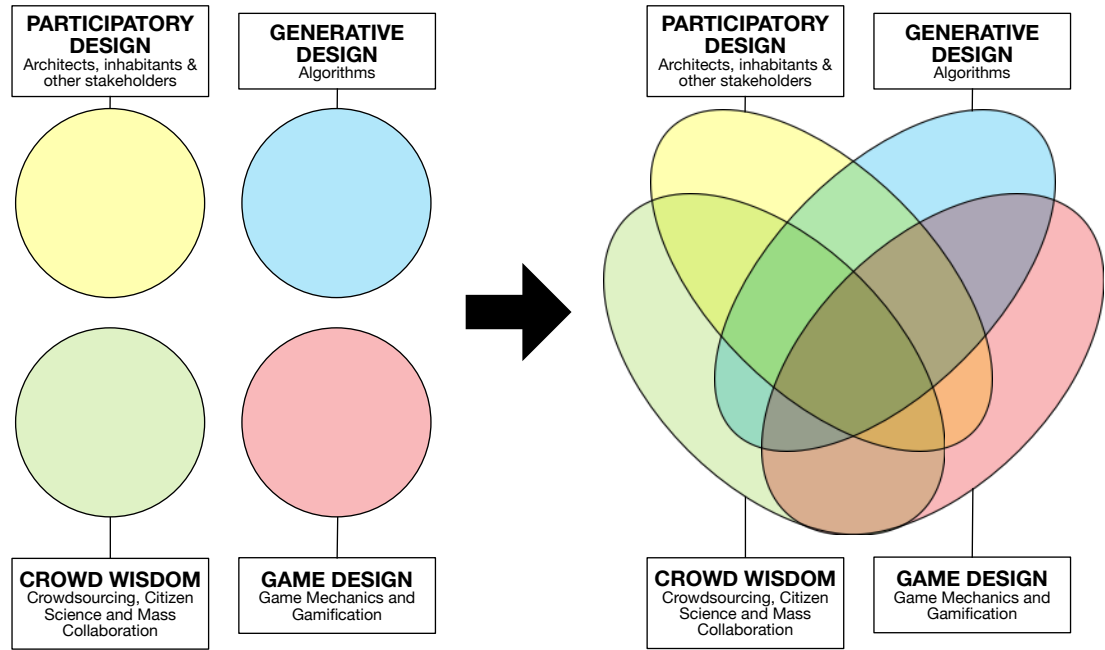


Figure 1.12 – Repeat of Figure 1.5. The four fields — on their own (left) and as a Venn diagram (right). Image credits: the author.

In this part, I review state of the art in four distinct yet overlapping fields: *Participatory Design*, *Generative Design*, *Game Design* and *Crowdsourcing/Citizen Science*. The emphasis is on the systemic exploration of the variety of possible overlaps between the four fields to identify the research gap (Figure 1.12). The four chapters that follow present each field in detail, while the fifth chapter in this part discusses the intersections between them.

As mentioned in *section 1.3: Methodology*, generative design techniques provide architecturally specific machine agency that forms the basis for interactivity, participation, and game mechanics in this work. Therefore the field of Generative Design gets a more dominant position in the following review and analysis. This has two purposes. First, I want to look at the full spectrum of generative design techniques to identify the suitable ones that adapt well to human-in-the-loop, game design practices, and highly concurrent co-creation. Second, I want to check for techniques that might have been overlooked in previous work. Such a new find is the newly emerging application of Marching Cubes and Wavefunction Collapse Algorithms in procedural game worlds.

Chapter 2

Participatory Design

This chapter introduces the fundamental concepts around participation and participatory design. By introducing several participatory architectural and urban projects, I argue that participatory design is an excellent platform for dealing with wicked problems but, at its current mode of practice, is overwhelming for both the experts and the non-experts. Furthermore, this chapter identifies the forms of input and engagement that a participatory design procedure demands, such as formalizing preferences definition, using games or game strategies as a medium for role-based co-production, and evaluation via evolving criteria.

2.1 Wicked Problems

The workflows of architects

Many studies of the architectural design practice exist as well as multiple theories of design have been put forth (Alexander 1964; Archea 1987; Brawne 2005; Cross 1982, 2011, 2013; Habraken 1987; Hays 1998; Johnson 2017; Lawson 2006a,b; Yaneva 2009). They are all different, just like no two architects would agree on a unified theory of design practice. However, they all agree on “the celebrated *intuitive leap*, that elusive but well-known moment when form and function seem to converge into a meaningful whole” (Kalay and Carrara 1996, p.108).

Nevertheless, it is essential to gain an understanding of how architects work. By understanding how experts create a design, we can start looking for answers to this thesis’s questions: Can design problems be split into clearly defined independent tasks? How can non-expert stakeholders be brought into the process? How can non-expert, non-stakeholders be brought into the process?

Singh and Gu 2012 bring attention to the consensus that design is a co-evolutionary process, i.e., the designer starts with an ill-defined problem. Their definitions of the problem and solutions grow and mutually guide each other as they work.

Meniru, Rivard and Bédard 2003 conducted a study with eight designers responding to the same brief and documented how each designer navigated between tasks such as Design brief, Site preparation, Building space, Building elements, and the tools they used. Highly individual from designer to designer, all top-down, from abstract to specific, use of multiple views (plans, section), all branched out to explore alternatives and merged back features of the alternatives into the final design.

While the goal of a design problem is ill-defined and every architect approaches it

differently, the architectural design process is composed of a varying series of goal-oriented cognitive activities with clearly defined sub-goals (Afacan and Demirkan 2011; Cross 1982; Do 2002). Most of these activities are, in essence, the production of a representational model (drawing, physical or 3D model, program description, etc.) of either the current understanding of the problem or of the current conceived possible solution.

The design problem commonly is expressed through two complementary concepts: the *design brief*, i.e., a specification of the need characteristics of the building; and *a set of aesthetic and phenomenological goals* that define the experience the building creates in its beholders, users, inhabitants, etc. (Elezkurtaaj and Franck 2002).

The central place of the intuitive leap in design theory results from, or causes, the inability for the design process to be formally modeled and explicitly represented. The reasons for this are accounted for in detail in the study of *wicked problems*, a concept developed in the 1970s by Horst Rittel and Melvin Weber (Rittel and Webber 1973).

What are wicked problems

Models of planning based on an idea of efficiency or optimization were in crisis already in the 1970s (Lange 2016; Rittel and Webber 1973). Problems of planning and design are “characterized by complexity, contradictions, and unforeseeable consequences” (Lange 2016). According to Rittel, a designer’s work has a *political* dimension since it results from the stakeholders’ and designer’s respective images of how the world is and how it ought to be (Lange 2016).

According to Rittel and Webber 1973 *wicked problems* have the following characteristics:

1. There is no definitive formulation of a wicked problem — problem formulations exist only in the context of a conceived possible solution.
2. Wicked problems have no stopping rule — work on the solution stops either when time or money runs out or when the designer finds the solution ‘good enough’ or ‘the best I can do within the limits of the project.’
3. Solutions to wicked problems are not true-or-false, but good-or-bad, i.e., subjectively and not objectively evaluated
4. There is no immediate and no ultimate test of a solution to a wicked problem
5. Every wicked problem is essentially unique, making every solution to a wicked problem a *one-shot operation*
6. Wicked problems do not have a finite number of possible solutions
7. Every wicked problem can be considered to be a symptom of another problem

Nancy Charlotte Roberts, a Professor Emerita at the Naval Postgraduate School, researching Design, Strategic Design, Wicked Problems, Terror Networks, and Organizational Studies, states that cultural, political, and technological changes give more and more central place of wicked problems in planning (Roberts 2001). She

distinguishes three types of problems. Type 1, simple problems, where stakeholders agree on the problem and on the solution to be applied; type 2, complex problems, where stakeholders agree on the problem but have no consensus on how to solve it; and type 3, wicked problems, where there is no agreement either on the problem or on its solution (Roberts 2001). Roberts goes on to offer three strategies for dealing with wicked problems based on the distribution and defiance of power: *Authoritative*, *Collaborative* and *Competitive* (Figure 11.1).

In the case of the traditional design process where the figure of the architect keeps all the power over the design (power is not dispersed), we have an *Authoritative* strategy. In *Notes on the Synthesis of Form*, Christopher Alexander argues that the growing complexity of design problems leads to an increasing body of information and specialist experience that needs to be taken into account when creating designs (Alexander 1964). Since a single designer cannot process all this information, design proposals are created out of a random selection of this information. Furthermore, Alexander argues that traditions are being dissolved, and architects cannot build upon their predecessors' work anymore as they could do until the Renaissance. Which tasks the architect with creating "clearly conceived forms without the possibility of trial and error over time" (Alexander 1964, p.4). "The intuitive resolution of contemporary design problems", Alexander writes, "simply lies beyond a single individual's integrative grasp". I regard this as true today, as it was in the 1960s.

We can find the *Collaborative* strategy in the case of housing cooperatives (Baugruppen, explained in the next section). The inhabitants are given the power to influence the design (power is dispersed). Yet, they do not contest the authority of the architect chosen by them to carry out the design.

It is challenging to point to a matching example at the scale of a building of the third strategy, *Competitive*. At the urban scale, there are many precedents for considering the city as the result of competition. As the American urban planner and theorist, Kevin Lynch, puts it, the city "is the product of many builders who are constantly modifying the structure for reasons of their own" (Lynch 1960). Probably informal settlements can be a fitting illustration, where there is no central designer figure, and everyone is competing for the limited resources of land, light, etc.

Models

According to Rittel, problems of design have an organizational character (Lange 2016). The architect who acknowledges planning as a wicked problem creates models of the project not as a solution but as a means for communication among the stakeholders, thereby allowing the co-evolutionary process of defining the problem and finding an acceptable solution to unravel (Lange 2016). Rittel focuses on two aspects of the use of models. First, models force planners to structure the problem and avoid decisions on a hunch, and second, the use of models as representations of opinions draws out the potential conflicting viewpoints among the stakeholders, which enriches everyone's understanding of both problem and solution (Lange 2016).

Takeaways

The following summarizes the qualities of the design process that must be respected to produce successful innovation in the field:

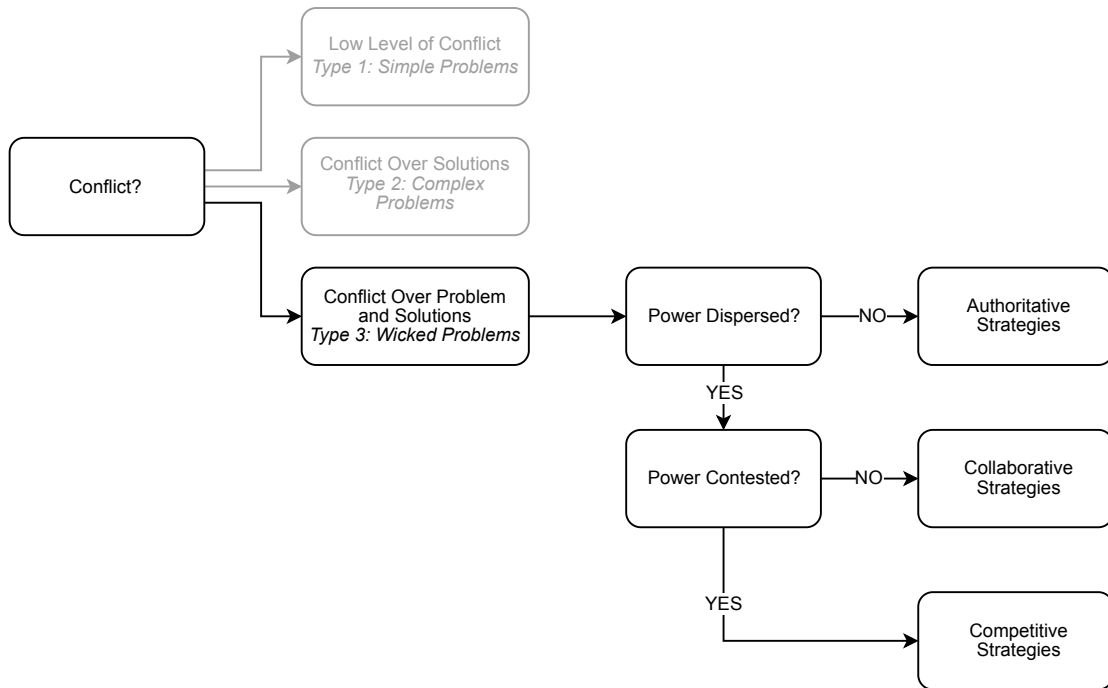


Figure 2.1 – The three strategies for solving wicked problems. Authoritative, Collaborative and Competitive (Roberts 2001). Image credits: the author after Roberts 2001.

- Every building is a design prototype even if tried and tested construction techniques are being used — Design is a co-evolution of problem and solution. Still, architects lost the benefit of gradual evolution over time by building upon their predecessors' work (Alexander 1964).
- No agreed-upon sequence of tasks or a set of tools constitute the design process. However, there exists a series of goal-oriented cognitive activities that have clearly defined sub-goals. This potentially opens up the opportunity for distributing them as tasks to multiple collaborators.
- Single authorship in architecture is less and less effective — overwhelming expertise leads to random selection while often key decisions are taken in early phase without enough information — encode expertise in algorithms and process information through massive participation.
- Participation in design is inevitable and of growing importance, following cultural, political, and technological changes in the last decades.
- When enlisting participants to contribute, use models to systematize design expertise and opinions.
- When employing algorithmic approaches, wicked problems' nature must be acknowledged – it is a trap to treat them as tame.
- The intuitive leap must be allowed to happen

2.2 Participation

As a procedure, participation is “a set of rules, techniques, and tactics for organizing people, issues, and things in the service of collective and equitable decision-making, getting things done, and or changing the way things go” (Kelty 2019).

Kelty 2019 offers a concise introduction to participation in his book *The Participant* and introduces the following cognate terms of participation:

- The verbs: Participate, Democratize, Engage, Collaborate, Cooperate, Involve, Include.
- And the nouns: Participation, Engagement, Cooperation, Collaboration, Involvement, Inclusion.

Cohen, Uphoff, et al. 1977 present several dimensions for participation. The first model has the dimensions *Who?*, *What?* and *How?*, where the *What?* dimension is divided into decision-making, implementation, benefits, and evaluation. The same authors in a later publication introduce two more dimensions on which participation works: scope and empowerment (Figure 2.2) (Cohen and Uphoff 1980).

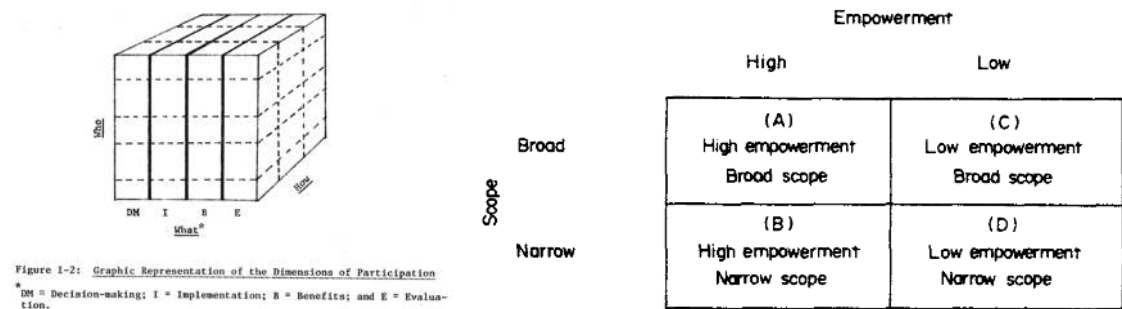


Figure 2.2 – The dimensions of participation. Left: according to tasks (Cohen, Uphoff, et al. 1977). Right: according to scope and power (Cohen and Uphoff 1980) Image credits: Cohen, Uphoff, et al. 1977; Cohen and Uphoff 1980.

One more dimension of participation in public policies is presented by Arnstein 1969, which the author calls the *Ladder Of Citizen Participation* (Figure 2.3). The *Ladder* can give a scale for measuring the *empowerment* axis introduced by Cohen and Uphoff 1980. It also clarifies that empowerment represents the balance of control between participants and facilitators/policymakers.

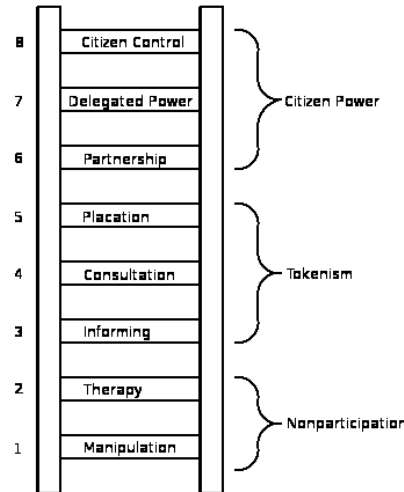


Figure 2.3 – The Ladder of Citizen Participation. Image credits: Arnstein 1969.

2.3 Participatory Design

In my research, I define participatory design as:

Participatory Design: a procedure that enables the involvement of stakeholders in the design and construction of a specific building or a city neighborhood.

Participatory design processes typically involve the roles of non-expert stakeholders (inhabitants, neighbors, citizens, business owners, etc.) as well as expert stakeholders (architects, engineers, urban planners, sociologists, etc.) (section 1.3.3).

Depending on the degree of *empowerment* and the *scope* of contribution, we could argue that all design is participatory. After all, it requires the client to provide the architect with a list of project requirements that influence the design outcome.

Unlike participatory art practices such as *Relational Aesthetics* where participation is a form of ad-hoc co-authorship with the artist to co-produce a work of art (Bishop 2012), in participatory design practices, has the participants' needs for their future use of the outcome of the participatory process in sight (Cross 1971; Kelty 2019).

Most of the research and practice on participation is focused on public space and public issues, with participants being citizens, policymakers, private partners, and other actors (Arnstein 1969; Hamers et al. 2017; Kelty 2019). However, residential buildings are rarely public. Instead, they are often privately, semi-privately, or collectively owned. In reality, the participatory design practice for residential buildings usually takes the form of cooperatives. Therefore the focus of this dissertation is concerned with participatory design practices mainly for the production of private, semi-private, or collective space. As this area has remained under-explored, case studies from public practices are drawn in.

An overview of participatory design practices in Scandinavian countries reveals that it moves from power-oriented to knowledge-oriented, sets focus on the process rather than the object, and centers more and more around the customer than the producer (Granath 2001).

Participation in design makes users able to take an active part in the inevitable

redesign and management of the designed environment (Granath 2001).

2.4 Participatory Design and Wicked Problems

The political and organizational character of design problems is broadly acknowledged in architecture (Lange 2016). In her research, Susanne Schindler, an architect, and historian who focuses on the intersection of policy and design in housing, investigates how policy, forms of ownership, and local regulation have a significant influence on the design process and outcomes of cooperatives as well as participatory public housing projects (Kockelkorn and Schindler 2019; Schindler 2017, 2020).

The design and construction of a building or a city is a wicked problem, and as such, its solution depends on those that will define it, design it, erect it and use it (Lange 2016; Rittel and Webber 1973). Participatory design promises that the “involving in the design process those who will be affected by its outcome, may provide a means for eliminating many potential problems at their source” (Cross 1971, p.6). According to Kelty 2019, p.1 the “power of participation, at its best, is to reveal ethical intuitions, make sense of different collective forms of life, and produce an experience beyond that of individual opinion, interest, or responsibility.”

Participatory design uses various *models* as an argumentative tool to negotiate conflicts between the multiple stakeholders and construction capacities (Lange 2016). According to Rittel, models offer themselves as a means of communication and require that experts structure the problem and synthesize the models (Lange 2016).

2.5 Examples of Participatory Design

The rapid development of social and behavioral sciences, economics theory, and information and communication technology since the 1950s has influenced and underpinned participatory design projects the world over (Cross 1971; Kelty 2019; Sanoff 1990, 2011; Simonsen and Robertson 2013). Three main characteristics of participatory design practice that follow this development can be observed:

1. The formalization of inputs, process and outputs
2. Role-playing simulations
3. Content creation

The characteristics are not isolated in practice but often go hand in hand in the same project.

2.5.1 The formalization of inputs, process, and outputs

The German phenomenon of Baugruppen, or housing cooperatives, is a group of several private parties, usually families, that pull together resources to purchase a plot of land, design a building that fits their preferences, and commission a contractor to build it for them. As such, they avoid middle man fees in the traditional developer model of housing construction and make sure they get an apartment responding to their preferences (Florian Köhl 2012; May, Ullrich and Steiger 2017; Ring 2007).

The Baugruppe project R50 in Berlin, designed and built between 2010 and 2013, by ifau and architects Jesko Fezer and Heide & von Beckerath (Figure 2.4), is a fitting example. The efficient layout of the flats, as well as the careful consideration of what shared rooms to have and where to place them, was developed jointly by architects and users in an elaborate and laborious planning process (Sauerbrei 2015). For this purpose, the architects asked the future residents in detail about their living needs (Figure 2.5), discussed these with them, let them sketch apartment layouts, and finally developed individually customized floor plans (Figure 2.6).



Figure 2.4 – The R50 project finished. Image credits: Heide & von Beckerath.

The provision of kits of buildings blocks for the stakeholders to arrange the building in a process directed by the architects is another strategy for participatory design. Figure 2.7 shows a floor plan layout kit for inhabitants prepared by BARArchitekten for their project Spreefeld Berlin¹ (Becker 2015). Such kits are great for helping the inhabitants visualize the spaces of their future home and daily life in it. However, the freedom they provide makes producing a feasible layout possible only with the architect's presence, guidance, and feedback. At the same time, they can feel quite constraining to the inhabitants since all options are systematized.

The Baugruppe is an excellent illustration of the emerging formalization of the participatory design of residential buildings, i.e., turning the otherwise very open-ended process into one confined by a system. That system can concern mainly the process, i.e., the way architects collect and document input from future inhabitants as shown by the survey forms and adjacency graphs created in the R50 project (Figure 2.5).

Additionally, that system can organize the layout and design of the building by prescribing the geometry of its parts as in the design kit in the Spreefeld Berlin project (Figure 2.7).

¹<http://www.bararchitekten.de/projects/sfb.html>



Figure 2.5 – Formalized participation in R50. Top row: Left, members of the R50 Baugruppe on one of their research tours. Middle and Right, the forms that members of the R50 Baugruppe filled out to document their preferences for their future flats. Bottom row: the surveys were used to create diagrams for the organization of the flats and for the properties of the common spaces that are then discussed with the whole group. Image credits: Heinemann 2011.

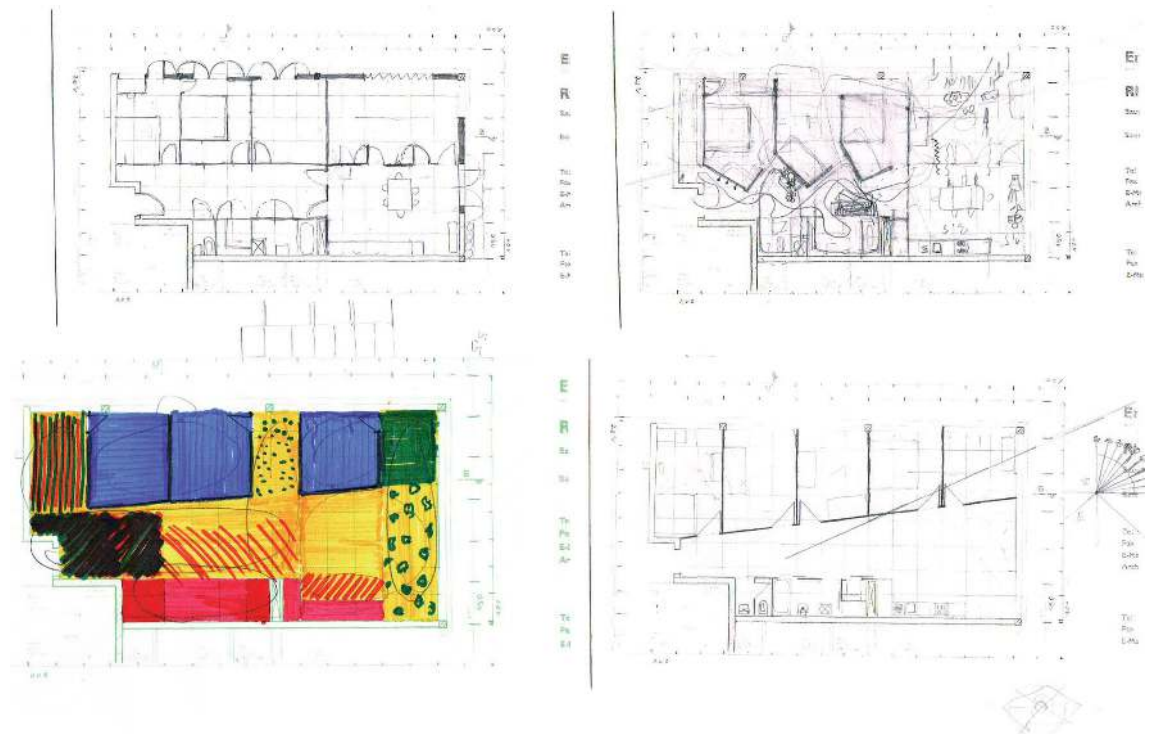


Figure 2.6 – Open-ended participation in R50. Each resident was given a blank apartment layout and asked to sketch options for its arrangement. These were taken into account by the architects for the final layout. Image credits: Heinemann 2011.

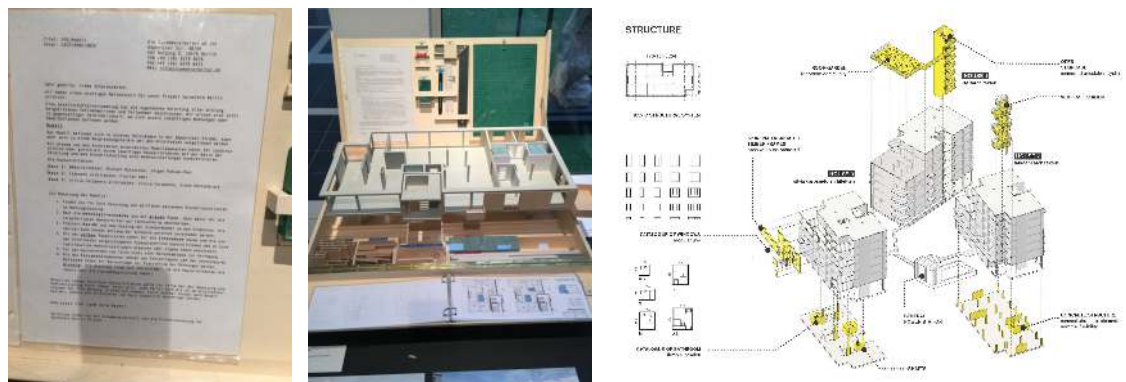


Figure 2.7 – Design kits. The boundary of the building and its structural scheme is defined. Inhabitants can place components out of a catalog of wet-room blocks, dividing walls as well as the facade panels with solid wall, with a catalog of windows and balconies. A catalog of sample layouts is provided by the architect in the book attached to the model. Image credits: the author; BARArchitekten.

2.5.2 Role-playing simulations

The educational benefit of using games as behavioral, social, and economic simulations has long been acknowledged (Dodig and Groat 2019; Jobs 1990; Tromba 2013). In the 1940s-60s, Buckminster Fuller drew on a metaphor of our world and everything we do in it as a game. *The World Game* imagined a computer game which would allow for the most intelligent distribution of resources (Fuller 1969).

Participation in the form of role-playing simulation games is much more common at the urban scale (Gaglione 2018; Mayer 2009; Tan 2014, 2017; Wynn 1985). According to Tan 2014, p.133, a “number of visionary urban experiments include elements of gaming, and point towards its promise as a generative method for cities.”

Wynn 1985 has used games in city planning. The problems are similar but different than in the design of buildings. The approach Wynn took is Case Study Simulations based on the case method of Harvard Business School (McNair and Hersum 1954).

Games offer a good mix of top-down and bottom-up participation opportunities, which corresponds well to the complexity of a city’s genesis (Tan 2014).

The architect Ekim Tan focuses on the shift from participation to self-organization enabled through gaming and the collective intelligence of players (Tan 2014). A continuation of the same line of research as Wynn 1985. Tan observes that games can simulate the self-organizing mechanisms of cities while unlocking conversations, conflicts, and their resolution by testing ideas and rules (Tan 2014).

At the same time, the direct correspondence between a game outcome and a chosen city design in Tan’s work (Figure 2.9) shows the constraining influence that the design of game components and rules, and by extension the facilitators, might have if a game run is seen merely as a design generating process. Documenting patterns over several games and the needed creative reinterpretation of the game outcome must still find their place in the process.

Game simulations are also used at the building scale. An example of involving a broader spectrum of stakeholders in the design process is how Susanne Hofmann and her practice Baupiloten uses participation to define the project requirements before design starts (Figure 2.10) (Hofmann 2018, 2019). Hofmann believes that participation creates architectural quality by fostering invention, social cohesion, and saving resources. The Baupiloten made a game for collaboratively developing programmes for schools (Figure 2.11) (Hofmann 2018).

“Die Baupiloten works with communities to develop briefs regarding desired spatial qualities and arrangements. The generic School Vision Game tool – which the practice devised in cooperation with the Hans Sauer Foundation, Munich – enables a school community to explore their complex and divergent needs and creatively negotiate and develop a shared pedagogical-spatial scenario, in 17 steps and 90 minutes. The participants are guided through the game with the assistance of action cards. This focuses the discussion on the essentials. Prejudice can be dissolved, and seeming discrepancies and conflicts transformed into synergetic potential. The results of the game offer an effective analysis of the future building’s spatial organisation, including its atmospheric qualities.” (Hofmann 2018)





Figure 2.11 – The Negotiating Game for School Development. Image credits: Baupiloten.

2.5.3 Users as content creators

The third, and probably most relevant, characteristic of participatory design practices is that the users of a building or an urban space are given a chance to co-design it.

The basic means for this is to provide inhabitants with the media and tools that an architect would use, e.g., a floor plan and a set of colored pencils, and ask them to express an idea graphically. This can be seen on Figure 2.6, where the inhabitants of the R50 Baugruppe project were asked to create options for the layout of their apartments.

Expecting inhabitants to possess the skills and experience expected from an architect perhaps stands in the way of genuine participation. Nevertheless, the schematic drawings that non-specialists can produce serve to visualize and understand limitations and tradeoffs. Furthermore, the user-created floor plans, shoebox models, and other artifacts offer themselves for interpretation by the architects in the final renditions of the design.

At the urban scale, according to McDaniel 2018 from the project Block by Block, the act of co-creation is “improving community engagement in the public meeting process, both across the board and for most underrepresented communities.”

Block by Block², an initiative started in 2013 by UN-Habitat and Mojang, the makers of the game Minecraft, and in the meantime already established its own foundation. In Block by Block, Minecraft was used as the medium where locals can model ideas for the rebuilding of their community spaces and virtually walk around them and discuss (Figure 2.12). Block by Block utilizes a 12-step method for its initiatives which includes (Block by Block team 2021):

1. The facilitators prepare the Minecraft model of the chosen site by
2. and select 30-60 locals as participants.
3. A 2-4 day workshop is organized where
4. participants are briefed about public space and design considerations,
5. walk in the area to observe and share reflections, then
6. participants are introduced to Minecraft basics and
7. a Minecraft modeling session in teams of 2-4 takes place to develop ideas followed by

²<https://www.blockbyblock.org>

8. a presentation in front of stakeholders, architects, and local policymakers.
9. A prioritization phase where participants and stakeholders collaborate to fine-tunes the proposals.
10. A planning phase that uses the Minecraft models to inform cost-estimates and professional design work,
11. A build phase to realize the project including again the local community and
12. finally, using the project to advocate for additional forward-thinking policies.



Figure 2.12 – Block by Block. Left: People Rebuilding Safe Parks In Johannesburg; Middle: the locations where Block by Block was active; Right: the key central square and pond in Kirtipur, an ancient, traditional settlement in the Kathmandu Valley region of Nepal in reality and as modeled in Minecraft. Image credits: Joakim Formo/Ericsson; Block by Block.

Finally, the ultimate form of participation through user content creation, in both design and construction, is the informal or non-planned settlement (Figure 2.13) or the phenomena of self-built structures (Mees 2017). Dating from the dawn of civilization, informal settlements engage the capacity of all inhabitants and citizens over time to produce ever-evolving, organic organizations that reflect local culture, local materials, and the relationship with natural conditions (Rudofsky 1964; Schaur 1991).

According to Eda Schaur, whenever human beings create a settlement without a comprehensive plan, they can, within the constraints of their cultural and social relationships, satisfy their needs and wishes and allow their settlements to develop as an entity as a result of actions and reactions (Schaur 1991, p.18). Schaur describes a process of self-organization that relies heavily on the ability of all people involved to communicate directly. At the same time, studies by Robin Dunbar have shown that the size of a cohesive group of people able to maintain stable social relationships is around 150 members (Dunbar 1992). Dunbar’s proponents suggest that larger groups are generally required to build their information exchange on more restrictive rules, laws, and enforced norms to maintain stability and cohesion within the group.



Figure 2.13 – Fès, a non-planned settlement in Morocco. Image credits: Georg Gerster.

2.6 Conclusion

Current *participatory design practices* acknowledge the true nature of wicked problems, taking into account stakeholders' values and preferences. As seen above, they incorporate this input by sandwiching the traditional design phases between a *Phase Zero* and a *Phase n+1* (Granath 2001; Hofmann 2018, 2019; Poplin 2012; Sanoff 1990, 2011; Simonsen and Robertson 2013).

Out of its 12-step method, step 10 represents the entire traditional design process with all its phases and challenges (Block by Block team 2021). The additional 11 steps, nine before and two after the architect's job is usually done, ensure participation.

Participation, as seen in the work of Susanne Hoffmann and Block by Block, requires the so-called *Phase Zero*. In my opinion, such examples show that the strictly defined *Design Stages* (German: Leistungsphasen) might be outdated and lead to an ever-increasing exclusion of the building user from the design process. It also shows that participation is relevant at the very early stage, when the problem, and not only the solution, needs to be defined.

Looking at participation in general, it is being standardized and formalized through platforms, algorithms, and software, which takes away its edge as a mechanism to introduce collective and equitable decision-making Kelty 2019.

“But in the twenty-first century, participation is more often a formatted procedure by which autonomous individuals attempt to reach calculated consensus, or one in which they experience an attenuated, temporary feeling of personal contribution that ends almost as soon as it begins.”
(Kelty 2019, p.1)

The threat of technology as a constraining chain is probably not yet as relevant in a disciplinary-specific way. Instead, the opposite is true. Algorithmic assistance and online platforms are much needed to enable experts and non-experts to overcome three obstacles in participatory design.

First, experts must become tool-makers and develop tools and methods such as games, survey forms, and model kits to stage a participatory design process. Very little exchange of methods among architects can be observed in practice, leaving

every architect to start from scratch should they want to stage a participatory design process. Furthermore, the custom-developed tools are usually not well integrated into the design workflow of the architectural practice, are often analog, and do not produce representational design artifacts, leaving the architect with an extra effort to develop yet more methods of input capture.

Second, generating content requires non-experts to possess expert skills or quickly learn them, which is unrealistic in the few days a typical participatory workshop lasts. The use of participation to develop project briefs is an effective way to make sure the clients understand what they want and be able to express it and judge the value of a proposed design. Yet it is often expected from participants to be design experts. Otherwise, they are not given many options. While the components of a project, whether programmatic units as in the case of Baupiloten, or building parts as in the case of Spreefeld Berlin, are made explicit and tangible to the participating future users of the building, the reason and expertise for combining them in a practical way that delivers a well-functioning and good-looking building is still reliant on the participation of the architect and not made explicit. The expert knowledge is hardly ever made explicit.

And third, the input from the participants in the form of requirements, discussions, game outcomes, and design alternatives constitutes a diverse data set that an algorithm cannot process. This leaves the need for a human expert, an architect, to process and synthesize the results, which often comes to the limit of a single human's capabilities. Facing information overload, a design synthesis is either skipped as in the work of Tan (Figure 2.9) or is based on a random, manageable selection of all the inputs (Alexander 1964, p.4), rendering the participatory process mute.

The use of design systems and generative design techniques in combination with automation in construction, reviewed in chapter 3, can potentially counter these problems.

Chapter 3

Generative Design

"The Machine is the architect's tool — whether he likes it or not. Unless he masters it, the Machine has mastered him.

The Machine? What is the Machine?

It is a factor Man has created out of his brain, in his own image — to do highly specialized work, mechanically, automatically, tirelessly, and cheaper than human beings could do it. Sometimes better." (F. L. Wright 1927)

From the dawn of the industrial age, architects have tried to encode the design process into rule-based systems of various levels of complexity. The primary purpose of this chapter is to identify which generative design techniques are suitable to explore my research questions.

First, it is interesting to explore which techniques can automate tasks that alleviate one of the core roles in the design process - expert and non-expert stakeholders and third party contributors - from having to perform those tasks. Concerning this, I look at which techniques can enable a given role to perform new, more challenging, design tasks.

Second, despite advances in generative design, most design problems remain computationally intractable. One of the goals of this work is to determine to what extent it is possible to stage a human-algorithm collaboration to create architectural designs. It is essential to investigate the potential for interactivity of the various generative design techniques presently available.

Finally, of particular interest is the techniques' potential for realizing the designs it creates with technologies for construction automation, i.e., robotic construction.

With the advent of Computer Science and Information Theory in the 1960s, much criticism of the lack of explicitness in the traditional design process emerged. A never-ending yet ill-fated effort began. Namely to bring in the architectural problems under the domain of formalizable, tame problems (Alexander, Ishikawa and Silverstein 1977; Y. Friedman 1980; Kalay 2006).

In parallel, studies of the design process, and more specifically of the wicked nature of design problems, most notably through the work of Horst Rittel (Lange 2016) have emerged. These are reviewed in more detail in section 2.1. Some of the efforts to create design systems and design automation, such as the work by Nicholas Negroponte and the Architecture Machine Group on Urban 5 (Negroponte 1970), stayed true to the nature of what architecture and architects do. And lately, with

the maturation of the Computer Science field, much more integrative approaches have emerged to bring architecture and computation together (Johnson 2016).

The emerging man-machine co-creative approach, as well as the growing appreciation of computers as communication platforms instead of merely using them as autonomously running machines, relate to how Rittel perceived *models* as something much more than representational devices.

3.1 Design Systems in Architecture

To use technology in ways enhancing creativity, others have aimed to encode the design process into a scientific, objective methodology (Alexander, Ishikawa and Silverstein 1977; Alexander 1964; Durand 2000; Y. Friedman 1980; Gropius 1965).

You can see this in the work of Jean Nicolas Louis Durand and Walter Gropius Figure 3.1. Both looked for ways to standardize the accepted aesthetic look and construction methods into modular systems.

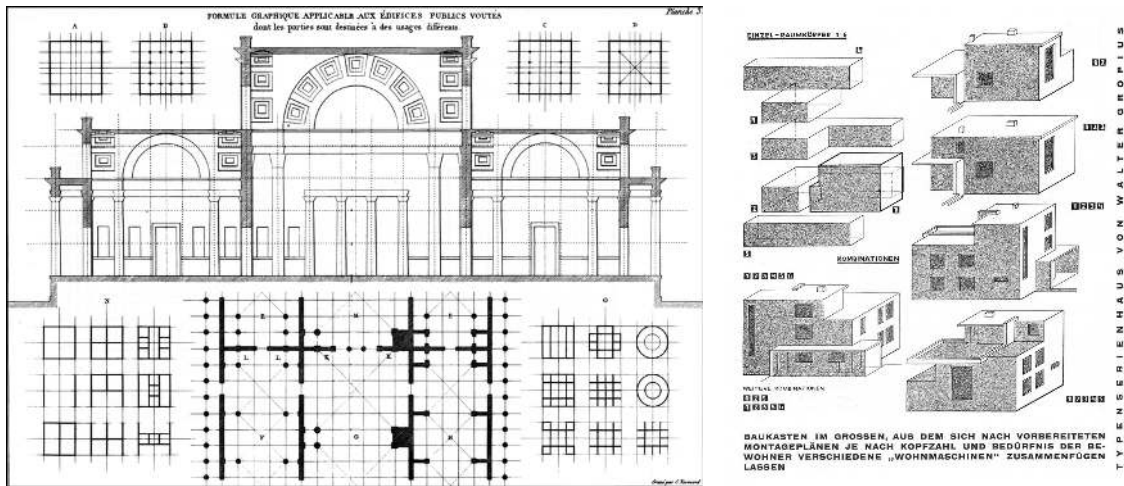


Figure 3.1 – Design systems. Left: Combinatorial modular grid, Jean-Nicolas-Louis Durand, 1821, Right: Baukasten im Grossen, Walter Gropius 1923. Image credits: Durand, Gropius.

Flatwriter by Yona Friedman

Yona Friedman was the first to try to democratize architecture by combining a modular design system with non-expert participation via a fictional computer device called the *Flatwriter* (See Figure 3.2) (Y. Friedman 1980). Using a keyboard, the user - a future inhabitant of a block of flats - designs their habitat of choice, with each key corresponding to a room type such as the kitchen or bathroom (See Figure 3.3). Once the plan has been submitted, all participating users are informed of any change in the flats around them should they want to adjust their layout.

The Flatwriter failed to become real. Firstly, because the technology to realize it was not yet commonly available. Second, and most importantly, it failed because the available configurable options were too constrained. The elements to which architecture had been reduced — rooms of various basic shapes, kitchen and bathroom equipment’s location, and apartment schemes — were too simplistic and reductive. Such modularization could not offer the fine palette of spaces responding

to the client’s emotionally-rooted preferences that a high-skilled architect can offer. It did not tap into the power of creative intuition. And third, it also had a built-in problem with realizability. The more complex the organization became in response to user preferences, the more complex the structure and infrastructure. Resolving all the manufacturing conflicts would have required such an extensive analysis by experts that the method would not justify its use.

Nevertheless, *Flatwriter* is the prototype that inspired generations of architects who envisioned a more participatory way of creating buildings. And today, computer games finally offer us a way to do this. After every player action, the game offers us a fully simulated result. We can judge it by its properties. We can learn from it and make it better iteratively.

The Generator Project by Cedric Price (Figure 3.4) is an example of using design systems in architecture to respond to, or trigger, perceptual changes in the mind and feelings of the inhabitants (Pertigkiozoglou 2017).

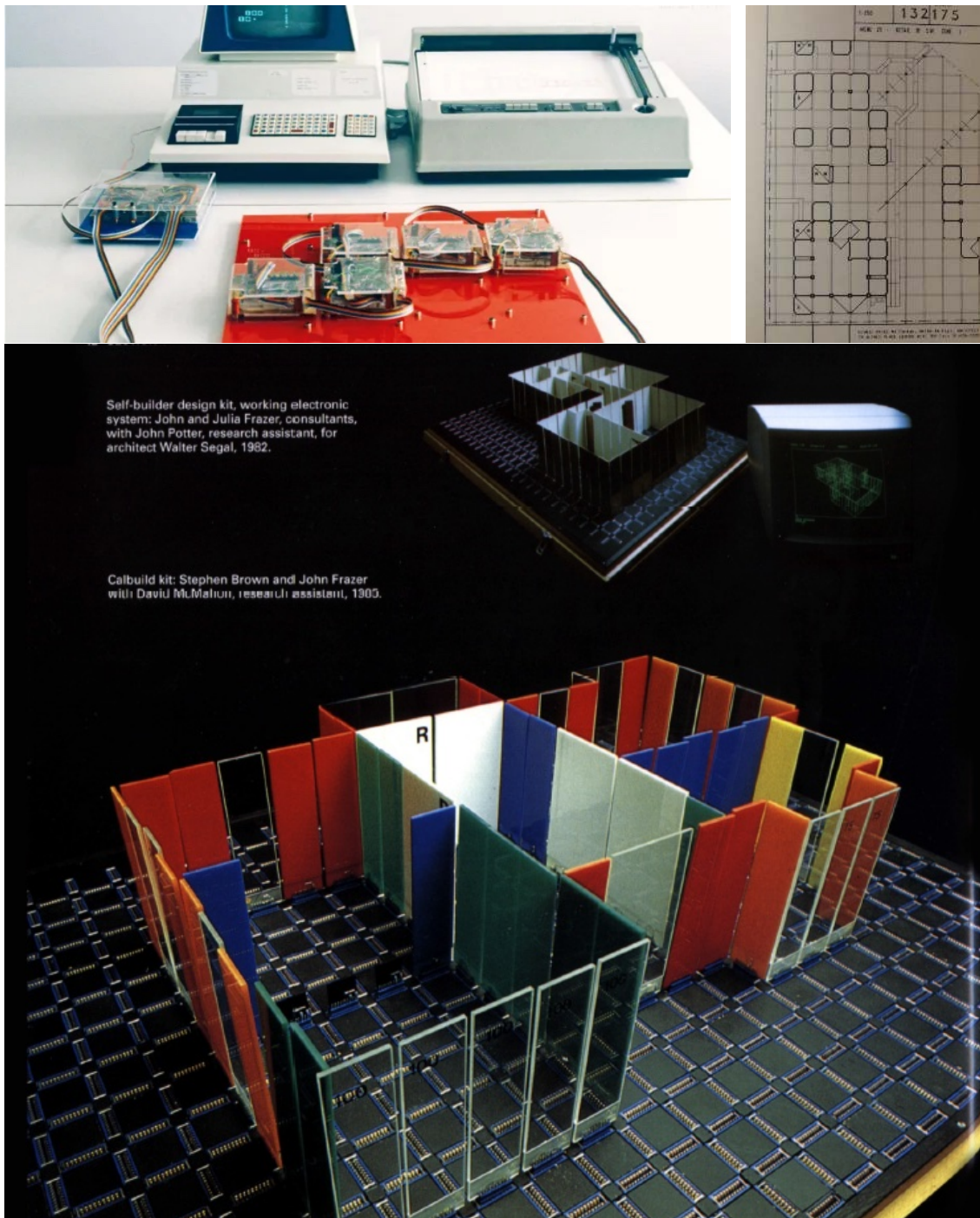


Figure 3.4 – Generator. The Generator project by Cedric Price (top) and a prototype build based on it by John Frazer (bottom). Image credits: Cedric Price, Frazer 1995.

Product Configurators

Here I present an overview of four existing strategies for encoding design content into digital modeling environments.

I looked at: 147 online configurators, targeting architectural and interior design; 104 CAD & 3D software products; and several concepts for procedural generation of design geometry. I sorted those into four strategies for encoding design content into modeling environments: *Implicit Modeling*, *Parametric configurators*, *Part Assembly* and *Procedural Generation* (Table 3.1).

Table 3.1 – Strategies for encoding design content. The amount of control offered to the experts and the non-experts in the four strategies for encoding design content into interactive modeling environments. Assisted sculpting, proposed in this work, is included for comparison. Data sources: Wikipedia 2019, cyLEDGE Media 2018. Image credits: the author.

	Implicit modeling	Parametric configurators	Part assembly	Procedural generation	Assisted sculpting
DESCRIPTION	The user can freely define all geometric, material and compositional characteristics of the final product. There is no distinction between expert and non-expert. Any CAD or 3D modeling software is in this category.	This is the classical car configurator. The user can configure the final product by selecting from limited options for colors and by specifying the geometric parameters of its parts (size, silhouette curve) which the expert has defined beforehand.	The user can configure the final product by arranging parts into a composition. The designer defines the parts and the rules to combine them.	Users edit rules as text or in a visual coding editor. Experts program rules and model parts as well as the generative algorithm itself.	The user can “sculpt” the composition of the final product into a voxel grid while the configurator takes care of the correct selection of tiles. The expert has modeled the possible tiles and defined rules for their placement.
APPLICATIONS WITH THIS FUNCTIONALITY	104	144	19	na	na
EXAMPLE	Rhino	Bene Chair, Tylco shelves, Anylamp, ShapeDiver	Traumgarten Playground equipment	Grammars, L-Systems, Wavefunction Collapse	An example is presented in this paper
Degree to which users can express own ideas from encoded content	Very low	Low	High	Medium	High
Degree to which experts can encode design ideas	Very low	High	Medium	High	High
Challenge experienced by user	High	Low	Medium	High	Low
Principle 1: Don’t let user start from scratch	No	yes	No	No	Yes
Principle 2: actions with Traceable, High impact	No	yes	No	no	yes
Principle 3: Give feedback on last action’s impact	No	yes	yes	no	yes

Let’s look at the four existing strategies for encoding design content in detail.

First, the conventional, implicit architectural design process is represented by all professional CAD or 3D modeling software. In this way of working, it is practically impossible for the expert to create a design system, i.e., to encode their design ideas into the modeling environment so that others can use them to create variations on the same design idea. However, to create design content for our interactive design system, it is beneficial to keep the experts in a 3D modeling environment that they are already familiar with.

Second, encoding designs into parametric configurators emerged with the phenomenon of Mass Customization. Mass Customization is a contemporary method in the product industry that allows companies to provide individually-designed prod-



Figure 3.5 – Online house configurator. The online configurator of the German startup *Mr. And Mrs. Homes* allows customers to select floor and wall finishings as well as resize the house floor plan to fit their site. Image credits: *MR + MRS HOMES* 2020.

ucts and services to customers at low production costs and in an industrialized method of production (Pine 1993). Online configurators allow non-experts to participate in the design process of their very own product. A classic example here is the car configurator. The user can configure the final product by selecting from limited options for colors and specifying its parts' geometric parameters (size, silhouette curve). An example of a configurator in architecture is the startup *Mr. and Mrs. Homes* (Figure 3.5). The creative control in this encoding strategy lies with the designer, the expert, who creates the content. The user has limited means to express their preferences and intents. Relevant for our approach is that users don't start from scratch and their actions have a traceable, high impact on the final design.

The third is encoding design ideas as an iterative assembly process where the users can configure the final product by arranging parts into a composition. Good examples here are the configurator for Traumgarten playground equipment as well as Lego toys (Figure 3.6). The most noteworthy characteristic in this strategy that I aim to keep in my case studies is the high degree to which the users can express themselves. Think of the fantastic creations players can create with Minecraft or Lego. Another benefit is that the control over the final design is equally balanced between the user who assembles and the designer who defines the shapes of the parts and the rules for their combination beforehand. On the other side, this strategy for encoding design can be challenging for the users as one player's action (place a block or a Lego piece) does not impact the modeled design, requiring thousands of clicks to produce a result. This demands expert-level skills from the user to plan the modeling process.

Fourth is encoding design ideas into procedural generation algorithms reviewed in more detail in the following section.



Figure 3.7 – Examples from the Pattern Language. Image credits: Alexander, Ishikawa and Silverstein 1977.

Alexander’s pattern language that they defined software design patterns. And those were a game-changer in programming.

3.2 Taxonomy of Generative Design in Architecture. With a Review 1970-2020

This section proposes a taxonomy for the field of generative design (Figure 3.8) that encompasses all techniques relevant for the computational creation of buildings. Each technique is presented with a short overview and examples. The overview helps identify:

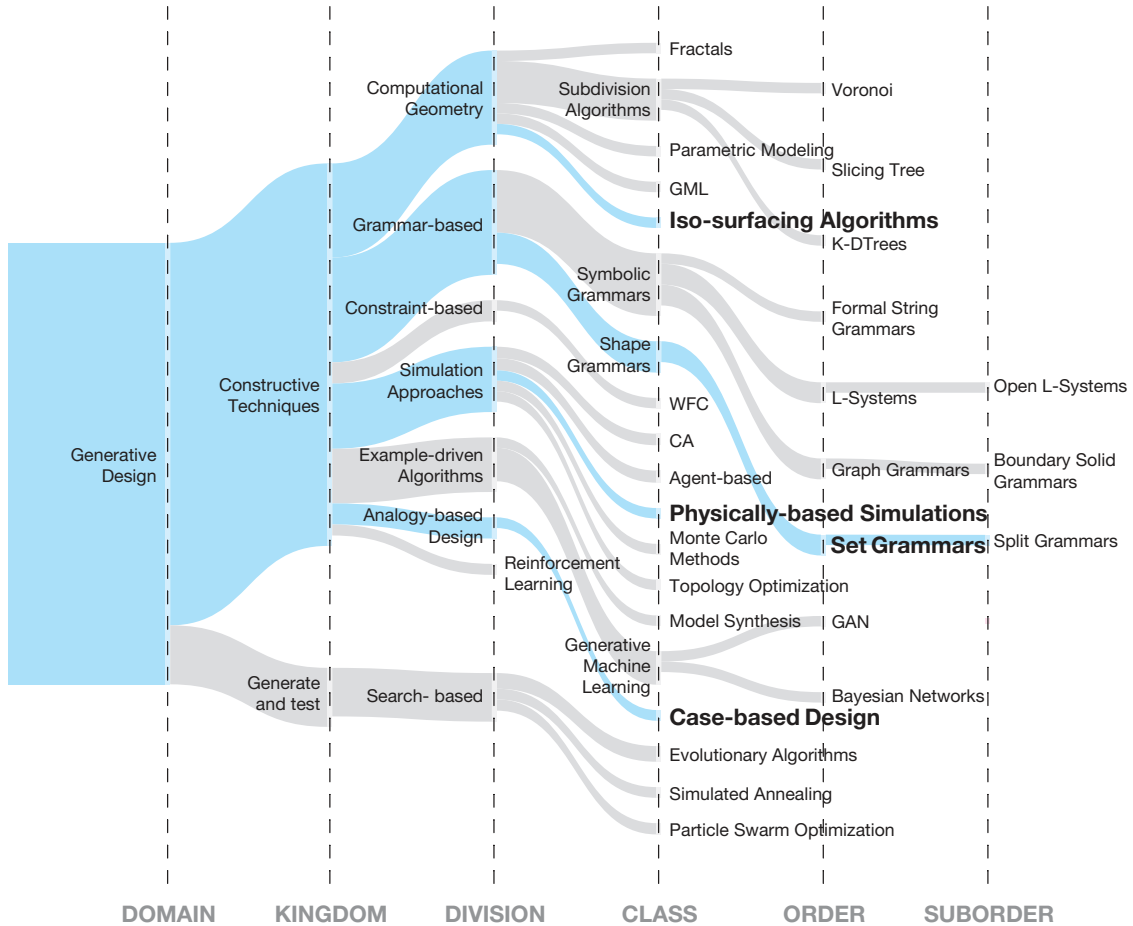


Figure 3.8 – Taxonomy of Generative Design in Architecture. See text for argumentation. Image credits: the author.

- The distinct steps that every generative technique has: setup, initialize, run, post-process, learn.
- Which techniques are better suited for the generation of buildings
- Which techniques are better suited to encode expert design knowledge
- Which combination of techniques show great potential and where that potential has not yet been explored in research.

3.2.1 Motivation

To stage a design process where individuals without prior training in architecture and design can contribute, expert knowledge and skills must be provided in some form either via the participation of experts or through automation. The review presented in this section focuses on the second option, i.e., the techniques to encode expert skills and knowledge in computational models.

Mountstephens and Teo 2020 state that “Generative Design systems produce designs by algorithms and offer the potential for the exploration of vast design spaces,

the fostering of creativity, the combination of objective and subjective requirements, and the revolutionary integration of conceptual and detailed design phases”.

To systematically explore the potential for using generative design techniques as an assistive method, one needs a map of the field of generative design.

The widely adopted *ACM Computing Classification System* by the Association for Computing Machinery (ACM) (*The 2012 ACM Computing Classification System* 2012) is the de facto ontology for the computing field. However, since it encompasses all concepts and methodologies related to computer science, it lacks the fine granularity needed for a comprehensive overview of techniques specifically for generative design.

The novel contribution of this survey is the inclusion of reviews and works from architecture, engineering, computer science, and computer games. Previous studies have focused only on one at most two of these fields. An example of the need for such a cross-discipline review is, for example, the works by Nam and Le 2012 and Sobey, Grudniewski and Savasta 2021 on generative techniques for generating layouts for yachts. Both works are relevant for architects as they address the generation of multi-story layouts under predefined constraints. However, they have been published in Marine Engineering journals and remain largely unnoticed in the architectural generative design field.

Even though surveys and reviews date back as far as 1975 (Mitchell 1975), I chose to review all the literature and not only the recent developments. The reason is that online databases’ new search possibilities could lead to discovering and putting together a collection of works that reveals new patterns.

The goal of establishing a taxonomy of generative design techniques is twofold. First, the terms in each category will be used to form the search phrases to source the works for review. A simple search with the common keywords *generative design* or *procedural generation* will not cover all works since many publications use in their titles and abstracts only the terms defining the specific generative technique they are presenting. Therefore to establish the search terms, I carried out an extensive review of surveys attempting to cover all available generative techniques and isolate their key terminology to use in the search.

Second, a taxonomy gives the possibility to argue for the potential of interactivity, or human-in-the-loop not only of individual works but also for the overarching technique category they belong to, which will be explored in the next section.

Much of the work in generative design has aimed at automating design processes to speed up the development of computer visualizations and game content creation. However, this non-inclusive approach ignores the nature of design problems, namely that they are wicked and require the involvement of stakeholders to formulate the problem definition and the criteria for acceptable solutions. Therefore, it is of interest for this review to uncover the areas in the generative design field with high potential for the involvement of designers and non-designers.

Mountstephens and Teo 2020 distinguish among many purposes of generative design three which are relevant here:

1. the automatic generation of a large number of designs,
2. the interactive generation of a large number of designs, and
3. the generation of fabrication-aware designs (additive manufacturing, prefab, or other).

3.2.2 Definitions

There is a lack of commonly accepted terminology in the field of generative design spanning the domains of architecture, engineering design, games (Chakrabarti et al. 2011).

In Generative Design actual design content is generated by automatically or interactively executed algorithmic descriptions of design procedures (Caetano, Santos and A. Leitão 2020; Mountstephens and Teo 2020).

Generative Design is distinguished from Parametric and Algorithmic design, albeit overlaps in their definitions exist (Figure 3.9).

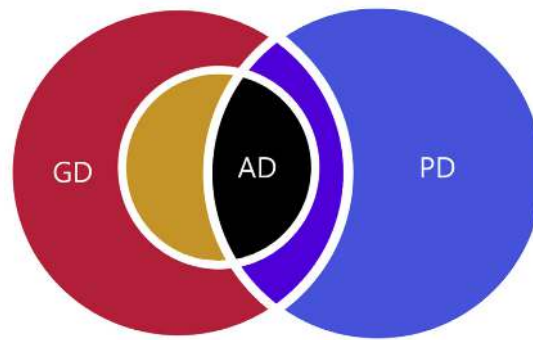


Figure 3.9 – Generative Design. According to Caetano, Santos and A. Leitão 2020 Generative Design (GD) comprises of Algorithmic Design (AD) and some techniques from Parametric Design (PD). Image credits: Caetano, Santos and A. Leitão 2020.

The first examples of a computer implementation of generative design techniques in architecture are the work by Armour and Buffa 1963 on solving the facility layout problem from 1963, the work by (Miller 1970) on computer-aided space planning from 1970, and most notably (Negroponte 1970) from the Architecture Machine Group (Figure 3.10).

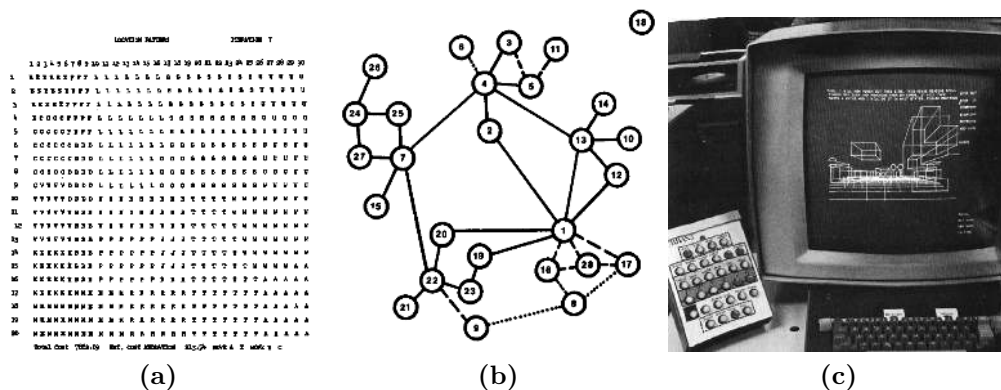


Figure 3.10 – Early examples of generative design in planning and architecture. a) a character matrix representing department locations for a facility layout produced using the generate-and-test heuristic algorithm with simulation by Armour and Buffa 1963, b) a bubble relational diagram of the departments of a branch bank in Southern California generated from an block diagonal matrix by Miller 1970 and c) the station running URBAN 5 man-machine designing environment by Negroponte 1970 Image credits: Armour and Buffa 1963, Miller 1970, Negroponte 1970.

In computer graphics and visualization and the game developing industry, the more widely used term is Procedural Content Generation (PCG) instead of generative design. Togelius, Preuss, et al. 2010 define PCG as the automatic or semi-automatic generation of game content. Of interest for this review are the techniques used for the generation of 3D models and 2D or 2.5D building layouts of cities and buildings. The simulation of urban growth of the city of Detroit by Tobler 1970 is probably the first example of generative design in computer graphics (Figure 3.11). The game *Rogue* is considered to be the pioneer in procedural generation in the game industry (Barton and Loguidice 2009; Yannakakis and Togelius 2011). It automatically generates dungeons for the player to explore.

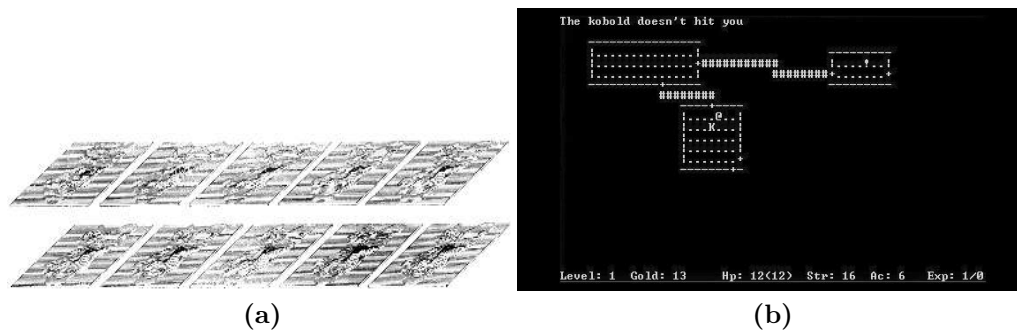


Figure 3.11 – Early examples of generative design in computer graphics and games. a) Simulation of Urban Growth from Tobler 1970; b) A typical procedurally generated game level with three rooms linked with corridors, the player’s character (the @ symbol) and an enemy (represented by the K) from the 1980 game *Rogue* (Barton and Loguidice 2009) Image credits: Tobler 1970, Barton and Loguidice 2009.

3.2.3 Existing surveys, classifications and taxonomies

Previous reviews of generative design techniques (Table 3.2) exist either as survey for a specific problem (space planning, cities, virtual worlds) (G. Kelly and McCabe 2006; Liggett 2000; Lobos and Donath 2010; Meller and Gau 1996; Nisztuk and Myszkowski 2018; Pérez-Gosende, Mula and Díaz-Madroñero 2021; R. M. Smelik et al. 2014), a specific technique (genetic, shape grammars) (Garcia 2017; Gu and Behbahani 2018; Knight 1999; Togelius, Yannakakis, et al. 2010; Tyflopoulos et al. 2018; Watson and Perera 1997), a combination of a specific problem and a technique (mostly surveys on space planning using genetic algorithm) (Calixto and Celani 2015; Dutta and Sarthak 2011; Sharma and Singhal 2014), general surveys (Caetano, Santos and A. Leitão 2020; Chakrabarti et al. 2011; Chase 2003; Ebert et al. 2002; Grobman, Yezioro and Capeluto 2009; Lara-Cabrera, Cotta and Fernandez-Leiva 2013; Mitchell 1975; Mountstephens and Teo 2020; Singh and Gu 2012), as well as notable sections in articles on original work or PhD and Master theses (Chau et al. 2004; Havemann 2005; Hoisl and Shea 2011; T. Kelly 2013; P. C. Merrell 2009; Rodrigues 2014; Ruales 2017).

In some categories, such as *grammar-based* and *simulation* approaches, there is a structural, and also cross-discipline, overlap among the various surveys. However, it can be observed that there are many subtopics such as constraint-based modeling, expert systems, or example-driven approaches where the lack of structural overlap

Table 3.2 – Existing surveys and reviews of generative design techniques. Key: Field: A-Architecture, CS-Computer Science, E-Engineering; Type: SA-standalone article, T-Section in a Thesis; Focus: G-General, ST-specific technique, SP-specific problem. Image credits: the author.

Title	Authors	Year	Field	Type	Focus	Cited
Techniques of automated design in architecture: A survey and evaluation	Mitchell	1975	A	SA	G	14
Case-based design: A review and analysis of building design applications	Watson et al.	1997	CS	SA	ST	114
Automated facilities layout: past, present and future	Liggett	2000	A	SA	SP	263
A case base of Case-Based Design tools for architecture	Heylighen et al.	2001	A	SA	ST	99
Revisiting the use of generative design tools in the early stages of design education	Chase	2003	A	SA	G	3
Design, innovation and case-based reasoning	Goel et al.	2005	CS	SA	ST	70
Generative mesh modeling	Havemann	2005	CS	T	G	6
A Survey of Procedural Techniques for City Generation	G. Kelly et al.	2006	CS	SA	G	136
Looking Back to the Future: An Updated Case Base of Case-Based Design Tools for Architecture	Richter	2007	A	SA	ST	26
Computer-Based Form Generation in Architectural Design — A Critical Review	Grobman et al.	2009	A	SA	G	24
The problem of space layout in architecture: A survey and reflections	Lobos et al.	2010	A	SA	SP	48
Search-Based Procedural Content Generation	Togelius, Yannakakis, et al.	2010	CS	SA	ST	179
Search-Based Procedural Content Generation: A Taxonomy and Survey	Togelius, Yannakakis, et al.	2011	CS	SA	ST	695
Computer-Based Design Synthesis Research: An Overview	Chakrabarti et al.	2011	E	SA	G	210
Towards an integrated generative design framework	Singh et al.	2012	A	SA	G	149
Procedural content generation for games: A survey	Hendrikx et al.	2013	CS	SA	G	499
Unwritten procedural modeling with the straight skeleton	T. Kelly	2013	CS	T	G	6
A review of computational intelligence in RTS games	Lara-Cabrera et al.	2013	CS	SA	G	64
Genetic Algorithm and Hybrid Genetic Algorithm for Space Allocation Problems - A Review	Sharma et al.	2014	CS	SA	SP, ST	11
A survey on procedural modelling for virtual worlds	R. M. Smelik et al.	2014	CS	SA	G	239
Automated Floor Plan Design: Generation, Simulation, and Optimization	Rodrigues	2014	A	T	SP	12
Specifications for computer-aided conceptual building design	Bernal et al.	2015	A	SA	G	65
A literature review for space planning optimization using an evolutionary algorithm approach: 1992-2014	Calixto et al.	2015	A	SA	SP, ST	22
A Panorama of Artificial and Computational Intelligence in Games	Yannakakis et al.	2015	CS	SA	G	176
Shape Grammars: A Key Generative Design Algorithm	Gu et al.	2018	A	SA	ST	4
Usability of contemporary tools for the computational design of architectural objects: Review, features evaluation and reflection	Nisztuk et al.	2018	A+CS	SA	SP	13
Computational design in architecture: Defining parametric, generative, and algorithmic design	Caetano et al.	2020	A	SA	G	28
Progress and challenges in generative product design: A review of systems	Mountstephens et al.	2020	CS	SA	G	1

shows the need for an overarching structure of the field of generative modeling techniques across the disciplines that make use of them.

Mitchell 1975 is probably the first review ever.

Watson and Perera 1997 presents an overview of Case-based design applications in building design.

Liggett 2000 focus on generative algorithms for the problem of space allocation, i.e., generating layouts of buildings and present examples that use constraint-based, genetic algorithm and simulation techniques.

The review by Chase 2003 presents the attributes of generative design concep-

tually but is relatively non-technical and lacks specifics of the reviewed techniques - pattern generation, parametric variation, and grammars.

Goel and Crow 2005 present eight factors that make design tasks challenging for Case-Based Reasoning. Still, one could argue they apply for any generative technique: breadth, stages, specification, complexity, collaboration, representation, integration, creativity.

Havemann 2005 presents the advantages of generative techniques: make complex models manageable; rely on sufficient hardware resources; are highly compact to store and transmit; allow for model-based reconstruction; make the similarity of modeling and programming explicit. He reviews several existing languages for geometric (generative) modeling and examples of grammar-based modeling.

G. Kelly and McCabe 2006 present five techniques for procedural city modeling - Grid Layout and Geometric Primitives, L-Systems, Agent-based, Template-based, Split grammars. However, they illustrate with only one example per category. The examples are evaluated on the following criteria: Realism; Scale; Variation; Input; Efficiency; Control; Real-time. They also present several procedural techniques which they state are not relevant for architecture or city generation but more for modeling natural objects: fractals, tiling, Voronoi texture cells.

Grobman, Yezioro and Capeluto 2009 present several groups of techniques with their advantages and disadvantages on Figure 3.12. Theirs is one of the rare surveys that include expert systems for architectural design.

Lobos and Donath 2010 review academic architectural works and commercial software dealing with the problem of space allocation (layout) in architecture. The review is rather non-exhaustive, and examples seem arbitrarily picked out. The authors define four categories of techniques: Expert Systems, Shape Grammar, Genetic, Constraint-Based.

The review by Togelius, Yannakakis, et al. 2010 focuses on search-based strategies of procedural content generation for games and presents mainly examples of using the evolutionary algorithm for generating content. Their review gives a helpful classification of representation types for generated content (See the following section).

Chakrabarti et al. 2011 present an overview of computer-based techniques for design synthesis in three main categories: function-based synthesis, grammar-based synthesis, and analogy-based design. Provides an overview of graph grammar modeling software - GrGen, Design Compiler 43, GraphSynth - applied to engineering problems but which might be relevant for architectural design. The focus of the presented examples is Engineering design, not architecture. However, the classification is relevant, extends the ones in other studies, and furthers the aim of this survey to source generative algorithms for architecture that were borrowed from other fields but might have been overlooked in previous surveys.

Singh and Gu 2012 state that the main incentives for adopting generative design in architecture are to use computational capabilities to support human designers and (or) automate parts of the design process to explore larger design spaces, achieve efficiency, cost reduction, optimization, accuracy, consistency. The authors distinguish five categories of generative design techniques: shape grammars, L-Systems, cellular automata, genetic algorithms, and Swarm intelligence. However, the given examples for Swarm intelligence are mostly related to work organization or the study of social insects. The authors present a table with the suitability of each technique

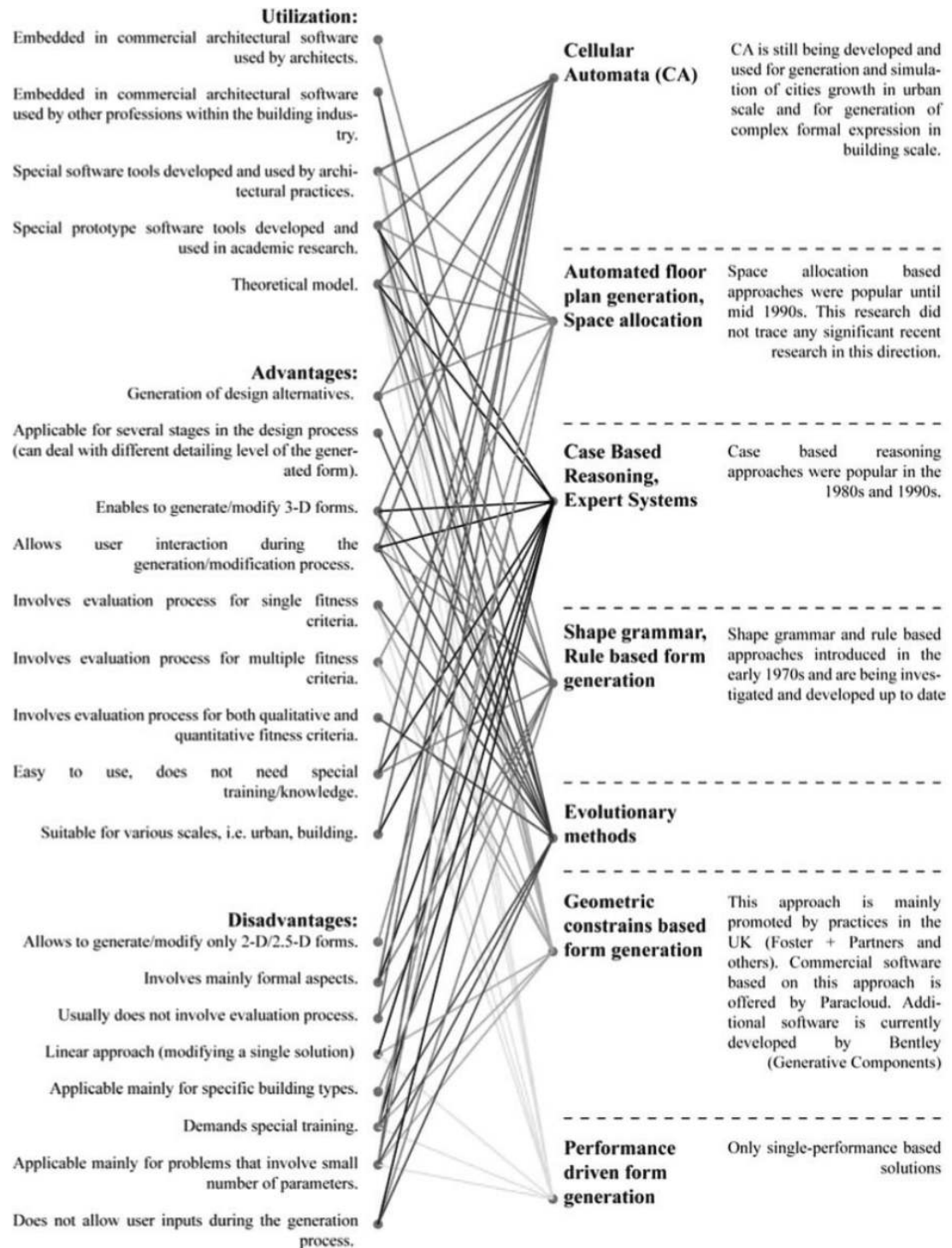


Figure 3.12 – Computer-based form generation. Main approaches according to Grobman, Yezioro and Capeluto 2009 Image credits: Grobman, Yezioro and Capeluto 2009.

to different design phases and list problems for each method, the modes for user intervention, and the challenges in the development of models using each method.

Hendrikx et al. 2013 present a *Taxonomy of common methods for generation game content* (Figure 3.13). The methods are explained in the appendix to the article. However, some groups, such as Image filtering, are not directly relevant for the generation of artifacts of the built environment as the taxonomy of Hendrikx

et al. 2013 is general, in a sense that it aims to include approaches for all kinds of game content and behavior. Furthermore, the reasons behind the particular taxonomy structure chosen by the authors are not given.

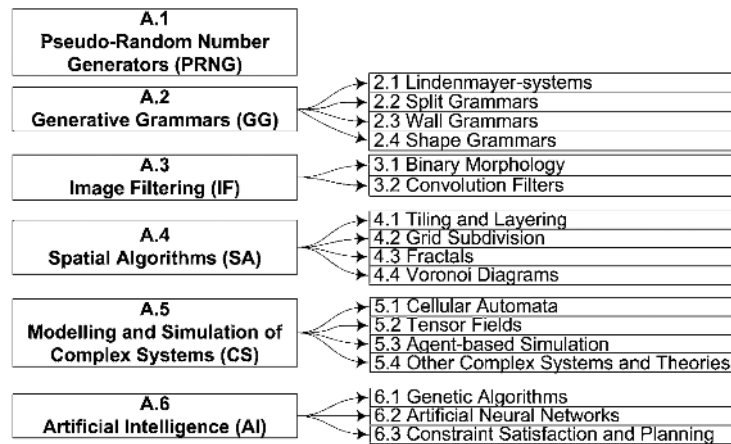


Figure 3.13 – Taxonomy of common methods for generating game content. Image credits: Hendrikx et al. 2013.

T. Kelly 2013 lays procedural modeling techniques on the spectrum from fully generic modeling (a programming language) to particular model instances (3D mesh of a bunny). The author includes variations of grammars (string, graph, shape), L-Systems, data flow programming (visual scripting, grasshopper), Simulation, combinatory modeling (model synthesis), and shape deformation.

Lara-Cabrera, Cotta and Fernandez-Leiva 2013 review computational intelligence techniques in the development of real-time strategy (RTS) games where content generation is one of many problems or tasks solved by algorithms. Their review suggests that for the task of PCG, mostly evolutionary approaches are used. In contrast, the other techniques — case-based reasoning/reinforcement learning, AI planning, influence maps, simulations, dynamic scripting, artificial neural networks, and fuzzy bayesian — are used for tasks concerning the strategy formulation and movement of opponents.

Sharma and Singhal 2014 focus on the problem of space allocation and review only genetic algorithm examples, many of which are outside the field of architecture.

R. M. Smelik et al. 2014 point out procedural modeling’s data compression and the reduced human intervention when creating content as the key factors that make it attractive. However, they find that non-technical creative professionals have not adopted procedural modeling techniques yet due to the poor controllability of most procedural models and the difficulty of predicting results from rules and input parameters. The authors classify the procedural modeling techniques for virtual worlds into six categories: stochastic, artificial intelligence, simulations, grammars, data-driven, and computational geometry. They include not only academic works but also ready-to-use software systems.

Rodrigues 2014 outlines six approaches to generating floor plans: Area assignment, Area partitioning, Space allocation, Hierarchical construction, Conceptual exploration, Design adaptation. The author presents a comparison of evolutionary approaches (Figure 3.14).

Bernal, Haymaker and Eastman 2015 give an overview of areas, including design generation, where computer-based assistance to designers makes sense and is or is

approach		eval.	geometry												topology		
authors	methods	oF	wD	eD	eW	iD	S	fL	sT	eL	eF	bB	aB	oO	sL	sA	
Jo and Gero (1996, 1998)	GA	g					•	•				•					
Schnier and Gero (1996, 1997)	GA	g															
Rosenman (1997, 2000)	GA	g					•										
Gero and Kazakov (1997, 1998)	GA	g					•	•				•					
Jagielski and Gero (1997)	GP	g					•	•				•					
Bentley (1998)	GA	g t							•			•				•	
Garza and Maher (1999, 2001, 2002)	GA	g t		•	•	•	•									•	
Elezkurtaj and Franck (1999, 2000, 2002)	GA/ES	g t					•	•								•	
Michalek (2001)	GA+SA/SQP	g t h c l					•	•				•				•	
Jackson (2002)	GP+L-system	g															
Virirakis (2003)	GP	g t		•	•	•	•				•					•	
Makris (2005); Makris et al. (2006)	GA	g t						•								•	
Bausys and Pankrasovaite (2005)	GA	g h l					•	•				•					
Homayouni (2007)	GA	g t						•								•	
Doulgerakis (2007)	GP	g t					•	•	•			•				•	
Banerjee et al. (2008)	GA	g t						•				?				•	
Serag et al. (2008)	GA	g															
Inoue and Takagi (2008, 2009)	GA+VD	g t l s						•				•				•	
Wong and Chan (2009)	GA	t														•	
Thakur et al. (2010)	GA/DA	g t s		•			•	•								•	
Verma and Thakur (2010)	GA/DA	g t s		•			•	•	•							•	
de la Barrera Poblete (2010)	GA+VD	g						•				•					
Knecht (2010, 2011)	GA/ES+K-D tree	g t						•				•				•	
Flack (2011)	GA/GP	g t					•	•	•			•				•	
Rodrigues et al. (2013a,b,c, 2014a,b)	ES+SHC	g t	•	•	•	•	•	•	•	•	•	•	•	•	•	•	

GA genetic algorithms; GP genetic programming; ES evolutionary strategy;
SA simulated annealing; SQP sequential quadratic progr.; L-System lindenmayer system;
VD voronoi diagram; DA dijkstra's algorithm; SHC stochastic hill climbing;
? undetermined; • implemented; g geometric; t topological; h heating; c cooling; l lighting; s walk distance;
oF objective function; wD wall dimension; eD exterior door; eW window; iD interior door;
S space units; fL floor levels; st stair; eL elevator; eF equipment/furniture;
bB building boundary; aB adjacent buildings; oO opening orientation; sL space location; sA space adjacency

Figure 3.14 – Comparison of evolutionary approaches to generative design. Image credits: Rodrigues 2014.

not developed (Figure 3.15).

Calixto and Celani 2015 present 31 papers using genetic algorithms to solve the space planning problem. They observed six techniques to construct the layouts: half plan, k-dimensional trees, shape grammars, blocking, one-to-one assignment, slice tree structure. As well as four methods to evaluate solutions: Quadratic assignment problem, interactive evaluation by human users, adjacency matrix, graph theory.

Yannakakis and Togelius 2015 give an overview of A.I. in games and divide the field into six key methodology areas: evolutionary computation, reinforcement learning, supervised learning, unsupervised learning, planning, and tree search. The section on Procedural Content Generation is relevant for this study and points to example works. Yannakakis and Togelius 2015 divide the techniques into *dominant* and *secondary* or *strong* and *weak*. The authors use the diagram on Figure 3.16 to denote the areas and target groups for which generative techniques are useful.

Gu and Behbahani 2018 offer a review of shape grammar applications and present a chronology of shape grammars (Figure 3.17). They classify Generative Design algorithms into three types replacement(grammars, L-Systems, parametric), evolution, and agent interaction (cellular automata, swarm intelligence, space colonization). The review distinguishes five types of grammar transformations: addition, subtraction, division or split, modification, and substitution. Addition, modification, and substitution have been explored in my research presented here. The authors divide control mechanisms into three general levels: internal (geometric, encoded in the rules), external (human or algorithm), and parallel (most common - using labels embedded in the shapes in the rules to give hints to an external agent).

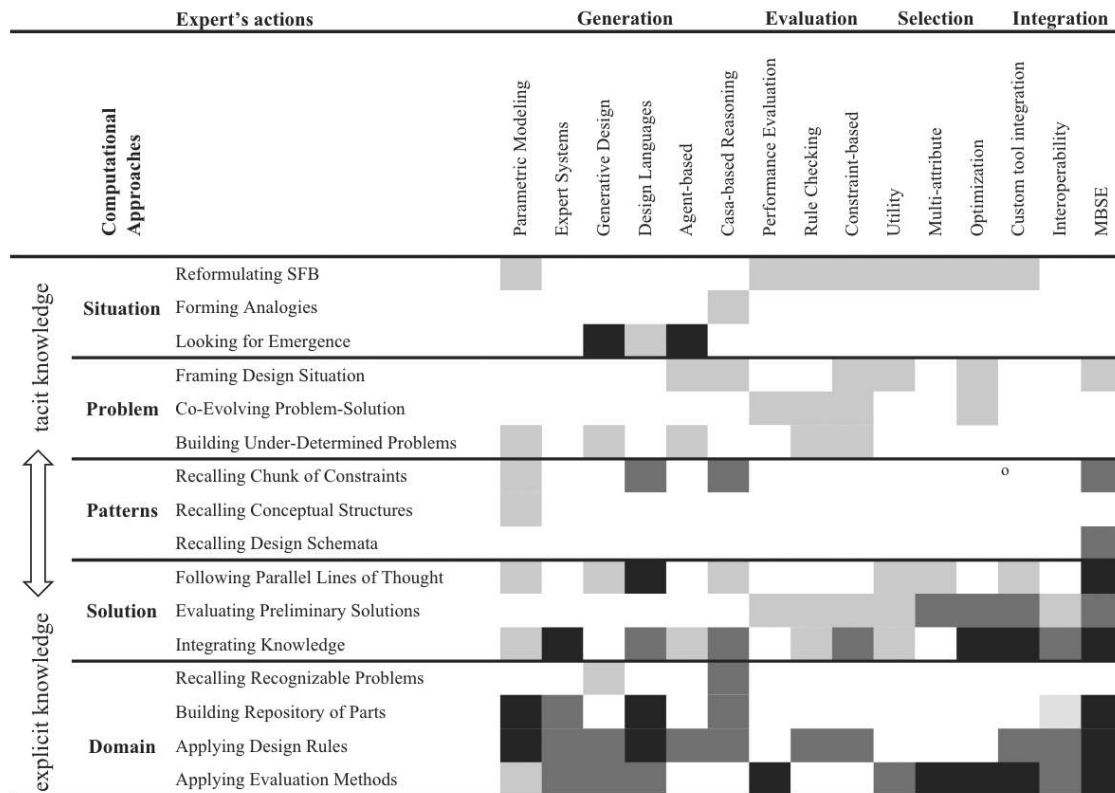


Figure 3.15 – Levels of algorithmic assistance on designer's actions. White: human-based, light grey: computer-aided, dark grey: computer-based, and black: computer-augmented. Image credits: Bernal, Haymaker and Eastman 2015.

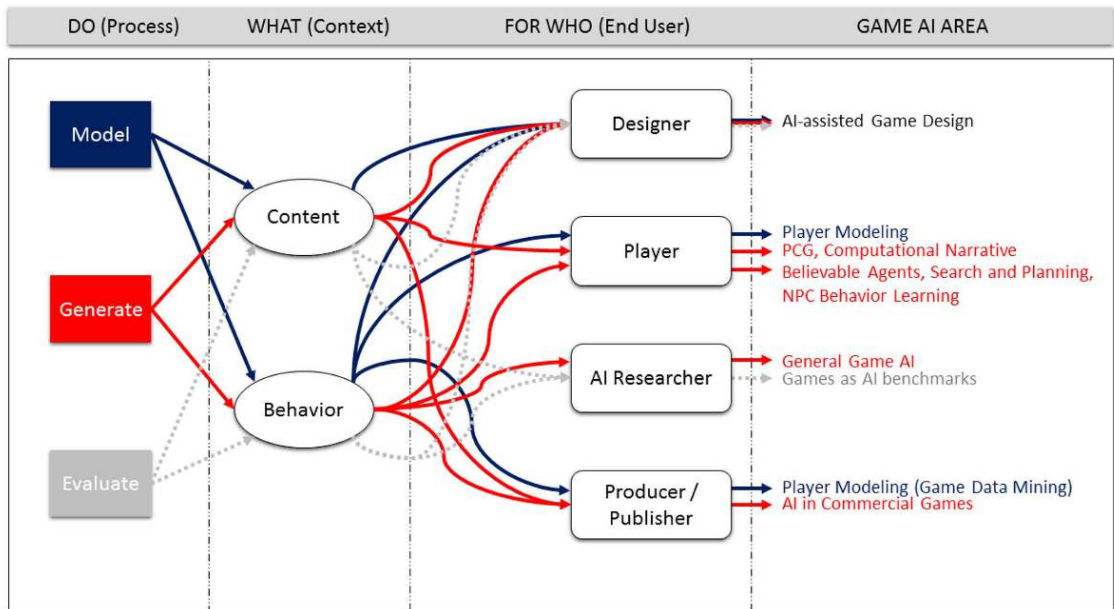


Figure 3.16 – Assistive generative techniques in games based on the tasks. The tasks are performed by designers, players, AI researchers or game producers. Image credits: Yannakakis and Togelius 2015.

The focus of the review by Nisztuk and Myszkowski 2018 is on techniques for automation and optimization of solving the floor plan problem in architecture. The study emphasizes the usability of the reviewed techniques in practice. It also includes

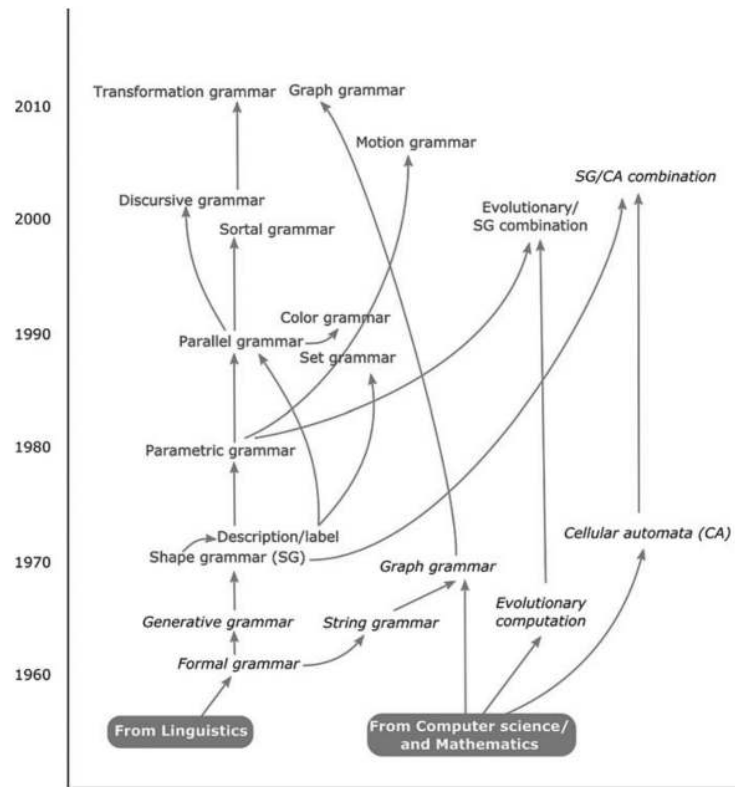


Figure 3.17 – The evolution of Shape Grammars. Image credits: .
Gu and Behbahani 2018

a survey conducted by the authors with respondents being practicing architects aiming to establish the degree to which computational design tools are used in practice. Lists chronologically 18 research prototypes and five commercial software which are not categorized or classified.

Caetano, Santos and A. Leitão 2020 define the terms Computational, Generative, Parametric, and Algorithmic Design basing the definitions on an extensive literature review.

Mountstephens and Teo 2020 review generative design systems “according to their primary goal, generative method, the design phase they focus on, whether the generation is automatic or interactive, the number of design options they generate, and the types of design requirements involved in the generation process”. They list: Shape Grammars, L-systems, Swarm Intelligence, Genetic Algorithms, Parametric Modeling and Topology Optimization.

The review papers by Dutta and Sarthak 2011; Heylighen and H. Neuckermans 2001; Hsu and R. J. Krawczyk 2003; Meller and Gau 1996; Pérez-Gosende, Mula and Díaz-Madroño 2021; Tyflopoulos et al. 2018; Watson and Perera 1997, as well as the review sections by Chau et al. 2004; Hoisl and Shea 2011; P. C. Merrell 2009; Ruales 2017, have not been summarized here as they do not add new information about types of generative techniques. However, the works they presented have been studied and, if relevant, included in the review.

3.2.4 Methodology

I formed a search phrase for each category of generative design techniques introduced in the review papers above. After a search in literature databases, the category was included in the taxonomy if relevant works were found.

The method follows the guidelines by Levy and Ellis 2006 and is similar to the method used by Larsen et al. 2019 for their review of mass-customization in the housing industry.

1. Search phrases for each technique were formed based on the terms used in the review papers for each technique and the terms used in the titles and abstract of the cited works in the review papers. The phrases are listed in Table 3.3. For example, for shape grammars the phrase is:

```
technique_qualifier_phrase = (shape-grammar* OR discursive-grammar*)
```

2. To narrow down papers to architecturally relevant ones a search phrase with qualifying terms was added:

```
object_qualifier_phrase = (facade* OR layout* OR "space plan*" OR  
floor$plan* OR city OR cities OR virtual-world* OR urban* OR house$  
OR housing OR villa* OR game$ OR building*)
```

3. A search phrase that qualifies the results to the subject of generative design was also compiled. The subject of generative design is concerned with one or more of the concepts of *generation* and *automation*, it is *computer-based*, *computational* or *procedural* and often uses *algorithms*. Therefore the *subject qualifier phrase* is:

```
subject_qualifier_phrase = (procedural* OR comput* OR generat* OR  
automat* OR algorithm*)
```

4. The final search phrase for each technique is:

```
technique_qualifier_phrase AND object_qualifier_phrase  
AND subject_qualifier_phrase
```

5. For literature databases that support filtering by category, such as the Web of Science, the categories were used to narrow the search.
6. The following databases were searched - Web of Science, CumInCAD, Google Scholar, and Microsoft Academic. The software *Publish or Perish*¹ was used to manage and export searches in Google Scholar and Microsoft Academic.
7. All searches were combined into one list as some papers showed up in more than one category.
8. Each publication was checked, and irrelevant ones were pruned.
9. I did a backward and forward search by collecting the papers that cited the relevant works and the papers that they cited. Those two new lists were then checked for relevant publications and added to the list of works.

¹<https://harzing.com/resources/publish-or-perish>

Table 3.3 – Search phrases for the identified generative techniques. Image credits: the author.

#	CATEGORY	SEARCH PHRASE	RECORDS FOUND (WOS)
	Generative Design	(procedural* AND (model* OR generat*) NOT ("procedural justice" OR "procedural language" OR "procedural sedation" OR "procedural fairness")) OR ("generative design*" OR "generating design*" OR "generative model*" OR "generating model*" OR (generat* NEAR/1 algorithm\$) OR (computer-generated) OR (computational-design) OR (computerized space planning))	746
A.	Constructive	-	-
A.1.	Computational Geometry	computational-geometr*	31
A.1.1.	Fractals	fractal*	78
A.1.2.	Subdivision Algorithms	subdivision OR treemap\$ OR voronoi OR halfplane* OR half-plane*	140
A.1.2.1.	Voronoi	-	-
A.1.2.2.	Slicing Tree	slic* NEAR/1 tree\$	2
A.1.2.3.	K-D Trees	k-d-tree\$	2
A.1.3.	Parametric Modeling	-	-
A.1.4.	Generative Modeling Languages	"generative modeling language" OR "generative ML" OR Hyperfun OR GENMOD OR PLaSM	1
A.1.5.	Marching Cubes	-	-
A.2.	Grammar-based	grammar*	256
A.2.1.	Symbolic Grammars	-	-
A.2.1.1.	Formal String Grammars	string-grammar* OR formal-grammar*	5
A.2.1.2.	L-Systems	Lindenmayer-system* OR L-System*	10
A.2.1.2.1.	Open L-Systems	-	-
A.2.1.3.	Graph Grammars	graph-grammar*	19
A.2.1.3.1.	Boundary Solid Grammars	solid-grammar*	0
A.2.2.	Shape Grammars	shape-grammar* OR discursive-grammar*	68
A.2.2.2.	Set Grammars	-	-
A.2.2.2.1.	Split Grammars	split-grammar* OR split-shape-grammar* OR CGA	18
A.3.	Constraint-based	Constraint-Satisfaction OR Constraint-based OR Constraint-solving	122

Continued on next page

Table 3.3 (continued)

#	CATEGORY	SEARCH PHRASE	RECORDS FOUND (WOS)
A.3.1.	Wave Function Collapse	wave-function-collapse	1
A.4.	Simulation Approaches	(growth NEAR/2 simulat*) OR (comput* AND morpho\$gen*) OR (behavio\$ral-model* AND design)	93
A.4.1.	Cellular Automata	"cellular automata"	431
A.4.2.	Agent-based	agent-based OR multi-agent OR swarm-intelligence OR agent-model\$ing OR self-organization	674
A.4.3.	Physically-based Mod- els	physically-based	48
A.4.4.	Monte Carlo Methods	markov-chain-monte-carlo OR MCMC	37
A.4.5.	Topology Optimization	topolog* NEAR/2 optimiz*	67
A.5.	Example-driven Algo- rithms	((example-driven OR example-based OR model-based OR component-based OR example-guided OR data-based OR data-driven) NEAR/10 (synthes* OR generat*)) OR model-synthesis	116
A.5.1.	Model Synthesis	model-synthesis	-
A.5.2.	Generative Machine Learning	(neural-network\$ OR artificial-intelligence OR deep-learning OR machine-learning OR bayesian OR (Probabilistic NEAR/2 Model*)) AND generat*	853
A.5.2.1.	GAN	Generative-Adversarial-Network\$ OR GANs OR GAN	59
A.5.2.2.	Bayesian Networks	-	-
A.6.	Analogy-based Design	analogy-based	3
A.6.1.	Case-based Design	case-based OR template-based OR pattern-based	190
A.7.	Reinforcement Learning	reinforcement-learning	216
B.	Generate-and-test	-	-
B.1.	Search-based	search-based OR Metaheuristic\$	227
B.1.1.	Evolutionary Algo- rithms	("genetic algorithm*" OR "genetic programming") OR (evolut* NEAR/2 (strateg* OR algorithm* OR technique\$ OR programming OR design))	1943
B.1.2.	Simulated Annealing	simulated-annealing	191
B.1.3.	Particle Swarm Opti- mization	particle-swarm-optimization OR PSO	366

Criteria for inclusion

This research focuses on low-rise, high-density residential buildings. Therefore to include a work in the techniques overview:

1. It must be generative, meaning that not humans but algorithms create the design or parts of the design.
2. must be a computer implementation, not merely a concept or a manually used system.
3. It must create a representation (plan, facade, or 3D model) of a building, a building fragment (window, wall), or a city for use in architecture or computer graphics (computer games). Products (including furniture), underground infrastructure (pipelines, sewers, etc.), generic shapes, computer games characters, text, textures, sound effects, music, terrain, landscapes, plants, and other natural structures were ignored. Pure street layouts without buildings were not included.
4. Reality reconstruction techniques, such as facade reconstruction from images or 3D scans, were not included.
5. Content generated for use in Games was also included if it was architectural, i.e., level designs or layouts or 3D models of buildings, dungeons, or cities. Platformer levels like the levels for super Mario bros were not included.
6. Finally, to be included, a work must be serious, i.e., respond to a requirement specifications for the generated buildings or cities.

3.2.5 Taxonomy introduction

The *Taxonomy of Generative Design in Architecture* is presented on Figure 3.8.

Taxonomies describe relationships between items, while classifications group the items according to one or two attributes (Lalonde 2021). As seen in the overview of existing surveys above, many classifications have been put forth which have informed this taxonomy. Among them, there are multi-dimensional, or poly-hierarchical taxonomies - example by Togelius, Yannakakis, et al. 2011 which has the axes of: (i) Online vs. Offline; (ii) Necessary vs. Optional Content; (iii) Random Seeds vs. Parameter Vectors; (iv) Stochastic vs. Deterministic Generation; (v) Constructive vs. Generate-and-Test. My research's focus gives the criteria to structure the taxonomy and map the relationships between the techniques. My work aims to explore ways to bring a user in the process, i.e., human-in-the-loop. Therefore it is essential to group techniques based on the type of input needed during the development phase, at the start of a generative loop, during the loop, and in the post-processing phase. All these properties are determined by the specific algorithms used in each technique. Therefore the taxonomy presented here is organized by the algorithm used to generate the results.

The potential for integrating participation from human co-designers, trained professionals, and non-experts for each class of techniques will be discussed later.

The ranks typically used in a taxonomy are shown on Figure 3.18. Table 3.4 gives an overview of how I map these ranks in the Taxonomy of Generative Design techniques presented here.

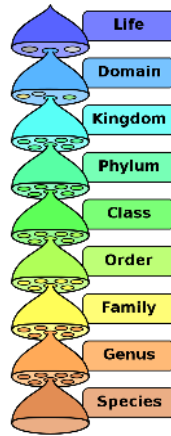


Figure 3.18 – The taxonomy ranks. Source Wikipedia https://commons.wikimedia.org/wiki/File:Biological_classification_L_Pengo_vflip.svg, Pengo, Public domain, via Wikimedia Commons. Image credits: Wikimedia.

Table 3.4 – Taxonomy ranks with examples. Image credits: the author.

Latin	English	What
regio	domain	The domain of generative design
regnum	kingdom	One kingdom for Constructive techniques and one for Generate-and-Test techniques.
phylum	division	A division is formed from classes of algorithms that consist of similar steps, for example Grammar-based deviation encompasses all algorithms that are rewriting systems.
classis	class	A Class is defined based on the type of data the algorithms run on. For example Shape grammars introduced by Stiny and Gips 1971 and Symbolic Grammars are both rewriting algorithms but the former runs on shapes while the latter on symbolic representations (strings, graphs).
ordo	order	An Order is the particular technique for example Set Grammars. Sometimes a suborder is also used as in the case of Split Grammars.
familia	family	Family is the specific combination of algorithms - for example Parametric models in combination with the Multi-objective Genetic Algorithm.
genus	genus	A Genus is the specific family as described in a given paper by its authors such as the Parametric models plus Multi-objective Genetic Algorithm implementation in (Gerber and S.-H. E. Lin 2014).
species	species	Species encompass all designs that can be produced with the same input parameters from a given algorithmic setup (Genus). (not tracked here)
singulis	individual	Individuals are the particular instance of a design produced with given input parameters. (not tracked here)

The *Constructive* and the *Generate-and-test* Kingdoms

Togelius, Yannakakis, et al. 2010 present a distinction of procedural content generation algorithms that is relevant to this work: Constructive vs. Generate-and-test.

A *Constructive* algorithm generates content once and makes sure it matches the

requirements. The challenge in this type of techniques is making sure that the content is correct or at least “good enough” as it is being constructed (Togelius, Yannakakis, et al. 2010). Examples from this kingdom are shape grammars (Stiny and Gips 1971) and simulation approaches such as cellular automata and topology optimization.

A *Generate-and-Test* algorithm creates designs in a loop. It generates a candidate design and tests it against defined criteria or lets the user evaluate it, determining whether to keep the candidate or discard it partially or entirely. Examples are all evolutionary algorithm approaches (Holland 1975) (Figure 3.19) and in general all search-based approaches (Togelius, Yannakakis, et al. 2010).

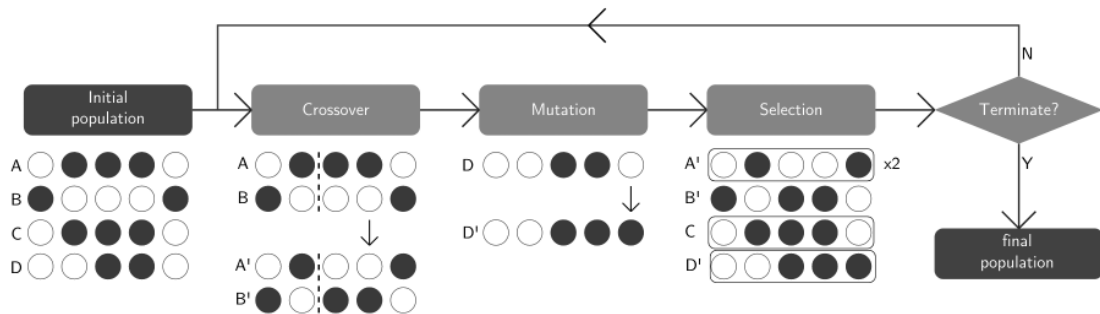


Figure 3.19 – The steps in an evolutionary algorithm. Image credits: Harding 2014.

It is of importance to note that almost all *Generate-and-Test* approaches contain at least one *Constructive* approach in their *generating phase*. This meta-level of guiding a constructive algorithm, probably the reason for term *meta-heuristics* denoting the techniques in this kingdom, presents the main advantages and the main challenges of the kingdom. The challenge of this kingdom of techniques is making sure the evaluation and selection process allows the exploration of the entire design space and is transparent for the designer.

A.1. Computational geometry

The *Computational geometry* division includes all techniques that involve sequential geometric operations and transformations such as extrusion, trimming, or construction of curves based on mathematically defined functions (de Berg et al. 1997).

The speed, the minimal required input, and the fact that they are often deterministic (de Berg et al. 1997). This makes them suitable for combining with meta-heuristic techniques from the generate-and-test kingdom.

An essential feature of the techniques in this group is, even for noise algorithms, that they are the opposite of heuristic, i.e., analytic in nature (de Berg et al. 1997). That is not to say they are deterministic and not stochastic since they can be both - noise is stochastic (relies on random numbers), Voronoi is deterministic (the same set of points produces the same Voronoi tessellation). But both are analytic, i.e., the noise algorithm doesn’t make empiric guesses but proceed to the next steps based on a clearly defined analytical function.

Noise algorithms (Perlin 1985), and in general Pseudo-Random Number Generation (PRNG), are probably the most well-known representative of this division and the earliest form of generative design techniques (Hendrikx et al. 2013). However, I

have not included them in the taxonomy as they have little relevance to modeling artifacts of the built environment and are more suited for modeling naturally occurring objects and phenomena such as textures, terrain, and smoke (G. Kelly and McCabe 2006).

Fractals, the other very popular mathematical generative technique, were introduced by Benoît Mandelbrot, regarded as the 'father of fractals'. Mandelbrot coined the term *fractal* in 1975 (Mandelbrot 1977, 1982) from the Latin *fractus* meaning broken (G. Kelly and McCabe 2006). Fractals are also not optimal for modeling artifacts of the built environment as there is rarely such a strong self-similarity in it. Hence they are superseded by the grammar-based approaches (G. Kelly and McCabe 2006). For example, the *Koch snowflake* fractal originally described by Koch 1904 with a mathematical equation can be represented as an L-System (Rani, Haq and Sulaiman 2011). Unlike *noise* however, I have decided to include them as there are examples of using fractals for generating buildings (Figure 3.20) (Ediz and Çağdaş 2007).

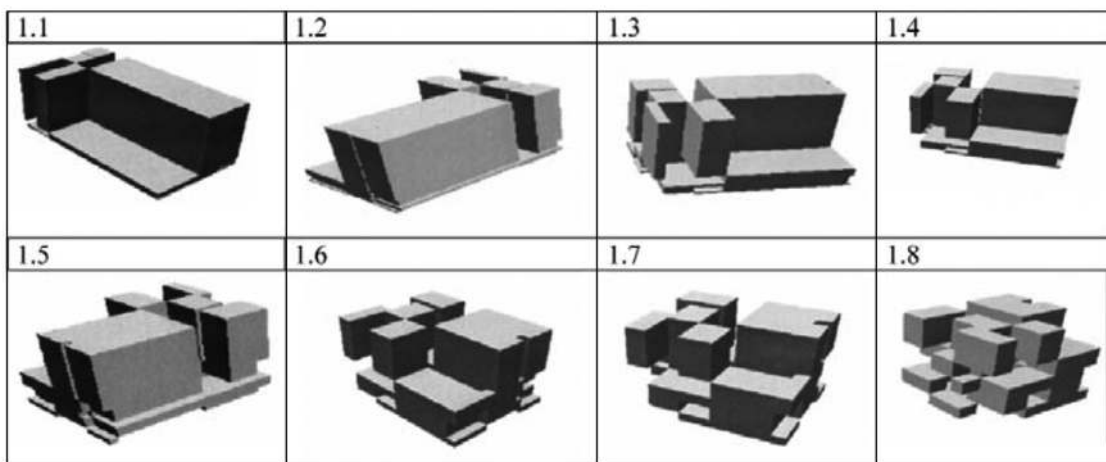


Figure 3.20 – Fractal-based form generation. Image credits: Ediz and Çağdaş 2007.

Subdivision algorithms such as *slice tree structure*, *Voronoi*, and other space-filling methods also belong to computational geometry division (Knecht and Koenig 2010; R. M. Smelik et al. 2014). The Voronoi tessellation introduced by Voronoi 1908 is probably the most well-known and visually present subdivision algorithm, as it became the symbol of the parametric and generative architectural design in the 2000s and onwards. A detailed illustrated description of its workings is given in (Aurenhammer 1991). Examples of the use of Voronoi tessellation for generating architectural design can be found in (Menges 2012) (Figure 3.23) as well as in (Coates, Derix, et al. 2005; Harding and Derix 2011; Koltsova et al. 2011). In the category of subdivision algorithms, (Otten 1982) introduced slicing trees and (Kado 1995) extended them (Figure 3.21). (Knecht and Koenig 2010) introduced k-d trees (Figure 3.22). And (Damski and Gero 1997) introduce the use of half-planes to generate floor plan layouts.

G. Kelly and McCabe 2006 present an example of *Grid Layout and Geometric Primitives* in the case of the work by (Greuter, Parker, et al. 2003; Greuter, Stewart and Leach 2004; Greuter, Stewart, Parker, et al. 2003) on computer-generated infinite cities, which considering its generation pipeline, also belongs to computational geometry (Figure 3.24).

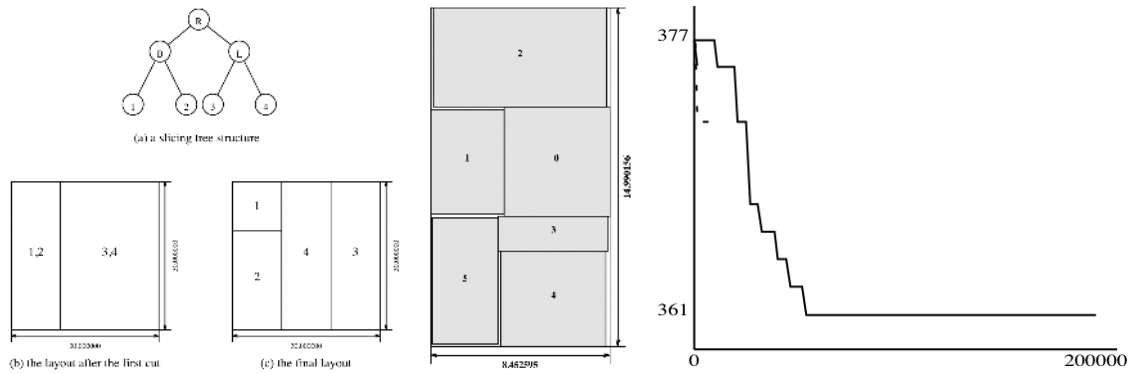


Figure 3.21 – Slice Tree Structure. The logic of Slice Tree Structure (left) as explained by Kado 1995. The best layout (middle) and the convergence of the Genetic Algorithm (right) for a layout problem with 6 facilities (Kado 1995). The horizontal axis indicates the number of evaluations, the vertical axis indicates the mean of the best individual scores. Smaller is better. Image credits: Kado 1995.

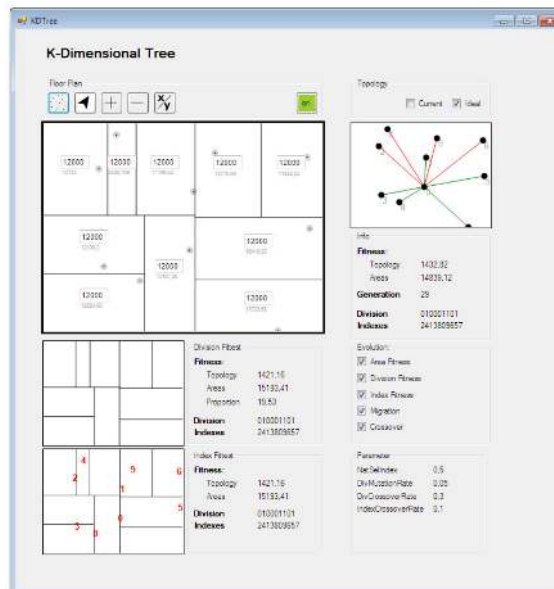


Figure 3.22 – K-d trees used to subdivide layouts. Image credits: Knecht and Koenig 2010.

Parametric modeling, or data-flow programming, is also here (T. Kelly 2013). Parametric models or associative models are topologically set up, and the links between the model's components are made dependent on parameters. Parametric modeling is very broad in relation to generative but offer an overlap (Figure 3.9) (Caetano, Santos and A. Leitão 2020).

Havemann 2005; T. Kelly 2013 are the only ones among the reviews to include general-purpose programming languages as a type of generative modeling technique. The authors name various *Generative modeling languages* such as Open Inventor, Generative Modeling language (Havemann 2005) (Figure 3.25), PLaSM (Paoluzzi, Pascucci and Vicentino 1995), HyperFun or GENMOD (Snyder and Kajiya 1992). These languages use mathematical approaches via parametric functions, and as such, the technique fits under computational geometry. They are suitable for automating the generation of specific designs and not for generic exploration of design spaces. They require an expert to model the design and then translate it to code.


```

1 Gothic-Window.Styles.Style-1 begin
2
3 { usereg !edgeBack !edgeWall
4   !wallSetback !bdOuter !poly
5
6   /stdGrey setcurrentmaterial
7   :poly 5 poly2doubleface dup edgemate
8   :edgeWall killFmakeRH
9   :bdOuter 4 div :wallSetback neg 5 vector3
10  extrude
11 } /style-main-arch exch def
12
13 { usereg !edgeBack !edgeWall !poly
14   /stdGreen setcurrentmaterial
15   :poly 5 poly2doubleface !edge
16   /gold setcurrentmaterial
17   :edge edgemate :edgeWall killFmakeRH
18   [ :edge ] [ (0.05,-0.3,1) ]
19   extrudestable pop
20 } /style-fillet exch def
21
22 { usereg !rad !mid !edgeBack !edgeWall !poly
23   /stdRed setcurrentmaterial
24   :poly 5 poly2doubleface !edge
25   :edge edgemate :edgeWall killFmakeRH
26   /gold setcurrentmaterial
27   :edge (0.05,-0.3,5) extrude !edge
28 } /style-rosette exch def
29
30 { usereg !bh !arcl !edgeBack !edgeWall !poly
31   /stdBlue setcurrentmaterial
32   :poly 5 poly2doubleface !edge
33   :edge edgemate :edgeWall killFmakeRH
34   /gold setcurrentmaterial
35   :edge (0.05,-0.3,5) extrude !edge
36 } /style-sub-arch exch def
37
38 end
    
```

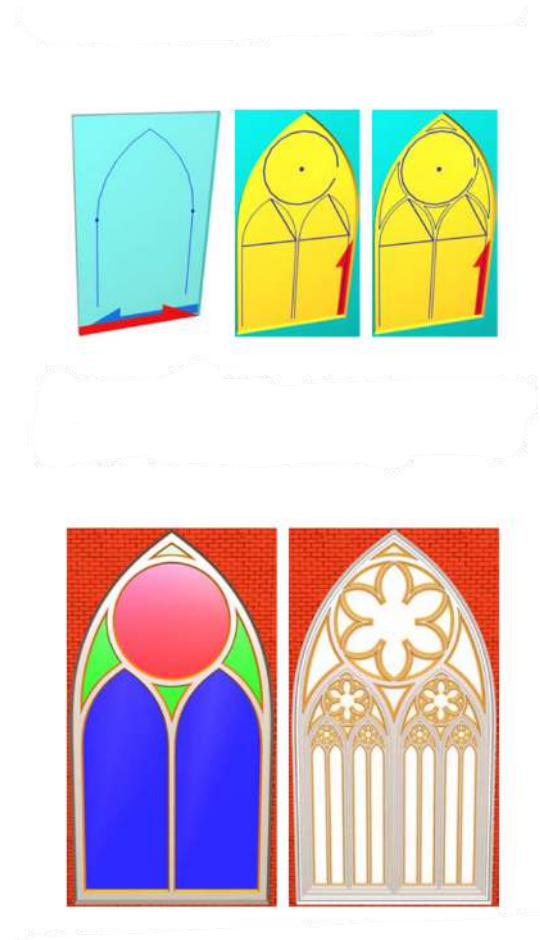


Figure 3.25 – Generative Modeling Language. A gothic window created using the Generative Modeling Language by Havemann 2005. Left, the code for style 1, top right steps, bottom right style 1 and style 8 of the windows generated in the work. Image credits: Havemann 2005.

et al. 2011; Krishnamurti 1993). It is easier to switch a sequential grammar into a parallel as it requires changing the mode of computation. At the same time, it is far more difficult to switch a grammar from symbolic into shape as it requires a complete redefinition of the grammar. Therefore I have chosen to use the *symbolic* vs. *shape* distinction in the taxonomy presented here. *Symbolic grammars* require an extra post-processing step to produce design geometry as in the work with graph grammars by J. H. Lee, Ostwald and Gu 2018 (Figure 3.26), while *shape grammars* operate on geometry directly, such as the shape grammar developed for the houses designed by the architect Alvaro Siza at Malagueira by Duarte 2001 (Figure 3.27).

Ilčík and Wimmer 2016 present an approach called *symbolic shape grammar* which might appear to challenge the distinction into symbolic grammars and shape grammars. However, the work of Ilčík and Wimmer 2016 in its essence is an extension of CGA-Shape by Müller, Wonka, et al. 2006 or a parametric shape grammar modeled with a generative modeling language instead of geometry.

L-Systems belong to the symbolic, string-based grammars (Alfonseca and Ortega 1996; T. Kelly 2013; Parish and Müller 2001; Singh and Gu 2012). In 1996, Měch and Prusinkiewicz 1996 presented the different types of L-Systems up until that moment and introduced the *Open L-Systems* which has seen probably the most notable use of L-Systems for generation of cities and buildings in the work of Parish and Müller

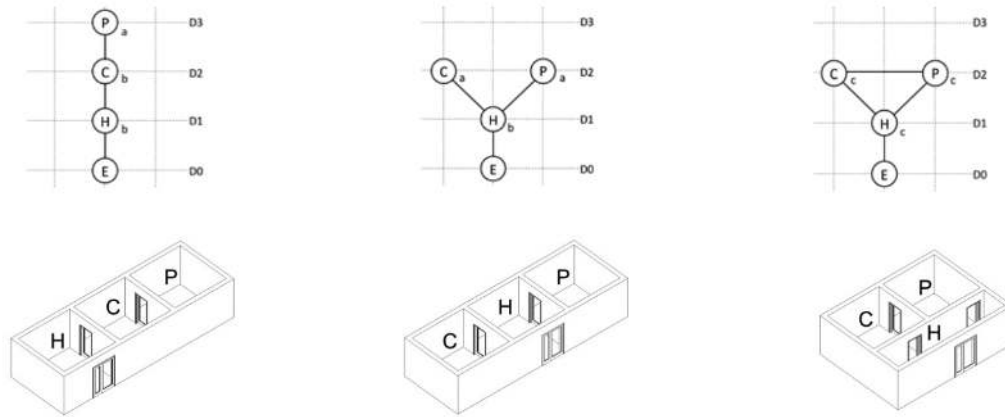


Figure 3.26 – Justified plan graphs. Based on Space Syntax (Hillier 2007). Spatial interpretations of justified plan graphs in the work by J. H. Lee, Ostwald and Gu 2018 on using graph grammars to generate justified plan graphs for houses. Image credits: J. H. Lee, Ostwald and Gu 2018.

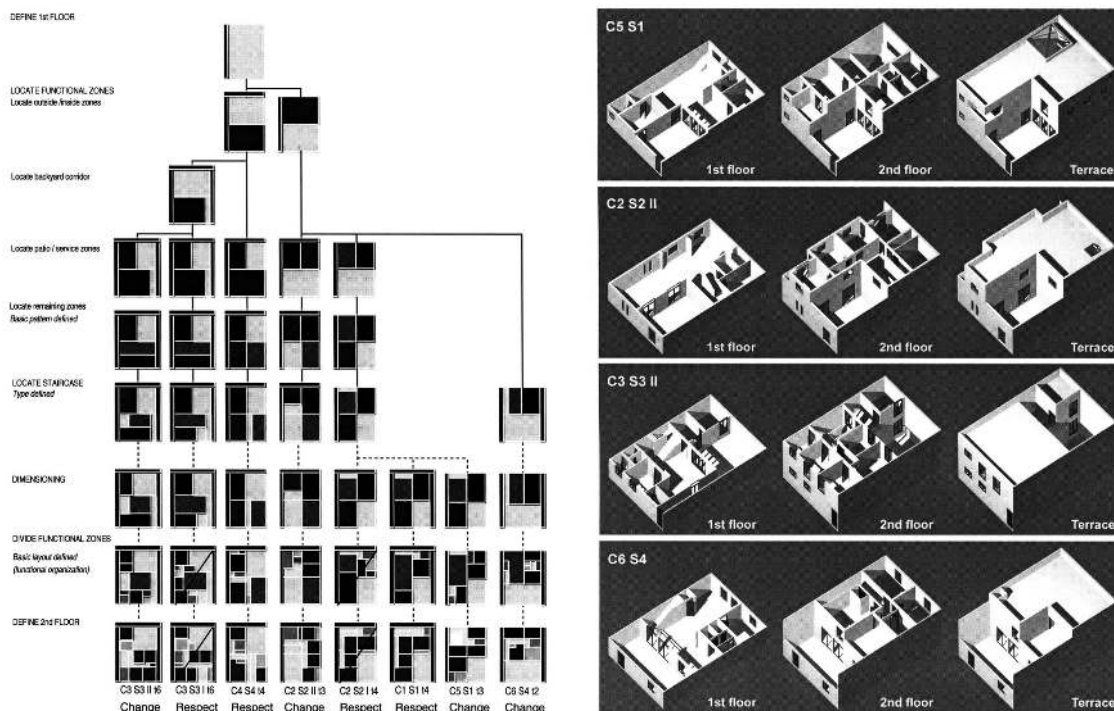


Figure 3.27 – Discursive Shape Grammars. Left, Partial tree diagram and right, four generated designs, showing the derivation of the house designs produced with the discursive shape grammars by Duarte 2001. Image credits: Duarte 2001.

2001. L-Systems are less suitable for modeling buildings than other grammar-based approaches because “a building is not designed with a growth-like process in mind, but a sequence of partitioning steps” (Wonka et al. 2003).

Graph grammars, introduced by Nagl 1976; Pfaltz and Rosenfeld 1969 belong to symbolic (string) grammars according to Chakrabarti et al. 2011; Gu and Behbahani 2018; Pavlidis 1972 and to spatial grammars according to Krishnamurti 1993. However, the categorization of Krishnamurti 1993 is confusing as it lists string, set, and graph grammars under spatial grammars and discusses shape grammars separately with the distinction that “unlike the other spatial grammars, shape grammars oper-

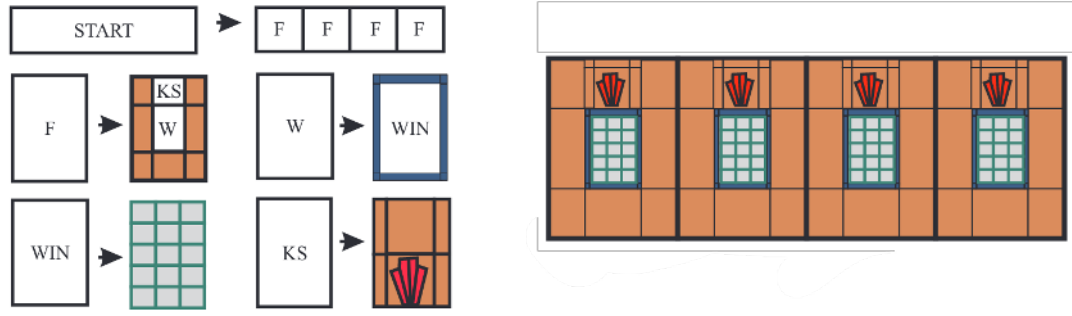


Figure 3.28 – Split Grammars. The rules for a simple split grammar (left) and its result (right). Image credits: Wonka et al. 2003.

ate directly on spatial forms”. This confirms the categorization of all other authors that graph grammars are symbolic, which is where I have categorized as well.

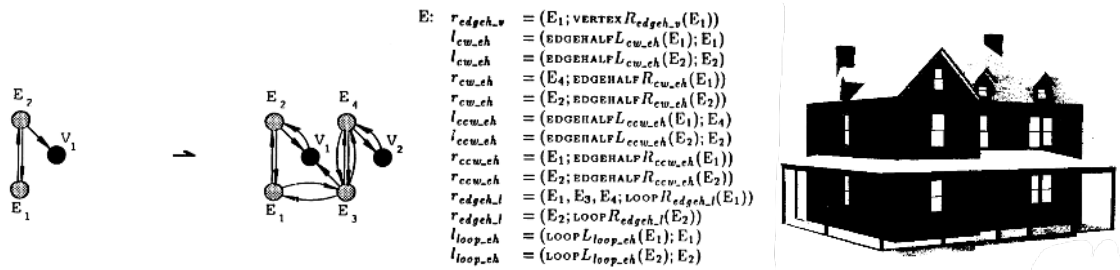


Figure 3.29 – Boundary Solid Grammars. A production grammar rule on a solid body representation as a graph (left) (J. A. Heisserman 1990) and a design of a Queen Anne house in the language of the Boundary Solid Grammar by J. Heisserman 1994. Image credits: J. Heisserman 1994; J. A. Heisserman 1990.

Krishnamurti 1993 place the *boundary solid grammars* introduced by J. Heisserman 1994; J. Heisserman, Mattikalli and Callahan 2004; J. Heisserman and Woodbury 1993 (Figure 3.29) under graph grammars.

Shape grammars introduced by Stiny and Gips 1971 are considered the most important algorithmic approach to design (Duarte 2001; R. M. Smelik et al. 2014). Gu and Behbahani 2018 state that grammars, in particular shape grammars, are very suitable for encoding design knowledge. Original shape grammars such as the Palladian grammar (Stiny and Mitchell 1978) were challenging to implement in a computer program due to the high complexity of interpreting the left-hand shape (LHS) of the grammar rules. This, in turn, is caused by the fact that production rules produced new sub-shapes unknown beforehand (J. P. McCormack and Cagan 2002, 2006; Stiny 1982). Set grammars, introduced by Stiny 1982, are a variation of shape grammars where the vocabulary is finite and predefined, as are the spatial relations between the shapes of the vocabulary which form the basis for the grammar rules (Stiny 1980a, 1982). By having only rules that keep vocabulary shapes intact and limiting the combinations between them, set grammars reduce the computational complexity of the problem of interpreting left-hand shape rules (Gips 1975; Wonka et al. 2003). Each design in the language defined by a set grammar can be taken apart into a set of parts that belong to the vocabulary set, while not the same is true for a design produced with a shape grammar (Stiny 1982). Wonka et al. 2003 introduced *Split Grammars* for large urban environments, and facade generation in particular (Figure 3.28), focusing on fast computation and no dead-ends

during the derivation (T. Kelly 2013). So far, they are the most practically relevant implementation of shape grammars as shown by the success of the *CityEngine* software which uses them, and the abundance of citations and publications that refer to split grammars. Knight 2003 introduced parallel shape grammars. Stiny 1980b introduced parametric shape grammars. Duarte 2001 proposes a four-step process for developing a shape grammar based on an existing corpus. It includes *rule inference*, *prototyping*, *reverse grammar*, and *testing*. Wonka et al. 2003 state that Shape grammars “are not suitable for automatic and semi-automatic modeling”. Chau et al. 2004 give an overview of the recent applications of shape grammars (Figure 3.30)

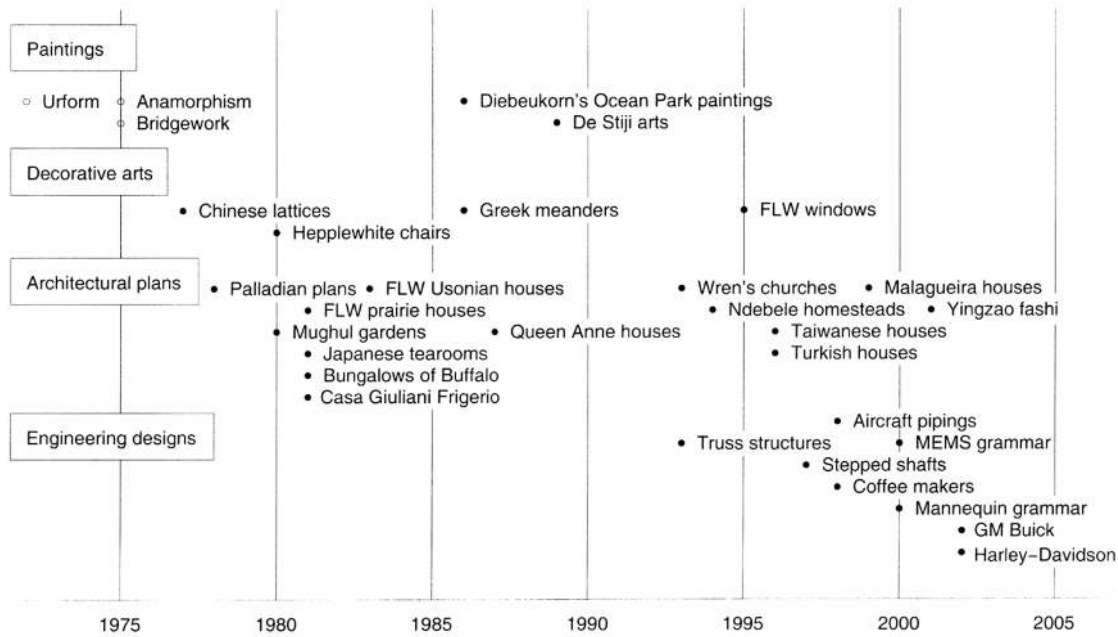


Figure 3.30 – Applications of shape grammars. Image credits: Chau et al. 2004.

Hoisl and Shea 2011 present review of interpreters for shape grammars. Furthermore, they present an interpreter for set grammars that is a step toward providing a general shape grammar interpreter within a familiar CAD environment (Chakrabarti et al. 2011). Chakrabarti et al. 2011 state the following issues to all generative grammars (they refer to a workshop presentation that was the basis for (McKay et al. 2012)): “(1) supporting designers to articulate grammars (i.e., vocabulary and rules) in software implementations, (2) defining ways to evaluate implementations, including identifying a set of benchmark problems, (3) better integration of generative grammar implementations with other software, e.g., CAD and CAE and (4) more methodological support for users in the process of defining a grammar since this process can also lead to better understanding of a design problem.”

A big problem with shape grammars is the shape detection, i.e., finding a LHS shape to trigger a rewriting rule (Chakrabarti et al. 2011; Jowers et al. 2010). Advanced algorithms have been applied to the problem (Chakrabarti et al. 2011; Chau et al. 2004; Hoisl and Shea 2011; J. P. McCormack and Cagan 2002, 2006). The voxel implementation of shape grammars presented in the *20.000 BLOCKS* case study in this work reduces the complexity through discretization (Savov and Tessmann 2017).

Another problem is the rule creation. Most examples use expert knowledge encoding, i.e., the author of the grammar models the rules. However, the interface

to do that is usually not suitable for any user as it requires programming or using custom-made environments. (Yogev, A. A. Shapiro and Antonsson 2010) for learning grammars rules by evolutionary optimization, but maybe not truly grammars while (Orsborn, Cagan and Boatwright 2008) present a method to learn the grammar for a car automatically from a corpus of vehicle examples.

Garcia 2017 classifies grammars according to design strategy into *grid*, *subdivision* and *additive* of which in the case of the voxel shape grammars adopted here both grid and additive have been considered. *Subdivision* was not considered due to the discrete quality of the voxel space.

A.3. Constraint-based modeling

Constraint-based approaches is another kind of constructive generative modeling technique (Charman 1993; Liggett 2000; Lobos and Donath 2010; R. M. Smelik et al. 2014). Constraint-based modeling techniques represent the generative task as a Constraint Satisfaction Problem (CSP) which is defined by a set of variables belonging to a finite domain of values and a set of constraints on these variables (Apt 2003; Mackworth 1977). A solution to CSP is an instantiation of the variables that satisfies all constraints (Charman 1993). According to Lobos and Donath 2010, p.146, the Constraint-Based Approach is similar to the architectural practice as “*design* is a combinatorial problem in principle, i.e., a constraint-based search for an overall optimal solution of a design problem”. CSP is mainly used to solve the Facility Layout Problem, i.e., generate floor plans.

A recent, emerging example of the use of Constraint-based modeling that opens up the technique to 3D application is the Wave Function collapse algorithm developed by Gumin 2016 (Karth and A. M. Smith 2017). Tigas and Hosmer 2021 show its application in combination with Reinforcement learning for the generation of building designs (Figure 3.31).

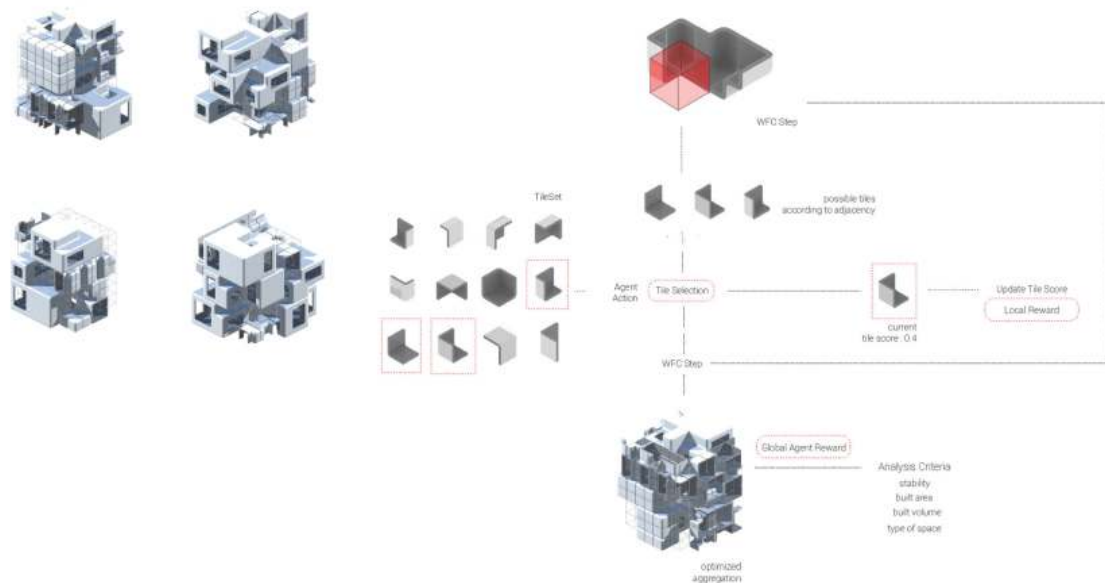


Figure 3.31 – Wave Function Collapse. Spatial Assembly using the Wave Function Collapse algorithm and Reinforcement learning. Image credits: Tigas and Hosmer 2021.

A.4. Simulation approaches

Simulation approaches are subset of the *constructive* approaches as the simulation algorithm *constructs* one design artifact iteratively. "Simulation approaches to procedural modeling involve the imitation of observed processes to approximate the interactions within a model, and between the model and its environment." (T. Kelly 2013). I include growth simulations, morphogenetic algorithms, and agent-based approaches in this category. This category covers phenomena like self-organization and self-assembly. L-Systems can also be considered growth simulation algorithms, but they are in the grammar-based approaches with all other rewriting systems due to their operation on strings.

T. Kelly 2013 places *Cellular Automata* under simulation approaches. Wolfram 1983 refers to (Schrandt and S. M. Ulam 1967; S. Ulam 1962; von Neumann 1951, 1966) to state that "cellular automata were originally introduced by von Neumann and Ulam (under the name of *cellular spaces*)". Cellular Automata are mostly used for urban planning and simulation of city growth (Batty 2005). An example of 1-D Cellular Automata used for generating architectural form is the competition entry for the San Jose State University Museum of Art and Design (Figure 3.32) by Mike Silver Architects (Silver 2006). Further examples of the use of CA in architecture can be found in (Coates, Healy, et al. 1996; Herr and Kvan 2005, 2007; R. Krawczyk 2002a,b).

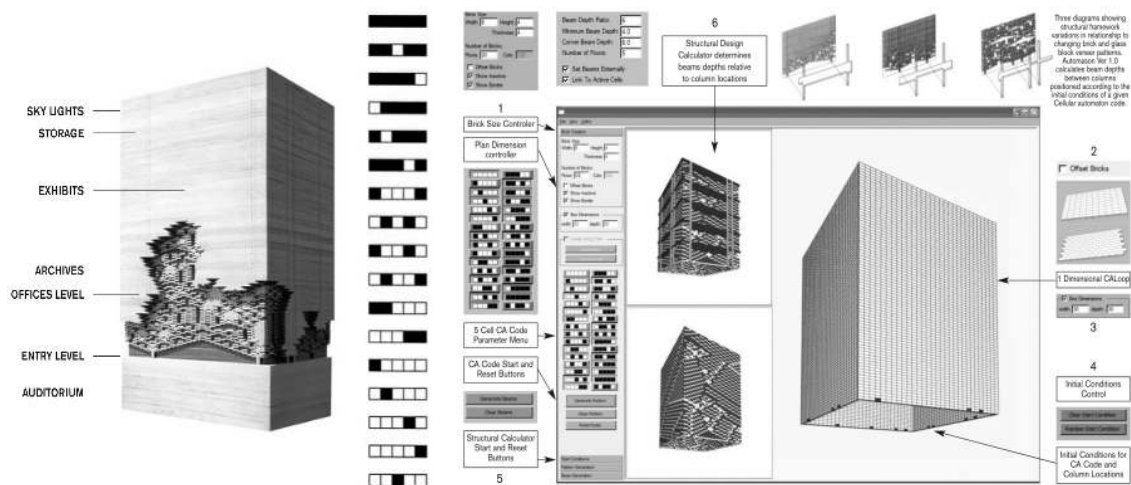


Figure 3.32 – Generating architectural form with 1-D Cellular Automata. Competition entry for the San Jose State University Museum of Art and Design by Mike Silver Architects. Left: result, Right: Interface of the Automason 1.0 software developed by Yee Peng Chia and Eric Maslowski used for the design by Mike Silver Architects. Image credits: Silver 2006.

R. M. Smelik et al. 2014 place *agent-based* modeling under simulation approaches. Swarm Intelligence, is used interchangeably with agent-based models by Singh and Gu 2012 and exhibits properties of self-organization (Macy and Willer 2002; Reynolds 1987). An example is the procedural city modeling by Lechner et al. 2003.

T. Kelly 2013 places physical simulations also under simulation approaches. *Physically-based* simulations use spring-based or force-based models to model building components and their relationships. Their aim is to find a stable layout state automatically, such as in the work by Arvin and House 2002 (Figure 3.33) and Michalek and Papalambros 2002.

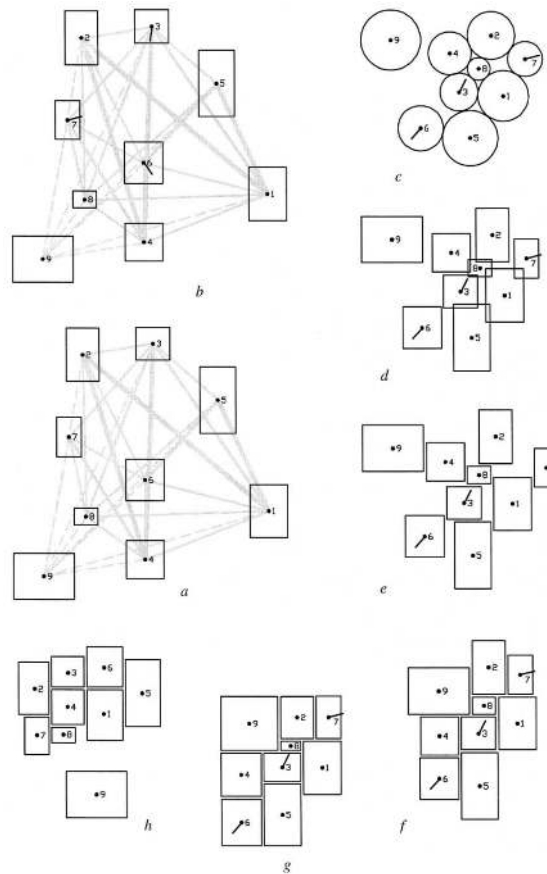


Figure 3.33 – Physically-based simulation. Sample results from the physically-based space planning work by Arvin and House 2002 Image credits: Arvin and House 2002.



Figure 3.34 – Floor plan optimization using the Metropolis algorithm. Metropolis is a Markov Chain Monte Carlo type of algorithm used in the work of P. Merrell, Schkufza and Koltun 2010. Image credits: P. Merrell, Schkufza and Koltun 2010.

T. Kelly 2013; Lara-Cabrera, Cotta and Fernandez-Leiva 2013 place *Monte Carlo methods* such as *Markov Chain Monte Carlo* also under simulations. It was used in (P. Merrell, Schkufza and Koltun 2010) to solve the floor plan layout (Figure 3.34).

Topology optimization saw a surge of attention and research with the advances in Additive Manufacturing (Tyflopoulos et al. 2018). As a generative technique, it is more relevant for mechanical engineering and not that much favored for architectural design because the same inputs produce the same results, so anyone theoretically can create forms with it, removing the aspect of individual, unique authorship that is highly valued among architects (Dapogny et al. 2017). The work of Arata Isozaki with engineer Matsuro Sasaki on the Florence Train station and Qatar Education City Convention Center (Figure 3.35) is the most notable example of using topology

optimization for architectural design (Sasaki, Itō and Isozaki 2007). Topology optimization as an engineering design technique might lack the needed creative space. Still, when combined with the concept of discrete design or digital materials, it leaves enough room for interpretation of the purely static model into a composition of design modules (Rossi and Tessmann 2017a). This makes the topology optimization technique suitable to combine with discrete generative design techniques such as set grammars or wave function collapse.



Figure 3.35 – Topology optimization. Architectural design using topology optimization as generative design technique in the work of Arata Isozaki with engineer Matsuho Sasaki. Left: Florence Train station competition entry. Right: Qatar Education City Convention Center. Image credits: Sasaki, Itō and Isozaki 2007; Dapogny et al. 2017.



Figure 3.36 – Design interpretations of topologically optimized voxel fields. Using the discrete modeling approach by Rossi and Tessmann 2017a, the same voxel field can have different design expressions. Image credits: Rossi and Tessmann 2017a.

A.5. Example-driven

Another category of *constructive* techniques is *example-driven algorithms* or data-driven algorithms. Here notable classes of techniques are the *model synthesis* and supervised machine learning techniques such as GAN. T. Kelly 2013 calls this category *combinatory modeling* referring to the property of this technique to combine samples extracted from an example model into new, similar models. However, the term might be confusing in regards to the term *combinatorial design* which is a different technique, or not even a technique but a concept.

Even though *Inverse procedural modeling* is a set of approaches that use examples to inform a generative algorithm (T. Kelly 2013; R. M. Smelik et al. 2014), it is not included in this taxonomy. *Inverse procedural modeling* typically uses one of the constructive techniques such as formal grammars and extracts its vocabulary and

rules out of facade images as in the work by Ripperda and Brenner 2009. Another example of *data-driven approaches* as R. M. Smelik et al. 2014 call this approach is the work by (Müller, Zeng, et al. 2007), also on deriving shape grammar generative model from facade images. Therefore, I consider *inverse procedural modeling* not as a standalone generative technique but as a strategy for informing generative algorithms.

This taxonomy aims to organize the variety of techniques based on the algorithms used to generate a design. How a given algorithm is implemented or how the necessary input data is sourced is not of concern. In the taxonomy presented here, under *example-driven* will be included only techniques where the data step is an integral part of the generative algorithm and not merely a configuration or fine-tuning step.

There are genuinely *example-driven* or *data-driven* techniques that rely on extracting information from single examples such as the work on model synthesis by P. Merrell 2007 or from datasets such as all supervised machine learning examples like the Graph Neural Network (GNN) approach used for Graph2Plan (Figure 3.38) by Hu et al. 2020.

Model synthesis stems from example-based texture synthesis (Wei et al. 2009) and has been developed by P. Merrell 2007; P. Merrell and Manocha 2008; P. C. Merrell 2009 (Figure 3.37).

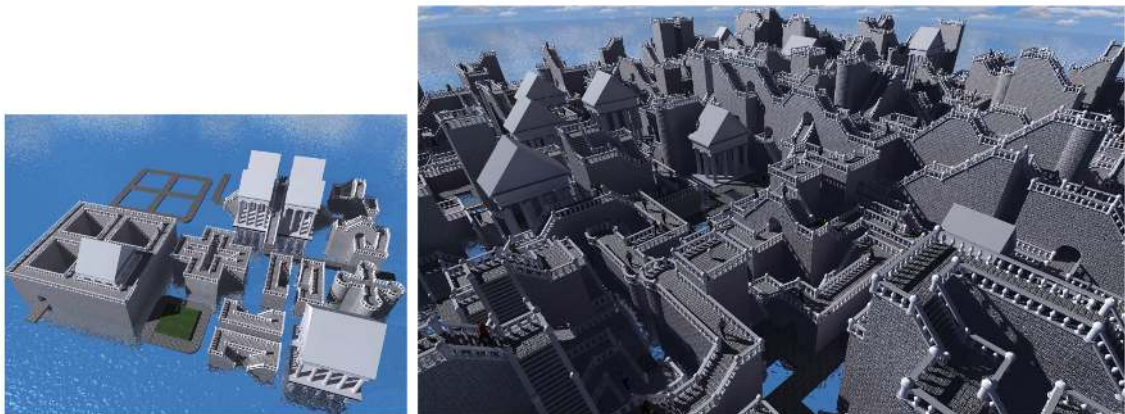


Figure 3.37 – Model Synthesis. From the example model (left), a larger model (right) is automatically created using model synthesis. Image credits: P. Merrell 2007..

The Wave Function Collapse (WFC) algorithm developed by Gumin 2016 can also run as an example-based technique when one passes a seed texture to it (Karth and A. M. Smith 2017). However, this automatic generation of the tiles needed for the WFC algorithm is another case of inverse procedural modeling, and as such, WFC is included under Constraint-based approaches.

A.6. Case-based Design

Case-based design, sometimes referred to as case-based reasoning or experts systems, is another example of generative design techniques (Chakrabarti et al. 2011; Grobman 2008; Lobos and Donath 2010). *Case-based design* is “the process of creating a new design solution by adapting and/or combining previous design solutions”

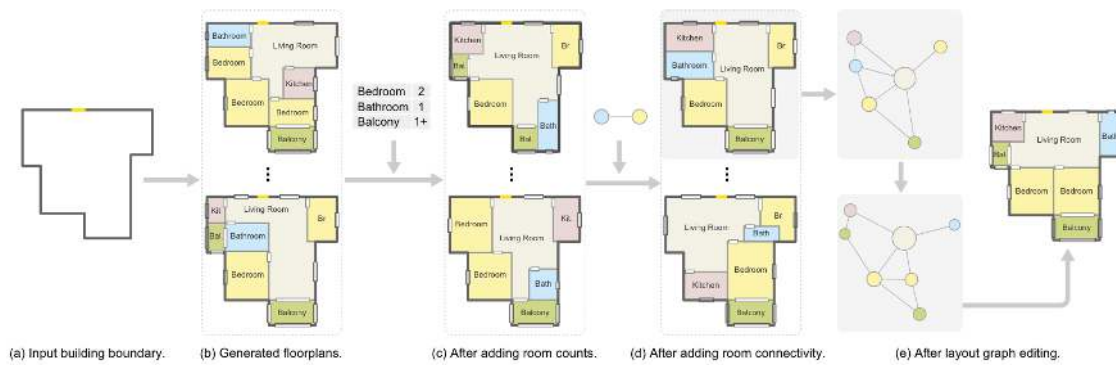


Figure 3.38 – Data-driven Generative Machine Learning. The graph neural network presented by Hu et al. 2020 can generate floor plans based on a building boundary (a-b), as well as allow users to specify room counts (c), room connectivity (d), and other layout graph edits (e). Image credits: Hu et al. 2020.

(Watson and Perera 1997). Case-based design is based on the work of case-based reasoning pioneered by Kolodner 1992 which in turn is inspired by the work by Schank on Dynamic Memory Theory (Schank and Abelson 1988) in the 1980s (Chakrabarti et al. 2011; Heylighen and H. Neuckermans 2003). Case adaptation and case combination are often performed using constraint solving techniques (A.3.) (Faltings 1996; I. Smith, Stalker and Lottaz 1996).

Chakrabarti et al. 2011 include case-based design in the larger group of *Analogy-based design*. Analogy-based design differs from example-driven approaches. The mapping of relations between objects characterizes the former while the latter is concerned with extracting attributes of objects (Gentner 1983).

Case-based design, and analogy-based design in general, is closest to the inspiration-based and experience-based approach that architects and designers traditionally use in their work, combining general and instance knowledge (Chakrabarti et al. 2011; Cross 1982; Richter 2007). This led to a lot of development of case-based design tools in the 1990s (Grobman 2008; Heylighen and H. Neuckermans 2003).

The two reviews of case-based design in architecture, by Heylighen and H. Neuckermans 2001 and Richter 2007, state that there is a noticeable shift from aiming to automate design to aiming at developing assistive systems for architects.

IDIOM (I. Smith, Lottaz and Faltings 1995) on Figure 3.39 is CBD similar to Homegrown (Green 2020).

Examples of computer implementation of generative case-based design are GPRS (R. Oxman 1990)(Figure 3.40), CADRE (Dave et al. 1994; Hua, Fairings and I. Smith 1996) (Figure 3.41) which represents cases as AutoCAD models and uses Constraint satisfaction techniques to do the case adaptation (Faltings 1996), FABEL (Börner et al. 1996), IDIOM (I. Smith, Lottaz and Faltings 1995; I. Smith, Stalker and Lottaz 1996) (Figure 3.39 and Figure 3.42), SEED (Flemming 1994, 1999; Flemming and Aygen 2001; Flemming and Chien 1995; Flemming and Woodbury 1995), SL_CB, which is based on SEED (J.-H. Lee 2002), DYNAMO (H. M. W. Neuckermans 2007) where users contributed actively and also through their interactions to the help the system get better. Further examples are found in (Afacan and Demirkan 2011; Carrara, Kalay and Novembri 1994; Kalay and Carrara 1996; Ketteler and Lenart 1992).

I explored the idea to represent cases as individual rooms instead of complete

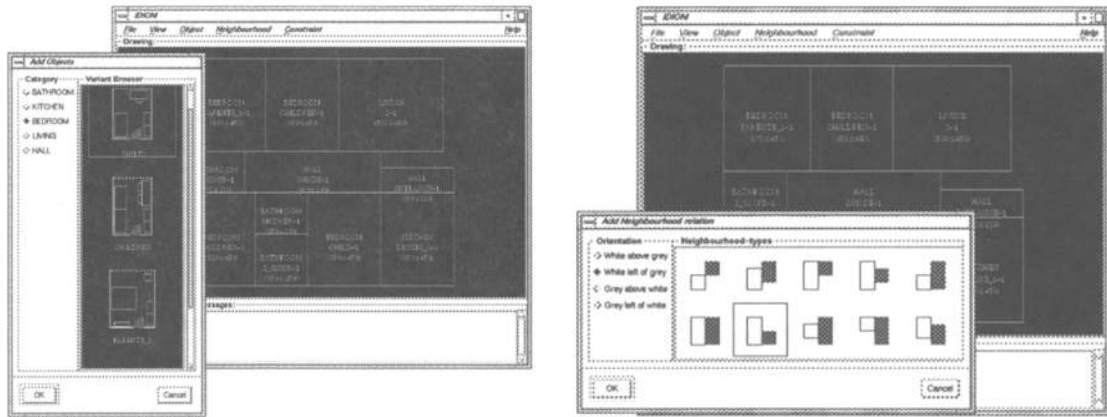


Figure 3.39 – Case-based Design (CBD), IDIOM. IDIOM, short for *Interactive Design using Intelligent Objects and Models*, is a CBD system for composing building layouts (I. Smith, Lottaz and Faltings 1995). Left: The object browser and the main design window of IDIOM. Right: Definition of neighborhood relationships. Image credits: I. Smith, Lottaz and Faltings 1995.

designs, as seen in IDIOM, in the case study Combinatorial Design using *20.000 BLOCKS* (subsection 8.4.4). And also the case-based player mission mode in *Sensitive Assembly* (chapter 7).

A recent application of case-based design ideas, very similar to those in CADRE, IDIOM, and FABEL’s SYN module, albeit not explicitly stated by the author, is Homegrown (Green 2020) for selecting and placing the top matching apartment layout from a library into a given apartment boundary (Figure 3.43).

Despite all promising qualities of the case-based approach to generative design and of the good adoption of case-based reasoning in other fields, no convincing breakthroughs in applying case-based design in architecture have been made since the 1990s (Grobman 2008; Heylighen and H. Neuckermans 2001).

Probably the biggest obstacle for making case-based design tools work in practice is building a large enough case base, following the representation and semantic structure chosen by the developers (Heylighen and H. Neuckermans 2001; Richter 2007). Heylighen and H. Neuckermans 2001 already hint at crowdsourcing case collection and processing and point out its educational benefits for the tool users such as architecture students. Chakrabarti et al. 2011 name two further challenges, automating the retrieval of fitting cases, which has seen promising progress, and automatic adaptation of a selected case or cases to the current problem where the high context-specificity of design problems proves challenging.

However, reinforcement learning (RL) might breathe new life in this field, as the combination of case-based design and RL techniques helps overcome their weaknesses Lara-Cabrera, Cotta and Fernandez-Leiva 2013. The combination achieves learning through trial and error (RL) and past experience (Case-based design). As such, it reduces the dimensionality curse of using RL in complex environments and the knowledge acquisition problem of CBD (Wender and Watson 2014).

B.1. Search-based approaches

Search-based approaches, also called metaheuristics, are inspired by evolutionary processes or simulation algorithms e.g., simple stochastic local search, simulated annealing, and particle swarm optimization (Singh and Gu 2012; Togelius, Yannakakis,

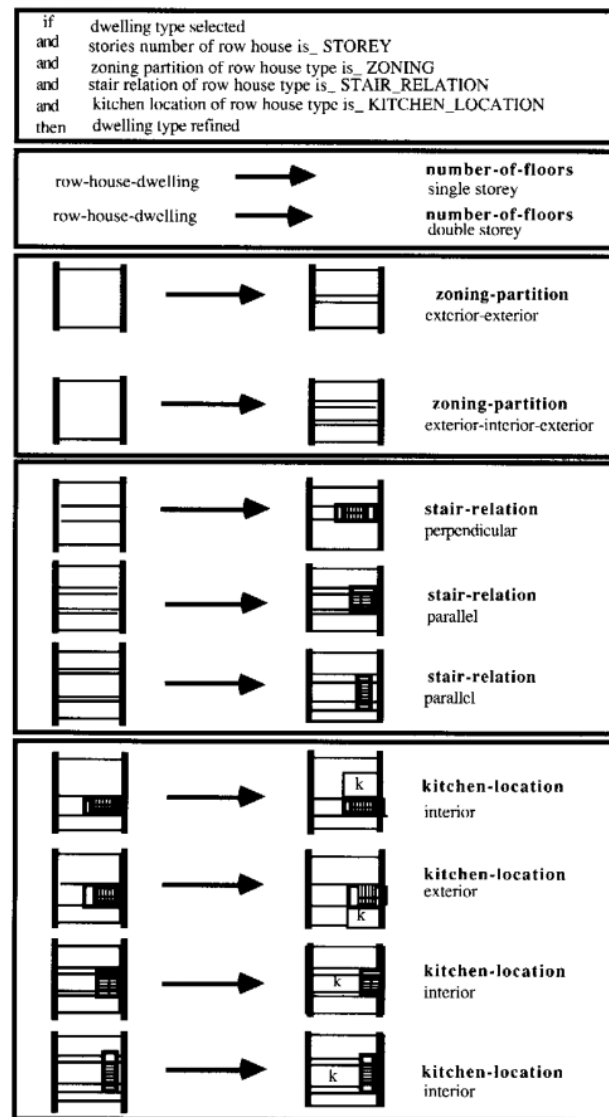


Figure 3.40 – Case-based design, GPRS. A configurator-like, case-based design tool called GPRS (A Generative Prototype Refinement Shell) developed by R. Oxman 1990. Image credits: R. Oxman 1990.

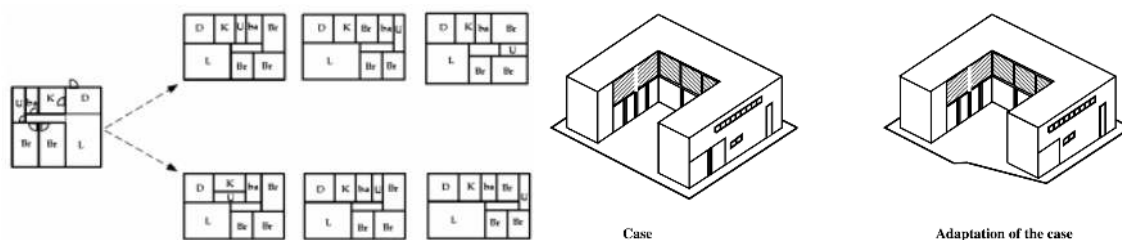


Figure 3.41 – An example of case adaptation in CADRE. Left, floor plan retrieval and adaptation (Dave et al. 1994), right adaptation of retrieved 3D model (Hua and Faltings 1993). Image credits: Dave et al. 1994, Hua and Faltings 1993.

et al. 2011). Figure 3.44 created by (Dréo and Candan 2011) presents an overview of the techniques in the field.

It is essential to clarify what is meant by *search-based* approaches here. Namely, the search over the space of all possible design candidates that could be better or worse solutions to the problem. In search-based approaches, the generated design

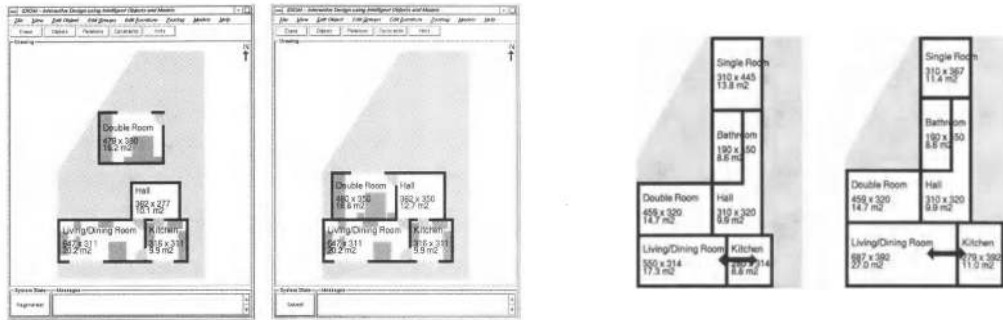


Figure 3.42 – Case adaptation in IDIOM. In IDIOM cases are individual rooms from architectural projects selected by architects that user can combine (left two images) and interactively adapt (right two images) with the help of relevant constraints. Image credits: I. Smith, Stalker and Lottaz 1996.

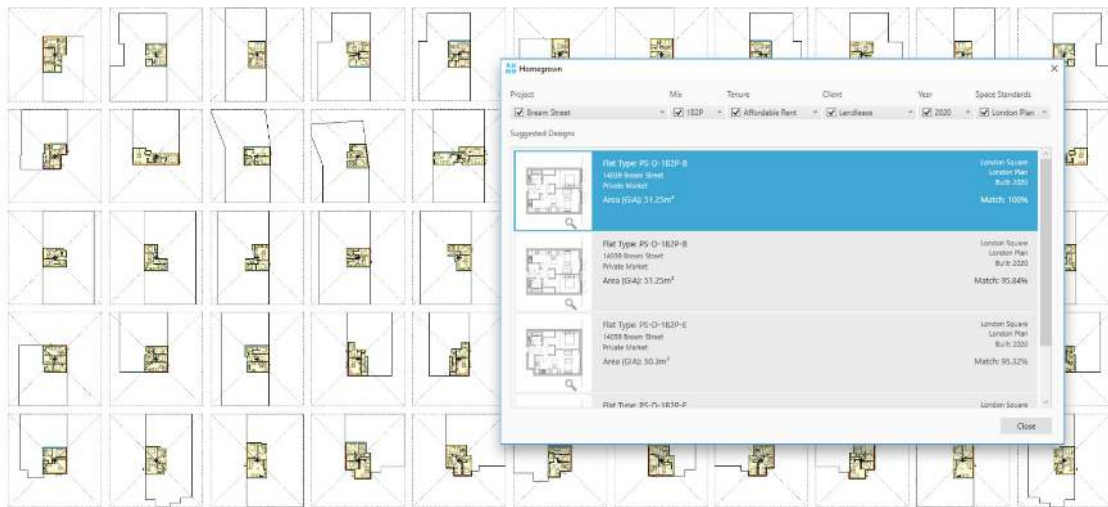


Figure 3.43 – Contemporary case-based adaptation in practice. *Homegrown* is an inhouse tool developed for architectural office *Allford Hall Monaghan Morris* by Green 2020. It fills apartment outlines with the closest matching apartment layout from a library. Image credits: Green 2020.

candidates are graded on one or more real numbers, using a function that assigns fitness (Togelius, Yannakakis, et al. 2010). The function does not simply reject or accept the candidate design. It grades it with real numbers, which allows for new design candidates to be generated that have better value than the previously evaluated ones (Togelius, Yannakakis, et al. 2011).

A different meaning of searching for solving a problem is given by Russell and Norvig 1995. Russell and Norvig 1995 define "solving problems by searching" as having a problem-solving agent that decides on the next action to take to arrive at a desirable state. This concept of search can be applied to many of the iterative approaches in the Constructive kingdom (A.), such as Constraint Satisfaction, Grammars, Topology optimization, Reinforcement Learning, or Agent-based methods, where at each step of the generative process of one candidate design, the algorithm must decide what its following action is. This *search for the best next action* does not apply to the group of search-based techniques presented here. Instead, the meaning of *search among possible design candidates* is used.

It is important to note that the search-based generative techniques are not generative on their own, but they aim to automate or assist search that otherwise a designer will perform. Search-based techniques have a constructive component and

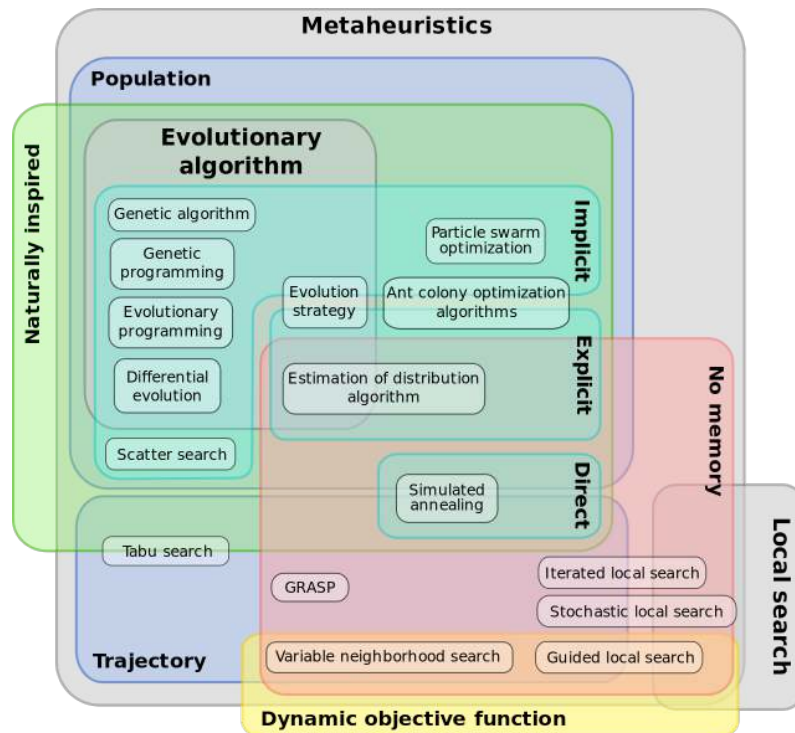


Figure 3.44 – Search-based approaches. Classification of the different metaheuristic techniques shown as an Euler Diagram by Dréo and Candan 2011. Image credits: Dréo and Candan 2011.

an evaluative component. In the case of evolutionary algorithms, the constructive component is the procedure for mapping a genome (encoded design) to a phenotype (design instance to be evaluated). The evaluative component is the fitness function.

Genetic Algorithms are meta-heuristics inspired by the evolutionary processes and introduced by Holland 1975 (Grobman 2008; Mountstephens and Teo 2020; Singh and Gu 2012). According to Menges 2012 “evolutionary computation is understood as a relatively inefficient optimization process, but is beginning to be recognized as being very effective in deriving unconsidered or hitherto unknown possibilities while maintaining an overall performance level”. Under evolutionary algorithms, I have included genetic algorithms, genetic programming, evolutionary programming, evolutionary strategies, and others (Figure 3.45). Even though there is a distinction among them (Yu and Gen 2010), in the sampled works, some authors have misqualified the particular technique they are using (Calixto and Celani 2015).

Note on Artificial Intelligence

Artificial intelligence is a very broad term - it includes evolutionary algorithms (Yu and Gen 2010), Reinforcement learning (Akizuki et al. 2020), GAN (Z. Wang, She and T. E. Ward 2019), GNN (Hu et al. 2020). As each of these techniques has a different approach to the actual genesis of the designs, it creates they would be categorized under different categories. Evolutionary algorithms are included under search-based approaches, data-driven techniques such as GAN and GNN under the example-based algorithms, and Reinforcement learning under case-based reasoning (Lara-Cabrera, Cotta and Fernandez-Leiva 2013). However, the general terms *artificial intelligence*, *deep learning*, and so on are included in the search phrases for sourcing work examples since many authors use those in the titles instead of the

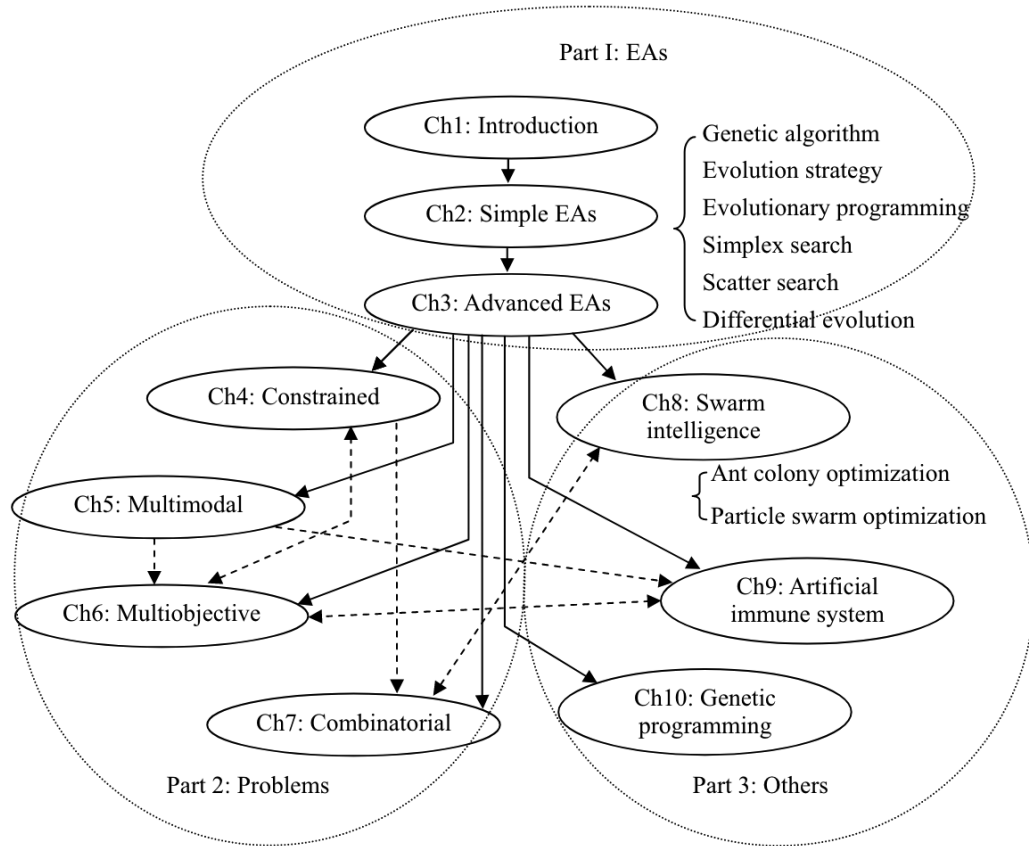


Figure 3.45 – Types of evolutionary algorithms. As presented in the book *Introduction to Evolutionary Algorithms* by Yu and Gen 2010. Image credits: Yu and Gen 2010.

specific technique they are employing.

Omitted techniques

Expert Systems, Rule-based reasoning and model-based reasoning Case-based reasoning and by extension, *case-based design* is a technique used in *Expert systems* together with rule-based reasoning and model-based reasoning (Chowdhary 2020, p.90). However, I have not included rule-based and model-based reasoning in the taxonomy. Rule-based reasoning is not included as it overlaps with techniques in A.1. Computational Geometry, A.2. Grammar-based, A.3. Constraint-based and A.4. Simulation approaches. An example of model-based reasoning are design patterns (Bhatta and Goel 2002; Goel and Bhatta 2004; Goel and Craw 2005) and Christopher Alexander’s pattern language (Alexander, Ishikawa and Silverstein 1977). Case-based reasoning relies on a *case base* which contains complete solutions and retrieves the most suitable one and modifies it (Heylighen and H. Neuckermans 2003). On the other hand, a catalog of patterns is a collection of partial, non-specific, but generalized solutions that need to be retrieved and integrated (Heylighen 2000). In their review, G. Kelly and McCabe 2006 refer to the model-based approach as *template-based generation* and give the work on road networks generation by Sun et al. 2002 as an example. However, I have not identified computer implementations of model-based reasoning for the generation of buildings.

Besides *case-based design*, Knowledge-based expert systems that use functional

modeling, ontology modeling, and rule-based modeling of procedural knowledge are not included. They are well-represented in task planning, controlling, diagnosis, as well as the more closely related field of mechanical engineering (Shang 2005). Nevertheless, I couldn't find relevant examples in the design of buildings. However, advances in digital fabrication, mass-customization, and industrial construction seem to create new opportunities for revisiting those techniques for architectural design (X. Yin et al. 2019).

Function-based Synthesis is a large field of research in mechanical engineering and product design (Chakrabarti et al. 2011) with some key works by Umeda et al. 1996, Stone and Wood 1999, and Bohm, Vucovich and Stone 2008. However, after a thorough keyword and backward, forward citations search, no works have been identified that are relevant for modeling architectural artifacts.

3.2.6 Observations

Key observations:

- The distinct steps that every generative technique has are: setup, initialize, run, post-process, learn.
- Set grammars, split grammars, Wave Function Collapse, Model Synthesis, as well as Case-based design through their suitability to generate building design and their ability to explicitly encode design intent are most suited for assistive design systems (Table 3.5).
- Combination between a *constructive* and a *generate-and-test* technique (genetic algorithm) are often seen. Combinations between two or more constructive techniques rarely occur but show great potential.
- Several combinations of techniques that are not yet explored in architecture but with great potential based on their geometric properties and strength and weaknesses. WFC plus split grammars. WFC plus MC - applied in games but not yet in architecture.

Steps in a generative technique

I propose to structure the use of generative design into the following distinct steps:

1. the development or setup step - the work and data needed to prepare the algorithm for generating designs. In some techniques, this step consists of two sub-steps. For example, data collection and training in the case of example-driven generative techniques.
2. defining the initial conditions for the generative procedure - the work and data needed to condition the algorithm for a run
3. the generative procedure or run step - some techniques have two sub-steps here, for example, reproduction and selection in genetic algorithms.
4. the post-processing step - any work or data needed to transform the output of the generative algorithm for further use.

5. learn and improve - the step to analyze the outcome and figure out strategies to improve the previous steps.

Different techniques focus on these steps differently. For example, the work by Parish and Müller 2001 on procedurally generated cities has a significant emphasis on the development step to define the L-System rules as well as in the initial condition defining step, where the user needs to paint maps of density and terrain features. Another example is the machine learning techniques used by Hu et al. 2020 which require a significant amount of well-curated data in the development step. Yet, a reasonably little information as to the initial condition to run.

The post-processing and the learn-and-improve step are not considered in the subsequent techniques analysis. They are not truly part of the technique but are part of its practical application. The post-process step contains manual tasks performed mainly by the experts to make the outcome usable in their design process. For example, 3D modeling if the outcome is a 2D layout. These tasks vary from expert to expert and highly depend on the individual workflow and experience. In the learning step, if present, the developer, with input from all other roles, evaluates the outcome and learns how to improve the previous steps of the technique.

Techniques most suited for generating buildings

Table 3.5 summarizes the observations for each class of generative techniques in regards to their suitability to generate buildings and encode design intent.

Suitability for generating building designs Techniques that are most suitable to generate building designs are Parametric modeling, Generative modeling languages, Shape grammars, Wave Function Collapse, Model Synthesis, Case-based Design, and Evolutionary Algorithms. As mentioned above, the suitability of Evolutionary algorithms to generate buildings depends on the choice of Constructive technique used in the genotype to phenotype mapping. Generative Machine Learning, while suited well for encoding design knowledge through the datasets that drive it, produce fuzzy or high-level results and, as such, are not suited, simply on their own, for the precision required from models of building designs.

Suitability for encoding design knowledge Techniques that are most able to encode and reuse design knowledge and intent are Shape Grammars, Wave Function Collapse, Model Synthesis, Generative Machine Learning, Case-based Design, and Evolutionary Algorithms. In general, these are techniques that operate with one or another form of database of designs. In contrast, most purely algorithmic techniques do not easily lend themselves to encoding design knowledge. While able to generate building designs with high precision and control, applying Parametric modeling and Generative modeling languages often relies on an expert. This makes them less suited for making design expertise available to others by explicitly encoding it in the generative technique.

Vocabulary-, example- and data-based techniques most suitable for assistive design tools Most suited for assistive design systems are the grammar-based techniques operating on integral sets of shapes, i.e., vocabulary, such as set grammars or split grammars, the methods using tile-sets such as Wave Function Collapse,

Model Synthesis, as well as the techniques using templates such as case-based design. These techniques, as shown above, are suitable for the generation of building designs. In addition, they can encode design intent explicitly.

Table 3.5 – Generative techniques rated on their suitability to generate buildings and encode design intent. *-low to none; **-medium; ***-highly suitable. Image credits: the author.

#	category	suited for buildings	can encode expertise
A.1.	computational geometry		
A.1.1.	fractals	*	*
A.1.2.	subdivision algorithms	**	*
A.1.3.	parametric modelling	***	**
A.1.4.	generative modeling languages	***	*
A.1.5.	Marching Cubes	**	**
A.2.	grammar-based		
A.2.1.	symbolic grammars	**	**
A.2.2.	shape grammars	***	***
A.3.	constraint-based modeling		
A.3.1.	Wave Function collapse	***	***
A.4.	simulation approaches		
A.4.1.	cellular automata	**	**
A.4.2.	agent-based	*	**
A.4.3.	physically-based models	**	**
A.4.4.	Monte Carlo methods	*	*
A.4.5.	topology optimization	**	*
A.5.	example-driven algorithms		
A.5.1.	model synthesis	***	***
A.5.2.	generative machine learning	**	***
A.6.	analogy-based design		
A.6.1.	case-based design	***	***
A.7.	reinforcement learning	*	**
B.1.	search-based		
B.1.1.	evolutionary algorithms	***	***
B.1.2.	simulated annealing	*	**
B.1.3.	particle swarm optimization	*	**

Technique combinations (Families)

Pipelines vs. single technique The application of generative design in Computer Science and for procedurally generated content for games papers tend to use approaches where several techniques are chained in a pipeline to produce the final result. The most frequent combinations are between a genetic algorithm and a

constructive system, as in the work by (Jackson 2002) on evolving designs generated with an L-Systems. Another example is the work by Mahlmann, Togelius and Yannakakis 2012 who used Genetic Algorithms to produce low-resolution versions of maps that get post-processed using Cellular Automata. Another example is the work by P. Merrell, Schkufza and Koltun 2010 who use a Bayesian Network to generate a building program, a Monte Carlo method (the Metropolis algorithm) to fit a floor plan to the program, and computational geometry techniques to generate the 3D models of the houses. On the other hand, the application of generative design in the field of architecture tends to focus only on one generative technique (Singh and Gu 2012). Singh and Gu 2012 note that this biases the design in a constrained direction. One possible explanation of this is the purpose of the developed technique. In computer science, the aim is usually full automation of the time-consuming and costly process of content creation (R. M. Smelik et al. 2014; Togelius, Yannakakis, et al. 2011). In contrast, in architecture, the objective is assistive, inspiration-inducing assistance in one step of the design process (Richter 2007; Singh and Gu 2012). Another related explanation is the object-oriented (result-oriented) thinking of architects compared to the process-oriented thinking of computer scientists.

Combinations of techniques afford interactivity The layering of techniques into generative pipelines has the potential to allow interaction. For example, Speller, Whitney and Crawley 2007 state that “the combination of [Shape Grammars] for managing the input and [Cellular Automata] for managing the output brings together the human intuitive approach (visualisation of the abstract) with a computational system that can generate large design solution spaces in a tractable manner.”

Combinations of techniques with unexplored potential While already used in the game industry as shown by Wender and Watson 2014 the potential of combining case-based reasoning with reinforcement learning for generative design has not been explored in architecture yet. Another potential is identified in combining split grammars (Wonka et al. 2003) with Wave Function Collapse (Karth and A. M. Smith 2017) as the rule selection mechanism. Further potential is identified in the combination of tree-based subdivision algorithms in combination with tiling techniques such as marching cubes (Savov, Winkler and Tessmann 2020) and Wave Function Collapse, as the combination can resolve the homogeneity of the square grid in WFC while enriching the otherwise too schematic layout resulting from Slice tree structure (Kado 1995) or k-d trees (Knecht and Koenig 2010) techniques.

Variety of necessary human engagement

The challenge in generative design techniques is considering social and political aspects (Heylighen and H. Neuckermans 2001). Even in the cases when that was the aim of the authors, the inability to represent them as dimensional requirements proves how challenging it is to reduce architectural design to quantitative constraints (Heylighen and H. Neuckermans 2001).

The need to generate content for content-generation techniques Generative techniques in architecture or games are created usually for a specific application or following a particular idea. The space of possible designs that such a specifically

developed method produces is narrow if no post-processing step for interpreting the outcomes occurs. This problem of sameness or boredom is exhibited by heavily procedurally generated games such as *No Man Sky*, where all worlds feel the same to players even if they look different. The gameplay suffers from this repetitiveness (Kollar 2016). The need for higher diversity in the generated designs prompts for larger vocabularies, larger rule sets, more samples — in general — more content for the generative algorithm. For example, Wonka et al. 2003 state that 200 rules in their initial version of *Instant Architecture* is not enough and that about 3000 rules need to be created, requiring six months of architectural work, for the system to create believable cities. This need for content generation is one potential application of crowdsourcing explored in this work. Very promising generative techniques are those using datasets, such as generative machine learning. However, unlike in computer science, where popular datasets, like the MNIST, are being used by various researchers, such datasets do not exist in architecture. Most likely due to the high diversity of designs but also other reasons. Human input can be used to collect and process the data required to build such datasets, where crowdsourcing and citizen science have already proven to show results in other fields.

The need for human-driven guidance and content evaluation for generative techniques Another example of the sameness problem above is the famous toothbrush generating system by architect Greg Lynn who asked the question “How do you know when to stop generating new designs, since all toothbrushes look the same after a while?”. Instead of generating more content for the algorithm, the question points to an alternative solution, providing a reasonable evaluation of when a design is fit for the problem at hand. As architecture deals with wicked, i.e., ill-defined problems, a weakness of all generative techniques is taking into account the social and political aspects of design since they cannot be reduced to dimensional constraints or explicit rules. That necessitates the engagement of not only the tool-maker and the architect but also of other stakeholders.

3.3 Human-in-the-loop Potential

The primary purpose of this review of generative design in architecture is to identify the techniques that can be made interactive and be used to develop co-creation environments where human actors and algorithms co-create designs.

Assessing the performance of a generative algorithm in terms of how good are the designs it creates is hindered from the very nature of design, namely that it is difficult to assess the quality of a design (Johnson 2016; Lange 2016; Mountstephens and Teo 2020; Rittel and Webber 1973). This makes the human input a necessity and the existence of multiple, equally acceptable design options (Mountstephens and Teo 2020; Rittel and Webber 1973). “Humans are required to evaluate multiple options, even if they did not create them manually” (Mountstephens and Teo 2020; Schulz et al. 2018).

In the past, present, and for the foreseeable future, Generative Design involves humans, and the design work is shared between man and machine (Johnson 2016; Lubart 2005; Mountstephens and Teo 2020).

The research conducted by Nicholas Negroponte and his department at MIT, *The Architecture Machine Group*, is an early example of using models for communication

between an architect and a computer (Negroponte 1970).

In this section, I explore the potential for a human actor to influence the design outcome of a generative technique by applying decisions based on their system of values. The property of various value systems to be non-quantifiable, not measurable, not comparable, and quite diverse across the stakeholders of a project is one of the reasons why we speak of wicked problems in architecture (Lange 2016; Rittel and Webber 1973). Trade-offs are a vital concept (Mountstephens and Teo 2020; Schulz et al. 2018) and decisions when and what to trade off can be made following a system of values.

Gero 1990 formulates the qualities of three types of designs (Figure 3.46): routine, innovative, and creative. Let us consider the state space of possible designs as defined by the encoded knowledge of a given generative design system. What is the chance we can leave that space and enter the creative one through human engagement? What is the result of human interaction? Does it lead to the production of routine design, the generation of innovative designs, or the creation of creative(out-of-the-box) designs?

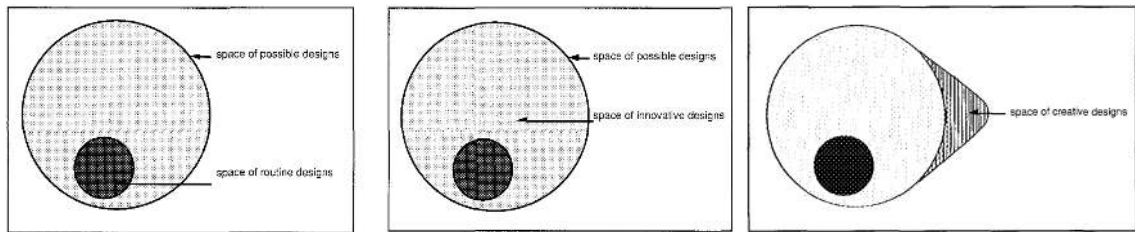


Figure 3.46 – The Space of possible designs. Routine Designs (left), Innovative Designs (middle) and Creative Designs (right). Image credits: Gero 1990.

The potential for human-in-the-loop of each division of techniques depends mainly on two things. The first one is the types of input needed to produce an outcome at each step. And second, how easy or tedious it is for the different roles in a project to provide this input. As defined in section 1.3.3, the four roles are: *tool-makers*, *expert stakeholders*, *non-expert stakeholders*, and *the crowd*. A key involvement of the expert roles is required in the setup and the run steps, and a key area of involvement of the role of non-expert stakeholder is considered to be the init and run steps.

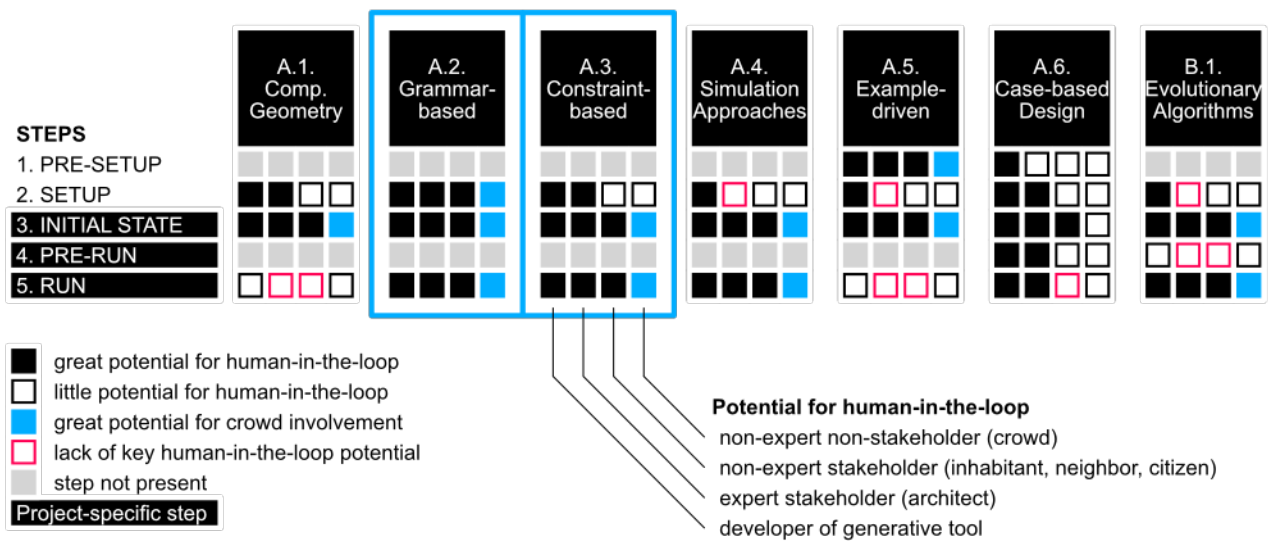
Lubart 2005 presents a framework to think of human-computer collaboration: Computer as a nanny, computer as a pen-pal, computer as a coach, and computer as a colleague. According to Lubart 2005, the computer as a nanny uses the computer to manage, i.e., shepherd the users, the computer as pen-pal is a communication platform, computer as a coach is an expert system, and computer as a colleague is a true collaboration. Based on this spectrum, I consider a generative technique to have great potential for human-in-the-loop integration when we can use it as a coach or a colleague. No user involvement or using the technique as a nanny or a pen-pal will give a generative technique low potential for human-in-the-loop integration.

As illustrated on Table 3.6, I argue that:

1. Discretized inputs, as found in Marching Cubes, Split Grammars, Set Grammars, or Wave Function Collapse, are more user-friendly for non-experts.
2. *Constructive* approaches offer a greater potential for taking diverse value systems of human actors than *generate-and-test* approaches.

3. Out of the constructive approaches the *grammar-based* and *constraint-based* ones offer the greater potential for engaging non-experts,
4. Case-based design is very suitable for engaging experts.
5. Computational geometry and simulation approaches tend to be driven mostly by their developers and offer the least potential for being guided by experts or non-experts.

Table 3.6 – Potential for human-in-the-loop. Image credits: the author.



3.3.1 Types of input representations

What input is required for a generative design technique to produce an output? And how does input to the algorithm relate to its output?

The type of representation of the content operated on by an algorithm is an essential means to distinguish between different techniques (Togelius, Yannakakis, et al. 2011). According to Togelius, Yannakakis, et al. 2011 there are five types of genotype to phenotype mapping on a spectrum from direct to very indirect. Since genotype is, in essence, the representation needed by a *constructive* technique to produce a design, the five types can be generalized for the various types of input required by any technique to produce a design. Therefore, I have used the scale of five representation types from Togelius, Yannakakis, et al. 2011, but gave them more precise names and given examples as listed below:

1. **cell states list** – the most direct type of representation. The design space is represented as a regular or irregular grid. Each cell in the grid must be assigned a state (wall, free space, door, room type, facade element) for the technique to produce a design. Examples here are the Marching cubes algorithm. A good interactive example is the game Block'hood (Figure 3.49) (Sanchez 2015). A tile-filling or grid-filling heuristic such as the Wave Function Collapse can be employed to reduce the amount of information the user needs to provide. In

my work, presented here, *Sensitive Assembly* (chapter 7) is a fitting example, where players define the cell states of wall components by physically removing blocks from the wall. This type of representation lends itself also very well for the creation of physical user interfaces, as in the work by D. Anderson et al. 2000 using physical lego-like blocks with sensors to create building designs (Figure 3.47).

2. **object list** – more indirectly. The design space can be continuous or discrete. The algorithm needs a list of the positions, orientations, and dimensions of walls or rooms to produce a design. The work by Arvin and House 1999 (Figure 3.33) is an example.
3. **rules list** – even more indirectly. The design space can be continuous or discrete. The algorithm needs a catalog of different reusable patterns of walls and free space and a list of rules guiding how they are to be arranged across the design space. Shape grammars are here, for example, the Palladian Grammar in Stiny and Mitchell 1978. Split shape grammar could also be considered in this category.
4. **requirements list** – very indirectly, based on high-level requirements. The algorithm needs a list of desirable properties (number of rooms, doors, area) to generate a design. An example of this is the work by P. Merrell, Schkufza and Koltun 2010 or case-based design precedents such as CADRE.
5. **a number** – most indirectly. All the algorithm needs to produce a design is a random number seed. The Perlin noise algorithm is an example.

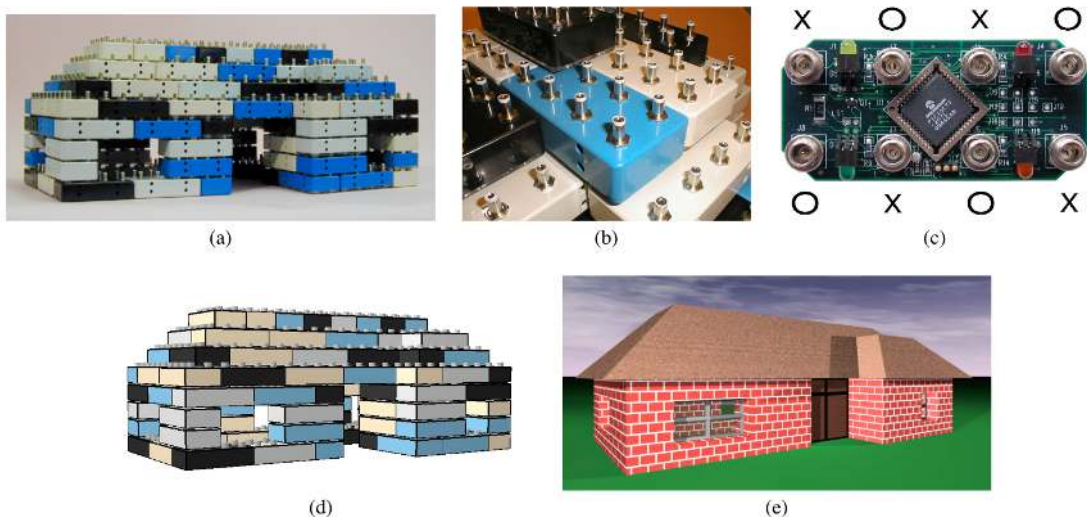


Figure 3.47 – Cell-based user input. This type of input representation is suitable not only for digital input but also for using physical components for the user to interact with. The physical components of a modeling system (a)-(c), their digital representation(d) as well as the generatively interpreted representation(e). Image credits: D. Anderson et al. 2000.

The more direct the mapping between input and design is, the more predictable it is for users how their actions influence the final result. As such, those kinds of representations are very suitable for non-experts. However, more information is

required in more direct mapping, i.e., the complete grid must be filled with cell states. If a heuristic is not employed, but the human actor must assign cell states manually to each cell, creating and editing a design can be tedious.

On the other hand, the more indirect or abstract the mapping between input and design outcome is, the less predictable it is for the users how their actions influence the design. However, an advantage is that the required information is compressed down to a list of requirements and sometimes can be as little as a single seed number.

(R. Smelik et al. 2010) present a concept and a framework called declarative modeling that integrates direct editing (2.) with procedurally generated content based on meta specification (4.) (Figure 3.48).

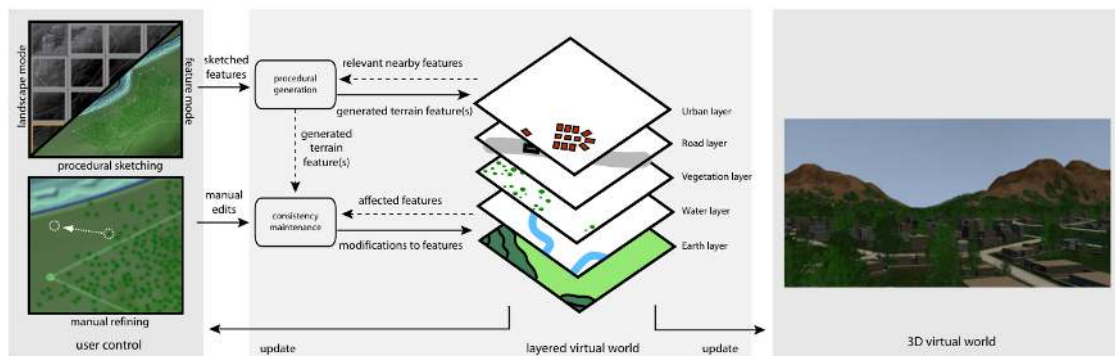


Figure 3.48 – Requirements and objects lists input. A combination between the more abstract requirements list input which R. Smelik et al. 2010 call declarative modeling and the more direct object list input. Image credits: R. Smelik et al. 2010.



Figure 3.49 – Block'hood by Jose Sanchez. Left: screenshot of the game, Right: an example of the urban ecology concept around which the game is built. Image credits: Plethora-project.

The review by R. M. Smelik et al. 2014 focuses on interactive and intuitive control. They list the following approaches: Sketch-based techniques, Visual editors, Inverse Procedural Modeling. Furthermore, R. M. Smelik et al. 2014 state that “user control issue is the reversibility of operations. Designers very often tend to (ab)use do-undo combinations, in order to assess the net effect of a particular operation: if it does not yield the desired result, they just backtrack. However, many PM methods are not always reversible, and it is therefore not always possible to exactly restore the previous model situation” (R. M. Smelik et al. 2014).

3.3.2 Constructive vs. Generate-and-test

A *Constructive* generative technique aims to create a design once while ensuring the result is an appropriate option (Togelius, Yannakakis, et al. 2010). Constructive approaches can be as simple as taking a few points as input and producing a space tessellation (Voronoi) or as elaborate as encoding design expertise within a grammar or a neural network and using heuristics to arrive at a design.

In a constructive approach, human input can act either as the information source or as the heuristic (Harding 2014, p.20). In a generate-and-test approach, on the other hand, human input primarily aims to enhance the meta-heuristic (Harding 2014, p.36). In a constructive approach, at each step of the construction process, the user potentially can interact and "guide" the construction process to the next step. This makes it more predictable what user actions lead to what results and intuitively master the tool by using it.

Generate-and-test approaches generally offer a low possibility for user intervention once fitness functions, genotypes, and termination conditions are defined (Singh and Gu 2012). Generate-and-test techniques generate tens, hundreds, and thousands of options and learn from their fitness values how to create better options in the next iteration. This large number of designs that must be created makes it infeasible to engage human actors in the *generate* sub-step for two reasons. First, the number of designs generated within a population is prohibitively large, and the process will be too time-consuming — better to have that automated. And second, the ability of the algorithm to learn to generate better options in the next iteration is rendered pointless from the fact that this learning must be passed on as instructions or feedback to the human actors who would then be expected to follow it. This also makes the human input pointless as it is prescribed, i.e., technically predictable and automate-able. The ability of the generate-and-test algorithm to learn requires the systematic exploration of diverse options and the automated application of the learning in the next iteration.

So it remains only feasible to engage human actors in the *test* sub-step of a generate-and-test approach. In the case of evolutionary algorithms, this approach goes under the terms Interactive Evolutionary Computation (IEC) and, the more popular, Interactive Genetic Algorithm (IGA) (Figure 3.51) (Brintrup, Ramsden and Tiwari 2007). Both have been applied in fashion design (H. S. Kim and Cho 2000), the generation and texturing of 3D game environments (Yoon and K.-J. Kim 2012) and the generation of floor plans (Banerjee, Quiroz and Louis 2008; Brintrup, Ramsden and Tiwari 2007; Quiroz et al. 2009). (Liapis, Yannakakis and Togelius 2012) present three types of evaluation (one automatic and two involving the user) on Figure 3.50. The ones involving the user are interesting here. In the first one, the user selection is direct, while in the second, the user selection influences the fitness function. However, engaging humans in a generate-and-test technique by making the *test* step interactive comes with the four main negatives.

1. the inability of human actors to evaluate a large number of designs due to cognitive or physical exhaustion (Brintrup, Ramsden and Tiwari 2007).
2. the challenge to ensure proper learn-by-comparison for the meta-heuristic algorithm if users rate on a binary scale (keep/reject) instead of a continuous scale (Togelius, Yannakakis, et al. 2011).

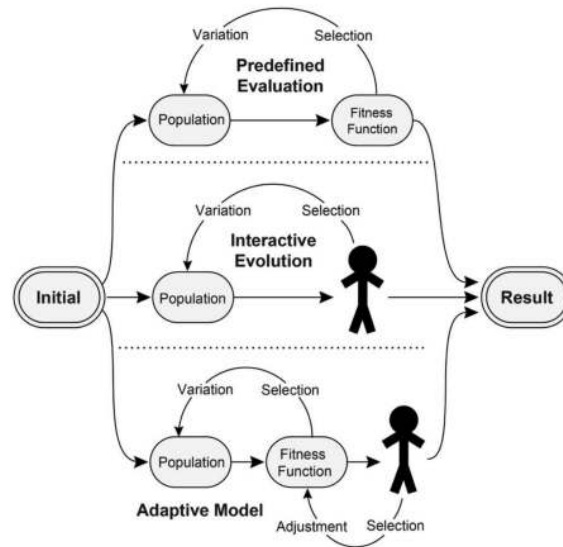


Figure 3.50 – Types of design evaluation. One automated and two involving the user - direct and indirect. Image credits: Liapis, Yannakakis and Togelius 2012.

3. the inconsistency with which humans rate on a continuous scale. This is due to fatigue; the influence from any quantitative rating presented to them; the influence from any better or worse designs they have seen until now, not remembering all previous design; the difficulty for a human to perceive minor qualitative changes between designs as differences and assign a rating; or the change in the implicit rating scale the user uses to rate as they are exposed to more information during the rating process (Brintrup, Ramsden and Tiwari 2007).
4. if the user makes a new input, the result changes entirely, making it difficult to intuitively "learn" to use the tool from simply using it.

As one possible solution to fatigue, Brintrup, Ramsden and Tiwari 2007 propose *active intervention*. This allows the user to modify a design to a far better qualitative state and reinsert it to speed up the convergence to a solution. Modifying a design to make it better practically means the user can guide the process's *generate* step with its corresponding constructive technique for genome-to-phenotype mapping. Therefore the *active intervention* solution to human fatigue speaks in favor of focusing on engaging human actors in constructive techniques instead of the generate-and-test techniques.

In conclusion, constructive approaches are better suited for taking qualitative human input into account through interaction than generate-and-test approaches. In constructive techniques, the human actors can genuinely influence the outcome with decisions based on their value systems. In contrast, the tool-maker already has set up the evaluation system in a generate-and-test approach. Hence, it is a given and, in a way, constraining.

Looking at the individual steps of a generative technique as defined in section 3.2.6, generate-and-test techniques offer the following tasks for potential human actors (Table 3.7). In the Setup step, the constructive technique for the genome-to-phenotype mapping and the fitness functions must be chosen. These are so specific that it is primarily the tool-maker who can perform them. In the initializing step,

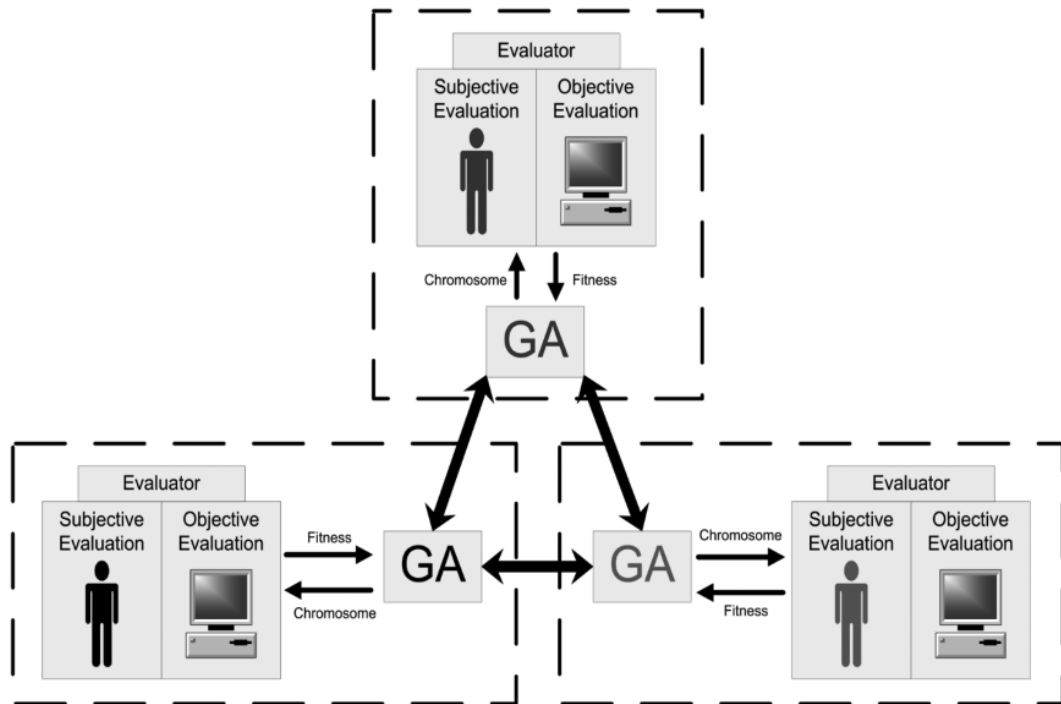


Figure 3.51 – Subjective and objective evaluation in Interactive Genetic Algorithms (IGA). Image credits: Quiroz et al. 2009.

the initializing tasks of the particular constructive technique chosen for the generating step must be completed, potentially making the tasks here open for all four roles (see the constructive techniques analysis below). In the run step, reproduction is automated, but testing can be made interactive to all four roles, albeit not truly suitable, as shown above.

Table 3.7 – Human-in-the-loop potential of Evolutionary Algorithms. Image credits: the author.

STEPS 1. PRE-SETUP 2. SETUP 3. INITIAL STATE 4. PRE-RUN 5. RUN	B.1. Evolutionary Algorithms 	<p>Example tasks in the SETUP step:</p> <ul style="list-style-type: none"> - define genotype-phenotype mapping (typically a constructive technique from A.x.) - define available fitness functions <p>Example tasks to define the INITIAL STATE:</p> <ul style="list-style-type: none"> - define the initial phase of the constructive technique (A.x) chosen for phenotype mapping - define mutation and other parameters - pick fitness functions and goals <p>The PRE-RUN step is the autonomous reproduction</p> <p>Example tasks in the RUN step:</p> <ul style="list-style-type: none"> - evaluate and rank population and select top designs for reproduction or as potential outcomes <p>Potential for human-in-the-loop</p> <ul style="list-style-type: none"> non-expert non-stakeholder (crowd) non-expert stakeholder (inhabitant, neighbor, citizen) expert stakeholder (architect) developer of generative tool
--	---	--

■ great potential
 □ little potential
 ■ crowd involvement potential
 ■ key role not involved
 ■ step not present
 ■ Project-specific step

3.3.3 Techniques with great potential for engaging non-experts

A.2. Grammar-based

In grammar-based generative techniques the tasks in each step are as follows (Figure 3.52).

1. In the setup step, an interpreter must be implemented, and the grammar (vocabulary plus rules) needs to be defined. For symbolic grammars, also a mapping between the grammar and the geometrical representations needs to be created (usually a computational geometry technique) (Gips 1999; Stiny 1980b; Togelius, Shaker and Dormans 2016).
2. In the initial state step, the starting seed for the grammar needs to be defined as well as any context that will influence the grammar derivation (Parish and Müller 2001; Stiny 1980b)
3. in the run step, the grammar derivation or production takes place (Wonka et al. 2003). Chase 2002 divides the derivation step into three sub-steps: select a rule, select a part of the model to modify, determine the matching condition to apply the rule (Figure 3.53).

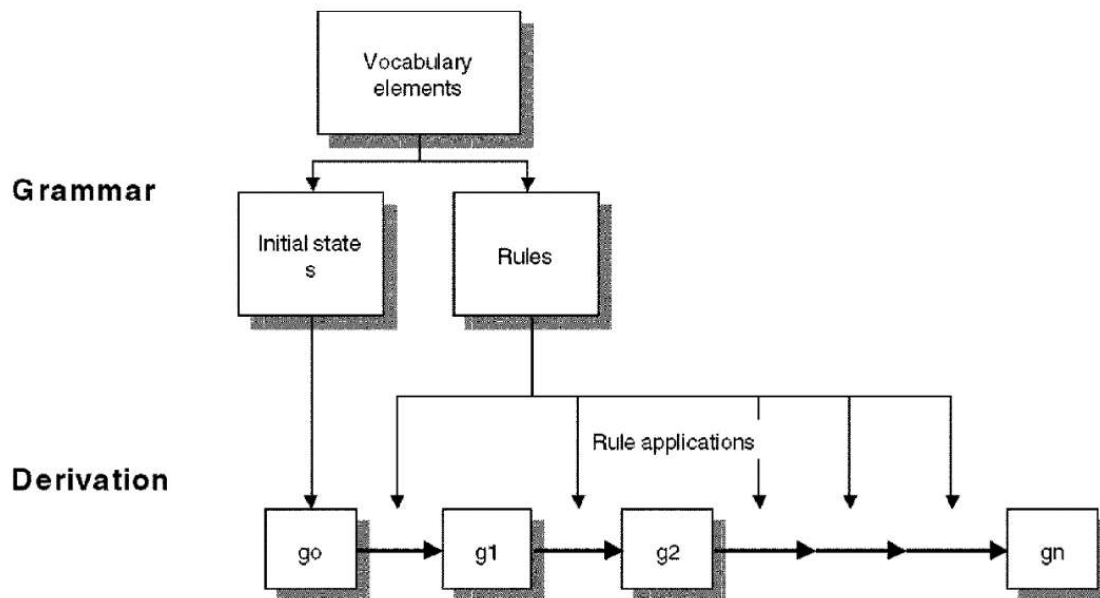


Figure 3.52 – Tasks for the user in grammar-based techniques. Image credits: Chase 2002.

The user's ability to influence the outcome of a shape grammar derivation is in three levels: modify the grammar directly, alter the starting/initial shape, influence the choice of the rule to be used in the next step either by attribute modification of the vocabulary present in the current state or through direct rule selection (Wonka et al. 2003). On Table 3.8 summarizes which of the roles could be engaged in which steps of a grammar-based generative modeling technique.

On Figure 3.53, Chase 2002 presents the possible scenarios for using a shape grammar interactively with two roles in mind, the developer and the designer. Scenarios 1, 2, and 3 are explored in the case studies presented in this dissertation,

Table 3.8 – Human-in-the-loop potential of Grammar-based techniques. Image credits: the author.

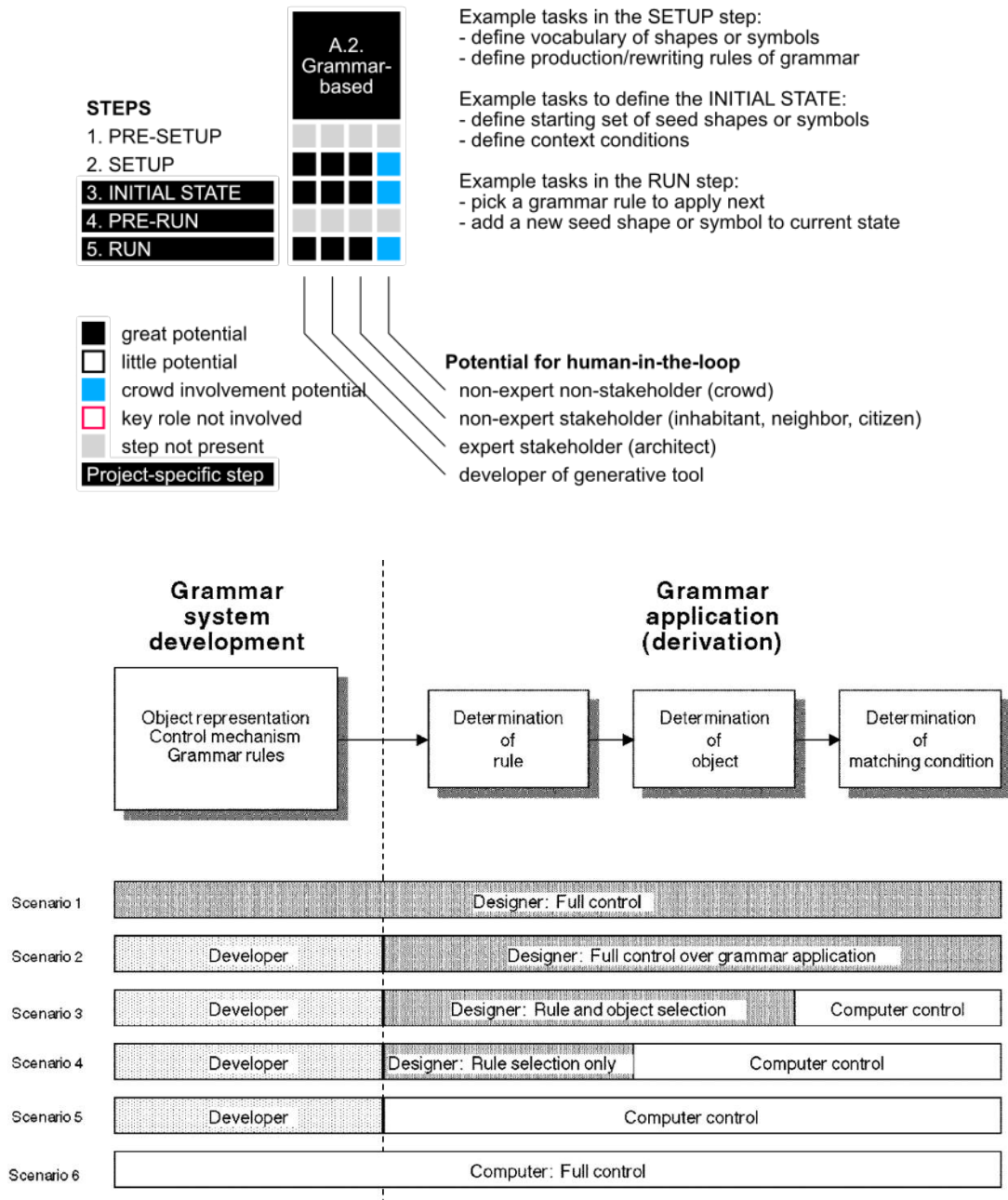


Figure 3.53 – Scenarios for user engagement in grammar-based generative techniques. The different users (designer, developer, computer) can have various degrees of control over the four steps in the development and application of a grammar. Image credits: Chase 2002.

where the distinction between 3 types of designer roles are made (expert stakeholders, non-expert stakeholders, and the crowd).

Shape grammars offer benefits for the engagement of non-expert users. Gu and Behbahani 2018 state that because “the design knowledge can be embedded into the grammar, users of the grammar do not need to have disciplinary expertise in order to generate a design.”

However, specifying shape grammars and deriving designs from them is not easily

accessible to non-experts and experts alike. Chakrabarti et al. 2011 explain that “a main roadblock to achieving wider impact, especially in conceptual design, has been to support designers in the iterative development of a grammar, without having to program it directly [...]”

Node-based editing, aka visual scripting, (Figure 3.54) makes rule specification and rule selection easier by minimizing the need to define grammars in text form (Patow 2012; Silva et al. 2013). However, it still requires specialized 3D modeling software and computational design skills.

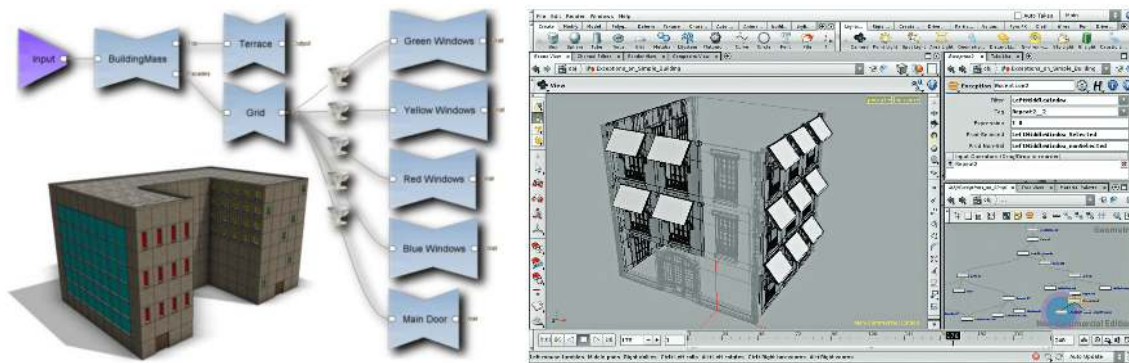


Figure 3.54 – Node-based shape grammar rules specification. Both examples are for the specification of a facade grammar. Left: from Silva et al. 2013, right Patow 2012. Image credits: Silva et al. 2013; Patow 2012.

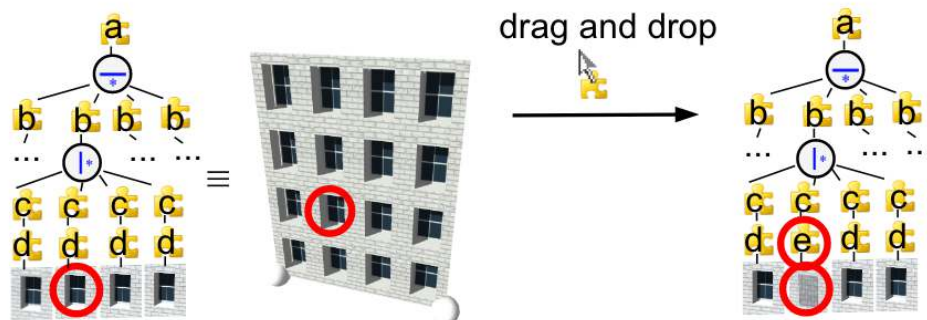


Figure 3.55 – Visual, drag-and-drop grammar rule editing. Image credits: Lipp, Wonka and Wimmer 2008.

Lipp, Wonka and Wimmer 2008 present a method for editing of the rule node in a shape hierarchy with direct manipulation and drag-and-drop over the generated 3D model (Figure 3.55).

(Beneš et al. 2011) present a method that uses Open L-Systems and user-defined wireframe guides to generate various types of objects (Figure 3.56). Each guide hosts a defined procedural model and is linked to the other guides to ensure topological continuity of the outcome. However, designing the guides themselves, which determines the design outcome, is still done entirely by the designer. The method can be seen as a method to automate the adding of details to a schematic design and offer little direct control over the workings of the procedural models running in the individual guides, which is shown by the patchwork character and the rigidity of the street layouts that the authors have generated (Figure 3.56 right).

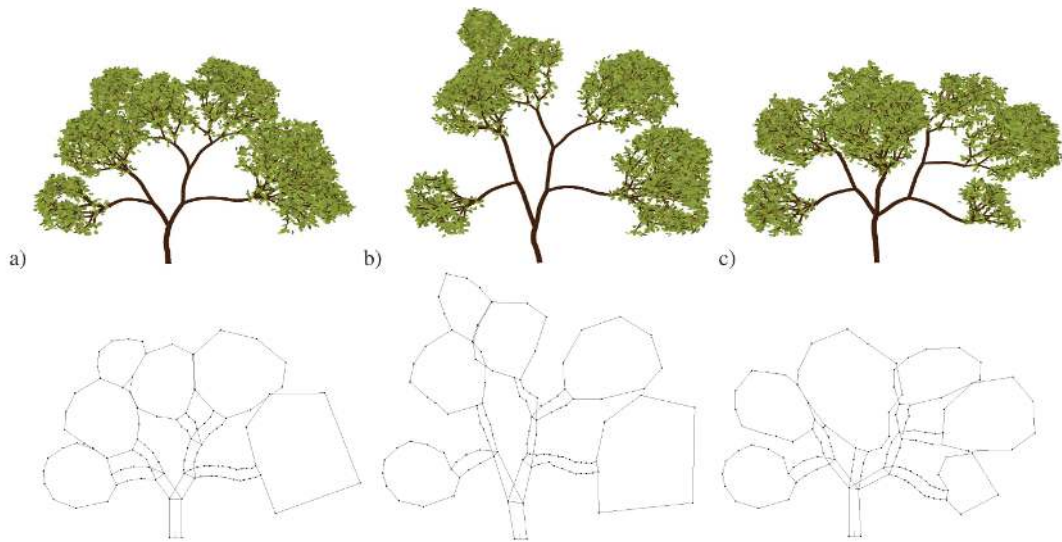


Figure 3.56 – Guided procedural generation using Open L-systems. a) A tree and its shape guides (lower row) are generated, b) the guides are interactively edited, and c) some guides are erased and edited individually. Image credits: Beneš et al. 2011.

Tching, Reis and Paio Jun-2016 tested five shape grammar interpreters and concluded that their interfaces are unfamiliar, outdated, chaotic, and unintuitive.

Besides the explicitness of grammars, making any design knowledge encoded in them accessible to any user, another significant benefit is the shift from focusing on individual designs to languages of designs, making it possible to explore a design problem (Stiny 1980a).

A.3. Constraint-based

Constraint Satisfaction is the assignment of a finite set of variables so that a finite set of constraints are satisfied (Donath and Böhme 2008). In architectural design and computer graphics, this often translates to arranging modules in space in relation to one another such that any constraints on module adjacency and the composition's overall properties (volume, height, etc.) are satisfied (Donath and Böhme 2008; Karth and A. M. Smith 2017).

The tasks in the individual steps in a constraint-based generative technique are:

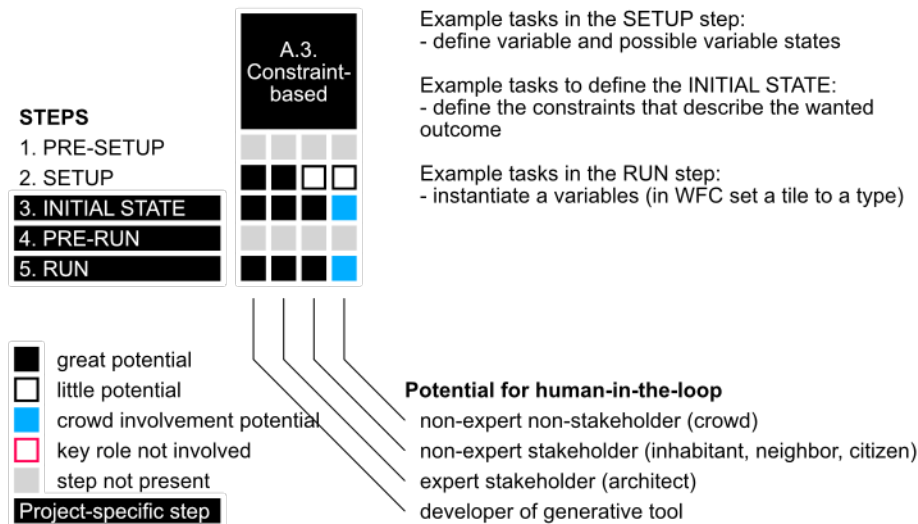
1. Setup - the developer, maybe also the expert stakeholder, define the representation of the design problem as a set of variables, their states, and the possible constraints
2. initiate - the constraints need to be defined. For example, in (Donath and Böhme 2008) a constraint for a building's setback is defined.
3. run - the variables must be set, i.e., in the case of wave function collapse, all grid cells must be assigned a tile type (Karth and A. M. Smith 2017). Or, if the variables are the rooms of a building represented as rectangles or boxes, a composition must be created.

As shown on Table 3.9, given a suitable, intuitive, and assistive interface, the init and run steps could be performed by all four roles. In contrast, the setup step is

rather specific and expected to be carried out by the tool-maker, with the expert’s involvement at most.

Three approaches can be observed in the stat-of-the-art. First, the user defines a set of constraints, and then, again the user attempts to create such a variable assignment to satisfy the constraints. That is seen in the work of (Donath and Böhme 2008; Koile 1997). Second, the set of constraints is given by the user, but an algorithm tries to create a composition of variable assignments that satisfy them. And third, the set of constraints and all possible variable states are predefined in the setup phase as a catalog by the tool-maker or the expert. The user simply partially or completely instantiates a variable assignment. The third option is most user-friendly for engaging non-experts and the crowd and has been explored in the case study *Project Reptiles*. The second option has been explored in the game *Rechteck2BIM*.

Table 3.9 – Human-in-the-loop potential of Constraint-based techniques. Image credits: the author.



3.3.4 Techniques with great potential for engaging experts

A.6. Case-based Design

The potential tasks for human actors in the steps of a case-based generative technique are:

1. setup – first, pre-setup, a knowledge representation must be defined, which the tool-maker must do. Then the case base must be filled with cases (buildings or building elements) described and represented as per the chosen representation format.
2. init step – the design problem at hand needs to be described in terms of the tags, keywords, and other representation forms so that a search can be done.
3. run – in the pre-run, the retrieval is performed. This can be done automatically by matching the initial step’s requirements or manually selecting a design or

a mix of the two. And in the actual run step, the chosen one or more cases are merged and adapted to the context and other specifics of the problem at hand (R. E. Oxman 1992).

As shown on Table 3.10, due to most case-based systems aiming at being assistive tools for experts and the highly specific methods for representing, retrieving, and adapting cases, only domain experts can potentially be users (Figure 3.57). At the most, the non-expert stakeholder could be brought in the problem specifying init step.

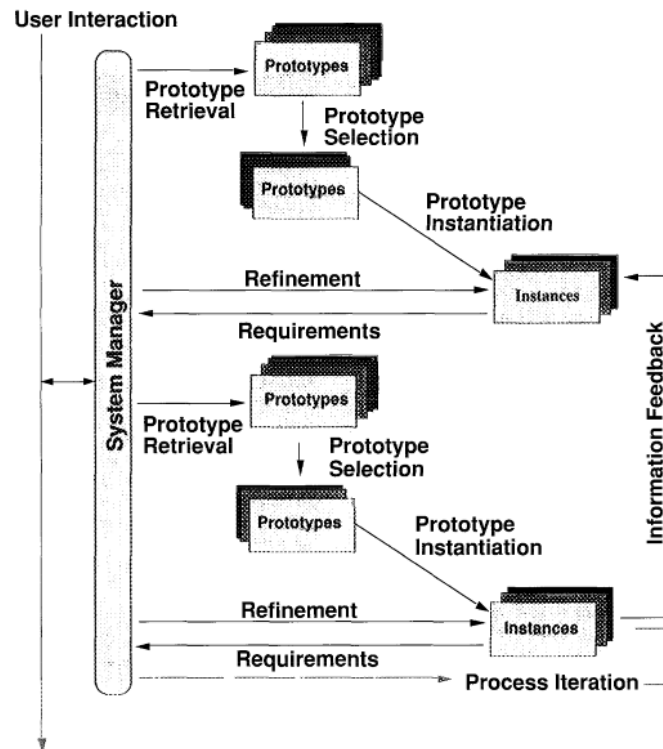
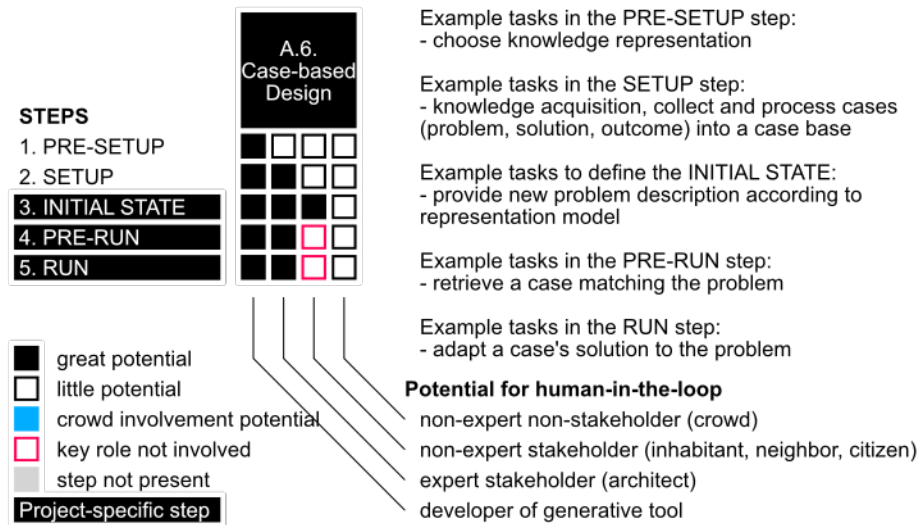


Figure 3.57 – User interaction in Case-based design. Image credits: Gero 1990.

Table 3.10 – Human-in-the-loop potential of Case-based Design techniques. Image credits: the author.



3.3.5 Techniques with low potential for human-in-the-loop

Generative techniques with a lower potential for engaging human actors are also better at running autonomously. Due to this, they are not able to accommodate the human input needed to address the wicked part of an architectural problem, so they will not be considered on their own in this work but only in combination with the techniques from subsection 3.3.3.

A.1. Computational Geometry

The division Computational Geometry consists primarily of well-known algorithms (Voronoi tessellation, Half-Plane Intersection, Kd-Trees) (de Berg et al. 1997) and techniques that employ a set of programmatic instruction written as a general programming language or as data-flow programming (visual scripting, parametric design). This means that the potential tasks for human actors in the individual steps of a computational geometry generative technique, shown on Table 3.11, are:

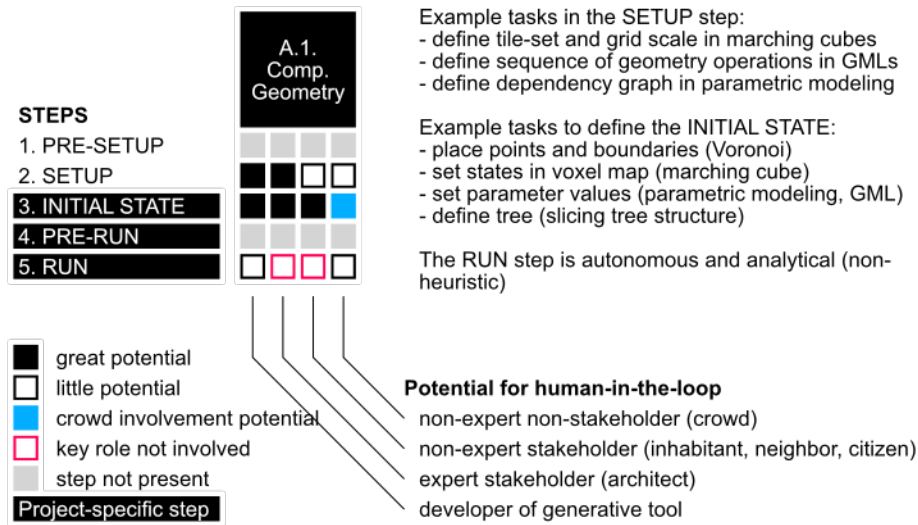
1. setup – the implementation of a chosen algorithm. This requires particular domain knowledge and coding skills, so it is usually performed by the tool-maker and, in some cases, by an expert stakeholder.
2. init – setting up the parameters needed for the algorithm to produce the geometry. The fact that the input is often minimal and simple, e.g., a set of points in the case of Voronoi tessellation, makes it suitable to engage all four roles
3. run – is more or less autonomous execution of the script, so there is little opportunity for involving a human actor.

Zboinska 2015 is an example of interactive generative modeling using computational geometry techniques.

Computational Geometry techniques offer control over the design outcome as the algorithms are often non-heuristic. Therefore, human input cannot be employed as a heuristic.

Computational Geometry techniques have been used in the case studies here only as a complementary step to automate a task. For example, in the automatic wall, window, and roof generating in the case of *Rechteck2BIM*. Or the use of Marching Cubes in *Project Reptiles*.

Table 3.11 – Human-in-the-loop potential of Computational Geometry. Image credits: the author.



A.4. Simulation Approaches

The primary mode of interaction in simulation approaches is for the user to introduce disturbances in the simulated environment. The simulation integrates them in its further growth and iterates itself into a new equilibrium state. An example is the shifting between manual and automated operations in the cellular automata model of Herr and Kvan 2007, used for the design of towers (Figure 3.58). Similarly, Vanegas et al. 2009 present a method for interactive design of cities using behavioral modeling through agents to generate the city layout and computational geometry approach to generate the buildings (Figure 3.59).

According to Singh and Gu 2012 Cellular Automata is a purely bottom-up generative modeling technique, and it offers low user intervention once cell dimensions, state rules, and initial states are defined. This applies in general to all other simulation approaches, such as agent-based and physically-based ones.

The simulation-based approaches are the ones very often used in city-building computer games such as *SimCity* and *City Skylines* where the user introduces new conditions in the environment and the city simulation iterates over them to give the player feedback on parameters such as urban density, economic prosperity or population happiness.

As shown on Table 3.12, in simulation-based generative design techniques, the tasks in the setup step are implementation of the simulation mechanisms and environment representations, in the init step is the introduction of starting condition in

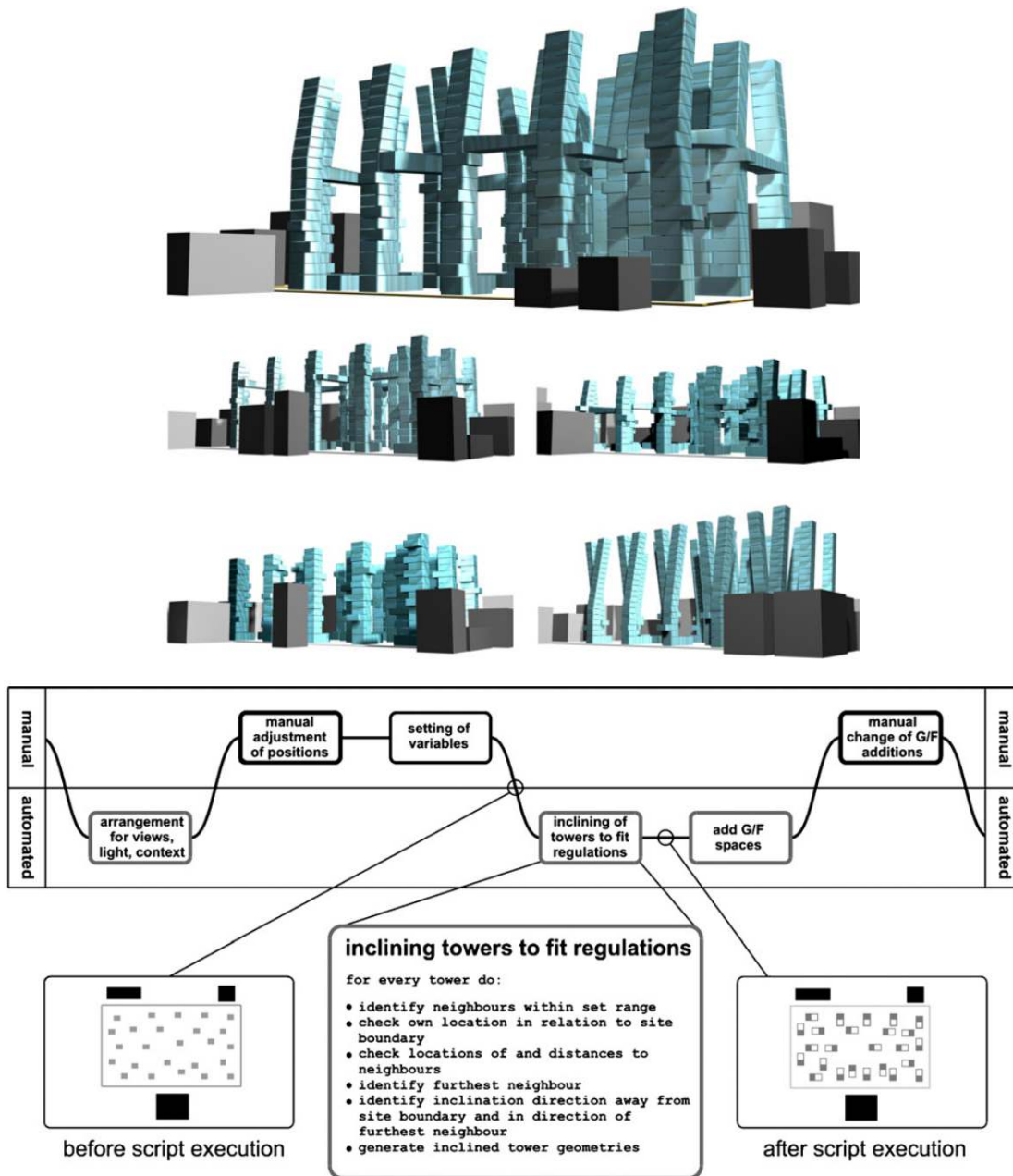


Figure 3.58 – To guide the design outcome the user changes the environment conditions in simulation generative techniques. Bottom: a diagram illustrating the shifting of manual introduction of new environment conditions by the user and the model's reaction to it. Top: the resulting towers produced from the cellular automata. Image credits: Herr and Kvan 2007.

the simulated environment. In the run step, this is the iterative introduction of new conditions in the environment to guide the autonomous simulation mechanisms.

The problem is that all rules for the simulation are already encoded in the simulation engine. Even if user interactions are allowed, they are within the mechanisms and evaluations built in the system. And since the development of a simulation requires highly specialized knowledge, it is often impossible for experts to participate in the setup step.

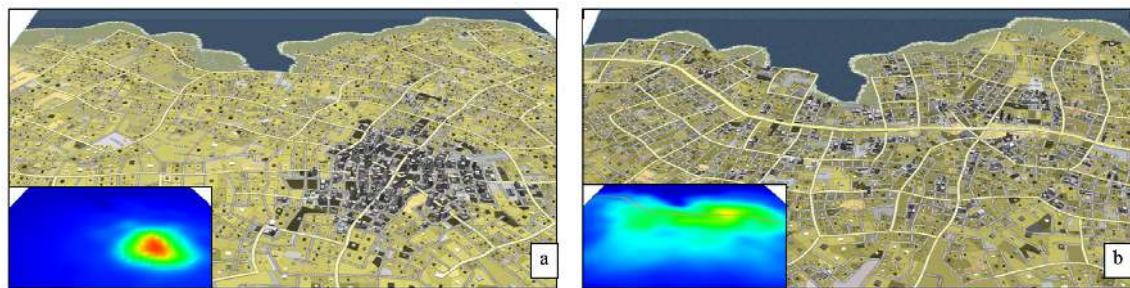
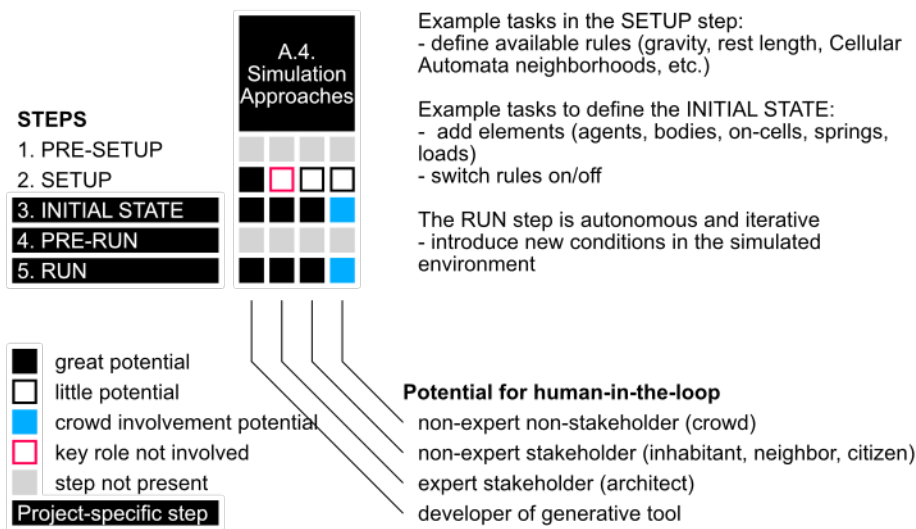


Figure 3.59 – Visually introduced disturbance in the environment. A designer can paint a new highway (visible in the false color map in (b)) and let the simulation transform the city from a dense downtown type (a) to a more widespread one (b). Image credits: Vanegas et al. 2009.

Table 3.12 – Human-in-the-loop potential of Simulation Approaches. Image credits: the author.



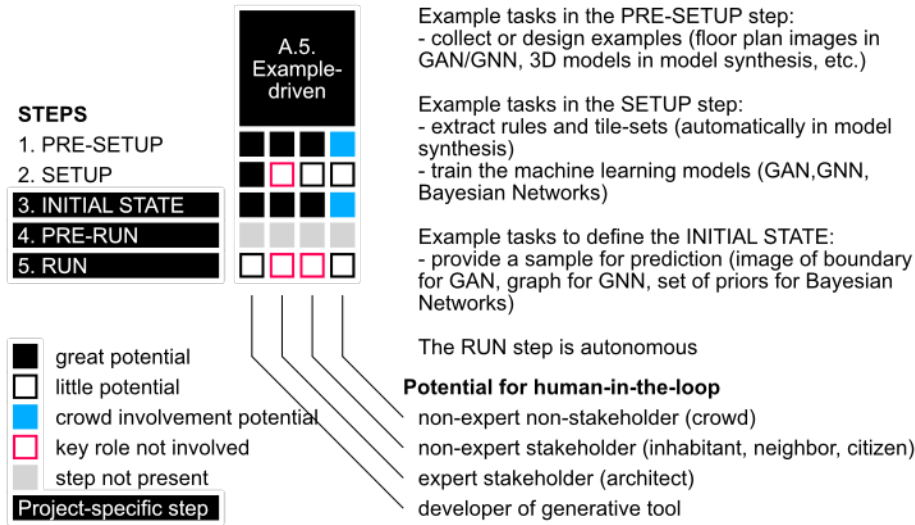
A.5. Example-driven

Example-driven techniques such as generative machine learning show great potential for engaging all roles in the setup step, i.e., the data/collection and training step. However, the later steps require specialized knowledge of human actors to get involved, and in the case of machine learning, models are often considered black-boxed.

As shown on Table 3.13, the tasks in the setup the collection of example, e.g., compiling a labeled database of floor plans as in the work of Hu et al. 2020 and the model training, in the init step, is the providing of a sample. The run step is entirely autonomous, spitting out a prediction based on the trained model.

As projects like *Galaxy Zoo* show, the compiling and labeling of a database, i.e., the pre-setup step, can be successfully crowdsourced to non-experts (Raddick et al. 2010). However, the later steps are more or less closed off for the engagement of the key roles. Hence the exclusion of this division of techniques from the case studies in the dissertation.

Table 3.13 – Human-in-the-loop potential of Example-driven techniques. Image credits: the author.



3.3.6 Takeaways

In design, the designer’s intentions influence the designed artifact (J. McCormack, Dorin and Innocent 2004). Since there is no algorithm to predict intentions, human participation will remain essential in all computer-assisted design processes, no matter their level of autonomousness.

In general, interaction is easier to stage in the generate-and-test category since, at the test step, the user can be engaged instead of algorithmic testing. The user actions can be reject/keep (Togelius, Yannakakis, et al. 2011) and rank on a discrete or continuous scale (Brintrup, Ramsden and Tiwari 2007). The user’s engagement is to browse through a large set of design variations (see examples below). This can create choice fatigue and other issues that (Brintrup, Ramsden and Tiwari 2007) summarize in section 6 of their paper. So while many works have been published that deal with interaction in the test step of a generate-and-test generative technique, the challenge of this research is to stage interaction during the constructive flow of a constructive generative technique. The hypothesis is that users will be more included and that the problem’s true, wicked nature will be acknowledged better.

What we are interested in is also the correlation between the algorithm and the generated model or the level of difficulty to predict the outcome by merely reading the algorithmic description (Caetano, Santos and A. Leitão 2020). This is important when the aim is to make the generative design techniques usable and user-friendly for non-experts and architects with no prior experience in computational design.

Social and political aspects cannot be reduced to dimensional constraints. Therefore they are challenging to model in all generative techniques. Can crowdsourcing be used to counter this weakness of algorithms?

Types of interactions

One point of view to take is of course the types of interactions. The types of input from human to drive the outcome, that I discuss in my case studies are:

- direct physical or digital object manipulation (move, scale) - *Rechteck2BIM*, *Sensitive Assembly*;
- set a cell/pixel/voxel to a state - *Project Reptiles*;
- rule selection and placement - *20.000 BLOCKS*;
- edit modules(vocabulary) in a catalog - *Project Reptiles*, *20.000 BLOCKS*;
- retrieve similar designs based on current design state - *Sensitive Assembly*, *Project Reptiles*.

Further type of input, not taken into consideration:

- Input as image maps or hand-drawn sketches (de Villiers and Naicker 2006; Parish and Müller 2001).
- Guides as polygon shapes modified by user in (Beneš et al. 2011).
- Control grammar rules in text form and split grammar rules in shape form (Wonka et al. 2003).

Crowdsource the Weaknesses

Another perspective to take is which step's tasks can be crowdsourced for each technique?

Which is to say - where are the weak spots in all these generative algorithms, and in which ones can crowdsourcing help? Knowledge acquisition in CBD is a weak spot — how can crowdsourcing help?

For example, in shape grammars, the labor-intensive creation and specification of grammars can be outsourced. It also makes sense to outsource the exploration of the design language defined by a given grammar by opening the derivation step for many participants and collecting the outcomes. Similarly, in the case of Wave function collapse, designing the tileset as well as the generation of designs can be outsourced.

3.4 Potential for Automation in Construction

The use of robotic processes in construction can be considered one of the strategies to encode expertise. Skilled workers usually perform construction tasks. Those skills are made available to non-trained participants by automating parts of the process. Therefore a vital component of the hypothesis in this work is the integration of construction constraints and specificities into the participatory design process.

This section reviews the precedents of robotic construction in architecture and how they relate to the taxonomy of generative design techniques. The aim is to identify generative design techniques with the highest potential for direct integration with an automated construction process. Two aspects can be automated: the production of individual parts and the assembly of those parts.

After reviewing the state-of-the-art, I identify three main approaches to robotic construction: design-agnostic fabrication, fabrication-aware computational geometry, and digital materials.

The concepts of holistic and non-holistic sets of parts are relevant when considering construction (Sanchez 2021). Sanchez 2021 defines a holistic set as a set of parts that constitute a whole that is fixed and unalterable. In contrast, according to Sanchez, the non-holistic set consists of parts that can be combined in topologically diverse ways, i.e., can produce multiple wholes due to the openness of the system (Figure 3.60).

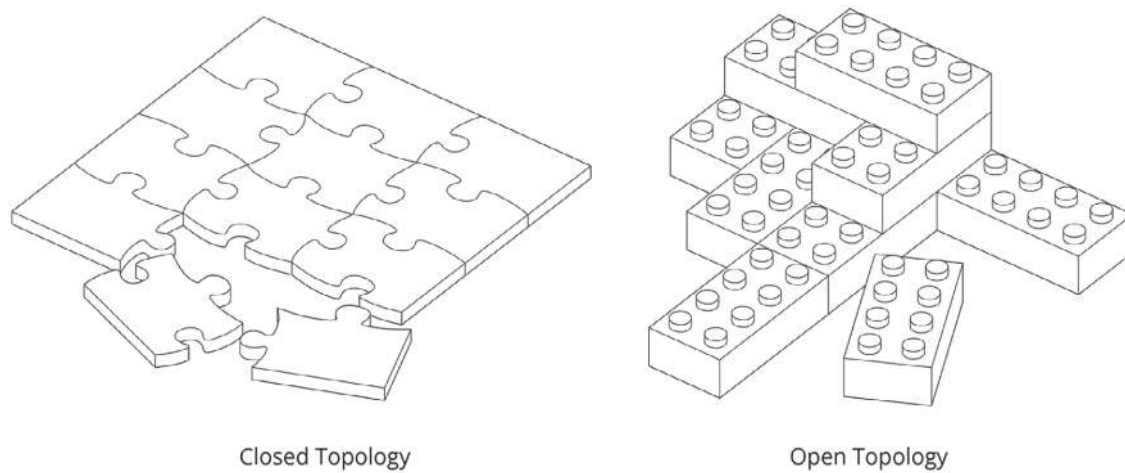


Figure 3.60 – Closed vs open Topologies. A jigsaw puzzle has a holistic set of pieces, i.e. there is one way to correctly put them together. Lego blocks constitute a non-holistic set of pieces, i.e. there are multiple correct ways to assemble them. Image credits: Sanchez 2021.

The digital material approach shows the highest potential for integration with the generative design techniques most suitable for participatory design – grammar-based and constraint-based. The case study *Sensitive Assembly* and some of the case studies with the *20.000 BLOCKS* framework use a basic version of the digital materials approach.

3.4.1 Design-agnostic fabrication

With 3D model slicing or waffle techniques, almost any shape can be digitally fabricated using a 3D printer, CNC mill, or laser cutter. This flexibility is powerful, but we cannot apply construction constraints into the generative technique as the actual digital fabrication happens in a post-processing step. I call this design-agnostic fabrication as the methods do not respect the specificity of the given design but are generic (N. Gershenfeld 2012). And the FabLab movement was based on this approach (N. Gershenfeld 2012). This application of 3D printing and robotic milling is considered analog, and not digital, fabrication (N. Gershenfeld, Carney, et al. 2015; Retsin 2020).

3.4.2 Fabrication-aware computational geometry

Most examples of robotic construction are using parametric design and *A.1. Computational Geometry* in general. Starting with the seminal example from Bonswetch 2006 of the Informed Wall (Figure 3.62), where every brick had a different rotation on the Z-axis, all the way to the latest advances in robotic construction shown by the work of Wagner, Alvarez, Groenewolt, et al. 2020; Wagner, Alvarez, Kyjanek,

et al. 2020 for the BUGA wood Pavilion (Figure 3.63) by ICD Stuttgart, the typical approach is to define the fabrication method and then create the design from a parametric model able to take all possible design states of the fabricated units (Figure 3.64). Fabrication-aware Computational geometry can be thought of as the production of a holistic set of digitally fabricated parts.

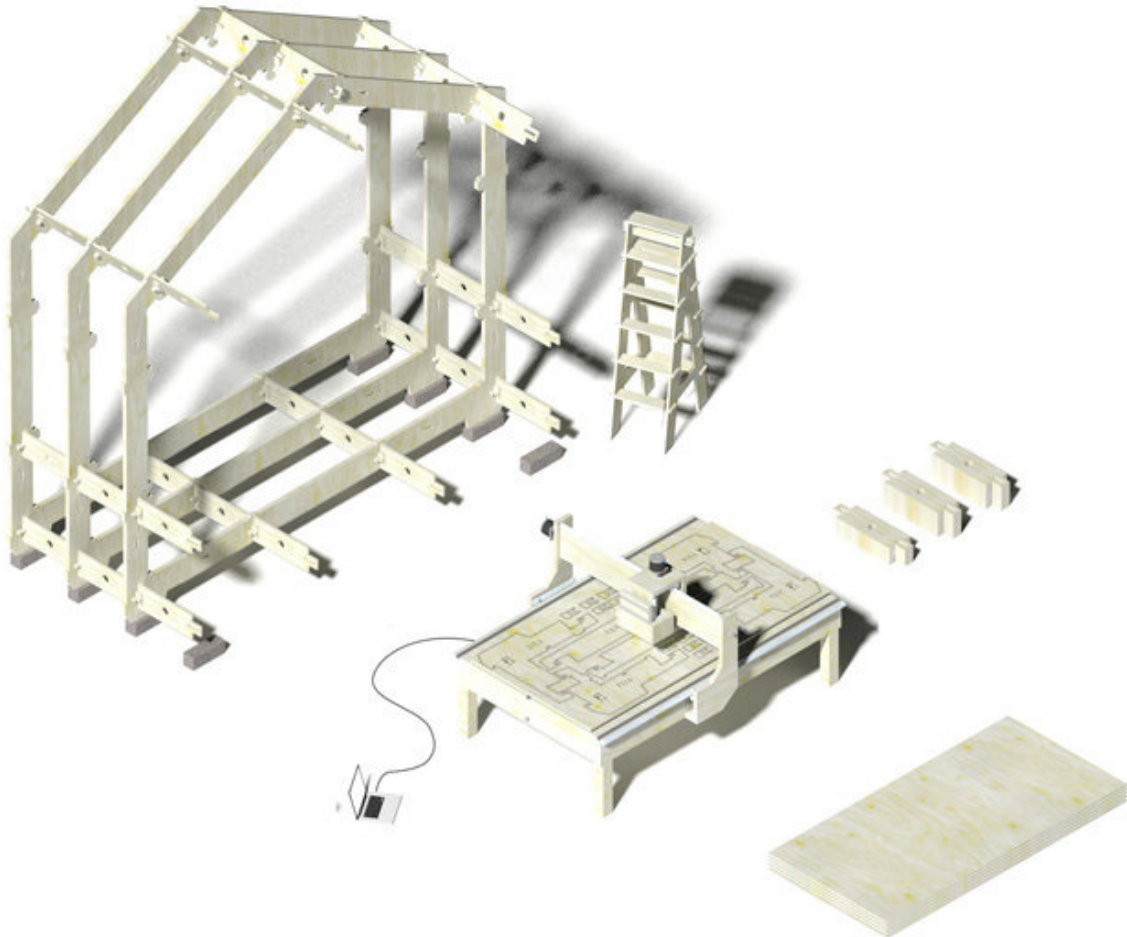


Figure 3.61 – Wikihouse. Wikihouse(Parvin 2013) is an example of fabrication-aware open design aimed at distributed manufacturing using small scale, private CNC cutting machines. Image credits: Parvin 2013.

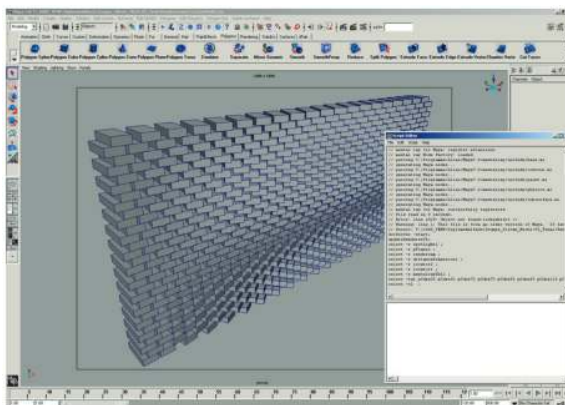


Figure 3.62 – Informed Wall. A parametrically defined brick distribution and the robotically fabricated physical artifact from the Informed Wall project by Bonswetch 2006. Image credits: Bonswetch 2006.

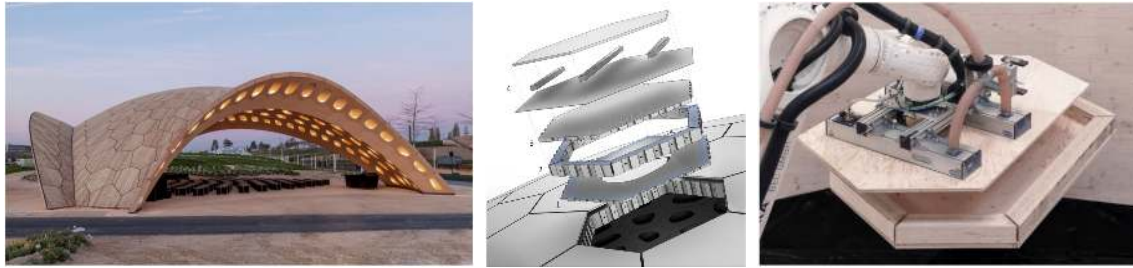


Figure 3.63 – The BUGA Wood Pavilion by ICD Stuttgart. A design that is a population of locally differentiated components on a surface (left), a fabrication aware parametric model of a generic component (middle) and snapshot of the robotic process of manufacturing of a specific component. Image credits: Wagner, Alvarez, Kyjanek, et al. 2020.

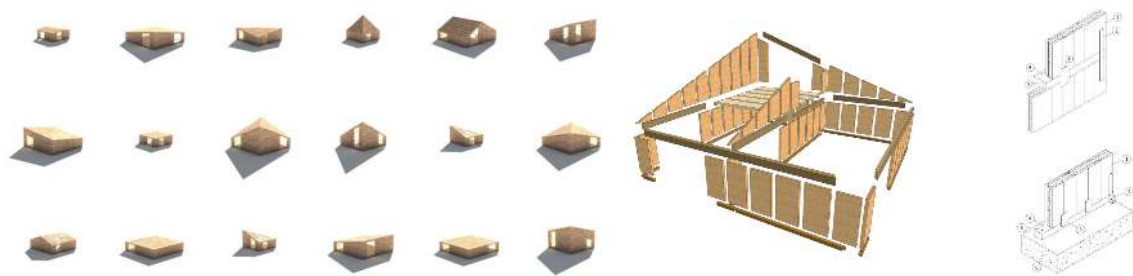


Figure 3.64 – Modularization of a parametric model. An extract of the possibility space (left) of a parametric model (middle) that is aware of the CNC production to be used in the final assembly (right) of a house designed by Bianconi, Filippucci and Buffi 2019. Image credits: Bianconi, Filippucci and Buffi 2019.

The work on robotically fabricated scale models of tower designs (Figure 3.65), presented in Budig, Lim and Petrovic 2014, is a fitting architectural illustration of the fabrication-aware computational geometry approach. Teams of students produced the tower designs during a design studio at the Future Cities Laboratory (FCL), Singapore-ETH Centre for Global Environmental Sustainability (SEC). Each tower design consists of housing units that are parametrically varied based on their vertical position and relation to sun and city views. The units are then assembled from sheet materials using the robot to cut, fold, place and glue the model pieces together.



Figure 3.65 – Parametric fabrication-aware scaled model of towers. Left: Parametric model of housing units; Second from left: unfolded wall surfaces; Third from left: the robotically fabricated tower model; Right: the Future Cities Laboratory (FCL) with three robot stations at the Singapore-ETH Centre. Image credits: Budig, Lim and Petrovic 2014.

My own project, Project Avocado (Figure 3.66), uses fabrication-aware compu-

tational geometry as well.

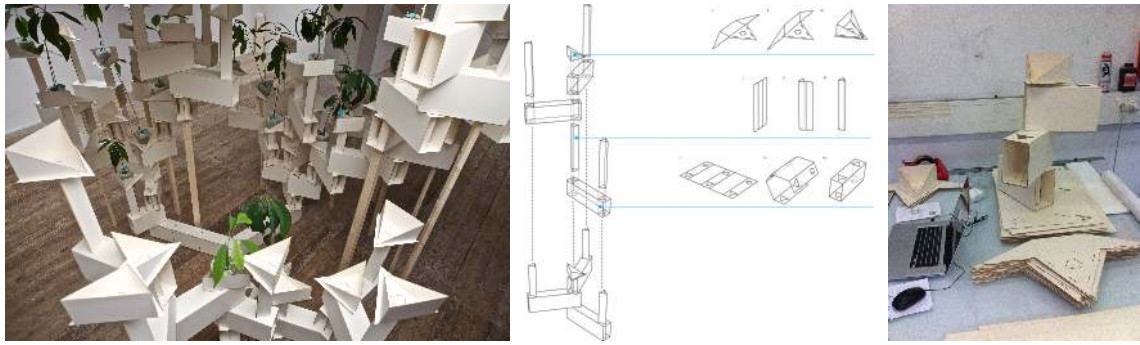


Figure 3.66 – Project Avocado. Final installation (left), unfolded unit model (middle) and laser cut modules(right). Image credits: the author.

Automated prefabrication of wooden frame houses (German: Holzrahmen Fertigbau), used by prefab house companies, also belongs to the fabrication-aware computational geometry approach (Vähä et al. 2013). However, it is more straightforward as the house designs are less generative and more of a design system, in the type of a product configurator (section 3.1).

3.4.3 Digital materials and discrete assemblies

Digital materials (Figure 3.67) are “reversibly assembled from a discrete set of parts with a discrete set of relative positions and orientations” (N. Gershenfeld, Carney, et al. 2015). The concept was first explored at the Center for Bits and Atoms led by Neil Gershenfeld at MIT in a series of key doctoral and master theses (Cheung 2012; W. K. Langford 2019; Popescu 2007; J. Ward 2010). Digital Materials entered the discipline of Architecture under the term *Discrete Design* through the projects and design studios led by Gilles Retsin (Figure 3.68) at the Bartlett (Retsin 2016) and generative tools such as WASP developed by Andrea Rossi at the Digital Design Unit in Darmstadt (Rossi and Tessmann 2018).

In digital materials, the design of the parts prescribes the freedoms of assembly and is often corresponding to the design of the robots and robot actuators (Figure 3.69) (N. Gershenfeld, Carney, et al. 2015; Tessmann and Rossi 2019). The digital materials approach allows for the use of much simpler and smaller robots, that are easier to program and could even be retained as part of the structures they assemble for subsequent reconfiguration (N. Gershenfeld, Carney, et al. 2015). Examples are the termite-inspired brick-placing robots by Petersen, Nagpal and Werfel 2011.

Digital materials constitute open topologies, i.e., are non-holistic sets of parts.

Digital materials and discrete assembly operate with modules of finite predefined shapes. The geometry of parts, and the options for assembly it allows, define a design language similar to the languages defined in a grammar-based generative approach (Rossi and Tessmann 2017c,d, 2018). This makes digital materials very suitable for combination with generative design techniques that use vocabulary catalogs and combinatory rules such as *set grammars*, *split grammars*, *marching cubes* and *wave function collapse*. Individual modules in a vocabulary set could be designed and produced using the fabrication-aware computational geometry approach as seen in the work of Retsin (Figure 3.68) (Retsin 2020).

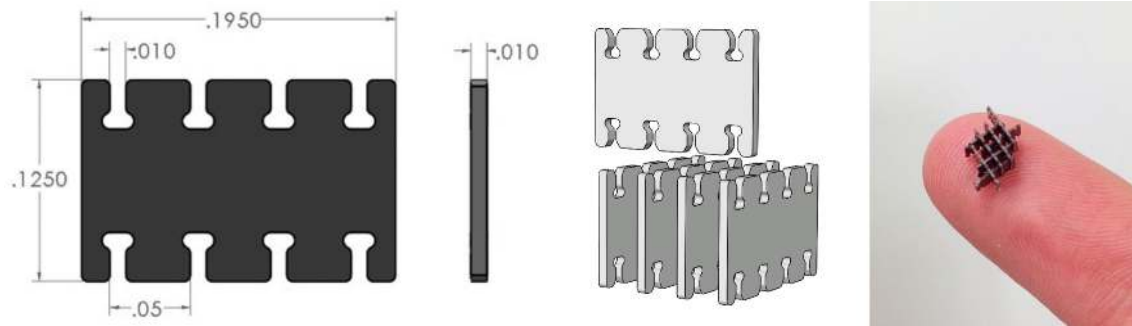


Figure 3.67 – Digital materials. A geometry of a part and the assembly of a sample digital material from W. Langford, Ghassaei and N. Gershenfeld 2016. Image credits: W. Langford, Ghassaei and N. Gershenfeld 2016.

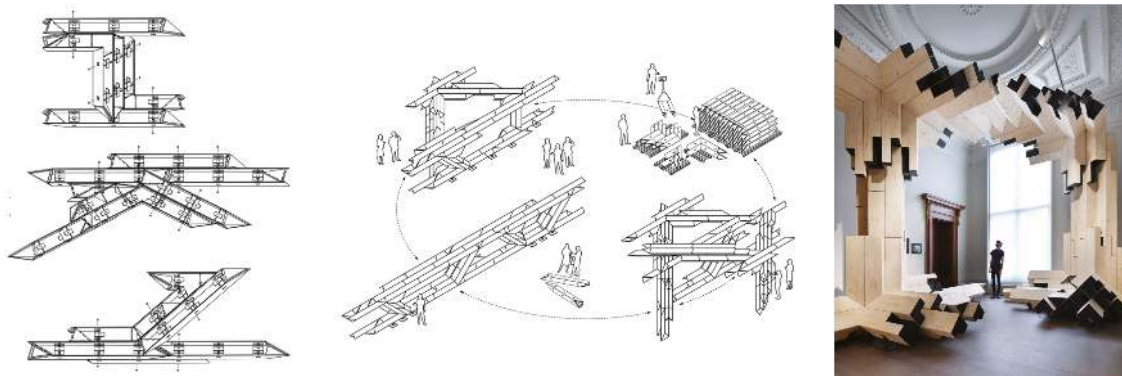


Figure 3.68 – Discrete design. Four types of modules and possible discrete assemblies. Image credits: Retsin 2020.

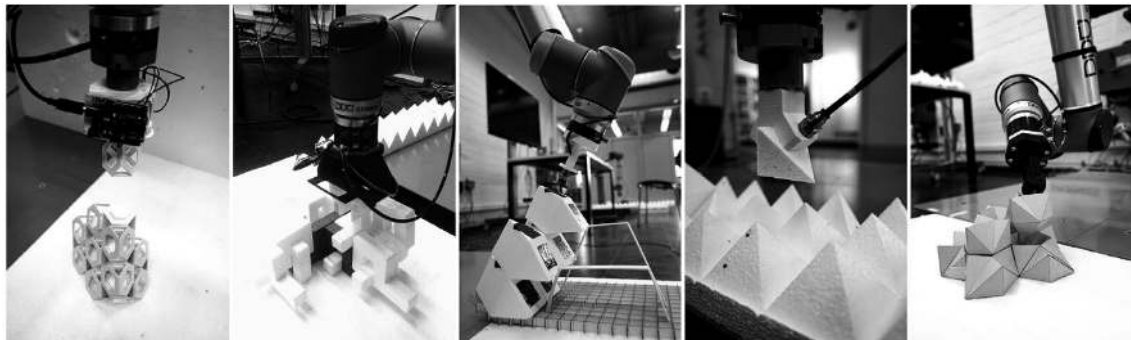


Figure 3.69 – Robotic discrete assemblies. The design of the parts and the design of the robotic grippers to pick and place them are tightly integrated. Image credits: Tessmann and Rossi 2019.

The case study Sensitive Assembly (chapter 7) explores the use of the digital materials approach with two types of units. Each unit is defined using the fabrication-aware computational geometry approach. The robotic process in *20.000 BLOCKS* also pertains to the digital materials approach.

3.5 Conclusion

This chapter reviewed architectural design systems and configurators and established the degrees to which architectural expertise can be encoded in different methods. I introduced a new *Taxonomy of Generative Design in Architecture* (Figure 3.8) and

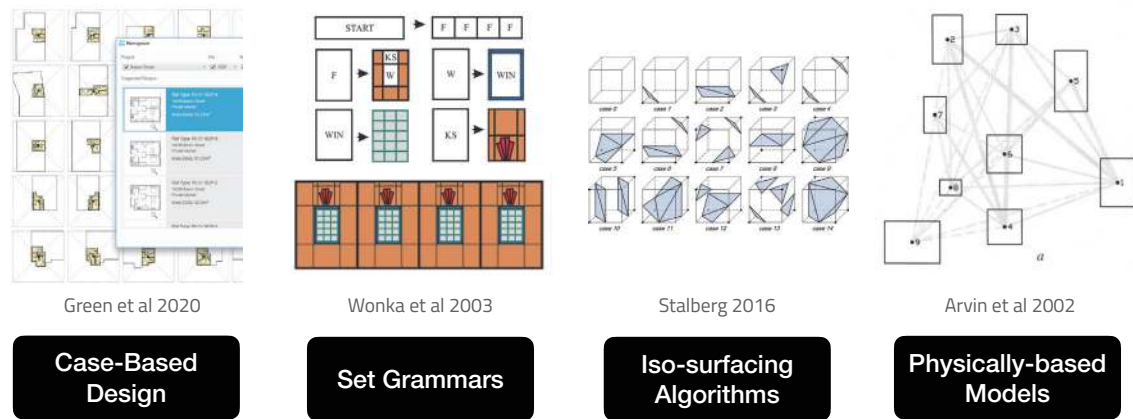


Figure 3.70 – The selected generative techniques. Image credits: the author.

discussed the potential to integrate human interaction and digital fabrication for each class of techniques. When combining the takeaways from this analysis, four techniques appear relevant to explore in my case studies. These are: (i) iso-surfacing algorithms; (ii) physically-based models; (iii) set grammars; and (iv) case-based design (Figure 3.70).

Chapter 4

Crowd Wisdom: Crowdsourcing, Citizen Science and Mass Collaboration

Amazing things can happen when you put technology in the hands of the many. (Apple, Inc.)

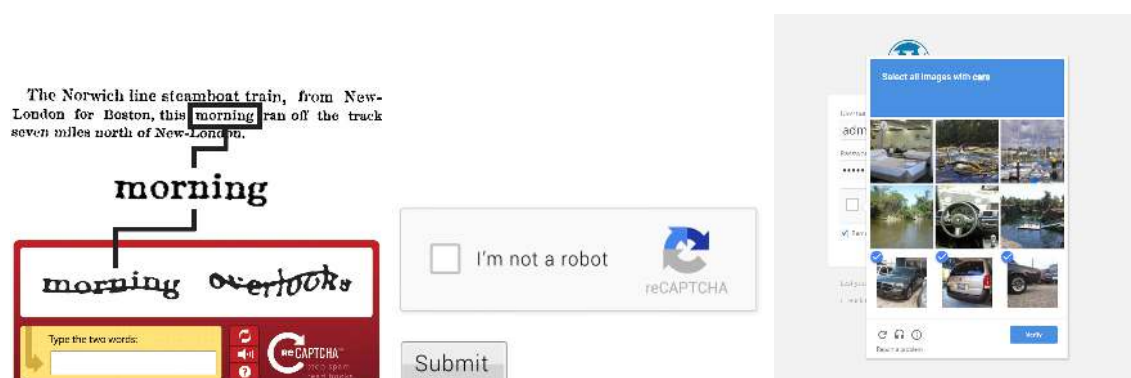


Figure 4.1 – ReCaptcha. Image credits: ReCaptcha.

The ideas that large, seemingly unorganized crowds of people acting on their behalf in a real or virtual space might exhibit some kind of a higher order or intelligence are not new at all. In the 18th century, Adam Smith argued the existence of an *invisible hand* of the market being moved by individual self-interest and resulting in the best interest of society. The father of cybernetics, Norbert Wiener, saw human behavior as patterns that encode messages to and from the environment, other human beings or machines in a feedback loop (Wiener 1950). This uncovered new insights into the studies of law, language, and society. Kevin Kelly, inspired by the developments in communication technology in the late 20 century, theorized of a Hive Mind, a self-organizing system of connected people and an ever-growing number of machines as the future of politics and society (K. Kelly 1994). And Philip Ball presented the concept of *physics of society*, or looking at people as particles in a simulation governed by laws similar to the natural laws of physics, as the means to model, predict and decide issues of city traffic, political and social phenomena and other complex modes of human activity (Ball 2004).

While all these ideas have been questioned and received well-argued criticism (Best and Kellner 1999), they have nevertheless helped individuals look at the world in a new way and create new, working, applicable technologies.

The more recent development is the understanding that the capacity of crowds for self-organization and complex decision-making can be intentionally tapped into and enlisted to solve a specific problem. We have seen several examples of this. First, navigation assistants like *Waze* get better with the continuous input from all drivers using it. Furthermore, the rapid growth of sites like *Wikipedia* where human knowledge is documented and organized by thousands of contributors. Cryptocurrencies, such as Bitcoin, replace centralized institutional trust with the computational power contributed by thousands of independent actors to a blockchain network. And last but not least, revolutions started and rapidly spread via Twitter as in the Arab Spring of 2011-2012. All the above are examples of using the decision power and creativity of non-experts to solve a problem previously in the guarded domain of a few chosen experts.

This chapter introduces the various forms and media that facilitate the coordinated action of online crowds towards a defined goal.

4.1 Definitions

In section 2.3, I defined *participatory design* as a procedure that enables the involvement of stakeholders in the design and construction of a specific building or a city neighborhood. The engagement of the stakeholders is the critical distinction between *participation* and the concepts presented in this chapter. In the related and sometimes overlapping domains of *crowdsourcing*, *citizen science* and *mass collaboration*, the engaged crowds of users, contributors, and players are not necessarily stakeholders of the final product. They take part because of other reasons instead.

It's worth noting that attempts to involve stakeholders at a massive scale have also been made, such as in the case of E-Participation (Herrmann 2016). However, those have failed so far due to the problem of not being able to reach all participants with the relevant information on time as well as issues with communication and coordination of the stakeholders' interests at that scale (Herrmann 2016). *Dunbar's number* suggests that the natural, cognitive limit for groups of collaborating people is around 150 members (Dunbar 1992).

4.1.1 Crowdsourcing

The term *crowdsourcing* is a combination of the words *crowd* and *outsourcing*. Jeff Howe coined the term in 2006 and defined it as “the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call” (J. Howe 2006). Daren Brabham says that “Online communities,[...] are given the opportunity to respond to crowdsourcing activities promoted by the organization, and they are motivated to respond for a variety of reasons” (Brabham 2013). In its essence, crowdsourcing is a business model (Brabham 2013).

As crowdsourcing gained popularity in research and as a business model, Estellés-Arolas and González-Ladrón-de-Guevara 2012 set out to formulate an integrated definition by reviewing more than 200 documents. Their definition:

“Crowdsourcing is a type of participative online activity in which an individual, an institution, a non-profit organization, or company proposes to a group of individuals of varying knowledge, heterogeneity, and number, via a flexible open call, the voluntary undertaking of a task. The undertaking of the task, of variable complexity and modularity, and in which the crowd should participate bringing their work, money, knowledge and/or experience, always entails mutual benefit. The user will receive the satisfaction of a given type of need, be it economic, social recognition, self-esteem, or the development of individual skills, while the crowdsourcer will obtain and utilize to their advantage that what the user has brought to the venture, whose form will depend on the type of activity undertaken.” (Estellés-Arolas and González-Ladrón-de-Guevara 2012)

According to Leimeister 2012 there are three categories of crowdsourcing: *crowdfunding*, *crowdvoting* and *crowdcreation*. *Crowdfunding* is how the Wikimedia foundation¹ funds itself or how Patreon² allows content creators to get financed by their followers. Often Kickstarter³ is considered crowdfunding but it is in essence a pre-purchase platform. *Crowdvoting* is what websites like Amazon or the various app stores use to rank items in their listings.

In my work, I am mostly focusing on *crowdcreation* which is when the crowd is expected to deliver content, whether that is an idea, a design, or simply a tag for an image (Leimeister 2012). Sanz-Blas, Tena-Monferrer and Sánchez-García 2015 split crowdcreation into *crowdwisdom* and *crowdproduction*.

The most common form of crowdcreation is to ask the crowd to perform micro-tasks that are part of a larger task relevant to the organization (Figure 4.2). The anti-spambot program *reCAPTCHA* (Figure 4.1) digitized 13 million books and articles, dating from 1851 till now, with 100 million daily uses (von Ahn 2011). It achieved this by chopping the scanned images of book pages into words and showing these words to internet users when they log into a website.

However, the micro-tasking model can be applied to well-defined problems where the criteria for a good solution are clearly defined. For example, in engineering, this approach can be applied. T. Fischer 2008, p.36 describes how the *Sputnik shock* in the 1960s prompted NASA to rethink the way things are designed. They started chopping engineering problems into small problems and solving those and then assembling the thing back together. In architecture, which deals with wicked problems, it is not immediately apparent how to chop the process of coming up with a new design into micro-tasks.

Several works have described how the crowdsourcing framework is put into practice (Gerth, Burnap and Papalambros 2012; Sanz-Blas, Tena-Monferrer and Sánchez-García 2015; Schall 2012). An integrated version of these precedents consists of the following seven steps (Figure 4.3):

1. choose a task, relevant for the organization,

¹<https://wikimediafoundation.org/> — The nonprofit Wikimedia Foundation provides the essential infrastructure for free knowledge and hosts Wikipedia.

²<https://www.patreon.com/> — Patreon is an American membership platform that provides business tools for content creators to run a subscription service.

³<https://www.kickstarter.com>

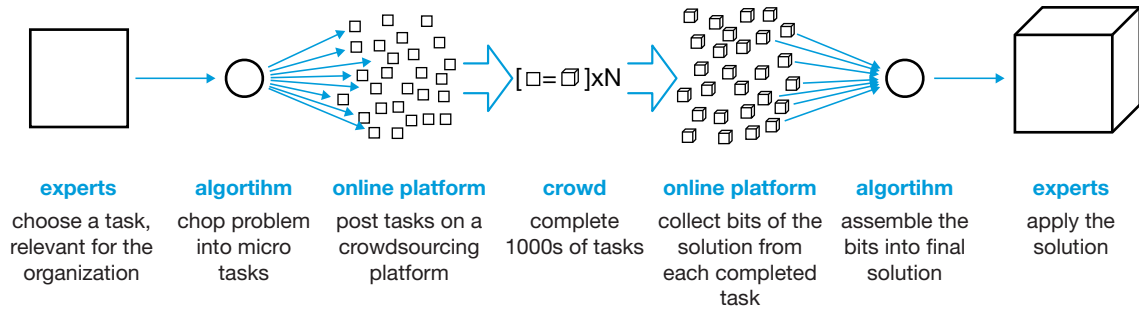


Figure 4.2 – Micro-task crowdsourcing model. Image credits: the author.

2. specify an open call,
3. contribute proposals for a solution,
4. evaluate proposals,
5. rank/sort proposals,
6. accept proposals as a solution,
7. apply the solution.

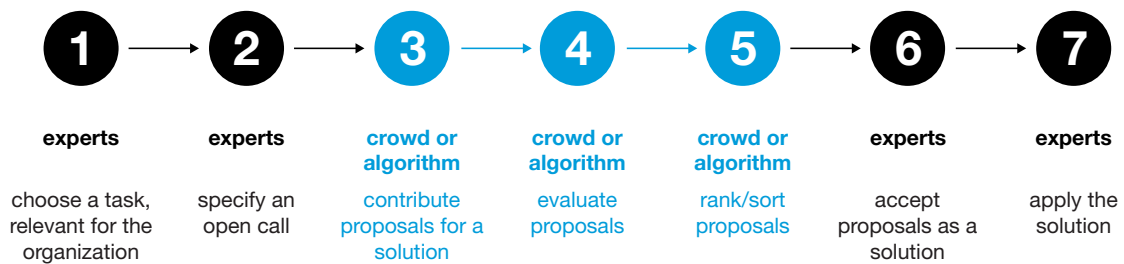


Figure 4.3 – Steps in a crowdsourcing framework. Image credits: the author.

In a non-crowdsourcing scenario, all steps are performed internally by the experts within the organization. In contrast, in a crowdsourcing scenario, some or all of steps 3 to 5 can take place online, publicly with the crowd's engagement. As *reCAPTCHA* shows, some of the steps from 3 to 5 can also be completed by an algorithm, making use of the human computation only for tasks that are intractable for an algorithm.

Threadless is another commonly cited example of crowdsourcing that illustrates how steps 3 to 5 are fully crowdsourced. *Threadless* is a T-Shirt company that runs an online competition for T-Shirt designs. Users submit and vote on the designs in a weekly cycle (Figure 4.4).

Lately, we've seen crowdsourcing be abused to reap value solely for the organizations using it, neglecting the motivations of the contributors and the value it brings to them. We see this departure from clear mutual benefit most clearly in *ReCaptcha*, which was one of the first crowdsourcing platforms and helped digitize 13 million books and articles, but now is solely used to force us to train AI algorithms for self-driving.

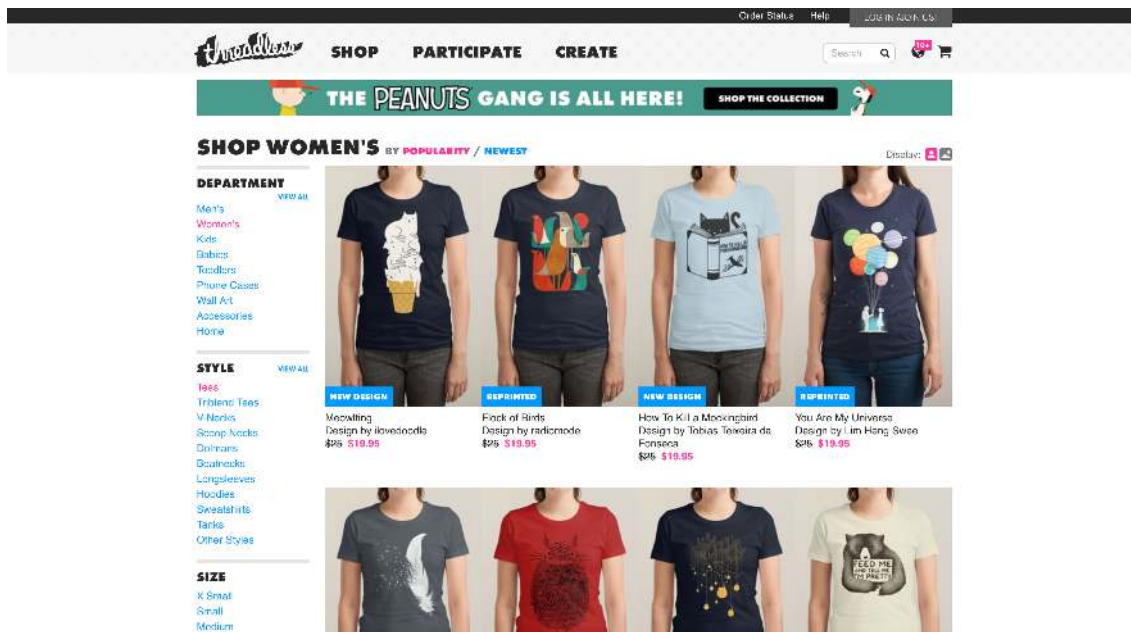


Figure 4.4 – Threadless. Image credits: Threadless.

This shift in the meaning of crowdsourcing is the reason why people now prefer the term *open innovation*. And in the context of science - citizen science.

4.1.2 Citizen science

In 2008, Chris Anderson proclaimed that the vast amounts of data available to scientists in recent years changed science from using models to define and test hypotheses to using statistical tools to find correlations in the data that explain the studied phenomena (C. Anderson 2008). However, this shift towards data-based science came with a new problem. Not all data can be collected automatically or processed with an algorithm. As Cook 2011 points out, the solution for both is to turn to the crowd:

“One reason for the sudden turn to crowd science is that it offers an imaginative answer to a central problem of 21st-century science: too much information.” (Cook 2011)

A precursor of the crowd’s involvement in gathering and processing data for science are the projects from the early days of the internet, such as SETI@Home (D. P. Anderson et al. 2002). SETI@Home did not require actions from users, just for them to share the computing power of their computers over the internet, but this created an opportunity to break down a computable problem into small distributable tasks (D. P. Anderson et al. 2002).

Citizen Science is a broad term with two primary meanings (C. B. Cooper and Lewenstein 2016; Eitzel et al. 2017). The phrase Citizen Science was used first by Alan Irwin in his 1995 book of the same name to denote *inclusion* (in the sense of broadening participation in science) as a way to align society’s interests and science’s responsibilities (Eitzel et al. 2017; Irwin 1995). This reading of science as being democratized is also known under terms like ‘activist science’ or ‘public engagement’ (C. B. Cooper and Lewenstein 2016). The second meaning emerged in 1996 from the work of ornithologist Rick Bonney and referred to projects where nonscientists contribute scientific data (Bonney 1996; C. B. Cooper and Lewenstein 2016). This second meaning of Citizen Science as contributory is what I use in my work.



Figure 4.5 – Galaxy Zoo. Image credits: Lintott et al. 2008.

Probably the Galaxy Zoo is the most often cited example of Citizen Science in this sense. The project was started in 2007 by Chris Lintott after a student of his manually completed the daunting task of categorizing 50,000 galaxy images only for them to realize they would need to classify a million galaxies (Cook 2011). The Galaxy Zoo was a website (Figure 4.5) where anyone, after completing a simple

tutorial, could classify photos of galaxies taken by the Hubble telescope (Lintott et al. 2008). Today the Galaxy Zoo has become the Zooniverse⁴ — a platform for citizen science and lists close to a hundred active projects in fields as diverse as arts, climate, and medicine.

The concept of *human-directed computing* narrows down further the aspect of Citizen Science I am interested in. It describes the method of using distributed human intelligence to carry out scientific tasks enhanced by algorithms in the data-collecting step as well as the data-processing step (S. Cooper, Khatib, et al. 2010). Seth Cooper used it when developing the well-known citizen science project Foldit.

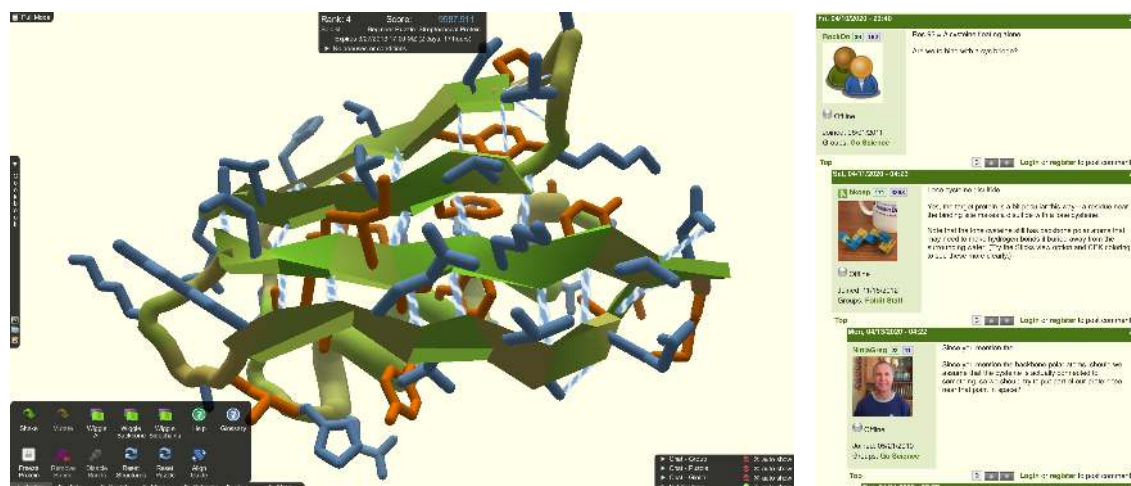


Figure 4.6 – Foldit. Left: the game interface; Right: a snapshot from a forum discussion. Image credits: Foldit.

In 2008, after years of failed attempts to develop an algorithm that can simulate the folding of protein molecules, a group of scientists at the University of Washington developed the video game *Foldit* (S. Cooper, Khatib, et al. 2010). The player is presented with a protein molecule, and their mission is to reshape it by connecting couples of atoms together (Figure 4.6 left). The player’s score increases as the energy required to keep the molecule in its current shape drops, meaning it is closer to nature’s solution. Nature decides who wins and who loses the game. Thousands of online players, driven by their desire to win and simultaneously help find proteins that could potentially cure deadly diseases, started to engage and understand the research aspects behind the game. They read scientific papers and shared their progress on the protein puzzles with each other (Figure 4.6 right). Foldit shows the potential of games to encode expert knowledge, provide real-time feedback and automate or outsource onboarding to the needed skill level so that anyone can contribute.

4.1.3 Mass collaboration

Wikipedia is the prime example of how you could combine the power of collaboration on a massive scale with a platform to create valuable content for everyone. To date, there are over six million articles written in English on the user-generated online encyclopedia, and it averages a monthly total of 7.6 billion views⁵. Growing and

⁴<https://www.zooniverse.org/projects>

⁵<https://en.wikipedia.org/wiki/Wikipedia:Statistics> — accessed 10.06.2021

maintaining the knowledge base of Wikipedia has evolved into a multilayered process with various roles (Figure 4.7) where mass collaboration is a leading concept. Other examples of mass collaboration are considered to be open-source software, Massive online open courses (MOOCs), crowdsourced science (Citizen Science), and even the maker movement that emerged around FabLabs (R. B. Shapiro 2016).

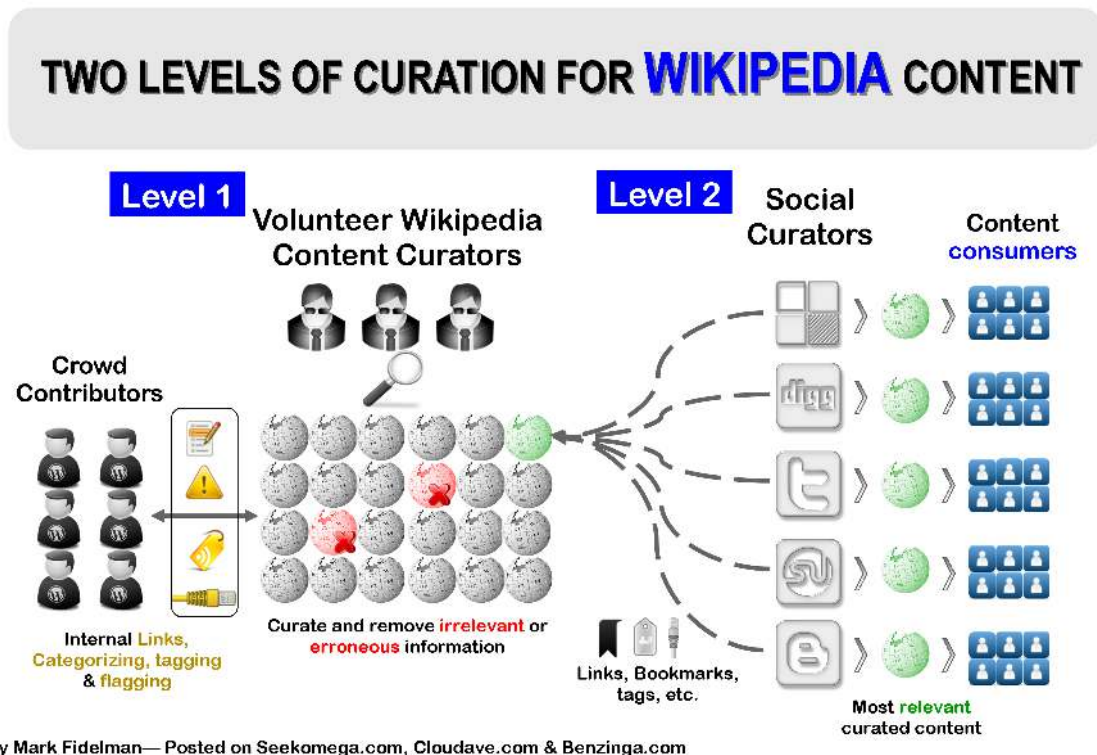


Figure 4.7 – Wikipedia Content curation. Infographic, Mark Fidelman, 2010, flickr, CC BY-NC 2.0: Link Image credits: Mark Fidelman.

Dillenbourg 1999 defines a situation of interaction between two or more people as *collaborative* if “peers are more or less at the same level, can perform the same actions, have a common goal and work together.” Furthermore, he distinguishes collaboration from cooperation by stating that cooperation is a division of labor on the task level, i.e., several people perform a set of tasks individually and then assemble the partial results in the final output.

In their book *Mass Collaboration and Education*, Cress, Jeong and Moskaliuk 2016b point out that in a formal aspect, mass collaboration “is characterized by the large number of people being (mass) involved in it, the digital tools they use (Web 2.0), and the digital products they create.”

The digital tools that make mass collaboration possible use the Internet as a communication channel and interactive web pages as the media where the production happens. These tools are characterized by their ability to make one user’s modifications immediately visible to the others and so allow users to observe others, share resources, coordinate work, and jointly create artifacts (Cress, Jeong and Moskaliuk 2016b). They also allow the storage and interaction with large amounts of data.

Unlike conventional digital tools where the user solves a problem they have, such as writing a text document they need or creating a 3D model they need, in mass

collaboration, participants create digital products that are of value not only for themselves but for other users as well.

Examples of products created in mass collaboration environments include texts with multiple authors such as the articles on Wikipedia, computer programs such as the Linux core or interactive projects on Scratch, labeled datasets such as the photo collection of galaxies categories via Galaxy Zoo, solutions to complex puzzles such as the folded protein molecules in Foldit, works of art such as the collaboratively painted canvas in Reddit’s r/Place.

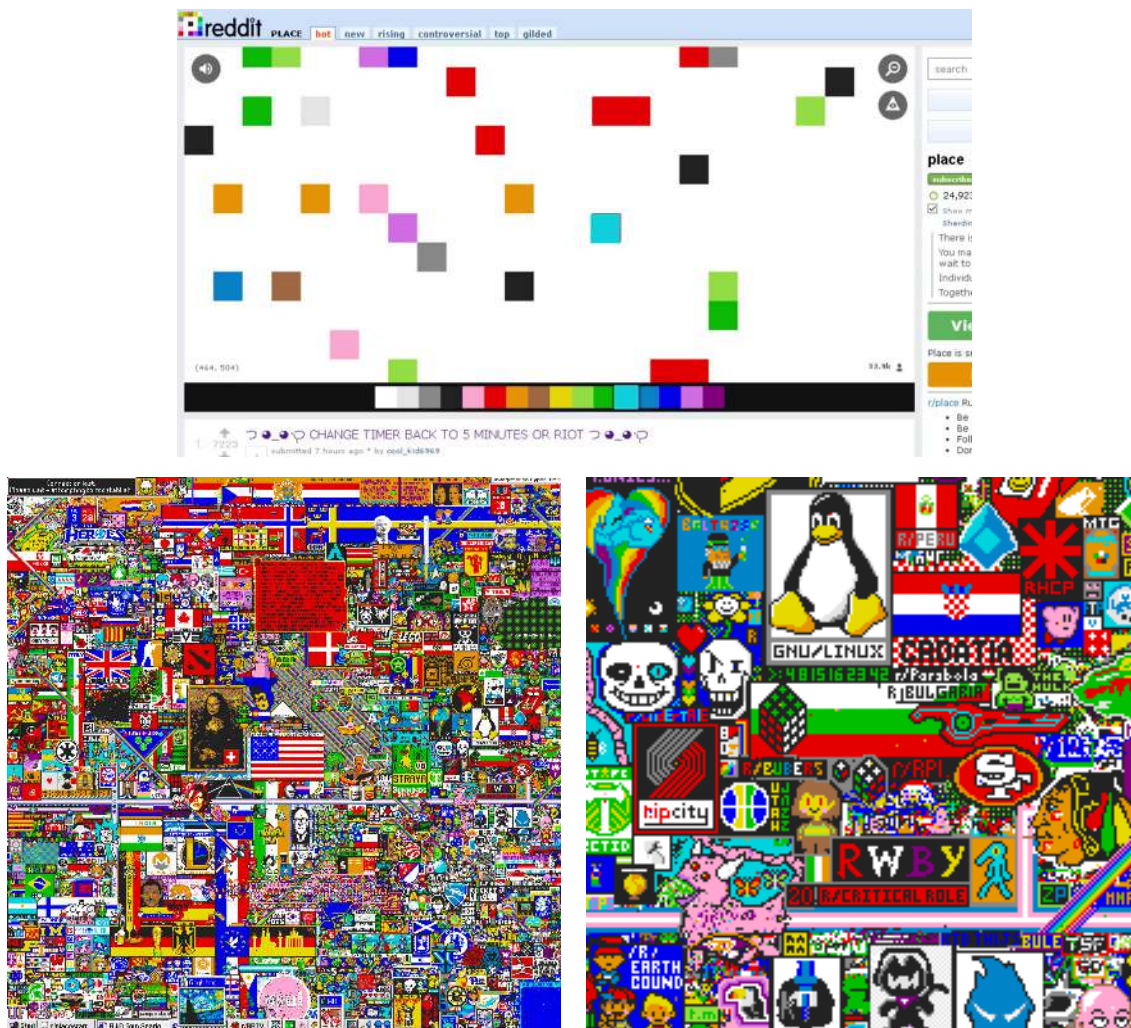


Figure 4.8 – Reddit r/Place. Top: A contributor's UI as they paint one pixel every five minutes; Bottom left: the final state of the whole 1000x1000 pixel canvas; Bottom right: a fragment of the final canvas state. Image credits: Reddit.

Reddit’s *r/Place* is another example of how tens of thousands of people can collaborate and compete to produce a collective work of art (Figure 4.8) (Rappaz et al. 2018). *Reddit* is an American social news aggregation, web content rating, and discussion website. Registered members submit content to the site, such as links, text posts, images, and videos. These are then voted up or down by other members. On 1 April 2016, *Reddit* put online a white canvas of 1000 by 1000 pixels and gave people 72 hours to paint something on it. Altogether, more than 250,000 people took part. Every participant had the same 16 colors to choose from and could paint only one pixel every five minutes. If you wanted to draw anything on the canvas, you

had to convince a large enough group of people to join your cause. Groups formed around nation flags, famous works of art, and symbols of internet culture but also more abstract ones such as *The Corner of Blue* or *The Green Lattice*⁶. People started Reddit communities (subreddits) to communicate and coordinate what they wanted to draw and how to defend it from being painted over by the other groups (Vachher et al. 2020).

Cress, Jeong and Moskaliuk 2016b note that even though a mass collaboration environment might have a large number of users, a specific product of collaboration, such as one *Wikipedia* article or one game programmed in *Scratch*, might be the result of only several users working together. As an example, Kittur and Kraut 2008 registered an average number of about 50 authors per article in the English Wikipedia. For comparison, Wikipedia has 130.000 monthly active users (MAU) and close to 40 million users in total as of 1.10.2020⁷.

An exciting mass collaboration project to do with architecture is *Build the Earth in Minecraft*. If we look at the *Build the Earth in Minecraft* project initiated by PhippenFTS, we can see how a goal can be achieved with collaborative work via a gaming platform. For the past few months, 200,000 players have been working together to recreate the Earth on a one-to-one scale within a Minecraft map (See Figure 4.10). It all started with a video shared on Youtube and Discord servers where task forces could be formed quickly followed (See Figure 4.9). This is an impressive achievement, but this project is quite simply a remodel! It lacks the ambition to imagine what the world could be, what humans could be, what the relationship between humans and nature could be. What architecture could be? But it shows that the willpower and the technical infrastructure are there to create something in a massively collaborative way.

A mass collaboration example from architecture is online forums where design proposals are discussed. This is a mix of both crowdwisdom and crowdproduction. Often next to the advice, contributors also provide alternative design solutions. A concrete example is the German *Hausbau-Forum* where individuals share the design for their single-family house, and forum members, experts, and other homeowners comment and propose solutions (Figure 4.11).

I carried out an analysis of the forum activity, which confirmed the finding by Cress, Jeong and Moskaliuk 2016b and Kittur and Kraut 2008 that only a tiny portion of the crowd's members are involved in a specific product of collaboration — in the case of hausbau-forum, a single forum thread (Figure 4.12). The analysis also shows that forum members can be classified in essentially three categories (Figure 4.13): (1) the super experts who often post both text and images; (2) the medium-level contributors, who have between 10 and 1000 contributions of which every tenth with an image and (3) the consumers, who tend to have very few posts, most likely about their own house.

Mass collaboration is an emerging and important topic in open source software development, science, and education (Cress, Jeong and Moskaliuk 2016a). So far, not many applications can be observed in other artistic, cultural, or business settings. Because of its newness, especially concerning the creative disciplines, I see the need for further research on the following topics:




1. theoretical consideration about mass collaboration in architecture,

⁶A descriptive video with time-lapse of Reddit's *r/Place*: YouTube link.

⁷<https://en.wikipedia.org/wiki/Special:Statistics>

BuildTheEarth interactive Map

On this map, you can see which builders are working on which project. Currently, we have **3361** ongoing build projects, which together cover about **5027** square kilometers.

-  Red markers indicate locations that users or build teams are already working on.
-  When you zoom in close enough, you will see blue squares. These demark claimable in-game Minecraft regions.
-  Orange squares indicate in-game regions that have been claimed by users or build teams.

Because of the large number of locations, not all markers are shown at the same time when you are zoomed out. Move around the map to load the markers in other areas. The location markers update every day.

Coordinates: Teleport command: `/tp 11 48.312428 13.579102`

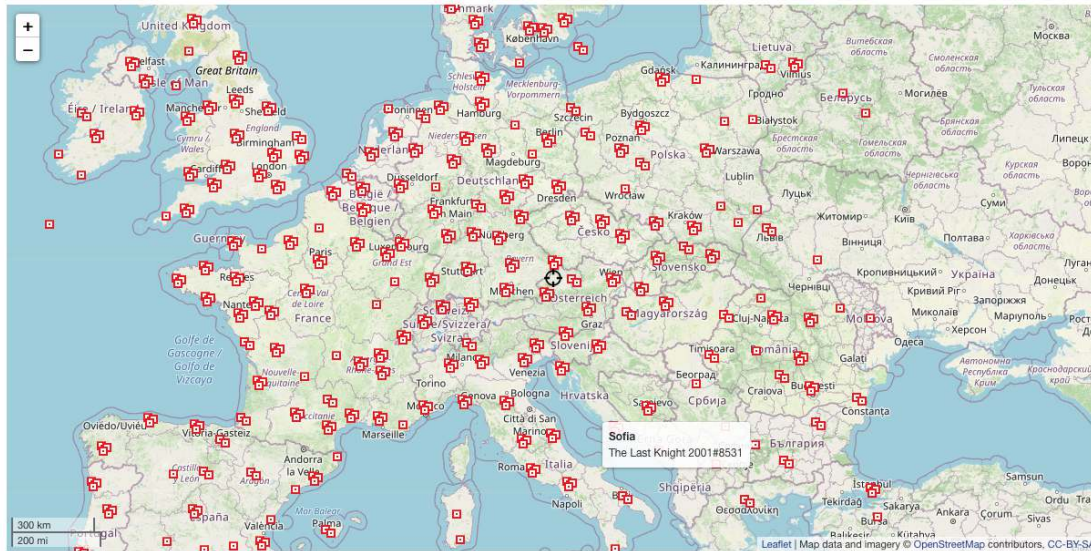


Figure 4.9 – BuildTheEarth interactive map. A segment of the world map showing part of Europe and the distributed remodel projects there, for the world 3361 in total as of 19.09.2020. Image credits: BuildTheEarth.

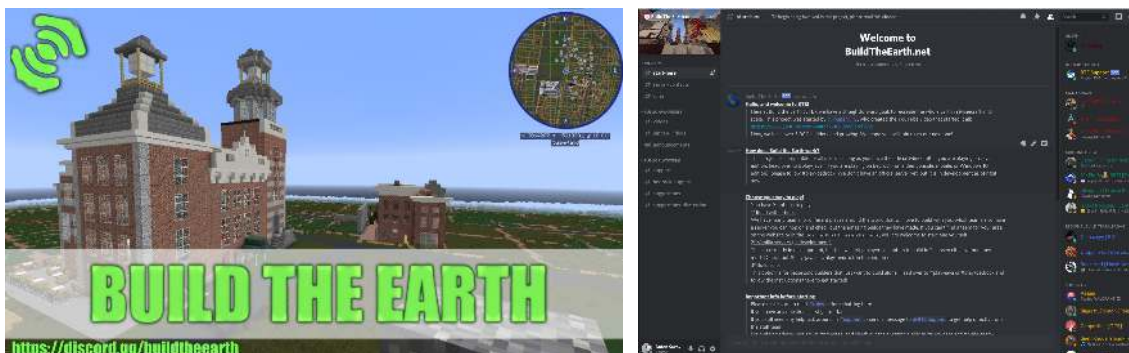


Figure 4.10 – BuildTheEarth in Minecraft. Left: an image, Right a screenshot from the discord server of the project used for communication and coordination of the project with 45,000 online members and 230.000 total members as of 19.09.2020 Image credits: BuildTheEarth.

2. description of individual cases of mass collaboration in architecture and
3. specification and development of mass collaborative digital tools suitable for design tasks.

In this dissertation, I mainly focus on the third.

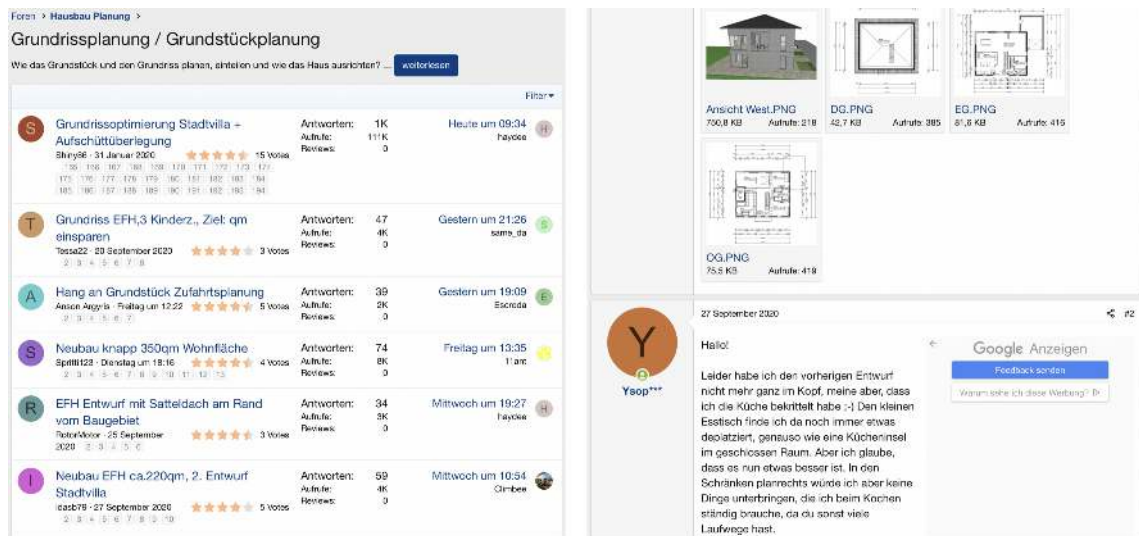


Figure 4.11 – Hausbau-forum. Image credits: www.hausbau-forum.de.

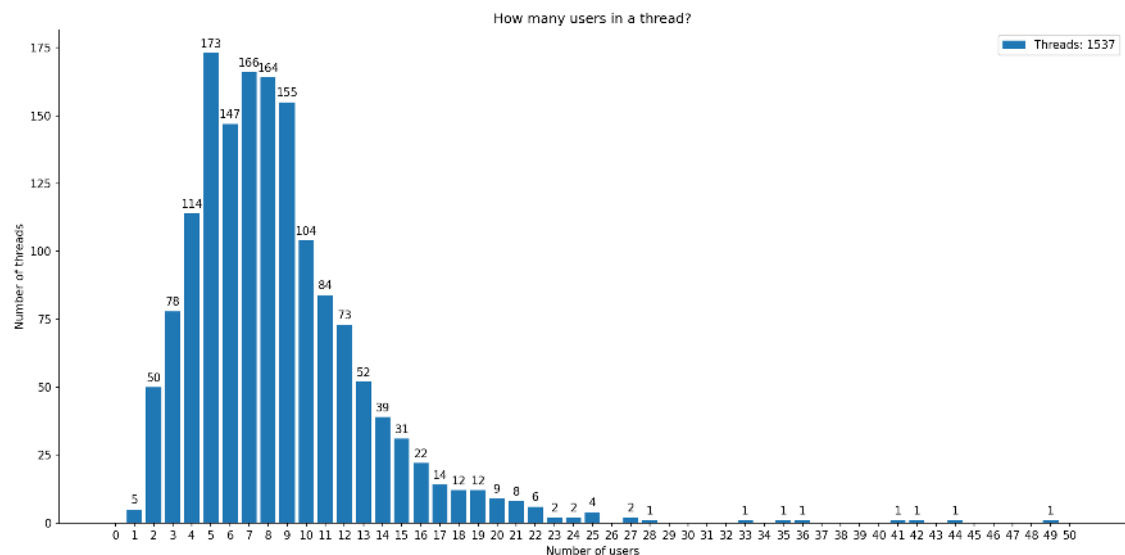


Figure 4.12 – Distribution of contributors per thread on hausbau-forum. Image credits: the author.

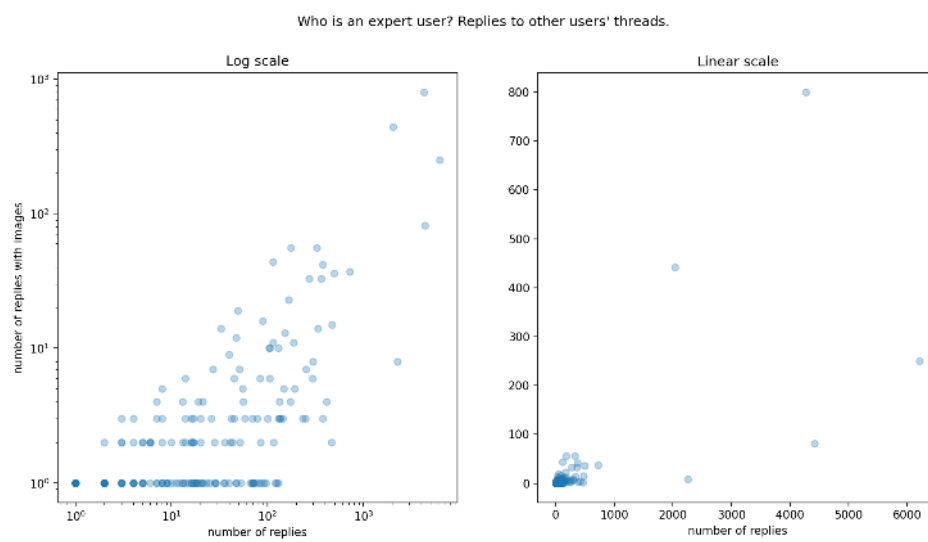


Figure 4.13 – Distribution of experts on hausbau-forum. Image credits: the author.

4.1.4 Self-organization

Working in the paradigms of crowdsourcing, mass collaboration, or citizen science means working with the idea of many agents contributing in parallel to the state of a system. That makes self-organization relevant. Manuel De Landa in his book *1000 Years of Non-Linear History* refers to three types of self-organization, “according to the type (or absence) of energy flow through the system: (a) conservative (crystallization, polymerization), (b) dispersive (solitons), and (c) dissipative (chemical clocks)” (De Landa 2000, p.277).

Self-organization is a mechanic to deal with any bias embedded in a system. On Wikipedia, a single-sided way of fact reporting or a contributor’s bias sorts itself out by the diversity and engagement of the community. If an article lacks a key point of view, someone might add it. If an article reports something false, someone might check and correct it. Oeberst et al. 2016 prove in three case studies that bias exists in Wikipedia articles. They found both individual bias in the form of hindsight bias as well as in-group bias, where people perceive and represent the group they belong to more favorably.

Regarding self-organization precedents in architecture, I would argue that the phenomenon of architecture without architects (Rudofsky 1964; Schaur 1991), or how informal settlements come to being is an example of stigmergic collaboration on a massive scale (Elliott 2016). Stigmergy is a mechanism that uses traces left in the environment for indirect coordination between agents (Marsh and Onof 2008). A classic example is ants’ chemical trails left to communicate directions and more to their group. In the case of informal settlements, we can argue that each successfully erected building is a trace that influences the following designers and builders. And we can confidently say that early society, even if they didn’t have a clearly defined role labeled as *the architect*, had construction experts that led and contributed to the collective architectural form built by the society.

4.1.5 Roles

Based on real-world stories, Eitzel et al. 2017 show how the “terminology used to describe participants can potentially change the way they are treated or how they feel about themselves and their participation in the activity.” Hutter et al. 2011 present a classification of the user groups found in a community (Figure 4.14).

An assembled group of people, treated as a group only by an organization, as the people filling out ReCaphas, is not a crowd in the complete sense of the term. The crowd members need to be aware of their belonging to the same higher level undertaking. There also needs to be peer-to-peer interaction.

The American journalist James Surowiecki lists five elements required to form a wise crowd in his book *The Wisdom of Crowds* (Surowiecki 2004):

1. Diversity of opinion, i.e., each person should have private information even if it’s just an eccentric interpretation of the known facts.
2. Independence, making sure that people’s opinions aren’t determined by the views of those around them.
3. Decentralization, i.e., people specializing and drawing on local knowledge.

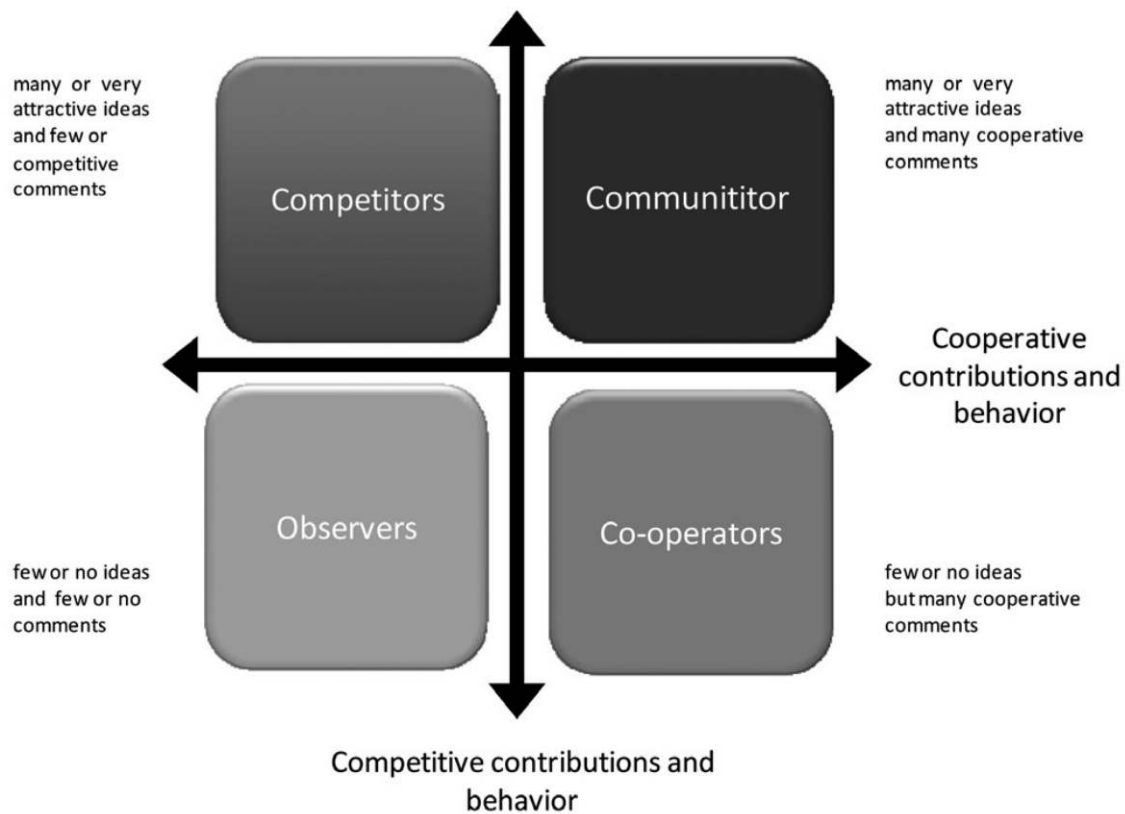


Figure 4.14 – Different User Types Identified in a Community. Image credits: Hutter et al. 2011.

4. Aggregation, provided by some mechanism for turning private judgments into a collective decision.
5. Trust, given by each person trusting the collective group to be fair.

G. Fischer 2016 describes the progression of different roles that can be found in rich ecologies of participation and collaboration, starting from unaware consumers to meta-designers (Figure 4.15).

In addition, Eimler et al. 2016 have observed weaknesses of massive online open courses that could be relevant for architectural projects as well. Namely, the high dropout rate and low level of participation. They speculate that combining small and large group interaction can increase engagement.

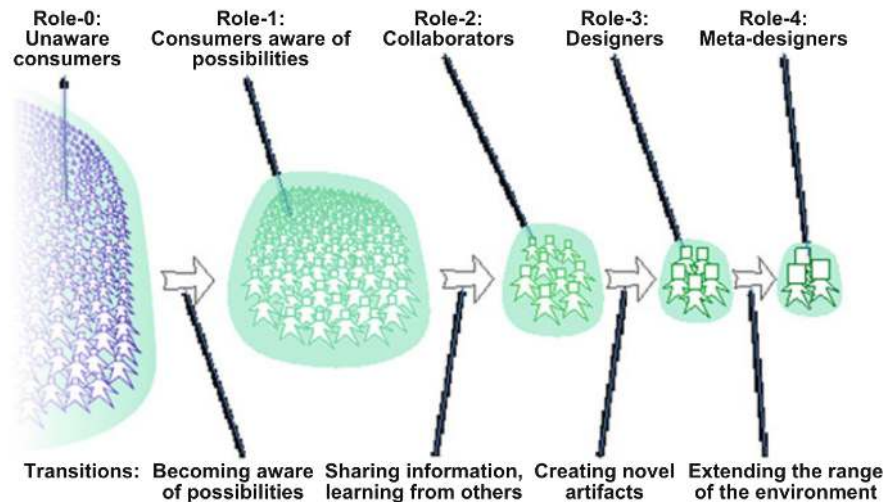


Figure 4.15 – Types of participants. G. Fischer 2016 identified different roles in rich ecologies of participation. Image credits: G. Fischer 2016.

4.1.6 Crowd wisdom vs. crowd stupidity

Unfortunately, crowdsourcing does not always result in wisdom. Sometimes we get crowd stupidity (de Mul 2016). Lanier 2011 argues that the “hive mind” can easily lead to mob rule, trolling, etc. Furthermore, Burnap et al. 2015 show that the expectation for contributions averaging out to a form of collective expertise within the crowd might not hold in engineering tasks. The authors propose that models of crowd consensus must follow the uneven distribution of expertise among contributors. However, their conclusions are based on a crowd simulation, not an actual crowdsourced study.

A design by committee⁸ often does not result in the best product (Lanier 2011). As Ratti and Claudel 2015 remind us, a horse designed by a committee is a camel. Ratti and Claudel 2015 state that in creative tasks, there has to be some sort of leadership. At the same time, it is a challenge to strike the right balance between the creative power of participants and the leading role.

4.2 Architectural Competitions as Crowdsourcing

Architectural competitions are the current form of open innovation in architecture, but they are under scrutiny as an exploitative practice, as the quote below shows. :

“... the evidence is building and the case becoming clearer: The competition industry in the U.S. is having equally as bad or worse effects on the conception of architecture than we already know it has on the business of architecture. The old argument that competitions drive architectural innovation is no longer credible. Developers, cultural institutions, and government agencies have mastered the use of design competitions as

⁸Design by committee is a pejorative term for a project that has many designers involved but no unifying plan or vision https://en.wikipedia.org/wiki/Design_by_committee.

publicity campaigns. Their claims of searching for the best ideas is just an alibi that unfortunately continues to seduce too many of our best talents. These drawn out exercises also make very little practical sense when it should be easy enough for clients to choose between architects... by picking up a few monographs or even just looking at their websites. The real justifications are simple. Developers and institutions gain fantastic and relatively affordable publicity from the mad traveling circus of design competitions. By helping them attract financing and donors, we encourage the proliferation of these sham exercises where enormous projects are fully rendered without contracts, necessary approvals, or even clear programs.” (Brown 2014)

The difference between architectural competitions and Threadless, for example, is in the number of times they go through the loop from an open call to a solution. An architectural competition gives out the brief, collects the designs, and chooses the winner. That is one iteration. In contrast, in Threadless, the brief doesn’t change — make the coolest T-Shirt graphic — and there are weekly iterations of the same brief. If one wants to participate, one can see the past weeks and what got voted high, what failed, get a sense of the trends from all the submitted entries and then begin designing.

Furthermore, there is no communication between the participants during the creative phase in architectural competitions.

This quote written for the field of science applies to architecture:

“Science is, for the most part, a closed society organized into little fiefdoms of highly trained specialists, which means only a few minds engage with any given problem. Before FoldIt, for example, a problem in protein folding was the exclusive province of a relatively small number of experts — even though, it is now clear, there are real contributions to be made by 13-year-old video gamers. The system is shaped in part by the force of tradition, but the larger challenge is that most scientific data is proprietary. A scientist works long and hard to generate original data, and then expects to reap the reward in the form of publishing the first research paper to describe some new phenomenon. She is not going to want to share this data with others, particularly strangers, any more than say, an investigative reporter would want to share his notes before a story has been written. Harnessing 1,000 people requires sending your data out into the world — something that science is loath to do. The scientist’s interest in keeping things private and getting credit, in other words, is directly opposed to society’s interest in tackling some problems with a hive of the best minds.” (Cook 2011)

The economic interest of each participant is to win the competition. So they don’t share. But that is a race to the bottom. A competition is crowdsourcing kind of. And *Arcbazar* is pushing that to the limit ⁹ - race to the bottom¹⁰. The falling prices and quality of services offered on platforms such as *Fiverr* and *Upwork* show

⁹<https://youtu.be/RzsnZJWrIkW>

¹⁰https://en.wikipedia.org/wiki/Race_to_the_bottom

how the whole gig economy, in general, is a race to the bottom ¹¹.

What if it was in the economic interest of every participant to get the best possible project? We must develop a business model for architecture that rewards architects when they share.

How smart is it for architects to every time invent the wheel? How many times do you walk into a building, and you think you've already seen that building on another site? But the architects of those buildings were not sharing information.

4.3 Conclusion

There are three axes of interest when looking at crowdsourcing models for crowd-creation. The precedents listed in this chapter are plotted on these three axes on Figure 4.16. The first one is the *beneficiary axis*. It follows the definition of (Brabham 2013) that the benefit of a crowdsourced task must benefit the organization that posts it as well as the contributors. An accurate crowdsourcing model will fall somewhere in the middle on that axis.

The second axis is the *mode of interaction* between the contributors. It can be predominantly competitive as in Threadless, primarily collaborative as in *BuildTheEarth*, or a mix of the two as in *r/Place*. The mixed model appears to have the advantage of establishing hierarchies that prevent the camel-horse-committee problem.

And the third axis is the *goal clarity*. It maps the precedents according to the nature of the tackled problems. At the one end, we find tame problems that can be chopped into microtasks. At this end, the goal for individual contribution is apparent, and progress is measurable. On the other end, we have wicked problems where stakeholders need to shape both the problem definition and the solution criteria. An example is a *Wikipedia* article's aim to encompass all points of view. Another example is the discussion relating to the design of someone's house on the *Hausbau* forum. At this end of the axis, the goal for individual contribution is not clear, and progress is therefore not measurable.

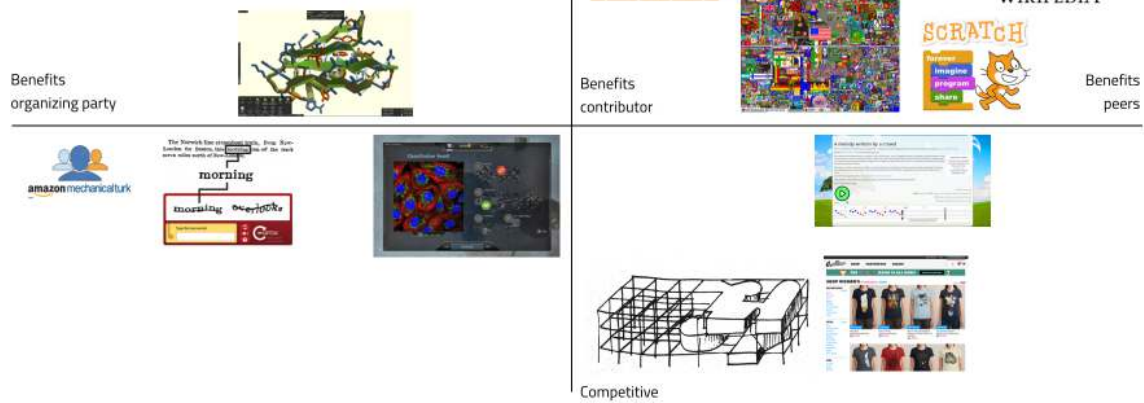
From the precedents in this chapter, we can observe that there are at least one of the following features present in an online crowd wisdom platform:

1. coordination among participants is made non-optional either by limiting a resource (as in the case of Reddit *r/Place*) or by having a process (as in the case of Wikipedia contributing chains)
2. each contribution is evaluated or validated, either by a community (as in voting on *Threadless*) or by an algorithm as in *ReCaptcha*,
3. the number of contributing users on a given challenge is small in relation to the number of total users,
4. rarely there are flat hierarchies, so they are avoiding the camel-horse-committee problem.

The ease of use of the provided tools and the ability for users to appropriate them for purposes not yet imagined by the developer is vital in most crowdcreation examples (Gürsimsek 2012; R. B. Shapiro 2016).

¹¹<https://news.ycombinator.com/item?id=19932360> and <https://news.ycombinator.com/item?id=17894111>

Beneficiary vs. mode of interaction



Beneficiary vs Goal clarity

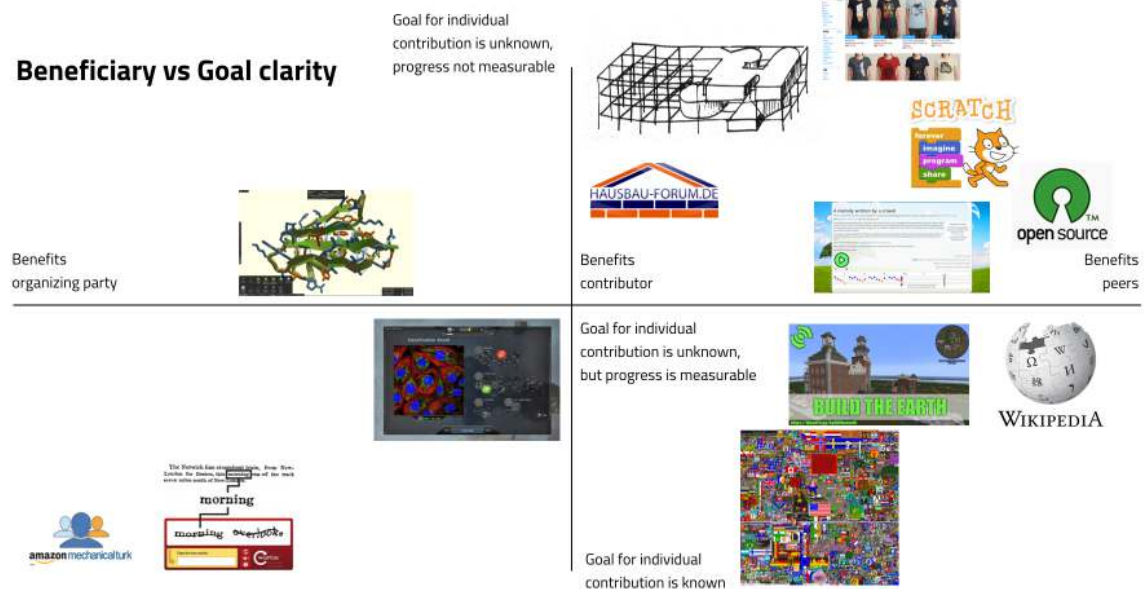


Figure 4.16 – Crowd wisdom precedents. Mapped on the beneficiary axis vs. the mode of interaction axis (top) and on the beneficiary axis vs. the goal clarity axis (bottom) Image credits: the author.

ReCAPTCHA and *Foldit* represent two strategies to achieve the engagement of the crowd. *reCAPTCHA*'s purpose is online legitimization — book digitizing is just a byproduct. *Foldit* engages the players directly with the problem of folding protein chains and makes it more accessible and fun by embedding it in a game. The first one is to hook human computation as a byproduct of an often repeated action of immediate value, such as blocking spambots on the Internet. The other is, through gamification, to create an experience for the participants that is entertaining on its own. We need to distinguish between crowdsourcing meant as massive outsourcing,

which is when we ask a lot of people to address the problem at hand directly, and crowdsourcing in the meaning of devising a byproduct-based problem-solving system. The second strategy might be more suitable for architectural problems because of the complexity of the problems at hand in architectural design.

Sustaining the high volumes of participants needed for successful crowdsourcing requires an engaging and rewarding experience for every participant with the help of game design. There are several precedents for the gamifying of architecture. In Martin Wynn's examples, a game is developed to understand the stakeholders' interests (Wynn 1985). Hence, the roles in the game strictly replicate those present in a real-estate development and include investor, city regulator, neighbor, etc. Similarly, UN-Habitat and Mojang used gaming environments such as Minecraft to visualize designs for public squares in Kenya and include the local communities in the process (BBC News 2012). Most recently, Jose Sanchez has used single-player gaming environments as a tool for architects, allowing the exploration of an open-ended design search-space (Sanchez 2013). The examples of gamification, as means of user engagement, described above, are not focusing on a specific architectural typology, the use of robotics or the use of massive human computing and are, therefore, distinguishable from the method I am proposing to develop.

Forms of collaboration fit the architectural design process, while cooperation, or parallel tasking, do not. This relates to the nature of architectural design problems being wicked, i.e., it is difficult to formulate a step-by-step solution to them, hence not possible to split the work into sub-tasks.

Crowdsourcing bears one significant distinction to the architectural precedents that sourced input from future inhabitants to produce a design. *Participatory Design* employs a one-to-one relationship between the participant and architectural output. In other words, one future inhabitant enters information for one residential unit, most likely theirs, and as such, influences the overall design. *Foldit* and *reCAPTCHA* establish a ratio of thousands, even millions, to one. One book is digitized by hundreds of millions of users, each of them typing only a word. In the same way, the same molecule is folded virtually tens of thousands of times, every time by a different player.

Design as a process is an iteration between creation and evaluation until a time limit, or another resource limit of the designer is reached. Possible approaches for engaging the crowd in this two-step iterative model are three.

1. the crowd co-creates architectural designs — The crowd contributes to the creation/generation of designs that get automatically evaluated by an algorithm. The algorithmic feedback informs the crowd to change their creation strategy.
2. the crowd evaluates architectural design — The designs are algorithmically generated and presented to the crowd, who then evaluate them. This approach is probably more suitable for the evaluation of the subjective aspects or in Elezkurtaj's definition of an ill-defined problem (Elezkurtaj and Franck 2002).
3. full crowd engagement - co-creation plus evaluation — The design creation step and the analysis step are both open-end for input to the crowd. The challenge here is finding where and how to plugin the computational power so that it contributes and amplifies the crowd's input.

We also need to look at what could be crowdsourced in practice. Architectural design as a service solves many problems and is multi-faceted. The aspect of design explored in my dissertation is that of organizing the spaces of a building, usually called *space allocation*. It is the task performed by an architect at the very early phase of design. It consists of laying out the rough proportions of the building and the relative position and orientation of rooms, zones, and openings.

Chapter 5

Game Design: Game Mechanics and Gamification

“You can look at games as very simple simulated learning environments. They constantly tell you how well you are doing by how well you score. The more you learn the underlying principles, the better you score.”.
(Steve Jobs 1990)

The quote by Steve Jobs above hints at what *game mechanics* are and the vast opportunities they open for both game designers and players. In the game of Simcity (Figure 5.1), for example, at any given moment, you have a city, however small that might be at the beginning. You start with a house or two, but it’s a functioning simulated city. Then you build more, you level up, and add more advanced buildings. You end up developing whole industries.



Figure 5.1 – Simcity. The game Simcity 5, as well as all previous version in the series, present the player with a challenge to run a city. The game simulates the city based on encoded principles of how economy, population and industries influence the growth of a city. The better the player learns those principles the higher score they can achieve. Image credits: Electronic Arts.

In my research, I am primarily interested in games for their iterative and interactive nature, which can automate learning and behavior guidance. Games achieve this by staging challenges and experiences for the players. This is the subject of game design. A secondary interest to the ideas presented in this dissertation are games and game technologies as human-computer interfaces, games as representation medium, and games as a gathering place.

5.1 What is a Game?

Picture puzzles and the Rubik's cube are examples of toys that rely on the player's spatial reasoning. They are not games. On the other hand, Tetris is a game that uses the player's spatial reasoning by giving them a challenge and measuring their progress. Jenga is an example of a game where players compete based on their intuitive understanding of structural statics.

A game must be fun to play, and fun in a game is hard to achieve (Koster and W. Wright 2013). In Juul 2005, Jesper Juul, a Danish game designer and an associate professor at the Danish Design School, defines a game very specifically to distinguish it from toys, play, and other related phenomena.

“A game is a rule-based formal system with a variable and quantifiable outcome, where different outcomes are assigned different values, the player exerts effort to influence the outcome, the player feels attached to the outcome, and the consequences of the activity are optional and negotiable.” (Juul 2005)

Jesse Schell, a video game designer and a Professor Carnegie Mellon University's Entertainment Technology Center, offers a more concise and generic definition:

“A game is a problem-solving activity, approached with a playful attitude” (Schell 2008, p.47).

Before arriving at this definition, Schell 2008 reviews multiple reports from the literature and distills a list of 10 qualities that make a game a game: Games are entered willfully; Games have goals; Games have conflict; Games have rules; Games can be won and lost; Games are interactive; Games have a challenge; Games can create their internal value; Games engage players; Games are closed, formal systems.

5.2 Why Games?

There is a broad range of reasons, within and beyond these qualities, for the interest of architects in games and of game designers in architecture: real-time, interactive, spatial (2D or 3D), easily accessible and ubiquitous, simulations, learning environments, story-telling, rule-based, a mix of control and uncertainty, multi-player, gathering places for collective exchange and experiences, representational medium, capacity to span between real and virtual space, an algorithmic agency in opposition or assistance to human player agency, create and communicate alternate realities, role-playing, immersive, etc. (Borries 2007; Oosterhuis 2001; Oosterhuis and Feireiss 2006; Oosterhuis, Hubers, et al. 2003; Walz 2010).

Oosterhuis 2001 points to real-time interactiveness both online and in the actual building via sensors as a way to stage architecture as a game that architects design and inhabitants play. The need for rich and diverse architectural environments in computer games and the aim to save production costs have pushed the development of generative design techniques, aka procedural content generation (See chapter 3). Liapis, Yannakakis and Togelius 2014 position computer games “as the ideal application domain for computational creativity for the unique features they offer: being highly interactive, dynamic, and content-intensive software applications”.

J. T. Wunderlich and J. J. Wunderlich 2011 describe the use of the game Minecraft for architecture. The authors present 21 case studies that “show that Minecraft can be used for the creation of architectures and towns, where group harmony and sustainability can be achieved collectively and individually through CrowdSourcing” (J. T. Wunderlich and J. J. Wunderlich 2011).

Games are of interest to my research for two main reasons:

1. First, for their *player base*, i.e., lots of people already in a community with similar in-game skills.
2. Second, the *game mechanics* that game design employs to challenge, guide, and educate a player into completing a mission.

5.2.1 Player base as human computation resource

Project Discovery (Figure 5.2) is a fitting example from Citizen science for the first reason. The project benefits from access to the players of a popular game by offering them virtual in-game incentives for completing a task that has otherwise no link or meaning in the main game.

Project Discovery is built within the game *EVE Online*. *EVE online* is a mass multiplayer online (MMO) space exploration game with very tight-knit and devoted player communities. As an example of this, in 2014, a 21-hour battle ensued involving 7,548 gamers, and spaceships valued at 330,000USD were destroyed (Moore 2014).

When it launched in 2016, it helped scientists categorize the human protein atlas by offering a special mission to the players of *Eve-Online*. Players carried out over 4.5 million classifications in the first two weeks of April 2016. In total, 44 thousand players processed 183.165 microscope photos of protein molecules with the required redundancy to reach a consensus (Peplow 2016). An algorithm cannot do this on its own due to the photos’ complexity, making them hard to analyze with computer vision.

Each player was rewarded with a badge and in-game currency, thus embedding the *Project Discovery* mission in the game economy but not in the gameplay. The project is currently in its third phase and engages the players to help understand the novel coronavirus causing COVID-19 and find a vaccine against it.

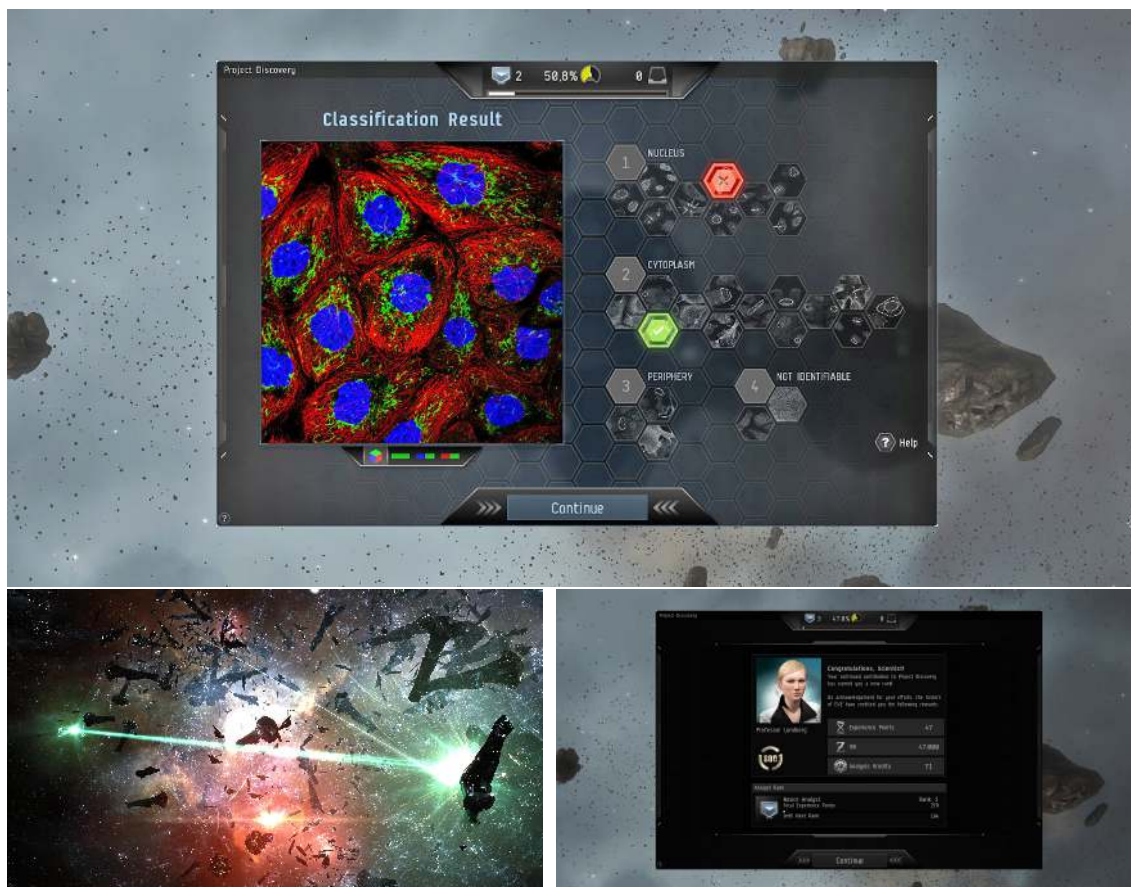


Figure 5.2 – Project Discovery. Bottom left: battle sample from the real game of *EVE Online* where *Project Discovery* unfolds. Top and bottom right: the mission screens of *Project Discovery*. Image credits: CCP.

5.2.2 Game mechanics as automation of guidance and education

A game mechanic is a mechanism with which players are allowed to affect the state of the game environment to overcome a challenge or with which the game poses a challenge and so triggers the players to act (Hunicke, LeBlanc and Zubek 2004; Järvinen 2008; Sicart 2008) (See also *Appendix A: Interview with the Game Designer Ben Buckton*). An example of the former is the mining in Minecraft, where the player hits blocks with a tool to remove them from the environment and collect them as materials. An example of the latter is the day and night cycle in Minecraft, which forces the player to build shelter or craft weapons as monsters appear in the night.

Miguel Sicart, a Professor in game design at the IT University of Copenhagen, states that “for any agent in a game, the mechanics is everything that affords agency in the game world” (Sicart 2008). In his Ph.D., the game designer Aki Järvinen defines mechanics as “means to guide the player into particular behavior by constraining the space of possible plans to attain goals” (Järvinen 2008, p.254).

Game mechanics are not the same as game rules. “Game mechanics are concerned with the actual interaction with the game state, while rules provide the possibility space where that interaction is possible, regulating as well the transition between states” (Sicart 2008). “Rules are normative, while mechanics are performative” (Sicart 2008).

Game designers are the architects of behavior, challenges, and experiences within a game. They make the rules and set out the goals for players and also are responsible for the game agency that the player would perceive as an opposing force (See *Appendix A: Interview with the Game Designer Ben Buckton*).

The use of game mechanics and game design to create full-fledged games with purposes other than entertainment is labeled *Serious Games* (Jefferies 2018; Strahringer et al. 2017). Usually, the fun factor is used as a means for motivating people to achieve goals relating to health, training, or social networks (Backlund et al. 2007; Jefferies 2018; Knöll 2018; *Nintendo WiiFit* 2015; Schreiner 2008; Strahringer et al. 2017). An example of a serious game application in science is *Foldit*.

Examples of the use of games in this full-fledged capacity in architecture are the game Block’hood by Jose Sanchez (Sanchez 2015) where one of the mechanics is to place architectural program blocks to keep resources in balance and Space Fighter by MVRDV (Maas 2007) which is a collection of mini-games with focus on the real estate market exploring the main mechanic of *location* (absolute and relative) and land as a scarce resource (Figure 5.3). They could be considered Serious Games for their educational qualities. However, from the perspective of the production of architectural design, they are not aiming to create designs for a specific project site and brief.



Figure 5.3 – Architectural games. Top: Spacefighter by MVRDV, Bottom: Block'hood by Jose Sanchez Image credits: MVRDV, Plethora-Project.

5.2.3 Game Design vs. Gamification and Gamefulness

It is important to note that the often circulating term *gamification* is not a notion connected with games as they are seen in my work. Gamification is using game design elements in non-game contexts to motivate and increase user activity and retention usually employed in interaction design and digital marketing (Deterding et al. 2011; Nacke and Deterding 2017; Zichermann and Cunningham 2011). In general, it means the use of game-like progression of rewards to manipulate user behavior by giving points for doing the 'right' thing (See *Appendix A: Interview with the Game Designer Ben Buckton*).

Examples of gamification, shown on Figure 5.4, are the reputation system for user contributions on the forum for programming questions Stack Overflow (Movshovitz-Attias et al. 2013; Papoutsoglou, Kapitsaki and Angelis 2020), the mayorships and badges awarded to people for checking into physical locations in Foursquare (Frith 2013) and the activity rings and challenges of the Apple Watch (Zhao et al. 2016).

Academia, as well industry, have criticized gamification, and the term has gained a bad connotation (Deterding et al. 2011). Even forerunner of gamification like

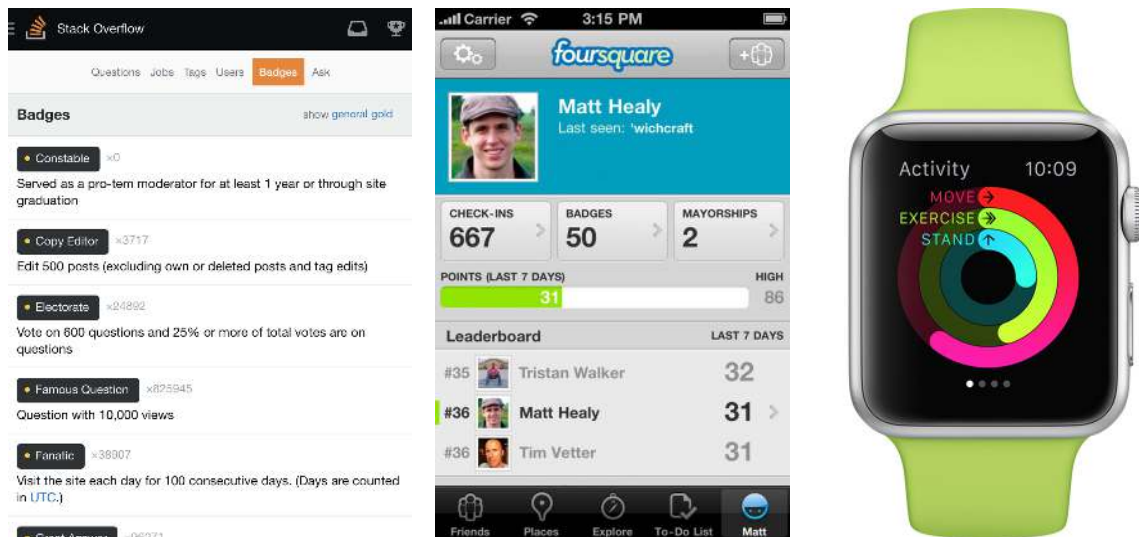


Figure 5.4 – Gamification examples. Left: *Stack Overflow* badges assigned to forum contributors, Middle: *Foursquare* gave points out for frequent checkins in places (2009), Right: An *Apple Watch* showing the activity rings. Badges are for longest streaks of closing them. Image credits: Stack Overflow, Foursquare, Apple, Inc.

Foursquare has moved away from these practices (Wilken 2016). In 2014, Foursquare packed some of the gamification features in a new companion app called Swarm and refocused the original Foursquare app on local search, i.e., a city guide. Nevertheless, game elements in a non-game context are still a research area with high potential, creating the need for a new term without the baggage. Deterding et al. 2011 make the case for *gamefulness*, introduced by McGonigal 2011 and seemingly accepted in this field of research (Bell 2018) (Figure 5.5). *Gamefulness* is complementary but different to *playfulness*.

The distinction between gamified processes and full-fledged games with serious purpose can be very subtle (Seaborn and Fels 2015; UpsideLearning 2015, p.27), and is explored in the case study Sensitive Assembly (chapter 7), which offer two modes of play. The first aimed to gamify the activity of building a predefined wall design, and the other, a game similar to *Jenga*, produced topologically optimized structures as a byproduct of the game.

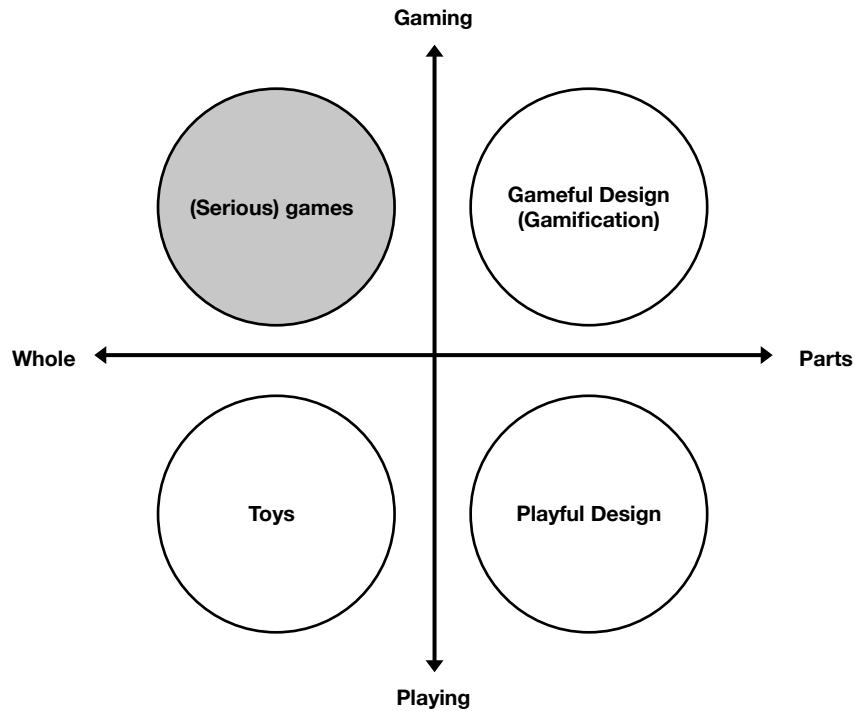


Figure 5.5 – Game, Gamification or Toy. The relative positioning of full-fledged Games (including Serious Games), Gamefulness/Gamification, Playfulness and Toys according to Deterding et al. 2011. The focus in my work is on the wholistic application of gaming, i.e. Games. The horizontal axis represents the spectrum from full-fledged game to a single game design element (avatars, narrative, feedback, reputation, economies, competition under explicit rules, time pressure, etc.). For a more exhaustive list of game design elements see (Reeves and Read 2009). The vertical axis represents the spectrum from unregulated, open play to goal-based gaming. Image credits: Deterding et al. 2011.

5.3 Conclusion

What makes games interesting for this research is their players and mechanics. Algorithmic simulation of spatial reasoning and the classification or arrangement of complex shapes is what humans excel at, while algorithms still fail to deliver relevant results.

Exploring both research questions — roles, tasks, and tools as well as design paradigms — requires going beyond gamification, i.e., making the existing design process and tools more engaging. The goal in the case studies presented in the following chapters is to employ game design and games as a new holistic design environment — one where people focus on achieving a game goal and, as a byproduct, create design solutions.

Chapter 6

Four Fields and Eleven Intersections

The Venn Diagram on Figure 6.1 shows the four fields presented in the last four chapters: *Participatory Design*, *Generative Design Techniques*, *Game Design* and *Crowd Wisdom*. The two ellipses on the left represent the fields of *human agency*. The two ellipses on the right represent the fields of *machine agency* or *automation*. Figure 6.2 enables the exploration of the first research question by presenting the design paradigms that emerge in the 11 intersections between these four fields. The state-of-the-art precedents, as well as my case studies, to be found in each intersection are mapped on Figure 6.3. Each case study offers an entry to explore the roles, tools, and tasks that concern the second research question. The lack of precedents in six intersections clearly outlined the research gap.

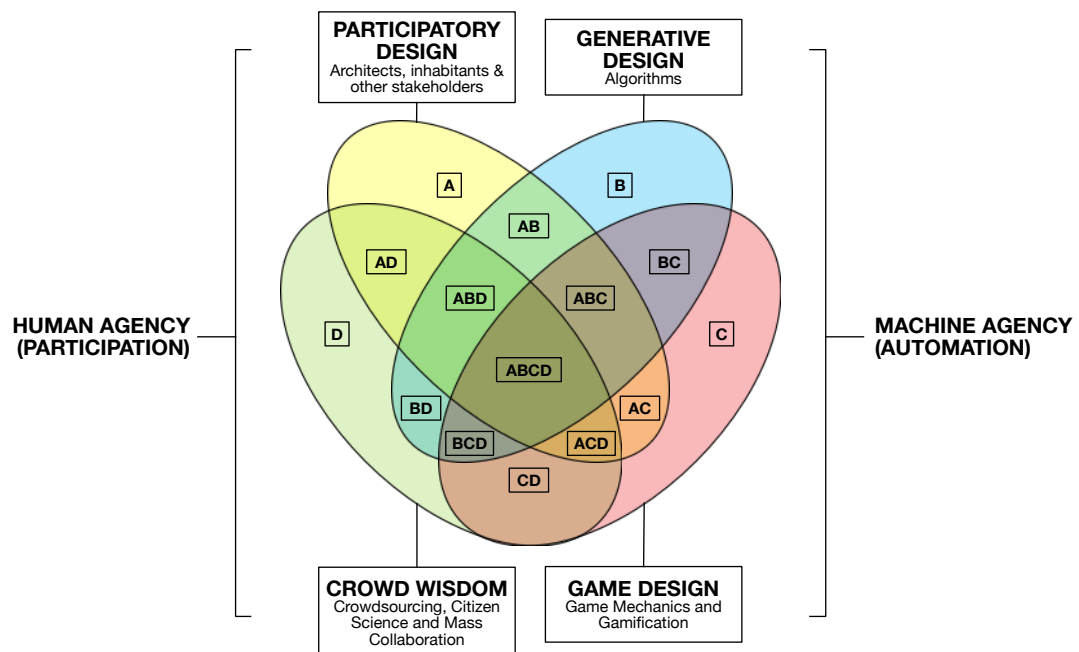


Figure 6.1 – Intersections of the four fields. Image credits: the author.

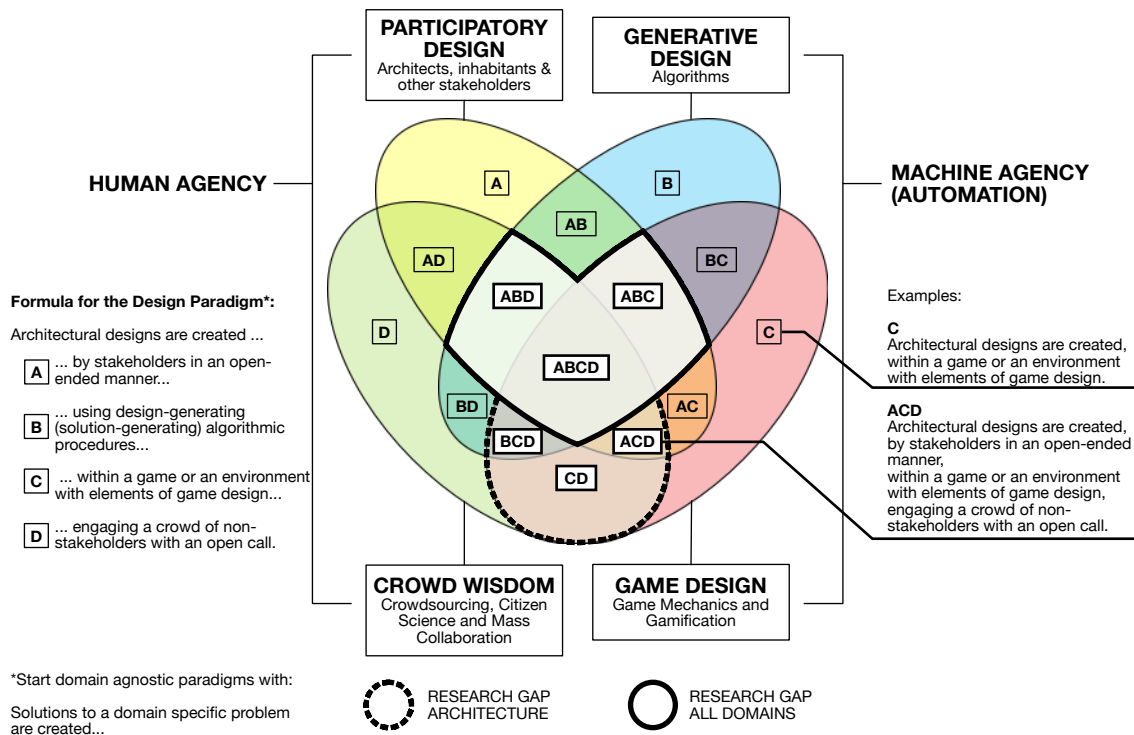


Figure 6.2 – Map of Design Paradigms. The design paradigms at the intersections of the four fields, their descriptions and the identified research gap. Image credits: the author.

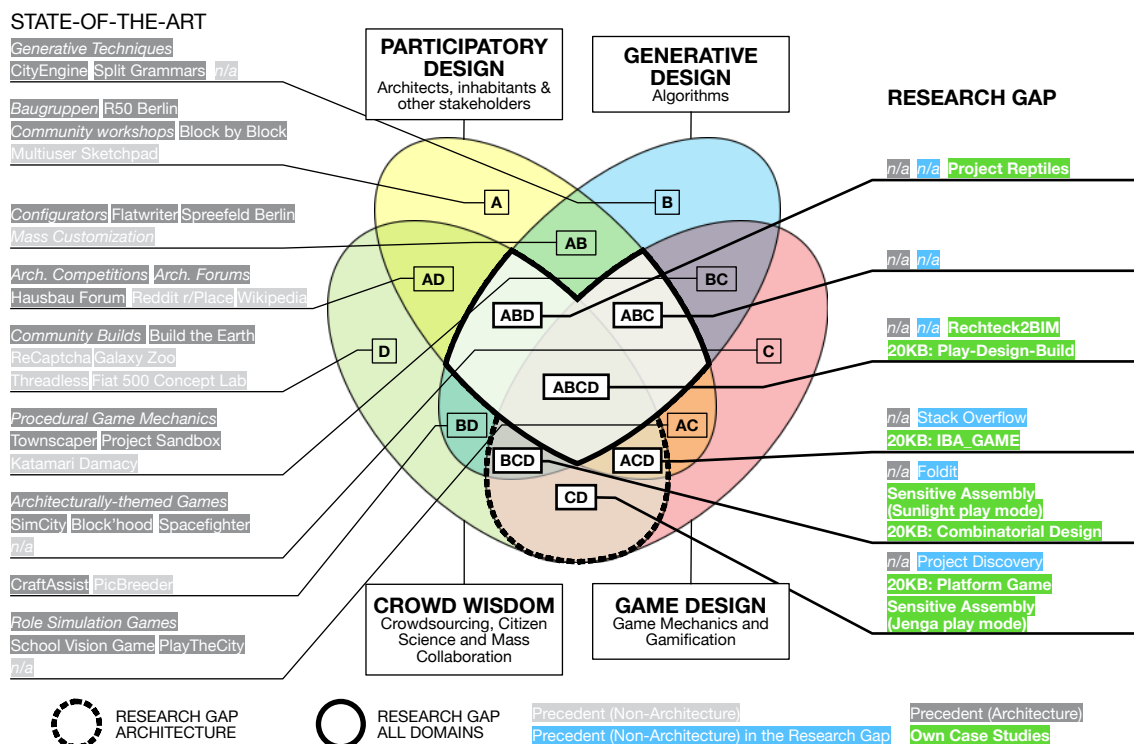


Figure 6.3 – Precedents and own case studies. The projects to be found in each respective design paradigm at the intersections of the four fields. The identified research gap is marked in thick black line. Image credits: the author.

6.1 The Four Fields

In *Participatory Design* (marked A on Figure 6.2) we have stakeholders creating designs in an open-ended manner with examples such as cooperatives like the Berlin *Baugruppe R50* (See chapter 2). Another example is *Block by Block*. Even though *Block by Block* uses the game Minecraft, it does not use game design elements but simply the game technology for a meeting and modeling space. Hence it is considered a pure participatory project.

In *Generative Design* (B), we have architectural designs created using algorithmic descriptions of design procedures. There are a plethora of state-of-the-art examples for this category, such as the *Split Grammars* used to generate buildings and cities in *CityEngine* (See chapter 3).

In *Game Design* (C), we have a game or an environment with elements of game design with the purpose of creating an architectural design. There are plenty of games with architectural topic such as *SimCity*, *Block'hood*, *Spacefighter* (See chapter 5).

And, fourth, in *Crowd Wisdom* (D), we would have an organized crowd of non-stakeholders creating architectural designs in an open-ended manner. An example here would be Community Builds such as the impressive 200,000 players project *Build the Earth in Minecraft* (See chapter 4). Similar to *Block by Block*, *Build the Earth in Minecraft* does not use any game design elements, but rather the open game world of Minecraft as a technology allowing easy modeling, collaboration, and communication. So it is a pure mass collaboration project.

In summary, we have precedents for all four fields in their pure state.

6.2 Six Double Intersections

At the intersection of *Participatory Design* and *Generative Design* (AB), we have configurators of all kinds where stakeholders create architectural designs using algorithmic descriptions of design procedures. Examples here abound — Yona Friedman's *Flatwriter* or the model box by BARArchitekten for the project *Spreefeld Berlin* (Figure 2.7). From non-architectural domains, one can look at many mass-customized products for a reference (Pine 1993).

Between *Generative Design* and *Game Design* (BC), we find games or environments with elements of game design with the purpose of creating architectural designs using algorithmic descriptions of design procedures. An example would be the game *Townscaper* by Oscar Stalberg as well as the game *Katamari Damacy* (??).

Mixing *Participatory Design* with *Game Design* (AC), we get stakeholders creating architectural designs within a game or an environment with elements of game design. All role-playing simulations such as the projects by Ekim Tan's *Play the City* (Figure 2.8) or Baupiloten's *School Vision Game* (Figure 2.11) are examples of this category.

At the crossing of *Game Design* and *Crowd Wisdom* (CD), we get an organized crowd of non-stakeholders creating architectural designs within a game or an environment with elements of game design. There are no architectural precedents in this category. However, I have identified examples from other disciplines by generalizing the category's description. By substituting 'create architectural designs' with



Figure 6.4 – Examples of Generative Design and Game Design. Left: In *Townscaper* by Oskar Stalberg the player creates mediaval towns by adding or removing coloured blocks. A generative algorithm take care that the each new town element has contextually correct shape. The algorithm is a combination of Wave Function Collapse and Marching cubes similar to the Model Synthesis approach presented by P. Merrell 2007. Right: *Katamari Damacy* is a game with a procedural game mechanic. In the game the player needs to roll a ball, which like a snowball, absorbs objects from the environment that fit the current size of the ball. As the player progresses and the ball gets bigger it can absorb larger and larger objects. Image credits: Stalberg; Namco.

'solve a domain-specific problem', the description becomes 'an organized crowd of non-stakeholders solving a domain-specific problem within a game or an environment with elements of game design'. *Project Discovery* is an example of this. I present two case studies here. The early explorations with 20.000 BLOCKS, under the title *The Platform Game* as well as the project *Sensitive Assembly* in its *Jenga mode*.

At the intersection of *Generative Design* and *Crowd Wisdom* (BD), we find an organized crowd of non-stakeholders creating architectural designs using algorithmic descriptions of design procedures. The category is being explored only as of recently, with projects such as *CraftAssist* (Figure 6.5). In *CraftAssist*, a recently published project by the A.I. group at Facebook, researchers gave Minecraft players the simple task to "build a house of 4x4 blocks" and recorded 2500 unique designs (See Figure 6.5) (Gray et al. 2019; Srinet et al. 2020). Designing a house is an extremely wicked problem. Everyone will approach it differently. But we can learn from this multitude of ideas if we collect them. The team used this large variety of designs to train an AI bot to build a house in the game for you.

When the stakeholder is not involved, since *Participatory Design* is missing here, the interests of the stakeholders are not represented as a driver in the generative design. The motivation of the third-party contributors drops, or the crowd starts hacking and guiding the system into open-ended explorations. *Picbreeder* (Figure 6.6) is an example from the non-architectural domain. In *Picbreeder* people were shown an image and eight ways it could evolve and asked to pick an option they like. A person will get tired after several images, so they give their progress to the following user. This, in effect, resulted in collaborative guidance of the generative algorithm (Secretan, Beato, D Ambrosio, et al. 2008).

And finally, by mixing *Participatory Design* with *Crowd Wisdom* (AD), we get stakeholders and a crowd of non-stakeholders creating architectural designs in an open-ended manner. Examples here are the traditional architectural competitions and online forums such as *Hausbau-forum*. This category relies primarily on communication technology to deliver successful projects, and the architectural domain lags significantly behind other disciplines. It is worth mentioning the advanced use of community discussion spaces and carefully orchestrated creative constraints of



Figure 6.5 – CraftAssist by Facebook AI Research. Image credits: Gray et al. 2019.

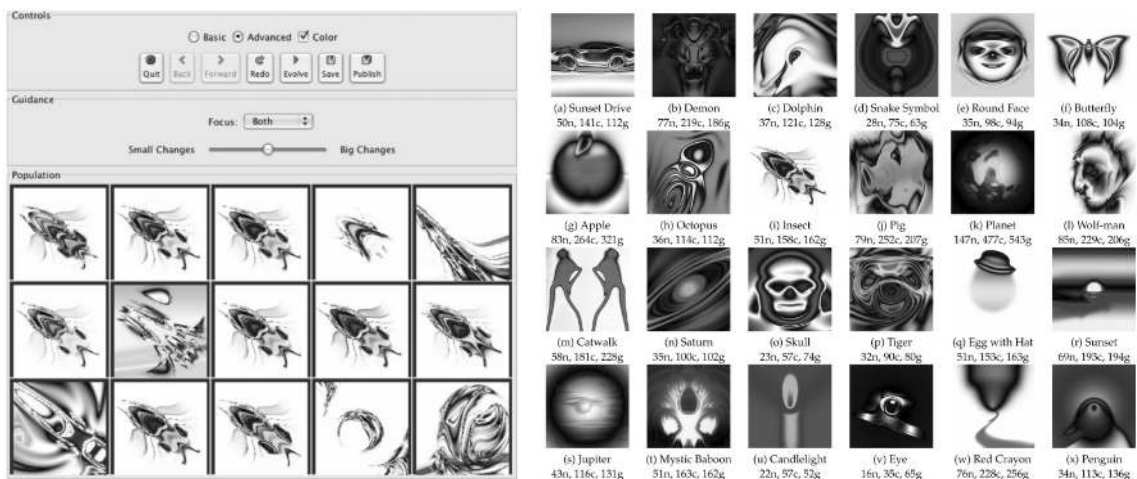


Figure 6.6 – Picbreeder. Left: the interface for the user to select and evolve images. Right, examples of images evolved with Picbreeder. Image credits: Secretan, Beato, D’Ambrosio, et al. 2011.

Reddit’s project *r/Place* as well as the intricate curating system for *Wikipedia*’s content (Figure 4.7).

To summarize, five crossings have architectural precedents, one — *Game Design* and *Crowd Wisdom* — is considered a research gap in architecture but has a precedent in other fields (*Project Discovery*).

6.3 Four Triple Intersections

At the intersection of *Generative Design*, *Game Design* and *Crowd Wisdom* (BCD), we have a crowd of non-stakeholders creating architectural designs within a game or an environment with elements of game design using algorithmic descriptions of de-

sign procedures. There are no architectural precedents. However, the game *Foldit* is a precedent from the field of molecular biology. First, *Foldit* assigns points to players as feedback for how well they have folded the protein molecule (Game Design). Second, thousands of players play it and communicate with each other by sharing tips and even partially completed levels (Citizen Science). And third, each player can use a set of automatic tools such as the *Shake* and the *Wiggle* tools to algorithmically seek better molecule backbone and sidechain positions (Generative Design) (Foldit Wiki Contributors 2021). My project *Sensitive Assembly*, more specifically the *Sunlight mode* is a study I conducted to explore this category.

At the triangle of *Participatory Design*, *Generative Design* and *Crowdsourcing*, *Citizen Science and Mass Collaboration* (ABD), we find stakeholders and a crowd of non-stakeholders create architectural designs using algorithmic descriptions of design procedures. Neither architectural nor non-architectural precedents exist. I present the *Project Reptiles* (chapter 9) as an initial exploration of what this category might bring.

Crossing *Participatory Design*, *Generative Design* and *Game Design* (ABC), we have stakeholders create architectural designs within a game or an environment with elements of game design using algorithmic descriptions of design procedures. Neither architectural nor non-architectural precedents exist.

And last, at the intersection of *Participatory Design*, *Game Design* and *Crowd Wisdom* (ACD), we have stakeholders and a crowd of non-stakeholders creating architectural designs within a game or an environment with elements of game design. Again no architectural precedents exist, but the gamified computer science forum *Stack Overflow* is an example from another domain.

Altogether, all four three-field intersections have no architectural precedents. Fortunately, two have precedents in other fields (*Foldit*, *Stack overflow*).

6.4 The Core Intersection of All Four Fields

The ultimate intersection of all four fields *Participatory Design*, *Generative Design*, *Game Design* and *Crowd Wisdom* (ABCD) gets us to stakeholders and a crowd of non-stakeholders creating architectural designs within a game or an environment with elements of game design using algorithmic descriptions of design procedures. Neither architectural nor non-architectural precedents exist. I present three own projects in this category. Two projects with the *20.000 BLOCKS* framework in Minecraft (*Combinatorial Design* and *Play-Design-Build*, see chapter 8) as well as the case study *Rechteck2BIM* (See chapter 10).

Part II

EXPLORATION

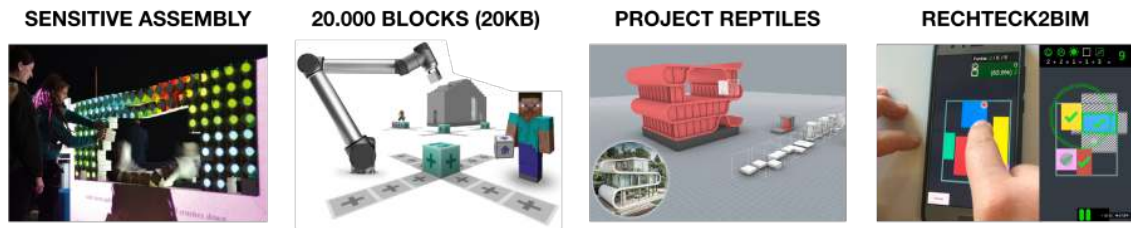


Figure 6.7 – The four case studies. Image credits: the author.

In this part, I present four case studies (Figure 6.7) that are the result of my work and student work I have supervised. The Exploration part concludes with the Discussion and Conclusion chapters.

Taken together, the case studies aim to explore the first research question: *What design paradigms lie at the intersection of participatory design, generative design, crowdsourcing, and games?* As such, the case studies are positioned in four of the six intersections defined as the research gap on Figure 6.3.

On its own, each case study explores one or more aspects of the second research question. Namely: *What roles, tools, and tasks can be offered at this intersection for the creative involvement of the various groups — architects, stakeholders, third-party contributors?*

Each case study is presented in its own chapter following a shared structure.

1. **Design Paradigm** — introduces the object of the participatory design and positions the case study on the map of four fields.
2. **Implementation and Setup** — describes the technical setup of the software and physical prototypes used in the case study.
3. **Machine Agency** — elaborates on the tools available to participants in the case studies. What are the used generative technique and game mechanics, i.e., the agency of the game opposing the player? The “Machine Agency” section of the case study *20.000 BLOCKS* introduces *Playable Voxel-Shape Grammars*. This generative technique from the class of shape grammars, mixed with game design, slowly took shape as the projects using *20.000 BLOCKS* were developed.
4. **Human Agency** — documents the involvement of people in the case study, i.e., which roles are involved, what tasks are crowdsourced from who to whom, and the game design around which the players interact with the prototypes.
5. **Take-aways** — present the key learnings from the respective case study.

Chapter 7

Sensitive Assembly

“All models are wrong, but some are useful.” (George Box 1979)



Figure 7.1 – The installation Sensitive Assembly. At NODE Digital Art Festival in Frankfurt, Germany, April 2015. Image credits: Oliver Tessmann.

7.1 Design Paradigm

Sensitive Assembly is an interactive installation and game presented at NODE15 Festival Digital Art in Frankfurt in 2015. The project was led and almost exclusively developed by me at the Digital Design Unit of TU Darmstadt. It is a prototype for interactive engagement in designing and assembling structures made out of identical

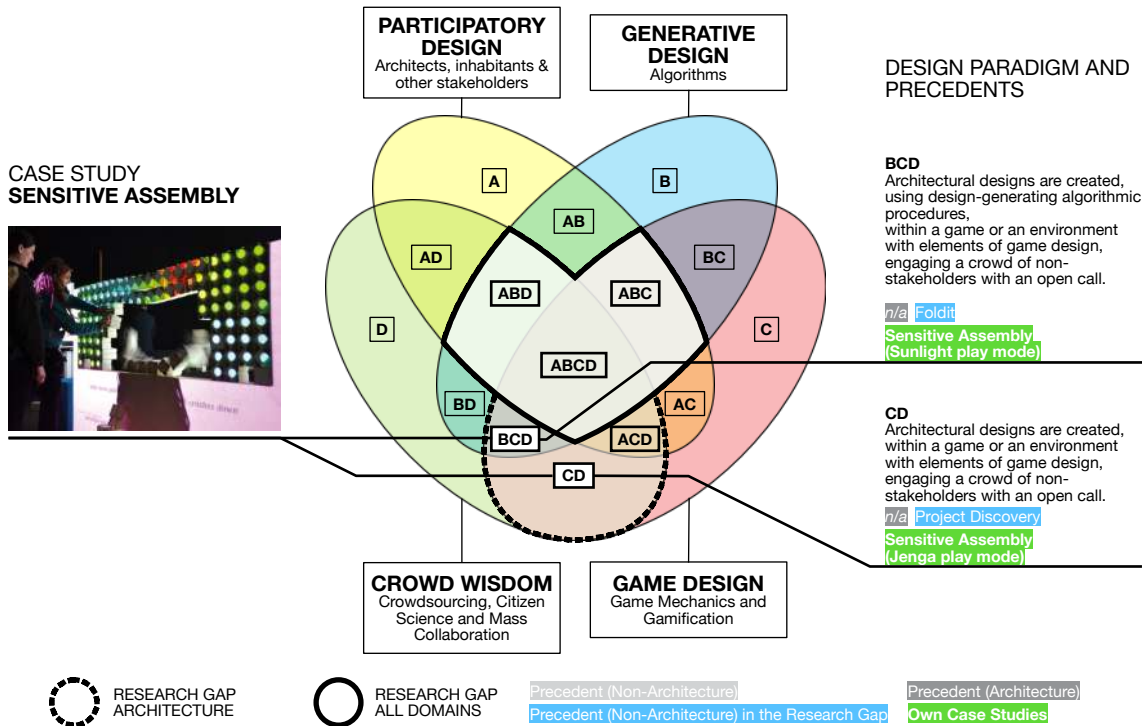


Figure 7.2 – The case study Sensitive Assembly on the map of design paradigms. The Sunlight play mode responds to design paradigm BCD (Generative Design + Game Design + Crowd Wisdom). At the same time, the Jenga play mode falls under design paradigm CD (Game Design + Crowd Wisdom). Image credits: the author.

small-scale elements. This process can generate structurally optimal designs for porous walls while remaining entertaining and engaging even for people outside the fields of architecture and civil engineering. Here, players unwittingly become designers of architecture and structure.

Turn-based game rules make the wall assembly a participatory and open-ended process. The use of lightweight building blocks enables participants to easily place and remove them (Figure 7.1). As in the famous tower-building game Jenga, where the game blocks of a tower are removed one by one, in *Sensitive Assembly*, players take turns removing blocks from a wall. Depending on the game mode, either a *predefined façade composition* must be recreated, or the gameplay results in a materially optimized structure that supports the top row of blocks.

The installation uses 3D scanning, structural simulation, and a state-prediction algorithm to aid the participants in near real-time in their decisions at each turn. Instead of solutions, algorithms could provide different bits of relevant information and assist in the design process. The main goal of participants in *Sensitive Assembly* is to generate and assemble structurally plausible façade designs with various opening configurations. *Sensitive Assembly* creates a motivating game experience that spans the digital and the physical world. Reasons for gamifying the assembly of block-made walls are to leave decision-making solely, and in this case literally, in the hands of humans.

Each game starts with a wall that is then perforated step by step and captured in real-time with a Microsoft Kinect 3D sensor. After each turn, the digital 3D model is automatically regenerated and updated in Rhino, thus presenting the newest

openings there. Then a simple structural analysis is conducted. The results of this analysis are, in turn, projected onto the wall as points of light in false colors, giving the players an indication of which blocks are relevant for the supporting structure and which can be removed. Therefore, augmented reality becomes an assistant in constructing and building.

Figure 7.2 shows the position of the case study *Sensitive Assembly* on the map of design paradigms within the four fields investigated in this work. *Sensitive Assembly* is designed as a game where players create and build wall designs, so it falls within the *Game Design* field (C). The case study does not address a specific project brief and, as such, does not have stakeholders, i.e., falls outside of the field *Participatory Design* (A). As a game, it engages third-party contributors in creating wall designs and, therefore, belongs to the field of *Crowd Wisdom* (D). The Sunlight play mode of *Sensitive Assembly* uses generative algorithms procedurally generate player missions from predefined wall designs, i.e., this particular mode belongs to the field of *Generative Design* (B) as well. The Jenga play mode does not utilize generative algorithms. The attributes listed above place the Sunlight play mode of *Sensitive Assembly* in the design paradigm BCD: *A crowd of non-stakeholders creates architectural designs within a game or an environment with elements of game design using algorithmic descriptions of design procedures*. At the same time, the Jenga play mode of *Sensitive Assembly* falls under design paradigm CD: *An organized crowd of non-stakeholders creates architectural designs within a game or an environment with elements of game design*.

7.2 Implementation and Setup

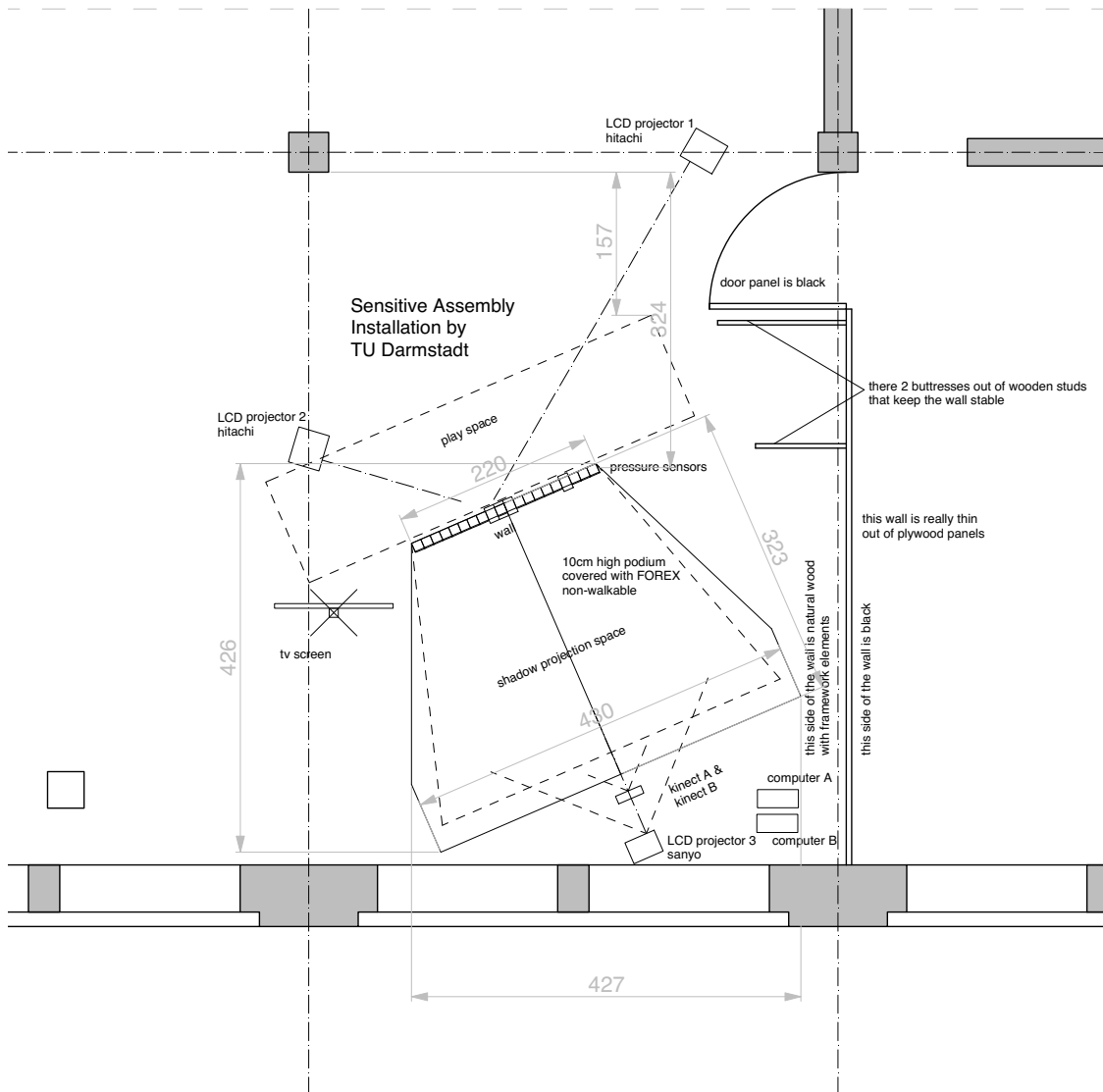


Figure 7.3 – Sensitive Assembly plan. Image credits: the author.

7.2.1 Physical setup

The plan of the setup of *Sensitive Assembly* can be seen on Figure 7.3. The installation consists of cardboard building blocks, a wall frame with a projection base, a sensing system, projection, and display facilities, as well as hardware and software that computationally link these elements (Figure 7.4).

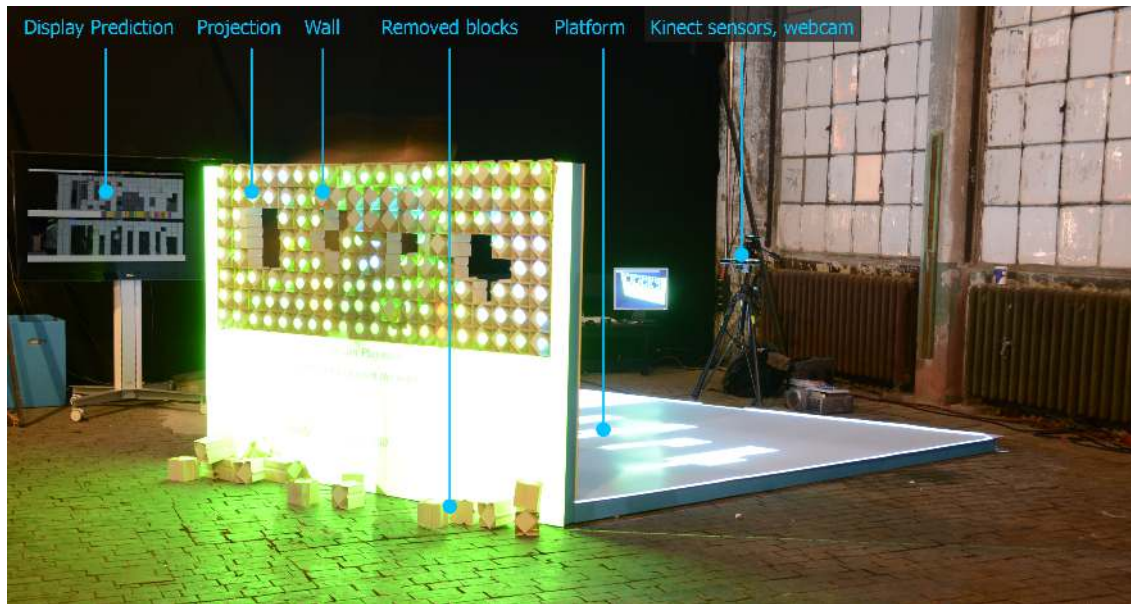


Figure 7.4 – The components of the Sensitive Assembly set-up. Image credits: the author.

Wall and wall blocks

The aim for the material system of the wall is to use the concept of digital materials, namely being “reversibly assembled from a discrete set of parts with a discrete set of relative positions and orientations” (N. Gershenfeld, Carney, et al. 2015). The material system of blocks described below is flexible and affords a variety of additive or subtractive assembly sequences (Figure 7.5).

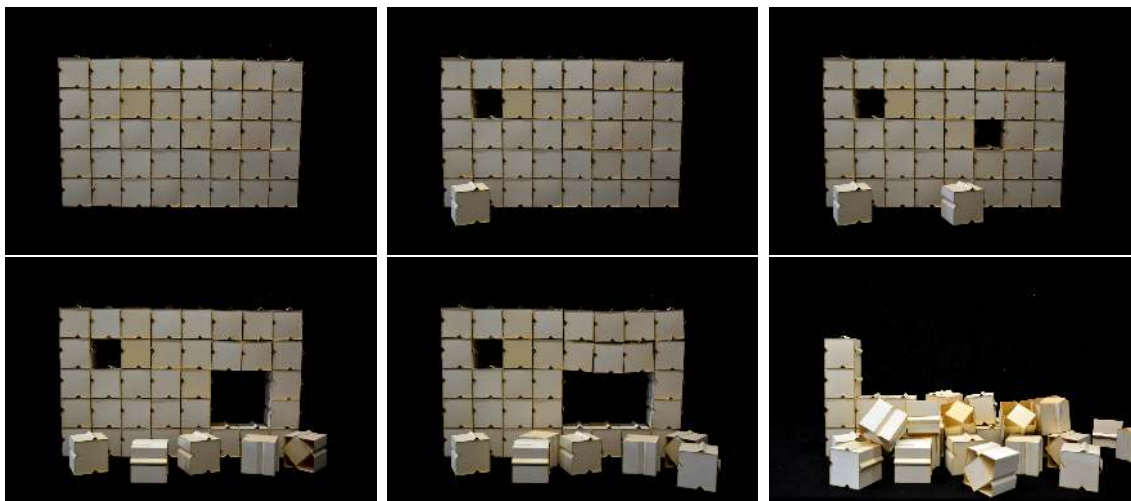


Figure 7.5 – A sequence of subtractive wall operations. From a fully closed wall until the wall collapses. Image credits: the author.

The first step was to develop a physical building block that could be stacked vertically and horizontally to make a wall-like grid. The grid-based approach allowed us to approximate a façade design into hollow and solid areas in the grid. An important factor was that the grid could take multiple states differentiated by the presence or lack of a block, thus allowing for many wall designs within the same system. Another requirement was that the grid was a statically indeterminate system, opening gameplay possibilities.

In the game Jenga (Figure 7.6), each block rests on three other, load-bearing blocks. This static indeterminacy makes it possible to remove blocks with no load-bearing effect. In *Sensitive Assembly*, this principle is translated to a wall made of cardboard modules placed on top of and next to each other without any adhesive. The static indeterminacy in *Sensitive Assembly* comes from the two notches and two protrusions in the blocks' design that allow blocks to interlock. A block is not only supported by its lower neighboring blocks but is also able to span distances that significantly exceed the block size, supported by lateral neighbors (Figure 7.5).

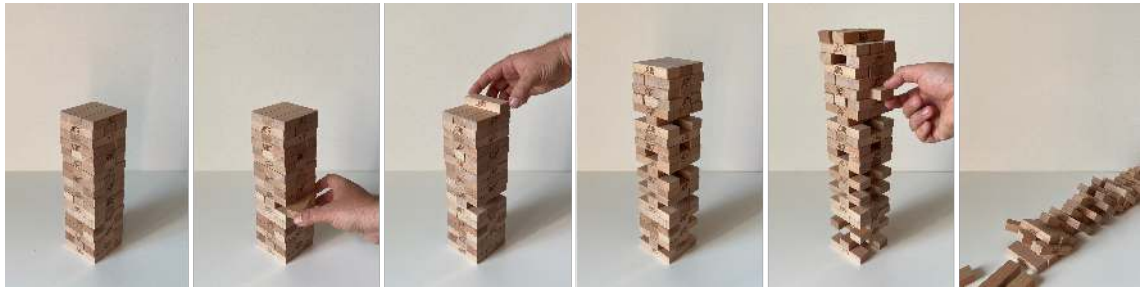


Figure 7.6 – The game Jenga consists of 54 blocks. The game starts with all blocks arranged in a tower of 18 rows of three blocks, each oriented in alternating directions. Players take turns taking a block out of the tower and placing it on top, paying attention not to knock the tower down. The last player able to make a successful turn before the tower collapses wins the game. Image credits: the author.

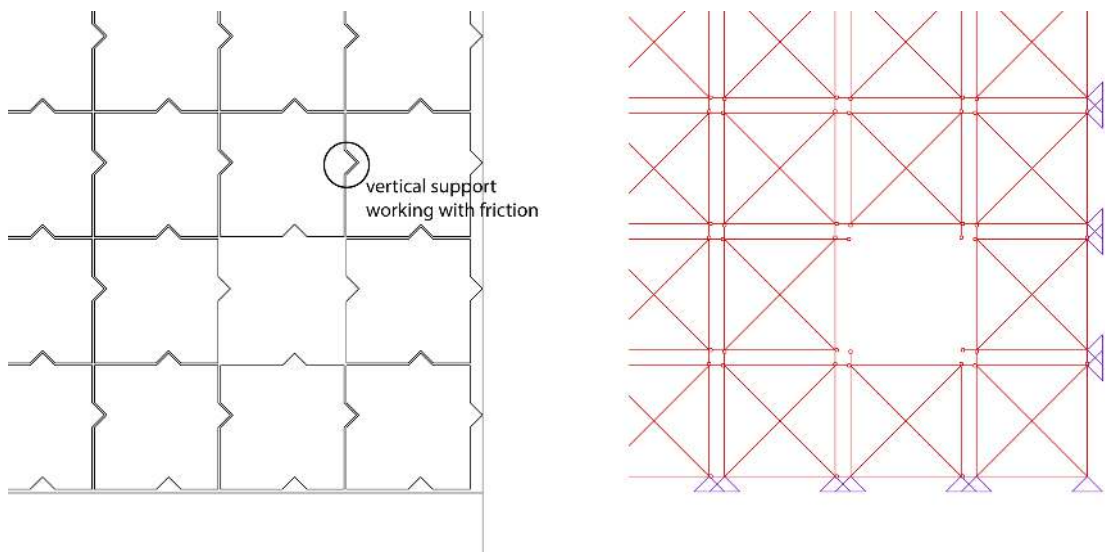


Figure 7.7 – The wall is made from interlocking blocks. Notches and knobs distribute the forces across all neighboring elements of every block. Left – schematic front view, and right – structural axial model. Image credits: the author.

Such a construction creates a statically indeterminate system that allows changing the force distribution within the wall (Matheson 1971). Removing one or several blocks does not cause an immediate collapse of the wall or parts of it. Instead, it leads to an activation of the lateral notches and knobs that become alternative supports. The structural ambiguity and redundancy require the tactile player experience in judging structural stability and are, hence, prerequisites for the formulation of game rules.

Fast and easy production of the individual blocks became an important aspect due to the large number of grid cells required to reach a grid resolution allowing the wanted creative freedom. We worked with a grid size of $20 \times 8 = 160$ blocks. The building blocks of the wall are made from two elements of laser-cut cardboard that can be easily assembled by hand in three steps as shown on Figure 7.9. The friction of the chosen material was high enough to allow for sound construction and low enough to allow for easy removal.

The blocks are designed to be easy to remove from the wall, simplifying the interaction of the participants with it. The four sides of the block that interface with neighboring blocks have slender notches. The fifth side, facing the Kinect 3D sensor, is completely closed for optimal detection. The sixth side, facing the players, is partly open for easier grabbing and pulling (Figure 7.10). It features a rotated square cap used as a projection surface during play.

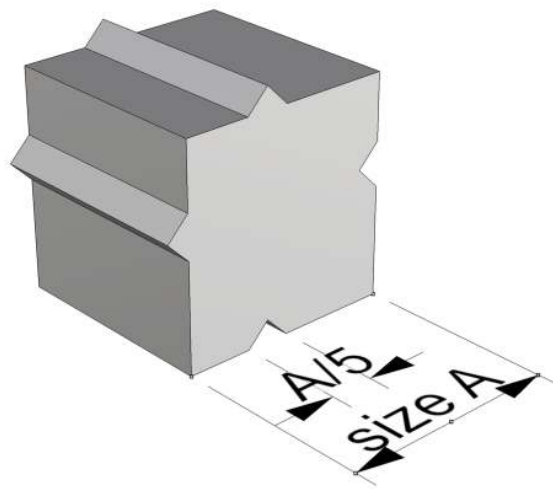


Figure 7.8 – The wall block design. Image credits: the author.

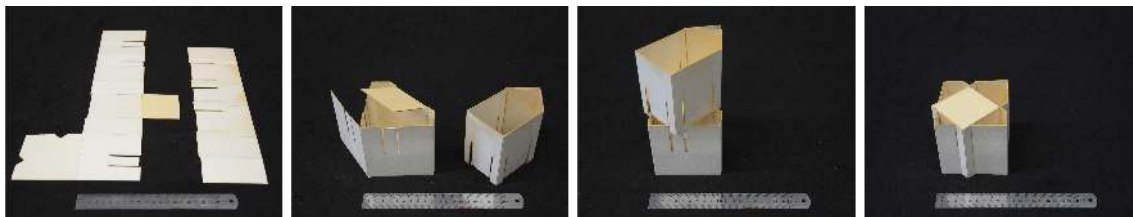


Figure 7.9 – The assembly sequence of a wall block. Two parts are lasercut from cardboard, then folded and dove-tailed into each other. Image credits: the author.

Vertical timber pillars on both sides of the wall frame the blocks and withstand the horizontal thrust in the system. An inaccessible white platform behind the wall acts as an anti-tilt counterweight. It is also used as a projection surface for the virtual wall shadow. Players stand in the front of the wall and look from the side or over it to see the projected shadow.

Equipment

The installation *Sensitive Assembly* consists of the following elements (Figure 7.4 and Figure 7.3):



Figure 7.10 – Grip and pull. The design of the blocks allows the easy gripping and pulling of blocks out of the wall. Image credits: the author.

1. A wall made from cardboard blocks arranged in horizontal rows.
2. A counterweight platform to project the virtual shadow on.
3. Two Microsoft Kinect devices, positioned at the wall axis on the opposite side of players so they can capture the wall states unobstructed. This Kinect constantly scans the wall to reconstruct its current state in a digital 3D model. The second Microsoft Kinect is to record the changing states of the wall during gameplay and feed these data into a machine-learning algorithm that seeks to predict the behavior of the wall.
4. Three LCD projectors used in the projection mapping setup. Two projectors displayed the instructions to players and the feedback on structural stability for each block. They were both mapped on the player-facing side of the wall from different angles. This creates redundancy to make up for any occlusions occurring from the players blocking one or the other projection beams. The third projector displayed the simulated wall shadow on the wall side opposite the players.
5. A webcam that captures the backside of the wall.
6. A screen that displays the webcam image of the current wall and the images of the prediction algorithm showing the expected future state of the wall.
7. A computer to manage the data capture from the Kinect and the webcam, compute the digital models and control the projection mapping.
8. A TV screen that showed the current feed of the webcam a photo taken during previous gameplays that best predicted the next wall states.

7.3 Machine Agency

7.3.1 Assistive systems

Central to the approach is the well-known concept of feedback. *Sensitive Assembly* expands the feedback loop beyond the digital realm, integrating it with the physical making — the assembly and disassembly. To aid the players' decision which block to remove next, *Sensitive Assembly* uses reality augmenting projection mapping to overlay non-apparent structural information onto the wall blocks. Matching

the computational structural evaluation to the perceived structural stability of the physical block grid was essential to create a smooth player experience.

There are essentially two technological means to augment the reality perception of players. The first is projection mapping, where an LCD projector projects digital information onto physical objects of interest. And second, Mixed Reality (Augmented or Virtual), where the player wears a headset with an active display that overlays digital information onto their vision. In *Sensitive Assembly*, the higher number and flux of participants is an essential generative factor. Therefore, we wanted to minimize the prerequisites for participation. The first strategy, with projection mapping, seemed more suitable, as it does away with the need for players to put on a headset. It also ensures everyone in the installation's vicinity can see what the players see for maximum engagement.

Simulation-based feedback

While people play, two Microsoft Kinect sensors and a webcam constantly observe the wall (Figure 7.11). A Grasshopper definition handled the geometrical reconstruction and various simulations.

The first Kinect sensor is linked to Grasshopper and used to generate a digital 3D model of the wall. The Kinect returns an array of points mapping the Kinect-facing surfaces of each block. Once a player removes a block from the wall, the change is also present in the point cloud of the sensor and leads to an update of the digital model. The point cloud is read by Quokka¹, an add-on for Grasshopper. The Grasshopper definition then sorts out the missing blocks using a grid of boxes by counting the point cloud points within each box. If the number goes lower than a given threshold, the corresponding block is counted as missing.

We observed that sometimes a block, while present in reality, would not be reconstructed in the digital model because of the calibration of the Kinect point cloud reinterpretation. That missing-block bug occurred most often when a large span was hanging down away from the strictly orthogonal rows and columns of the wall matrix (Figure 7.26). In most games, besides during the final stage, this did not affect much the output of the structural simulation because of the sheer number of present and correctly detected blocks. However, improving the precision of block recognition can help to distribute even more correct information to players — especially towards the end of the games when there are very few blocks left in the wall.

Based on the reconstructed wall topology, an axis model is generated and analyzed with *Millipede*², an add-on for McNeel Grasshopper by Sawako Kaijima and Panagiotis Michalatos. The model is used to simulate and analyze the structural capacity of the current wall configuration. I chose the metric *Y rotation* — rotation around the Y-axis, i.e., perpendicular to the wall plane — as the metric to display to the players (Figure 7.12). Among other available metrics such as *displacement*, *stress* or *momentum*, the feedback from the *Y rotation* metric proved to be closest to the tacit experience of players, hence most intuitive to understand. The determined structural deformation from the analysis model, measured for each block, is then mapped to a color gradient. The resulting false-color dot grid is sent to Processing

¹<https://www.food4rhino.com/en/app/quokka>

²<https://www.grasshopper3d.com/group/millipede>

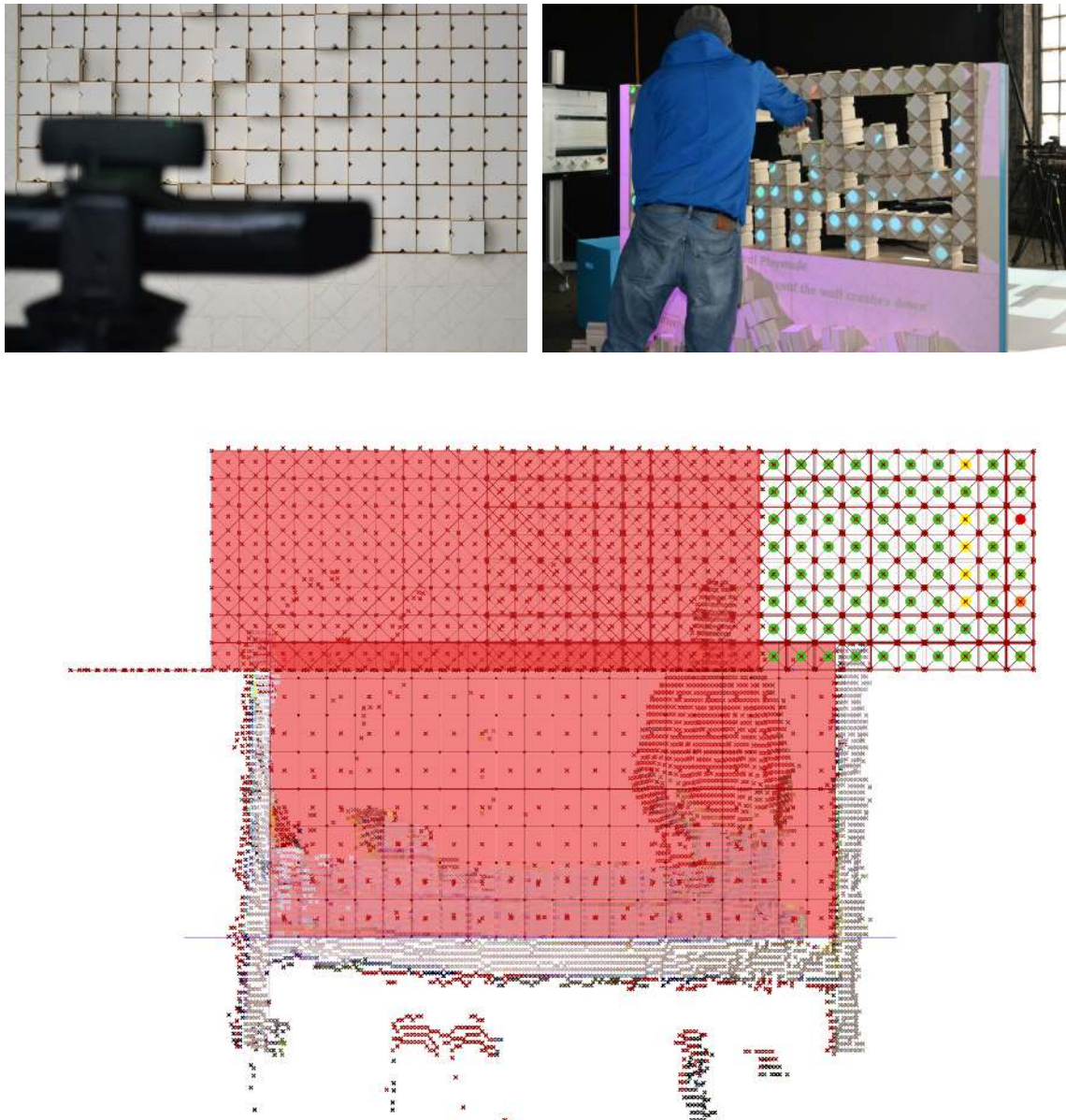


Figure 7.11 – Wall state reconstruction. Top-left: the Kinect facing side of the wall (photo: Christian Leicher), Top-right: Player interacting with the augmented wall. Bottom: Kinect point cloud data, including a player, and its reinterpretation as a block wall and a structural model. Image credits: the author.

via UDP for projection onto the wall.

The false-color dot grid is projected onto the wall and guides the players. Blocks colored in red have higher deformation, i.e., they are needed for the structural integrity of the wall, and green blocks can be taken away (Figure 7.13). The front projection mapping illuminated only small areas of the blocks (Figure 7.1). The rest of the wall surface was not cast with light. Thus, the wall received a media skin revealing its invisible structural capacity as a diagrammatic infographic (Figure 7.14). The augmentation supported the players' intuitive understanding of the structural behavior of the wall.

The last step in Grasshopper calculates the wall's shadow from an imaginary light source. The grid of boxes is transformed into a mesh, and a shadow outline is generated on the XY plane using a virtual light source represented by a 3D vector.

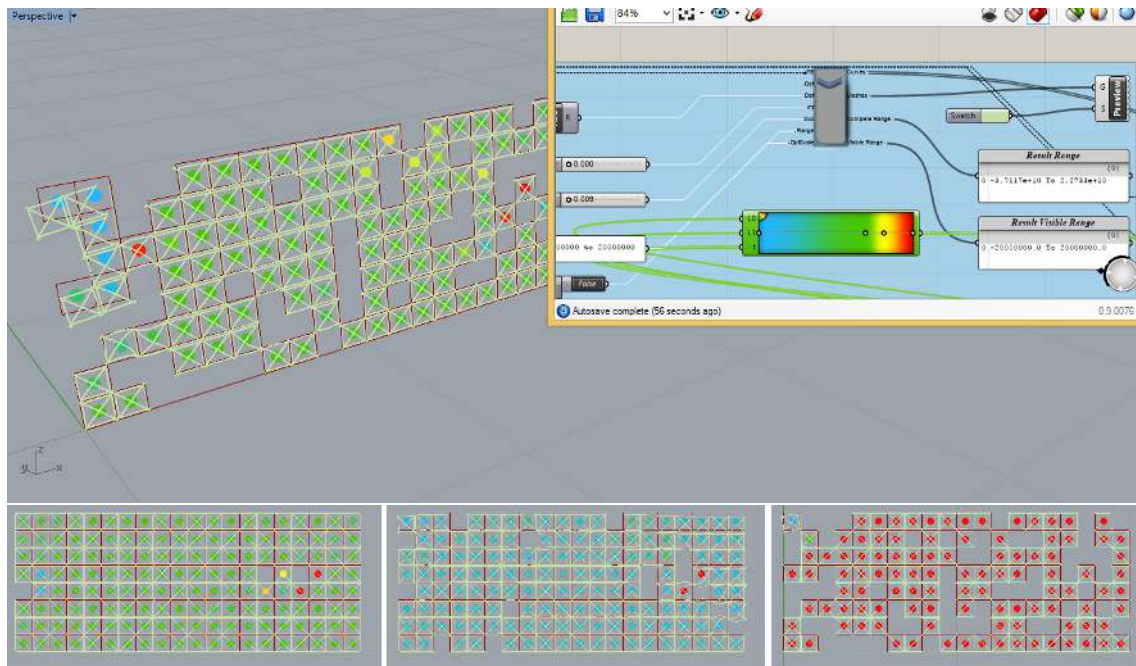


Figure 7.12 – Structural analysis. Deformation simulation is running in real-time in Millipede, an add-on developed by Sawapan for McNeel Grasshopper: Top: the simulated deformation is shown to illustrate the relevance of choosing Y Rotation as the feedback metric. Lower row — three steps showing various amounts of missing blocks and the comparative structural deformation of the present blocks. Image credits: the author.

The shadow outline is prepared as a list of points and sent to Processing via UDP.

A custom-developed Processing sketch managed the game logic for the various game modes and the projection mapping. Using the User Datagram Protocol (UDP) with gHowl (add-on for McNeel Grasshopper) and the UDP library for Processing, I set up a bi-directional real-time link between Grasshopper and Processing.

The Keystone library for Processing handled the projection mapping allowing a perspective distortion of the projected image for flat projection surfaces.

The geometry received from Grasshopper (structural color coding and shadow) is rendered onto a Keystone surface in the liquid crystal display (LCD) projector screens (Figure 7.13). The Keystone surface has four corner handles, which we could use to match its perspective correction to the actual artifact of the wall and shadow plate.

The process described above is shown on the flow chart on Figure 7.15. A full feedback loop runs nearly in real-time, although it has a slight lag of 10–40 seconds due to its many computational and transformational steps.

Prediction and machine learning

The second feedback mechanism in *Sensitive Assembly* predicts the near future states of the wall structure. Figure 7.16 shows a snapshot that demonstrates the finding of both similar states during different games and accurate prediction of collapses.

Stig Anton Nielsen developed the prediction algorithm for his doctoral work at the Chalmers University of Technology in Gothenburg. The algorithm combines cluster analysis and sequential data mining to detect temporal re-occurrences in

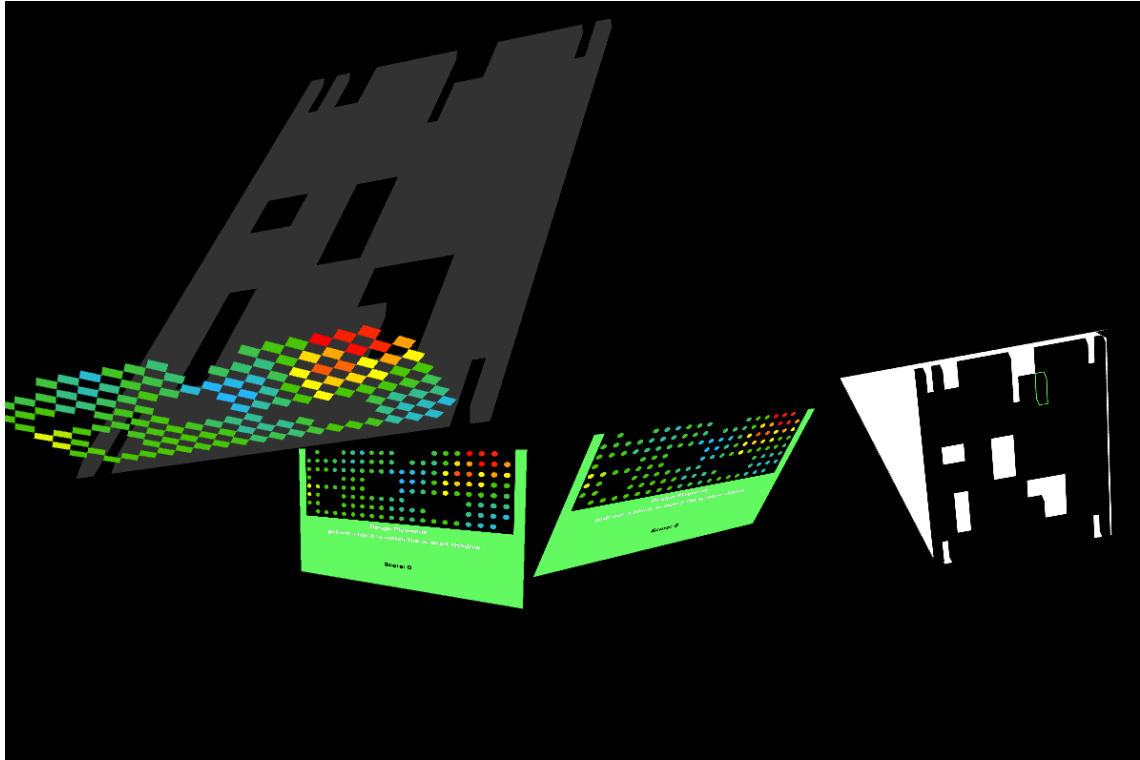


Figure 7.13 – Projection mapping textures. The wall and shadow images are transformed to match the perspective correction resulting from the falling angle of the projection. Left: Virtual model for debugging purposes; middle: two front projections; and right: the shadow projection. This augmentation subsystem also acted as a game user interface (UI), displaying scores, missions, and other gameplay-related information over the lower portion of the wall. Image credits: the author.

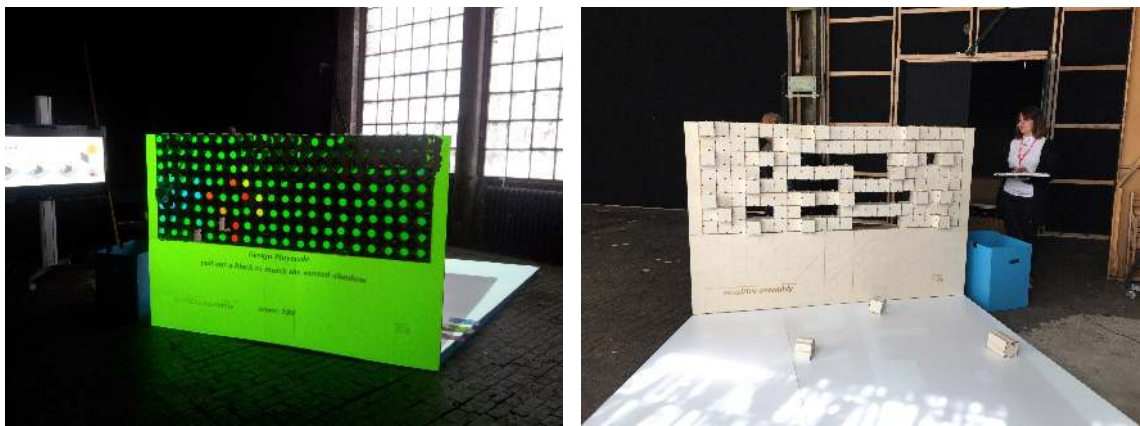


Figure 7.14 – Wall sides. Left: The wall shape and its projected infographics on the front. Right: On the backside the wall's shape is very much perceived as an architectural design. Image credits: the author.

massive flows of data (Cabanés and Bennani 2010). This kind of machine learning enables the machine to predict future states once it detects sequences of events similar to previous occurrences. After numerous games have been scanned, the algorithm recognizes specific perforation patterns in the wall that had caused the wall to fall in previous games. Based on this past information and the interpretation of repetitive patterns, the algorithm predicts the future behavior of the perforated wall.

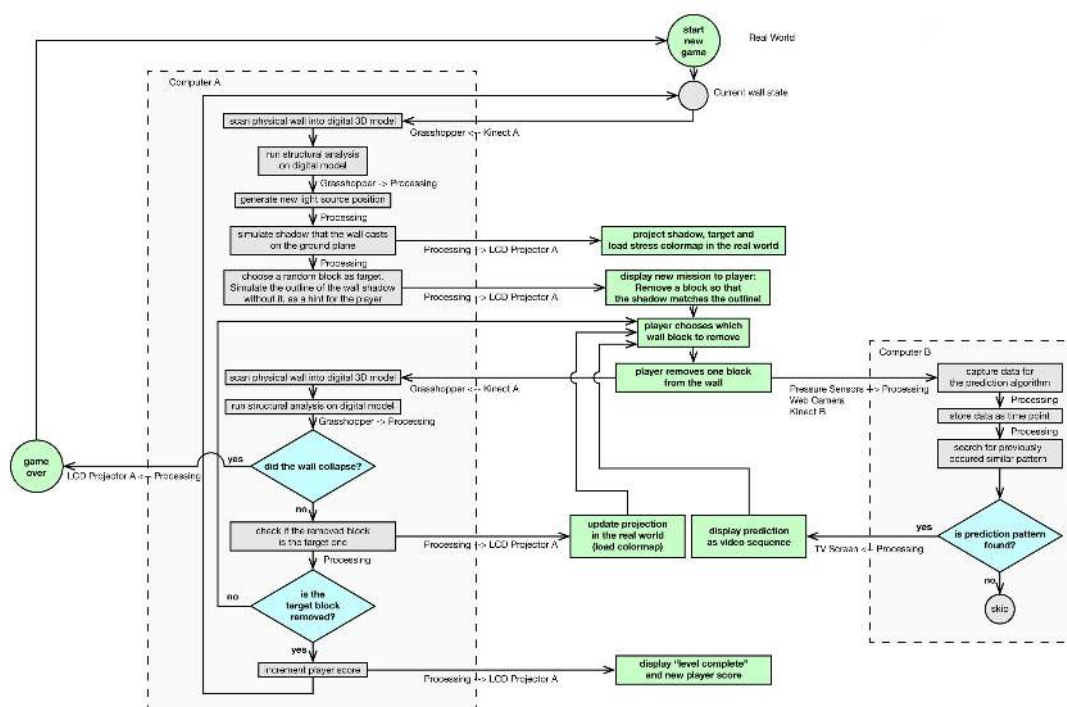


Figure 7.15 – The flowchart of the feedback loop. Green items show events in the real world. Gray ones are digital model steps. On the left are the steps of the program running on computer A and facilitating the structural simulation augmentation and the gameplay interface. On the right are the steps of the prediction algorithm running on computer B. Image credits: the author.

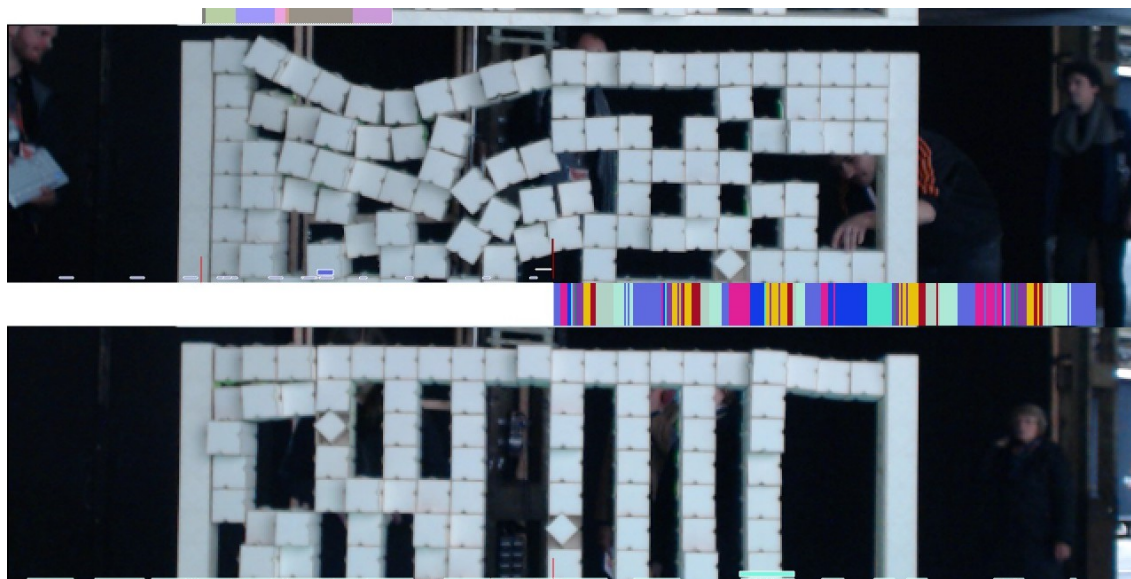


Figure 7.16 – Collapse prediction feedback. Images on the monitor show both the current state of the wall in the bottom half, and the next five 'moves' from the prediction on the top half. The predicted images run in continuous loop, until another prediction is found. If no valid prediction is found the top half is blank. Image credits: Stig Anton Nielsen.

The algorithm analyses the wall's porosity, edge contours, and structural coherence. Increasing porosity, together with longer horizontal (stressing) edges, increases collapse probability. The algorithm uses the image material recorded in past games to represent the future behavior of the structure using images of actual wall states from the past so that the information is presented in a way that is understandable for humans.

The prediction must capture both data and a representation, which can be retrieved and displayed to the player when necessary. The Kinect sensor was used to capture data, and a webcam was used to capture the representation. The Kinect captures 3D point cloud data from each step of the game, but before using the data for the algorithm, features were extracted by looking at the internal relations of the 3D points. These features — *edginess*, *porosity* and *coherency* — are not directly structural analysis, but they are designed to characterize the structural state of the wall (Figure 7.17). *Edginess* looks at the distribution of edges in the structure, *porosity* looks at the distribution of small holes, and the *coherency* describes longer stretches of continuous matter in the structure.

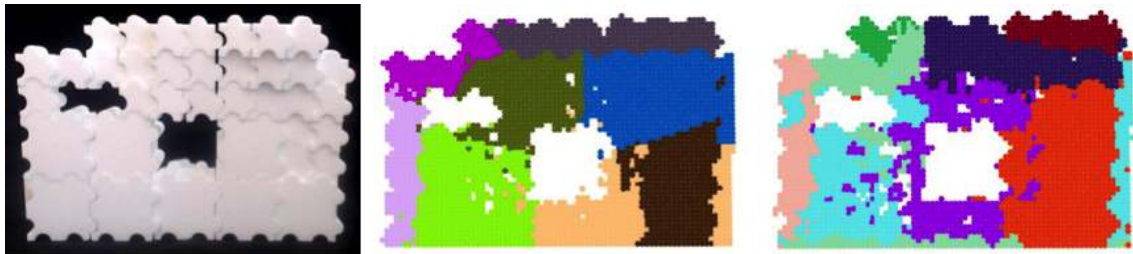


Figure 7.17 – Edginess, porosity and coherency. A prototype mock-up of the installation, where relevant features were designed for extraction. The left image shows an image of the structure, and the other two images (middle and right) visualize the different features: edginess, porosity and coherency. Image credits: Stig Anton Nielsen.

The algorithm is not so much concerned with the actual structural integrity of the wall. Instead, it develops its own understanding through this feature space and the sequence of states. After experiencing and remembering previous games, it becomes able to compare the current game to prior games. Obviously, all the games are different, but they have similar sequences after discretizing irrelevant features. For example, the start of the game P may be similar to the games A, B, and C, while the end of game P is similar to games I, J, and K. When P is at its start, the system will use start sequences from games A, B or C to predict the continuation, while at the end of game P, the system will use sequences from the games I, J or K for prediction. If the current game P is sufficiently unique, no prediction will be shown, but data and representation from game P are still recorded and used for predictions in the following games.

Initially, the predictive representation did not give helpful information. However, after a day of training on numerous games of the type *Jenga*, it could be observed that the system predicted collapses with an accuracy of just a few moves ahead. The system was also very reliable in retrieving similar games from the database.

In contrast with the structural analysis through Millipede, the prediction algorithm is not re-creating an additional digital representation of the wall to simulate its behavior but instead analyses patterns over time. As the notches in the building blocks led to a relatively complex structural behavior, the shortcomings of the struc-

tural analysis were balanced by the prediction. However, unlike the static model, which provided feedback from the first game, more games were needed to train the algorithm and improve its precision before it began being useful.

7.3.2 Browsing of player-created wall designs

Every wall state is recorded during the gameplay, and the resulting dataset can be made searchable. The dataset-browsing functionality is implemented in Rhino/Grasshopper and requires the user to be in the role of an expert. However, with a custom-developed user-friendly interface that doesn't require specialized software to run, anyone — expert or non-expert — could potentially use the browsing mode to find designs according to given criteria.

The user looking for a specific design can search either by specifying a range of blocks the wall must have (Figure 7.18) or by passing a rough design (Figure 7.19). The returned results are sorted by structural performance, with the most stable designs shown on top. The two search functions can be combined in a complex search.

The *search-by-blocks* functionality can be helpful when limited material supply or material budget is available. That way, the person looking for a design can control the final costs. They can specify the minimum and the maximum number of blocks in the wall and the maximum number of blocks in the top row.

The *search-by-design* option is more powerful. In this mode, the person looking for a design can roughly sketch where they prefer to open the wall or where they want to have the wall closed. Generating a wall pattern from the sketched outlines is easily possible by simply voxelizing the wall outline and removing the voxels that fall in an area specified by the user as a wall opening. However, the user-generated designs can be helpful as a source of inspiration, offering a moment of surprise as they would not always follow the most efficient or the most optimal way to distribute the wall blocks into the wanted facade pattern.

Furthermore, there is the added benefit that the search results have already existed in reality, and as such, they are validated for their real-world performance and feasibility. The search results are sorted with the most structurally sound designs first. Suppose the user is looking for a facade with a rectangular window. In that case, they might get suggested a facade with an arc window as a structurally stable option that satisfies their criteria.

The 21.000 recorded designs during the NODE exhibition constitute a proof-of-principle dataset. The designs are somewhat rudimentary with their resolution of 20 by 8 blocks, yet the search method can be applied on datasets with more complex designs.

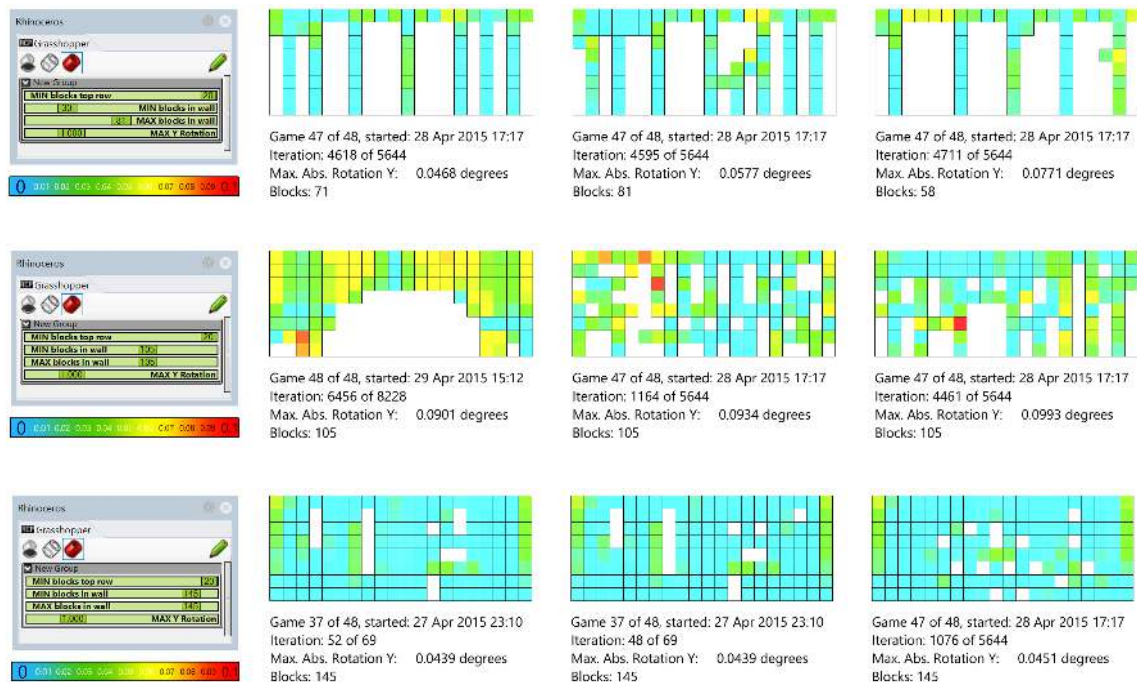


Figure 7.18 – Searching by a number of blocks. The expert user looking for a suitable wall design can specify an upper and lower limit. The top three most stable designs that contain the wanted number of blocks are returned. The top 3 row shows the most stable designs with a block count between 20 and 81. The middle and bottom rows show the top 3 most stable designs with 105 and 145 blocks. There are a total of 21844 designs to search from. Image credits: the author.

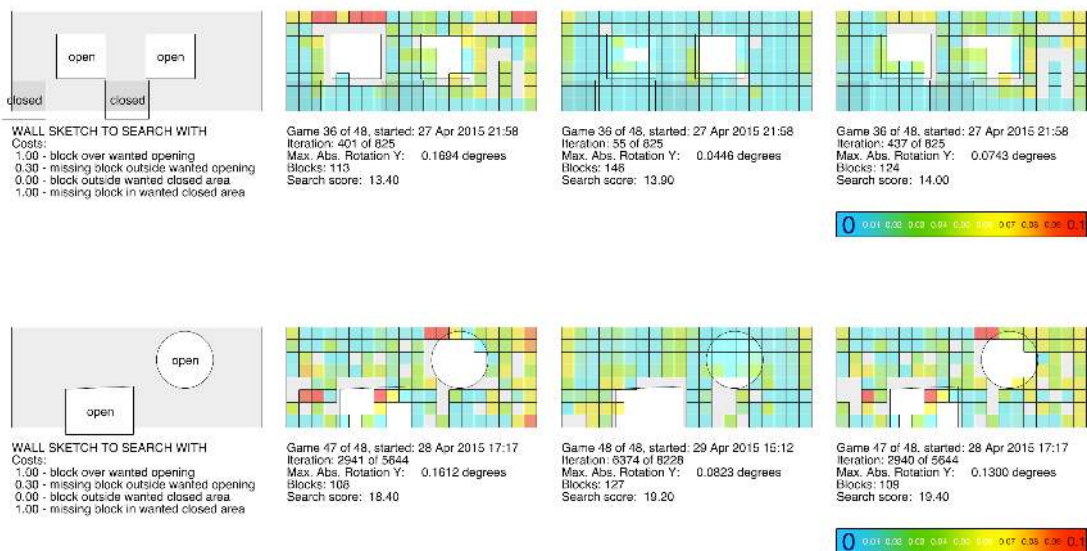


Figure 7.19 – Searching by design. The user can sketch a facade design by specifying the outlines of wanted openings and areas that must remain closed. Various costs can be assigned for calculating the difference between designs. A difference of +1 block and a difference of -1 block can have different costs. Each row shows the search sketch on the left and the top three most fitting designs next to it. Image credits: the author.

7.4 Human Agency

Human agency in *Sensitive Assembly* is constituted mainly by the visitors of the NODE Art Festival. As players interact with the installation, they are in the role of third-party contributors, i.e., the crowd.

The exhibition visitors played two game modes described in the previous section: Sunlight mode and Jenga mode.

In the span of seven days, more than 100 participants played about 50 games, consisting of more than 21,000 wall states.

While people were playing, we tracked whether they stayed for an entire game or just a few moves, whether they played alone or collaboratively, and to what extent they used the projection mapping feedback for structure and shadow, respectively.

An additional mode of engagement, that of an architect using the browsing tool to instantiate wall states from the database, was not tested in a user study. It was used to develop the concept of *autocomplete* that was later tested in the *Project Reptiles* case study.

7.4.1 Game design

Using the physical setup and the two feedback mechanisms as a framework allowed the development of several game modes.

The game modes' main requirement was to let the players become unwitting façade designers and engineers.

Two main approaches are possible:

1. *Additive* — starting with zero blocks and following a set of rules to add blocks until a design emerges iteratively;
2. *Subtractive* — starting with a fully closed wall and following a set of rules to remove blocks one by one until a design emerges.

Quick in-house tests showed that the *subtractive* approach was more engaging for potential players because the wall is present on site all the time, drawing attention and inducing curiosity. An additional engaging feature of this approach is that the wall collapses in an effective and entertaining way after a critical number of blocks are removed.

All participants were briefly instructed on site what the game rules of the current game mode were.

We distinguish two main aspects of participant involvement:

1. Collaborative assembly – where the design solution is known but is to be built using non-specialized labor;
2. Collaborative design – where the design solution is unknown beforehand but must comply with the given material and environmental constraints.

Collaborative assembly — Sunlight play mode

The Sunlight play mode aims to involve participants in the assembly process of a façade. The game mode is built on one of the architectural functions of a wall,

namely, as a sunlight filter. The overall game goal here was to subtract wall blocks corresponding to the desired insolation from a given light source (virtual sun) (Figure 7.20).

The Sunlight play mode works with predefined façade designs, which vary in how they challenge the grid structure. Some, for example, a regularly perforated façade, are more stable, while others, such as a set of random openings or ribbon window façade, are less stable (Figure 7.21 left). We used this difficulty parameter to introduce progressing game levels.

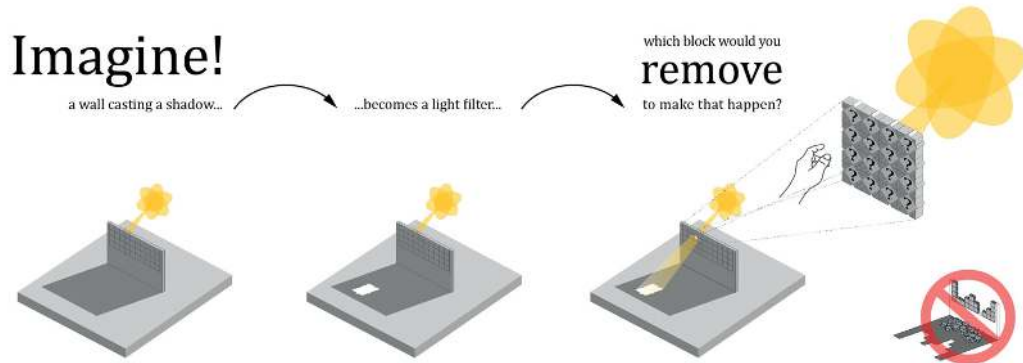


Figure 7.20 – Instructions for the Sunlight play mode. Image credits: the author.

The game rules of the Sunlight play mode are formed around a single player interacting with the wall. A predefined façade design (Figure 7.21 right) with multiple openings was split into single-block missions: the first beginning from a solid wall, and the last resulting in the predefined design. Each mission consisted of a random block automatically removed in the digital model. Light falls through this new aperture, and the simulated wall shadow gets perforated. This new shadow shape is projected onto the platform. The player must guess which block to remove from the physical wall to match its perforation to the digitally generated shadow projection.

The Processing sketch then checks whether the list of missing blocks contains the box number given as a mission to the player. If so, a success message is displayed, the player's score increases, and the next mission is given.

Collaborative design — Jenga play mode

The well-known game Jenga is the inspiration for the collaborative design play mode. To turn players into co-designers, we needed an objective design evaluation criterion, which was both computable and tactile. Structural performance emerged as an excellent measurable parameter for such a criterion. Two or more players take turns in removing blocks from the wall, which increases its porosity and makes it less and less stable (Figure 7.22). The game ends when the top row of the wall collapses. The winner is the last person to successfully remove a block from the wall (Figure 7.23). The idea of this arcade mode was to let players decide which block to remove hence turning them into co-designers.

In this game mode, the overlaid representation of the digitally calculated structural performance aided the players in deciding where to draw a block. The wall

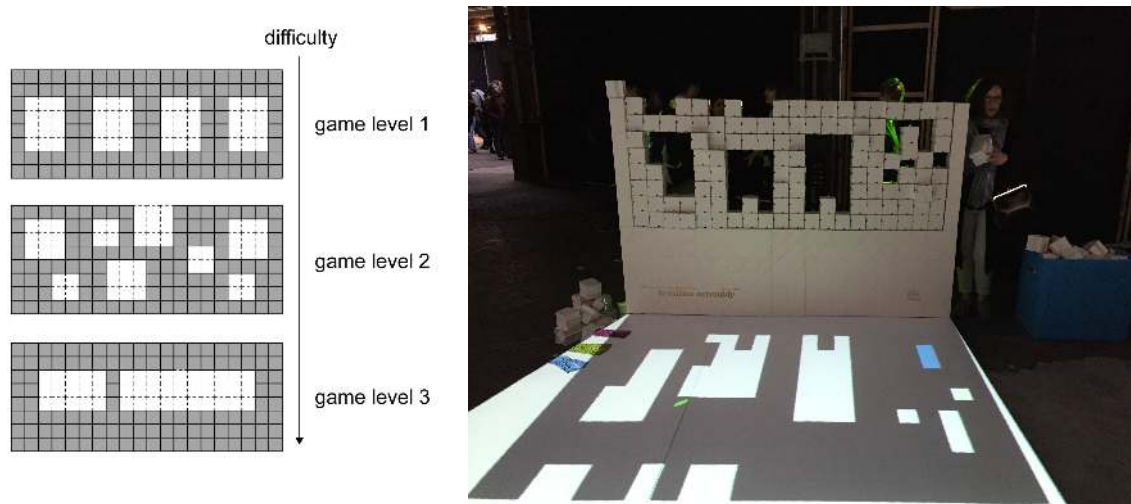


Figure 7.21 – Sunlight play mode. Left: Three difficulty levels from the single player mode. Right: A snapshot from a game – the mission target is the blue rectangle in the shadow. Image credits: the author.

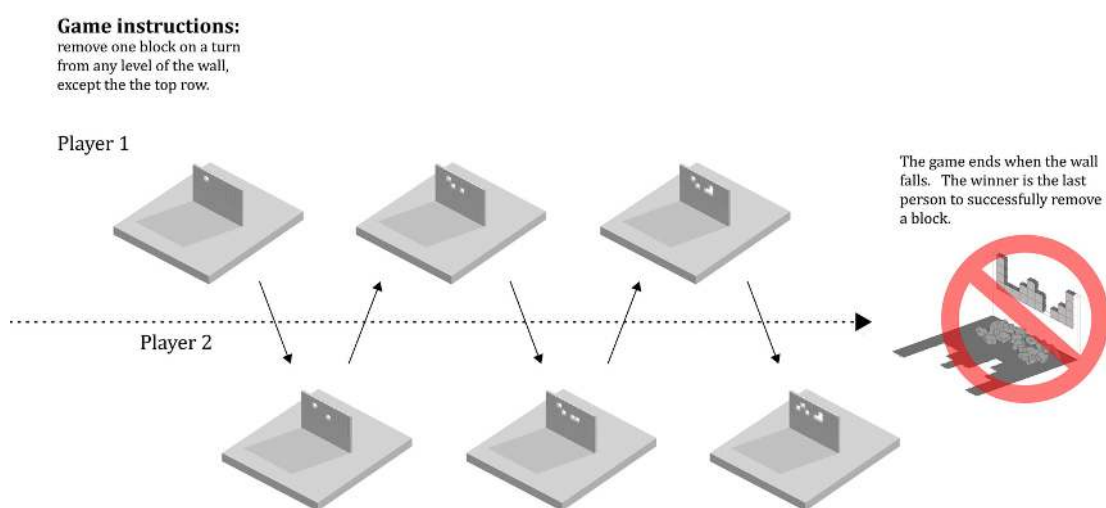


Figure 7.22 – Instructions for the Jenga play mode. Image credits: the author.



Figure 7.23 – Jenga play mode. Left and middle: Two players playing against each other. Right: The game is over when the top row of blocks collapses usually leading to the whole wall shattering. Image credits: Oliver Tessmann.

itself and the feel of each block were also giving vital tactile information to the users whether the block was structurally important or not. The predicted wall state, shown on the external monitor, was either warning the players of an impending collapse or reassuring them that no collapse would occur within a reasonable time-frame. With the information from the prediction, the structural representation, and the physical 'feel' of the elements in the wall, the players could improve decisions and adjust their level of caution for each modification they made to the wall.

7.5 Takeaways

The following takeaways are based on digitally recorded wall states, observations on site, and informal conversations with the players.

7.5.1 Takeaway 1: Challenges vs. Instructions

. The Jenga play mode was better in engagement than the Sunlight play mode. This is most likely due to the players experiencing a challenge they need to solve versus being given step-by-step instructions. Also, the much faster game pace and the social component added to the game when people played against or with each other contributed to making the Jenga play mode more popular. The two players acted as rivals, usually at the start of the game. Towards the end, when the moves became more and more challenging, the players often switched to a collaborator mode – discussing together which blocks are safer to remove (Figure 7.24).

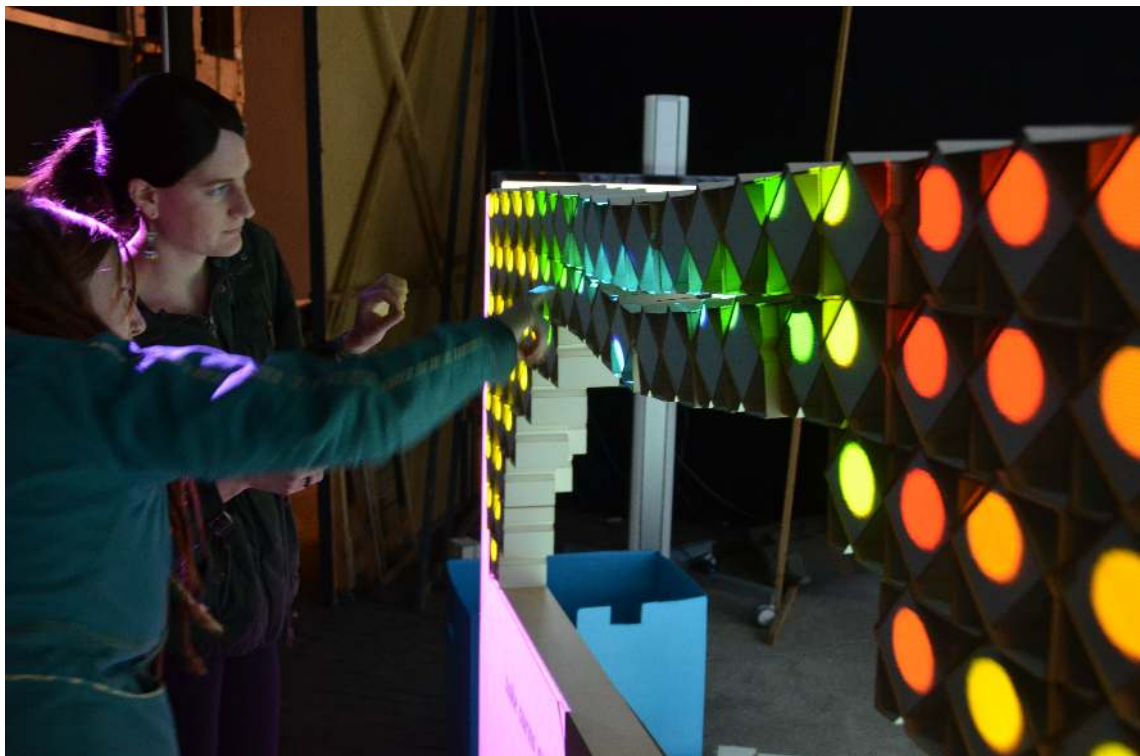


Figure 7.24 – Competition and collaboration. Rival players acting as collaborators and discussing which block is the safest to remove next in the Jenga play mode. Image credits: Oliver Tessmann.

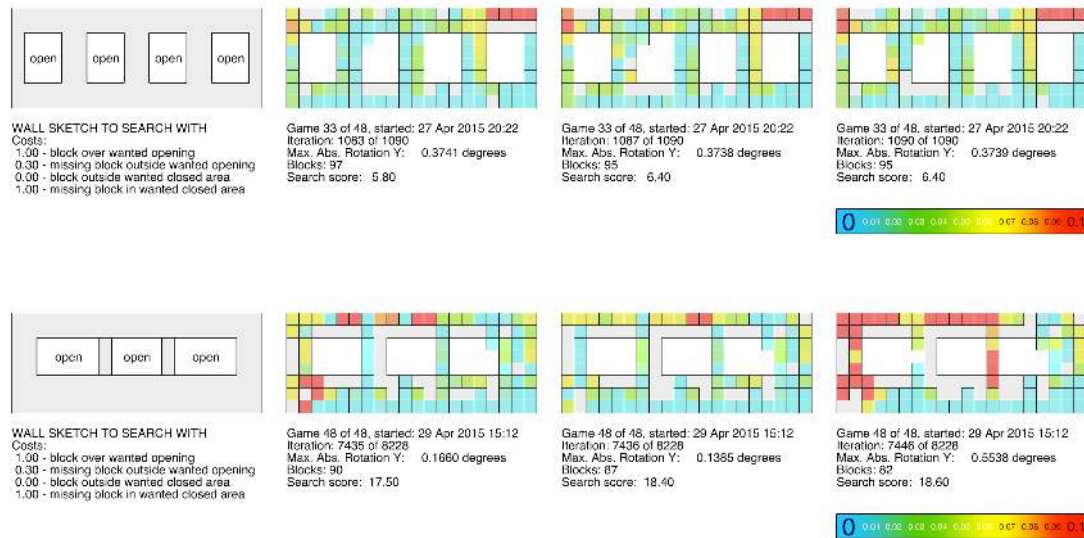


Figure 7.25 – Fuzziness in the player created designs. A search-by-design for the predefined facade designs from difficulty level 1 (top row) and difficulty level 3 (bottom row) offered as game missions to players in the Sunlight play mode. The top 3 results show the inevitability of fuzziness. Image credits: the author.

7.5.2 Takeaway 2: Embrace fuzziness

In the Sunlight play mode, players were asked to assemble predefined wall designs. The sequence of blocks to remove given to players was not always structurally compatible with the predefined facade design. It led to fuzziness in the outlines of the assembled wall openings (Figure 7.25). Sometimes the game asked the player to remove a block, which led to another block falling as well. On other occasions, the player pulled the wrong block. While structurally compromising, this fuzziness could hold potential for design expression and design inspiration. And as such, it should be embraced and encouraged instead of prevented.



Figure 7.26 – Topological optimization games. Left: An extreme result from a post and beam game type; and right: an 'Arch' game type in progress. Image credits: the author.

7.5.3 Takeaway 3: Design as a byproduct of play

When players pursue a goal that has a meaning for the posed game challenge, they are immersed in the game experience. For example, in the Jenga play mode, various structurally optimal designs emerged in multiple games (Figure 7.26). Those could be categorized into structural typologies such as post and beam constructions and arches. When asked after the end of the game, the players often expressed that they did not intend to form such a textbook structural example. The first couple of moves in the game were decisive for developing the structural typology. If the first blocks were removed from the sides of the wall, a post and beam type, similar to concrete highway bridges, emerged (Figure 7.26 left). Blocks removed from the center of the wall at the start of a game led to structural arcs (Figure 7.26 right).

Towards the end of a typical game, the wall states were not intended or premeditated by the players but emerged purely as a byproduct of them trying to win at the posed game challenge. Nevertheless, they bear architectural qualities relevant to a given design context. Producing these designs and their architectural qualities required only players' onboarding to the needed skills to play the game. No special architectural knowledge needs to be communicated. The collection of such byproduct design and the subsequent search based on particular architectural qualities can be a significant source of design solutions or inspiration.

7.5.4 Take-away 4: Design Autocomplete

The ability of expert users to search in a database of designs that had already existed momentarily in the real world holds a great promise for relevant design assistance. Especially the *search-by-design* mode can be very powerful in letting the user provide a simple sketch that is then autocomplete with an editable, real-world validated design.

7.5.5 Conclusion

Sensitive Assembly demonstrated that the engagement of festival visitors in a wall design and assembly process was successful. The augmentation provided a helpful information layer for the shadow-filter game modes, while it remained mainly in the background during the Jenga-like arcade mode. The gamification of the method led to entertaining engagement and showed that people preferred fast-paced, collaborative gameplay. After being taught for a day, the prediction algorithm was very precise towards the end of each game when prediction mattered the most. However, the players rarely used the prediction because it was displayed on a TV screen, not on the wall itself, causing players to forget about it when they were in flow.

The case study outcomes showed high potential in further investigating collaborative building challenges.

The straightforward design and assembly process prototyped with *Sensitive Assembly* could be applied successfully for prototyping façades or feature walls. The inclusion of generative design search and basic comparative markers for performance, such as shadow casting and statics, allows for a quick turnover of prototyping cycles. The method can be applied both in an office setting, used by designers and architects, and in an extended environment where a group of non-experts can also be involved.

The linking of game design, reality augmentation, and machine learning can facilitate an assembly process that is easily transportable and requires less expert knowledge.

Converging machine sensing and computation with human interaction offers a promising strategy in architectural design, participatory processes, and the engagement of non-experts in the process of becoming. Speculation on real-world case scenarios could be the automated placement using robots of a solid envelope and the local community's subsequent removal of wall blocks until they create a porosity pattern providing the wanted sunlight effects and visual connectivity. Increased access to a broad range of low-priced sensors and a novel generation of robots equipped with sensors that allow for man-machine cooperation will provide new strategies for materializing.

Chapter 8

20.000 BLOCKS

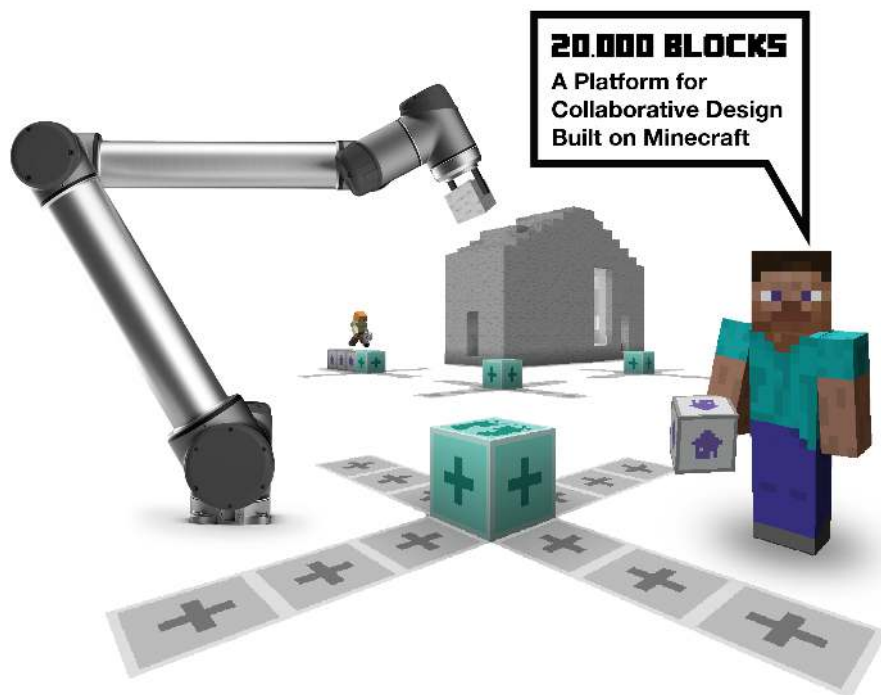


Figure 8.1 – 20.000 BLOCKS, framework illustration. Image credits: the author.

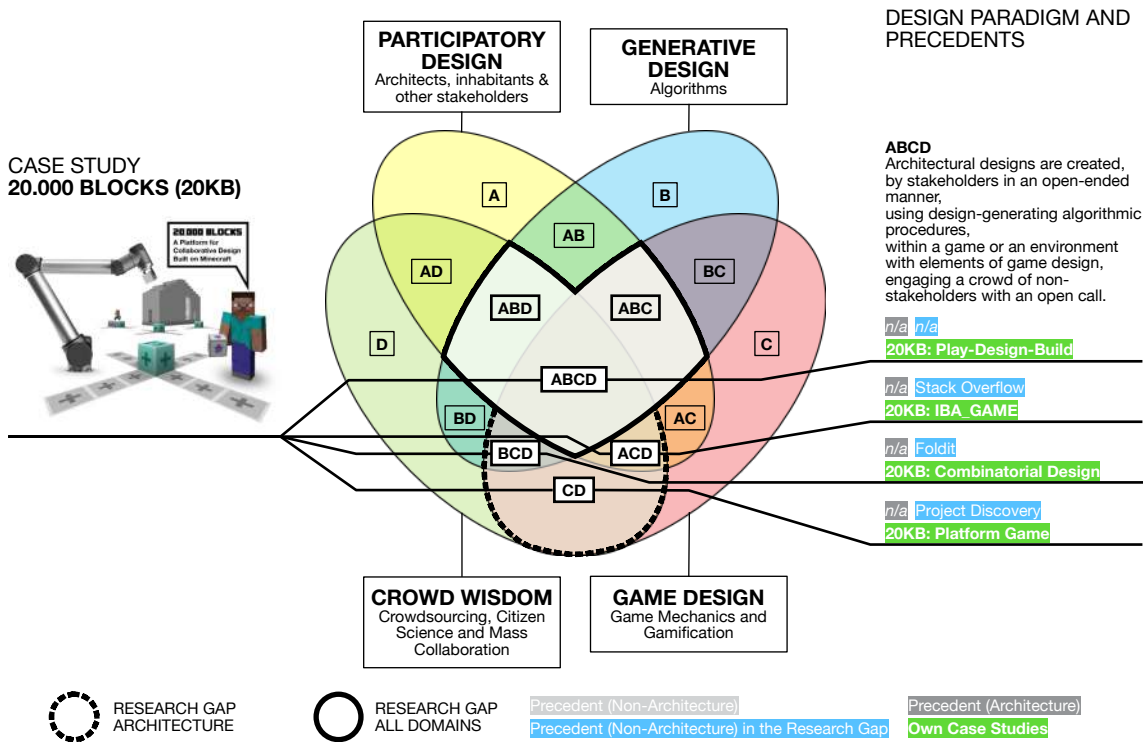


Figure 8.2 – The projects in 20.000 BLOCKS on the map of Design Paradigm. Image credits: the author.

8.1 Design Paradigm

The case study *20.000 BLOCKS* comprises a set of custom-built digital tools and several projects realized within this framework (Figure 8.1). The case study combines a user-friendly game environment, multiplayer gaming, participatory generative design techniques, performance feedback from specialized parametric design tools, and robotic fabrication (Figure 8.3). *20.000 BLOCKS* uses the concept of *playable voxel-shape grammars* to enable and guide anyone to design a building in Minecraft and have a model of it assembled by a 6-axis robot arm, effectively spanning the digital and the physical worlds. The research project was led by me and run at the *Digital Design Unit (DDU)* at Technische Universität Darmstadt from 2015 to 2018. It centered around the question:

Can gameplay mechanics guide groups of non-experts throughout the collaborative creation of architectural designs?

A major finding of the case study is the mechanisms for calibrating the balance of influence on the resulting designs between the Experts and the Players.

For a meaningful inclusion of the non-architect in the design process, the architectural expert knowledge needs to be partially encoded into an agile, open, and generative rule-based system. This requires a modeling environment that combines the ease and popularity of computer games with the control and evaluation tools of Computer-Aided Architectural Design (CAAD) software. The case study *20.000 BLOCKS* presents such a system with three key components:

1. *Guiding rules* — based on playable, voxel-shape grammars and employed to direct the participants towards feasible design solutions.

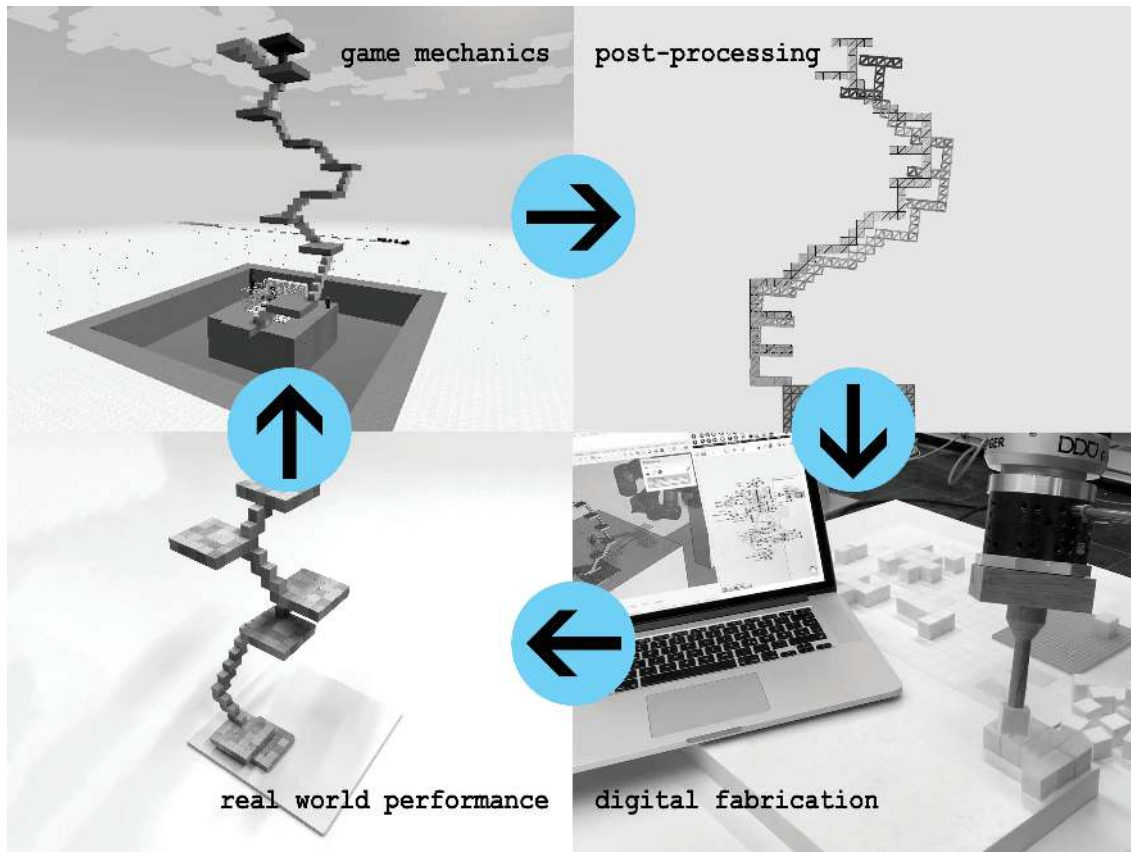


Figure 8.3 – The main algorithmic components of the 20.000 BLOCKS framework. Participatory design is augmented with game mechanics, automated structural analysis and robotic fabrication. Image credits: the author.

2. *Verification routine* — to automatically process the resulting designs on key parameters and performances.
3. *Fast feedback* — to keep the players aware of what they are creating, allowing for decision corrections to happen in short cycles.

Tuning the three components — the guiding rules, the verification routines, and the feedback — influences the outcome of the designs created by participants.

8.2 Implementation and Setup

The *20.000 BLOCKS* technical framework aims to facilitate the prototyping and testing of various building designs in short cycles by online communities. Games are accessible to people of all ages and backgrounds and can process large amounts of data input from many players. Hence the approach seeks to crowdsource within a game environment, mainly focusing on online multiplayer games. On the other hand, automated verification routines are needed to test the reliability of many architectural designs created by the non-trained crowd. Parametric design software takes input from specialists and can do advanced performance analysis and control robotic fabrication.

Therefore at the core of the technical realization of *20.000 BLOCKS* includes a modification of the game Minecraft to support the definition of playable voxel-shape

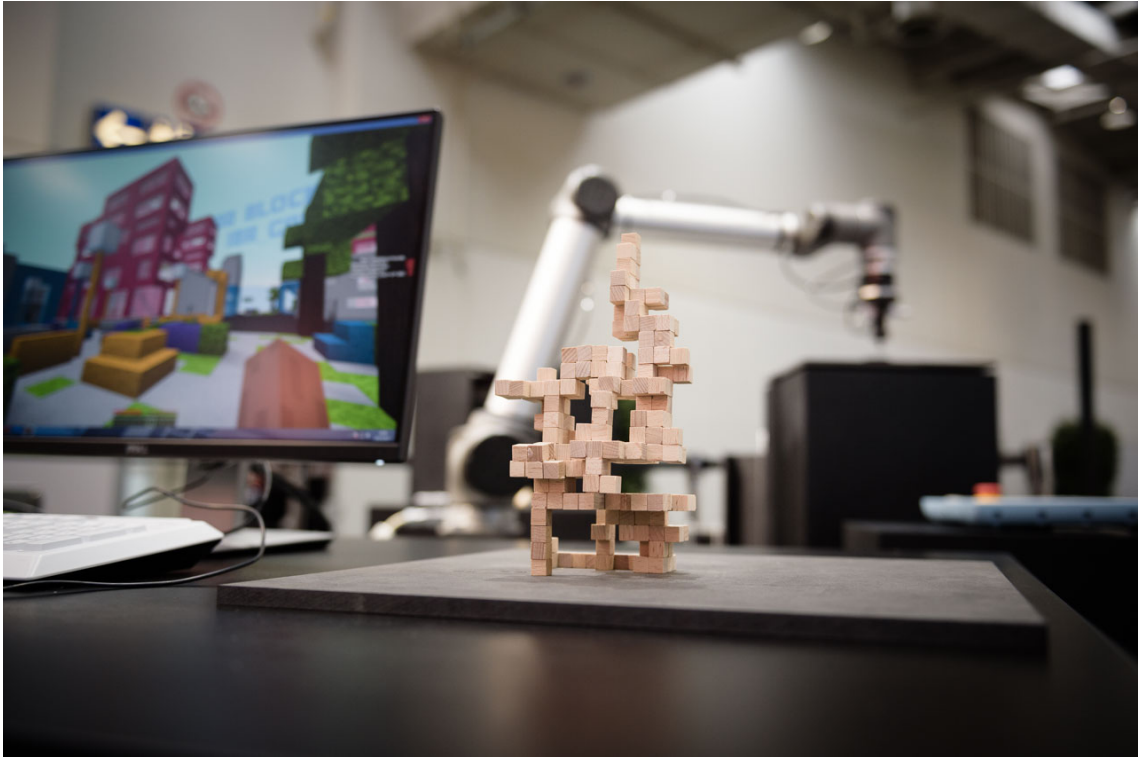


Figure 8.4 – 20.000 BLOCKS Game and Robot shown at CEBIT 2017. Image credits: Futurium, Rottmann 2017.

grammars. In more detail, the framework consists of:

- a modified version of Minecraft using the ComputerCraft MOD and own custom scripts, written in Lua.
- a custom mod for Minecraft that implements commands for getting, exporting, and placing blocks in the Minecraft world to facilitate the exchange with Grasshopper
- a set of Grasshopper components and scripts to read and write 3D data to and from Minecraft worlds
- a pick and place robot routine in Grasshopper using the add-ons Scorpion and later Robots to build models that were imported from Minecraft
- a WebGL visualizer that runs natively in a web browser to visualize models created by players in 20.000 BLOCKS Minecraft world
- a set of sample analysis routines implemented in Grasshopper for analyzing: statics, vistas, etc.
- command-line python scripts that can read from and write 3D data to a Minecraft world

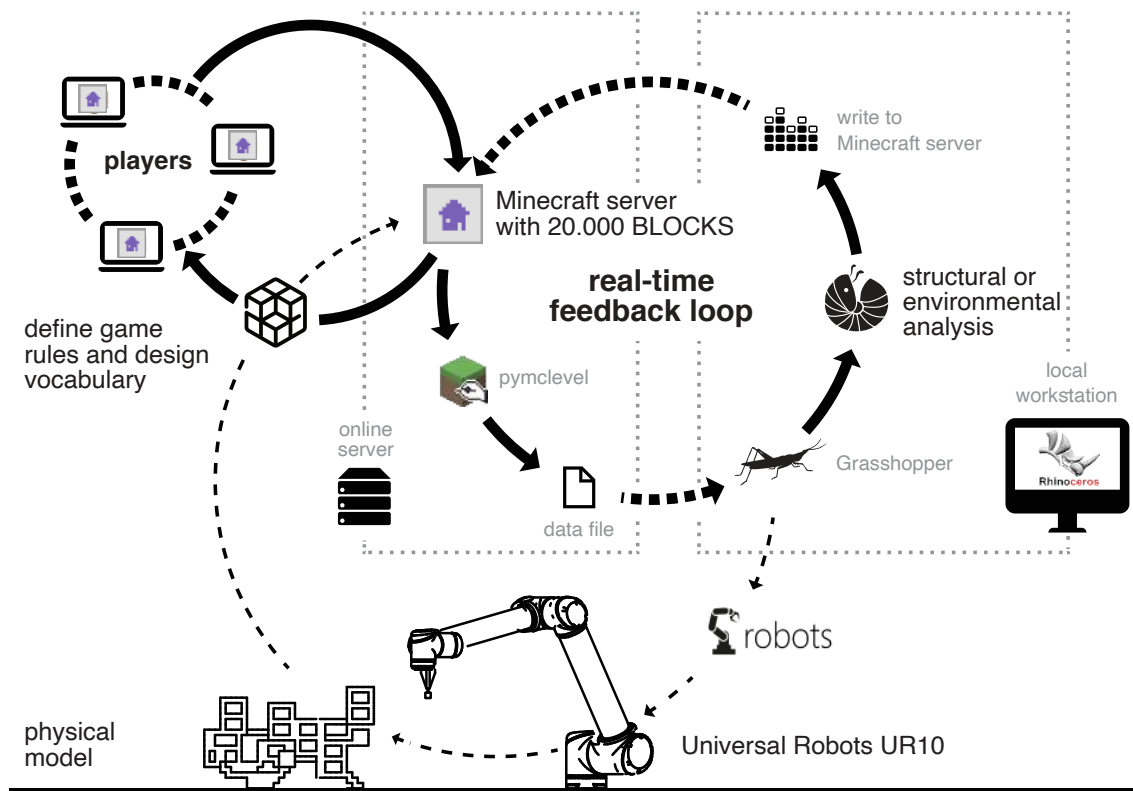


Figure 8.5 – Hardware and software components of the 20.000 BLOCKS framework. It consists of custom scripts for Minecraft that use the concept of playable, voxel, shape grammars, a set of tools for real-time link between the game and structural or environmental verification done in Grasshopper, and a robot fabrication process to build models of the player-created designs. Image credits: the author.

Minecraft

20.000 BLOCKS uses Minecraft, a game with simple graphics and rules. In Minecraft, players build and demolish cubes of 1x1x1 meters. In 2015, when *20.000 BLOCKS* started, more than 40 million people played it (T. Warren 2016). As of May 2020, Minecraft has 126 million monthly active players (Chiang 2020).

Minecraft is a sandbox game where players can choose on their own which goals and adventures to pursue. The game consists of a 3D procedurally generated world where materials are spread for the players to mine. Players can combine various materials to *craft* new objects such as pick-axes, wooden planks, buckets, doors, etc. Each Minecraft world can be loaded on a server and made available for online access to other players offering multiplayer functionality. Minecraft’s rich catalog of materials and objects, plus the ability for in-game scripting, allows the creation of custom maps where the map creator can define a goal or an adventure. The project *20.000 BLOCKS* uses such a custom-made map to define game rules that guide groups of players to create architectural designs.

Minecraft is a good choice for this research because of its abstract, non-photorealistic, voxel world. Fröst and P. Warren 2000 conclude that “low-detail of sketch-like real-time 3D models often promotes creativity and discussion.” Michalatos 2016 confirms this, stating that “the more elaborate and specialized the ontology, the less suitable the software becomes for the early stages of design where ambiguity can be more productive.”

Minecraft’s block size of 1 meter offers a suitable resolution for mass-modeling architectural concepts from the scale of a room up to urban planning. Finer details such as fenestration or furniture are challenging to model in Minecraft but also irrelevant to this research. Instead, the focus is on investigating modes of player engagement to produce schematic architectural designs. In addition, the granularity of Minecraft’s world space but also of the players’ actions facilitates concurrent editing from multiple players (Michalatos 2016). Furthermore, the size of a player’s Minecraft avatar in relation to the shapes they model creates an immersive perception for a one-to-one architectural scale.

I also needed a way to record, document, and observe the creation of designs by participants. The ultimate granular model, according to Michalatos 2016 is the one where “every single click is registered in a database with a timestamp attached to it.”

The games *Half-life* and *Second Life* have been used to conduct similar case studies in architecture (Fröst 2003; Gürsimsek 2012; Segard, Moleta and Moloney 2013). However, these precedents are targeted mostly at architects and focus entirely on the design’s shape, ignoring functional organization. Furthermore, the worlds of *Half-life* and *Second Life* lack the constraints and the regularities of the Minecraft world - a coarse 3D voxel grid, resource system, world-editing mechanics — which are beneficial for my research.

Minecraft has four game modes: survival, creative, adventure, and spectator. *Survival* is the default game mode where players need to scout the world, collect materials, craft tools, and survive hunger and encounters with enemies. *20.000 BLOCKS* requires a custom-defined game logic and does not use this mode. *Creative* is the mode where players have unlimited health and resources and can fly. The impressive Minecraft builds, familiar from the media, are created in this mode. *20.000 BLOCKS* uses this mode for the participants in the role of Experts allowing them to create catalogs of elements for the players to assemble into designs. *Adventure* mode allows strict control over what parts of the world and which materials players can break, collect and place. This makes it suitable to stage the player experience in *20.000 BLOCKS*. By default *20.000 BLOCKS* participants are put in Adventure mode and can only use the materials, catalog elements, and tools defined by the experts who created the particular 20.000 BLOCKS project. The *spectator* mode was used for taking screenshots or time-lapse videos when a game was in progress.

Minecraft modifications for *20.000 BLOCKS*

20.000 BLOCKS changes the rules of the Minecraft world by letting players in the role of Experts create games within the game. Experts can design their catalogs of elements and define the rules for their placement, and the rewards players get when placing them.

The space of a *20.000 BLOCKS* game is shown on Figure 8.6. It consists of a spawn area, a building zone, and a catalog area.

20.000 BLOCKS uses a Minecraft mod called ComputerCraft, which allows the development and running of scripts that can interact with the Minecraft world and the state and positions of players in it. The scripts are written in the programming language Lua. The playable, voxel-shape grammar and game logic for creating

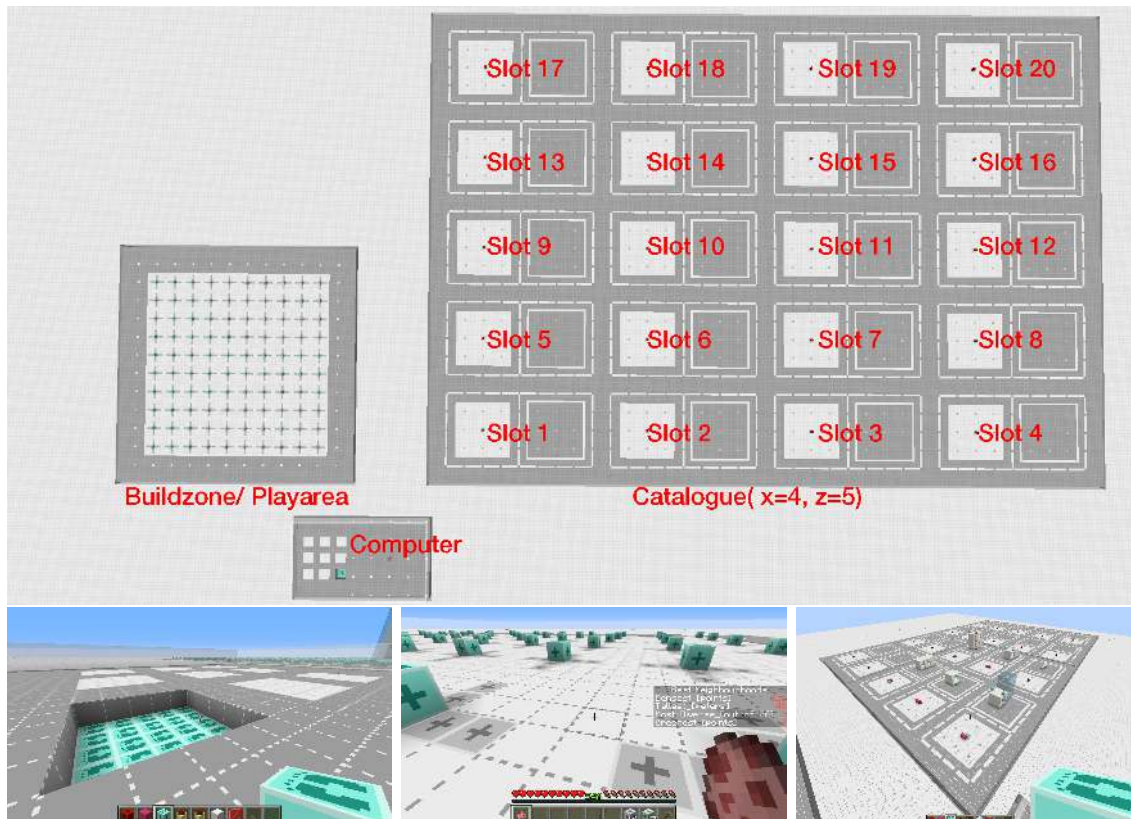


Figure 8.6 – The layout of a 20,000 BLOCKS game space in Minecraft. The spawn zone contains the computer running the *20,000 BLOCKS* scripts and players end up in it upon joining the server. They need to step on the blue pad (bottom left) to start or join a game. A game usually begins as an empty grid of activator blocks (bottom middle) where player can build the key shapes from the catalog in order to place a desired catalog module. The catalog area (bottom right) consist of slots with a white area to model the key shape and a gray area to model the replacing shape. Image credits: the author.



Figure 8.7 – Screenshots of 20,000 BLOCKS running on a ComputerCraft computer in Minecraft. The ComputerCraft mod enabled the easy and iterative prototyping of game logic and voxel-shape grammar logic for *20,000 BLOCKS*. The mod adds in-game computers that can run Lua scripts interacting with the game world and the players. Image credits: the author.

resource economies are implemented in Lua scripts. Figure 8.7 shows the ComputerCraft computer block and its command console running the main *20,000 BLOCKS* script.

A custom-developed Minecraft mod, written in Java, was needed to facilitate the import and export of geometry from a *20,000 BLOCKS* Minecraft world to Grasshopper. Its sole functionality was exporting the 3D model of the world within

a given region defined by the coordinates of its two corners.

20.000 BLOCKS uses the Minecraft trading system to let players decide what resources to buy with their rewards and, as such, exercise choice on what to build next. The trading system is provided by the villager non-player characters (Figure 8.8). For example, for building an element from the catalog, a player can receive one emerald as a reward. They can trade emeralds for various building materials or special items such as water (Figure 8.9). Besides facilitating a game economy, Villagers served a secondary, more architectural purpose. To trade with a villager, players need to walk up to them. This incentivized players to build walkable structures so they could walk back to the place where the villager resides.



Figure 8.8 – Villagers are used in 20.000 BLOCKS as material shops. The bottom part of the left image shows a player next to a villager in the process of trading. The left screenshot also shows a structure in progress built by the player to reach the red platform flying on top. The Villager trading system lets players get new materials in exchange for emeralds or other items received during the game (right). As players need regular access to villagers, they create structures walkable from the ground upwards and back. Image credits: the author.

Material packs a player can buy from the Villager/Trader

Item	COST	GAIN	Notes
A pack of 32 building blocks	1 emerald	30 blocks	this is the baseline
1 block of water	13 emeralds	5 water buckets	Water and garden items should not be attainable from the quantitative rewarding only. Only if a player builds a structure then the reward should be enough to buy those items.
1 block of dirt	13 emeralds	10 dirt blocks	
1x seeds	13 emeralds	10 seeds	

Figure 8.9 – Balancing the material costs in the Platfrom game. The consideration when deciding what costs what for the players in the villager trade shop. Image credits: the author.

To make it as easy as possible to get started with *20.000 BLOCKS*, all required mods to run *20.000 BLOCKS* in Minecraft were packed in a Modpack and distributed over TechnicLauncher, the standard platform for Minecraft customizations. A detailed list of instructions on setting up the Minecraft components of *20.000 BLOCKS* is provided in *Appendix B: Getting Started with 20.000 BLOCKS in Minecraft*.

Computational feedback and verification routines

At the heart of the project lies a real-time link between the online Minecraft server and analysis routines in the parametric design software Grasshopper (??). I used

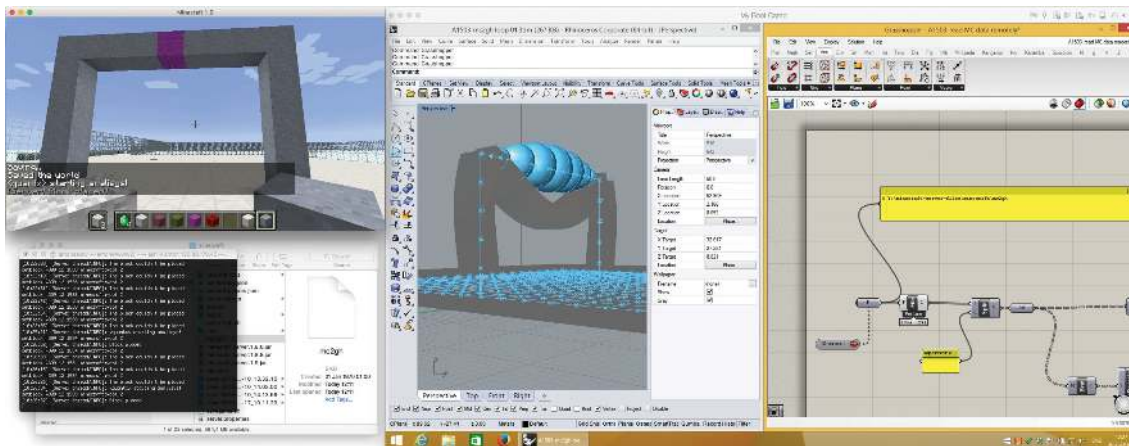


Figure 8.10 – Structural verification of Minecraft shapes in real-time. The block marked in magenta is a new secondary mission for the players. A player builds the gate in Minecraft (top left). The tracking *pymclevel* script (bottom left) captures the modified world and passes the new blocks coordinates to Grasshopper. Grasshopper creates a Millipede structural model and determines the blocks with maximum deformation. If the deformation is more than a specified threshold, Grasshopper sends a command over SSH to the Minecraft server to change the material of the block to magenta. Image credits: the author.

the open-source Python library *pymclevel* by Vierra 2014 to get geometry from Minecraft to Rhino/Grasshopper, which is then reconstructed in a format suitable for structural performance evaluation via the add-on *Millipede* by Sawako and Panagiotis 2014. The results from this analysis are fed back to the Minecraft server, either by creating new blocks or changing the material/color of existing ones. For this, I developed my own Grasshopper components that can send commands to the server over SSH protocol¹.

Along with the shape grammar, this information is used to help to guide the players' actions in the game world to create 'viable' structures.

Achievement system and progress feedback

Experts can use the achievement system to define goals and rewards to have players engaged and aware of what to do next and how well they are progressing in a given *20.000 BLOCKS* game. Achievements can be of the following kinds:

- reach a specified location marked in a special material,
- build N copies of a particular element from the catalog,
- build a combination of two elements next to each other,
- collect X amount of points,
- support a structurally weak spot.

Rewards can be either points or material blocks and items.

20.000 BLOCKS can provide on-the-fly feedback to the players in several ways: posting messages to the in-game chat, using scoreboards, showing particle effects, and changing the material of blocks in the game world.

¹<https://unix.stackexchange.com/questions/13953/sending-text-input-to-a-detached-screen>

In the *chat* console, messages are displayed to inform players what to do when they join the server. While playing, if a player successfully builds a new element out of the vocabulary, a message tells everyone (Figure 8.11). Goal achievements are also broadcast in this fashion.

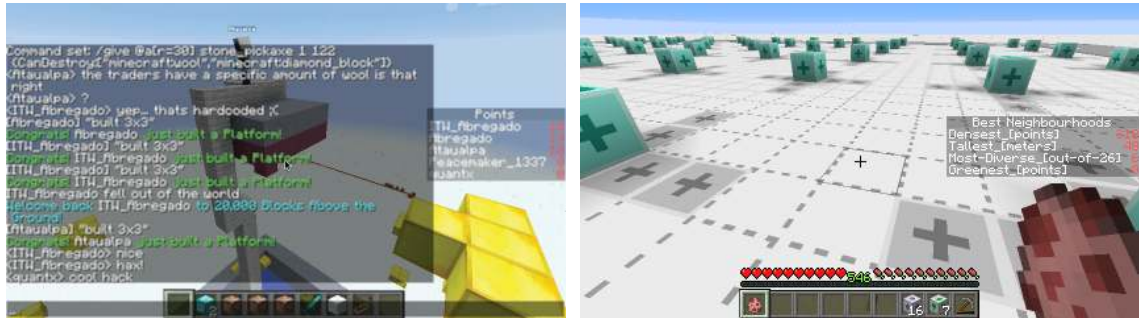


Figure 8.11 – Minecraft systems used for player feedback in 20.000 BLOCKS. The game chat is used to communicate to players on their's and other players' progress. Customizable scoreboards, right part of the left screenshot, can track players progress in points on a chosen metric. Scoreboards can also track the highest achievement on a given metric to incentive players to beat the record as shown on the right screenshot. Image credits: the author.

Scoreboards can be defined by the participants in the Expert role. They track the points that players have received, for example, for building a particular element of the catalog (Figure 8.11). Scoreboards can also track the top score on a given metric to incentivize players to beat it. For example, in the *IBA_GAME* project, scoreboards were used to track how green, dense, tall, and diverse the neighborhood the players are building is.

Particle effects are used to visually attract players to critical positions in the game world. The most important of these is to inform a player that the system is verifying a newly built vocabulary element.

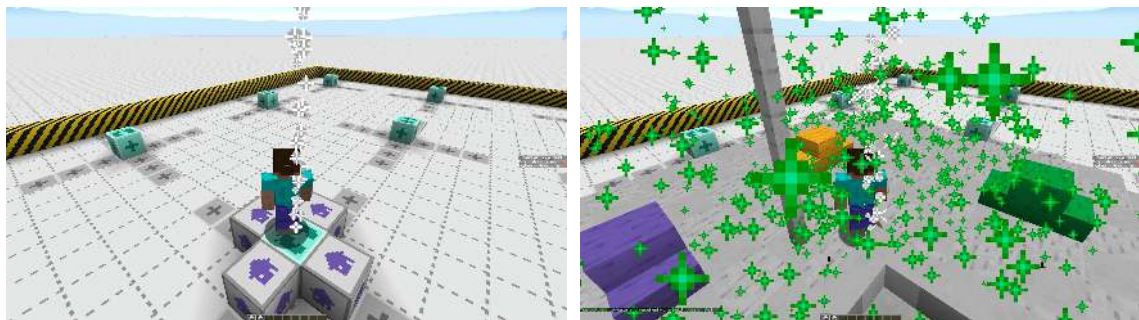


Figure 8.12 – Shape activation in 20.000 BLOCKS. Two screenshots take a second apart. The left one shows a player activating a key shape from the catalog and the particle effect letting them know that the shape detection is in progress. The right screenshot shows the placed element from the catalog and the green particle effect for success. Image credits: the author.

Changing the material of a block in the game world can be a means to give a location-based mission to players. One example is having players aim to step on a platform of a given material to collect points or win the game (Figure 8.8). The players are then incentivized to build their way to this platform.

Additionally, after running a verification routine, a structurally weak spot could be marked in Minecraft and turned into an optional, secondary mission. This would award them extra points for resolving the structural problem (Figure 8.10).

Robotic process

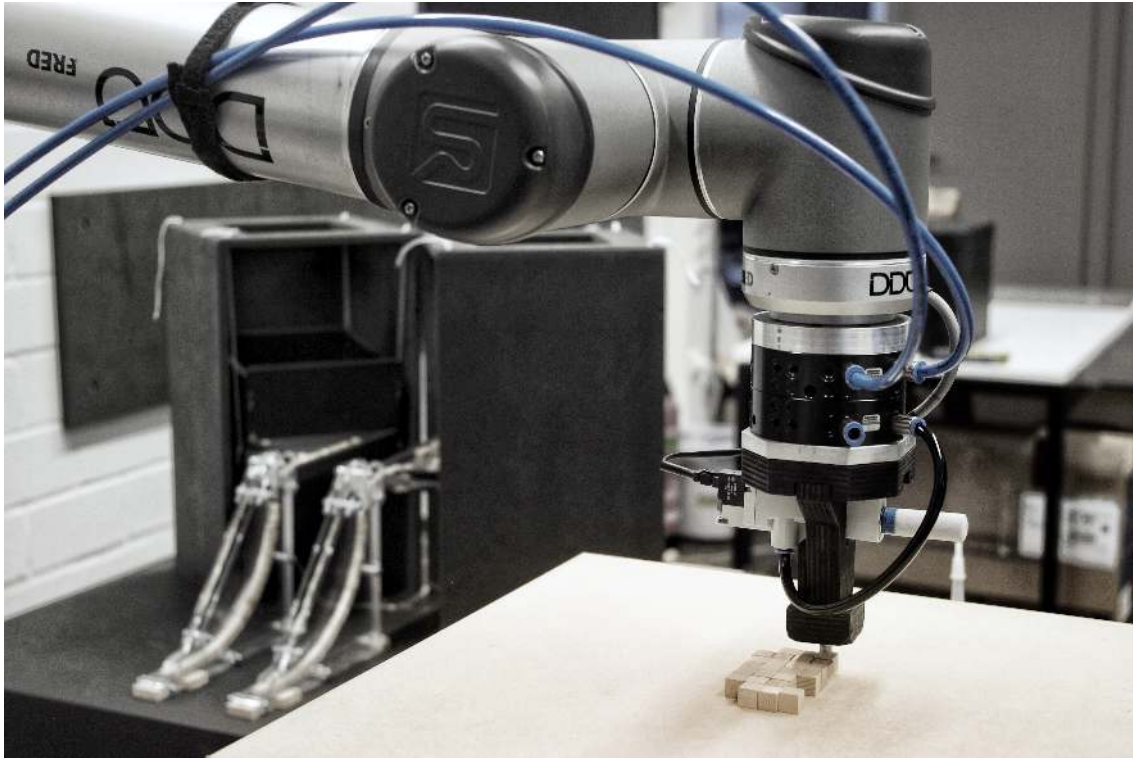


Figure 8.13 – The robotic process in 20.000 BLOCKS. A close up photo of the robot gripper placing a block. Image credits: Rui Nong 2018.

I developed a pick-and-place robotic process for *20.000 BLOCKS* (Figure 8.13). The process served as a proof-of-concept that player-created designs, when defined with *playable, voxel-shape grammars*, would require very little post-processing to be automatically assembled. This made the design system fabrication-aware.

The robot fabrication process produced models of the player-created design on a scale of 1:100 or 1:50. The models had a twofold purpose. First, it is used to verify the stability and constructability of a game result. The second purpose of the models is to visually and materially represent the player-created designs as a form of feedback for both Players and Experts.

The full robot setup (Figure 8.14 and Figure 8.15) is conceptualized as an exhibition installation where all components of the process are exposed and labeled for maximum educational benefit of the visitors (Figure 8.16). The robot setup was exhibited together with *20.000 BLOCKS* play stations on several occasions such as the fair CEBIT (Figure 8.17) and info events of the Technical University of Darmstadt (Figure 8.18).

The assembly routine is shown on Figure 8.19. It starts by feeding the point information for each Minecraft block to a Universal Robots UR10 robot arm. It grabs wooden cubes with a vacuum gripper (Figure 8.20 and Figure 8.21), pushes them into the glue and positions them on the correct spot according to the digital model (Figure 8.22). The block dispenser consists of a vibrating tray that feeds blocks into a slide where the robot picks each cube. The glue station consists of a vertically positioned syringe. It is squeezed by a spindle rotated by a stepper motor controlled with an Arduino board.

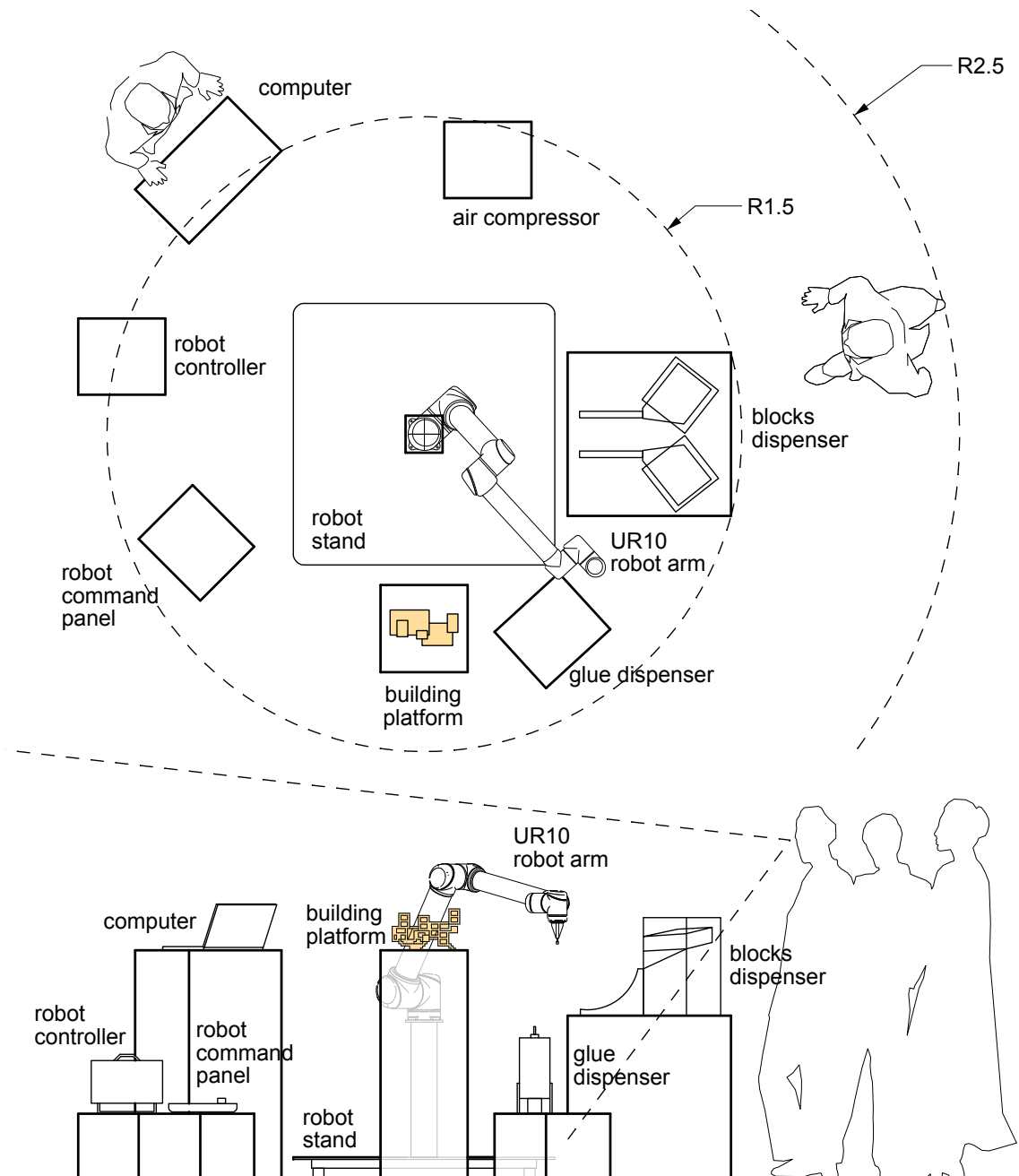


Figure 8.14 – The plan and side view of the latest iteration of the robot process setup in 20.000 BLOCKS. Image credits: the author.

The path planning and robot control is done in Grasshopper with the add-on Robots by Soler 2021 (Figure 8.23). The signals needed to control the Arduino board for dispensing glue are also sent via Robots and the UR10 robot controller. A laser sensor is used to check whether the robot has grabbed a block successfully. On the rare occasion when there is no block at the gripper, the process stops and waits for the human operator. This avoids faulty builds with missing blocks.

If a cube is cantilevering, the robot places supporting, non-glued cubes underneath it (Figure 8.24). These are removed by shaking the model after it is finished (Figure 8.25). To distinguish between supporting blocks and model blocks, we used blocks in two colors, natural wood and black.



Figure 8.15 – The robot setup. Image credits: Tabita Cargnel 2016.

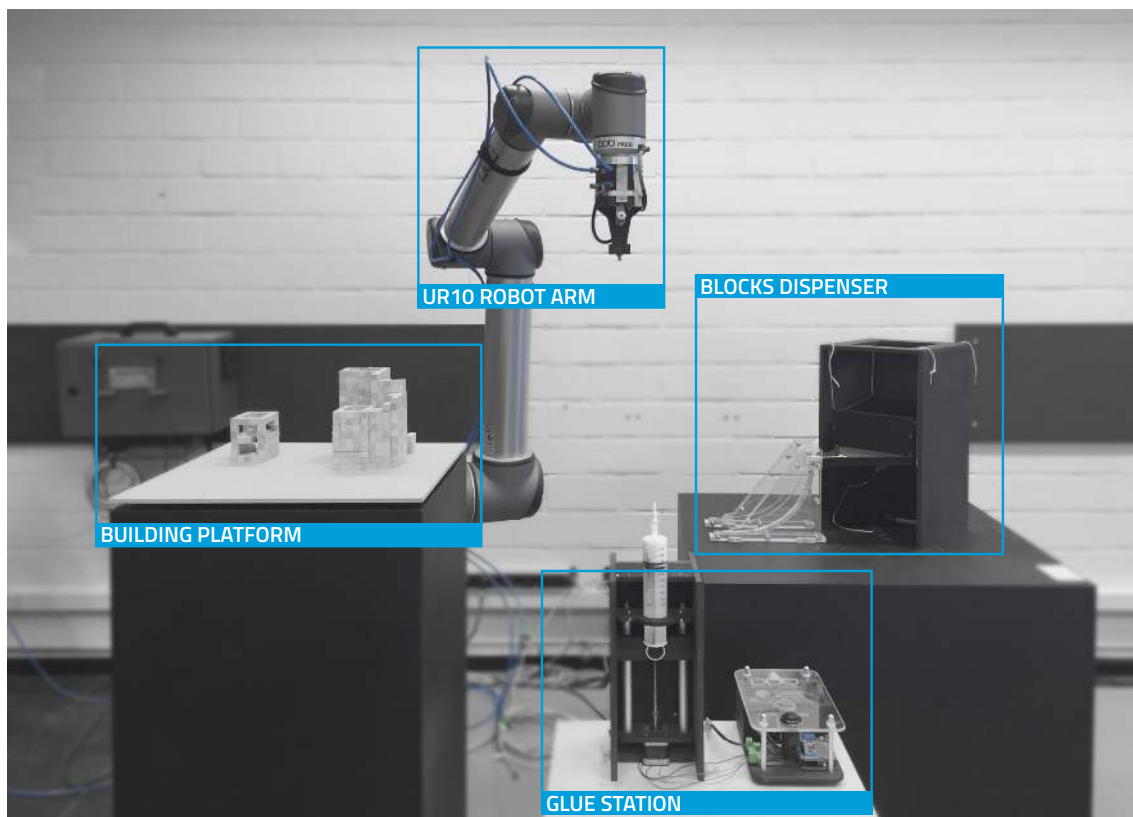


Figure 8.16 – The components of the robot process. Image credits: Rui Nong 2018.



Figure 8.17 – The 20.000 BLOCKS robot installation at CEBIT 2017. Image credits: Futurium, Rottmann 2017.



Figure 8.18 – The robot setup of 20.000 BLOCKS installed at an info event of Technical University Darmstadt. Four play stations are seen in the background. Image credits: Tabita Cargnel 2016.

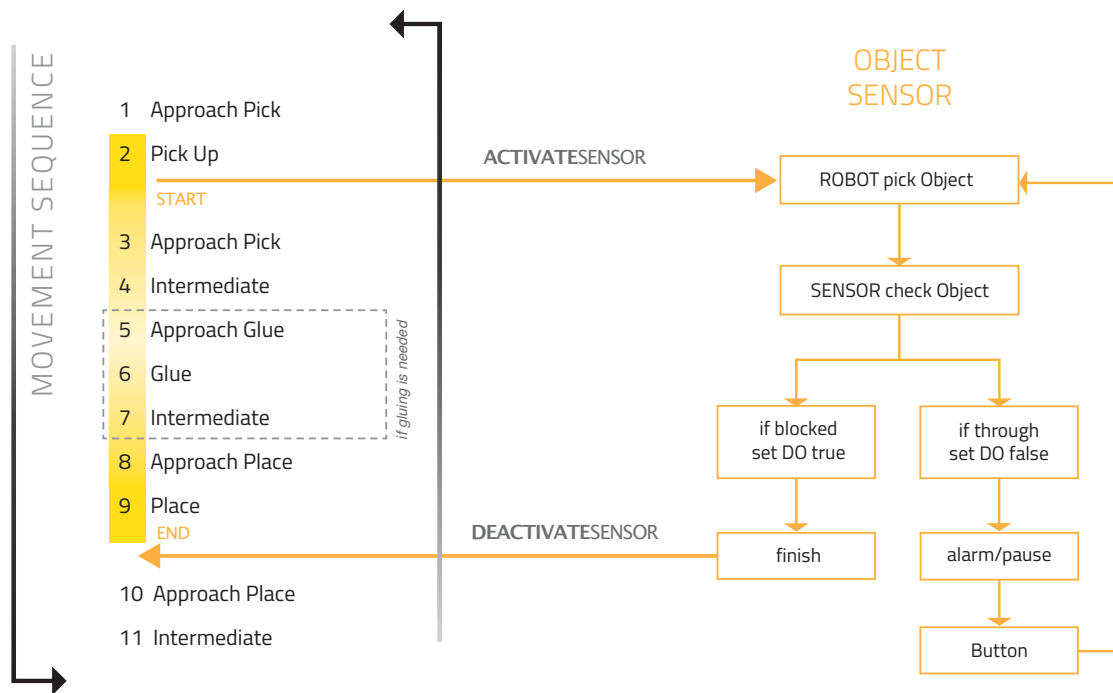


Figure 8.19 – The sequence of robot actions in the pick-and-place routine used to build the models in 20.000 BLOCKS. Image credits: Rui Nong 2018.

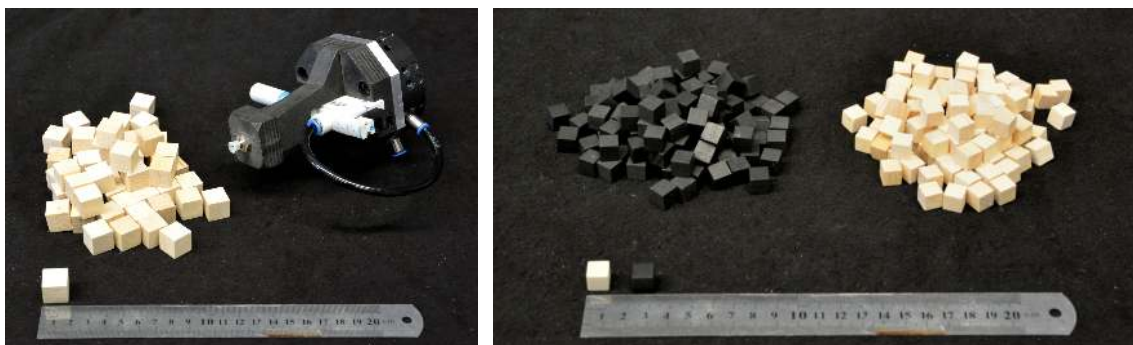


Figure 8.20 – The robot gripper and the three types of cubes used to produce the robotically assembled models in 20.000 BLOCKS. Natural wood 1cm and 2cm big and black 1cm. Image credits: the author.

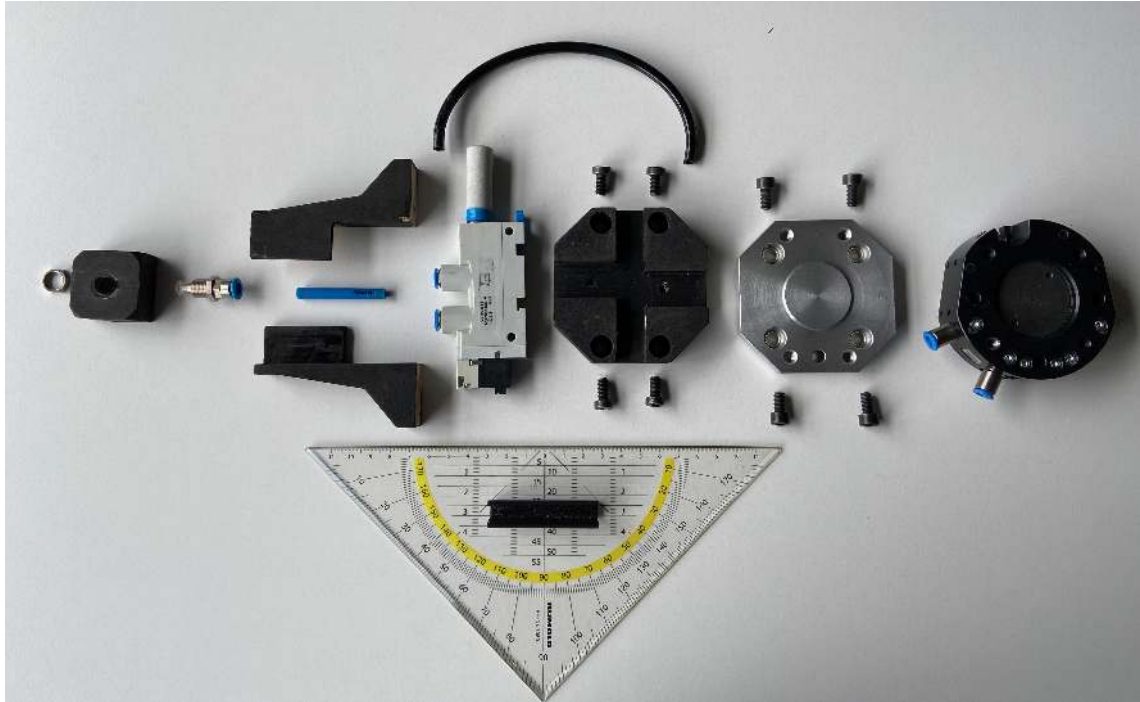


Figure 8.21 – The components making up the custom-designed suction gripper in 20.000 BLOCKS. Image credits: the author.



Figure 8.22 – Close-up photos of the robot process components. Left: the robot picking up a cube from the slide of the block dispenser. Right: the robot dipping the cube on the glue-dispensing syringe. Image credits: Rui Nong 2018.

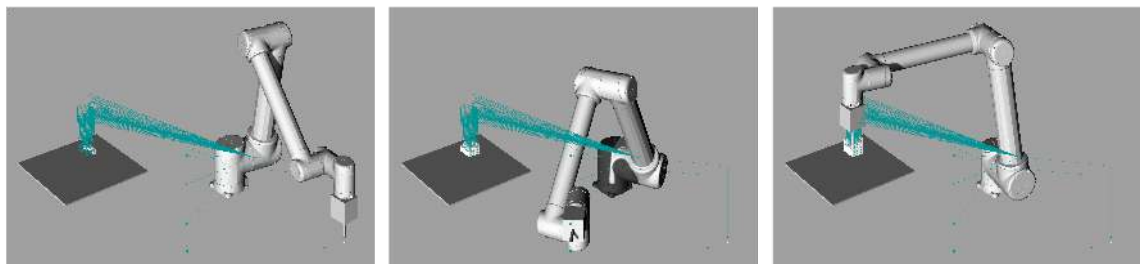


Figure 8.23 – The simulation of the path planning using the Robots add-on for Grasshopper. Left: pick position, Middle: glue position, Right: place position. The paths for the whole model are shown in blue-green color. Image credits: the author.

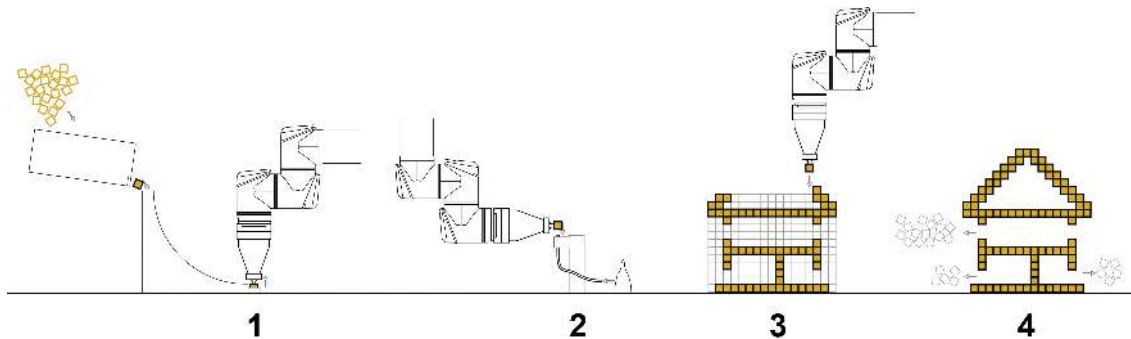


Figure 8.24 – The robotic fabrication process in 20.000 BLOCKS. From left to right: cubes in a vibration tray are falling into a slide from where the robot picks them up one by one; applying a glue drop to the bottom, left and front sides of the cube, if the cube is from the model; placing the cube in the model; shaking off the the supporting cubes. Image credits: Jörg Hartmann 2016.



Figure 8.25 – Models produced by the robot. Left: removing of the supporting cubes of a finished model. Right a set of test models assembled by the robot. Image credits: the author, Rui Nong.

Online visualizer

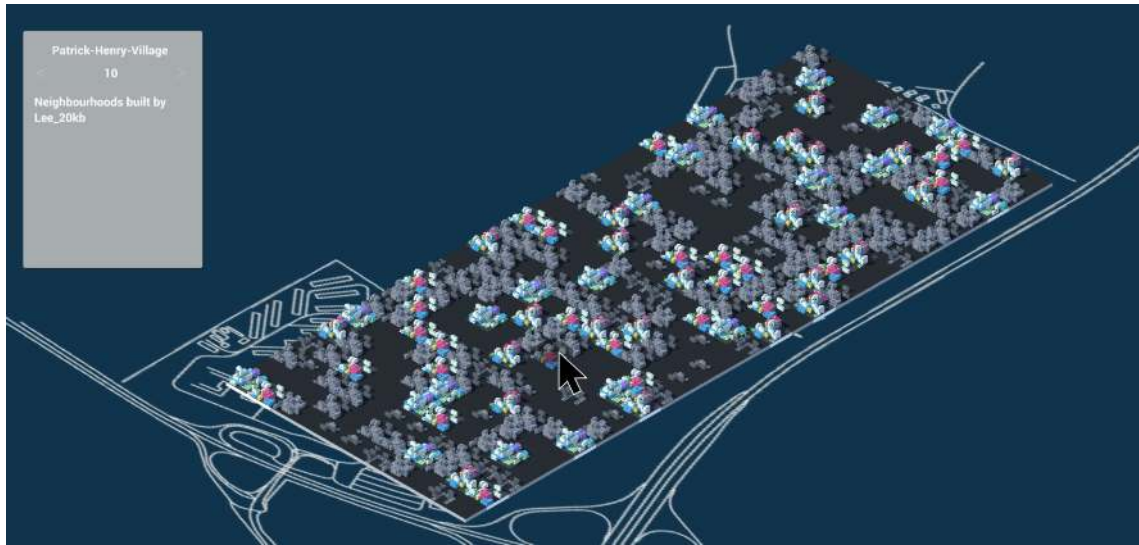


Figure 8.26 – The online visualizer set to campaign view. Showing the designs created by player *Lee_20kb* in the *IBA_GAME*. Image credits: the author.

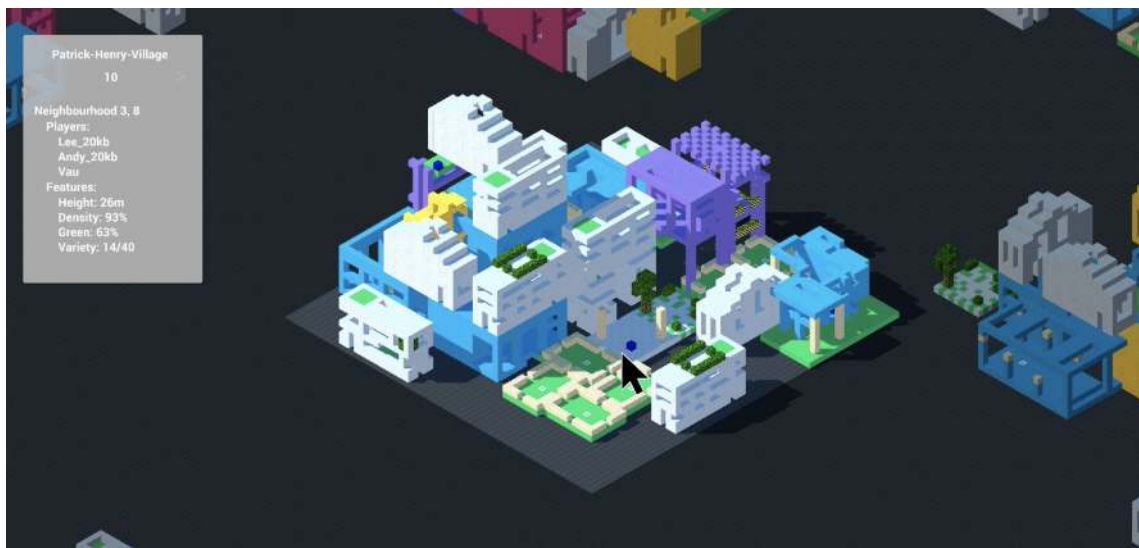


Figure 8.27 – The design view of the online visualizer. Showing a design from the *IBA_GAME* campaign created by three players and with a height of 26 meters. Image credits: the author.

The purpose of the visualizer is to enable anyone to browse the player-created designs within a given *20.000 BLOCKS* project even if they do not own a copy of Minecraft or access to Grasshopper. The visualizer runs in a browser using WebGL technology to display 3D models. The visualizer is realized using the game engine Unity. The recorded designs for a project can be viewed in three modes: *campaign* view, *design* view, and *element* view.

The *campaign* view displays all designs created in the given project as a matrix (Figure 8.26). The user can filter out designs created by the same player in this view.

The *design* view lets the user browse a specific design created in that project

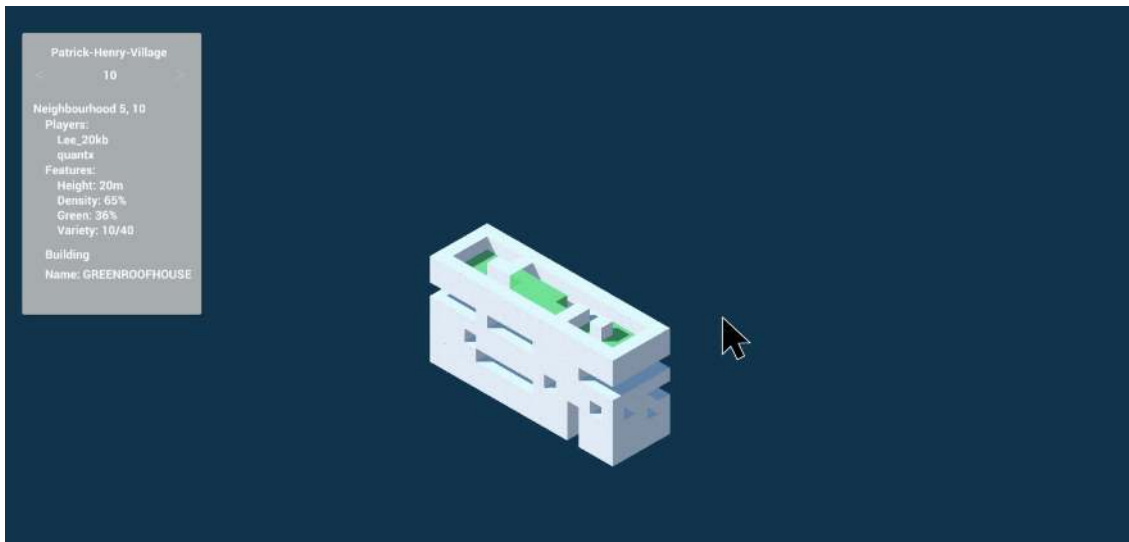


Figure 8.28 – The element view of the online visualizer. Showing an element from the type *Green Roof House* from a design created by the players *Lee_20kb* and *quantx*. Image credits: the author.

(Figure 8.27). The user can see which players created it and how well the design performed on the project-specific metrics.

The *element* view lets the user zoom in further into a single catalog element placed in the design. The element title and the player who placed the element are named.

8.3 Machine Agency — Playable Voxel-Shape Grammars

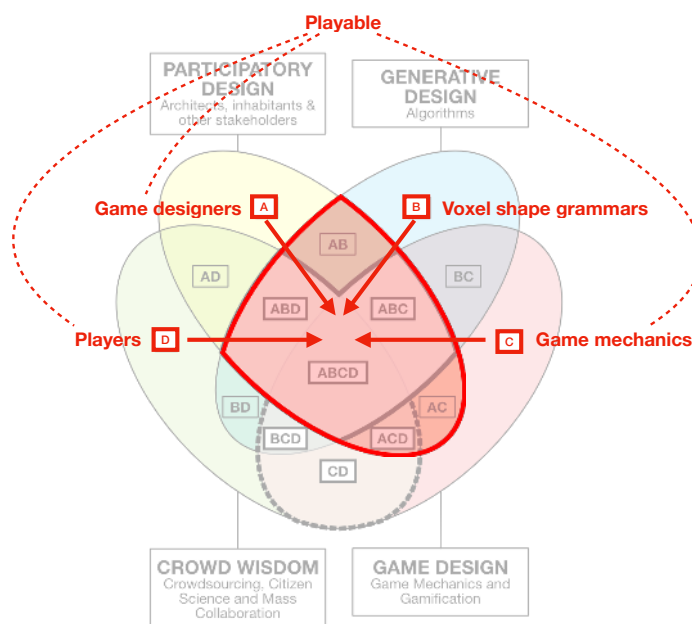


Figure 8.29 – Components of the playable, voxel-shape grammars. The novel generative technique enables the exploration of the intersection between the four fields. Image credits: the author.

In this section, I introduce a novel technique, namely *playable, voxel-shape grammars*. It facilitates the exploration of both main research questions in this work. First, it enables the testing of the design paradigms at the intersection of *participatory design*, *generative design*, *game design* and *crowd wisdom* (Figure 8.29). And second, the technique allows prototyping and exploring various design tools and tasks for both expert and non-expert user roles.

The case study *20.000 BLOCKS* applies playable, voxel-shape grammars at the scale of a building and the urban scale. It shows that the technique has an enabling effect upon both experts and non-experts by letting them interactively and immersively explore design options and learn by making.

At its core, *playable, voxel-shape grammars* is a generative design technique from the family of shape grammars, extended with game mechanics as possible grammar rules. Its implementation in a 3D game environment enables third-party contributors and stakeholders acting as players to create architectural designs by combining elements from a predefined vocabulary. At the same time, expert stakeholders, i.e., architects, can freely specify the elements of the vocabulary, the combination rules, and guiding game mechanics with no required programming or other special skills.

The use of a *vocabulary* to facilitate the encoding of architectural design principles emerged early on in the process of my work and has evolved over several implementations and test iterations. It borrows from the principles used in the various design systems reviewed in section 3.1. Stiny 1980a defines *vocabulary* in the context of design as “a limited set of shapes, no two of which are similar.” The *shape grammar* generative formalism can create rules-based design systems around vocabularies and, as such, capture design intentions and expert knowledge. See subsection 3.3.3 for detailed account on Shape Grammars state-of-the-art.



Figure 8.30 – Voxel shapes in the language defined by the voxel-shape grammar shown in Figure 8.35. As experienced by players using the grammar in Minecraft. In the foreground, an activation state for rule 1. Image credits: the author.

This section presents:

- a definition of playable, voxel-shape grammars (subsection 8.3.2);
- three types of guiding mechanisms that experts could use to control the play-driven generative process (subsection 8.3.4).

8.3.1 Problems with shape grammars

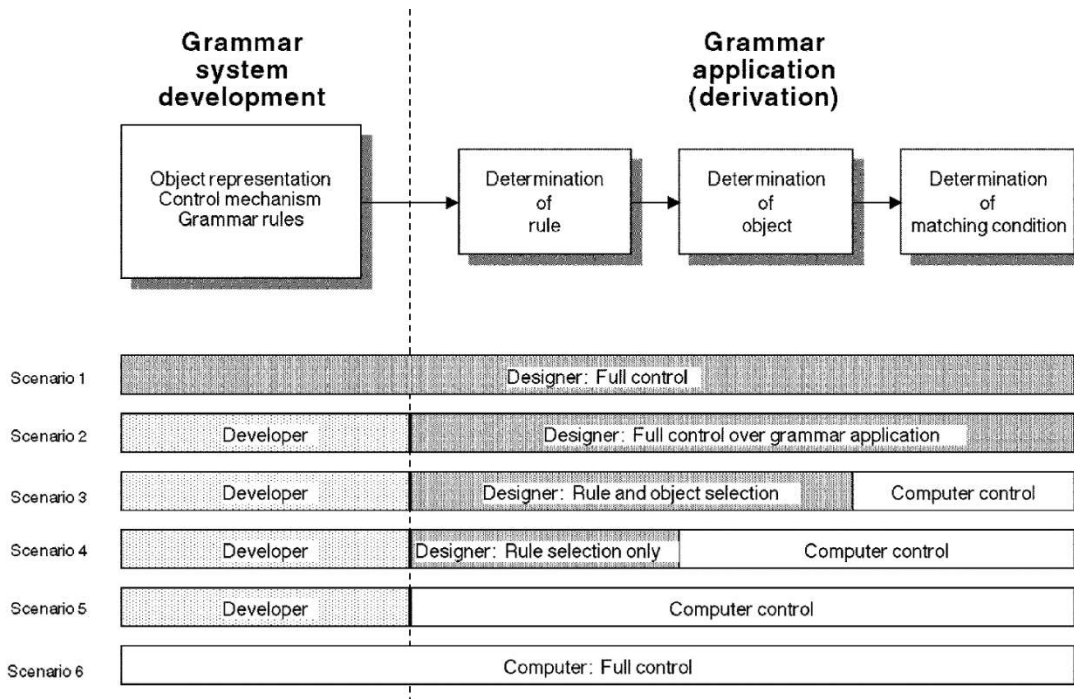


Figure 8.31 – Steps in the development and application of a grammar. Image credits: Chase 2002.

Shape grammars have been successfully used to encode aspects of expert architectural knowledge, and thus help their users, primarily designers, to explore a large set of design solutions or quickly generate two-dimensional representations of designs with desired formal qualities (Chase 2002; Duarte 2001; Müller, Wonka, et al. 2006; Stiny and Mitchell 1978; Stiny and Gips 1971). The application of shape grammars consists of mainly two activities (Figure 8.31):

1. *Specification* — designing the grammar, which consists of shapes and rules;
2. *Derivation* — iterating multiple times through the rules to generate designs.

The second activity maintains the usefulness of shape grammars as a design tool, subject to their implementation as automated systems run by a computer (Ruiz-Montiel et al. 2013). There are three main challenges in using them as design tools:

1. shape grammars are difficult to implement as an automated computational design technique (J. P. McCormack and Cagan 2002, 2006);
2. shape grammars are difficult to specify in a computer environment (Chase 2010; Gips 1999);
3. computer-automated derivation of designs from shape grammars is difficult to explore and control (Wonka et al. 2003).

Implementation problem

The implementation problem stems from the high computational complexity in matching the currently created shapes in the derived composition with the left-hand shapes from the grammar rules (J. P. McCormack and Cagan 2006; Wonka et al. 2003; Wortmann and Stouffs 2018). The more rules are aggregated into a derived design, the more computationally expensive it becomes to determine which grammar rules to apply next.

Symbolic grammars such as L-Systems (Parish and Müller 2001) and Graph Grammars (J. Heisserman 1994), as well as a symbolic representation of Shape Grammars as in Split Grammars (Wonka et al. 2003) and CGA (Müller, Wonka, et al. 2006), solve the matching by representing grammar systems in a machine-readable format. However, they lose the ability for shape grammar specification in a graphic, human-friendly way as in the original, albeit manual, specification of shape grammars (Stiny and Mitchell 1978; Stiny and Gips 1971).

Set grammars, first defined by Stiny 1982, are a candidate solution that is both human- and machine-friendly. Set grammars are a variation of shape grammars where the vocabulary is finite and predefined, as are the spatial relations between the shapes of the vocabulary, which form the basis for the grammar rules (Stiny 1980a, 1982). Set Grammars have a lower complexity in the rule-matching step as they do not have scalar shifts and partial shapes (Figure 8.32). An example of computer implementation of set grammars is the work by Andrea Rossi on the Grasshopper add-on WASP for discrete design (Rossi and Tessmann 2017d, 2018). However, Rossi’s implementation still requires professional 3D modeling software (Rhino) to specify the vocabulary and a symbolic specification of the rules as strings. This makes it unsuitable for this research.

Specification problem

An essential aspect of a successful shape grammar implementation is that the users can model the shapes and rules themselves. Usually, shape grammars have been defined in a closed, expert coding environment with an interface defined solely for the shape grammar (Chase 2010; Gips 1999). Therefore, a potential new user of the shape grammar cannot rely on computer skills they already have and must learn new, advanced ones.

The use of generic, accessible, and easy-to-use 3D-modeling environments to implement shape grammars has not been well explored. The following 21-year-old quote from Knight 1999 states the problem rather well, and unfortunately, not much has happened since then to address it:

Currently, though, few computer implementations are practicable for students or practitioners. Most do not have interfaces that make them easy for nonprogrammers to use. More efforts have gone to computational problems than to interface ones. Implementations of simple, restricted grammars that are visual and require only graphic, nonsymbolic, nonnumerical input are needed. (Knight 1999)

Furthermore, shape grammars have been mostly vector-based (Stiny 1980b; Woodbury 2016), and the shapes generated with grammars are only notational graphs or abstract 2D compositions (Duarte 2005; Martin 2006; Stiny and Mitchell

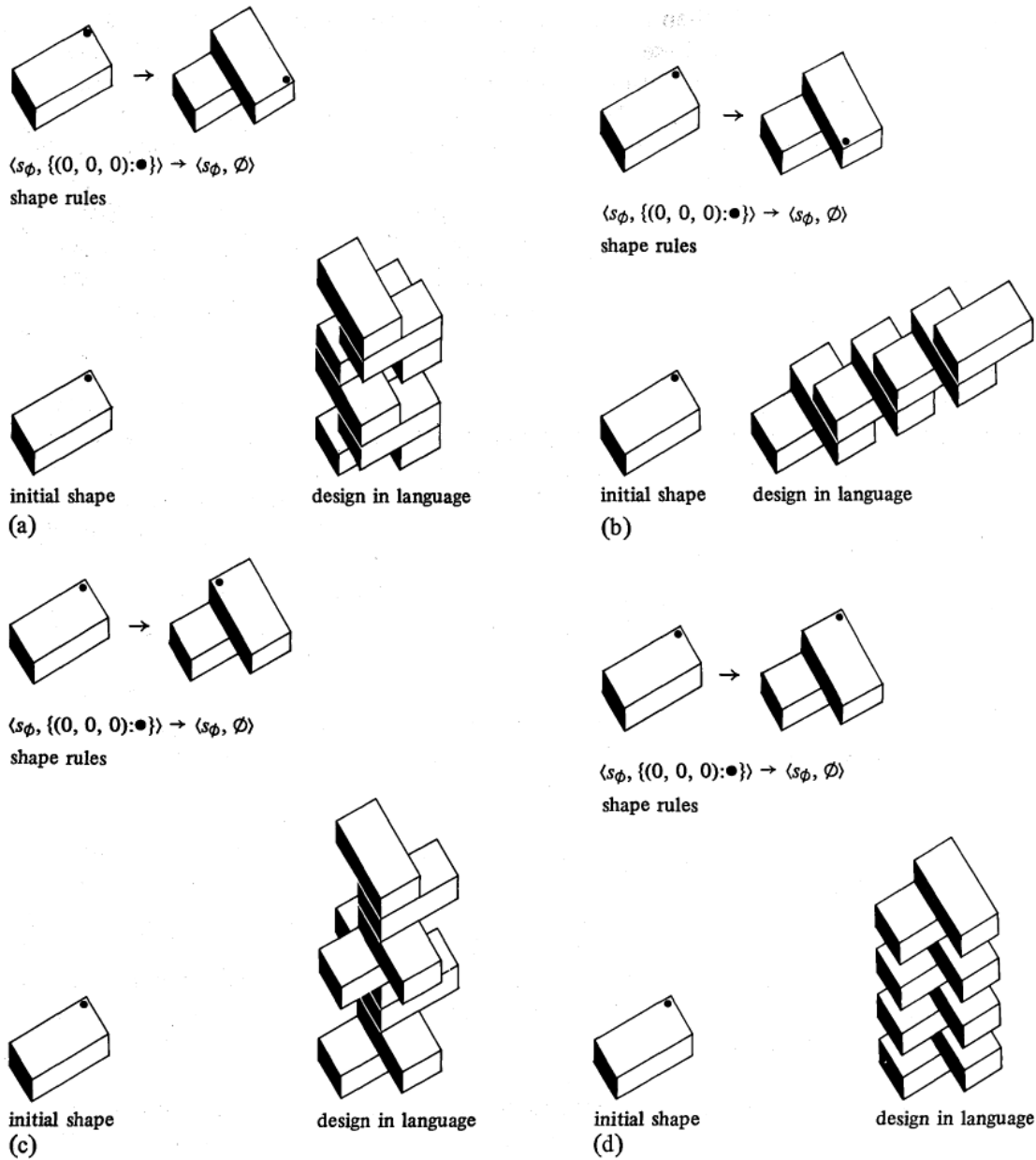


Figure 8.32 – Combinatorial design with shape grammars. Four examples of a set grammar by Stiny 1980a. The geometry of the replacing rule in all four is the same. The variation in the languages is achieved by varying the location of the label (the dot) in the right-hand shape in the rule. Image credits: Stiny 1980a.

1978). I argue that this makes them too abstract for the non-expert and requires drafting or 3D modeling skills to specify them.

Playable, voxel-shape grammars, on the other hand, allow the specification of the vocabulary as well as the replacing rules of the grammar entirely graphically, i.e., geometrically, in any voxel-space 3D environment. section 8.2 provides details on the implementation in the game of Minecraft used in my work.

Derivation problem

Most existing research on shape grammars uses a manual approach to derive designs from a shape grammar (J. P. McCormack and Cagan 2002, 2006; Stiny 1982). But,

as Ruiz-Montiel et al. 2013 state, “pencil-and-paper execution might be tedious and therefore useless for the sake of discovering many new solutions.”

Only few precedents use computer-automation approach to the derivation phase (Gips 1999; Müller, Wonka, et al. 2006; Ruiz-Montiel et al. 2013; Wonka et al. 2003). However, they often aim to fully automate the derivation process and leave little, mostly global control, to the designer. The user cannot influence the result while the process runs and is directly presented with an outcome. Togelius, Yannakakis, et al. 2011 call this a *generate-and-test* approach to generative design. The user is presented with candidate solutions to evaluate or rank. However this leads to choice fatigue and inconsistent selection (Brintrup, Ramsden and Tiwari 2007; Togelius, Yannakakis, et al. 2011).

As I argued in section 3.3, a *constructive* approach is preferable in generative design workflows where human participation is desired. The *constructive* approach consists of making sure a design is good as it is being constructed (Togelius, Yannakakis, et al. 2011).

The playable voxel-shape grammars method uses a combination of game design and feedback mechanisms to facilitate a navigable derivation process using the *constructive* approach.

Precedents

The use of shape grammars in voxel space has only been previously explored on two occasions. Crumley, Marais and Gain 2012 presented an extension to shape grammars called voxel-space shape grammars, in which the shapes and rules are defined in vector form and only the generated shapes — after the iteration is complete — are voxelized in a post-processing step. A more appropriate naming for their method would be voxelized, vector-shape grammars. Therefore, our method is still novel, as the entire grammar is defined and iterated in voxel space.

S. Friedman and Stamos 2013 introduce the notion of a voxel grammar in their work on procedural tree generation. However, their grammar definition is not visual but numerical, and as such, lacks many of the advantages of the original shape grammar formalism, which “allows for algorithms to be defined directly in terms of [...] shapes” (Stiny 1980b). Our implementation here suggests that the entire voxel-shape grammar (shapes and rules) is defined in a visual, user-friendly way.

8.3.2 Definition

Voxel shape

Stiny 1980b defines a vector shape as follows:

A shape is a limited arrangement of straight lines defined in a cartesian coordinate system with real axes and an associated euclidean metric.

Similarly, let us define voxel shape as a limited arrangement of voxels in a discrete voxel grid and an associated euclidian metric (Figure 8.33).

A voxel-shape will define all further terms we find in *Introduction to Shape and Shape Grammars* by Stiny 1980b but for the discrete voxel space instead of continuous cartesian vector space. For example, a scale operation on a voxel shape needs to have an integer number for the scale factor — for example, 2x — to turn one

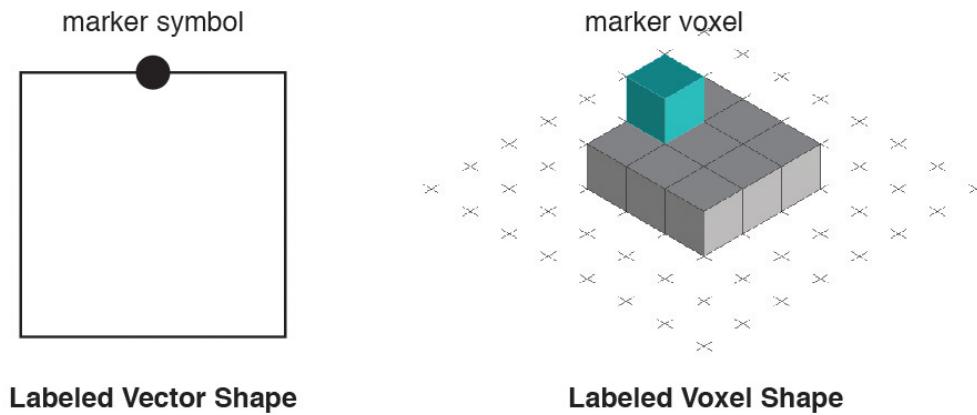


Figure 8.33 – A shape and a voxel shape. A vector shape as defined by Stiny 1980b and a voxel shape as defined in this research. Image credits: the author.

voxel into a cube of eight voxels: $2 \times 2 \times 2$. Rotations would go only in 90-degree increments and so on. A less strict definition, allowing real number scale factors and rotation angles, is also possible. Still, it would need a post-processing voxel detailing routine, such as marching cubes (Lorensen and Cline 1987), and thus introduce unnecessary complexity for the current purposes of the technique (Crumley, Marais and Gain 2012).

The original shape grammars formalism uses vector shapes that are simple representations of architectural elements such as walls. Because of its 3D nature, the proposed voxel-shape grammar could be used to represent both elements (walls, slabs) as well as massing volumes of a building's or city's elements.

An essential aspect of shape grammars is the labeled shape, consisting of a shape and symbols, used as a matching mask for the grammar rules. The marker symbols in vector space shapes are notational. In our voxel shapes, we use voxels with unique attributes for markers (Figure 8.33).

Voxel-shape grammars

A shape grammar is defined by rules and an initial shape and is, in essence, a language of shapes. Shape grammar rules consist of a left-hand shape that needs to be matched for the rule to trigger and a right-hand shape, also called replacing shape (Figure 8.34).

In voxel-shape grammars I call the left-hand shape a *key*, and the right-hand shape an *element* (Figure 8.35). For a voxel shape to serve as a *key*, it must contain at least one marker voxel.

Having vocabulary shapes and replacing rules defined entirely in voxels has a twofold benefit. First, it requires no symbolic representation to specify a grammar which makes it user-friendly. Second, matching shapes to determine which rules are triggered takes place in the discretized voxel place and, as such, has lower computational complexity than a generic, continuous space, vector-based shape grammar.

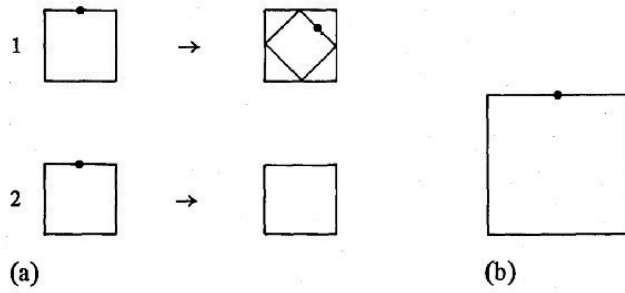


Figure 1. A simple shape grammar that inscribes squares in squares. (a) Shape rules, (b) initial shape.

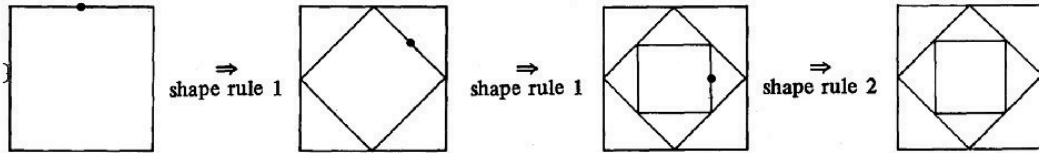


Figure 2. Generation of a shape using the shape grammar of figure 1.

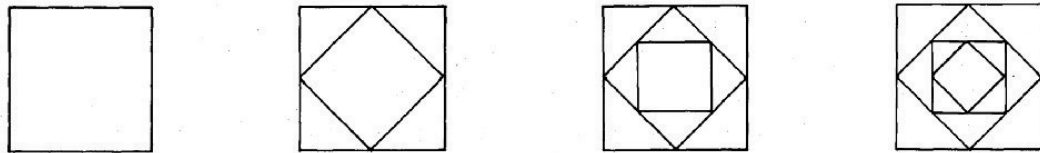


Figure 3. Some shapes in the language defined by the shape grammar of figure 1.

Figure 8.34 – Shape grammar specification. The original graphic specification of shape grammars as it appeared in *Introduction to Shape and Shape Grammars* by Stiny 1980b. Image credits: Stiny 1980b.

Playable voxel-shape grammars

Voxel-shape grammars, as defined so far, would solve the *Implementation* and *Specification* problem yet still have the *Derivation* problem of shape grammars as described in subsection 8.3.1.

I introduce game mechanics as a component of the grammar rules to increase a designer's influence over the grammar derivation process. In effect, this means two things. First, a player can trigger specific rules by modeling a key shape and placing the marker voxels themselves. And second, the position of the player can be used as a marker symbol too, i.e., as part of the left-hand side of a rule Figure 8.36.

Game design helps control player actions and is facilitated by guidance mechanisms. The grammar author encodes their design intentions and expert knowledge by defining the guiding mechanisms. They can incentivize a player to trigger a specific grammar rule as it gives them the ability to place new voxel materials or unlock new rules from the grammar. It can also incentivize a player to reach, i.e., build themselves and access to, a particular position in the voxel world as it rewards them with new abilities as well (Figure 8.37). A well-defined guiding mechanism would be integrated into the game mechanics and would, in essence, provide a desirable advantage to the players for their next moves.

Differing from shape grammars, where the generation is automated, and each iteration step produces the conditions for the following, *playable voxel-shape gram-*

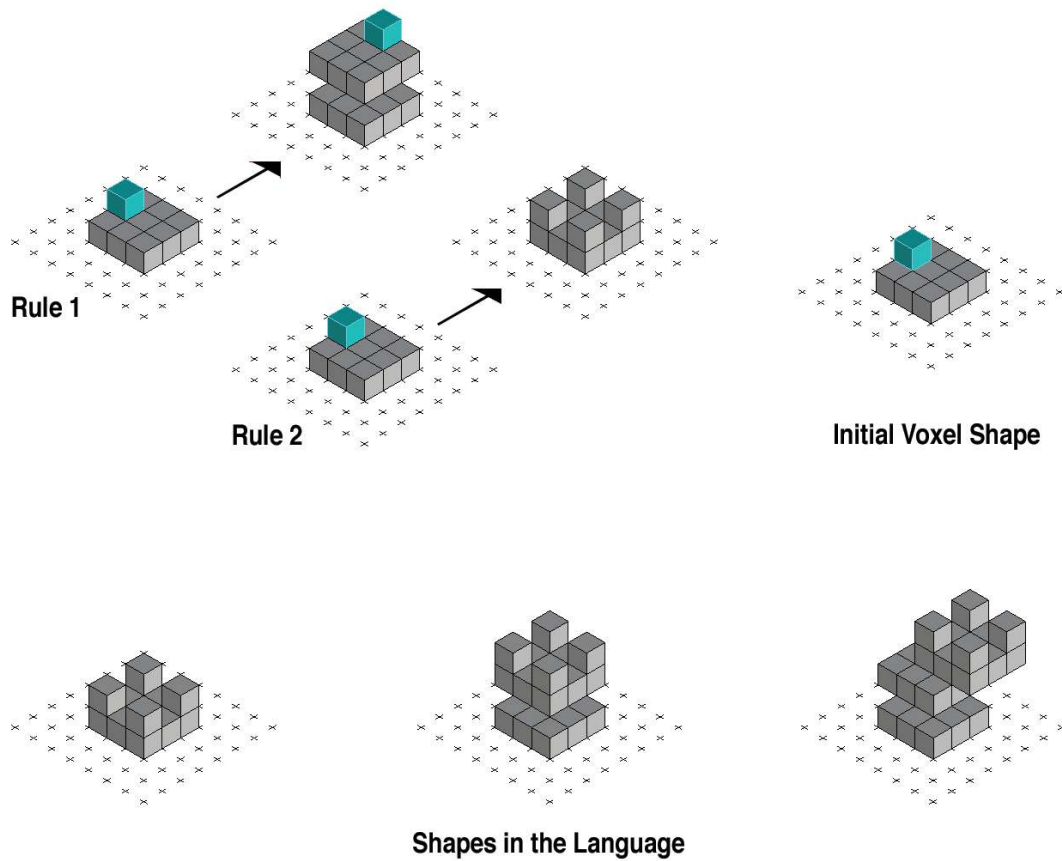


Figure 8.35 – Voxel-shape grammar specification. A voxel-space 3D version of the 2D vector-shape grammar from Figure 8.34. Both vocabulary shapes (elements) and replacing rules are defined entirely in voxels. The bottom row shows some of the shapes in the language defined by Rules 1 and 2. Image credits: the author.

mars require that a player initiate all, or at least some, iterations. This makes a conscious choice, i.e., design decisions, essential for the exploration process.

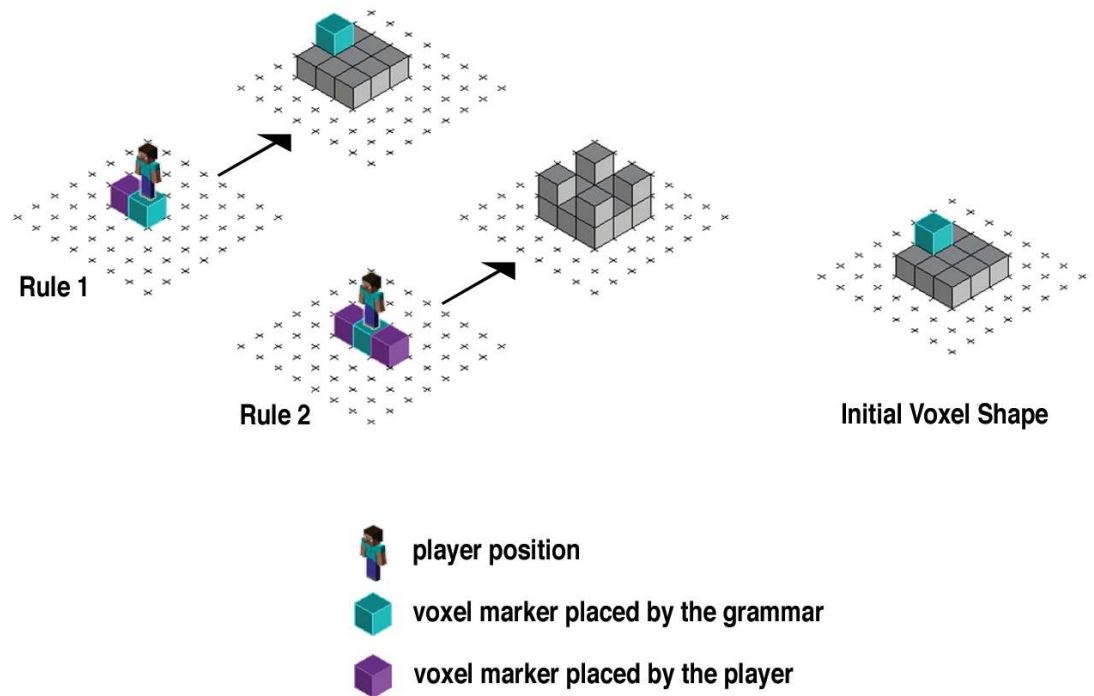


Figure 8.36 – Playable voxel-shape grammar specification. Left — rules, right — initial shape, bottom row — step-by-step derivation of the grammar. Besides the voxels' state, the rules take into account also the player's position. Image credits: the author.

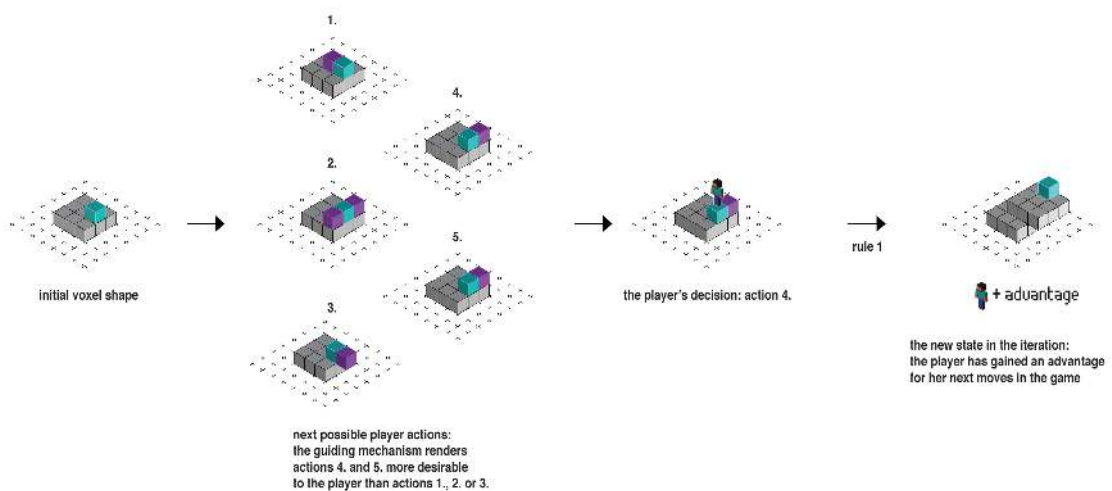


Figure 8.37 – Game mechanics implemented with playable voxel-shape grammars. A guiding mechanism provides a desirable advantage to the players for their next moves and thus could influence their decisions. Image credits: the author.

8.3.3 Features of the rule system

Types of markers

I define two types of marker voxels:

1. the *activator block* — on which a player steps to activate the detection of a masking shape they had built around it (Figure 8.36);
2. the *extension block* — which is a marker that replaces the activator block upon successful detection.

Some of the grammar rules require the extension block in their masking shapes. That means some shapes in the grammar act as seeds, for example Rule 2 on Figure 8.38, and others as extensions, for example rules 3–6 on Figure 8.38. The extension block ensures that structures grow out of themselves coherently instead of populating the design space with too many scattered, independent structures.

Key shapes

Player-placed voxels, i.e., resources, are in essence the material with which the player *draws* or models the key shape to trigger a specific rule. For example, in *IBA_GAME*, I used two resources that encode a set of meta-design features into the voxel shapes:

1. an urban resource — shapes are more solid, with vertically proportioned windows and enclosed;
2. a green resource — shapes are lighter, with horizontally proportioned windows and more open (Figure 8.39).

When using either type of the two resources to build the same key shape, the player would expect the same type of resulting shape, with the same meaning during the derivation process but with different design qualities.

Grid-enforcing rules vs. freely spaced ones

When designing the playable voxel-shape grammar, the expert needs to make sure that all rules are spatially compatible with each other. Incompatibility would mean that the replacing shape of the rule triggered last destroys the integrity of shapes created with previously triggered rules.

One strategy I enforced in the rules is growth in predefined steps. The markers were always placed in a shape according to a grid of 9 x 9 x 8 voxels (Figure 8.40).

Another strategy, which avoids the rigidity of using a grid, is for the shapes to come with the markers in a position that allows for the making of stepped structures and more spatially diverse designs. To avoid too many unwanted overlaps, new structures only grow from existing ones (seeds) to allow us to detect rule adjacency (Figure 8.41).

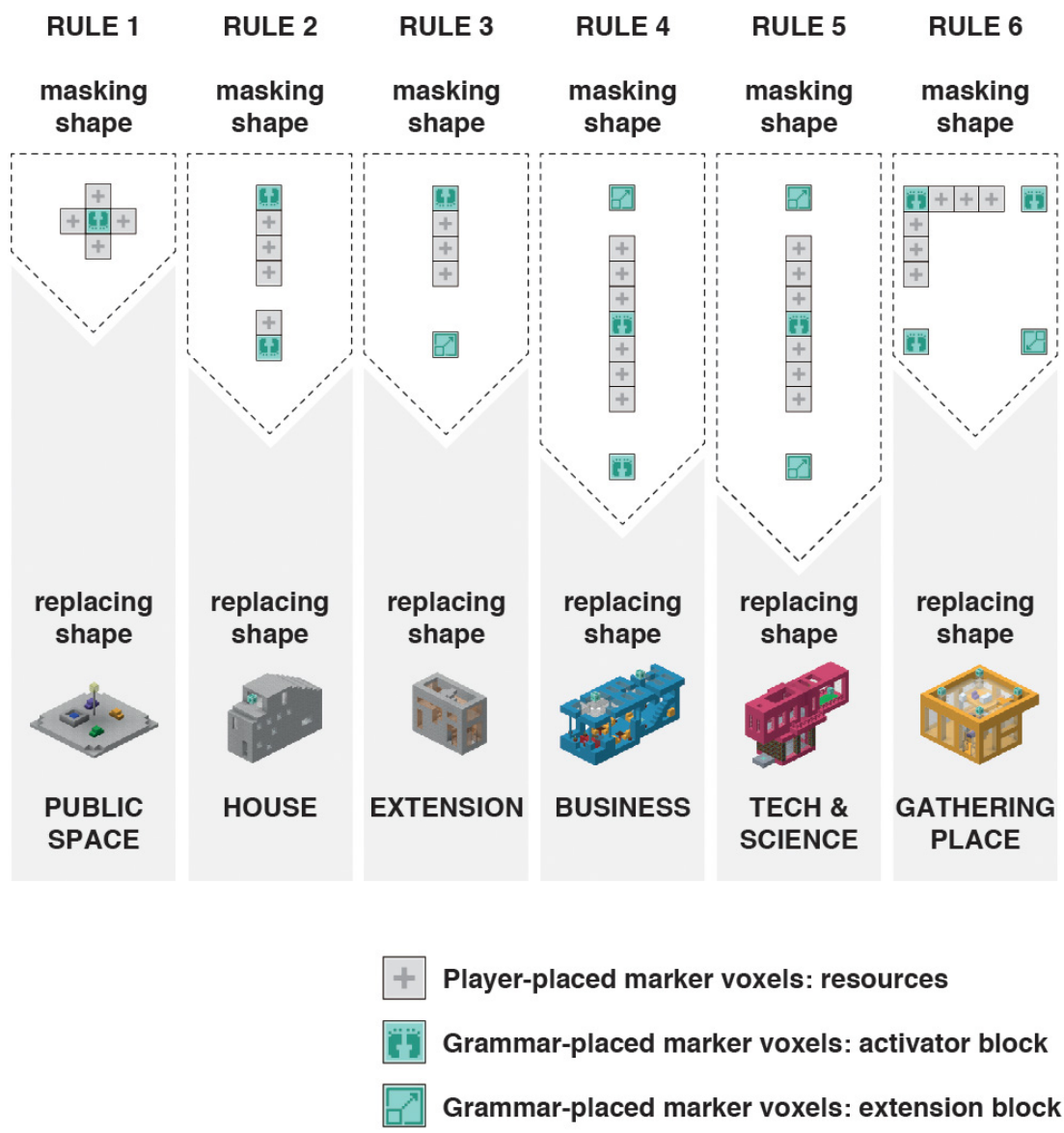


Figure 8.38 – An example of a voxel-shape grammar at the urban scale. It enable players to place pre-designed buildings. Rules 1 and 2 can be triggered anywhere in the voxel grid by modeling there key shapes. Rules 3–6 require the existence of a House produced with Rule 2 as it contains the extension block needed to trigger them. For more see the section on *IBA_GAME*. Image credits: the author.

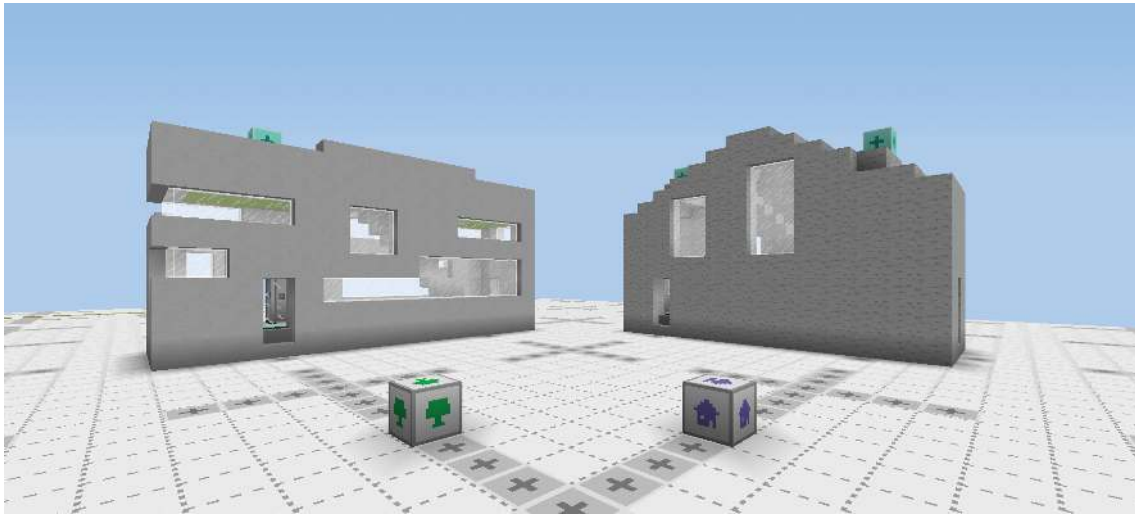


Figure 8.39 – Two resources that describe a set of metadesign features of the voxel-shapes in the grammar. Left: the *green* resource and the house associated with it. Right: an *urban* resource with its house variation. Image credits: the author.

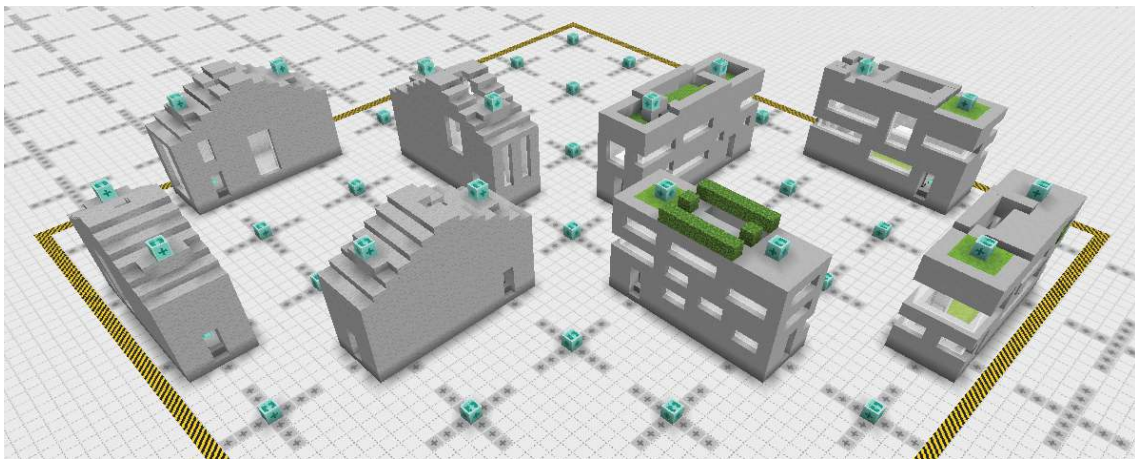


Figure 8.40 – Design variations based on shape orientation. The HOUSE RULE (rule 2 from Figure 8.38) in the urban scale grammar. The underlying grid of 9x9 voxels is visible on the ground. Image credits: the author.

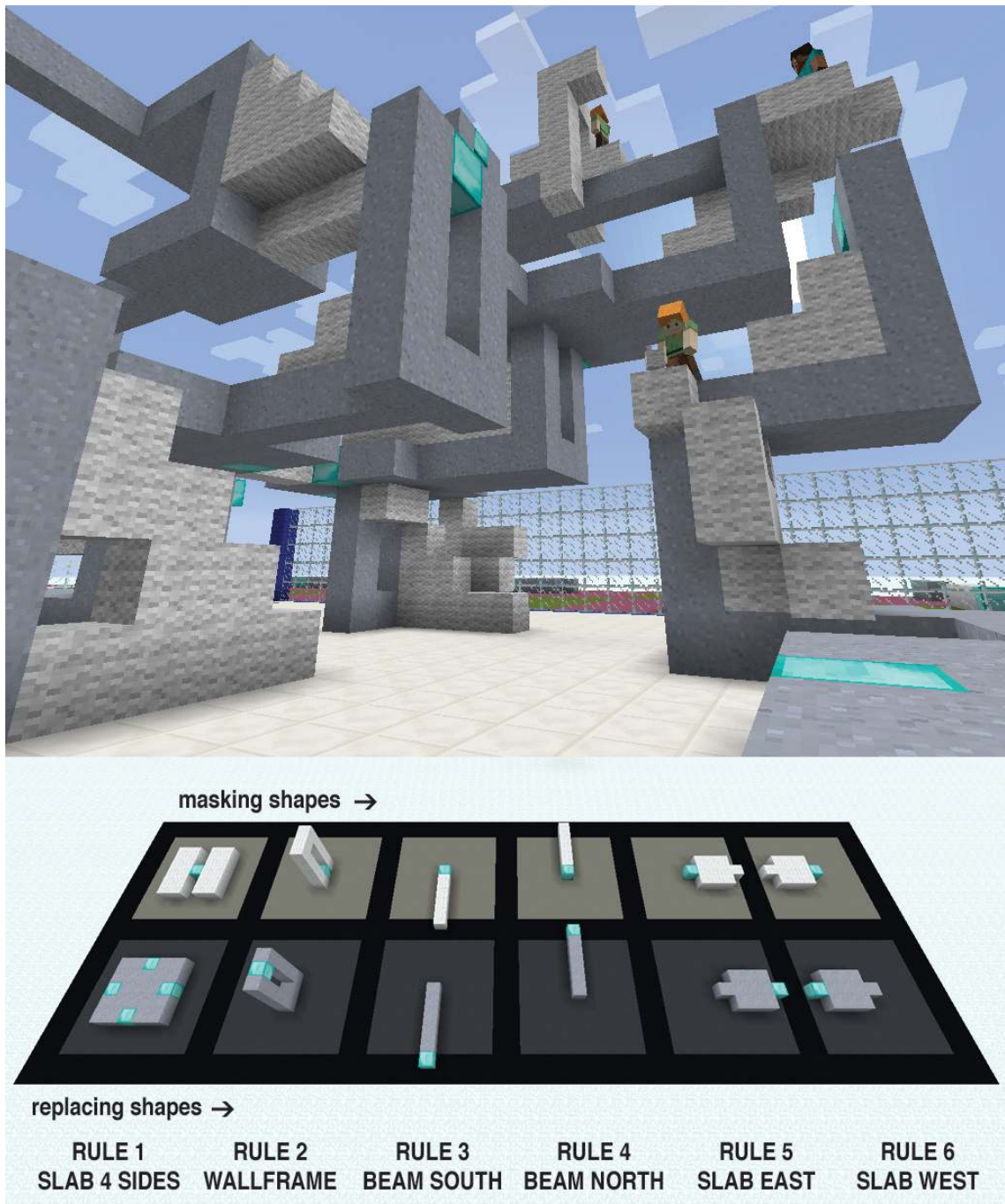


Figure 8.41 – A schematic design for a residential building created with a 6-rule voxel-shape grammar. The blue voxels are the marker voxels needed to trigger a specific rule. Note how the replacing shape in each rule contains new marker voxels. This constrains players to create designs that grow out from a seed shape. Image credits: the author.

8.3.4 Game mechanics: features of the guidance mechanisms

The term *playable* in the name of the *playable, voxel-shape grammars* technique is facilitated in the form of an economy. If a player has access to the resources and the territory needed to create a key shape, they could trigger a grammar rule and gain the rewards given by the rule. Game design can be used as a generative force and at the same time as a control mechanism over the outcome. Control mechanisms are a means to guarantee feasible designs (Ruiz-Montiel et al. 2013).

I developed three types of guidance mechanisms to enable designers to encode design intentions into a *playable voxel-shape grammar*: a quantitative game-goal system; limiting choice with the shape design; and performance-based feedback.

Quantitative guidance mechanism

We used a soft goal system that qualitatively guided the growth as players chose different goals. For example, in *IBA_GAME*, we defined four server-wide goals: collect as many as possible green points, build the most buildings, build the tallest structure, build as many different buildings as possible (Figure 8.86). Each voxel-shape rule in the grammar rewards the player differently on all four metrics.

An example of a goal given to the players is to build up to a certain height, e.g., 30 blocks. At the same time, their initial resources allow them to build only 12 blocks high. Each player can create a shape out of the vocabulary for which they get rewarded with additional 12 blocks of material. However, the shape is designed to consume 10 of their blocks yet is just one block high. That means that the maximum height the player can reach now is 15 blocks. Players can combine the shapes from the vocabulary to form the best strategy to reach the height of 30 blocks and win. They can also interact with the other players by building together or stealing their achievements.

Another type of goal, for example, could be to build a certain number of a given vocabulary shape with the constraint that, again, the material needed for them is initially insufficient.

Limiting choice with the shape's design

If players pursue a soft goal such as height, they will aim to build as high as possible with as little effort and expense as possible. To prevent the stacking of the same shape on top of itself, a shape can be designed so that players cannot model the key shape that triggers the same rule on top of it (Figure 8.42).

Performative guidance mechanism

Additional game mechanics can be created by evaluating the performance of the designs created by players. This does not require extra steps or skills from the players. I developed a pipeline that exports the voxel model to Grasshopper every time a rule is activated. Then it runs an analysis routine on it, and *reports* back the results to the players by color-coding certain blocks of the structure. Figure 8.10 in section 8.2 shows an example based on structural performance.

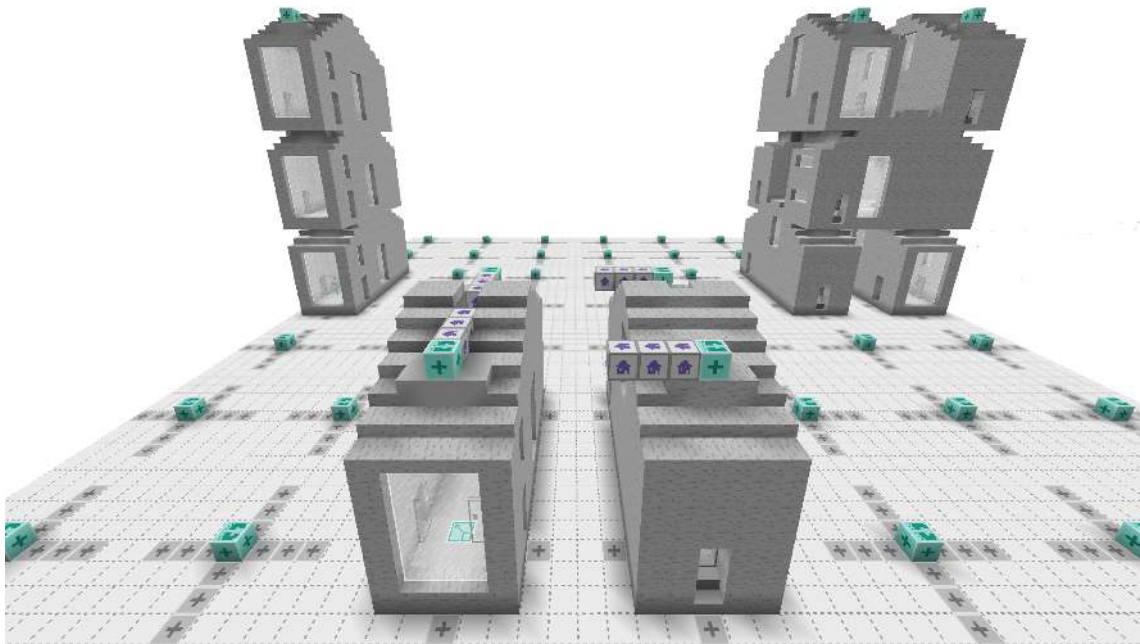


Figure 8.42 – Limiting player choices with design. Left: With a rule that allows users to stack the same house over itself, the most efficient way for a player to increase their height score is to stack them up. Right: If the voxel shape, by design, prevents direct stacking but allows crisscrossing, the design outcomes are qualitatively different. In the foreground are given the respective shape designs. The key shapes for triggering the rule that produced them are the combinations of one blue block and three blocks with purple house symbols. Image credits: the author.

8.3.5 Application scenarios

I developed and used the concept of playable voxel-shape grammars in the project *20.000 BLOCKS*. My implementation consists of a Minecraft server with the ComputerCraft mod and a set of custom scripts that drive rule detection, player/goal tracking, and exporting results. See section 8.2 for further details. I explored the use of playable voxel-shape grammars at two scales: building and urban.

At the building scale, the shapes in the grammar could represent physical elements of a building, such as walls and slabs, the massing of the volume of a room, or a fragment of a building representing its architectural syntax. The rules represent the logic for those elements to aggregate next to each other to form a building. This approach was applied in the following *20.000 BLOCKS* projects: Platform Game, Play-Design-Build, and Combinatorial Design.

At the urban scale, the shapes in the grammar represent pre-designed whole buildings (or mass models of buildings), and the rules represent the possibilities for these buildings to exist next to each other in a city. This approach was applied in the *IBA_GAME*, a game created for IBA Heidelberg using *20.000 BLOCKS*, where players can create small neighborhoods.

8.3.6 Contribution to the field of generative design

With *Playable, Voxel-Shape Grammars* I offer an extension of the shape grammar formalism, most specifically of set grammars. Unlike generic shape grammars spec-

ified with vector shapes, *playable, voxel-shape grammars* are specified entirely in voxel space. This makes them easier to specify and implement in a computational model.

Furthermore, *playable, voxel-shape grammars* open the generative process to human agency. Grammar designers can add direct participation and guiding mechanisms to the otherwise purely algorithmic grammar derivation process by using game mechanics in the grammar rules. The aspect of playability allows the architects, specifying the grammars, to steer the resulting architectural designs while keeping the player focused only on the game mission and not on the artifact they are creating.

Playable, voxel-shape grammars are a form of encoding architectural knowledge that enables twofold crowdsourcing. First, it makes architectural knowledge available to non-experts to create designs. At the same time, it allows an expert, i.e., an architect, to expand the design space by creating a new grammar.

The technique can be used for the generation of schematic architectural designs. I target use scenarios in the early design phase, when all stakeholders explore the relationship between the project elements informally and loosely, often in massing models.

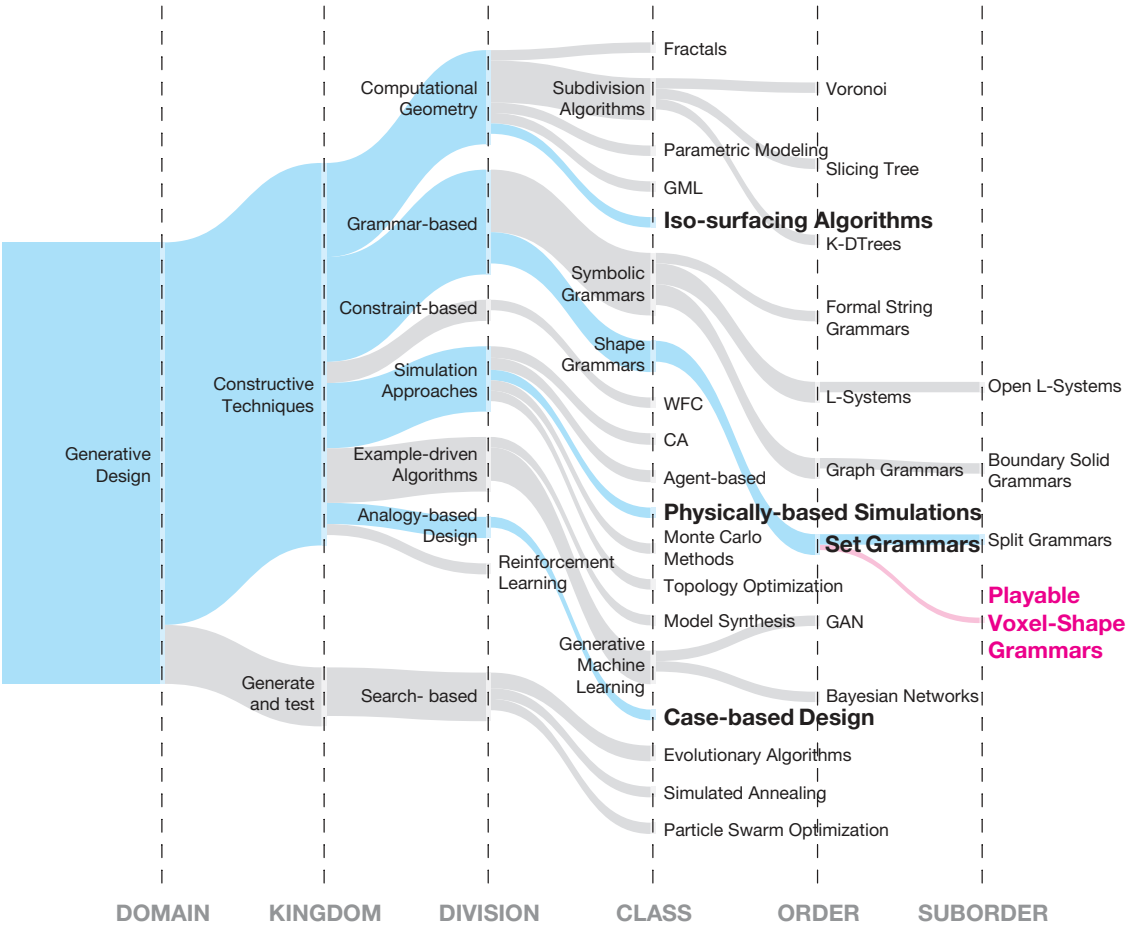


Figure 8.43 – Taxonomy extension. The *Taxonomy of Generative Design in Architecture*, introduced in section 3.2, extended with the *Playable voxel-shape grammars*. Image credits: the author.

As defined in this section, the Playable voxel shape grammars are an extension to

the shape grammars formalism, more specifically the Set Grammars (Figure 8.43). With the *Playable voxel-shape grammars* technique, architects can encode architectural logic and principles in the grammars and easily modify them. The grammars are volumetric and thus open to immediate interpretation and analysis for their architectural potential without further transformation or translation steps.

Furthermore, the method allows for the integration of design and fabrication. An additional benefit of using voxel-shape grammars instead of vector-shape grammars is the ability of embedding material specification rules in the grammar by varying voxel material (Michalatos 2016; Rossi and Tessmann 2017c; Stiny and Gips 1971).

8.4 Human Agency

The *20.000 BLOCKS* case study engaged people in three of the four roles: experts, crowd, and tool-makers. The role of the non-expert stakeholder was not addressed.

The first role is that of the *Tool-makers* or *Facilitators*. As researchers, this is the role that a few collaborators and I took. The role is to develop the technology that others can use to create games that generate designs.

The second role is the *Experts*, whose knowledge is encoded a specific *20.000 BLOCKS* that they create. These are the people setting and modifying the vocabulary and defining the goal. The game industry analogy would be of a Level designer who is in charge of creating the challenges within a game. Experts can edit the catalog to express the design ideas they would like to explore. Furthermore, they can change the game design by defining player achievements, goals and tuning the economy of resources distributed to players. Experts can also provide verification routines in Grasshopper and link them to the Minecraft world. They post-process the player-created designs to detail them further and digitally fabricate them. Lastly, experts have access to search tools to browse through all player-created designs and observe patterns or use them to autocomplete their sketched-out designs.

Players are the third role. The experience of the players is tightly controlled. The Organizers decide which materials they can place and break and where they can walk. The players are given a goal but insufficient resources to achieve it. To progress, players build shapes out of Minecraft blocks, choosing from an architectural vocabulary defined by the Experts. Players are rewarded with resources for creating one of the shapes. While players compete to reach the goal, a building emerges out of the shapes that they have built.

The description of each project that follows presents how the architect's role defined its architectural and game design features and the modes of engagement for the players.

Figure 8.44 shows the hierarchical structure of the case study *20.000 BLOCKS*. The technological framework enables the specification of various projects that are mini-games with their game pieces and economies. Each project, or *20.000 BLOCKS* game, produces an architectural design when played. Each gameplay delivers a different design alternative recorded for later post-processing and analysis.

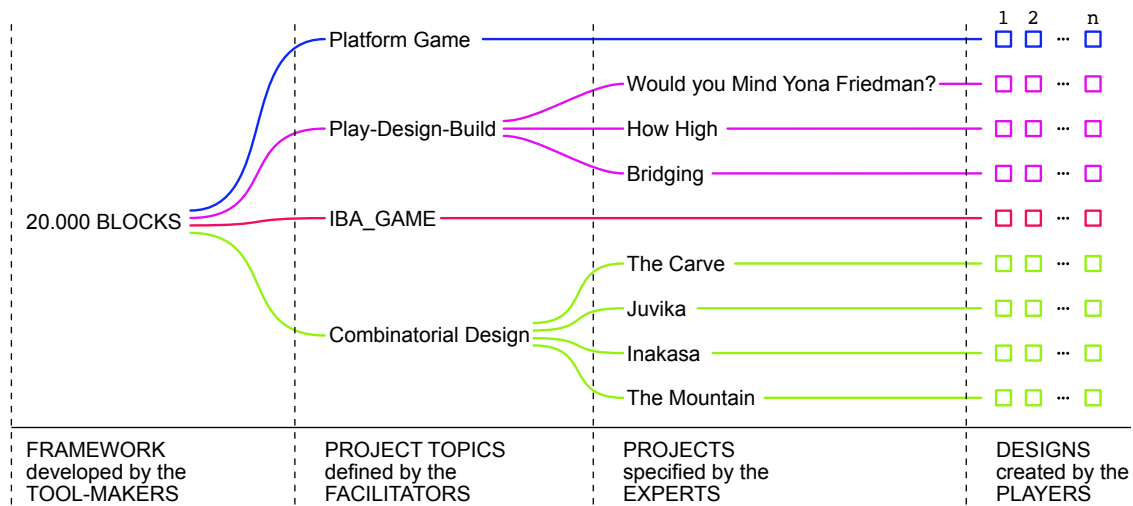


Figure 8.44 – Structure of the case study 20.000 BLOCKS. The case study is organized into projects created with the framework. Some projects are created within the same overarching brief or topic. Each project is a game defined by the experts. When played, these games produce designs created by the players. Image credits: the author.

8.4.1 Platform Game

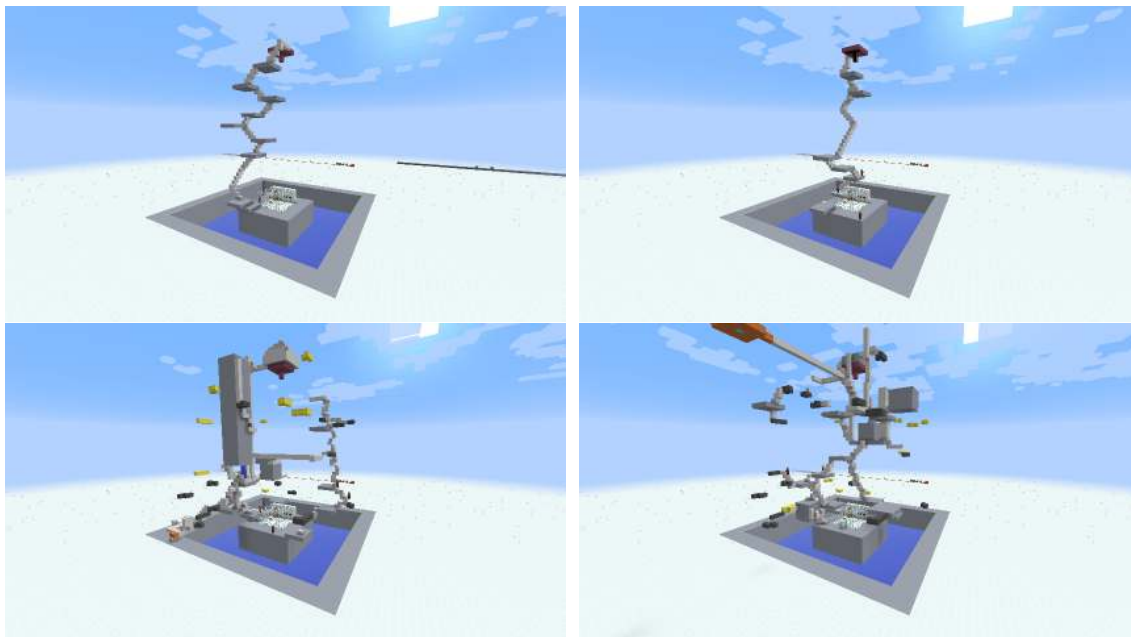


Figure 8.45 – Results from Platform game. Top: Two game results from a game with hard vocabulary and one main goal. Bottom : Two screenshots from gameplay with hard vocabulary and secondary goals, marked in gold. Image credits: the author.

The *Platform game* is the first project realized with the framework *20.000 BLOCKS* and, in essence, served as prototyping ground for many of the framework features used in the later projects. The game designer Ben Buckton and I developed the project intuitively and iteratively. We would implement a feature, playtest it with a group of players in informal play sessions and either keep it, modify it or drop it. The project followed a technological evolution as well, since the game logic was first implemented using command blocks (Figure 8.47) in vanilla (unmodded)



Figure 8.46 – Platform game screenshots. Left: The Spawn zone of the Platform game with players waiting for the next game to start. The spawn zone is the area where players end up upon joining the *20.000 BLOCKS* Minecraft server and wait to be teleported to the building area. Right: a game has just started and players are building the first platforms. Image credits: the author.

Minecraft and then reimplemented in Lua scripts using the ComputerCraft mod, which allowed more developing freedom (Figure 8.7).

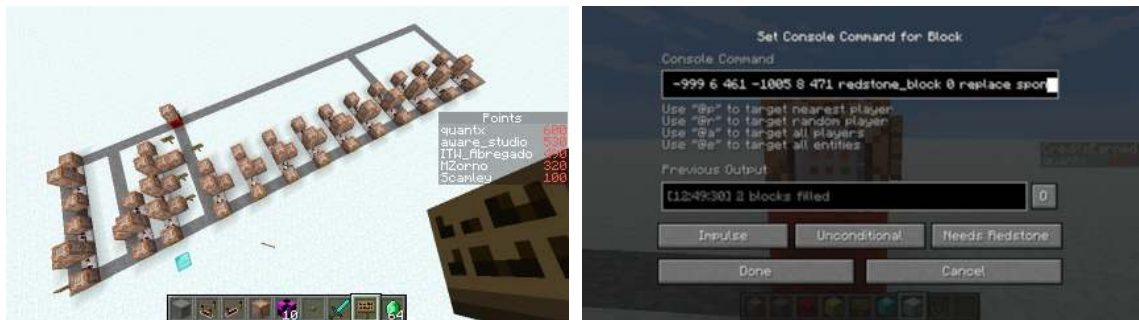


Figure 8.47 – The game logic in an early-version 20.000 BLOCKS world. The logic was implement using command blocks (left) which can execute a single instruction (right). By placing command blocks in sequence one can create loops, if conditions and more complex routines. Image credits: the author.

The goal of the *Platform game* is to be the first player to reach the platform flying at the height of 50 meters above the starting island (Figure 8.48). The island with dimensions 20x20 blocks floats in the middle of an abyss with the size of 40x40 blocks. Players started with limited materials and had to build elements out of a catalog of architectural shapes to get in-game currency. They can exchange the currency (emeralds) for building materials. If a player did not use the materials, they had to build a recognizable shape from the catalog they ran out of resources and could not progress in the game.

The aim is for a game to last approximately 20 minutes, after which the result is saved, and the building area is reset. The results had less architectural potential if the game design was not balanced well. For example, if the players received too many resources at the start of the game, they would build one straight column all the way up to the goal and finish the game in seconds (Figure 8.51 right).

Soft vocabulary

We conducted the first tests with a visual catalog of architectural elements, i.e., soft vocabulary. The players were shown the catalog with elements such as rooms, terraces, etc. (Figure 8.49) and given the task to replicate them in the Minecraft

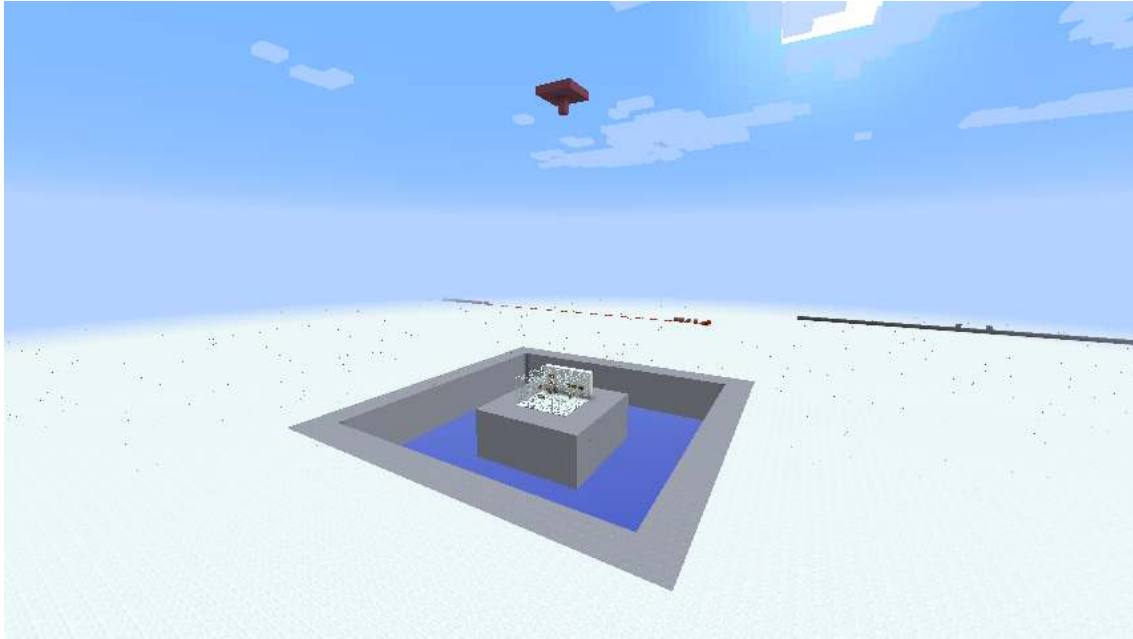


Figure 8.48 – Initial state of the Platform game. The players begin on the island and have to reach the platform in red flying above it. Image credits: the author.

world. Players were given limited materials at the start and promised a resupply if they successfully built an element of the catalog. A player could build anywhere within the confined building site in Minecraft. They would then come to the Organizers, who would visually verify if the created element matches the catalog and award the player the corresponding points and resources. This happened verbally in the game chat. The goal was also soft — get the highest number of points. Each element in the catalog brought different points to the player who built them, based on the element's complexity.

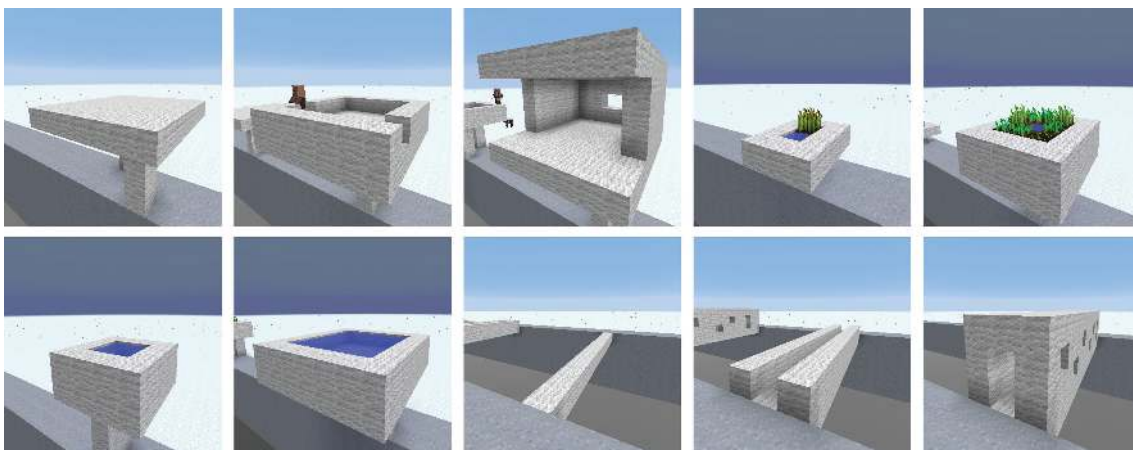


Figure 8.49 – The soft vocabulary. Initially we used a visual catalog, including three variations of platforms, three bridges, two gardens and two pools. Image credits: the author.

Figure 8.50 shows a typical result from a game with the soft vocabulary. The outcomes were rather fuzzy designs as all the aspects of the system — the guiding rules, the verification process, and the feedback — were negotiable between the player and the organizers. Another hindrance, we noticed, was that the elements in the vocabulary were too numerous and too complex to be remembered by players

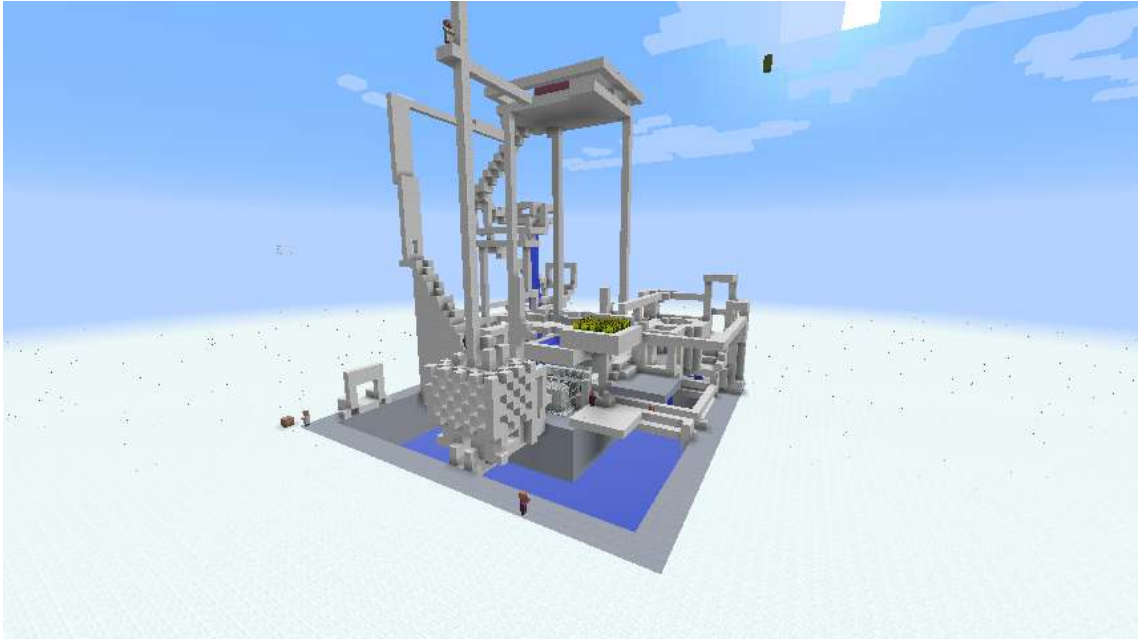


Figure 8.50 – A typical result from a game session using the soft vocabulary. It is difficult to isolate the individual vocabulary shapes that the player had replicated as they modified them while building and organically connected them together into one fuzzy structure. Image credits: the author.

easily. This required the players to refer to the vocabulary too often, breaking their flow of play. Furthermore, it was also slow — a game was marked as completed when the confined building site was filled up, which took around two hours. Therefore, we looked for ways to make the rules stricter by automating them.

Hard vocabulary

The guiding rules of the hard vocabulary method we tested were based on three automated routines:

1. Detection routine: The elements that players build to gain rewards are recognized automatically by the game logic. We described one element — a platform of 5x5 blocks — in code using Minecraft’s programming language and command blocks (??). The player places an activator block (blue diamond) and positions their character on top of it to activate the detection routine (??). Only exact copies of the catalog structures will be recognized and rewarded, thus reducing the fuzziness of the design solutions.
2. Trading: The resources for building need to be purchased from Non-Player Characters (NPCs) — a Minecraft villager — in exchange for emeralds, gained when successfully building an element from the vocabulary. Villagers can be summoned only at the ground level. This ensures that the structure being created can be walked up and down (Figure 8.8).
3. Goal detection: We defined the main goal, achievable within 15–20 minutes that gives a clear end to each session (Figure 8.8). This limits the time for creating a design and delivers multiple player-created designs under the same conditions. Reaching the goal automatically triggers a save of the built structure, resetting the game. When we noticed that game plays resulted in self-

similar linear structures (Figure 8.45 top row) we introduced a set of secondary goals (Figure 8.51 left) as incentives to break the linearity of the gameplay and create spatially richer designs (Figure 8.45 bottom row).

The game loops of the *Platform game* with and without secondary goals are shown on Figure 8.52.



Figure 8.51 – Secondary goals. Left: the secondary goals are shown in yellow (gold) material. Each gold block rewards the player who first steps on it with points after which it turns black and is not active. Right: When the players have too many resources at start that are not incentivized to build any of the shapes in the catalog since they can finish the game right away by jump-building a column underneath themselves. Image credits: the author.

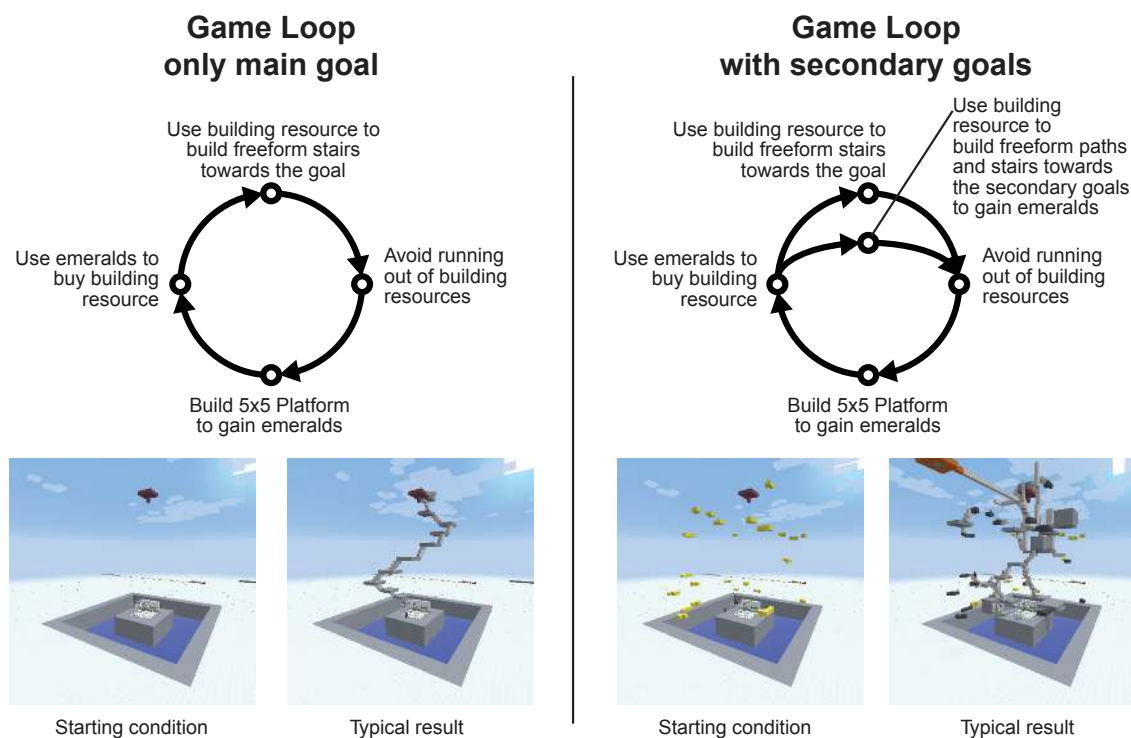


Figure 8.52 – Game loops. With and without intermediate goals. Image credits: the author.

At this stage of the project, many structures built by the players had cantilevers that would not sustain the force of gravity. To unlock the true architectural potential of mass participation, we need a set of evaluations that relate, and possibly rate, a shape on its performance as an architectural structure in the real world. Most of the game outcomes were verified postfactum using Grasshopper/Millipede. The

results prompted us to change the elements in the vocabulary until the play results had less problematic overhangs (Figure 8.53).



Figure 8.53 – Structural simulation of game results. A player-created structure (left) and its simulated deformation after analysis in Grasshopper with Millipede (right). Image credits: the author.

The feedback from the game actions was printed automatically to the game chat (Figure 8.11) and kept the players informed of what to do in the game, who scored a new point by building a 5x5 platform and when the game was over, being saved and reset. This proved very useful and successful in keeping the players aware.

Modifiable vocabulary

The latest iteration of the guiding gameplay mechanics implements a vocabulary of predefined design elements that follows combinatorial rules (Figure 8.54). This uses the concept of playable voxel shape grammar.

It has all the three main principles of the *Hard Vocabulary* variation used in the Platform Game, with the difference being that detectable structures are not described in code but are built on dedicated slots next to the building platform as a visual catalog. This allows us to modify and prototype the vocabulary much faster. It furthermore allowed us to separate the roles of Organizer and Expert without needing to teach Minecraft scripting.

The fact that the vocabulary consisted of more than one element softened the play outcomes. Therefore, we tried a system where the trigger blocks were placed automatically with the vocabulary element and not by the players. We called this new notion *Grammar* because it meant the *Experts* could define which of the elements could be built upon each other, thus opening or limiting choices for the players.

We didn't use the game chat as extensively as in the *hard-vocabulary* approach and relied on players orienting themselves in the game world. This proved confusing for most people.

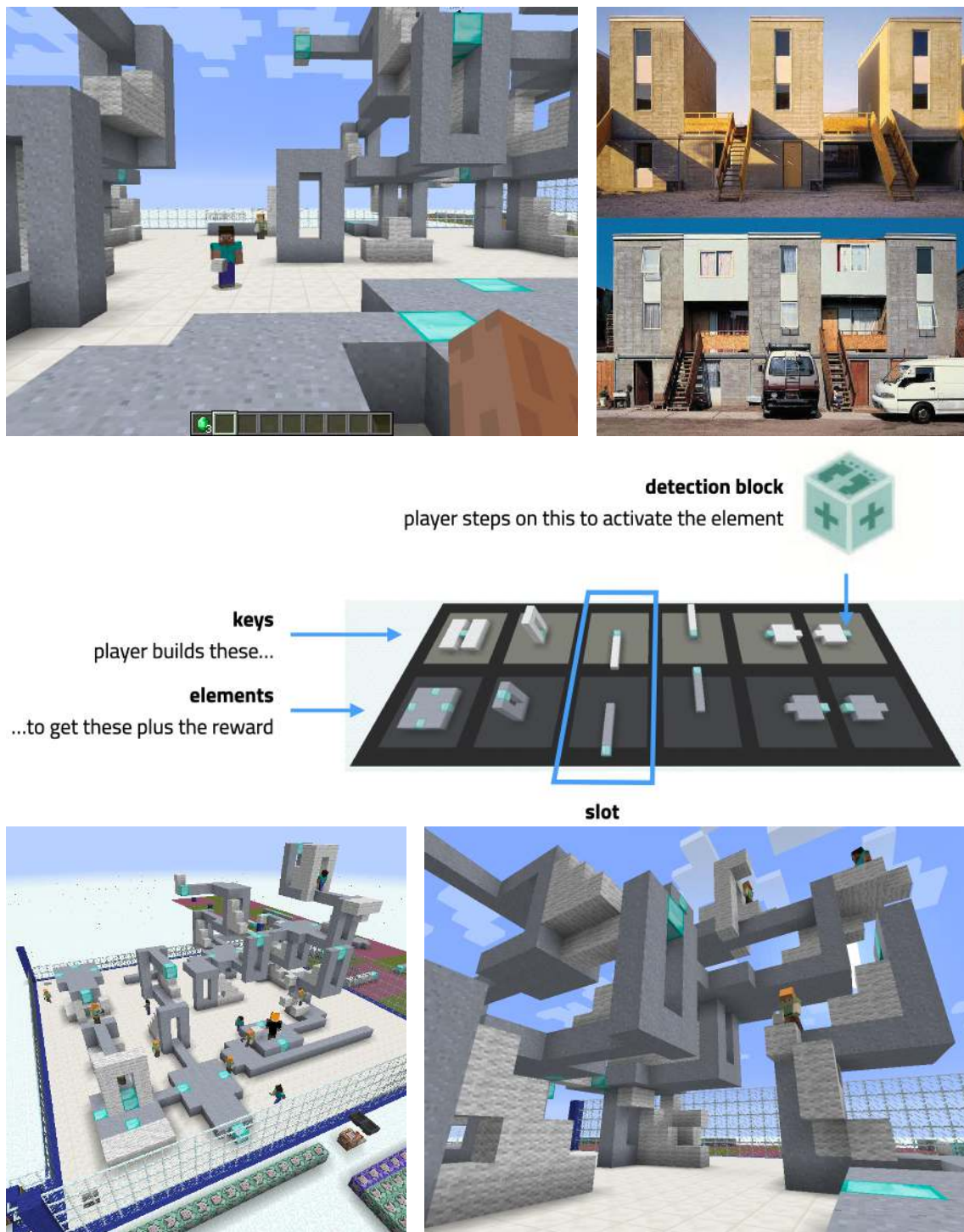


Figure 8.54 – The grammar from the intro game in Play-Design-Build. Left: A schematic design for a residential building created using a voxel-shape grammar with 6 rules in *20.000 BLOCKS*. The concept of “free build” allows for players to define structure using the grammar (in grey blocks) and also fill the gaps with stairs, enclosing structures etc (seen here in white blocks). The project Quinta Monroy by Elemental and architect Alejandro Aravena (right) is a suitable reference as it also combines a basic structure predefined by the architect and freely build infills by the inhabitants. Middle row: the shape grammar. Bottom row: two more in game screenshots from games with the same grammar. Image credits: the author.

8.4.2 Play-Design-Build



Figure 8.55 – The 20.000 BLOCKS team and workshop participants at SmartGeometry 2016. Image credits: the author.

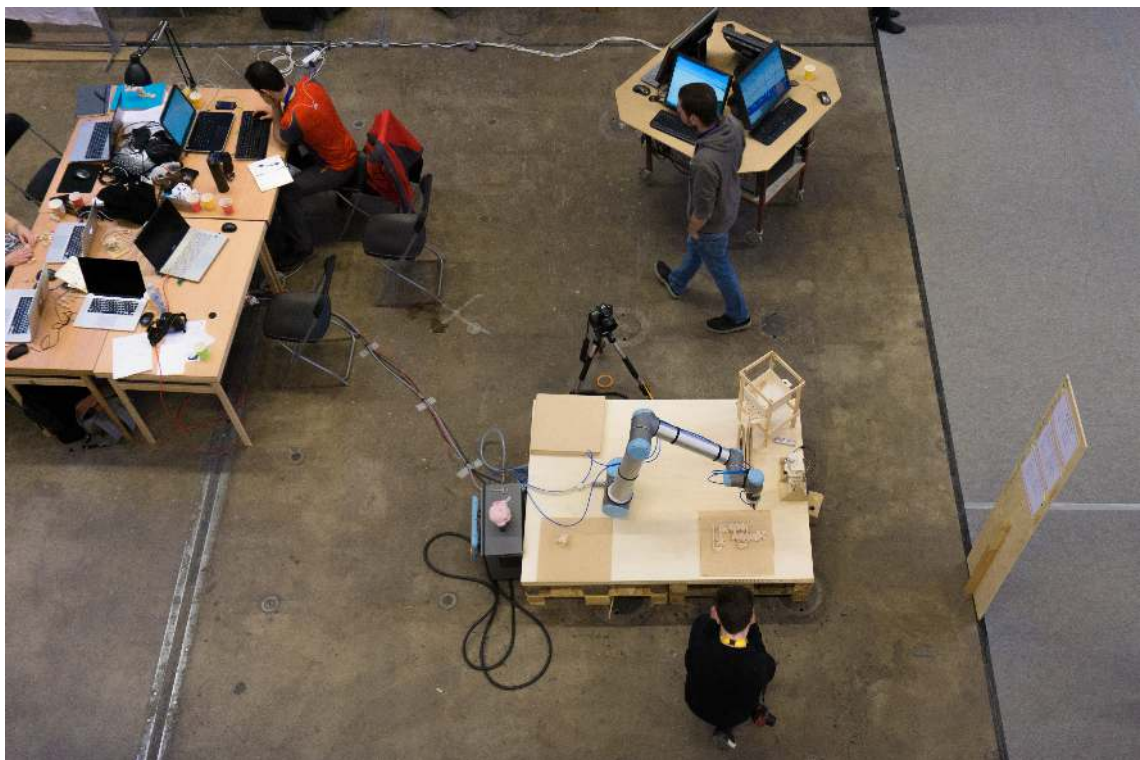


Figure 8.56 – The 20.000 BLOCKS setup at SmartGeometry 2016. With a four-way game terminal and a robot arm. Image credits: the author.

20.000 BLOCKS: Play-Design-Build is the title of a cluster(workshop) at Smart-Geometry 2016 in Gothenburg that I and the game designer Ben Buckton lead together (Figure 8.55 and Figure 8.56). The cluster explored the question: Can gameplay guide non-experts through the creation of collaborative architecture designs?

The cluster introduced participants to the complete cycle of the *20.000 BLOCKS* framework (Figure 8.57). We moved from architectural vocabulary and game mechanics defined in Minecraft, through geometry being imported and post-processed in Rhino/Grasshopper (Figure 8.58), to a digitally fabricated scale model created with a robot arm.

Eight architects and engineers, divided into three teams, used our implementation of playable voxel-shape grammars and defined components of the buildings within the grammar (Figure 8.41). The participants were provided with an example game map, with a grammar consisting of 6 rules shown on Figure 8.54. The three projects are titled: *Would you mind Yona Friedman?*, *How High* and *Bridging*. Each project had up to 16 shape rules and was played by attendees of the conference. The three projects yielded around 50 player-created designs.

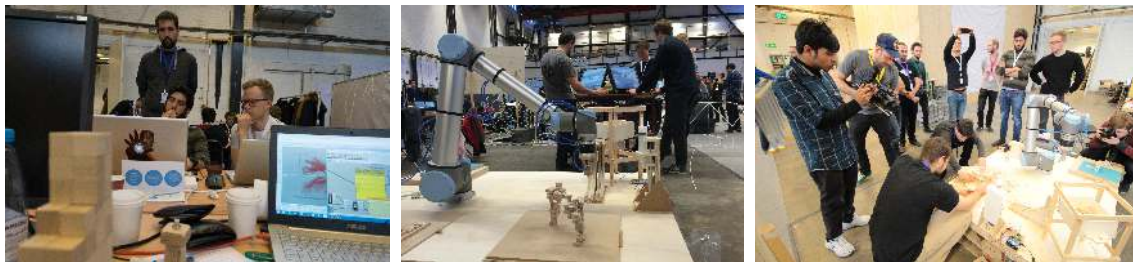


Figure 8.57 – Photos from the 20.000 BLOCKS cluster at SmartGeometry 2016. Left: project discussion between the participants. Middle: the robot setup with the four-way game terminal in the background. Right: A model built by the robot is being cleaned up from the support blocks. Image credits: Jörg Hartmann, the author, Andrea Quartara.

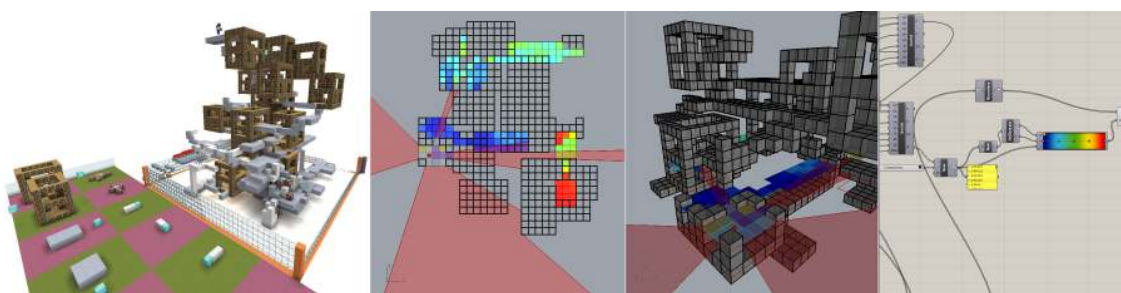


Figure 8.58 – Isovist analysis. A model created by players in Minecraft in the project *How High* is imported and analyzed in Grasshopper. Image credits: Quartara, Emmanuelli, Montnemery, the author.

Would You Mind Yona Friedman?

The project *Would You Mind Yona Friedman* was created by Marios Messios and Max Rudolph (Figure 8.59). The interesting exploration in this project is the change of scale. The authors chose to map the size of one Minecraft clock to 10 meters instead of the standard 1 meter. The topic of the project is a megastructure over the

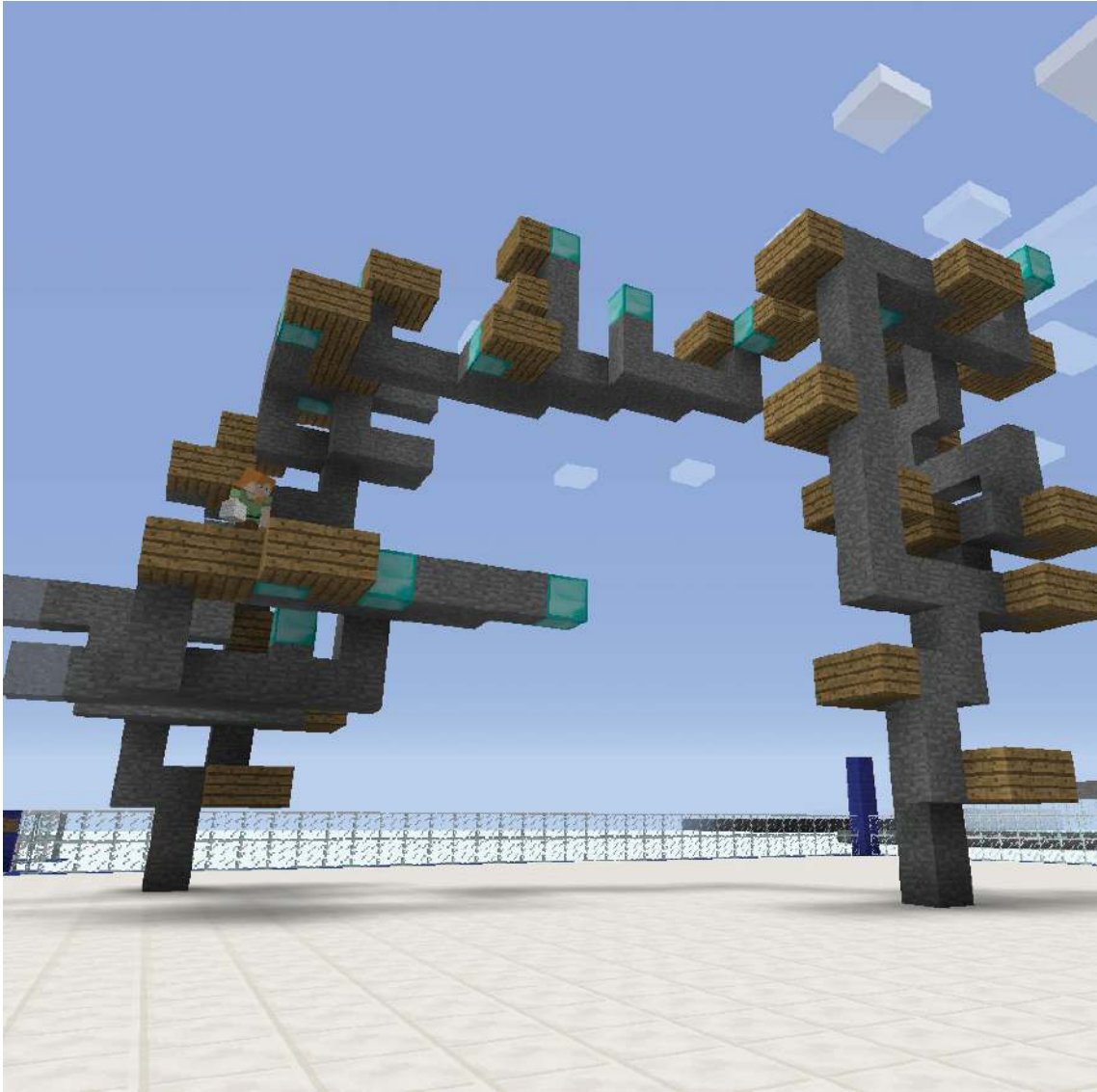


Figure 8.59 – A 20.000 BLOCKS game map titled WOULD YOU MIND YONA FRIEDMAN?. Created by Marios Messios and Max Rudolph during our cluster at SmartGeometry 2016. Image credits: Messios, Rudolph.

city of Paris, inspired by the work of Yona Friedman (??). The catalog of vocabulary and grammar rules reflected this change of scale as well (Figure 8.61). Using different materials, the authors represented different residential unit typologies or units of the building's infrastructure. This allowed for rather small elements in the vocabulary and rules that created branching structures. The game design created an open-ended challenge. Players needed to build elements from the vocabulary catalog to collect game currency (emeralds) to buy additional building materials (Figure 8.62). There was no overarching game goal, and players could build in vertical and horizontal directions and stop when they felt their structure was finished. The project authors post-processed two of the game results (Figure 8.63) into schematic designs (Figure 8.64) on which they carried out a structural analysis using the live-link to Grasshopper (Figure 8.65). The design created by two players was sent to the robot for assembly (Figure 8.66). One of the design alternatives was rendered as an architectural speculation (Figure 8.60 and Figure 8.67).

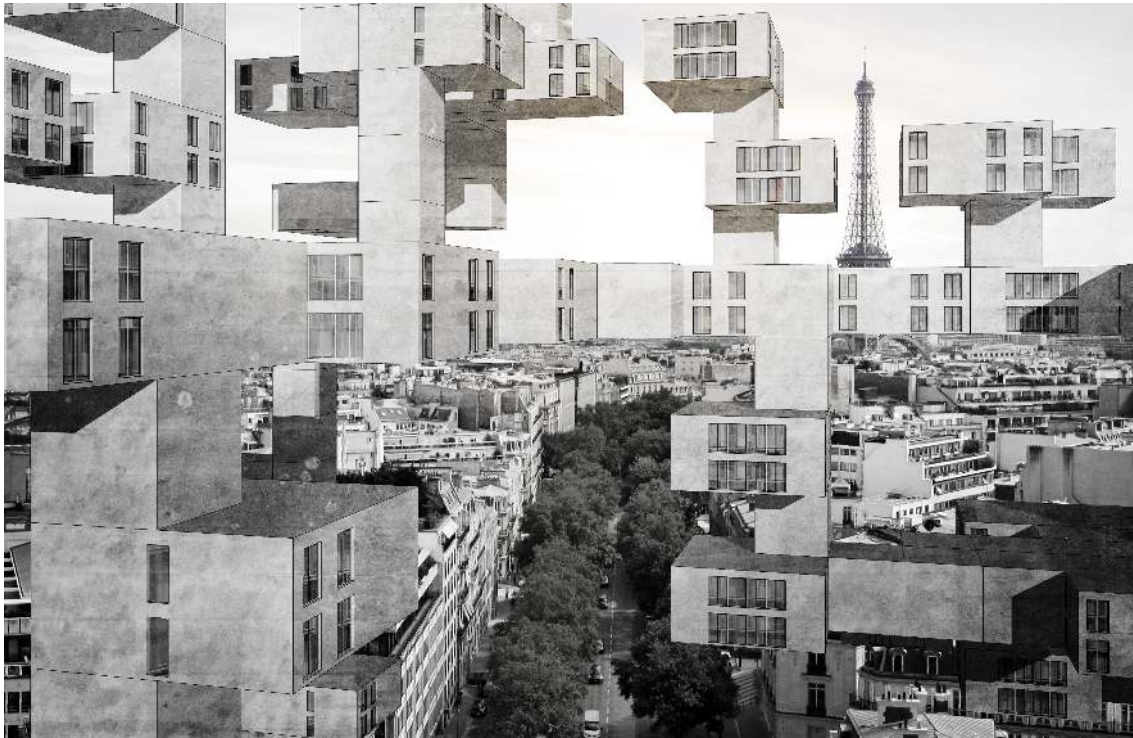


Figure 8.60 – Architectural rendering. An architect's rendition of a player-created schematic design in the *20,000 BLOCKS* project *WOULD YOU MIND YONA FRIEDMAN?*. Image credits: Messios, Rudolph.

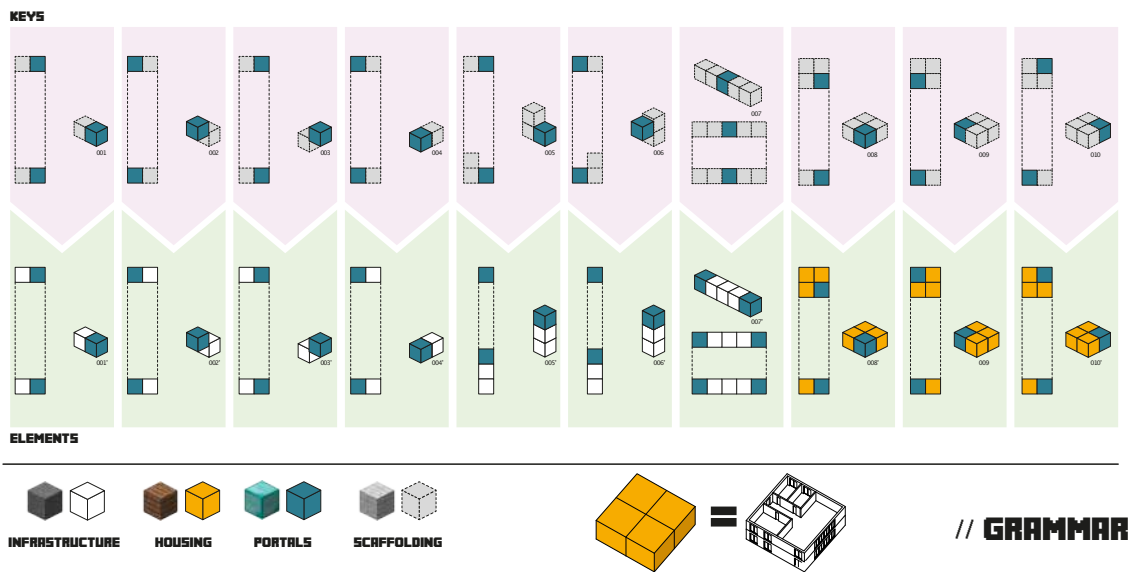


Figure 8.61 – The vocabulary and replacing rules in *Would you mind Yona Friedman?*. The project used a scale of 1 block = 1 meters instead of 1 meter. Image credits: Messios, Rudolph.

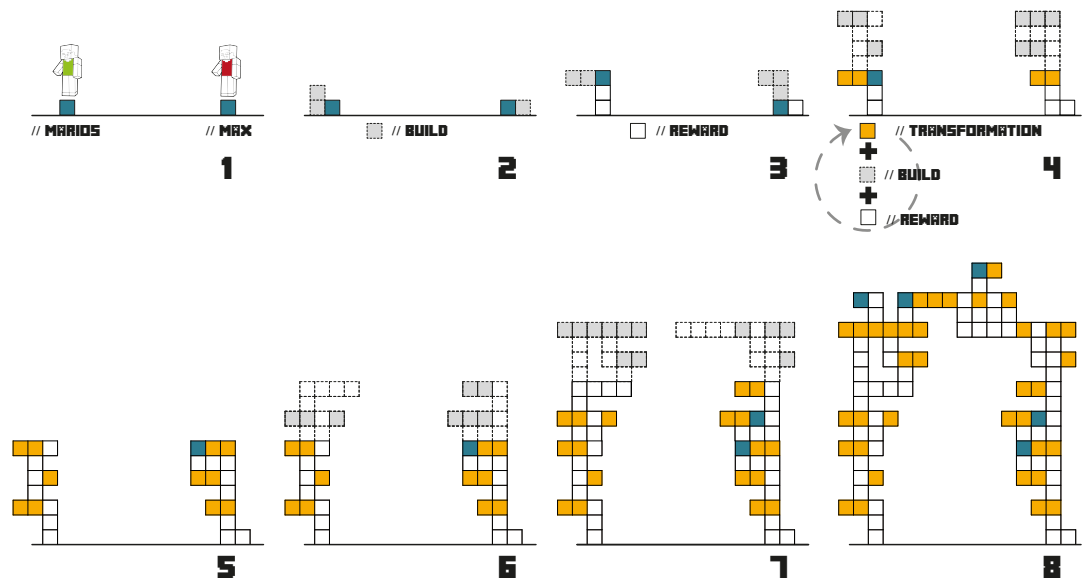


Figure 8.62 – A typical game progression in the project *Would you Mind Yona Friedman?*. Image credits: Messios, Rudolph.



Figure 8.63 – In-game screenshot from the two selected game results. The left one is created by two players, the right one by six. The grammar rules produced structures that appear to have grown out from predefined points in the city. Image credits: Messios, Rudolph.

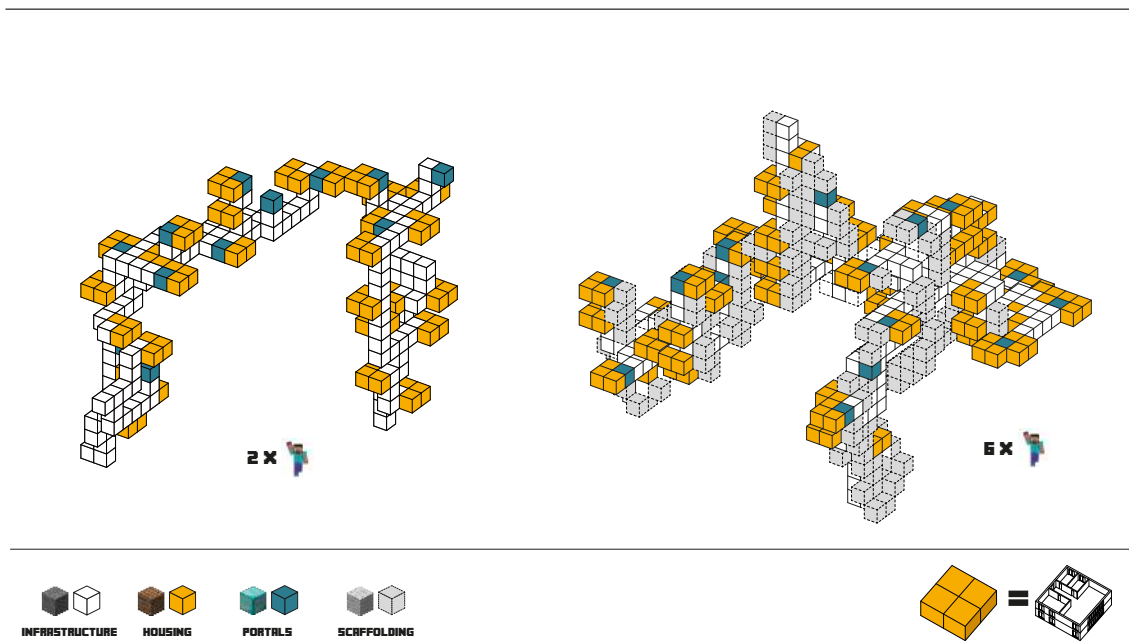


Figure 8.64 – Programmatic composition of the two game results. Image credits: Messios, Rudolph.

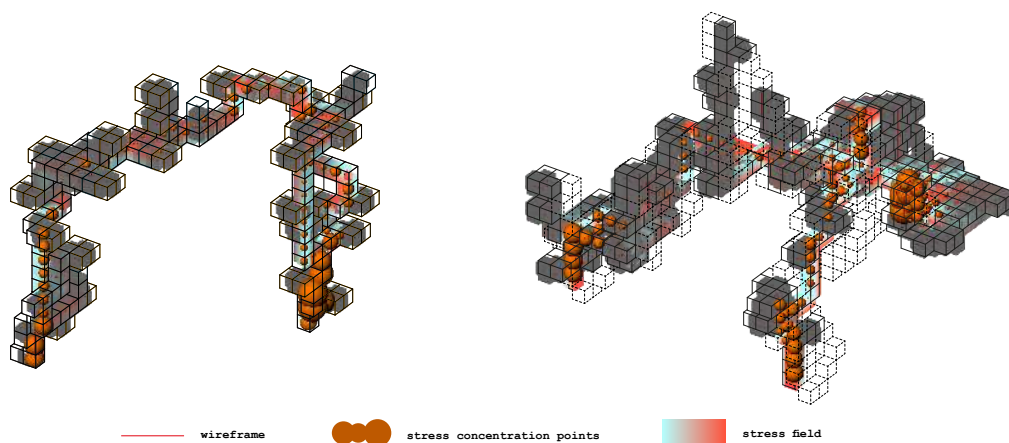


Figure 8.65 – Structural analysis of the two game results. Image credits: Messios, Rudolph.

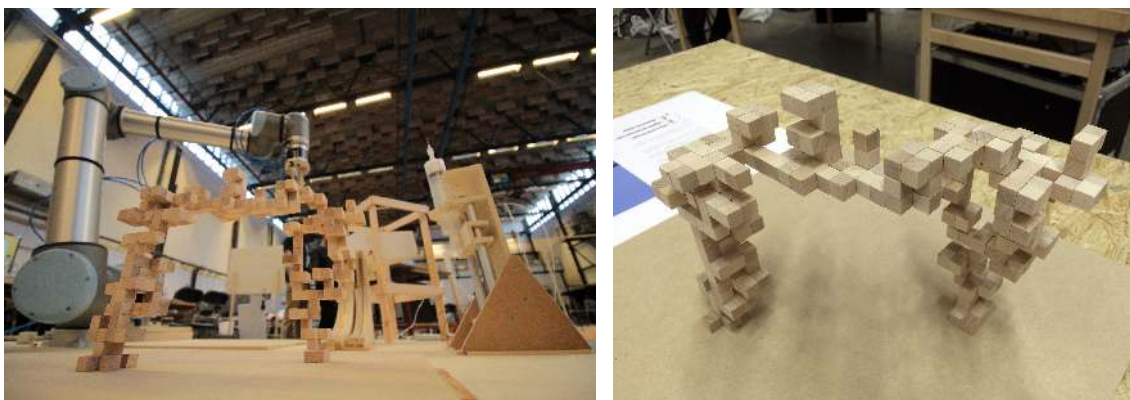


Figure 8.66 – Model photo. A model built by the robot of the design variant from *Would you mind Yona Friedman?* created by two players being assembled by the robot arm (left) and the final model (right). Image credits: the author.

Andrea Quartara



Figure 8.67 – Architectural rendering. A player-created design of a megastructure hanging over Paris in *Would you mind Yona Friedman?*. Image credits: Messios, Rudolph.

How High

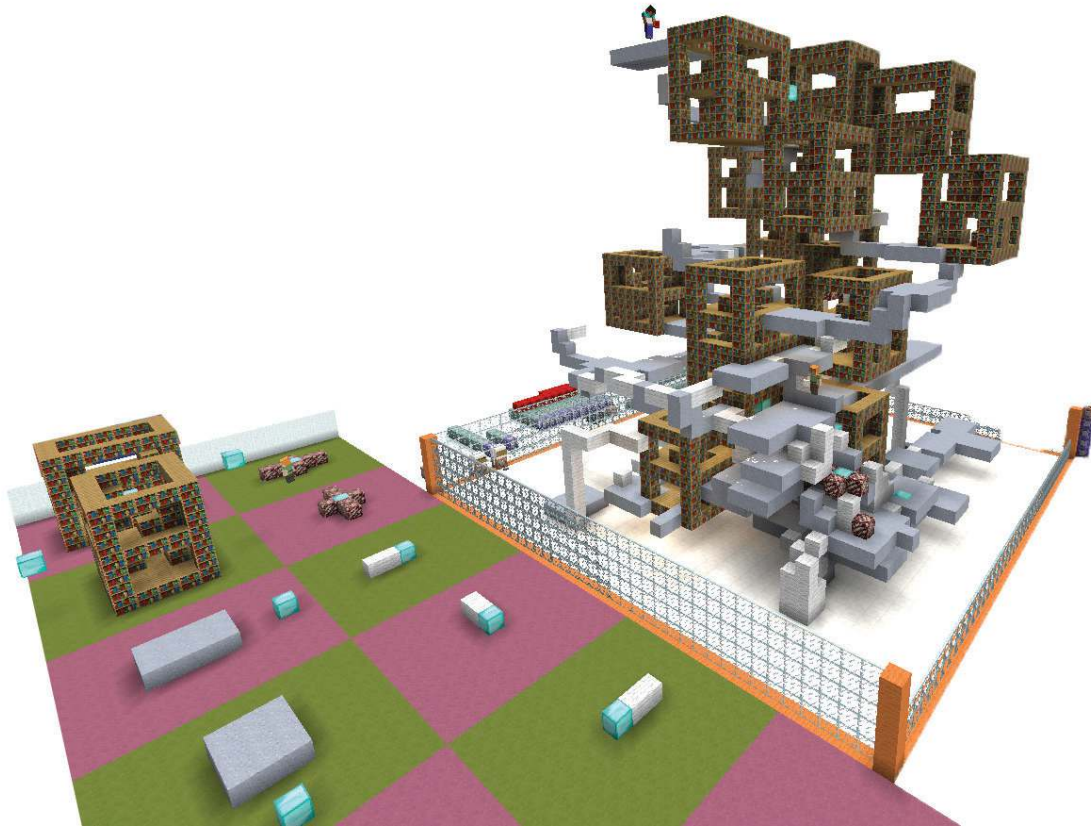


Figure 8.68 – A 20.000 BLOCKS game map titled HOW HIGH. Created by Andrea Quartara, Marc Emmanuelli and Ulrik Montnemery during the cluster at SmartGeometry 2016. The area where the vocabulary is defined is shown in front. The goal for player is to be the first to build 5 housing units (the brown material). Image credits: Quartara, Emmanuelli, Montnemery.

The project *How High* was created by Andrea Quartara, Marc Emmanuelli, and Ulrik Montnemery. Their team followed an achievement-based game design. The first player who builds five housing units wins the game. The player journey to victory was controlled with the playable voxel-shape grammar (Figure 8.69). It required the placing of infrastructural elements first to collect the needed materials as well as unlock the grammar keys to place a housing unit (Figure 8.70). The evolution of the grammar rules was evident and easy to follow in this project (Figure 8.71). The authors went through several cycles, which begin with defining vocabulary shapes, rewards, and grammar rules. Then they played the game and checked whether the game design is balanced, i.e., if it is not too easy to build a housing unit or too tedious to create the required infrastructural units, is the game economy creating a challenging yet rewarding player experience. The authors selected one of the player-created designs to post-process. They ran a structural analysis using the live-link to Grasshopper (Figure 8.72) and built a physical model of it with the robot (Figure 8.73).

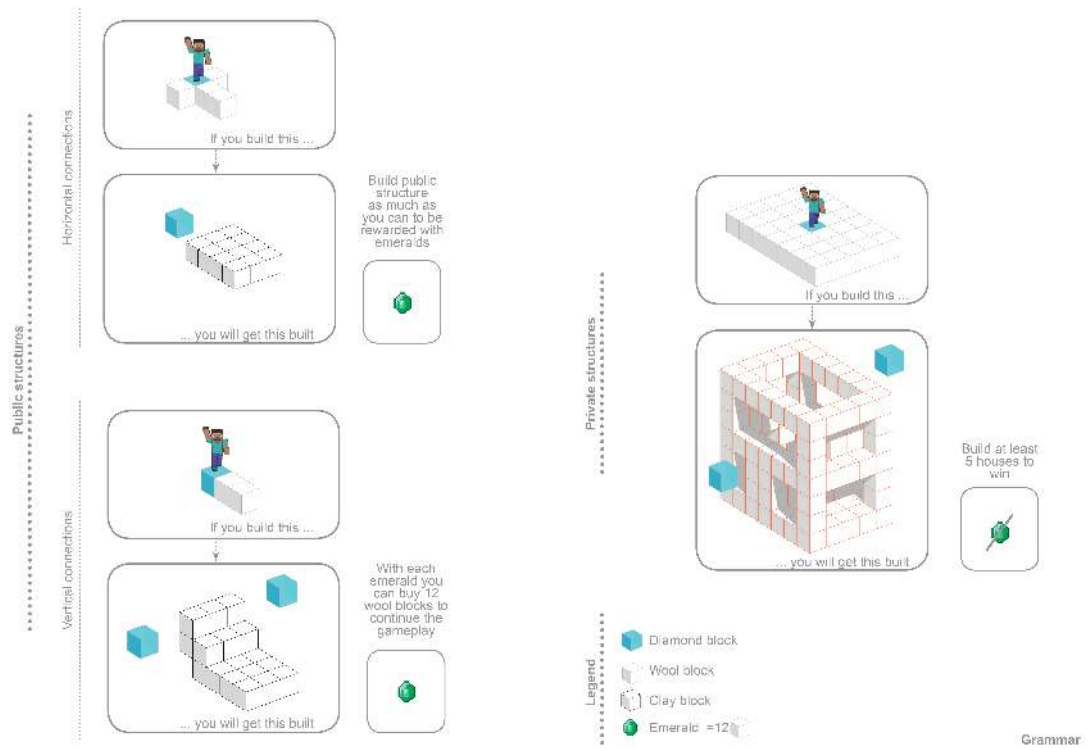


Figure 8.69 – The grammar in the 20.000 BLOCKS project How High. It consisted of 3 rules. One for horizontal connectivity, one for vertical connectivity (stairs), and one for housing units. Image credits: Quartara, Emmanuelli, Montnemery.

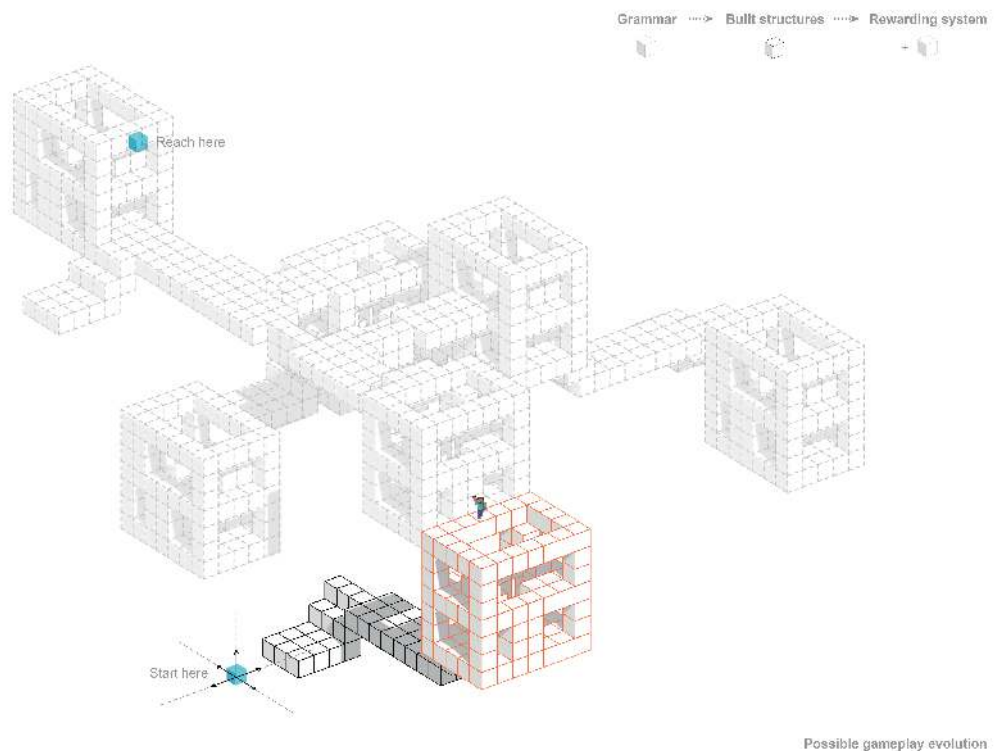


Figure 8.70 – A game sequence. Image credits: Quartara, Emmanuelli, Montnemery.

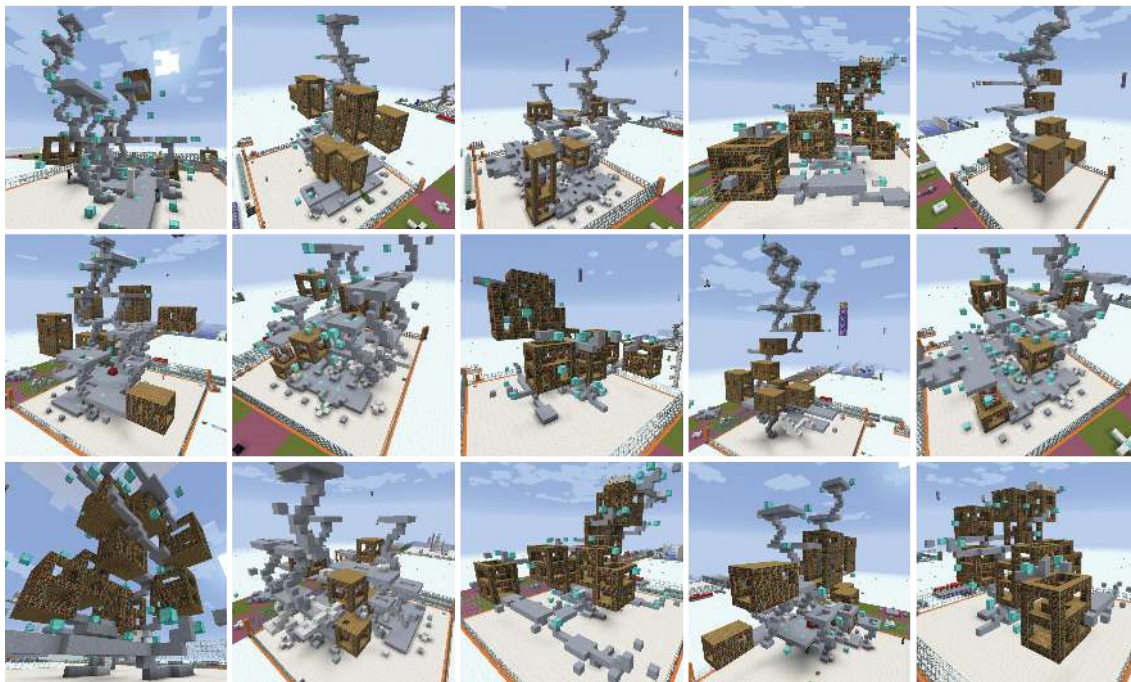


Figure 8.71 – Grammar evolution. Screenshots of designs created with various iterations of the grammar rules and vocabulary of the project *How High*. The evolution of grammar rules and vocabulary shapes goes always through several cycles of define-play-review. Image credits: the author.

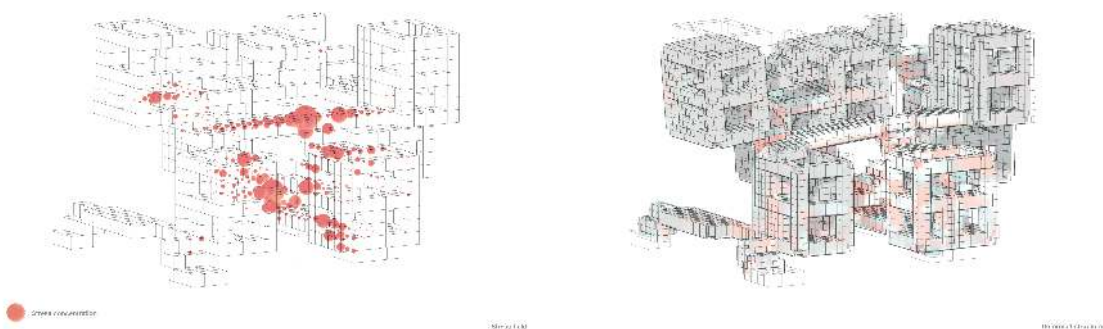


Figure 8.72 – Structural analysis of one on the player created designs in the *How High* project. Image credits: Quartara, Emmanuelli, Montnemery.

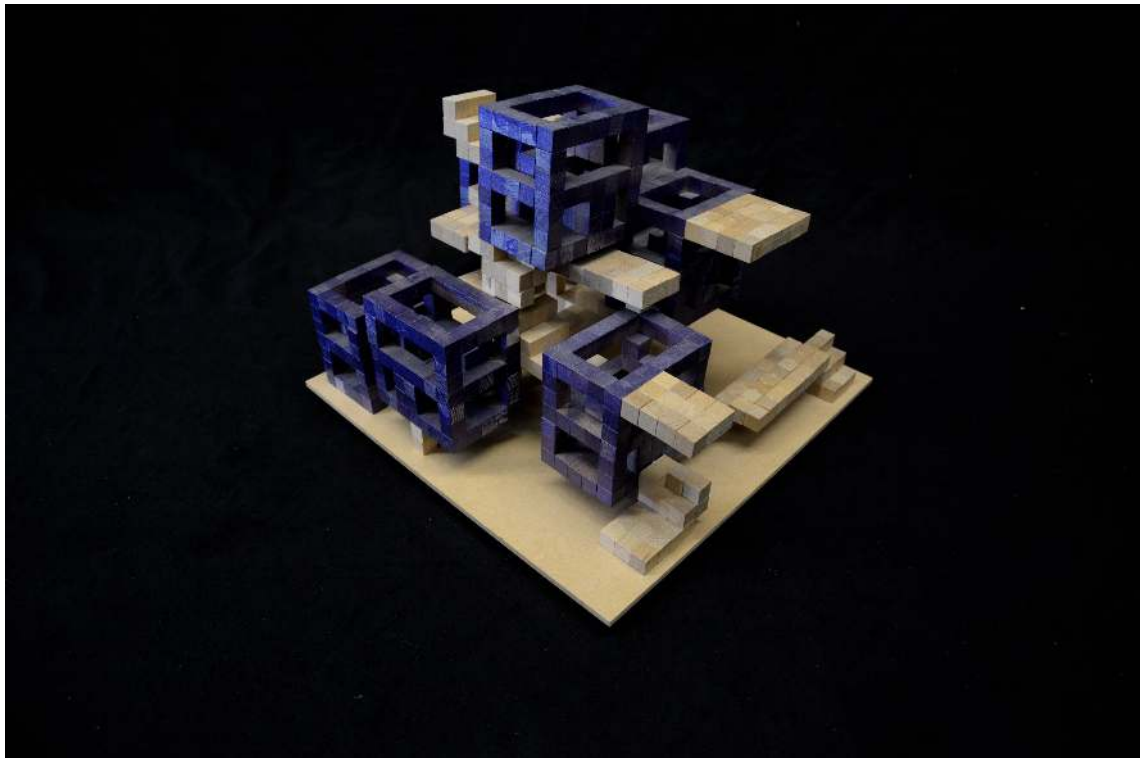


Figure 8.73 – Model photo. A robotically built model of a player-created design from the project *How High*. Image credits: the author.

Bridging



Figure 8.74 – The 20.000 BLOCKS project Bridging. It lets players collaborate in two competing teams to create bridge-like residential buildings over linear obstacles such as railways. Image credits: Gösta, Eliasson, Choudhury.

Bridging is a project from the *20.000 BLOCKS* cluster at SmartGeometry 2016 designed by Alexander Gösta, Samuel Eliasson, and Ashris Choudhury. The architectural goal of the project is to create an alternative design for inhabitable bridges over a river, a railway, or a highway (Figure 8.74). The game design combines both competitive and collaborative strategies. The players in *Bridging* are split into two teams, each starting on a platform on the opposing sides of a dividing wall (Figure 8.75). The first team to build a bridge from their platform to the middle of the wall ridge wins the game. The grammar consisted of elements that incorporated a living unit and allowed either forward, sideways or upward aggregation (Figure 8.76). A typical game progression is shown on Figure 8.77. Figure 8.78 shows two designs created by players. The project authors selected one player-created design to post-process and analyze (Figure 8.79). Figure 8.80 shows the physical model of the chosen design, built by the robot.

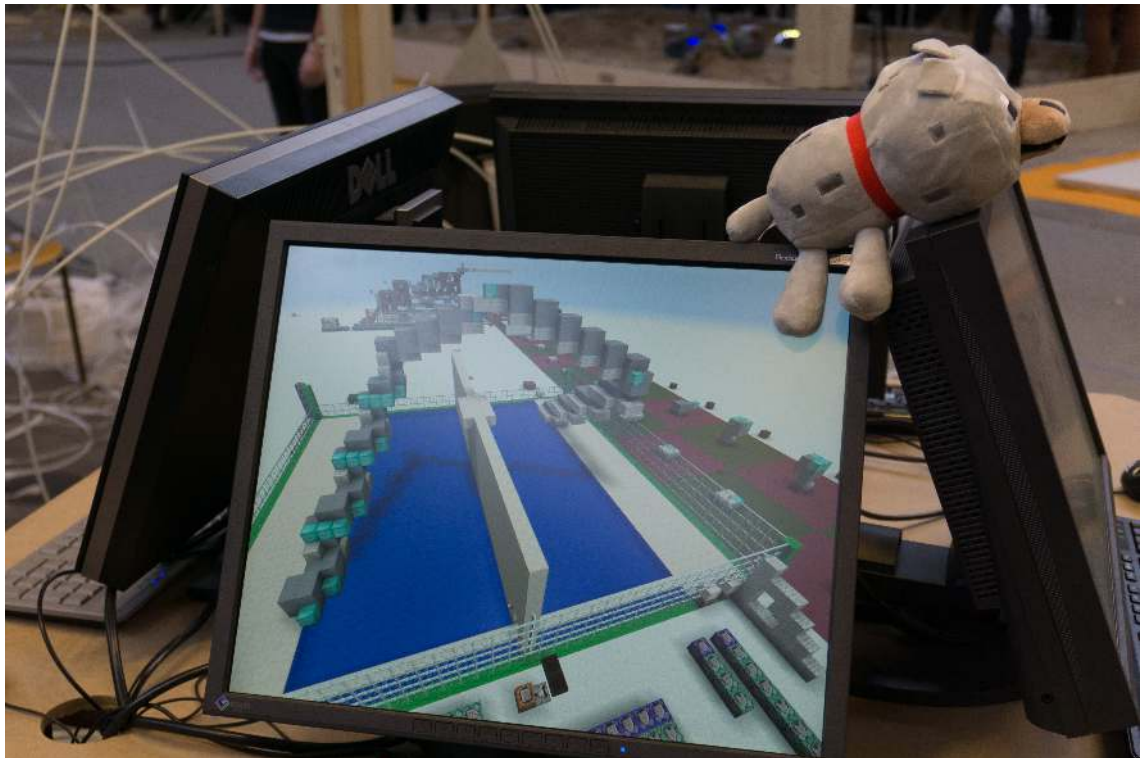


Figure 8.75 – Game result from the Bridging game. Image credits: Jörg Hartmann.

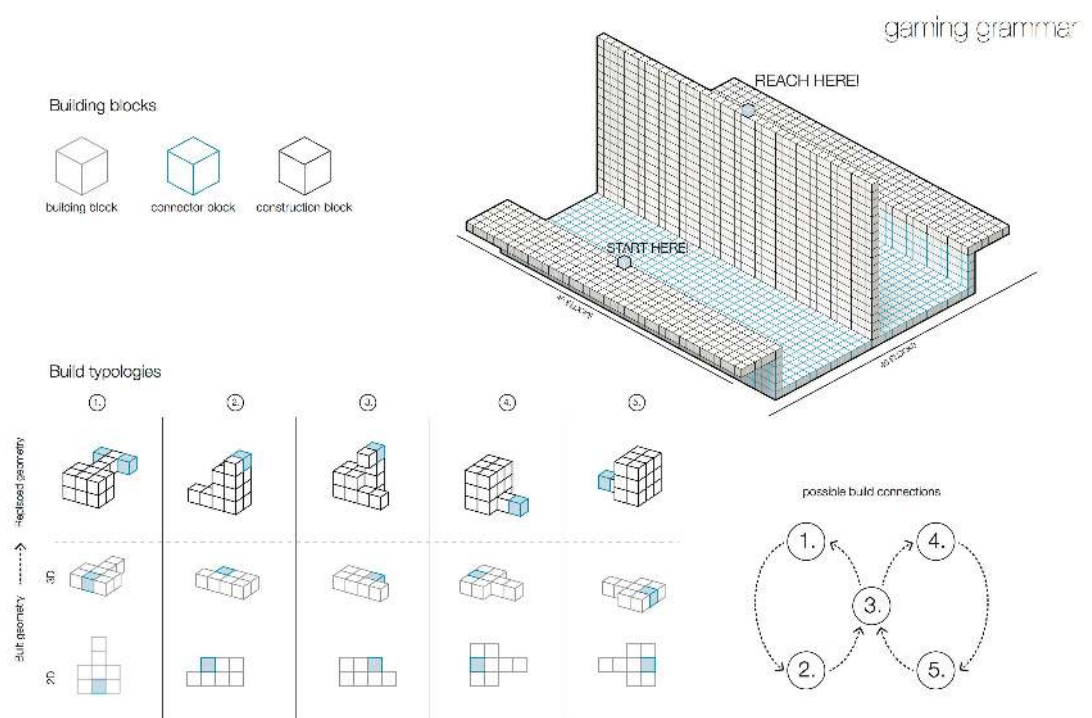


Figure 8.76 – The vocabulary from the project Bridging. Image credits: Gösta, Eliasson, Choudhury.

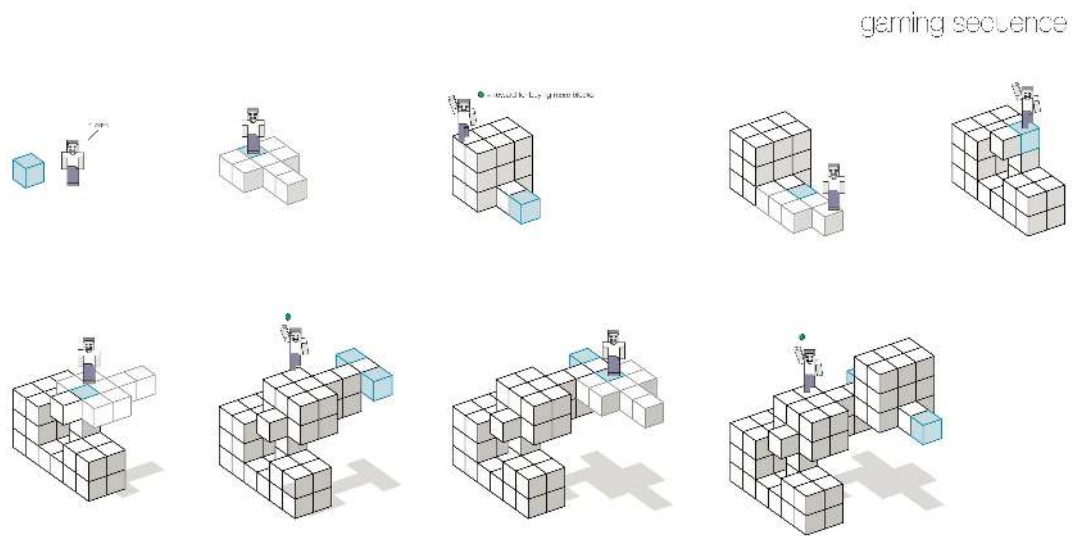


Figure 8.77 – Vocabulary-based game rewards. Players are incentivized to guide the growth process of the design towards achieving high score. Image credits: Gösta, Eliasson, Choudhury.

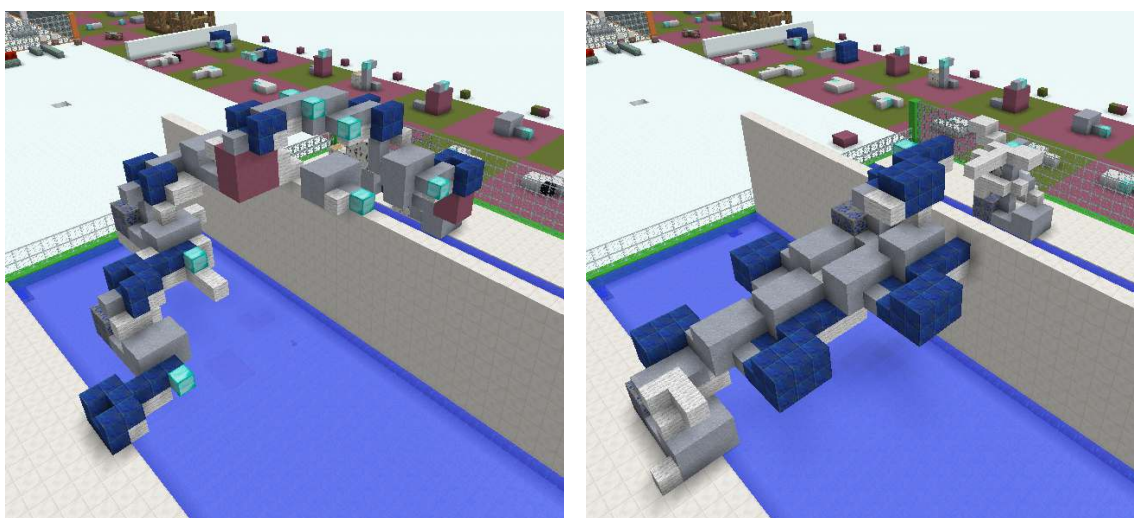


Figure 8.78 – Two designs created by players of the Bridging game. Parts of the vocabulary catalog area is visible in the background. Image credits: the author.

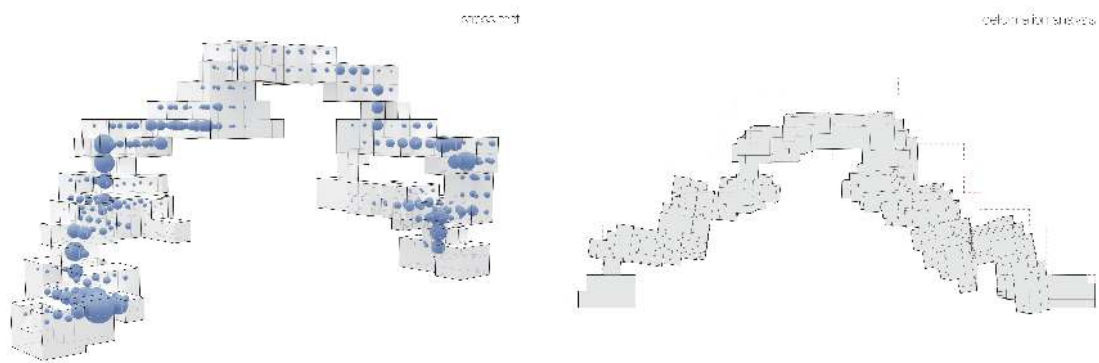


Figure 8.79 – Structural analysis of a player-created design in the project Bridging. It reveals the areas with highest rotational deformation (left) as well as simulates the deformation under self-weight (right). Image credits: Gösta, Eliasson, Choudhury.



Figure 8.80 – Model photo. A robotically built, physical model of a player-created design from the *Bridging* game. Image credits: Eliasson.

8.4.3 IBA_GAME



Figure 8.81 – IBA_GAME poster. Image credits: the author.

In *IBA_GAME*, players create hundreds of small neighborhoods to form a new city quarter. The city quarter is at the grounds of the former Patrick Henry Village (PHV) — located west of the city center of the German city of Heidelberg. The *IBA_GAME* was developed for IBA Heidelberg between August 2016 and March 2017.

Context

The PHV is a 100-hectare area, used by the U.S. Army until 2013 (Figure 8.82). Its most characteristic feature is the rows of barracks that housed the army officers.

Attracted by information about the Platform game that I posted online, the mayor of Heidelberg, Eckart Würzner, saw an opportunity to engage the younger citizens in the ongoing participatory design process for the PHV.

IBA Heidelberg approached us to create a game targeting 10-17 year-olds. IBAs — the International Building Exhibitions — are one of Germany's most influential and innovative urban development instruments in Germany, going back to the beginning of the 20th century.

The game, *IBA_GAME*, joined an extensive program for *Planning Phase Zero* of the PHV focussing on *knowledge-based urbanism*. The program had the world-renowned architectural offices of *MVRDV*, *Carlo Ratti Associati*, *ASTOC*, *Ramboll Liveable Cities Lab* and *Bohn&Viljoen* produce four visions for the neighborhood. A series of citizen forums meant to bring the architects into dialog with the locals (Figure 8.83). Finally, the architectural office *KCAP*, taking into account the four visions and the citizen input, produced a dynamic master plan for the future of the PHV (Figure 8.84).

Citizens interested in playing the *IBA_GAME* could do so at each of the three citizen forums as well as anytime from their home computer (Figure 8.85). We also held play sessions at the TU Darmstadt.



Figure 8.82 – Aerial photo of the Patrick Henry Village. Image credits: Stadt Heidelberg, Kai Sommer, 2010.

Our game’s purpose was two-fold. First, it attracted the younger generation that would otherwise not go to the citizen forum discussion. The younger kids had to ”bring” their parents, which further increased the discussion attendance.

However, the second purpose is didactically even more important. The game design of *IBA_GAME* did not envision clear winners. Instead, each player could choose to play for one out of four metrics: greenness, height, program diversity, and density (Figure 8.86). In the game, the players experience the trade-offs that maximizing one or another of the metrics would require. Hence the game developed a greater awareness among the participants and the architects alike that a participatory design process does not simply mean collecting a wish list of preferences but is an iterative process of establishing priorities and making trade-offs.

This unique context confirmed the practical relevance of the new modes of staging participatory design processes to be found in the intersection of crowdsourcing, game design, and generative design.

In the *IBA_GAME* project, a team of architecture students, a game designer, a game artist, and I were in the role of *experts* as well as *tool-makers*, i.e., we created the content and the rules of the game not only the game implementation.

Game description

Upon joining the *IBA_GAME* server, players are teleported to a tutorial area (Figure 8.88) that introduces them to the context of the game and the game rules.

Three hundred square build zones make up the territory of the new city quarter (Figure 8.87). Each game that players play fills up one of them. A game starts as an empty zone with 7x7 slots, where players can place buildings. They create a new neighborhood by building houses and extending them with various functions such as rooms, gardens, businesses, etc.



Figure 8.83 – The participatory process for Planning Phase Zero of PHV. Top: A workspace prepared for discussion with the citizens. Bottom: Architect Winy Mass from *MVRDV* explaining to the audience his vision for the *Patrick Henry Village*. Image credits: IBA Heidelberg, Christian Buck, 2017.

The game is based on the *20.000 BLOCKS* framework and features a voxel-shape grammar of 40 buildings of six types: houses; housing extensions; businesses; science and tech spaces; gathering spaces; and public plazas (Figure 8.38). Each building type has variations for both the URBAN and the NATURE resources. If players bridge between two buildings, they get four varieties of TECH & SCIENCE SPACES depending on the orientation and resource they use. If players extend a corner of a building, they create GATHERING PLACES. There are eight varieties, and one can create other buildings on their roofs. A double extension creates BUSINESSES. There are eight varieties of businesses.

IBA_GAME had 140 players who played it 820 times. The shape grammar rules went through about ten significant revisions in the course of development.



Figure 8.84 – The dynamic masterplan for PHV by KCAP. Left: A rendering from Carlo Ratti's vision for the PHV. Right: The IBA Heidelberg team, the invited architects and the *IBA_GAME* team in front of the big model of KCAP's dynamic masterplan for PHV. Image credits: Carlo Ratti Associati, IBA Heidelberg, Christian Buck 2016.



Figure 8.85 – Photos from *IBA_GAME* play sessions. Image credits: IBA Heildeberg, Christian Buck, DDU, Tabita Cargnel, 2016.

Vocabulary evolution

Over a month, we tested several iterations (Figure 8.90) and approaches to creating a vocabulary at the urban scale for *IBA_GAME*. Figure 8.89 shows the principle that proved helpful in practice when designing a new vocabulary iteration. First, one starts testing the vocabulary as 2D plates to make sure the city territory can be covered well, then moves to 3D wireframing to set up the rules of vertical stacking and then fleshes out the elements in the last stage. The needed corrections could be implemented at each stage without throwing out too much work.

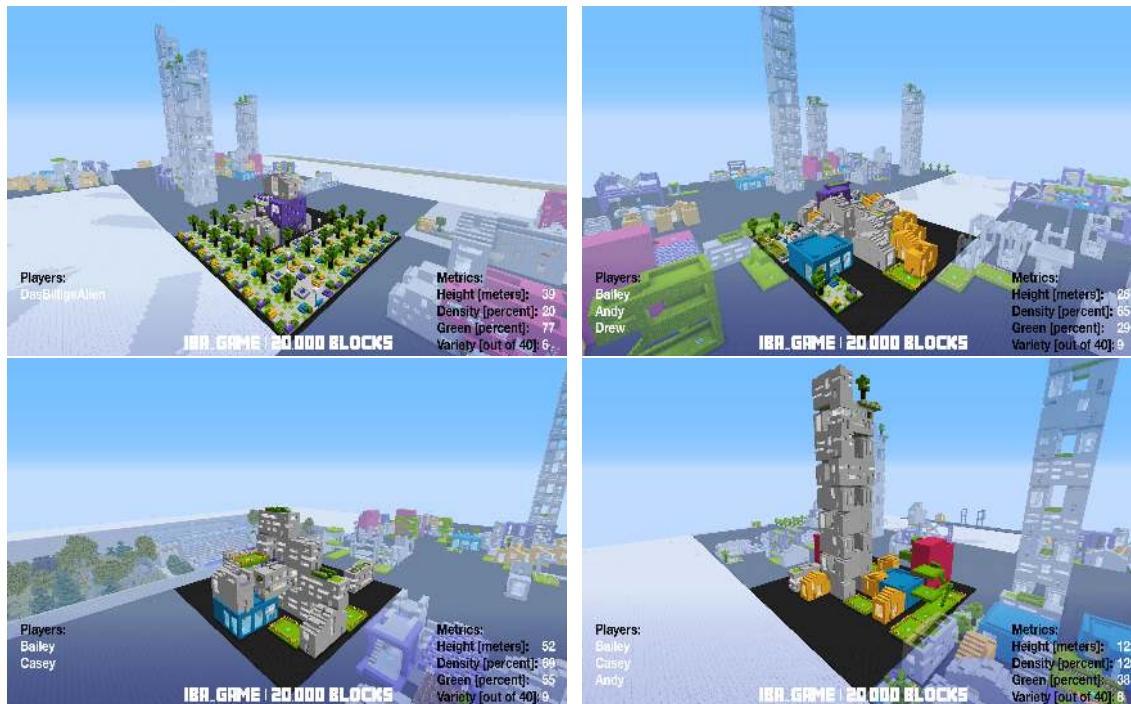


Figure 8.86 – Neighborhoods created by players in IBA_GAME. The four samples here represent the diversity of outcomes depending on which of the four metrics the players decided to prioritize: greenness, height, program diversity, and density. Image credits: the author.

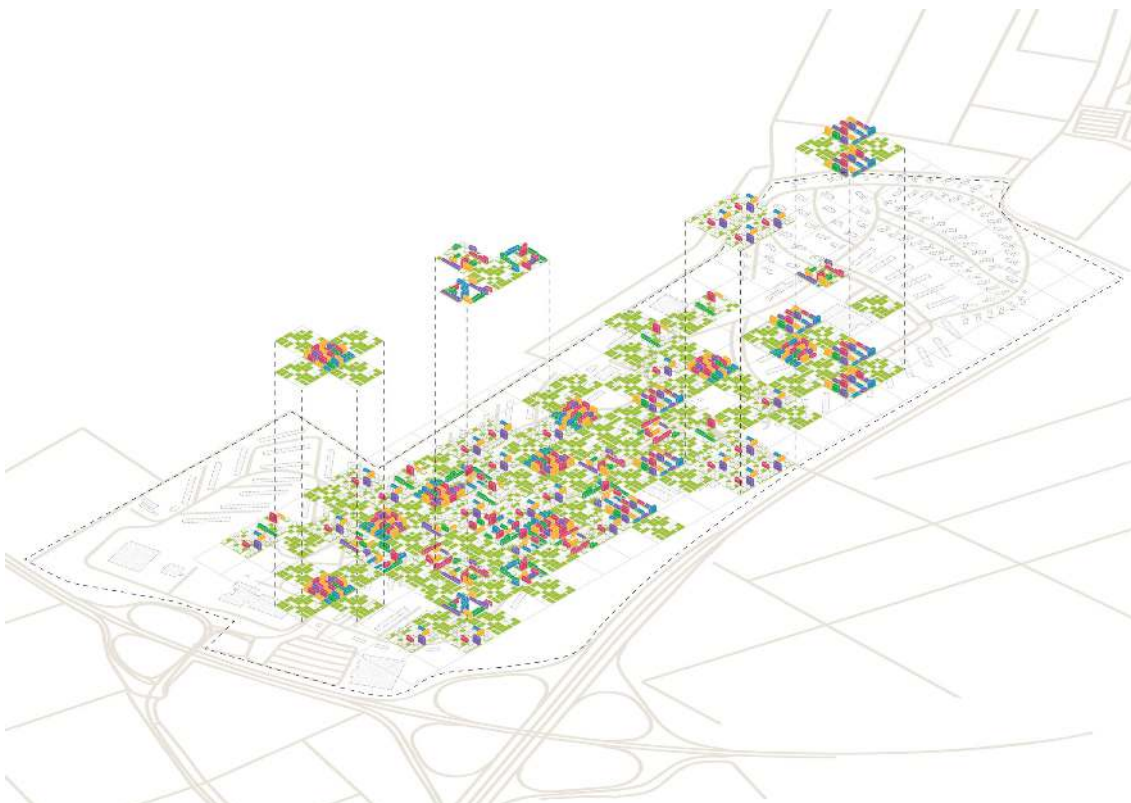


Figure 8.87 – The Patrick Henry village divided into neighborhoods in the IBA_GAME. Image credits: Marios Messios.

Game design

The grammar principle that found ground in our playtesting was treating houses as units a player can place anywhere on the board. Players could extend a house with

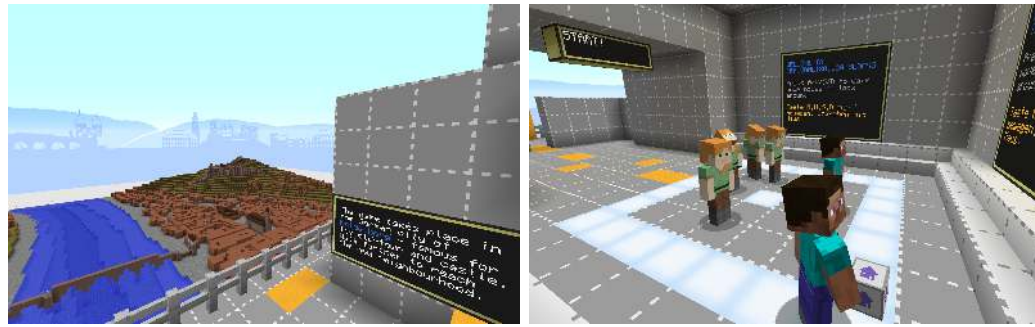


Figure 8.88 – IBA_GAME tutorial area. Left *IBA_GAME* takes place in Heidelberg. Right players entering the game. Image credits: the author.



Figure 8.89 – The three stages of developing a new vocabulary iteration. Left: land-blocking, each element is modeled as 2D plates. Middle: wireframing to establish vertical stacking rules. Right: flesh out the elements to give them architectural details. Image credits: Ben Buckton.

private spaces (Figure 8.91), but also complement it with commercial and public functions (Figure 8.92). This ensured coherence of the urban designs and avoided fragmentation. Each typology of elements received its own color: housing (white), business (blue), tech and science (pink) and entertainment (yellow) (Figure 8.38). The territory of the Patrick Henry Village was divided into square neighborhoods in a grid. Players could play the neighborhoods one by one and fill out the whole settlement. For each neighborhood, four parameters help players understand the quality of what they are designing: Height, Density, Greenness, Variety.



Figure 8.90 – Grammar and vocabulary evolution. Each row shows a selected iteration from the evolution of the shape grammar used in *IBA_GAME*. Working with generic architectural fragments, shown in the top and the second row, proved too complex for players. The more intuitive approach was to treat elements as means to block out the city's territory (lower two versions). Image credits: the author, Messios, Rudolph.

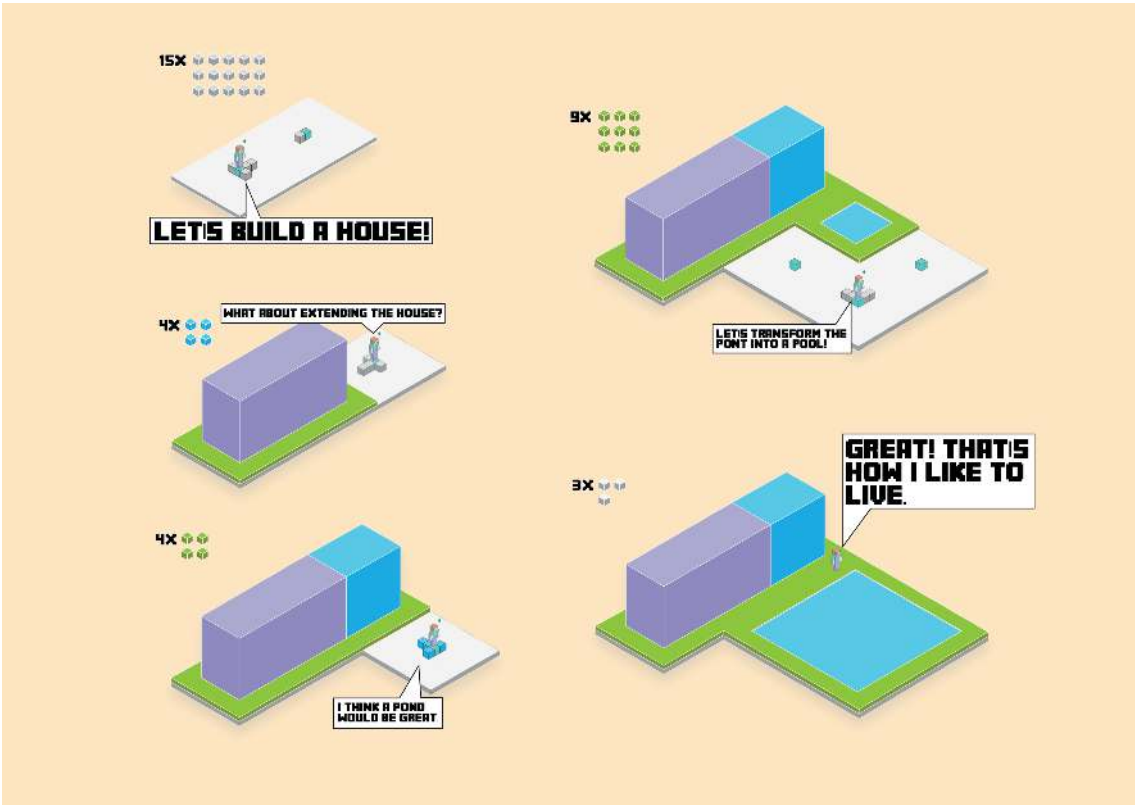


Figure 8.91 – A grammar logic for extending a house. Image credits: Max Rudolph.

<div>house</div> <div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>2 free slots 2 diamonds</div><div><div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>places 2 golds</div><div><div><div></div><div></div></div></div></div>	<div>extension row</div> <div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>1 free slot 1 diamond 1 gold</div><div><div><div></div><div></div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>places 1 gold</div><div><div><div></div><div></div></div></div></div>	<div>extension major</div> <div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>4 free slots 4 diamonds 1 gold</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>places 2 golds 1 green 1 blue next floor?</div><div><div><div></div><div></div></div><div><div></div><div></div></div></div></div>	
	<div>pool</div> <div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>3 free slots 3 diamonds 1 blue</div><div><div><div></div><div></div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>places nothing</div><div><div><div></div><div></div></div></div></div>	<div>filler blocks</div> <div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>1 free slot 1 diamond</div><div><div><div></div><div></div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>places nothing</div><div><div><div></div><div></div></div></div></div>	
<div>extension rise</div> <div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>2 free slots 2 diamonds 1 gold</div><div><div><div></div><div></div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>places 1 gold 1 gold up</div><div><div><div></div><div></div></div></div></div>	<div>bridge</div> <div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>1 free slot 1 diamonds 2 golds</div><div><div><div></div><div></div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>places 3rd floor 2 diamonds up</div><div><div><div></div><div></div></div></div></div>	<div>extension pond</div> <div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>1 free slot 1 diamond 1 gold</div><div><div><div></div><div></div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>places 1 blue</div><div><div><div></div><div></div></div></div></div>	<div>extension green</div> <div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>1 free slot 1 diamond 1 gold</div><div><div><div></div><div></div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div>places 1 green</div><div><div><div></div><div></div></div></div></div>

Figure 8.92 – The grammar logic for extending the buildings in IBA_GAME. Image credits: Max Rudolph.

8.4.4 Combinatorial Design

Combinatorial Design is a Student Course I thought in the winter term of 2017-2018 at the TU Darmstadt. A total of 12 students worked in four teams. Students worked at the intersection of Architecture and Game Design to explore the design problem of density. The design goal is to achieve a design with the highest occupied volume providing the highest possible quality of life (Figure 8.93). Students decide how *quality of life* is defined for their project.

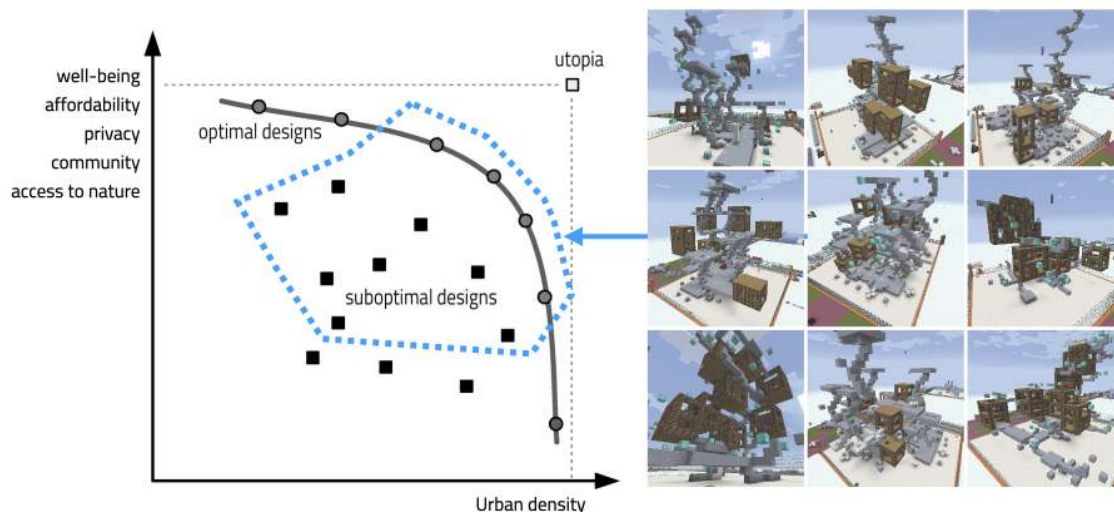


Figure 8.93 – Pareto front of residential designs. On a chart where density is represented as the X-axis and quality of life as the Y-axis, there is a line where no higher density can be achieved for a given quality of life. This is the Pareto front. While the front extremes are well explored in architecture with single-family housing urban sprawl and residential high-rises, the middle is less known and probably rich in interesting residential models. *Combinatorial design* aims to create games that enable the exploration of design in the middle. Image credits: the author.

Students begin with an existing building and use its organizational logic to create alternative designs with higher or lower density. The reference must be an existing residential building with the following qualities:

- low rise - max 7-8 floors
- high density
- mostly residential function
- orthogonal designs (no curves) for compatibility with the voxel space
- existing buildings only - no-unrealized projects

Students were introduced to the concept of shape grammars — collections of visually defined, geometric rules used to automate the generation of formal representations of designs for buildings and cities. Each team extracts the essential elements of an existing building and models them as a shape grammar in Minecraft. Students document the combinations that other players create using these elements and compare them to the original building (Figure 8.94).

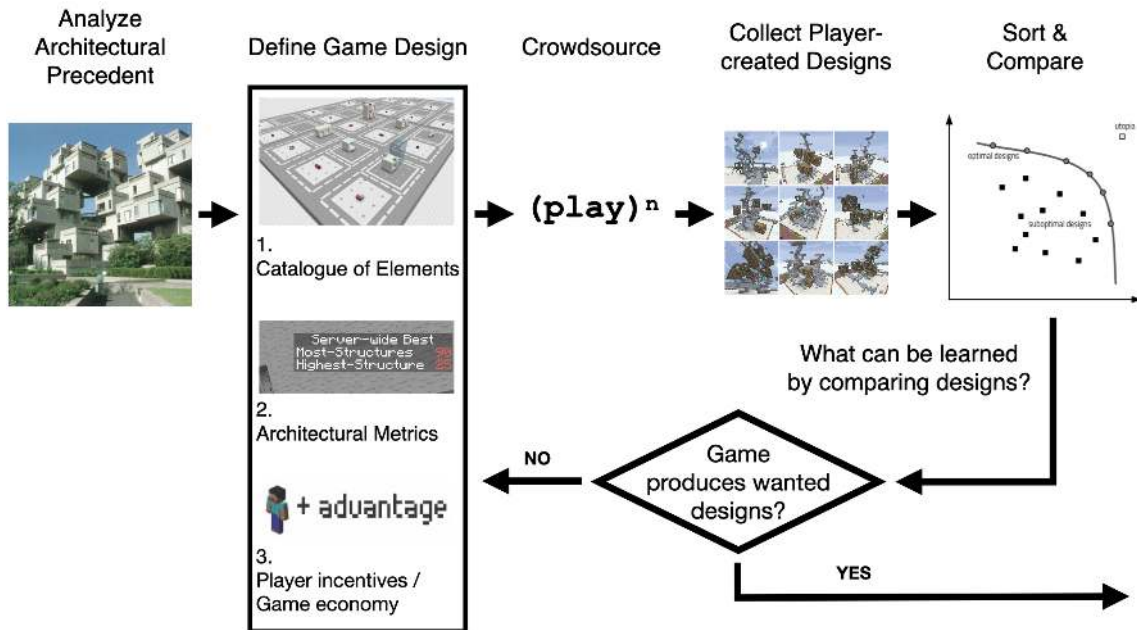


Figure 8.94 – The method of the course Combinatorial Design. At the core is the fine-tuning of game design (shape grammar, score system and game economy) until the game produces design with desired density and qualities. Image credits: the author.

Each project represents the spatial organization of a residential building as an economy of occupied spaces in a discrete 3D matrix (Figure 8.97) and the accessibility of these spaces to the ground, the sun, and other aspects (Figure 8.98). The design constraints are encoded as combinatorial and shape rules in the grammar. An example is given on Figure 8.96. In Habitat 67, having a garden is a design constraint. It is encoded in a shape grammar so that the garden's minimum space is marked as occupied when an element is placed. An important question is at what compromise for the quality of living is higher density possible, and can it be measured? Each project transformed the rules governing its economy into game mechanics.

The students received as an example the analysis of the Habitat 67 building in Montreal by Moshe Safdie (Figure 8.95). A grammar defined in *20.000 BLOCKS* based on the building's complex spatial organization was also given to students.



Figure 8.95 – Habitat 67 in 20.000 BLOCKS. The building Habitat 67 in Montreal by architect Moshe Safdie - left in reality and right, as recreated with a playable voxel shape grammar in Minecraft using *20.000 BLOCKS*. Image credits: n.n..

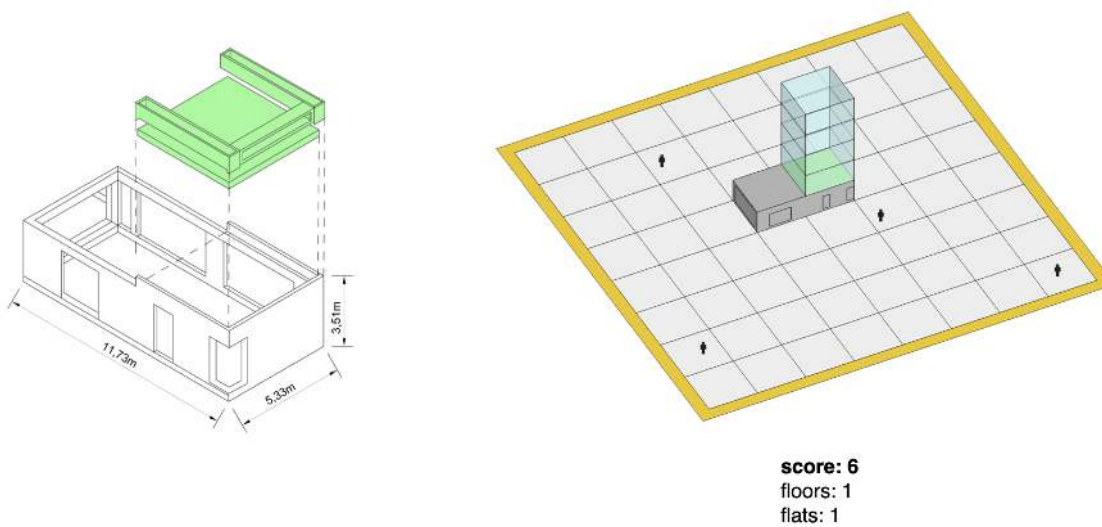


Figure 8.96 – A residential unit from Habitat 67. Remodeled as a game piece in *20.000 BLOCKS* that includes the four levels of air above the garden present in the original building. As such the game piece occupies 6 slots in the 3D matrix - two for the unit and 4 for the garden. Image credits: Mingquan Ding.

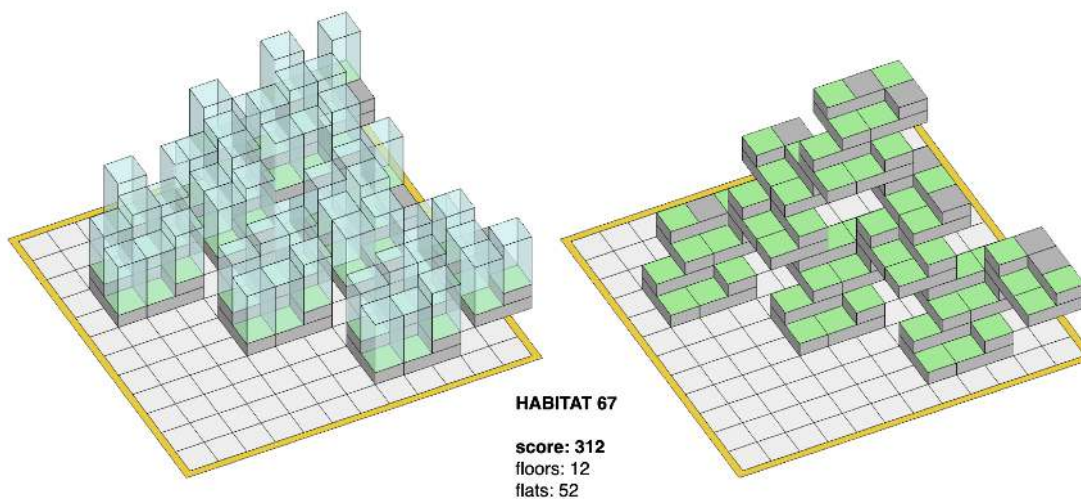


Figure 8.97 – Density as score. By counting the cells a design occupies it can be assigned a measurable score for density. Image credits: Mingquan Ding.

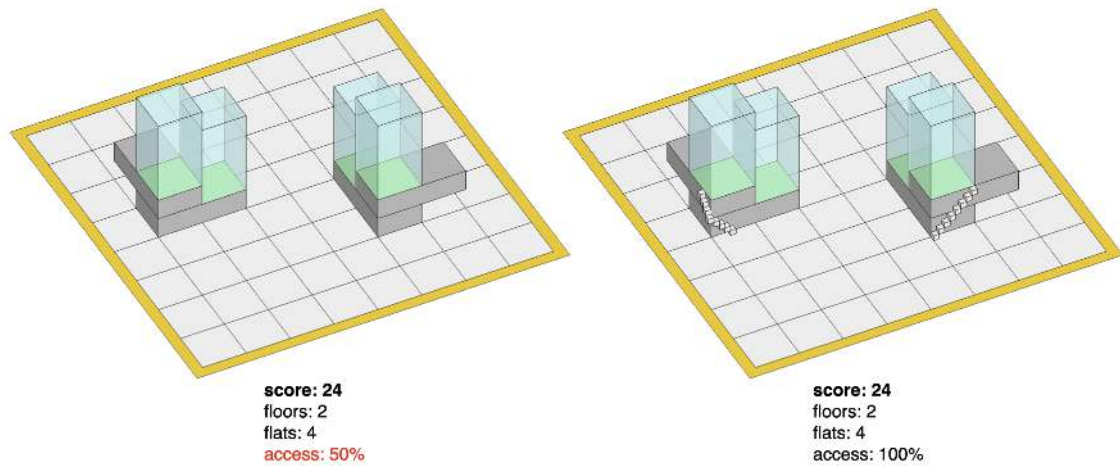


Figure 8.98 – Access rule. Density is not all that matters for quality of living. For example, a very densely packed design might result in a unit without access to sunlight or even to the street. Four units are placed in the example on the left, but only two have access to the ground. The players can freely build access routes to improve the design quality, as shown on the right. Image credits: Mingquan Ding.

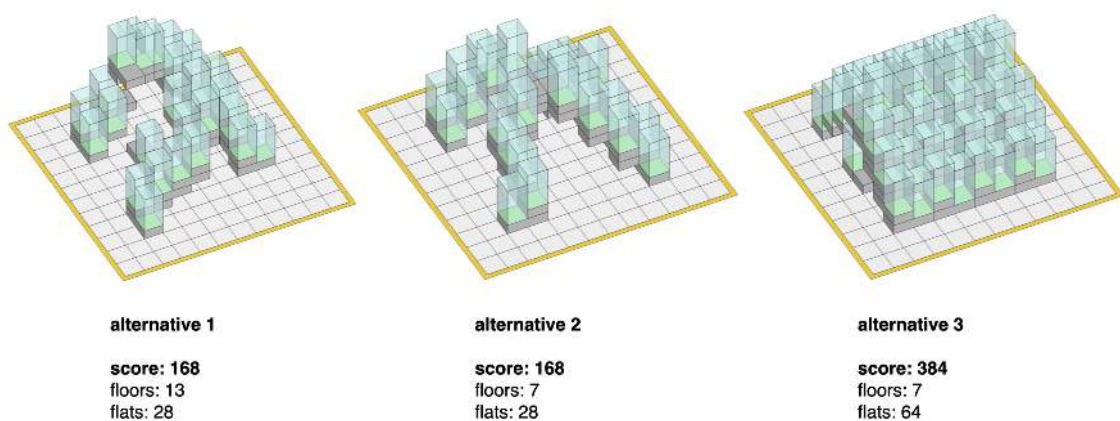


Figure 8.99 – Designs from the Habitat67-inspired grammar. Image credits: Mingquan Ding.

The Carve

The project *The Carve* was developed by Annabell Schneider and Lukas Stöckli. The architectural reference (Figure 8.100) they chose is an apartment building on the waterfront in Oslo named *The Carve* designed by the Norwegian architectural office *A-Lab*.

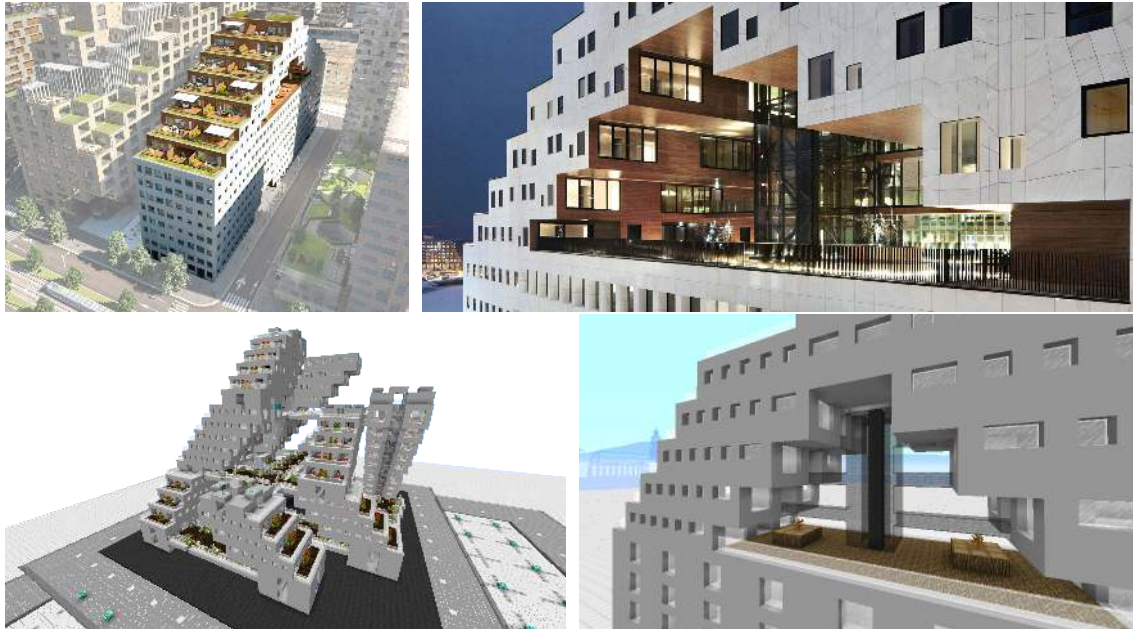


Figure 8.100 – The Carve. Top: the reference building The Carve. Bottom: Replication and Variation using playable voxel-shape grammars in *20.000 BLOCKS*. Image credits: A-Lab, Schneider, Stöckli.

The building analysis and the design goals lead to the grammar rules on Figure 8.101. On the ground, players place foundation units. The apartment units can be placed only on the foundation units, not directly on the ground. Apartment units have windows on one side and a terrace on the other. Airspace was reserved on both window and garden sides to ensure enough sunlight inside the rooms.

There are shared terraces between the buildings for the units inside the courtyard. The authors called them *collaborative terraces*, and they are an essential part of the gameplay. Players are given the points for placing an apartment only if it is within four cells from a collaborative terrace, thus ensuring designs that are interconnected and form communities (Figure 8.102 left).

The other important aspect of the game's design is the direction change. Only two orientations for the flats are possible in the original building — east and west. To give the structure more chances to grow in a space, it's possible to place the units in every direction. And players are rewarded for direction change with extra points, which ensures designs that are porous and let sunlight and views inside (Figure 8.102 right).

The authors decided to half the scale of the Minecraft representation to make gameplay easier and faster, i.e., 2 meters in the original building equal one block in Minecraft. The grammar development went through several iterations (Figure 8.103).

The final shape grammar, shown on Figure 8.104, consisted of four elements: foundation, terrace, connection, and apartment. A screenshot of the catalog as

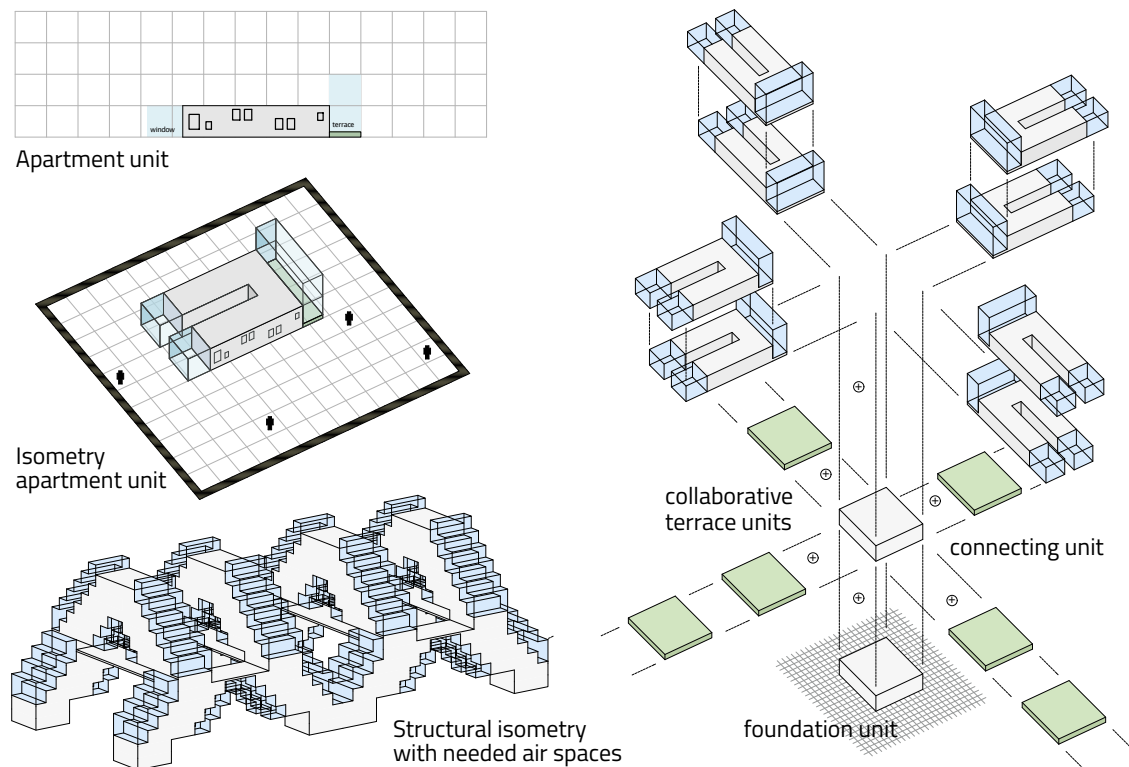


Figure 8.101 – The organizational principle encoded in the grammar. Image credits: Schneider, Stöckli.

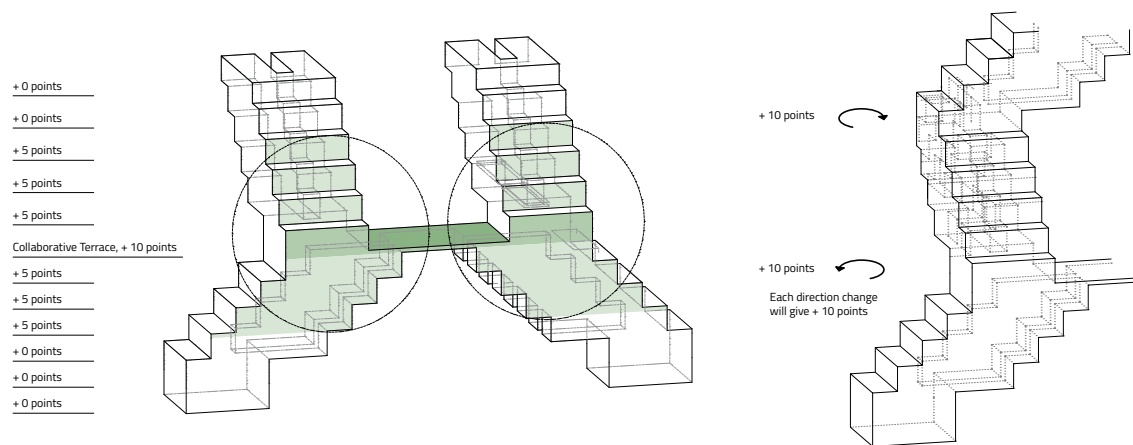


Figure 8.102 – Game mechanics. Left: the collaborative terraces game mechanic. Only apartments within four cells of a shared terrace element will bring points to players. Right: the orientation change game mechanic. Players get 10 extra points for changing the orientation of apartments. Image credits: Schneider, Stöckli.

modeled in Minecraft is shown on Figure 8.105, and the whole game area is shown on Figure 8.106. Although relatively simple, the grammar allows the creation of visually and spatially rich designs (Figure 8.107).

Eight game results were collected, analyzed, and plotted on the Density vs. Quality chart on Figure 8.108. The authors used the Collaborative Terraces points to measure the building's community life quality. The players contributing the designs were categorized into Beginners, Hobby, or Professional Gamers (Figure 8.109).

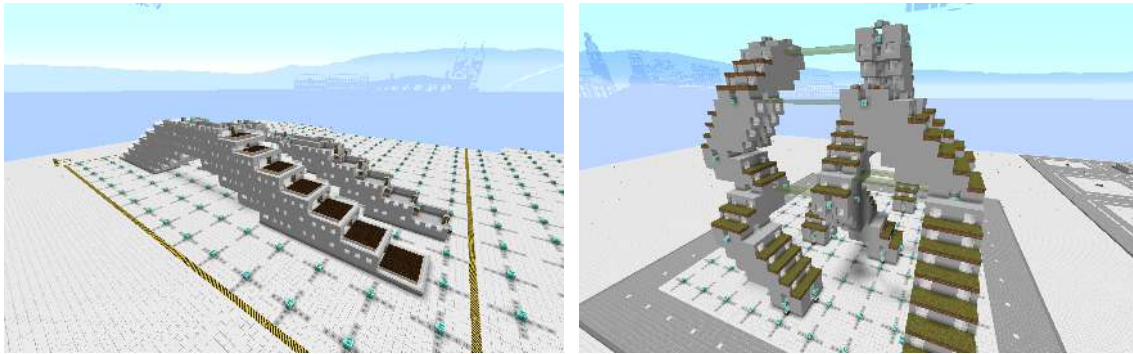


Figure 8.103 – Iterations from the grammar development of The Carve. Image credits: Schneider, Stöckli.

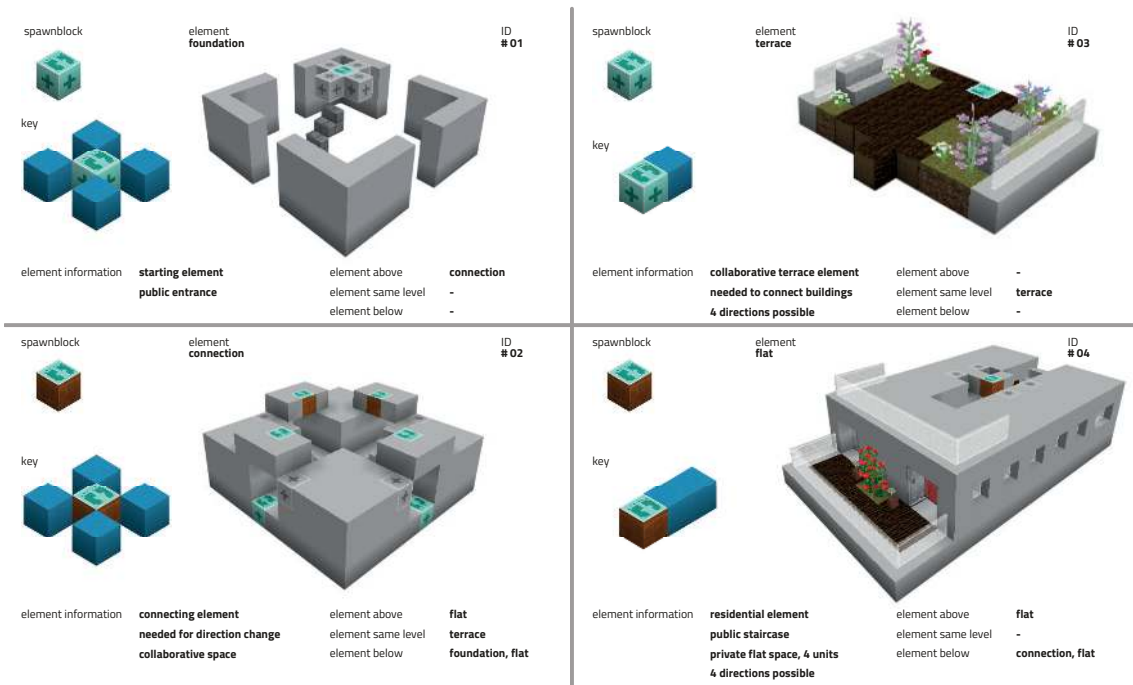


Figure 8.104 – The elements in the grammar of The Carve with their keys and activator blocks. Image credits: Schneider, Stöckli.

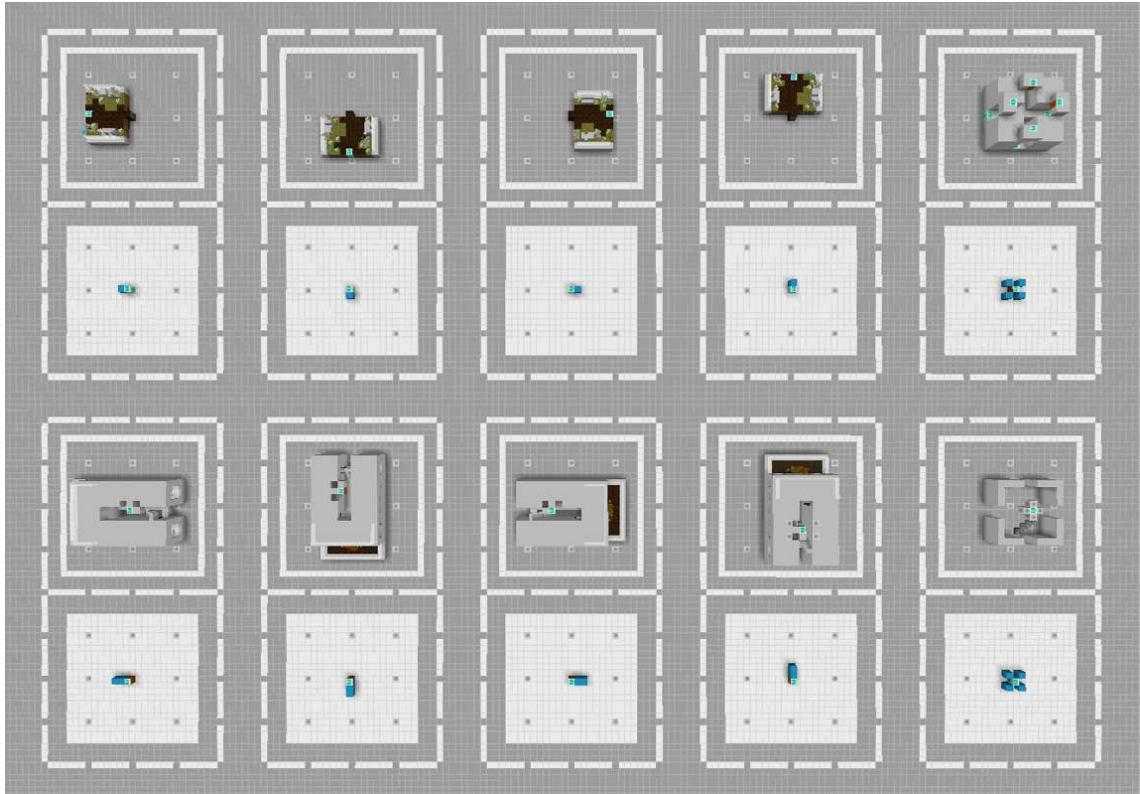


Figure 8.105 – A top view of the catalog area in Minecraft. Image credits: Schneider, Stöckli.

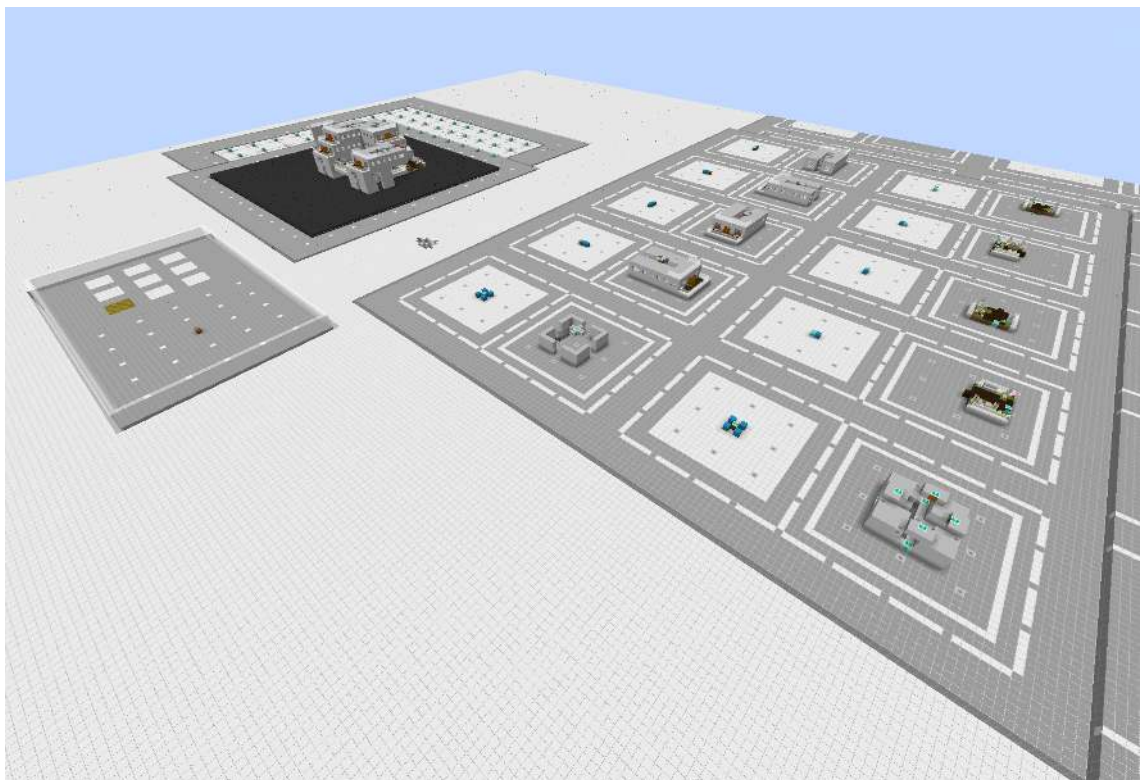


Figure 8.106 – The game area. With the catalog in the front, the building zone in the far back and the spawn area with game logic computer on the left. Image credits: Schneider, Stöckli.



Figure 8.107 – Variety of player-created design in *The Carve*. Image credits: Schneider, Stöckli.

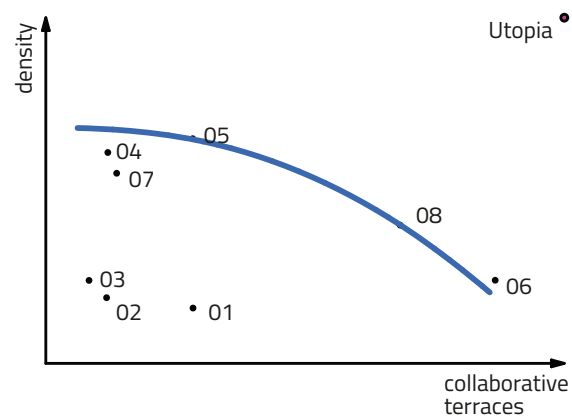


Figure 8.108 – Pareto Front. The 8 designs created by players in *The Carve*, plotted according to their score on Collaborative Terraces and their Density. Image credits: Schneider, Stöckli.

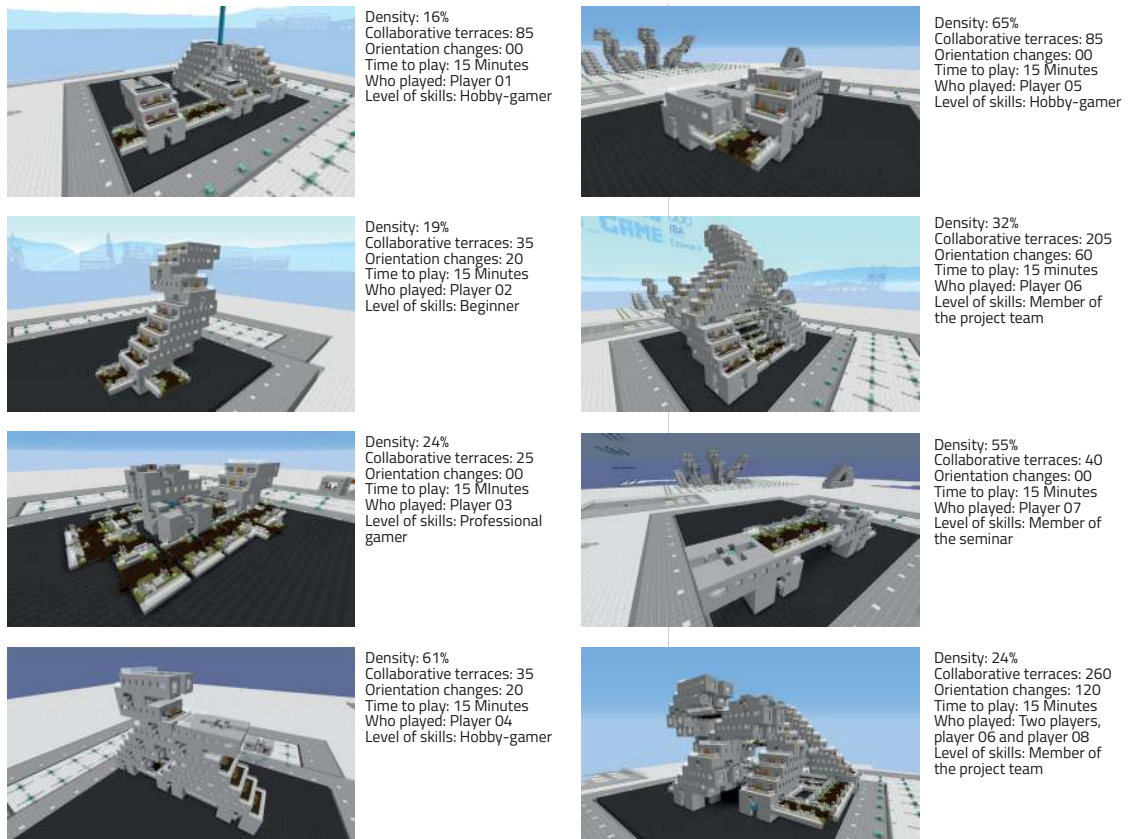


Figure 8.109 – The eight designs created by players in *The Carve*. Image credits: Schneider, Stöckli.

Juvika



Figure 8.110 – Juvika. The reference building *Space Blocks* in Hanoi and a game result from the *20.000 BLOCKS* project *Juvika*. Image credits: Kazuhiro Kojima, Schäfer, Abu-salha, Mayer.

The project *Juvika* was developed by Julia Schäfer, Viola Abu-salha and Alexander Kay Mayer. The reference they chose is the *Space Blocks* building in Hanoi, Vietnam, by architect Kazuhiro Kojima from 2003 (Figure 8.110).

The project follows the linear organizational logic of the reference building, which has a length of 80 meters (Figure 8.111). Two other characteristic features of the build are turned into game mechanics. First, the building is oriented inwards and has multistory voids, which result in a rule not to block three cells of air next to each element (Figure 8.112). Second, stairs with various topologies connect the rooms of one apartment arranged around the multistory voids. Players are required to freely build the stairs to provide access to the rooms above ground (Figure 8.113).

The grammar reflects these mechanics and consists of four main elements: a common space, an apartment, and two spaces with vertical voids (Figure 8.114). Players get rewarded for placing elements next to each other or on top of each other (Figure 8.115). The point distribution incentivizes players to place apartments on or next to voids to increase density. At the same time, it discourages them from placing vertical voids on top of or next to each other to provide feasible apartment topologies.

A total of 18 designs were collected, analyzed and plotted on a chart comparing density vs total quality (Figure 8.116). The total quality is calculated with

$$G = P - |Q - 50| - |D - 50| + 0.5 * K,$$

where G = the Total Quality,

P = the Points acquired in the game,

Q = the sum of voids and common spaces placed in the game,

D = the relative density,

K = the complexity, i.e. the total number of occupied cells.

All 18 designs and their total quality points are given on Figure 8.117.

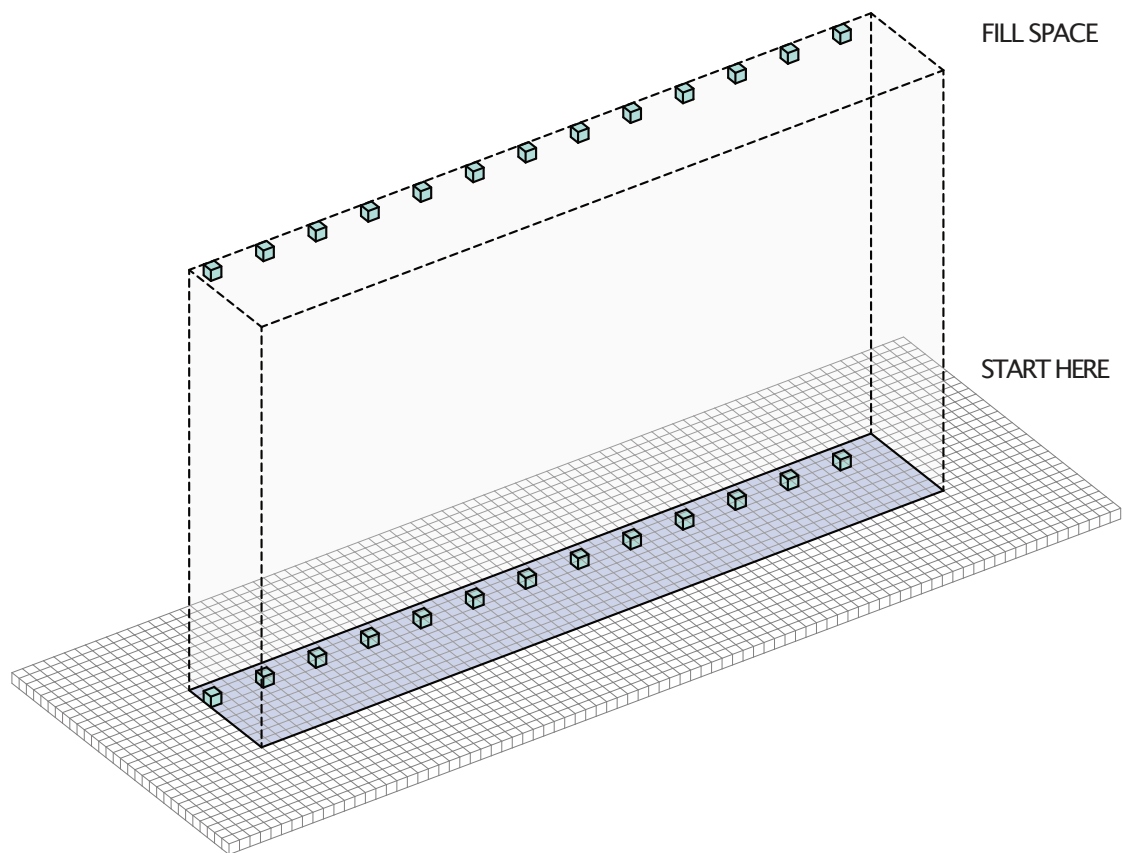


Figure 8.111 – The game field of Juvika. It reflects the linear composition of the reference building. Image credits: Schäfer, Abu-salha, Mayer.

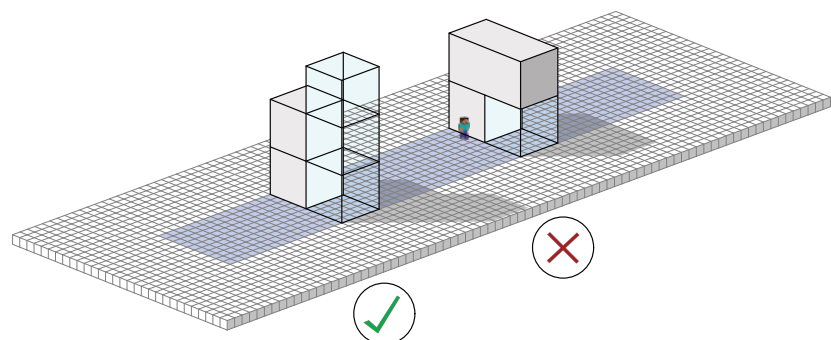


Figure 8.112 – The vertical voids rule in the grammar of Juvika. It ensures the inner spatial quality of the game results is similar to the reference building. Image credits: Kazuhiro Kojima, Schäfer, Abu-salha, Mayer.

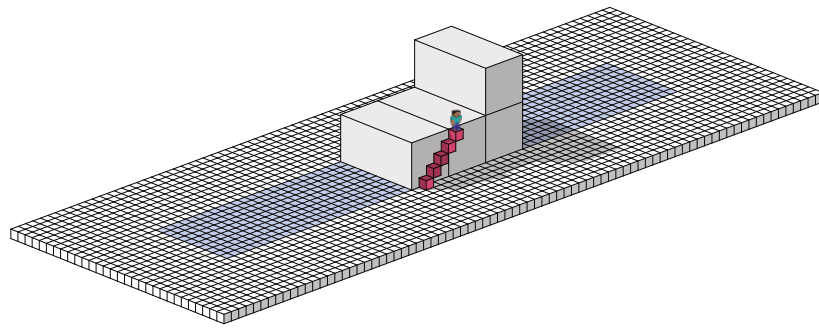


Figure 8.113 – The stairs game mechanic in Juvika. It lets players freely build stairs with the resources they win in the game to ensure the topologically diverse vertical communication present in the reference building. Image credits: Kazuhiro Kojima, Schäfer, Abu-salha, Mayer.

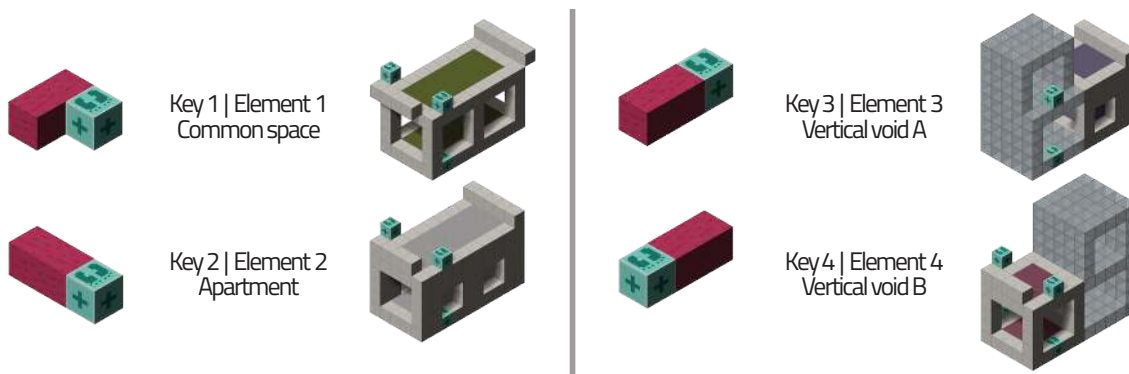


Figure 8.114 – The shape grammar in Juvika. Image credits: Schäfer, Abu-salha, Mayer.



Figure 8.115 – The point system in Juvika. It incentivizes players to increase density and build feasible apartment topologies. Image credits: Schäfer, Abu-salha, Mayer.

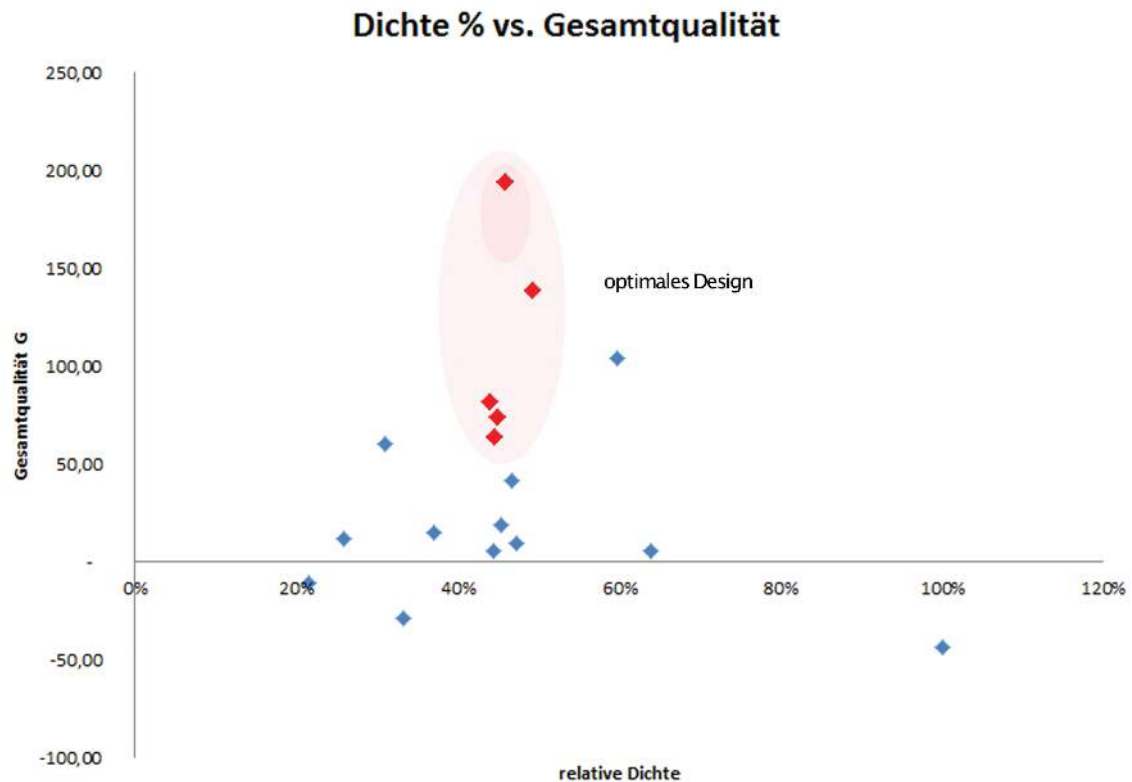


Figure 8.116 – Density vs Quality. The 18 designs created by players in *Juvika* are plotted according to their density and total quality. Image credits: Schäfer, Abu-salha, Mayer.



Figure 8.117 – All 18 designs collected in Juvika. Image credits: Schäfer, Abu-salha, Mayer.

Inakasa



Figure 8.118 – Inakasa. Left: the original Inakasa building in Las Palmas de Gran Canaria. Right: a photo collage of the Minecraft version of the building, created in the *20.000 BLOCKS* project *Inakasa*. Image credits: Acosta, Martín, Sajak, Markgraf.

The project *Inakasa* is developed by Julia Sajak and Moritz Markgraf. The reference they chose is the INAKASA building in Las Palmas de Gran Canaria from 2005 by architects Alexis López Acosta and Xavier Iván Díaz Martín (Figure 8.118). The building is a perimeter block development with an inner courtyard. It has a mixed program, with offices on the ground and first floor and 34 apartments on the upper four floors. The sloped opening connecting the courtyard to the street, and the sloped roof topology, require a level of abstraction when recreating them in the orthogonal world of Minecraft.

The shape grammar that the authors specified after analyzing the building is organized in enclosing elements instead of spaces which makes it very compact in comparison to other *20.000 BLOCKS* projects (Figure 8.119). The sizes of the vocabulary elements are in the closest integer values to the apartments' components derived in the analysis (Figure 8.120). The first family of element combinations provides platforms and access for the apartments and starts with placing a Pavilion element (Figure 8.121). The Access element is used for both horizontal and vertical connectivity. Players can mine the blue material from the floor and place it in the form of stairs as they see fit (Figure 8.122). The second family of element combinations enables the laying out of the apartments and needs a Pavilion element as the base (Figure 8.123).

The game design incentivizes players to build apartments in a perimeter block typology, providing them with loggias as in the reference building. A price of an element is determined by the difference between the resource blocks a player needs to place to form the activation key and the resource blocks they get in return. All elements bring four blocks, but some require 2 or 3 to be created. The access element is the cheapest, incentivizing players to build more of it and making it attractive to create perimeter block building typologies. The element that is a must, the pavilion, is the most expensive. But players need to place it anyway to build more rewarding elements and complete the goal. The windows are also the most expensive facade element, loggia and door being cheaper to encourage the desired apartment typology - loggias and many rooms.

The team worked iteratively to create a non-architect-friendly grammar and game design that produced buildings similar to the original. They tested three iterations of the grammar in a total of 13 games with 2-3 players, each bringing

them closer to the desired outcome. The first phase proved complex for people to remember and follow the combinatorial sequences, so the team created a player guide and a key catalog closer to the building zone (Figure 8.124). The second phase focused on finding the suitable game duration and balancing the rewards system. The team tested 15, 20, and 30 minutes game length, settling on the last one as optimal (Figure 8.125). The third phase explored the combinatorial possibilities (Figure 8.126).

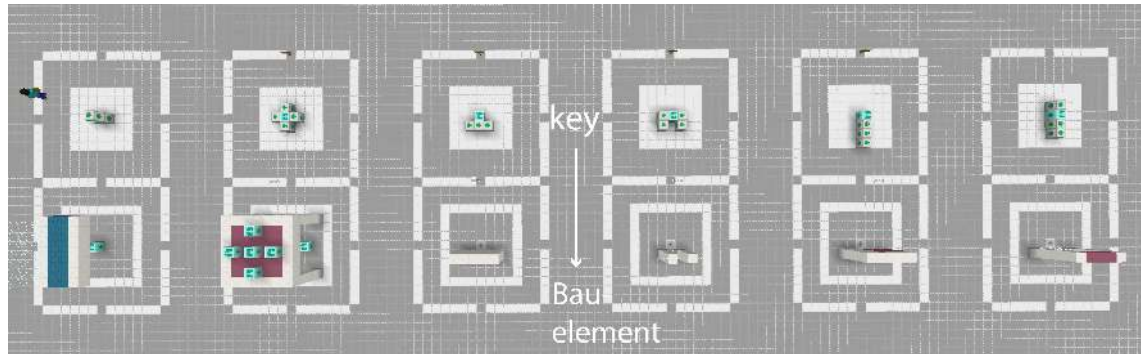


Figure 8.119 – Inakasa Grammar. A screenshot of the catalog area of the final version of the shape grammar in *Inakasa*. Image credits: Sajak, Markgraf.

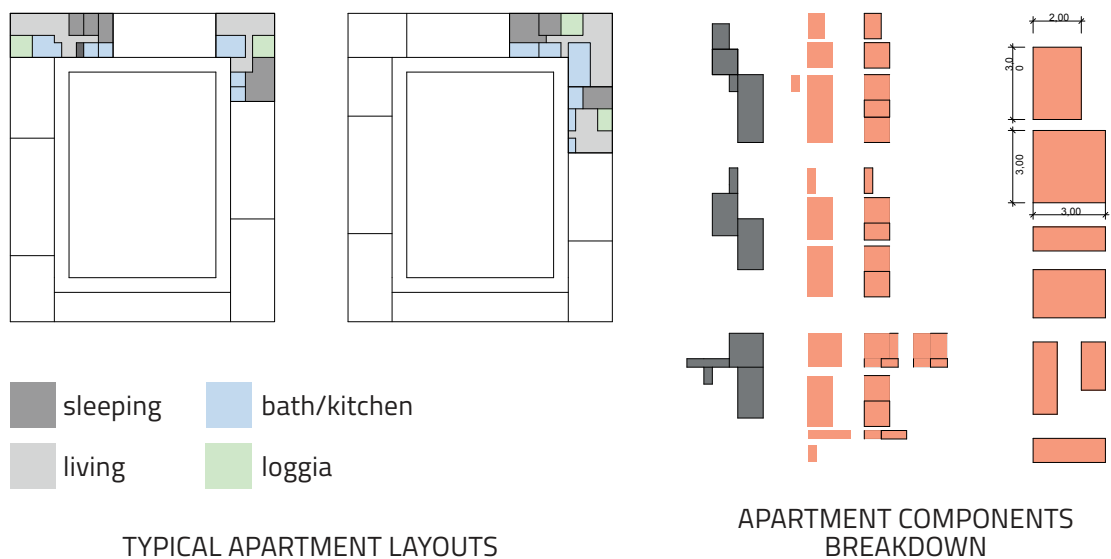


Figure 8.120 – The breakdown of apartment components in the Inakasa building. All apartments in the building have a loggia and this requirement is transferred in the game design as well. Image credits: Sajak, Markgraf.

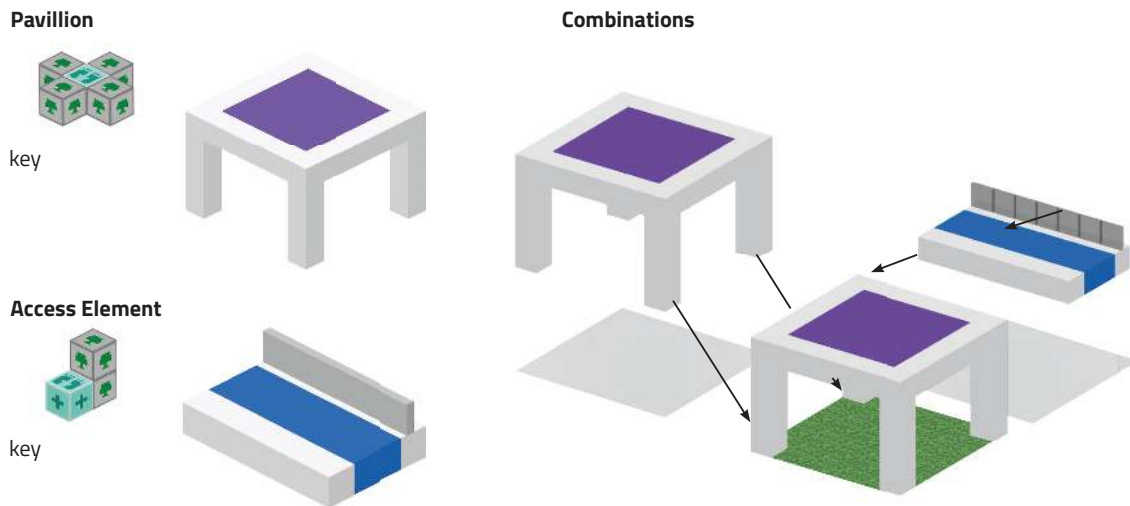


Figure 8.121 – Grammar combinations 1. The Pavillion and the Access element from the *Inakasa* grammar provide the base for creating apartments. Image credits: Sajak, Markgraf.

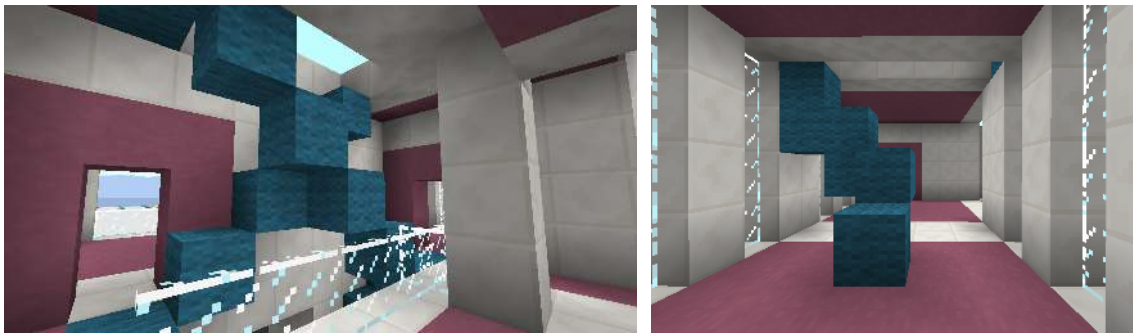


Figure 8.122 – Material sourcing for free building. The blue floor of the Access element can be mined by players in the game and subsequently placed to form stairs as needed. Image credits: Sajak, Markgraf.

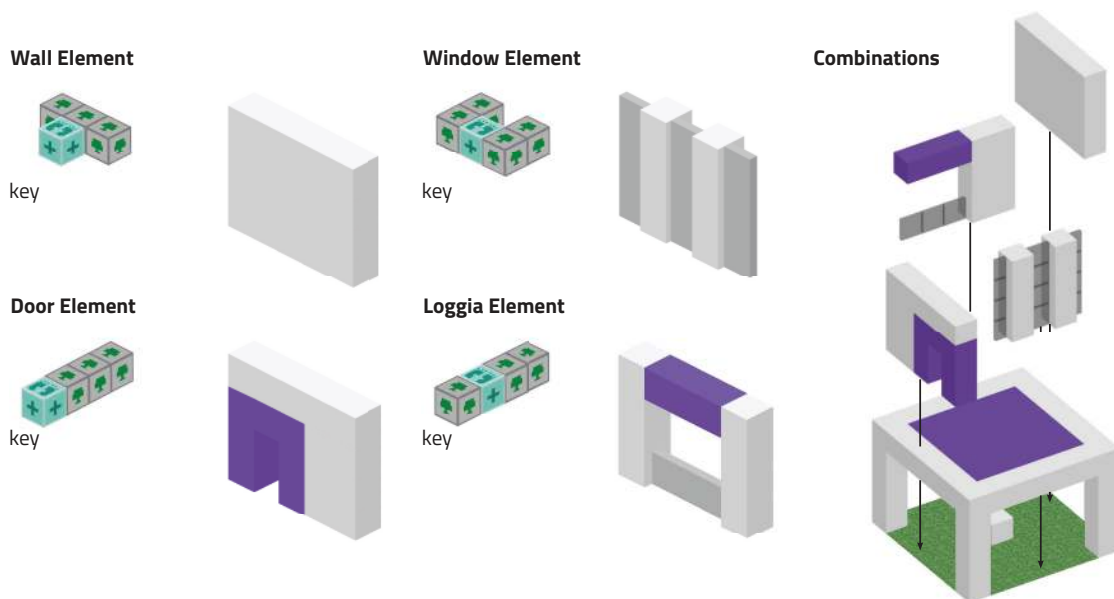


Figure 8.123 – Grammar combinations 2. The apartment components can be combined in various ways to create the linear and L-shaped apartments observed in the original building and provide them with loggias. Image credits: Sajak, Markgraf.



Figure 8.124 – Grammar iteration 1. The first phase of developing the grammar uncovered means to improve its comprehensibility for non-architects. Image credits: Sajak, Markgraf.



Figure 8.125 – Grammar iteration 2. The second phase of developing the grammar focused on balancing the game design to incentivize player to create perimeter block developments and loggias. The game duration of 30 minutes was also determined by playtesting. Image credits: Sajak, Markgraf.



Figure 8.126 – Grammar iteration 3. The combinatorial possibilities of the grammar are explored in the third, final phase of developing the grammar. Image credits: Sajak, Markgraf.

The Mountain

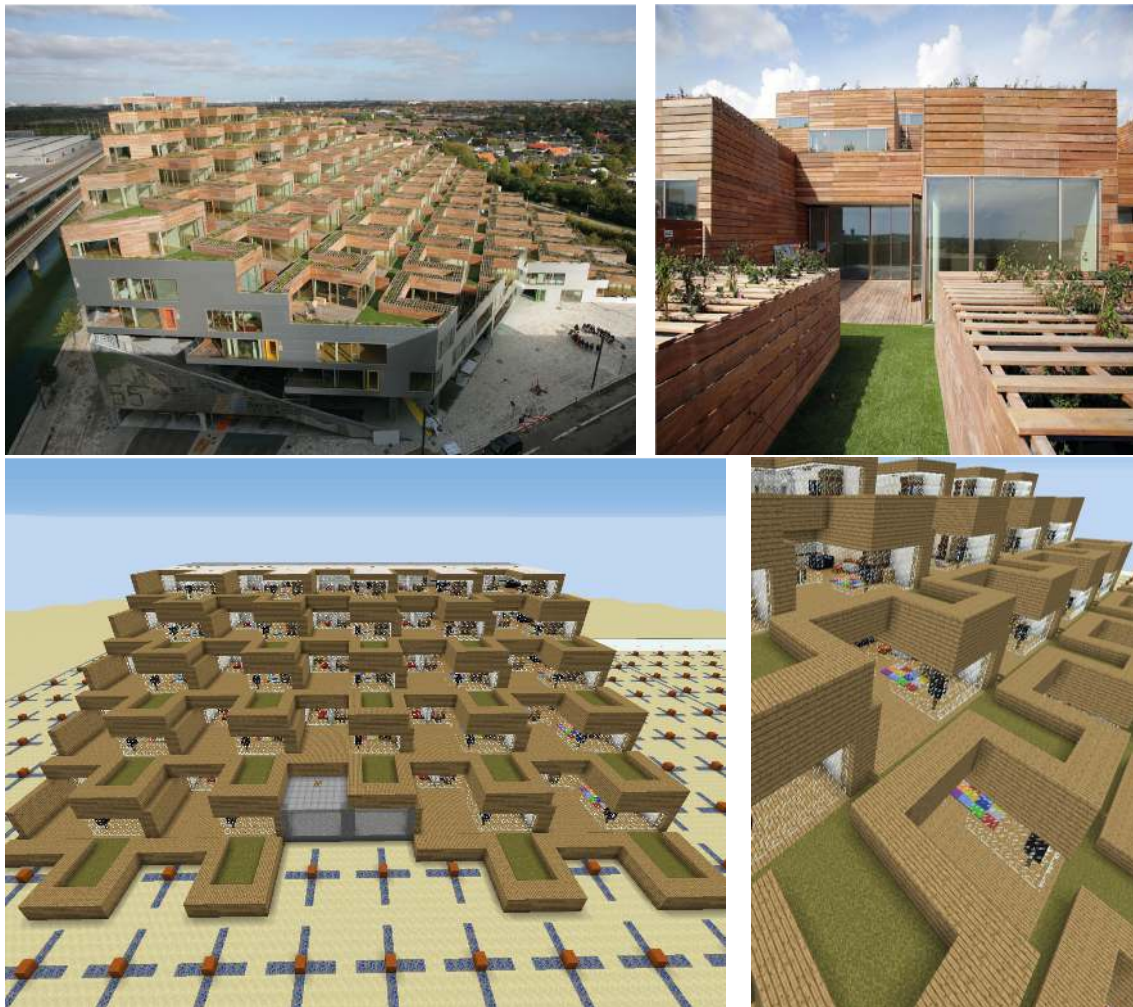


Figure 8.127 – The Mountain. The Mountain Dwelling (top row) by PLOT (later BIG) is the reference building for the *20.000 BLOCKS* project *The Mountain* (bottom row). Image credits: PLOT, Anjadini, Krzywik, Diduch.

The project *The Mountain* is developed by Shilla Anjadini, Timotheus Krzywik, and Tadeusz Diduch. The reference building is the Mountain Dwellings by PLOT, later restructured into Bjarke Ingels Group (BIG) from 2003. The building features 80 apartments arranged in a terrace-like manner to form a stepped slope (Figure 8.127). Underneath the apartment levels, there are parking levels.

The shape grammar is derived from the analysis of the building's floor plan of the residential levels and their vertical organization (Figure 8.128). The parking levels are not included in the grammar. The team broke down the building into four elements: Apartment, Garden, Stairs, and Corridor (Figure 8.129). The catalog in *20.000 BLOCKS* featured all four possible orientations of each element, opening up the combinatorial possibilities for design (Figure 8.130).

The goal of the game is to reach the highest points. Players start with 50 points and have 10 minutes to complete a game. Corridor and Stair elements cost 2 points (Figure 8.131). If connected via Corridors and Stairs to the street level, apartments bring three points. Otherwise, they give one point. Gardens bring the three points. One point is taken away if something is placed on top of them, and another point

is taken away if an element is placed in front of them (Figure 8.132). The three sample calculations in a sequence on Figure 8.133 show how denser buildings do not always result in a higher score.

The team advised players to spend their initial budget of 50 points on the access elements first and then dock the residential units and gardens (Figure 8.134). Four game results, created in these two phases, are shown on Figure 8.135 and Figure 8.136.

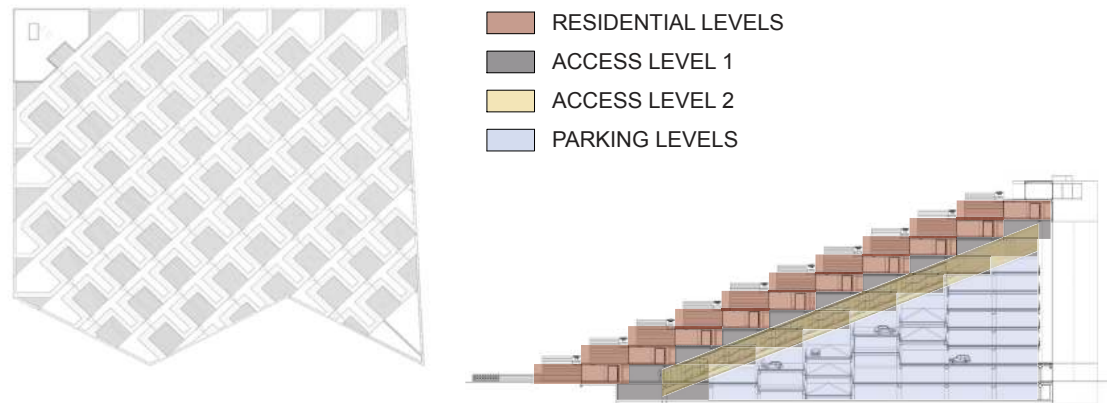


Figure 8.128 – Building analysis. Left: a top view of the Mountain Dwellings building. Right: a section of the building showing the terrace-like organization. Image credits: PLOT, Anjadini, Krzywik, Diduch.

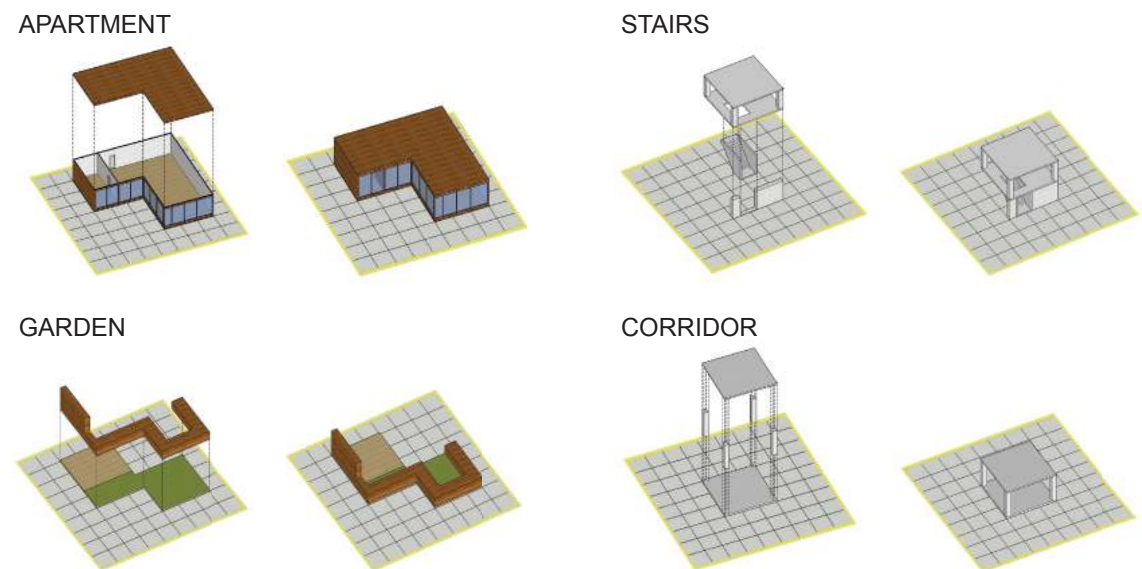


Figure 8.129 – Building elements. Image credits: Anjadini, Krzywik, Diduch.

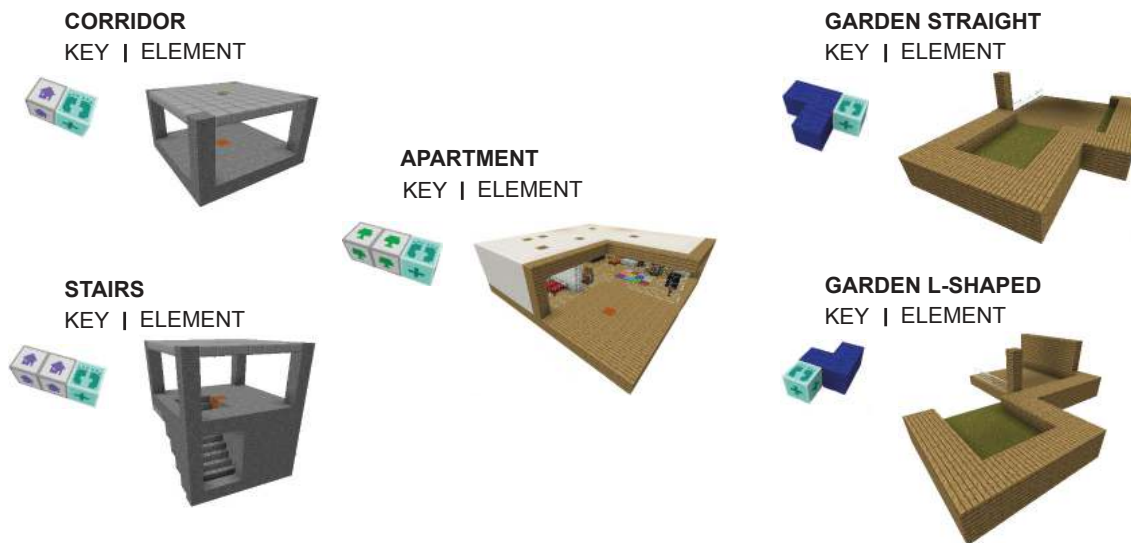


Figure 8.130 – The vocabulary catalog in The Mountain with its activation keys. Image credits: Anjadini, Krzywik, Diduch.





ELEMENT	POINTS PLACED	POINTS ONE CONDITION FULFILLED	POINTS ALL CONDITIONS FULFILLED
	- 2	-	-
	- 2	-	-
	+1	-	+2
	+1	+2	+3

Figure 8.131 – The reward system in The Mountain. It incentivizes high density by giving points for each added apartment. At the same time it rewards players for equipping the apartments with open-air, sun-lit gardens. Image credits: Anjadini, Krzywik, Diduch.

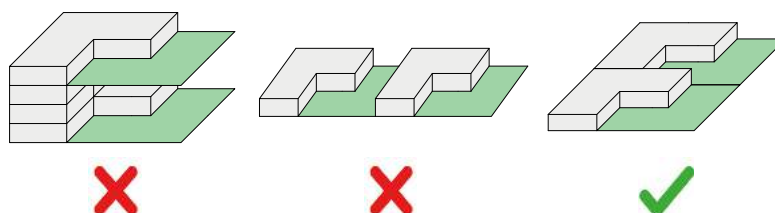
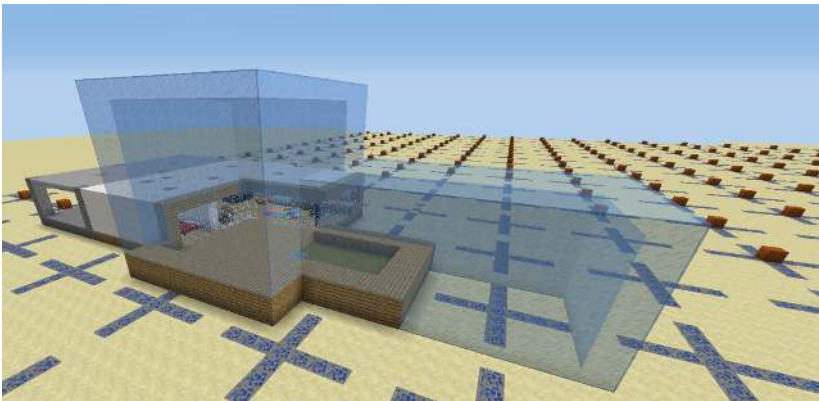
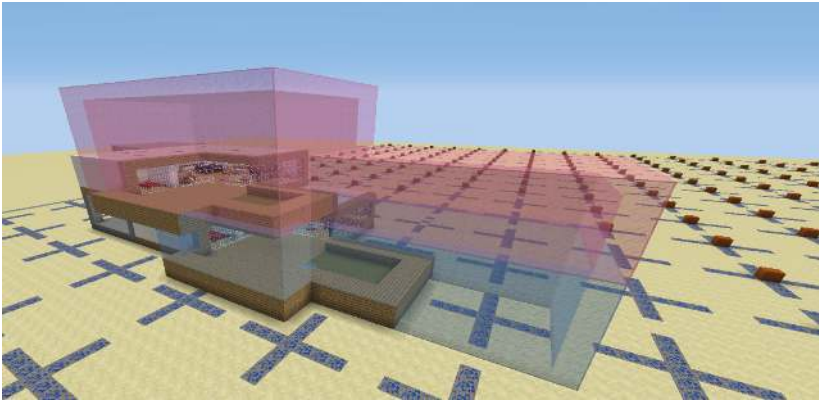


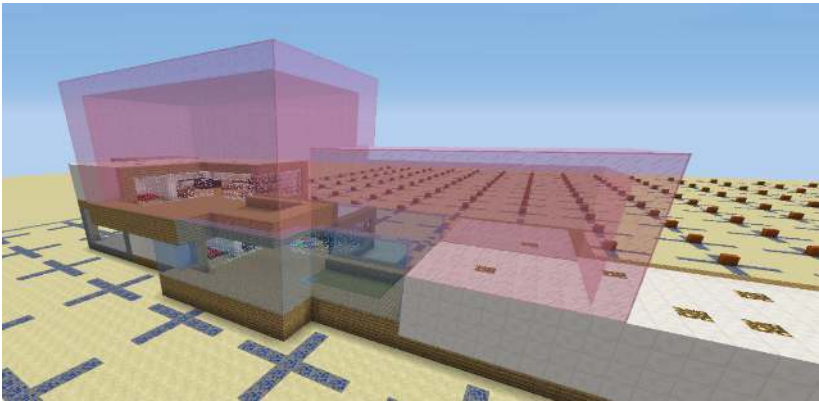
Figure 8.132 – Placement rules. If gardens are not blocked on top and on the front they bring the most points. Image credits: Anjadini, Krzywik, Diduch.



1x Apartment with access	+2
1x Garden free top & front	+3
TOTAL	5



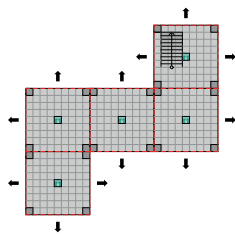
2x Apartments with access	+4
1x Garden (red) free top & front	+3
1x Garden (blue) free front	+2
TOTAL	9



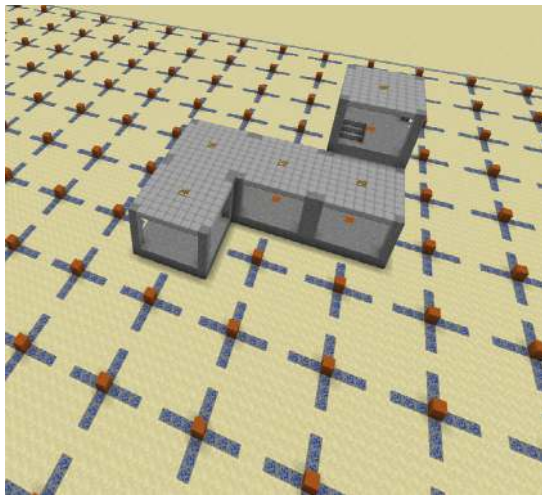
2x Apartments with access	+4
1x Apartment no access	+1
1x Garden (red) free top & front	+3
1x Garden (blue) blocked	+1
TOTAL	9

Figure 8.133 – A sample game sequence and the points it brings to the player. The second step gives the same points although the third one is denser, i.e. has more apartments. Image credits: Anjadini, Krzywik, Diduch.

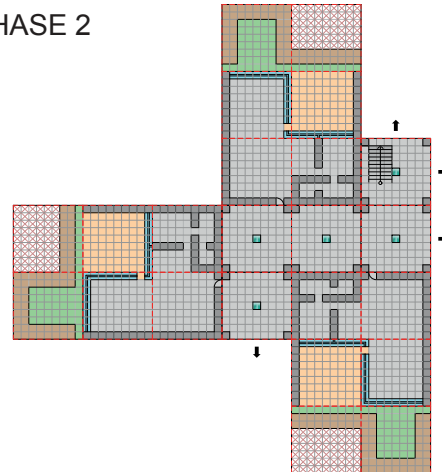
PHASE 1



3x CORRIDORS
1x STAIRS



PHASE 2



+3x APARTMENTS

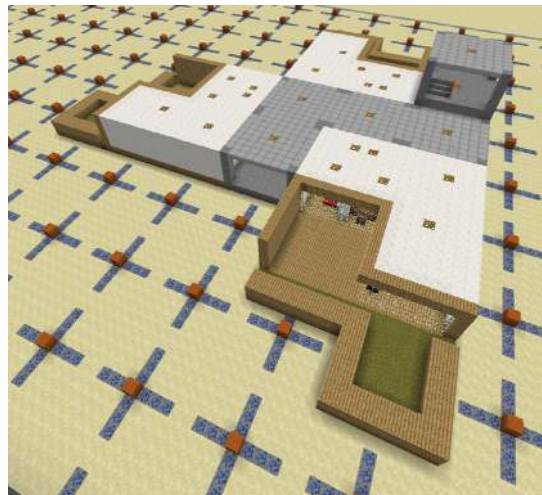
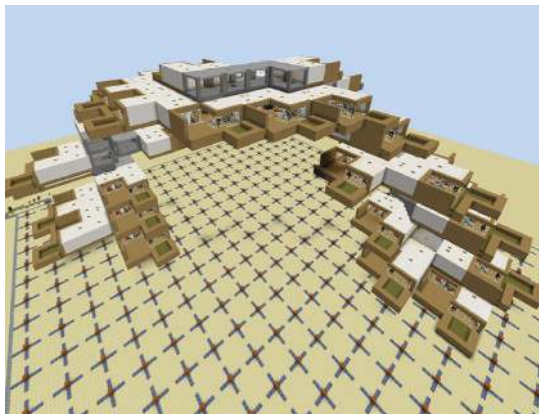
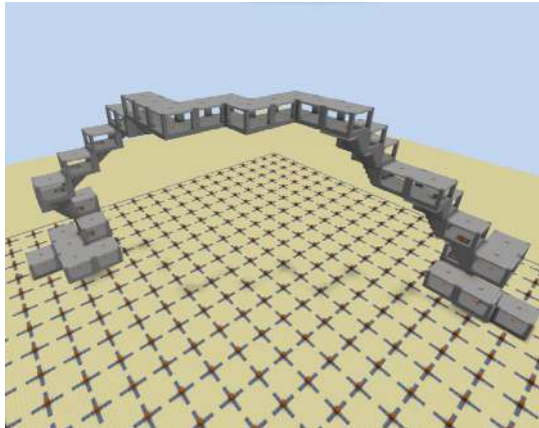


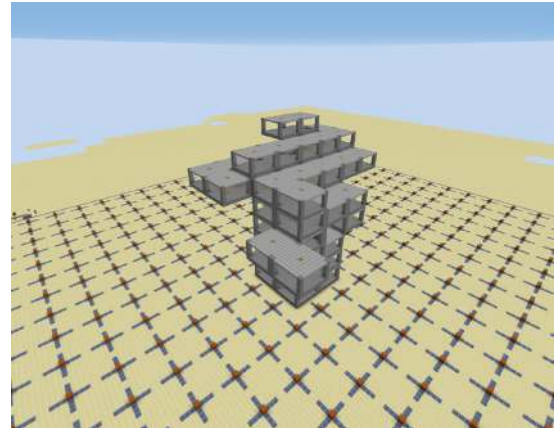
Figure 8.134 – The two game phases in The Mountain. Players build the elements ensuring access first and later add the apartments onto them. Image credits: Anjadini, Krzywik, Diduch.

RESULT 1



26x Corridors
14x Stairs
32x Apartments
32x Gardens

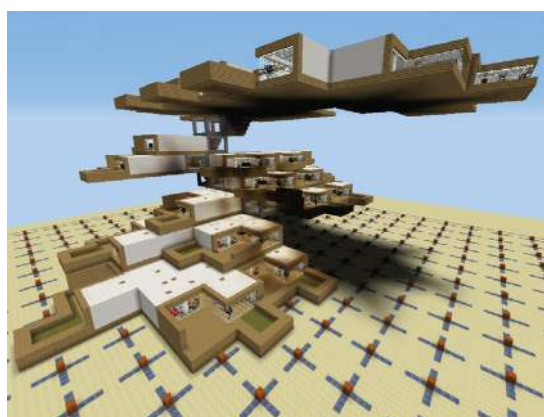
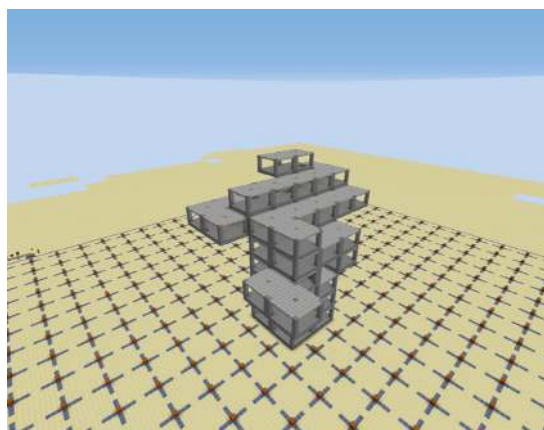
RESULT 2



42x Corridors
8x Stairs
23x Apartments
23x Gardens

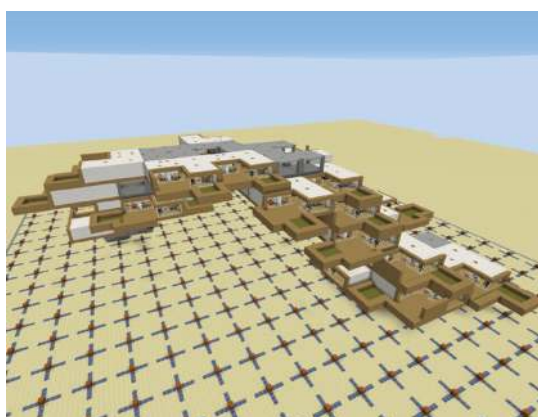
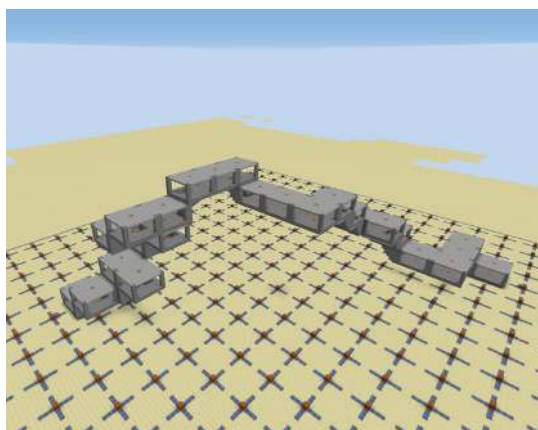
Figure 8.135 – Designs 1 nad 2 from **The Mountain** game. Image credits: Anjadini, Krzywik, Diduch.

RESULT 3



30x Corridors
9x Stairs
33x Apartments
33x Gardens

RESULT 4



20x Corridors
11x Stairs
26x Apartments
26x Gardens

Figure 8.136 – Designs 3 nad 4 from **The Mountain** game. Image credits: Anjadini, Krzywik, Diduch.

8.5 Take-aways

With the framework *20.000 BLOCKS*, architectural experts can encode their design knowledge into custom-developed multiplayer game design in Minecraft. Non-expert players are constrained and guided by these game mechanics, enabling them to create unique architectural results. The 3D environment of Minecraft immerses the players in the gameplay and increases motivation and engagement. The game mechanics are built around three components: guiding rules, verification routines, and fast feedback. The framework employs a real-time link between the game Minecraft and Grasshopper. The framework offers robotic fabrication to prove the viability of player-created designs, where the digital results are brought to reality at 1:100 scale.

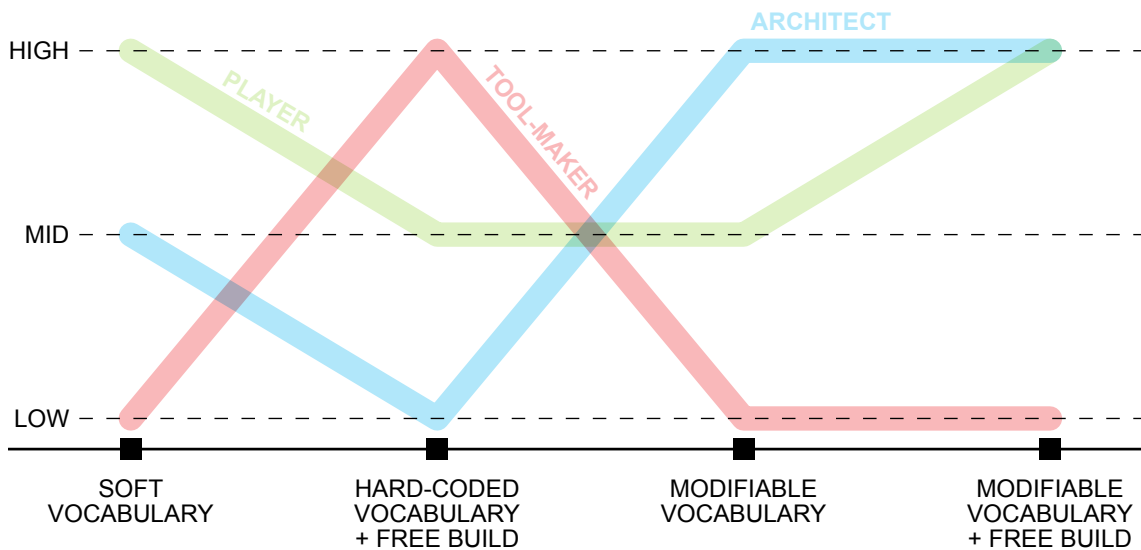


Figure 8.137 – Vocabulary types explored in 20.000 BLOCKS. The ability of the different roles to influence the final architectural design outcome depending on the type of vocabulary used. Image credits: the author.

The three vocabulary types — soft, hard-coded and modifiable — were put through online playtesting. The hard vocabulary was additionally tested at events at the *Digital Design Unit (DDU)* at TU Darmstadt with university students, as well as at *Invent the World (ITW)* with 7–10-year-old kids. The modifiable hard vocabulary was tested in *Play-Design-Build* and *Combinatorial design*. The two projects also tested how other architects, acting as Experts, could use the modifiable vocabulary to define the use-case and the design expression for the projects.

The method proposed with *20.000 BLOCKS* relies on players using the Minecraft map to model grammars and generate structures. Currently the *20.000 BLOCKS* modpack has more than 2000 plays and 330 downloads². In various projects and prototypes, around 20 different maps were created. Each with a different vocabulary catalog and game rules defined a *playable voxel-shape grammar*.

Possible applications I see for the method of game-to-CAD-to-Robot transfer of geometry are:

- In the research field, new forms of architecture could be explored that transgress

²<https://www.technicpack.net/modpack/20000-blocks.861360>, accessed 2.10.2020

established typologies. This is helped by engaging the unbiased minds of the non-experts.

- In the practice, a specific design task could be crowdsourced, tapping into the decision-making power of inhabitants, neighbors, and investors by defining the corresponding in-game rules and providing suitable background evaluation algorithms.

8.5.1 Features of the voxel-shapes

Design The abstracted modeling environment with a voxel size of 1 x 1 x 1 m could be limiting. However, since the aim is to generate schematic architectural designs, this level of abstractness works in our favor.

Walkability An essential requirement to achieve an immersive experience in the generative phase, i.e., in the play phase, is that the resulting structures be walkable for players to reach the newly added markers. Otherwise, the iteration process is blocked. That means that in both urban or building scales, all possible sequences of the grammar's shape rules need to create a walkable combination when placed.

Furthermore, the size of a player's Minecraft avatar in relation to the voxel shapes they model — and subsequently iterate through in the game world — creates an immersive perception for a one-to-one architectural scale.

Color Coding The possible further grammar growth points need to be very clearly visible to the player while in play. Therefore, the textures of the marker voxels and the building blocks need to be different. Furthermore, with larger sets of rules, such as in *IBA_GAME*, where we used six types of rules with 2–4 variations each, a color coding of the kinds of rules helps the player learn the grammar faster.

The ideal scenario to test the method in practice is the massing out of an architectural concept, such as defining the rough placement and orientation of rooms in a building or determining the location and infrastructure of buildings in an urban design scheme.

Key findings:

1. Harder rules are better than soft ones in delivering a feasible architectural design.
2. If the rules become too hard, the players no longer feel part of the design and, as such, are unmotivated to play.
3. If the rules are too soft, the resulting structures become too chaotic. This makes them difficult to verify and construct. In addition, players find those designs confusing to navigate and play through.
4. Robotic fabrication is possible but currently too slow to provide meaningful feedback.
5. An easily modifiable vocabulary allows other architects to participate as experts, opening our method to more possible applications.

6. Clear, non-architectural, time-bound goals make participation more accessible and entertaining than tasks where players need to understand the spatial and architectural qualities of the game elements they are building.

8.5.2 Takeaway 1: Gradient of control

The case study *20.000 BLOCKS* reveals that calibrating to what extent do Experts and Players have control over the design outcomes of the games is an ongoing challenge and primary focus of this research.

As Brabham 2013 states, for a well-functioning crowdsourcing model, the locus of control regarding the creative production of goods must exist between the organization and the crowd. If the locus of control is closer or more prominent in the community, such as in the case of open-source software or Wikipedia, or if the power is mainly in the organization's hands, such as when a company wants the community to merely vote for the color of a product, we are not seeing an actual crowdsourcing model (Brabham 2013).

Therefore, I consider the balance of control that players and experts had when analyzing the results. I imagine it on a gradient scale of soft to hard, where the soft end of the spectrum gives more control to *Players* and the hard end of the spectrum, more control to the *Experts* (Figure 8.138). Finding the right balance between hard and soft requires constant testing. We can calibrate the balance better with every project iteration and every game played.

The type of vocabulary — soft, hard-coded, or modifiable — influences the degree to which different roles can influence the outcome as well and affects the balance between them (Figure 8.137).

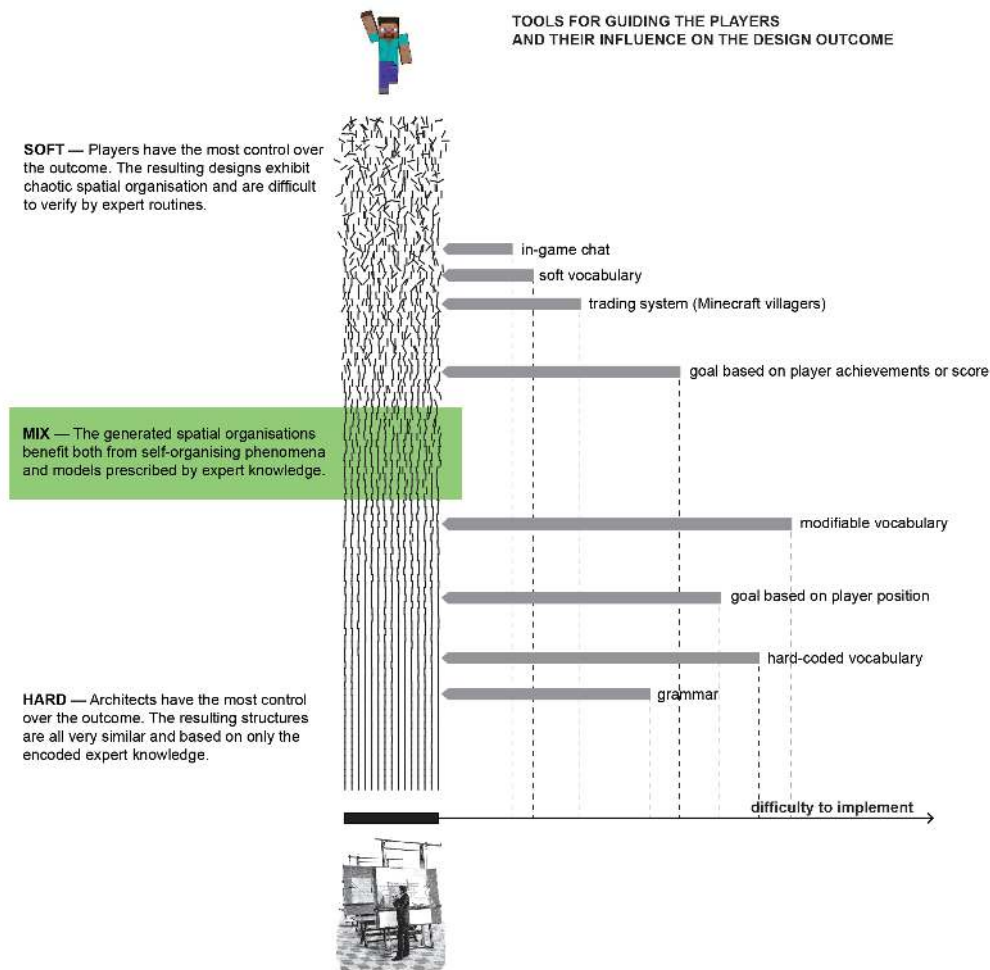


Figure 8.138 – Gradient from soft to hard. Each tool empowers players or transfers an expert's will differently. It also comes with a certain cost in effort to implement. Image credits: the author.

8.5.3 Takeaway 2: Robot process slow

From all 57 game results, we selected, post-processed, and robotically manufactured five models. The robotic process is too slow to keep up with the digital iterations. The speed is 200 blocks per hour, and an average design requires 1200 to 2000 placed blocks to build, i.e., 6–10 hours per model. This includes both the model blocks as well as the supporting blocks. As the research focuses on game design as a guiding instrument, the models' analysis is secondary.

8.5.4 Takeaway 3: Need for automated post-processing

The ability to involve a team of experts in the making and testing of the architectural vocabulary holds the potential for feasible and well-performing solutions. To make this viable, the architectural field's main hurdle is developing post-processing routines that can quickly turn player-generated designs into CAD models for further specification.

8.5.5 Takeaway 4: Design system

We also developed an in-house design style guide to regulate, across our team, how and when sections of a shape rule could overlap with another (Figure 8.139). This saved the work of having to try all combinations separately.

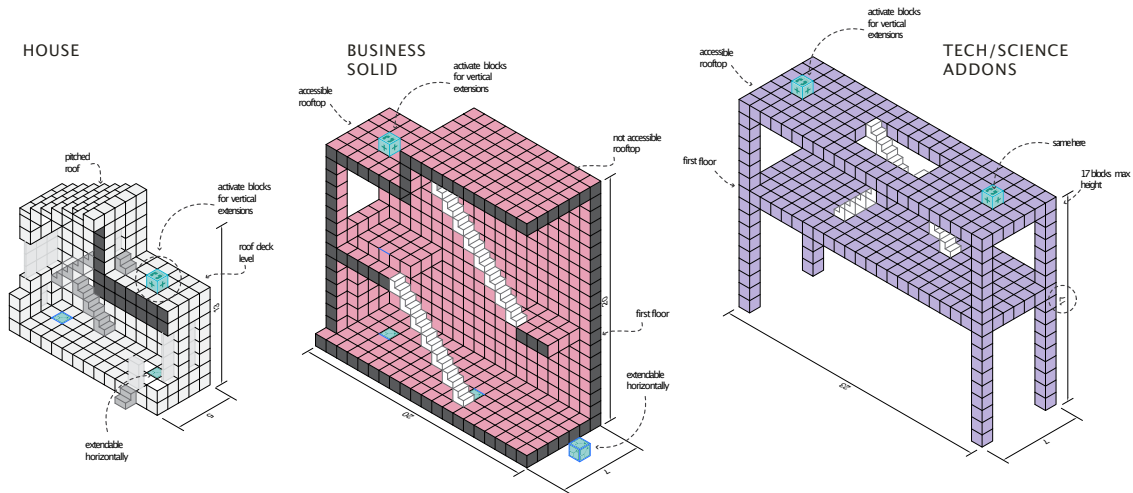


Figure 8.139 – Design style guide. An extract from the design style guide for the urban scale grammar of *IBA_GAME*, which was developed and shared within our team. Image credits: Marios Messios.

8.5.6 Future work

The research could benefit from the following future developments:

- A way to implement the grammar logic entirely on the server-side so that players can participate with a Vanilla (unmodified) Minecraft client. The need to download and use a customized Minecraft version reduces the number of potential players.
- A generative visualizer, which could simulate a game being played so that a designer can capture some of the most common conflict/bugs in advance, like mismatching shapes and broken walkability, without having to play the game over and over again.
- The collected player-created designs can be made available through a graphical search interface, similar to the one described in *chapter 7: Sensitive Assembly*.
- The collected data (grammar definitions plus the generated shapes) can be used to train a machine-learning algorithm to generate either grammar sets or schematic designs.

Chapter 9

Project Reptiles

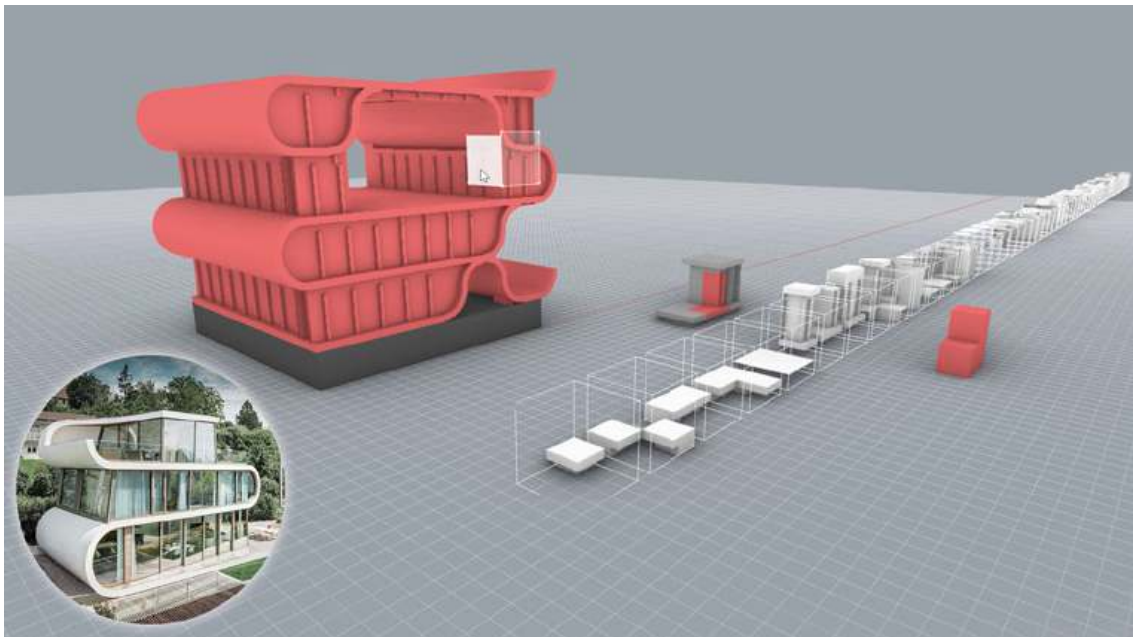


Figure 9.1 – A tileset from Project Reptiles in use. Image credits: Roger Winkler.

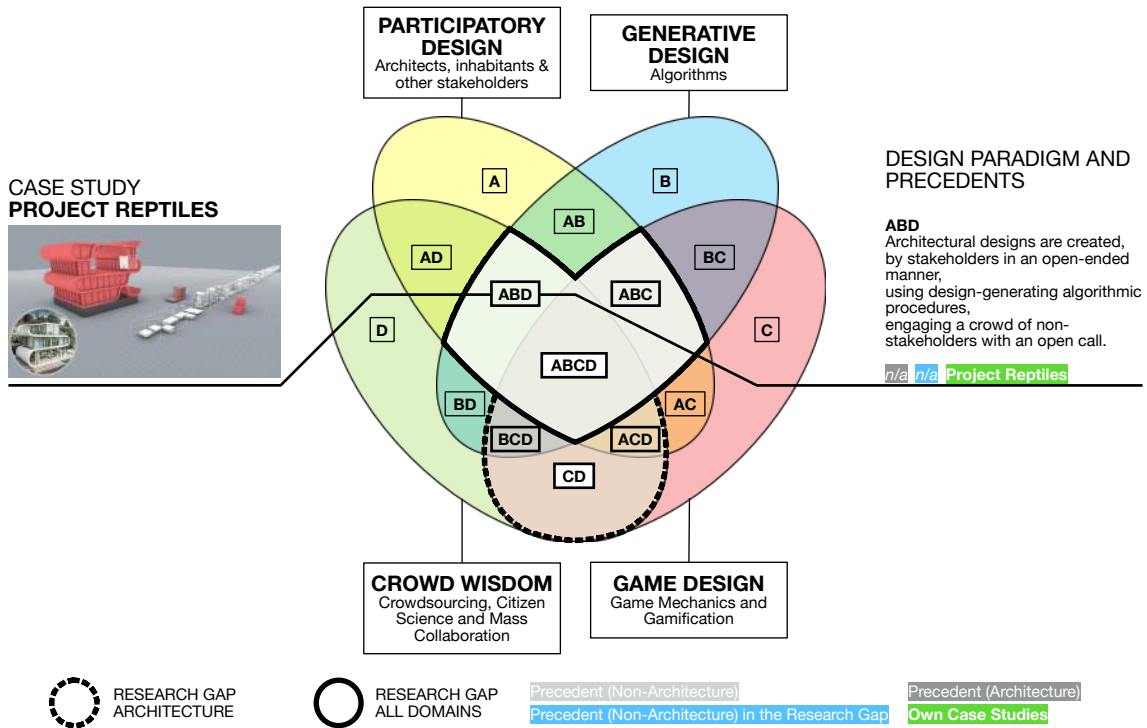


Figure 9.2 – Project Reptiles on the map of design paradigms. Image credits: the author.

9.1 Design Paradigm

Project Reptiles is a case study comprising of a custom-made tile-based building configurator (Figure 9.1) tested in a series of workshops with architectural students and online, non-architect participants. The digital tools at the core of the case study were implemented by Roger Winkler under my guidance throughout a series of student courses, culminating in his master thesis at the Digital Design Unit of the Technical University of Darmstadt. Semi-structured interviews with architectural offices and mock design competition, part of Roger Winkler’s master thesis, complete the case study.

Project Reptiles uses an iso-surfacing, tile-based generative algorithm to create designs. The project involves the architects in the role of stakeholders in creating both tilesets and designs. It also engages the crowd in creating designs. No game design or game mechanics are employed in the project. As such, *Project Reptiles* falls under the ABD design paradigm (Figure 9.2) where, *architectural designs are created by stakeholders in an open-ended manner, using design-generating algorithmic procedures, engaging a crowd of non-stakeholders with an open call.*

With *Project Reptiles*, I explore how the barrier to participation of both experts and non-experts can be lowered through algorithmic assistance. For non-experts, the focus is on enabling the visual expression of their ideas and preferences. The hypothesis is that a generative system proposing near-complete designs will be beneficial. At the same time, for experts, i.e., architects, the focus is on maintaining design freedom when using such a generative system without requiring additional computational design skills or knowledge in programming.

Csikszentmihalyi’s idea of Flow — having the right match of skills and difficulty — presents a framework to visualize the means of lowering these barriers (Csik-

szenzmihalyi 2008). According to Csikszentmihalyi, a high level of skills and low difficulty would trigger boredom (Figure 9.3). On the opposite side, a low level of skills and high difficulty would trigger anxiety.

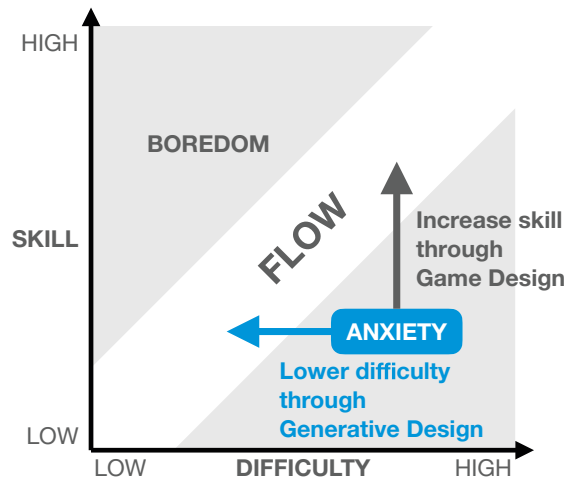


Figure 9.3 – Flow through assistive algorithms. *Project Reptiles* explores how assistive algorithms can lower the difficulty in producing designs for non-experts and in keeping design freedom when working with generative systems for experts. Image credits: the author.

While *Sensitive Assembly* and *20.000 BLOCKS* explored the educational benefits of game design in increasing the participants’ skill levels, the aim of *Project Reptiles* is to explore the degree to which assistive algorithms can lower the difficulty for both experts and non-experts without requiring a change in skill.

The *Project Reptiles* case study builds the base to:

- Introduce *assisted sculpting*, a strategy for encoding design content into participatory computational design systems that uses iso-surfacing and constraint-solving.
- Describe the technical implementation of an *assisted sculpting* digital environment with potential applications in creating massing models and schematic designs.
- Reflect on how architects can use thousands of crowdsourced designs created by non-experts.

9.2 Implementation and Setup

The computational design system is implemented in Grasshopper. It is an interactive voxel field where users can click to activate or deactivate a voxel. It is inspired by the game *Brick Block*, which is the source of the initial tileset in this case study (Stalberg 2016). The initial iso-surfacing ruleset distinguishes solely between spaces enclosed by or outside the modeled object. The tileset that implements this simple rule is topologically identical to the tileset used in the Marching Cubes (MC) algorithm, which has 256 tiles. When rotated, mirrored, and inverted variations are excluded, this number goes down to 15 unique tiles. In architecture, however, up-facing and down-facing surfaces are different, i.e., the lower surface of an overhanging room

is not the same as the upper surface of a flat roof. Therefore, the tileset can be reduced to no less than 53 unique tiles, 15 of which can be considered as the basic shapes that define a design family (Figure 9.4).

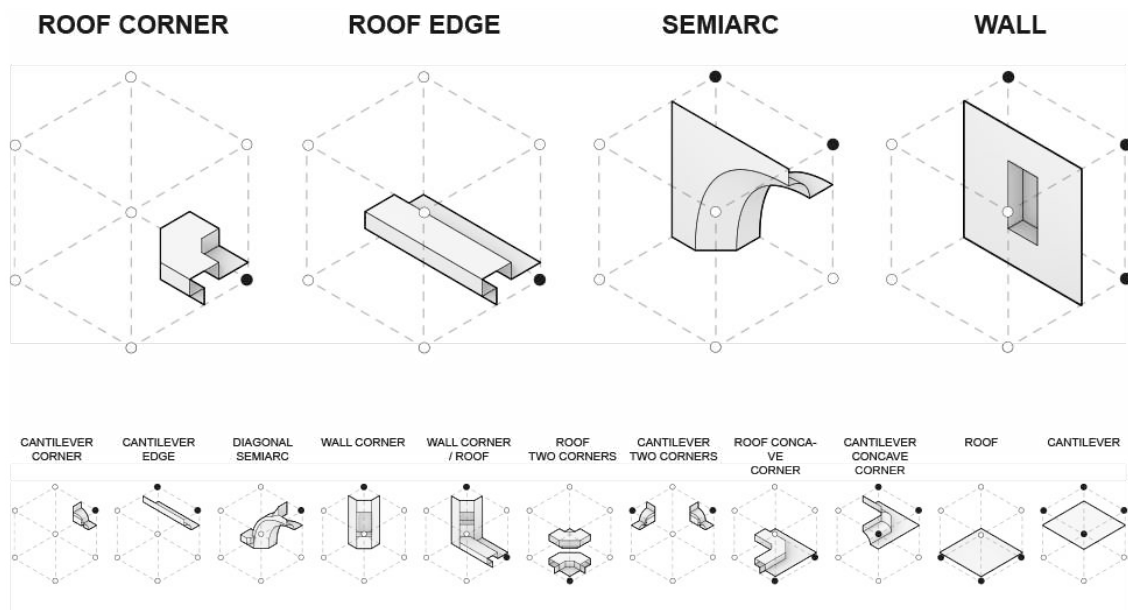


Figure 9.4 – Brick Block tileset. The 15 basic shapes in the Brick Block tileset. Image credits: Roger Winkler.

The implementation includes an interactive sculpting mode and a tileset editing environment for Rhino, allowing an architect to represent a new design family by modeling a new tileset and instantly sculpting something with it (Figure 9.5).

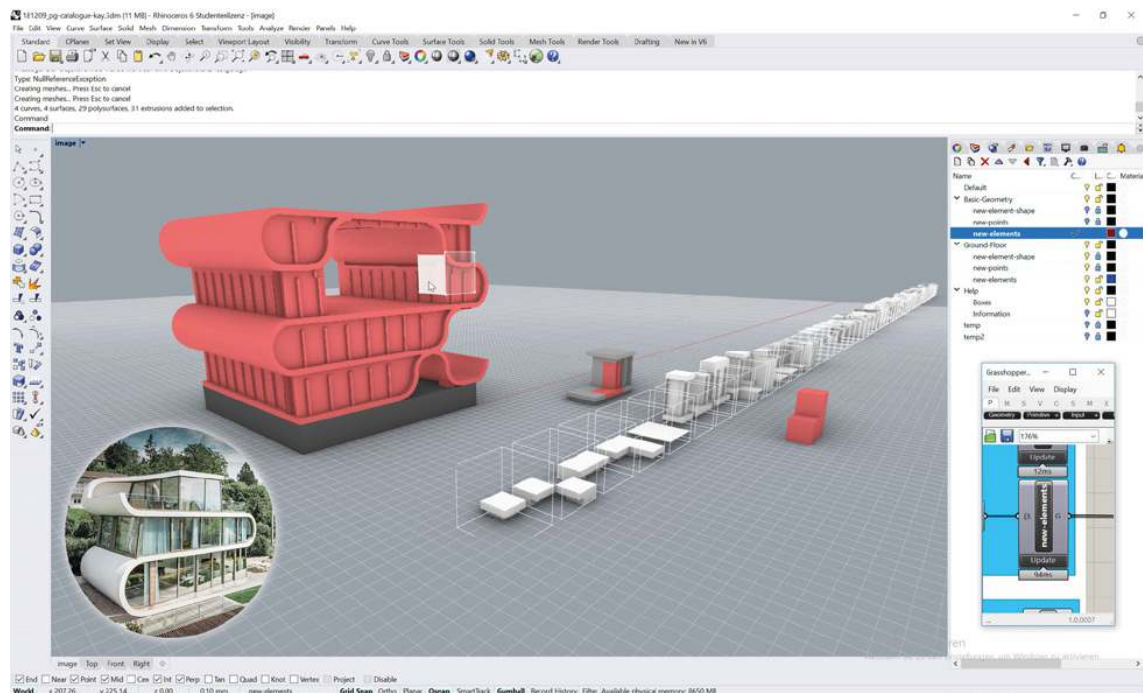


Figure 9.5 – Example tileset modeled by an architecture student. Assisted sculpting environment and the tileset editor in Rhino/Grasshopper. The screenshot shows a design variation using a tileset inspired by the *Flex House* by *Evolution Design*. Image credits: Roger Winkler.

Sculpting mode

Blocks are placed and deleted by pressing the left or right mouse button at the corresponding position on the sculpture. The system calculates the ratio of areas accessible from the ground floor for an imaginary person walking in the designed building. The blue button activates the preview of accessible and inaccessible areas (Figure 9.6).

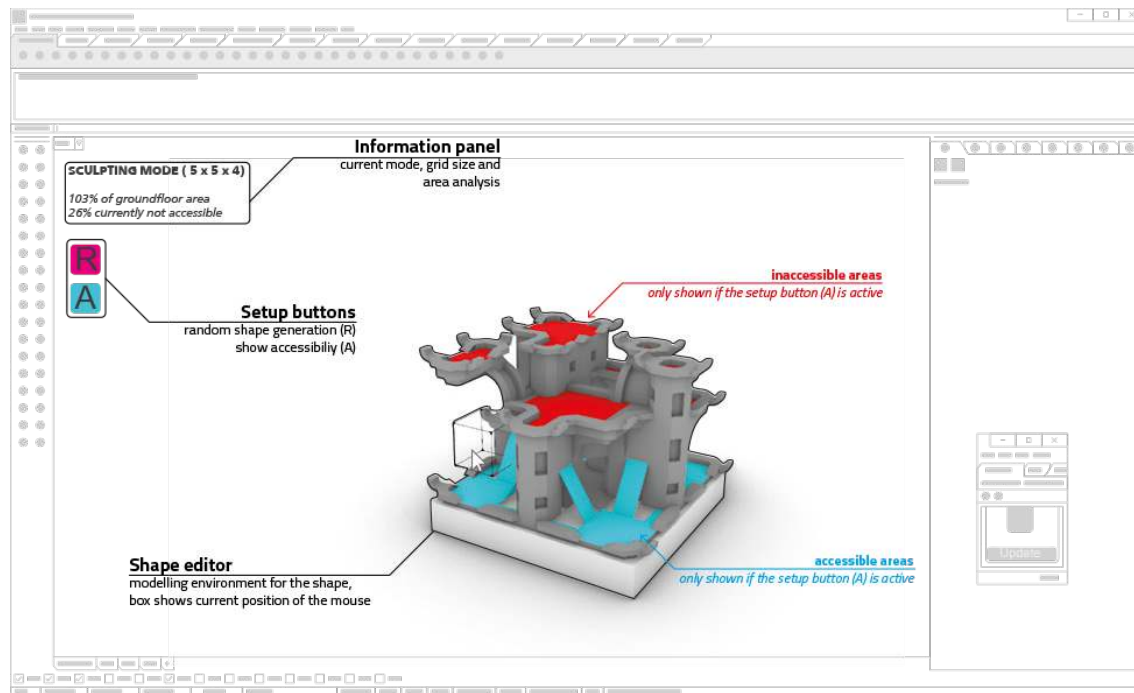


Figure 9.6 – User interface in the sculpting mode. Image credits: Roger Winkler.

By pressing *CTRL+D*, the user can switch to a second mode, wherein the tile that is currently being pointed at is highlighted. The corresponding catalog index is shown in the information panel, which helps transition to tile editing mode. By activating the pink button (R), a random shape is generated that can also be edited so that users do not have to start from scratch.

Tileset editor

The tileset editing mode supports the expert while designing a tileset that represents a design idea they have in mind (Figure 9.7). The following features are implemented:

- navigate through the tileset (arrows, left/right)
- turn on/off the display of the reference surrounding tiles (control + D) — this helps the designer match a tile's side edges with the tiles that could go next to it.
- switch to the sculpting mode (tab)

To update the edited tiles for the sculpting mode, the user must click on the Update button in Grasshopper. The info panel in the top left shows him the state

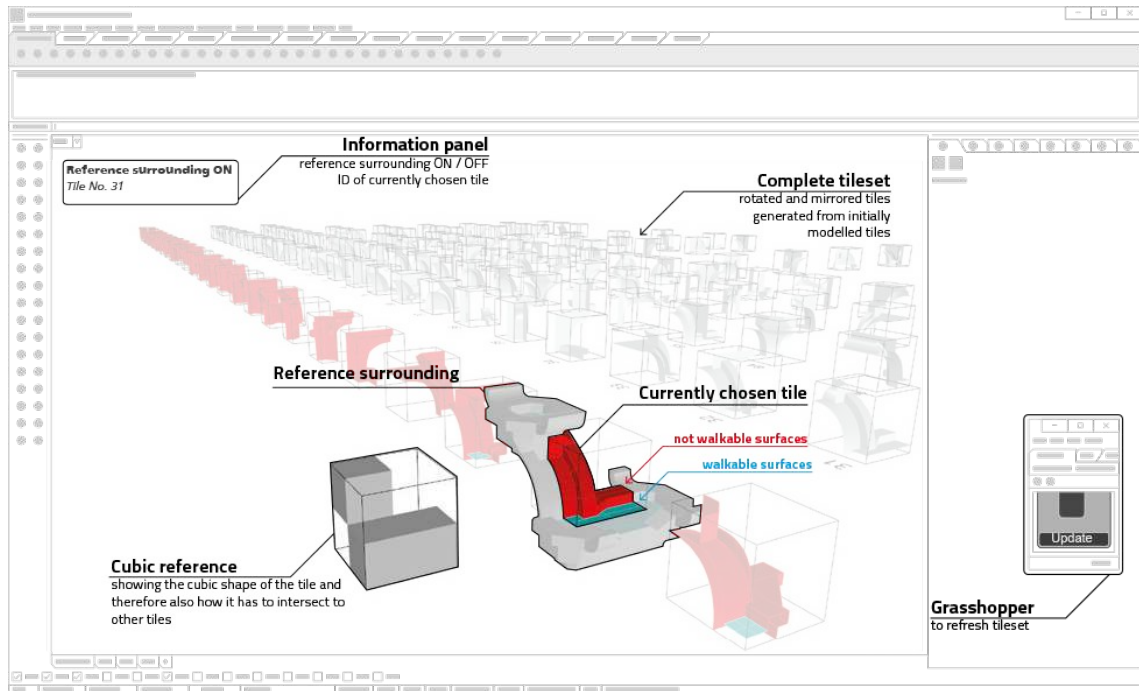


Figure 9.7 – User interface in the tileset editing mode. Image credits: Roger Winkler.

of the surroundings and the current tile number. For the system to calculate the accessibility and the current floor area in the sculpting mode, the user has to model on two different layers — a 3D model, shown in red, and a walkable surface, shown in blue. The editor automatically generates the rotated and mirrored variations of the tiles. The architect controls the order in which tiles are rotated and mirrored, adds constraints for each unique tile for a specific design case (i.e., different window on the north facade only), and defines locations for specific tiles (i.e., separate tileset for ground floor).

Site-specific online configurator

After implementing the sculpting mode of *Project Reptiles* in Rhino as described above, Roger Winkler implemented it in the Unity game engine as well (Figure 9.8). This allowed the creation of a WebGL-based online building configurator for a specific site in Frankfurt (Figure 9.9). The user can show or hide the urban context with the *Environment* button in the configurator. This is helpful, for example, if a neighboring building hides a part of the design during the modeling process.

The maximum building volume has the size of 24 x 11 x 5 blocks for a total of 1375. A block is a cube with a side of 4 meters. Users can add blocks with the left mouse button and remove them with the right one.

To make the design process as fast as possible for the user, two different modeling tools were developed and implemented: Voxel Brush Size and Presets (Figure 9.10). The Voxel Brush Size users can increase or decrease the number of blocks edited with a single mouse click. With the help of the presets, users can either fill the site with blocks entirely and subtract what is needed, place a block-perimeter development and again subtract or add as required, or delete the current designs and start from scratch.

Quantitative information about the current configuration created by the user

supports them in creating design proposals.

First, if a block's distance to the facade is more than two blocks in all four cardinal directions, it is colored in red (Figure 9.11 left). This indicates that the block will not receive sufficient daylight. The user can activate a transparent mode to reveal internal red blocks hidden by other building parts.

The second feedback function calculates the number of inhabitants so that the user can develop a feeling for the scale of the design. Since residential buildings of different typologies have different occupant densities, the number of occupants is determined using a database of actual buildings as a reference. The database of buildings is represented as a three-dimensional space where the axes are the area of the site, the building's ground area, and the total floor area. Any design created in the online configurator can be positioned in the space, and the three closest reference projects are determined in real-time (Figure 9.11 right). The average occupant density of these projects, together with the floor area of the user's design, is enough to estimate the potential number of occupants.

The online configurator records each design state linked to a session id. In addition, the users are given the option to submit a design they have created, adding a short comment with the submission. All user submissions and intermediate states of the model are recorded as a binary string for later reconstruction and search (Figure 9.12).



Figure 9.8 – Online configurator. The WebGL-based site-specific online configurator in *Project Reptiles* developed in Unity. Image credits: Roger Winkler.

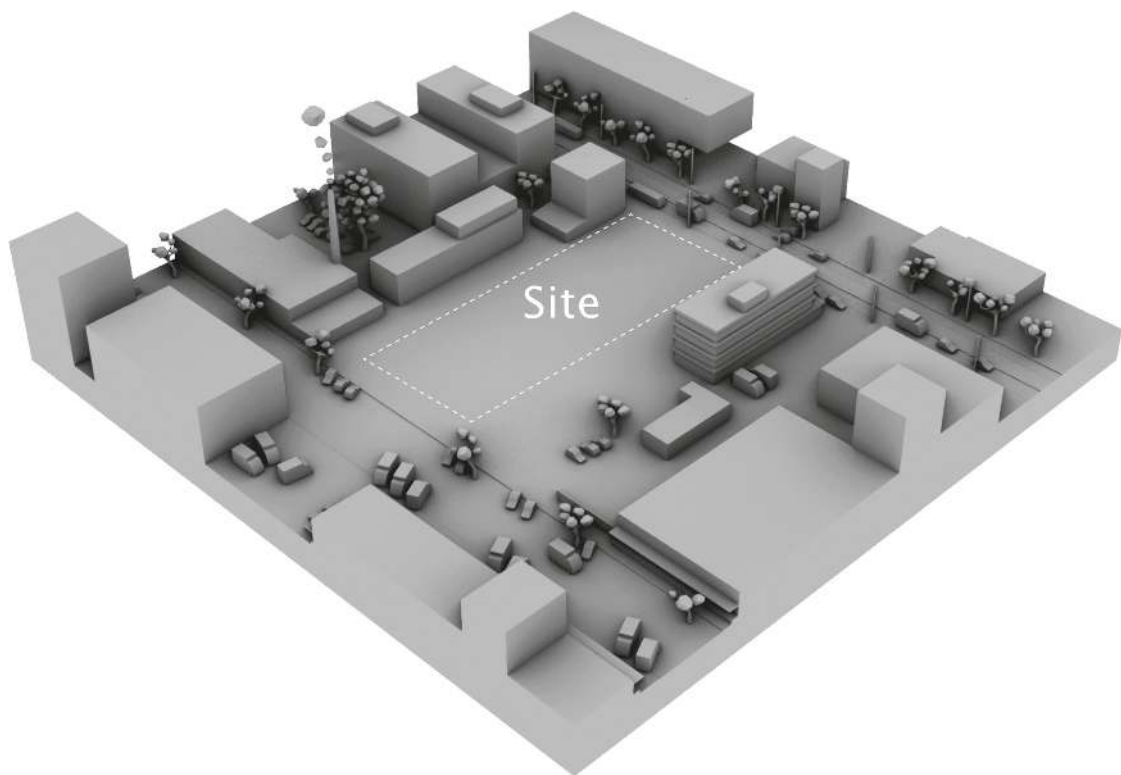


Figure 9.9 – Site 3D model. An actual site was used to crowdsource designs for the second phase of *Project Reptiles*. The site is on Hanauer Landstrasse 200 in Frankfurt am Main. Image credits: Roger Winkler.

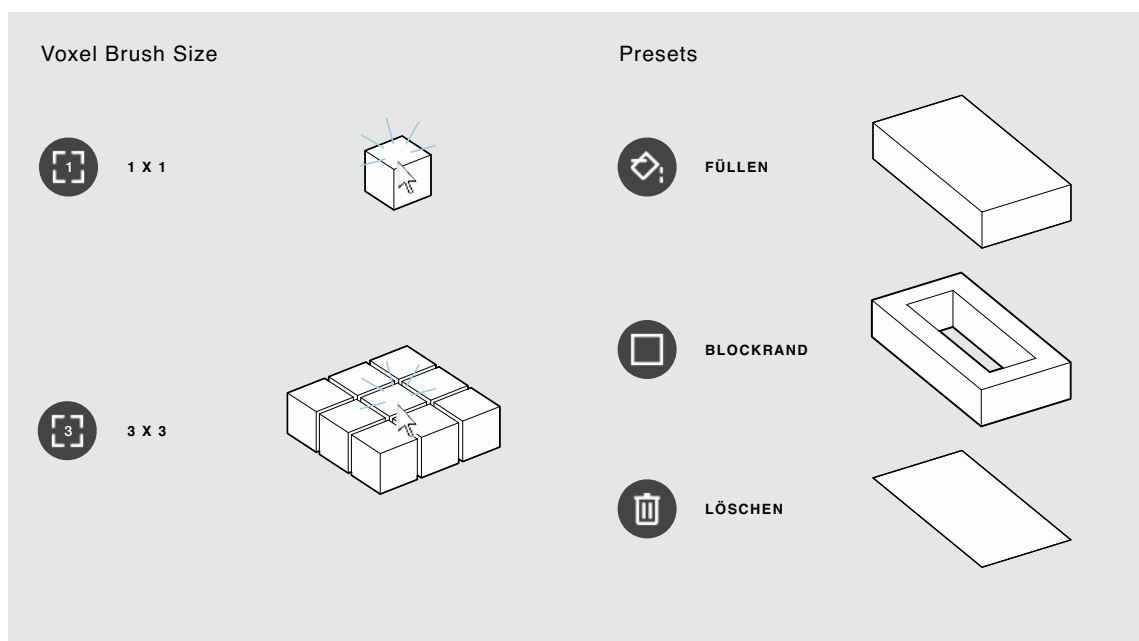


Figure 9.10 – Tools available to the user in Project Reptiles. Image credits: Roger Winkler.

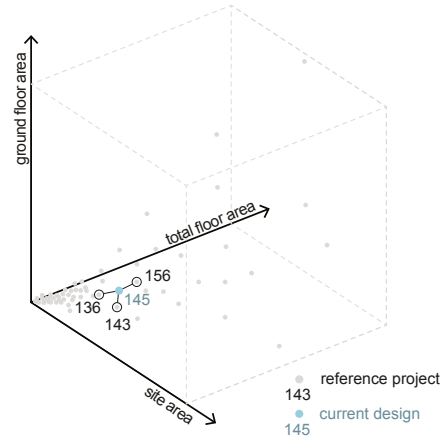
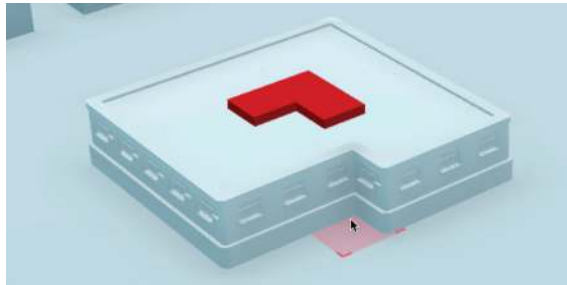


Figure 9.11 – Design metrics. Left: the configurator marks in red the blocks that do not get sufficient daylight. Right: the estimate for number of inhabitants is based on a database with actual reference projects. Image credits: Roger Winkler.

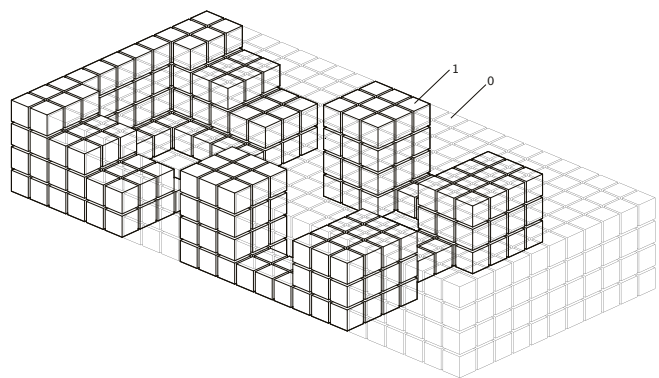


Figure 9.12 – Data representation. Left: The recorded binary string for a design with the optional user-provided text comment. Right: reconstruction of the same design in 3D. Image credits: Roger Winkler.

Data filtering tools

Two data browsing modes helped explore the potential of the collected crowdsourced designs to assist an architect in proposing a schematic design for a given site.

The first one is search by parameters. It is similar to the filter function when searching for a flight on travel portals (Figure 9.13) or optioneering interfaces for browsing of computer-generated massing models in computational design software (Figure 9.14). The user can narrow down the crowdsourced designs by their total floor area and then sift through them one by one to find a fit (Figure 9.15).

The second mode is *Autocomplete* and is more powerful in its assistive potential. Similar to autocompletion in search engines (Figure 9.16), the user can give a schematic massing as a 3D search phrase, and the algorithm sorts the crowdsourced designs by similarity Figure 9.17. The formula for calculating the distance between two designs is shown on Figure 9.18. It operates on the assumption that a design having the same amount of blocks but placed elsewhere would be more desirable than a design missing these blocks entirely.

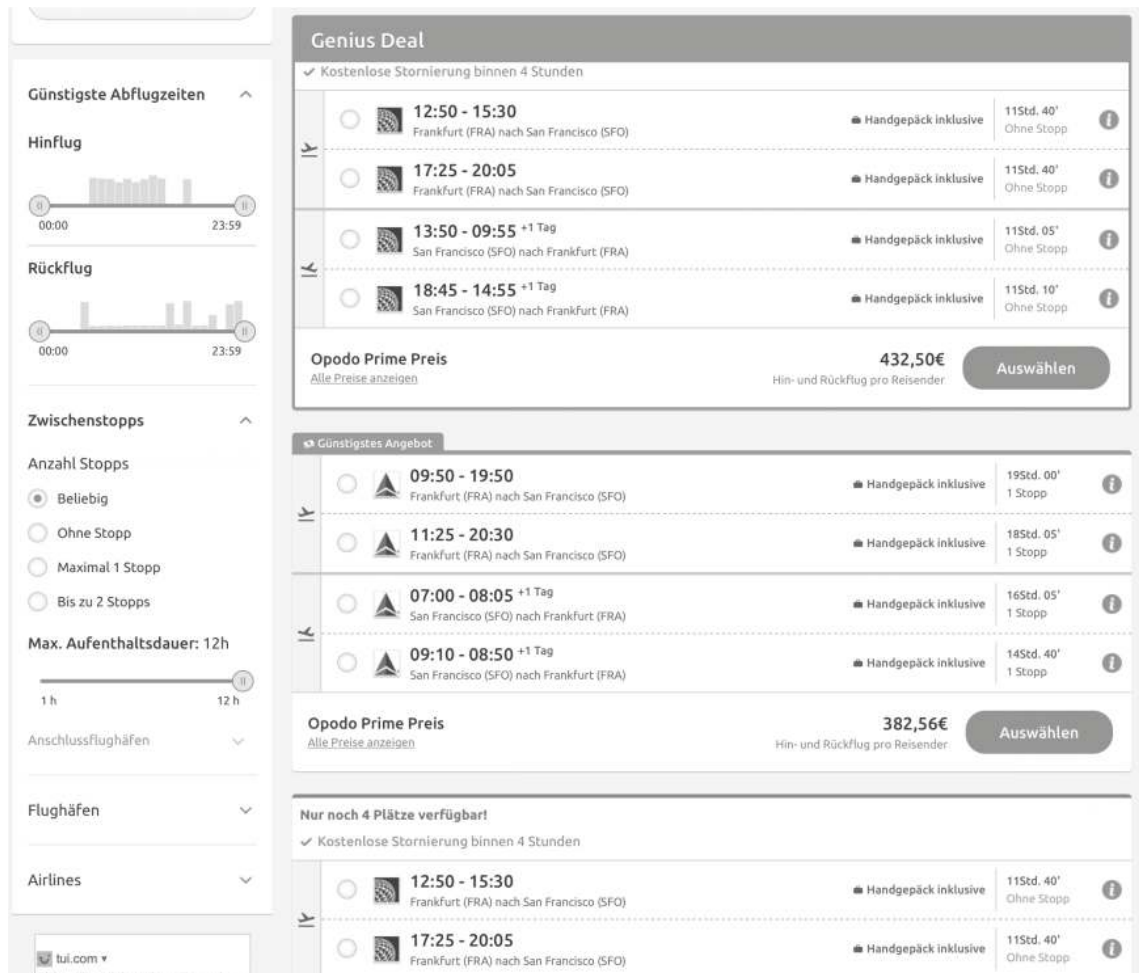


Figure 9.13 – Flights filtering interface. Image credits: Opodo.

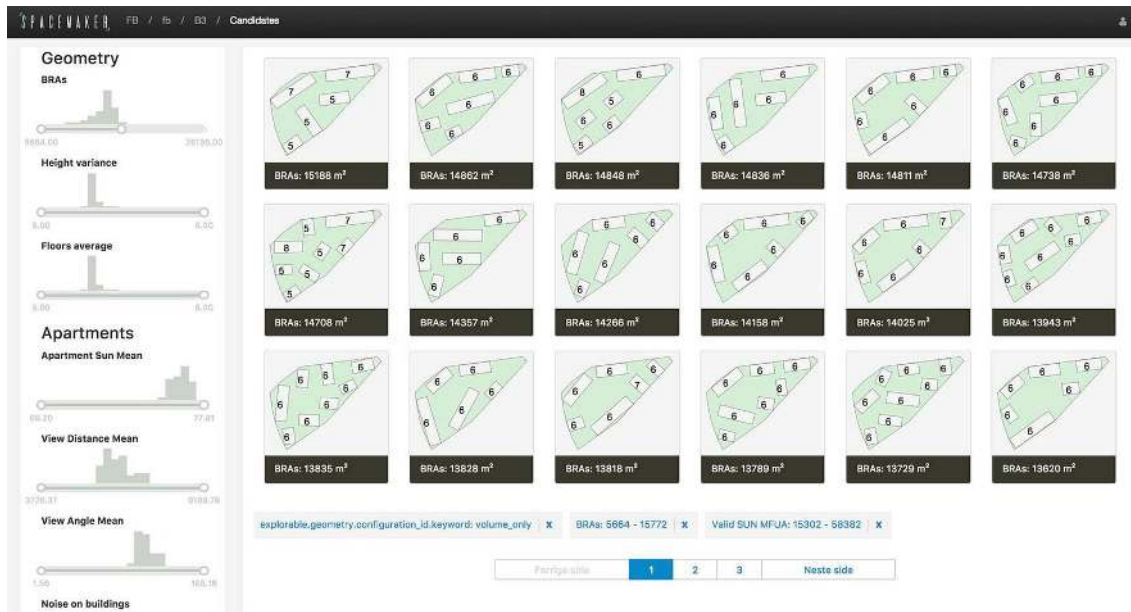


Figure 9.14 – Spacemaker Optioneering Interface. Image credits: Spacemaker.

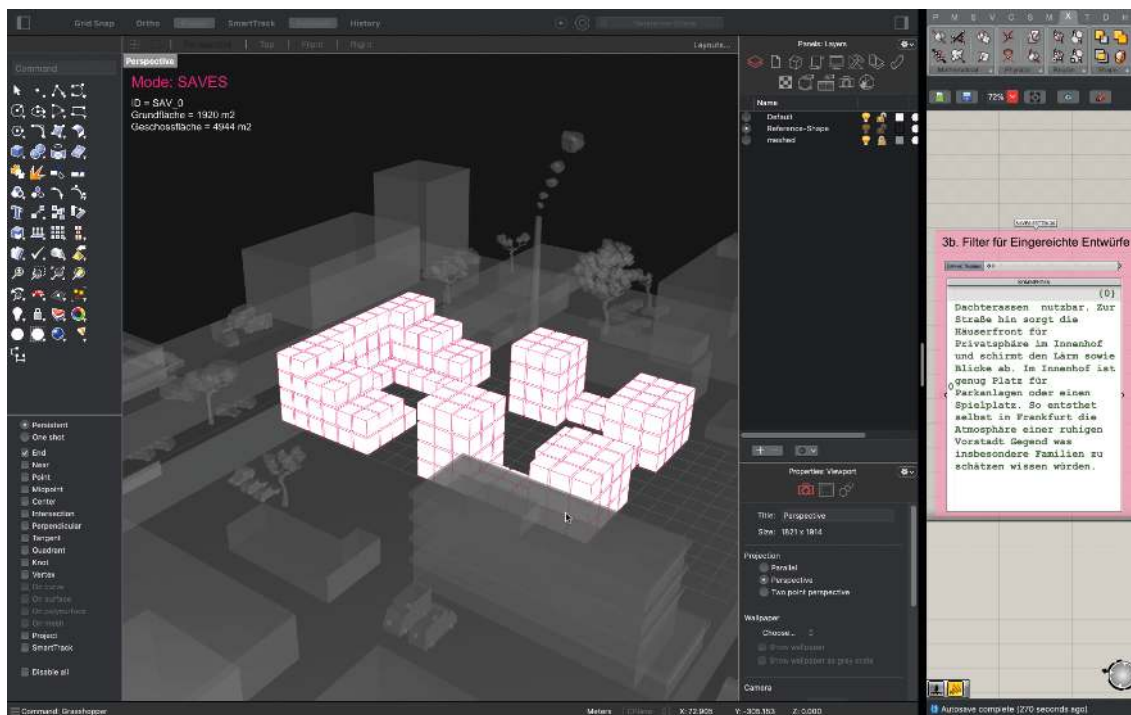


Figure 9.15 – Project Reptiles filtering interface. Image credits: Roger Winkler.



Figure 9.16 – Google autocomplete. Image credits: Google, Inc.

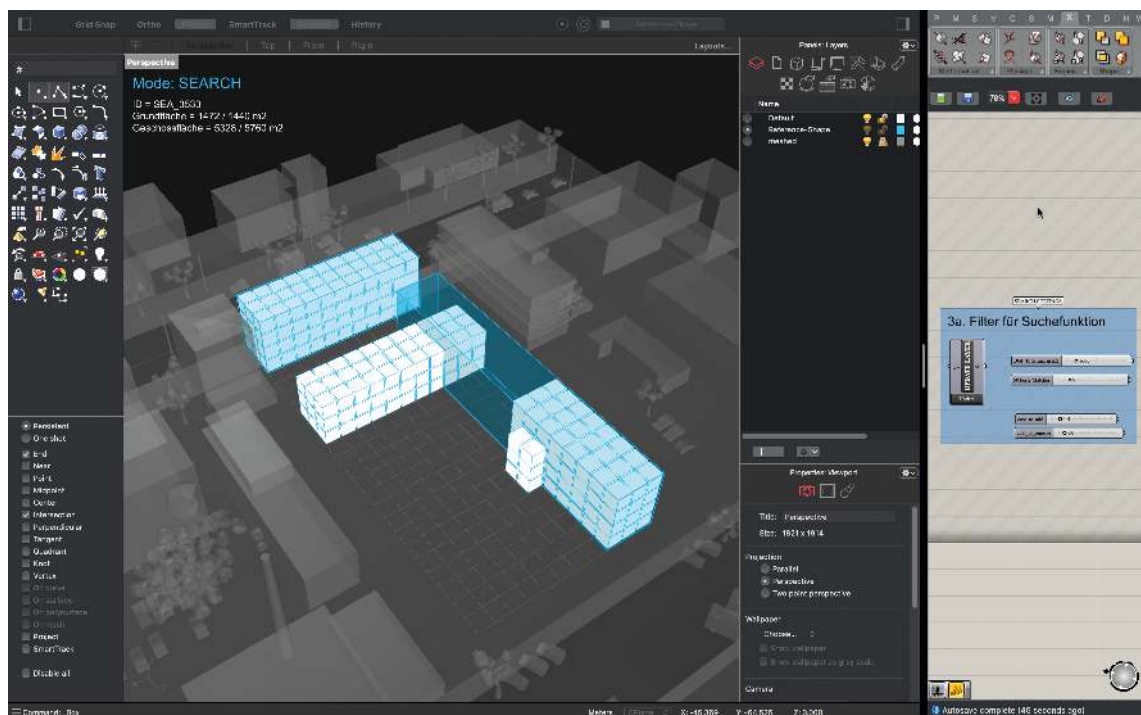


Figure 9.17 – Autocomplete in Project Reptiles. Image credits: Roger Winkler.

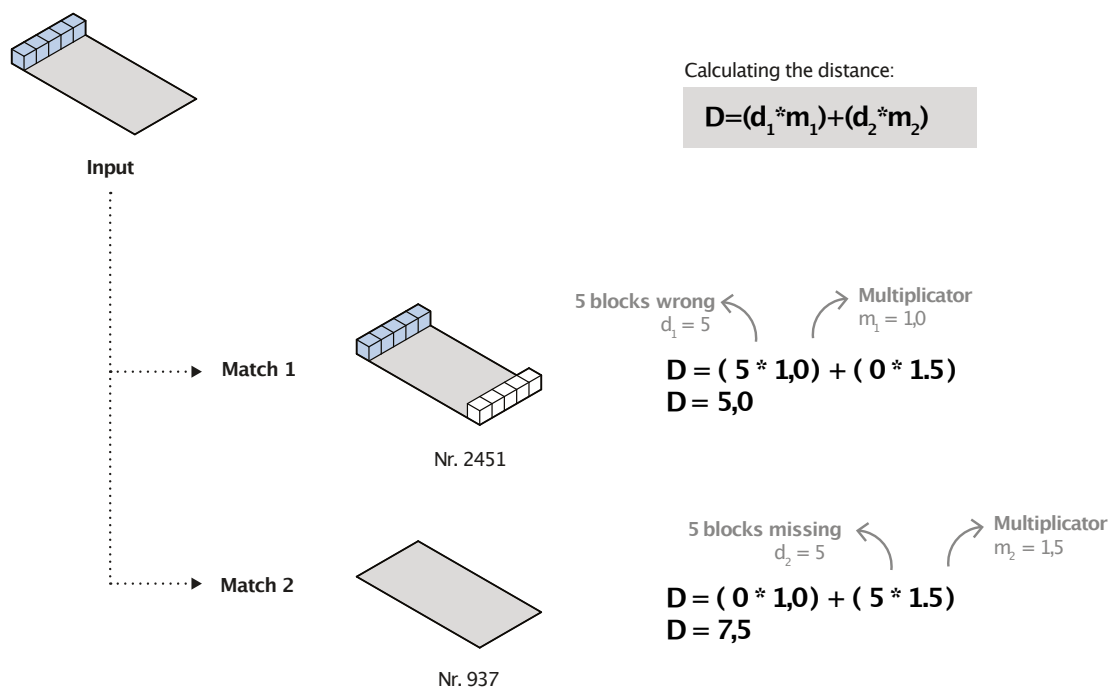


Figure 9.18 – Similarity metric. The formula for calculating the distance between the 3D search phrase and the crowdsourcing designs. Image credits: Roger Winkler.

9.3 Machine Agency

As described in *chapter 3: Generative Design*, a purely automated generation process is most often opaque and non-iterable. The user cannot influence the result while the process runs but is simply presented with an outcome. On the other hand, for a non-expert, it is not easy to advance in a design task if not assisted by a system that offers a considerable influence on the design, a kind of automated intelligence where expertise is encoded. The idea is similar to a driver-assistance system that is not an autopilot driving the car on its own but supports the driver in each of their actions while driving.

This approach could allow easy involvement of the non-expert via point-and-click actions. The user can *sculpt* the composition of the final product into a voxel grid while the configurator takes care of the correct selection of tiles. This gives the user a traceable, high-impact interaction - click only once to add volume to the building, and this volume is added with windows, roof, and walls matching perfectly to the rest of the building's envelope. In that aspect iso-surfacing, is used to *assist* the non-expert while modeling (Figure 9.19).

Iso-surfacing works similarly to how two drops of water lying on a surface merge into one drop if close enough to each other. This is different from how two Lego pieces placed next to each other remain two separate parts. The Marching Cubes (MC) algorithm works in the way described above (Figure 9.20). It creates closed iso-surfaces from a scalar field mapped to a voxel grid which makes it suitable for our approach (Lorensen and Cline 1987).

The method proposed here, *Assisted Sculpting*, uses a combination of iso-surfacing and constraint-solving to let experts encode architectural designs into interactive modeling environments. *Project Reptiles*, similar to *Brick Block* by Oskar Stalberg (Figure 9.21), uses the core functionality of the Marching Cubes (MC) algorithm to automatically create architectural structures from the iso-surfaces it generates based on a scalar field which the user can modify.

The expert models the tiles using the MC tileset as a topological base and defines constraints for their placement similar to the rules in the WFC algorithm. The WFC algorithm takes care of design variability, the MC algorithm takes care of turning a simple point and click action into a coherent building geometry.

The case study *Project Reptiles* consists of two phases:

1. Crowdsourcing of tileset designs
 - (a) Develop an application for *assisted sculpting* with a tileset editor.
 - (b) Ask architects to model existing designs with it and document the extent to which the given design was encoded as a tileset.
2. Crowdsourcing of building designs
 - (a) Develop a web-based online configurator for massing studies of a building on a specific site.
 - (b) Crowdsourcing alternative designs with an open call.
 - (c) Conduct a mock competition with architectural students for the same site, providing half of them access to the crowdsourced designs and tools to browse and filter them.

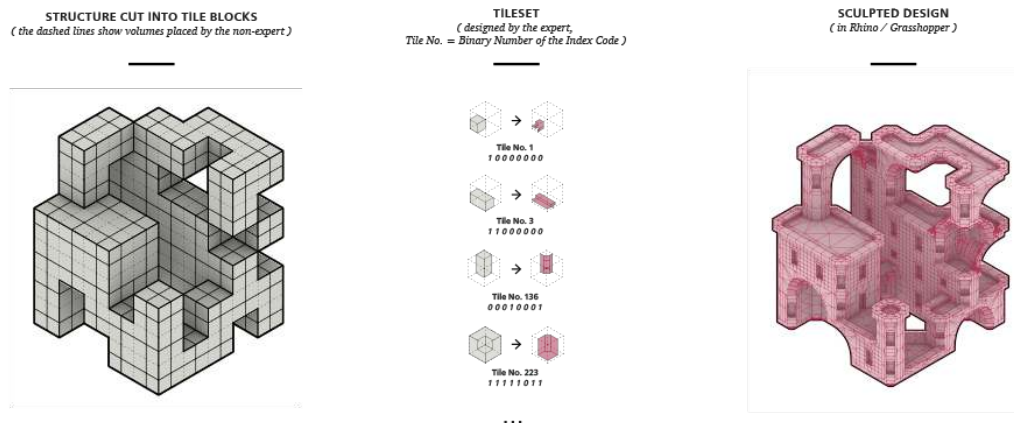
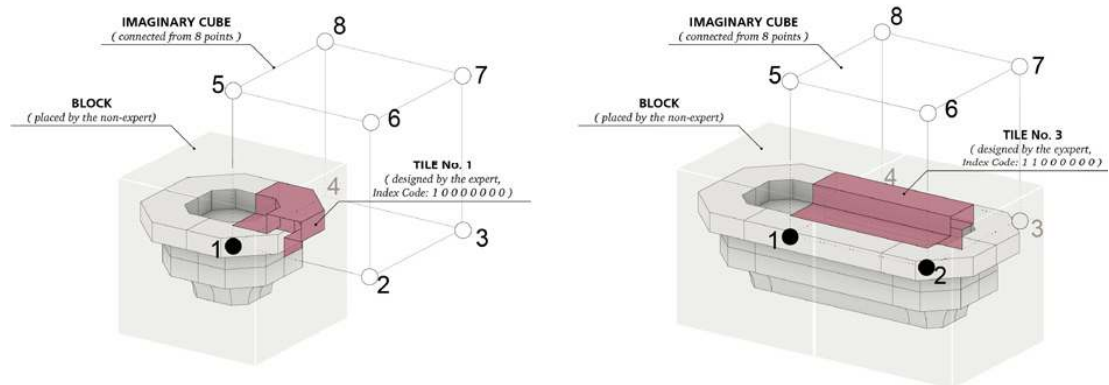


Figure 9.19 – Marching cubes logic on architectural tilesets. The aggregation of tiles in assisted sculpting creates unbody designs instead of part assemblies due to the iso-surfacing algorithm behind it. Image credits: Roger Winkler.

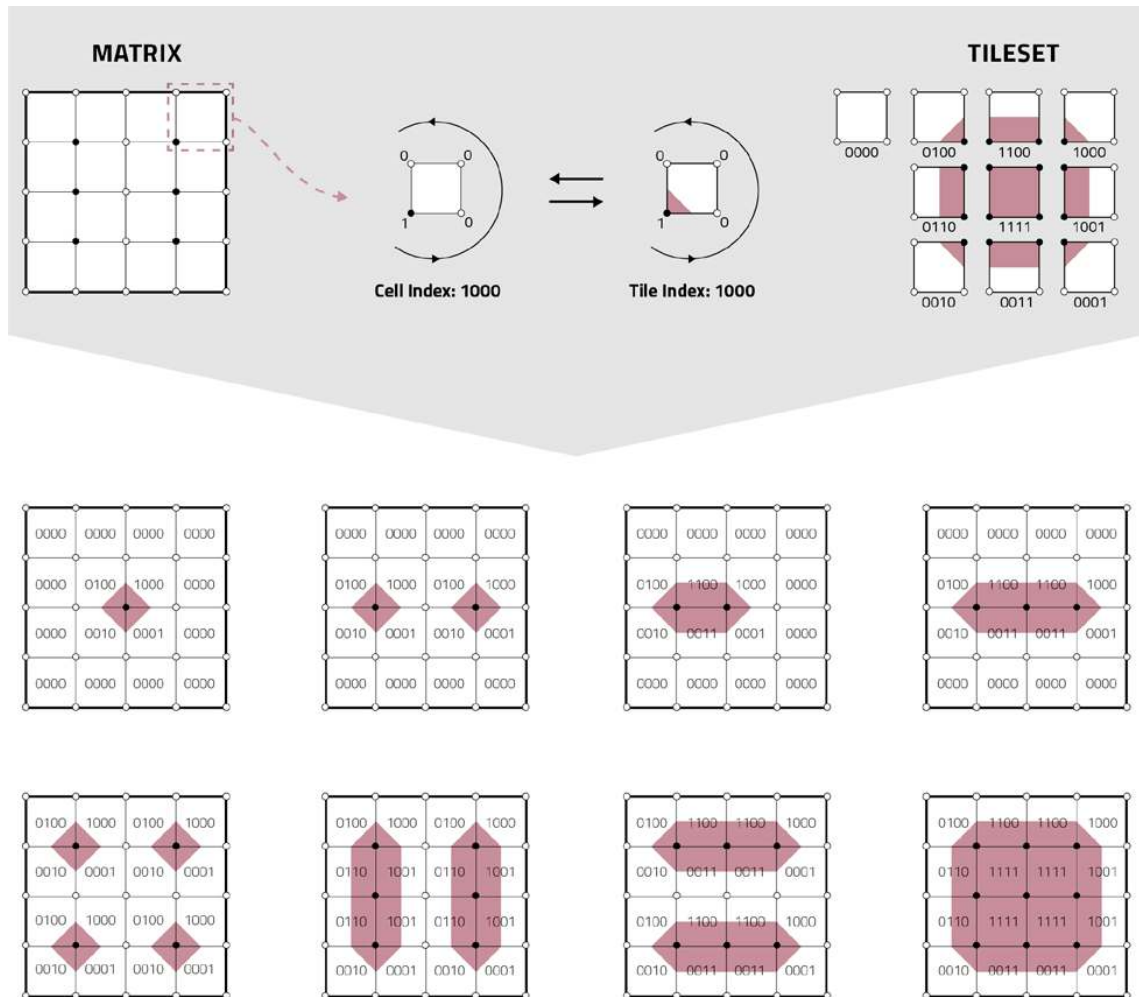


Figure 9.20 – Marching squares. The components of the Marching Cubes Algorithm in 2D (known as Marching Squares): a 2D matrix, a tileset, a ruleset and possible results. Image credits: Roger Winkler.

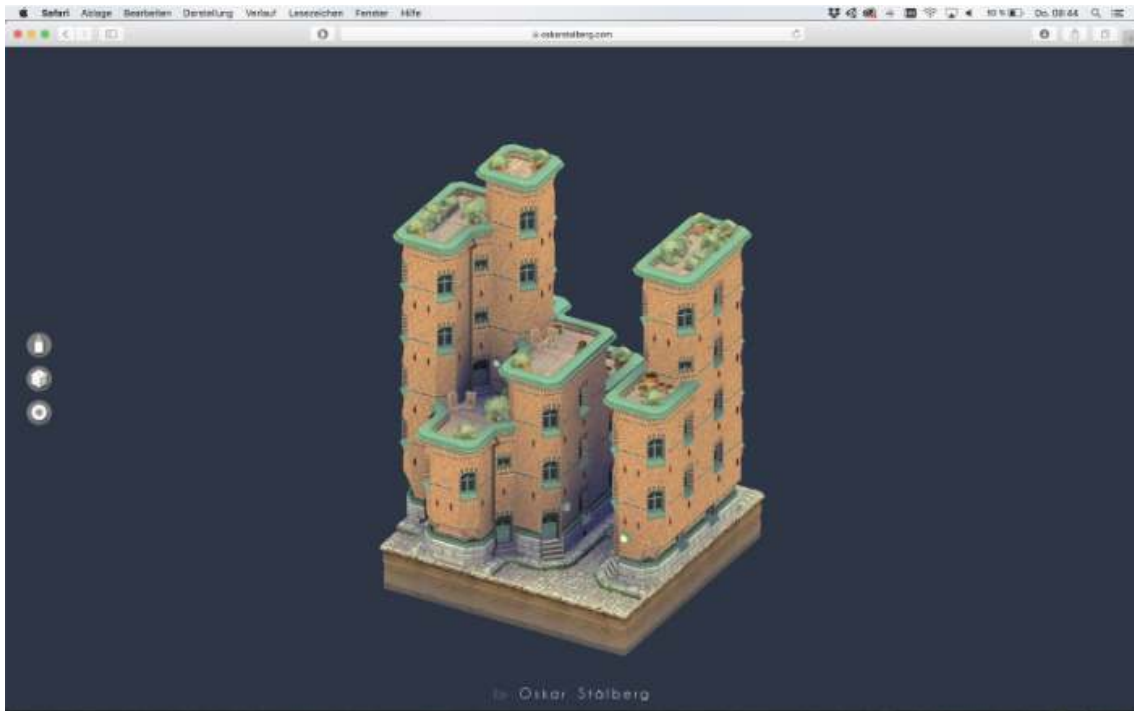


Figure 9.21 – Brick Block. The browser-based game *Brick Block* by Oskar Stalberg uses a custom tiles set for the marching cubes algorithm to enable the interactive generation of building envelopes (Stalberg 2016). Image credits: Oskar Stalberg.



original Villa Savoye



rebuilt Villa Savoye



sculptured variations of the Villa Savoye

Figure 9.22 – Design options. Variations of an architectural reference created using the iso-surface tilesets method. The tileset is based on Villa Savoye by Le Corbusier. Image credits: Roger Winkler.

9.4 Human Agency

9.4.1 Crowdsourcing the design of tilesets

The goal of crowdsourcing the tileset design is to determine to what extent architect can maintain their creative freedom when using the constraint-based generative design technique that forms the basis for the *Assisted Sculpting* approach.

Setup

We conducted two workshops with architecture students. In both workshops, the participants had six hours to work on their tilesets (Figure 9.23).

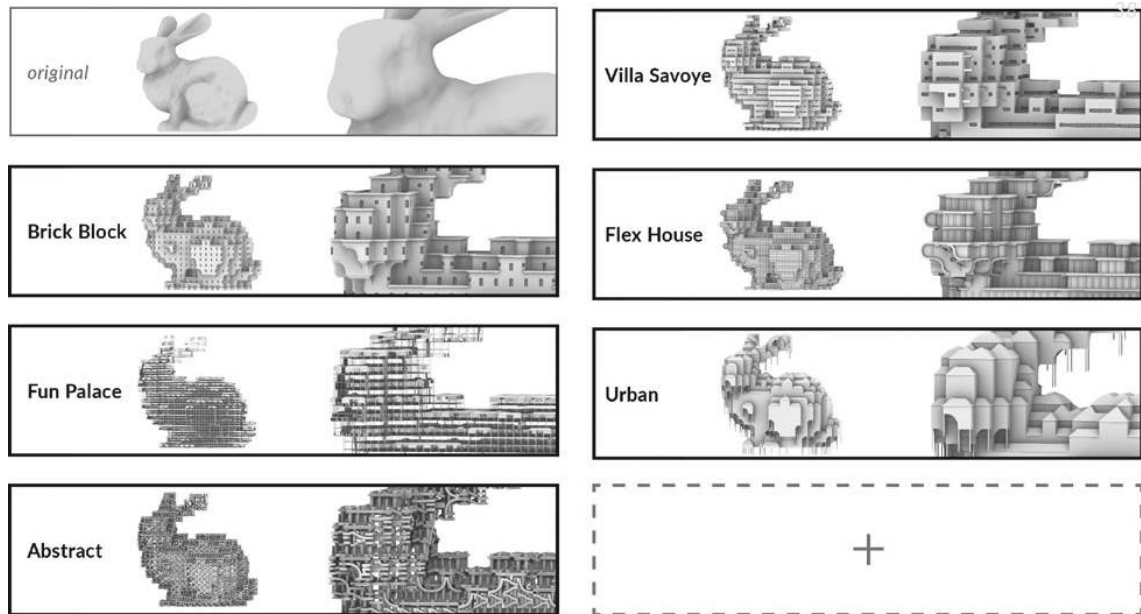


Figure 9.23 – Tilesets catalog. The tilesets created by workshop participants mapped onto a voxelized version of the Stanford Bunny. Image credits: Roger Winkler.

In the first workshop, the focus was on creating massing models of buildings. Students were asked to decompose an architectural reference, such as the Villa Savoye by Le Corbusier, into tilesets (Figure 9.22). The second workshop focused on solving the space allocation of a schematic design for a building. Students were asked to model access elements such as ramps and stairs into tilesets.

We advised the participants to model the 15 key tiles first, which provides two main advantages:

1. The surrounding conditions for all other tiles are given by the first 15. Unexpected problems during the modeling process, where the surfaces of two adjacent tiles do not match, could be prevented.
2. These 15 tiles are needed to build the first simple volumes (Figure 9.4).

In the workshop, students were supported by implementing new features to match the needs of their projects. For example, we introduced a rule that two different wall tiles must alternate on subsequent floors to support the continuously curved façade found in the Flex House (Figure 9.5). To avoid starting from scratch,

we introduced a random shape generator. Another feature was developed to include elements (points, curves, surfaces) in the tileset that were post-processed to generate structures beyond the voxel grid, such as the ramps within the Fun Palace project. Furthermore, the continuity of accessible areas could be achieved with tilesets in the second workshop (Figure 9.24).

The degree to which the tileset replicated the original design was documented as follows. The architect observed a geometric rule characteristic of the design of the building in question. For example, the first floor in Villa Savoye is made up of elements that are piloti, and the house has the typical modernist ribbon windows (Figure 9.22). We checked whether these geometrical design rules could be successfully recreated in our design system.

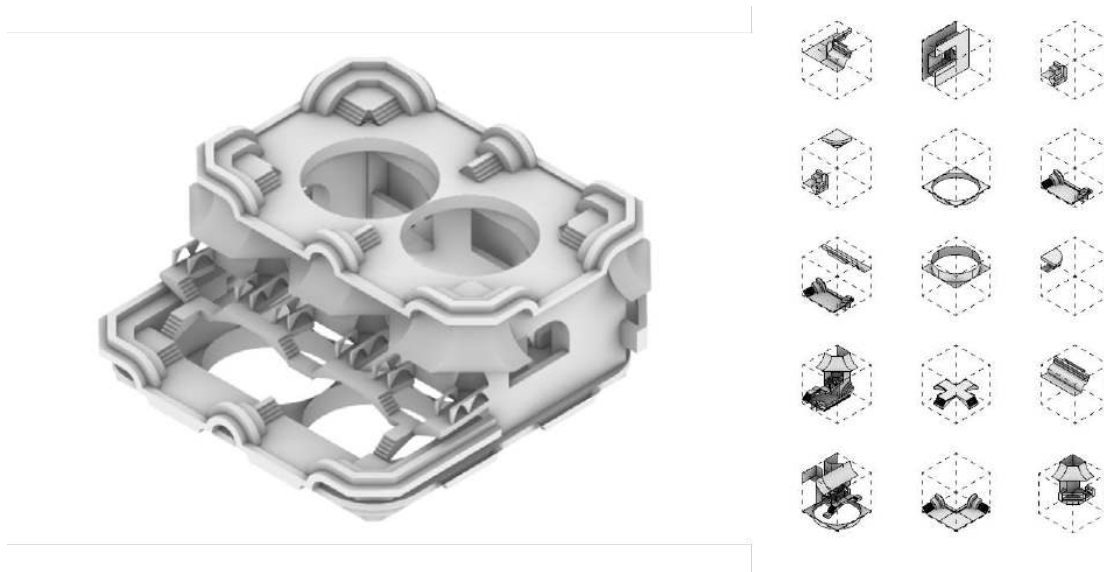


Figure 9.24 – Connectivity tileset. Structure created by a workshop participant with the first 15 tiles. Image credits: Roger Winkler.

Findings

Key findings from the Tileset crowdsourcing phase:

- In general, the observed geometric rules in a given design could be successfully remodeled as tilesets. However, the method is limited to largely orthogonal designs.
- Rules and tiles were more easily defined and added into the script when participants worked with existing buildings as reference.
- When the participants focused on infrastructural typologies without explicit references, this was more difficult, i.e., a clear idea of the designer is a prerequisite for modeling the tileset.
- Workshop participants were often overwhelmed by the many dependencies between the tiles when starting to model. To help with that, we showed them the 15 critical tiles to begin from.

- Introducing a random shape generator made it easier for the participants to understand the logic of the iso-surfacing algorithm and, therefore, start modeling new tilesets.

9.4.2 Crowdsourcing the design of buildings

Setup

The second phase in *Project Reptiles* aims to explore the potential for crowdsourced designs to augment the abilities of architects when proposing schematic designs for a specific site. The crowdsourcing phase used a very simple tileset focusing more on the envelope variations than on the detailed facade design. The project by Oswald Mathias Ungers for the Roosevelt Island Housing competition from 1975 shown on Figure 9.25 is the conceptual reference. The second phase included a user study where architectural students participated in a mock competition to design a building for a plot in Frankfurt.

The process begins with an open call to the online crowd to configure designs in the online configurator. The online configurator gives feedback on quantifiable parameters. To truly tap into the online crowd's imagination and creativity, the aim is to take their focus away from the mere optimization of the number of inhabitants and their access to daylight. Therefore the users are shown illustrations and atmospheric representations of urban living, such as views of the skyline, green roof terraces, gatherings of people around the dinner table, etc.

Participants are recruited using ads in social media (Figure 9.27). More than 3.500 people saw the ads, of whom 943 watched the promotional video, and 73 started the configurator. We recorded a total of 155 design sessions, resulting in more than 9.000 design states and 35 submitted designs (Figure 9.28).

With the crowdsourcing completed, ten architectural students are given 8 hours to propose three design options for the site in Frankfurt (Figure 9.29). Half of the architects, the control group, design using techniques they are familiar with (Figure 9.26). The other half, the test group, are given access to the crowdsourced designs and are introduced to the database browsing tools and the autocomplete function.

The 30 proposals from both control and test group are anonymized, standardized and mixed together (Figure 9.30). Each project included a physical model in styrofoam (Figure 9.31), the same key perspective, a section, a site plan (Figure 9.32) and a short text description. Subsequently a mock jury ranked all designs in three phases (Figure 9.33). The ranked designs are given on Table 9.1.

Findings

Three main factors influenced the quality of proposals from the crowdsourcing step:

1. the profile of the participants,
2. the number of participants and
3. the feedback given by the configurator.

It can be noted that the greater the diversity and number of participants in the crowdsourcing step phase, the greater is also the range of suggestions submitted.

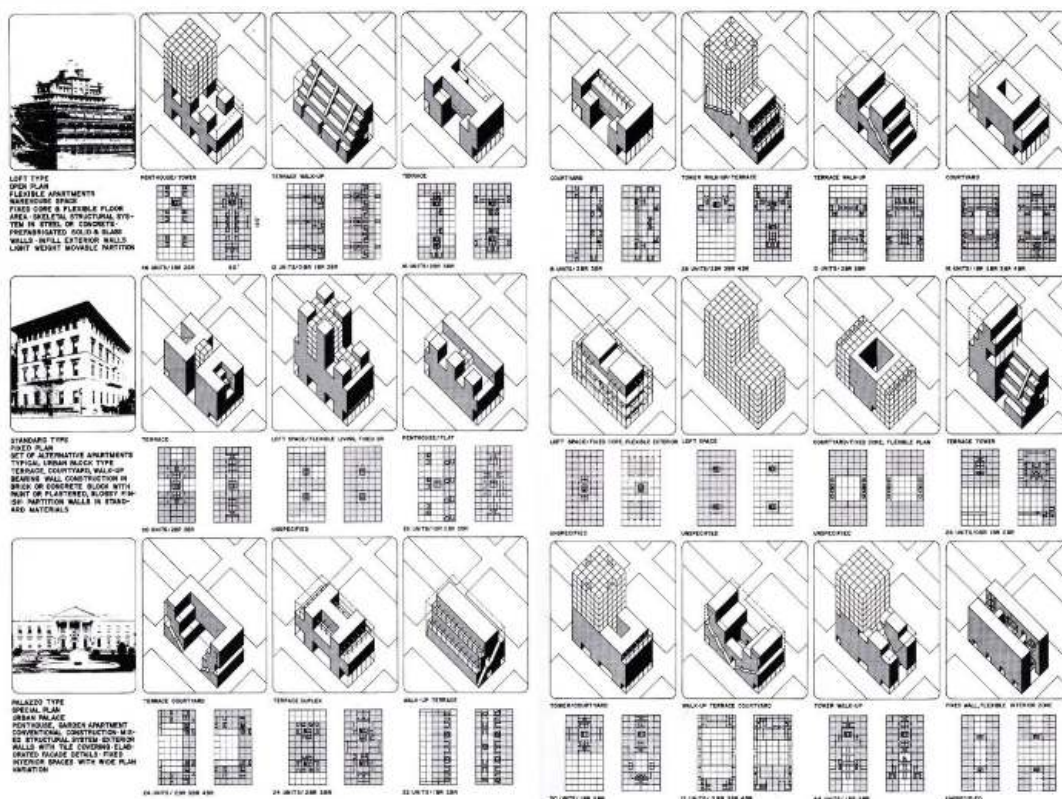
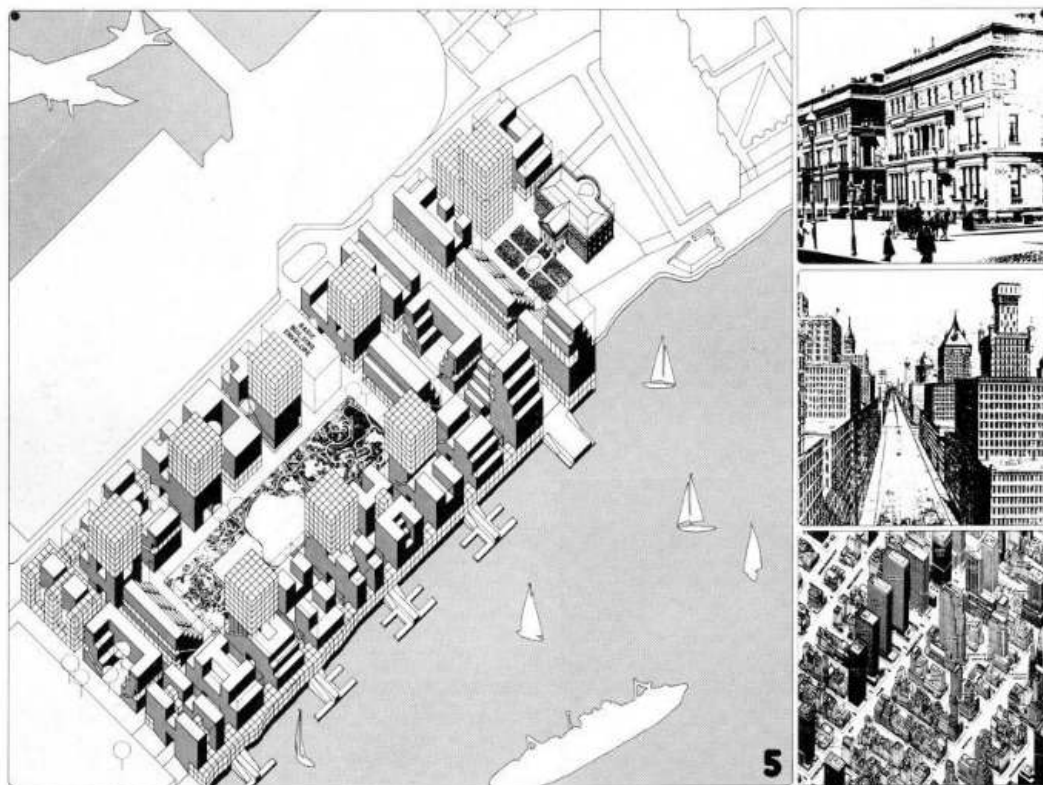


Figure 9.25 – Project by Oswald Mathias Ungers for the Roosevelt Island Housing competition, 1975. Image credits: Ungers.

In this context, intrinsic and extrinsic motivations for the specific project are additional influencing factors. For example, a significantly higher design quality can be

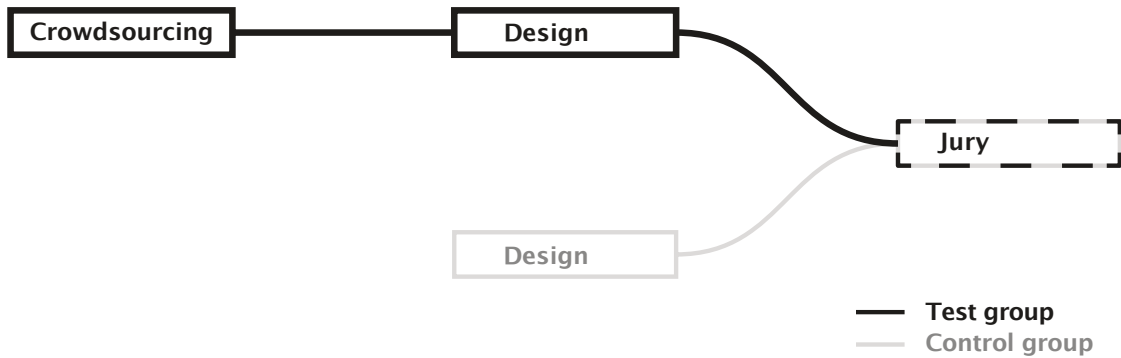


Figure 9.26 – User study setup. Image credits: the author.



Figure 9.27 – Project Reptiles ad on Facebook. Image credits: Roger Winkler.

expected if the configurator is used in an architectural office to collect ideas.

Furthermore, the quality of the feedback in the configurator directly influences the quality of the design proposals submitted. The case study gave feedback on the number of inhabitants and access to daylight. It would be interesting to quantify other qualities, such as how many blocks would directly see the skyline, the river, or the lake.

The tileset used in the 3D representation is also a kind of feedback. The user study used one tileset of the many crowdsourced in the first phase. A more diverse selection of tilesets and switching between them would allow the user to respond more specifically to the design task.

The factors that influenced the design process of the architectural students who worked with the collected results were:

1. the number of participants,
2. the search tool and
3. the quality of the submitted proposals from crowdsourcing.

The profile of the people participating in the workshop had a significant impact on the overall design process and results. Practicing architects with experience in

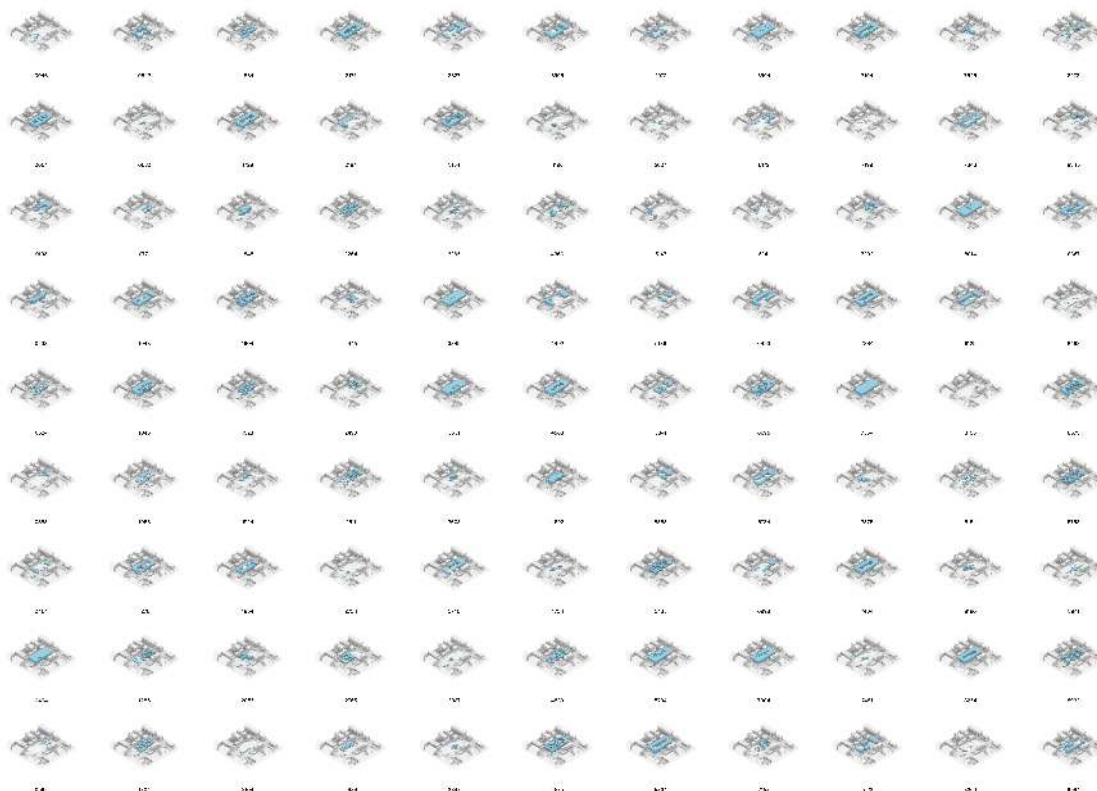


Figure 9.28 – Crowdsourced designs. A fraction of the more than 9.000 collected design states in the crowdsourcing step. Image credits: Roger Winkler.

residential projects and dealing with competition guidelines could probably judge and process the submitted design proposals much better.

In addition, the control and test groups each had the same amount of time to complete the task. However, since the participants in the test group first had to familiarize themselves with the search tool and the submitted designs, they had less time for the actual design process. It could be advantageous to introduce the search tool to the test group first and then have both groups start working on the task. This way, both the test and the control groups would have the same amount of time to develop design ideas.

The feedback provided to the crowdsourcing participants would also have been of interest to the test group in the search tool. Thus, it would not only contribute to the understanding of the submitted designs but could also influence what the architect designs.

The mock competition resulted in almost all designs produced with the help of the crowdsourced design being dismissed at the first or second round. Only one made it to the final round and got second place.

The jury's composition was very homogeneous, with all members being architects from the academic environment with a similar academic focus. A more diverse jury would benefit the evaluation of the results.

Another influencing factor was the choice of evaluation criteria provided to the jury. Ideally, the evaluation criteria should be aligned with the feedback given to the crowd and the design requirements for the workshop participants so that a completely transparent evaluation process would emerge. This was not considered in the execution of the process in this work.



Figure 9.29 – Photos from the mock design competition. Image credits: Roger Winkler.

The designs created by the test group, influenced by the crowdsourced designs, appear to be wild and less orderly. Therefore, most of them can be distinguished from the designs of the control group. The noise can result from a mismatch in the evaluation criteria provided in the three steps or from the computational augmentation of the design process. This might have led the jury to exclude such designs, even if they had been good according to quantitative and qualitative criteria. Possibly, the aesthetic senses are not used to this kind of semi-algorithmic order.

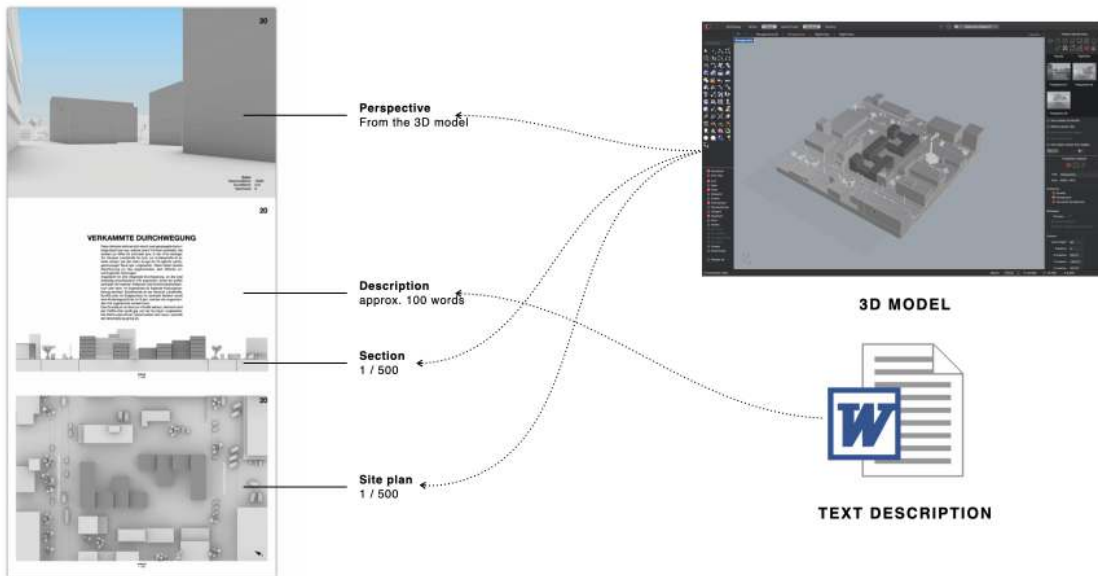


Figure 9.30 – Normalization step. All 30 entries are normalized before the mock jury ranks them. Image credits: Roger Winkler.

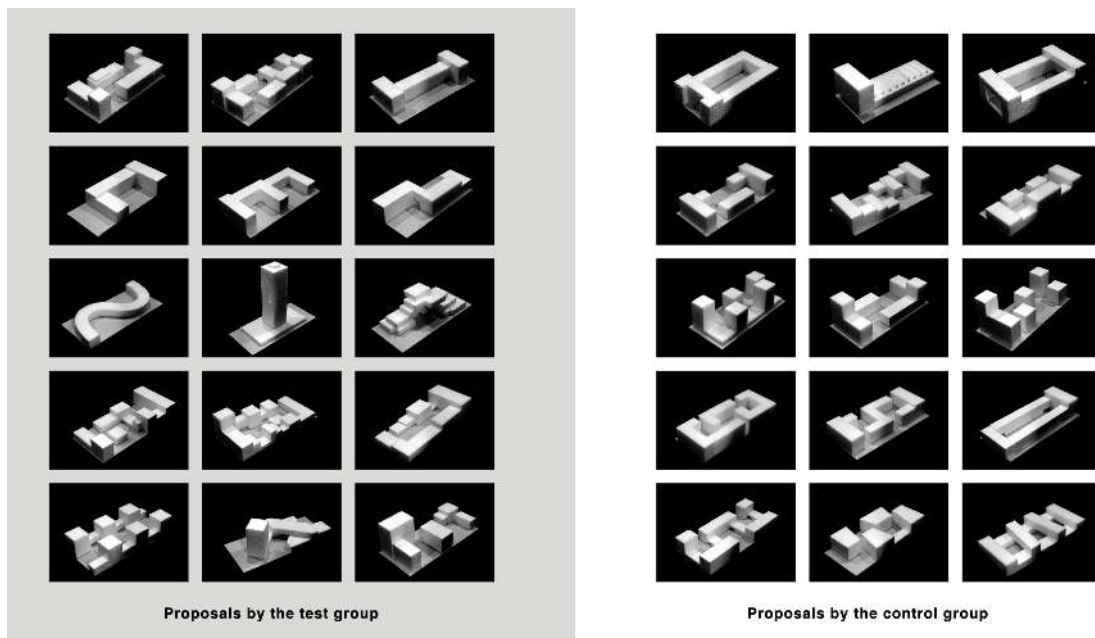


Figure 9.31 – Model photos. The photos of the models on the left are from the test group, on the right from the control group. Image credits: Roger Winkler.

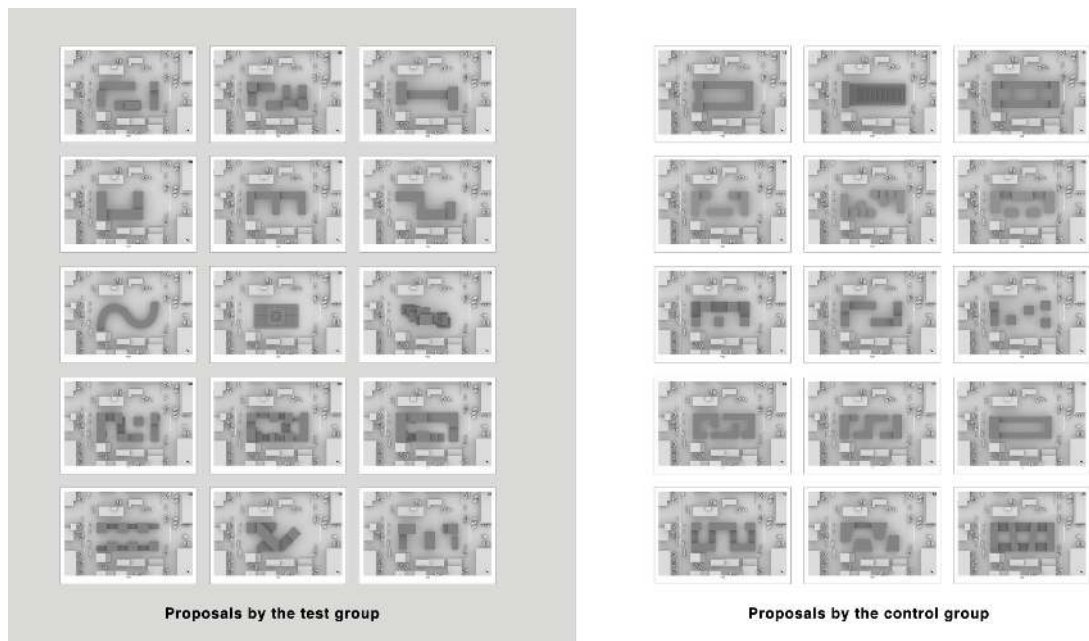


Figure 9.32 – Site plans. The site plans on the left are from the test group, on the right from the control group. Image credits: Roger Winkler.



Figure 9.33 – Photo from the ranking process. Image credits: Roger Winkler.

Number	Area	Groundfloor Area	Floors	Place	Excluded
10	10100	2620	6	1	finalist
18	10100	1850	10	2	finalist
08	10100	2380	6	3	finalist
01	10600	2070	9	4	finalist
20	10400	2160	6	5	finalist
04	11900	2400	5	6	finalist
24	10800	3000	5	6	finalist
06	9700	2270	9	8	finalist
13	10000	2230	5	9	finalist
03	10000	2048	6	10	2nd round
30	9500	2000	5	10	2nd round
25	9700	2500	25	10	2nd round
17	7000	948	10	10	2nd round
22	7900	2400	5	10	2nd round
12	13000	2640	5	10	2nd round
09	12400	3000	9	10	2nd round
27	10400	1400	12	10	2nd round
19	10100	2110	6	10	2nd round
15	9700	2324	6	11	1st round
28	9800	2436	6	11	1st round
05	9900	2048	6	11	1st round
16	9500	1968	6	11	1st round
11	5200	1726	3	11	1st round
29	8500	2760	8	11	1st round
07	9200	3050	5	11	1st round
02	10800	2300	6	11	1st round
26	9600	1730	10	11	1st round
14	7200	2450	6	11	1st round
23	9500	1810	6	11	1st round
21	10900	2220	5	11	1st round

1 Design by the test group

1 Design by the control group

Table 9.1 – Ranking. The ranking of the designs submitted in the mock competition after the mock jury. Image credits: Roger Winkler.

9.5 Take-aways

The *Project Reptiles* case study introduces the concept of *assisted sculpting* which uses a combination of iso-surfacing and constraint-solving to help non-experts in exploring design options. Design tasks such as massing models and schematic space allocation were tested, the former being more successful. The user studies show that *assisted sculpting* is easy for non-experts and could bring them into a creative flow state. At the same time, the tileset editing mode offers a balanced creative challenge for architects making full use of their expert skills and knowledge. Users cannot edit the procedural algorithm. However, an interface for experts is provided to define the elements with which these procedural protocols operate.

A potential use case scenario for the approach presented with *Project Reptiles* is prefab and industrial construction. For example, a manufacturer of prefab houses (Fertigbau) would have a standard set of details and measurements that could be procedurally described and encoded into a tileset. That would ensure that all automatically generated designs based on the user's changes in the program would be compliant with the manufacturing process of the prefab company.

Project Reptiles reveals the following three principles in lowering the difficulty of a design challenge for both experts and non-experts:

1. Don't let users start from scratch — lowers the difficulty to start.
2. Give users a set of tools that have traceable, high impact on the design via simple actions — lowers the difficulty to express an idea.
3. Give users feedback to help them judge the value of what they have created — lowers the difficulty to decide which action to take next.

The feedback provided to users at all phases of the crowdsourcing pipeline must be based on the same parameters to avoid noise.

As an outlook, different input techniques can be explored. Since the only input needed for our method is a scalar field, there are various options available to stage the interactions that generate these fields. It can be done via a point-and-click as presented in this work. It can also be based on attractor logic, structural stress or other performance-driven analysis, partially automated tile placement, etc. Adding more quantitative feedback, such as solar analysis or cost estimates, could lower the challenge for the user in deciding what change to the design to make next. A similar effect can be pursued by staging the modeling experience like a game. This would also enable the exploration of the collaborative qualities of the computational design system and involve as many non-experts as possible.

Chapter 10

Rechteck2BIM

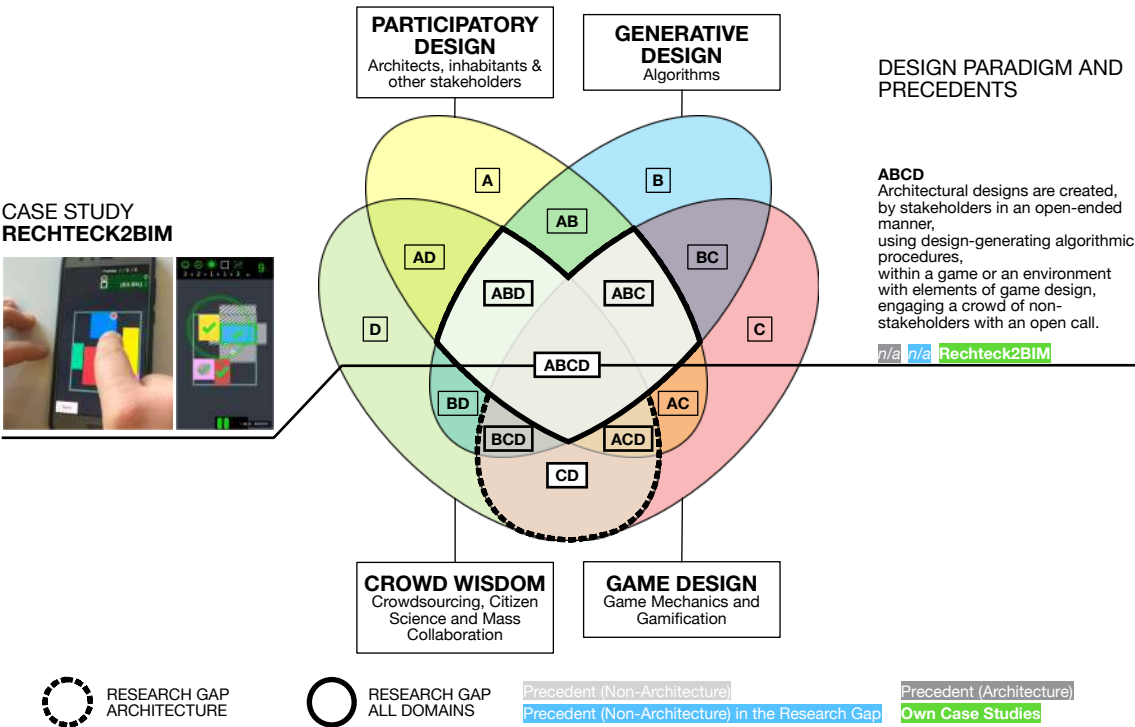


Figure 10.1 – Rechteck2BIM on the four fields map. The case study *Rechteck2BIM* explores the design paradigm at the intersection of all four investigated fields. Image credits: the author.

10.1 Design Paradigm

One of the most basic tasks for an architect is floor plan design. A survey with architects conducted by Nisztuk and Myszkowski 2018 reveals that the generation of functional layouts is one of the tasks architects would like to have automated the most (Nisztuk and Myszkowski 2018). Much work has been done in computationally generating residential layouts (Duarte 2005; Green 2020; Hu et al. 2020; P. Merrell, Schkufza and Koltun 2010; Stiny and Mitchell 1978; Zawidzki and Szklarski 2020). However, this state-of-the-art requires that generated designs are post-processed manually to give them the desired design expression.

When designing a new building, architects alternate between various media. They use the floor plans and sections to establish a functional relationship between the internal spaces of the building, as well as between the building and its context. Facades, digital 3D models, and physical models are used to find and represent the design expression of the building. Design expression is characterized by the raster pattern and layout of windows, facade cladding, the geometrical features where facade surfaces meet (corners, roof to wall, wall to the ground, etc.), and the materiality of the building.

Much work has been done in computationally generating residential layouts (Duarte 2005; Green 2020; Hu et al. 2020; P. Merrell, Schkufza and Koltun 2010; Stiny and Mitchell 1978; Zawidzki and Szklarski 2020). However, this state-of-the-art requires that generated designs are post-processed manually to give them the desired design expression. Some works have tried to automate the transfer of design quality from a small sample model to larger, automatically generated models such as Model Synthesis by Merrell and Wave-function Collapse (Karth and A. M. Smith 2017; Khokhlov, Koh and Huang 2019; P. Merrell 2007; P. Merrell and Manocha 2008, 2009; Tigas and Hosmer 2021). The approach was presented in the previous case study — *Project Reptiles* (See chapter 9). However, this work does not allow the designers to intuitively control the generated layouts' properties.

In the case study *Rechteck2BIM* I explore how the tasks performed by architects and homeowners when laying out the schematic floor plan for a new single-family house could be crowdsourced from the role typically carrying them out to another role.

Rechteck2BIM is a series of digital prototypes around this goal. The prototypes create a dynamic, real-time link between a user-friendly interface to create a floor plan layout and a detailed 3D BIM model to produce drawings and cost estimates. The procedurally-generated 3D model allows to run various analytics on the floor plan and provide feedback to the user.

The *Rechteck2BIM* case study employs techniques from all four fields of interest in this work: Participatory Design, Generative Design, Game Design, and Crowd wisdom. As such it is positioned within the design paradigm right at the core intersection ABCD (Figure 10.1). The paradigm postulates that architectural designs are created by stakeholders in an open-ended manner, using design-generating algorithmic procedures within a game or an environment with elements of game design, engaging a crowd of non-stakeholders with an open call.

10.2 Implementation and Setup

The *Rechteck2BIM* case study is set up as a series of prototypes structured in two main topics: generating design and computationally analyzing and sorting designs.

The *Rechteck2BIM* case study contains the largest number of prototypes from all case studies and presents the highest diversity in terms of means to implement them. The design-generating prototypes were implemented mainly in the game engine Unity. When special geometrical processing or generation was needed, the Unity prototype interfaced with Grasshopper. The Grasshopper model was also used to automatically generate a BIM model by sending the geometry to ArchiCAD. All prototypes for floor plan analysis were implemented in Grasshopper.

10.3 Machine Agency

Rechteck2BIM explores three approaches for user input, each picking a different generative technique from the taxonomy introduced in section 3.2 and making it interactive: simulation-based, constraint-based, and case-based. As all approaches concerned the user with abstract 2D shapes representing the rooms in a floor plan layout, I developed a pipeline to procedurally turn a composition of rectangles into a 3D model of a house. Additionally, computational analysis of floor plans was explored with architectural students in courses taught by me at the TU Darmstadt and my prototypes.

10.3.1 Interactive Floor Plan Generation

Simulation-based floor plan configurator

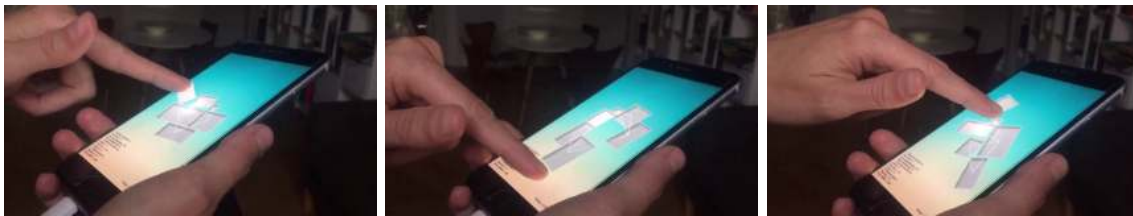


Figure 10.2 – Spring-based prototype for Rechteck2BIM. Image credits: the author.

The first approach uses a spring-based simulation to let the user create a floor plan (Figure 10.2). The user can move around rectangles representing the wanted rooms, snap together, and see a 3D house model defined by the rectangle composition. The rectangles are connected with springs if they represent rooms that need to be connected or adjacent in the house layout. The implementation builds on the doctoral work of Arvin 2004 described by Arvin and House 1999, 2002 and shown on Figure 10.3. The spring-based approach proved difficult for the user to control as it reevaluated all room positions after each interaction. This is especially true for more complex room programs with more than 5-7 rooms where the spring forces are far from equilibrium in most of the wanted manual arrangements (Figure 10.4). Synchronizing two consecutive floor plans in multistory houses becomes practically incomprehensible.

The spring-based prototype served as the base to develop the procedural 3D model generator as shown on Figure 10.5.

Constraint-based floor plan game

The second approach is a rule-based game that assigns the user points for placing rooms next to each other in an architecturally meaningful way. The game was implemented for Android devices under my supervision by five bachelor students in computer science.

Instead of forcing the room arrangement with physically-based simulation, the constraint-based game prototyped a method to measure fitness for a floor plan based on how close it meets desired qualities. I defined five game mechanics.

1. *adjacency mechanic* — rewards preferred adjacency between two rooms (Figure 10.7)
2. *separation mechanic* — rewards preferred separation of two rooms (Figure 10.8)
3. *daylight mechanic* — incentivizes providing daylight access for a room that requires it (Figure 10.9)
4. *footprint mechanic* — aims for minimizing the total area occupied by all rooms (Figure 10.10)
5. *stairs mechanic* — incentivizes placing stairs in suitable rooms (Figure 10.11)

The game shows the points achieved in each category and a total score (Figure 10.6). A game mission is defined by describing the brief for a new house as a list of rooms and their preferred adjacencies, exposure to daylight, etc. Multiple games can be played with the same brief as a challenge. The game mechanics offer the user an engaging, playful way to try hundreds of options for arranging a given set of rooms into a floor plan quickly.

Translating the constraints to game mechanics gives the user a feeling of control and choice. The ability to guide the rectangles towards a wanted arrangement is better than the spring-based prototype. The feedback from the game mechanics creates a predictable, learnable challenge to maximize the points for a room composition. The *stairs* mechanic allowed the configuration of multistory floor plans. The resulting layouts are shape compositions used to generate a 3D model and a BIM model.

However, describing a well-crafted coherent floor plan as a set of constraints is a task of high computational complexity (Liggett 2000; Z. Lin and Yingjie 2019; Pérez-Gosende, Mula and Díaz-Madroñero 2021). Often, users can find an arrangement that gives maximum points yet has gaps between the rooms; the outlines of upper floors have large overhangs or are architecturally problematic in other ways. Fine-tuning the rules and adding additional game mechanics built around a floor plan's performance can help to constrain the outcomes to architecturally meaningful layouts.

The constraint-based approach holds the highest potential for crowdsourcing novel floor plan solutions from the three tested methods. A robust filtering system based on computational floor plan analysis is needed to sort through them.

Case-based floor plan configurator

The third approach to allowing a user to define a design by describing the components of its architectural program is case-based. It relies on a large enough database of floor plans with good architectural quality. A search algorithm lets the user find the floor plans that best match their preferences and the site constraints.

The goal is to allow the user to define her preferences visually and have immediate feedback on the consequences of her choices. Two alternatives for the visual definition of preferences were prototyped: a comparison-based and a symbol-based.

Meizi Ren explored the comparison-based alternative in her student work supervised by me. Ren builds upon research by Jansen, Coolen and Goetgeluk 2011 stating that specific dwelling characteristics can be better described with the use of

floor plan images instead of text. The user needs to define her priorities according to six floor plan attributes such as the size of the master bedroom or the size of the living room (Figure 10.13). In the priority assessment, the importance of every two attributes is determined by showing two similar floor plans, distinguished only by those two attributes (Figure 10.14). A short description helps people to understand the implications of their choice. Participants are asked to pick one of the two floor plans. There are 16 questions for the priority assessment resulting from all possible attribute couplings (Figure 10.15). Every question presents at least one original floor plan available on the real estate market. Around 50 people completed the online survey (Figure 10.16). After the user completes the questionnaire, their results are mapped to an idealized floor plan histogram (Figure 10.17). The histogram allows to calculate a rank for every floor plan in the database (Figure 10.18). The user can then see a sorted list of top matches and an image of the floor plan (Figure 10.19).

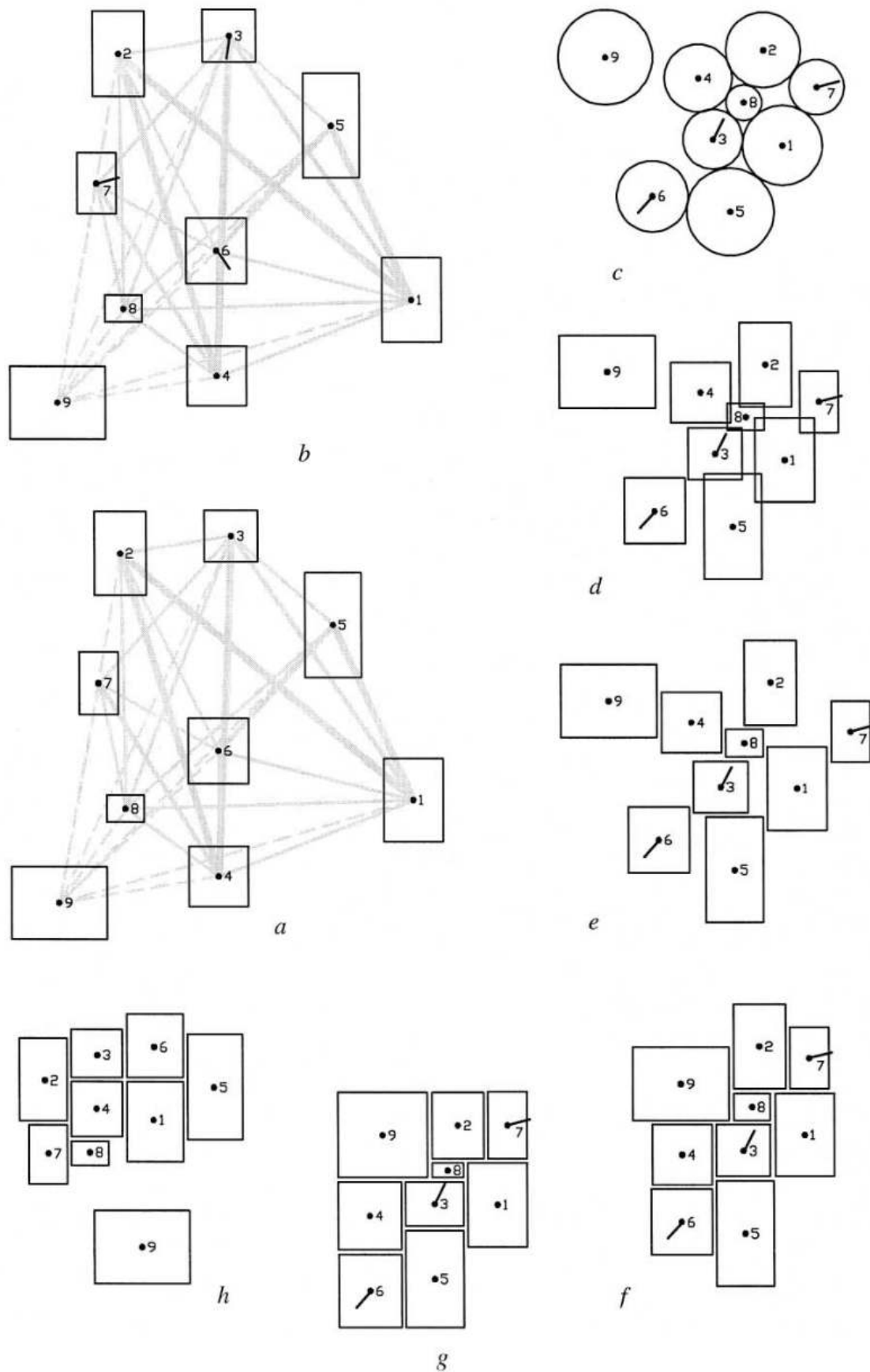


Figure 10.3 – A spring-based simulation approach to floor plan generation. Various stages as described by Arvin and House 2002. *a,b*: initial states, *a* has an extra interior objective for room 3, *c,d,e*: stages of the autonomous topological resolution. *f,g,h*: user manipulated resolutions with various gravity and alignment constraints turned on. Image credits: (Arvin and House 2002).



Figure 10.4 – 3D model generated from room arrangement. A more complex room program loaded in the spring-based configurator as a mobile app and the realtime view of a floor plan and 3D model on the desktop screen. Image credits: the author.

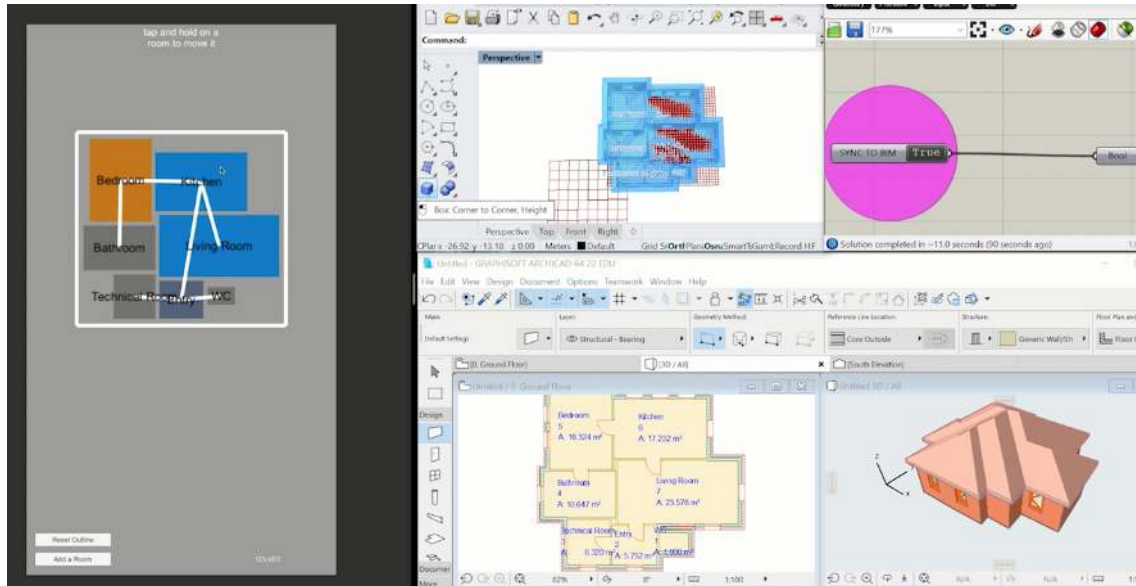


Figure 10.5 – Prototype of a simulation-based floor plan configurator from the case study Rechteck2BIM. Left: The user defines the rooms and their arrangement. Room connectivity is shown with white lines. Right: A 3D (top) and a BIM (bottom) model are automatically generated to provide feedback to the user. Sample feedback includes total area, cost estimate, the house's looks, the sun's shine in the morning, etc. Image credits: the author.

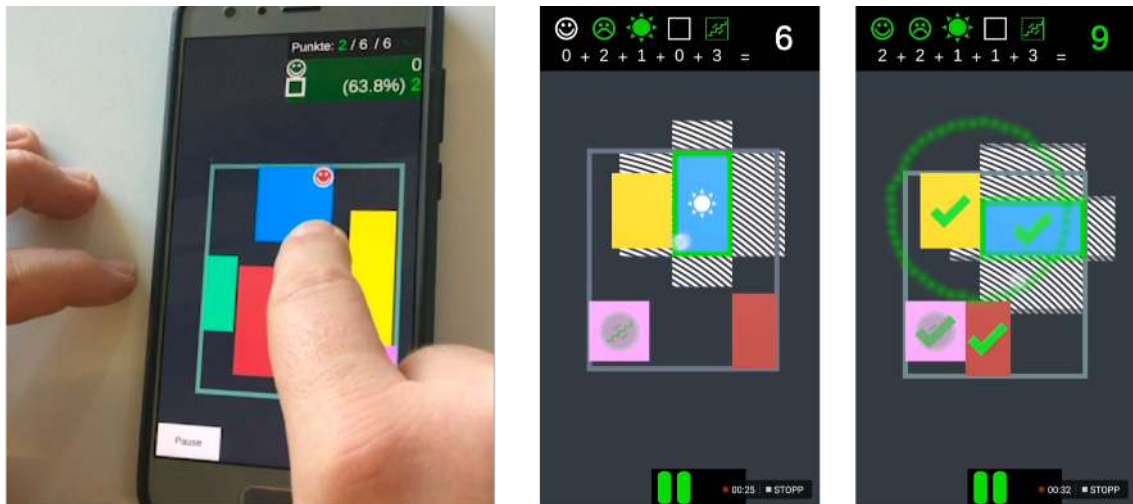


Figure 10.6 – Constraint-based floor plan game prototype. It uses five game mechanics to give the player feedback on how close is the composition they have created to the wanted layout features. The top of the game screen shows the points for each game mechanic separately and a total game score. Image credits: the author.

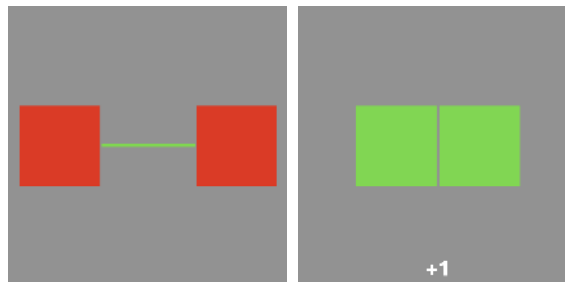


Figure 10.7 – Adjacency mechanic. The *adjacency game mechanic* gives a point if the player slides together two rooms that preferably must be next to each other in the floor plan. For example master bedroom and master bathroom. Image credits: the author.

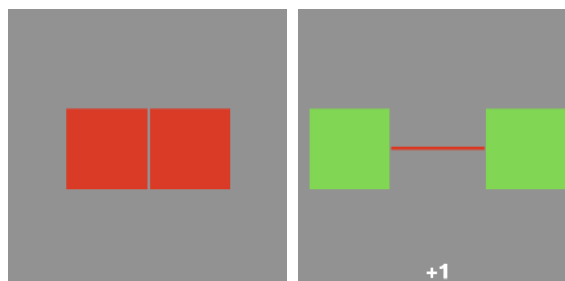


Figure 10.8 – Separation mechanic. The *separation game mechanic* gives a point if the player slides two rooms apart if they need to be non-adjacent in the floor plan. For example bedroom must be away from an elevator shaft. Image credits: the author.

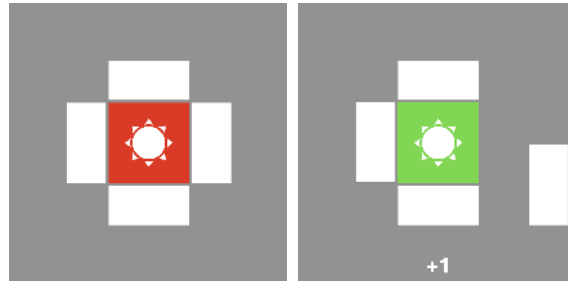


Figure 10.9 – Daylight mechanic. The *daylight game mechanic* gives a point if the player leaves free at least one side of a room that requires daylight. For example a living room. Image credits: the author.

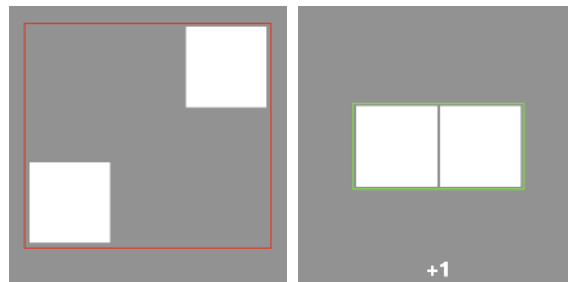


Figure 10.10 – Footprint mechanic. The closer the bounding box area of all rooms is to the sum of the areas of the room the more points does the *footprint game mechanic* give the player. Image credits: the author.

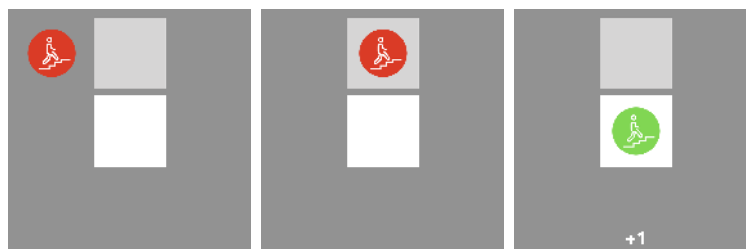


Figure 10.11 – Stairs mechanic. The *stairs game mechanic* gives a point if the player places the stairs in a room that is suitable. For example stairs are ok in a foyer or a living room but not in a bathroom. Image credits: the author.

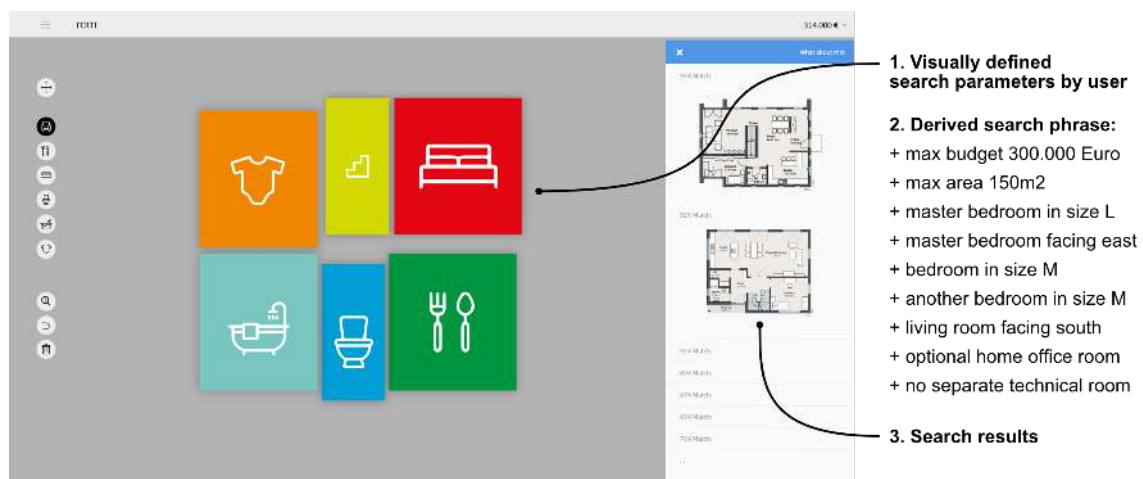


Figure 10.12 – Case-based floor plan suggesting tool. Image credits: Roger Winkler.



Figure 10.13 – Floor plan attributes. The six floor plan attributes which the user can express a preference for and priority as well. Image credits: Meizi Ren.



Figure 10.14 – Sample question. An example from the visual questionnaire. Question 4 aims to establish whether the user will prefer a larger master bedroom over a larger living room or vice versa. Image credits: Meizi Ren.

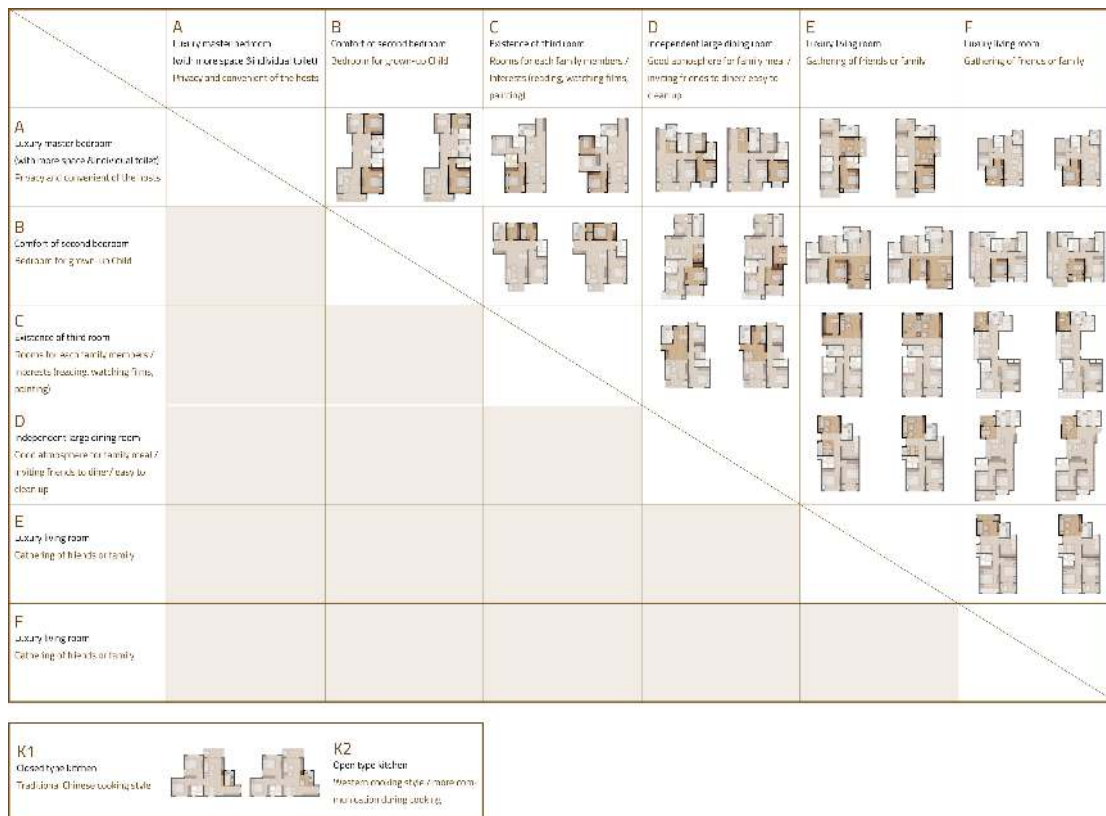


Figure 10.15 – Attribute-ranking matrix. All possible comparisons between the six floor plan attributes A-F result in 15 questions to establish the user's prioritization of their preferences. The 16th questions establishes whether the user prefers an open or closed kitchen (K1-K2). Image credits: Meizi Ren.

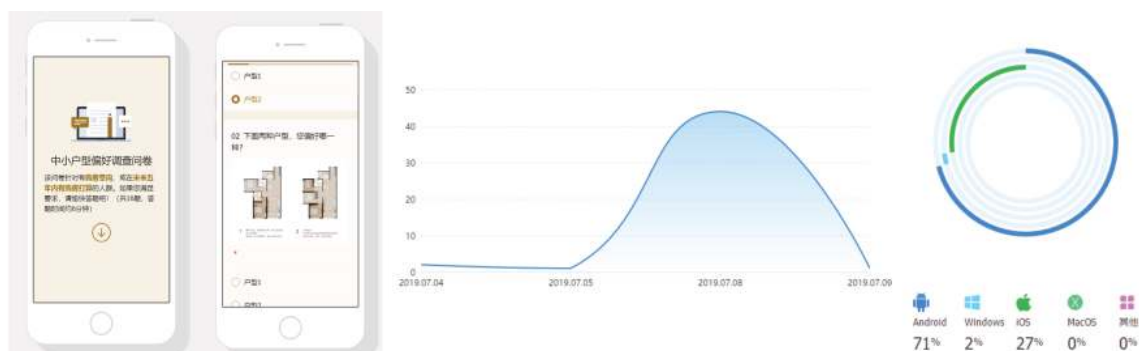


Figure 10.16 – Survey results. Around 50 people responded to the open call to complete the online survey with the visual questionnaire. Image credits: Meizi Ren.

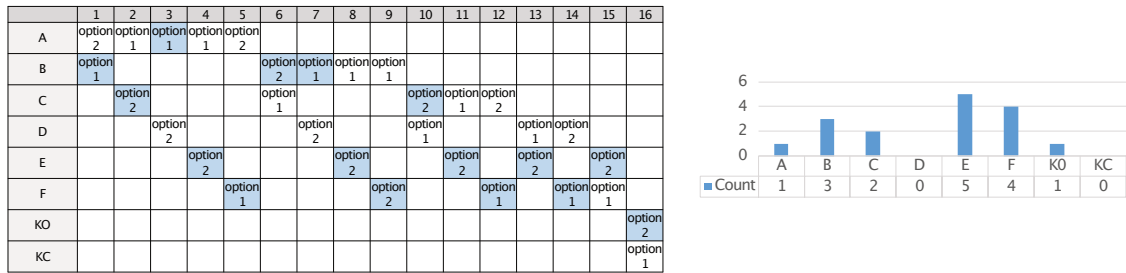


Figure 10.17 – Respondent histogram. Left: the matrix to map answers to attributes. Right: a histogram from the answers of participant Nr. 2 from the 50 who took the online survey. Image credits: Meizi Ren.

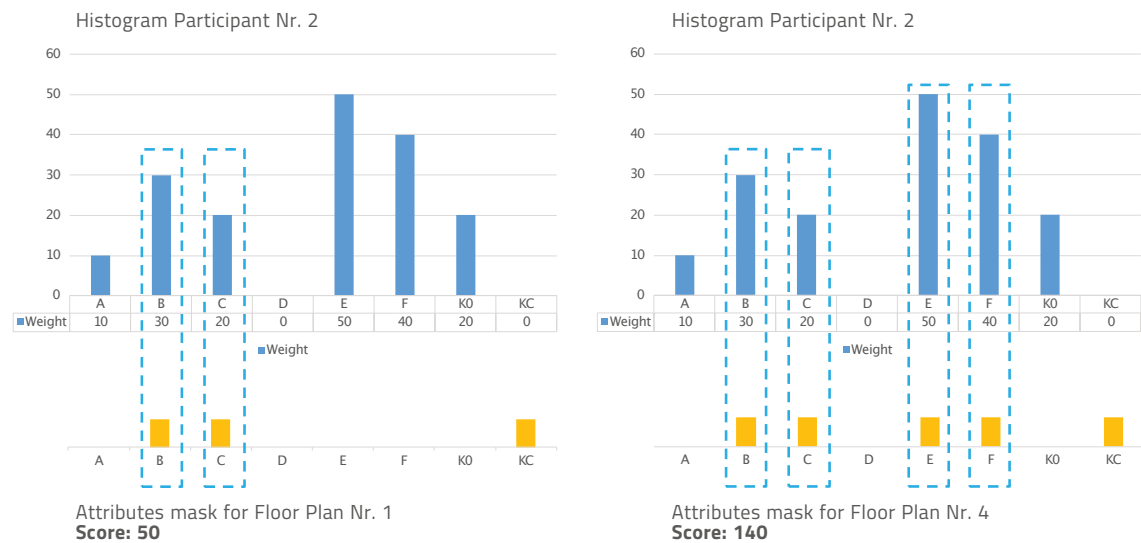


Figure 10.18 – Floor plan matching. Left: the attributes mask of floor plan 1 multiplied by the preference histogram of participant Nr. 2 gives a score of 50. Right: floor plan 4, with a score of 140, is a better match for the same user. Image credits: Meizi Ren.

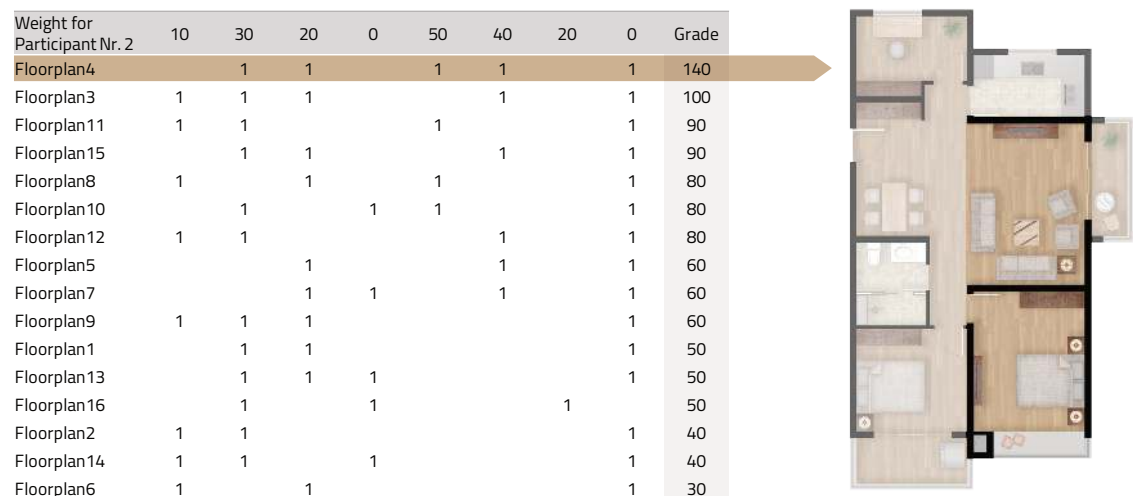


Figure 10.19 – Top matches. Left: the sorted list of floor plans according to their match with participant Nr.2. Right: the top choice, floor plan nr. 4. Image credits: Meizi Ren.

The case-based prototype is developed as a software product prototype by me in collaboration with Roger Winkler. The user pulls cards representing the wanted room onto a canvas (Figure 10.12). The relative size and position of the cards define the requirements of the architectural program. The requirements are program are represented as a graph (Figure 10.20). The search algorithm uses graph edit distance (Sanfeliu and Fu 1983) to sort the floor plans in the database and presents the closest matches to the user in near real-time. The user can review the matches and, if needed, add or change information to the card configuration on the canvas.

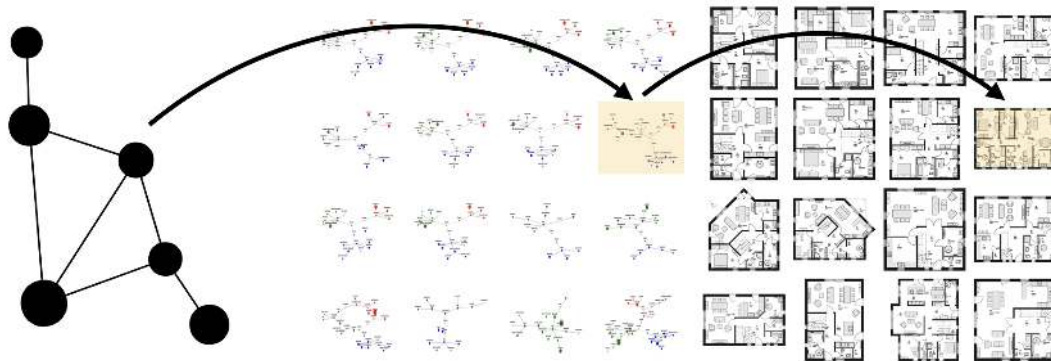


Figure 10.20 – Graph-similarity matching. In the case-based prototype for search of matching floor plans the user preferences are represented as a graph. Using graph edit distance as similarity measure the closest graphs of floor plans in the database are found and their graphical representation shown to the user. Image credits: the author.

Feedback from users and architects reveals that, of the three approaches to user input, the case-based approach is the closest to the current way the market for architectural services operates. It is also most suited to be used by a prospective inhabitant as the tasks for the users address rather pragmatic aspects of floor plan design. The approach is especially useful when the user's choice is limited to a finite set of floor plan designs.

Procedural 3D model generator

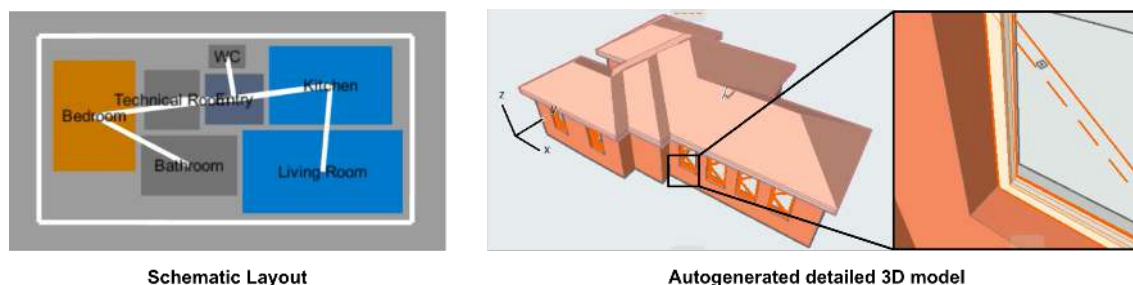
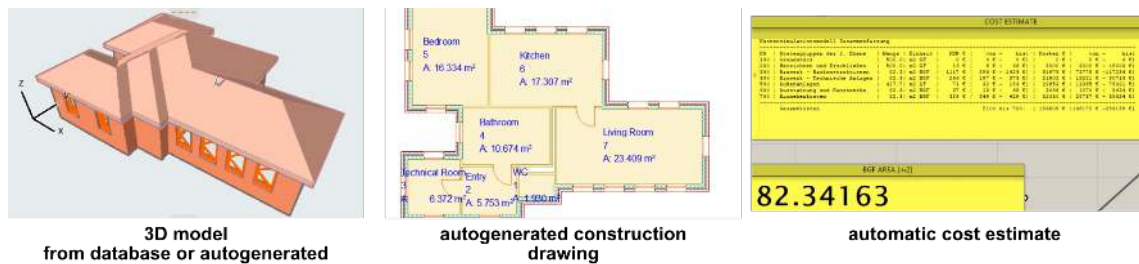


Figure 10.21 – Procedural 3D model generation. Image credits: the author.

I implemented a procedural 3D model generator in Grasshopper to produce a detailed 3D model from a schematic composition of rectangles (Figure 10.21). It automatically generates walls around the rectangles, positions doors based on the needed connection between rooms, places windows, generates the roof shape that fits the boundary, etc. Using an existing addon for McNeel Grasshopper to connect to Graphisoft ArchiCAD, I transform the 3D model into ArchiCAD objects such

as walls, doors, and windows to generate a Building Information Modelling (BIM) model. The BIM model is used for cost estimates (Figure 10.22).



10.3.2 Computational floor plan analysis

Feedback to the user based on automated computational analysis of the design solution is important in all approaches presented in this work. It is one of the means to encode expert knowledge and make it available to non-experts. However, I would argue that it is much more important in the approach used in *Rechteck2BIM* for two main reasons:

1. Expert stakeholders cannot iteratively fine-tune the generative system by encoding project-specific expert knowledge in it like in the vocabulary-based approaches of *20.000 BLOCKS* and *Project Reptiles*.
2. Non-experts edit an abstract representation of floor plan qualities and are presented with a design that is supposed to satisfy them. Their ability to judge whether this is true is relatively low, so they need the assistance of automated computational analysis to understand the consequences of the changes they make to the input over the outcome.

I have explored the integration of automated computational floor plan analysis in several student courses and my prototypes.

A Thousand Floor Plans

The course *A Thousand Floor Plans* looked at techniques to compare and sort thousands of variations of residential floor plans for the same design brief. The course explores apps and app making as a potential new task for the architect. The course took place at the DDU in TU Darmstadt in the winter term of 2016/2017. It was taught by me and the tutors Roger Winkler and Felix Dannecker.

Twenty architecture students worked in 8 teams. Each team of students was asked to sort a large set of floor plans based on one or more criteria such as views, energy, daylight, orientation, water, costs and time, spatial organization, etc. They developed a Grasshopper tool for architects to navigate the set of designs.

Students start with thousands of design options for the same house represented as floor plans. Which criteria can you use to sort them? Is there a *best* one? How can we empower an architect to choose one of the thousand options for their clients? And most importantly, how could we empower the homeowner to choose one design for their house?

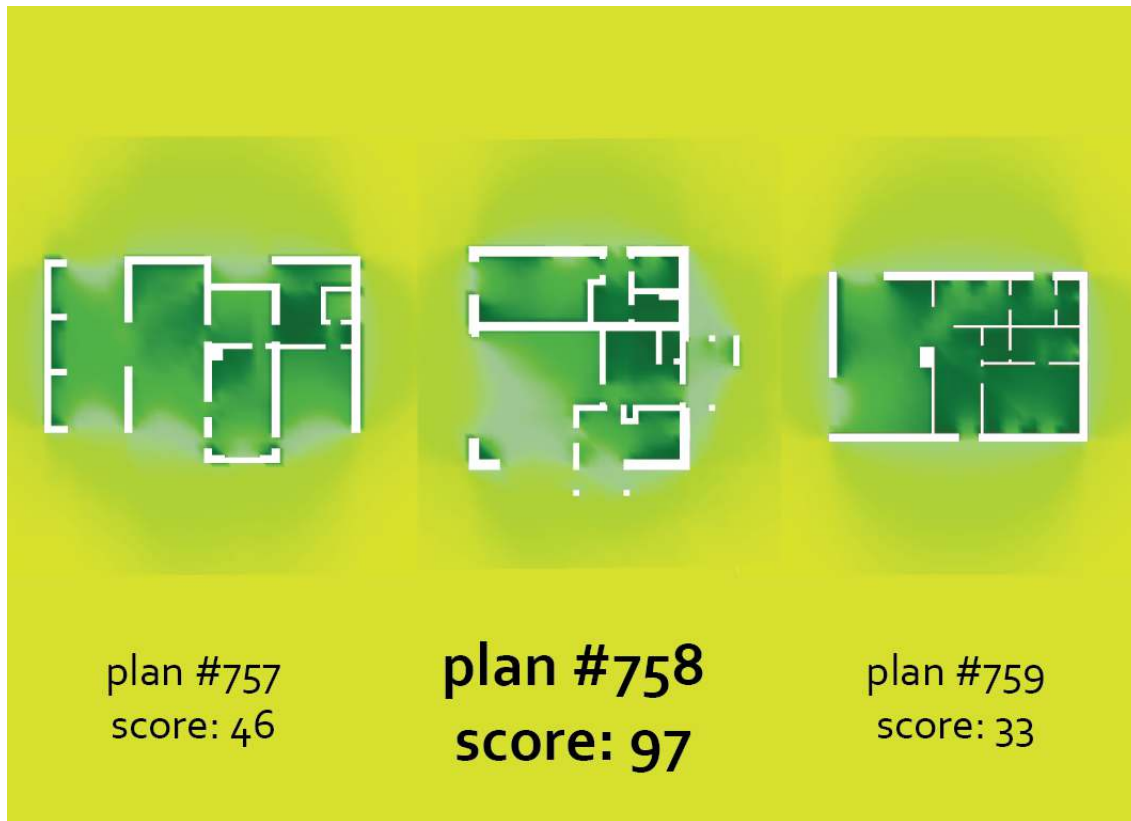


Figure 10.23 – Computational floor plan analysis. How would you choose from a Thousand Floor Plans? Image credits: the author.

Students also created a visual concept for a mobile app targeted at homeowners browsing through thousands of floor plans. The main challenge is to consider strategies for translating the numerical results of an analysis routine into experiences from daily life which the homeowners can relate to. For example *having a sunspot on the breakfast table with my morning coffee in the kitchen*.

The students produced the following six projects.

CHASELIGHT by Mariona Carrion, Morgane Hamel, and Louise Hamot. The app aims to empower homeowners to find the best floor plan where they can enjoy their daily activities, such as drinking coffee, with suitable levels of daylight (Figure 10.24). Daylight is measured in intensities (luxes), but how can you explain it to a non-expert? Different activities need different light ambiances. Users select the activity, and the app links this activity to a range of suitable light intensities. Users see a floor plan with every room painted in a different color. The darker ones are the rooms where it is less suitable to do the action and vice versa for the lighter ones. If they want to know more about a floor plan, they can click on one room and see exactly where they can carry out these activities and when. It's then in the user's hands to say if the floor plan suits them or not.

COSY HOME by Jörg Hartmann, Stefanie Joachim and Max Sand. The team used the Grasshopper add-on Honeybee to determine the thermal comfort of individual rooms in the floor plan (Figure 10.25). The parameters for analysis are the project's location, the energy consumption based on the façade structure, and

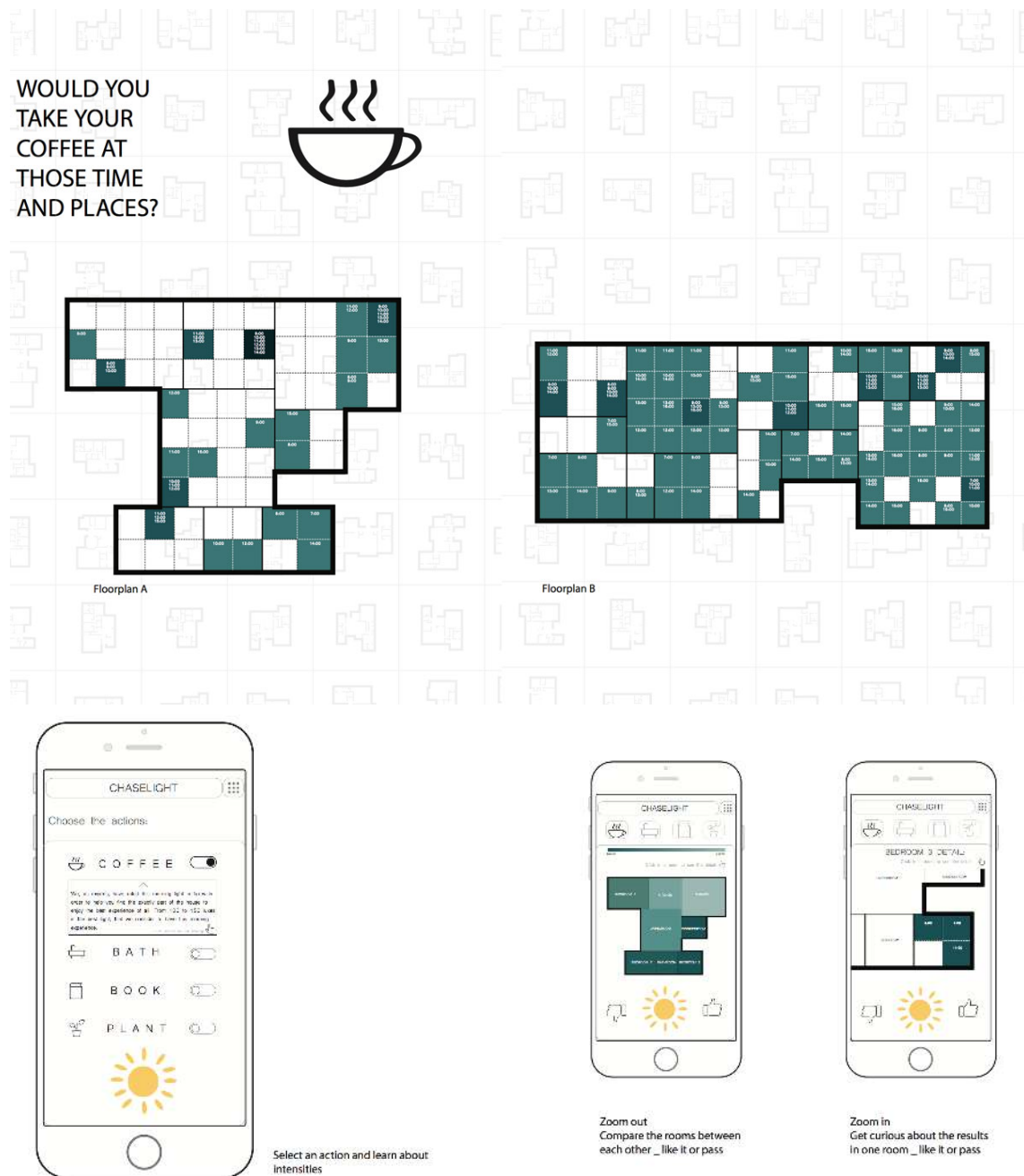


Figure 10.24 – CHASELIGHT App concept. Image credits: Carrion, Hamel, Hamot.

the façade opening to determine the comfort. The app concept uses the analysis procedure to show the three most suitable floor plans for the user. The results are displayed in color for comparison purposes.

ENERGETIC ASSESSMENT by Luisa Ruffertshöfer, Marc Ritz and Gerrit Walser. The students considered renewable energy as the main focus of contemporary sustainable assessments. The results from their survey confirmed this is one of the most critical aspects for inhabitants as well. With the **ENERGETIC ASSESSMENT** app, people can compare various floor plans on how much energy they could provide through solar systems in a whole year (Figure 10.26). The app calculates the kWh production per m² of roof surface for the heat and the electric gain through

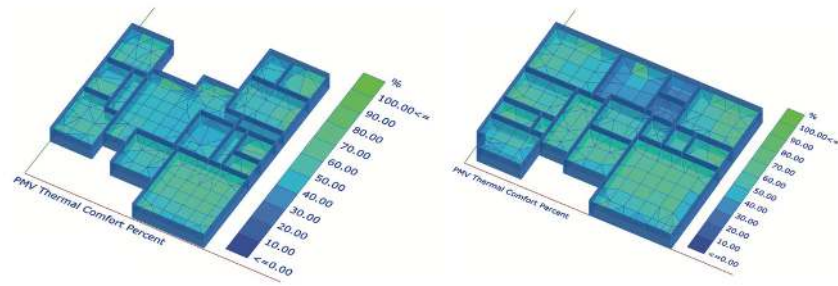


Figure 10.25 – COSY app concept. The app evaluates the thermal comfort of the rooms in a floor plan and presents the user with the top matches to their preferences. Illustrations by Jörg Hartmann, Stefanie Joachim and Max Sand. Image credits: Hartmann, Joachim, Sand.

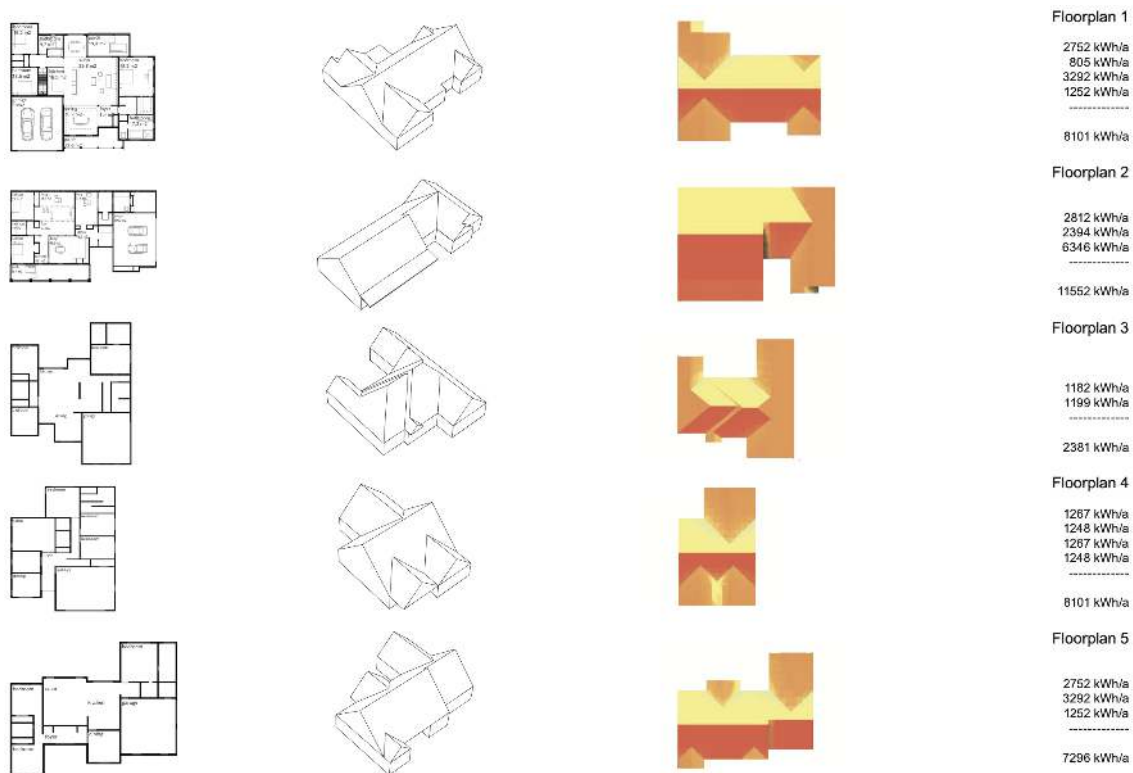


Figure 10.26 – ENERGETIC ASSESSMENT App logic. The app compares house designs based on the amount of solar energy they can farm, which depend on their location, orientation and roof shape. Image credits: Ruffertshöfer, Ritz, Walser.

a solar system. The Grasshopper add-on Ladybug handles these calculations and their technical aspects.

SAVING MONEY by Ana Sophie Sánchez Wurm and Ana Baraibar Jiménez. The students decided to compare floor plans based on the household's consumption of electricity (Figure 10.27). Expenses, as well as potential gain from the rooftop solar panels, are taken into account. The results depend on the number and ages

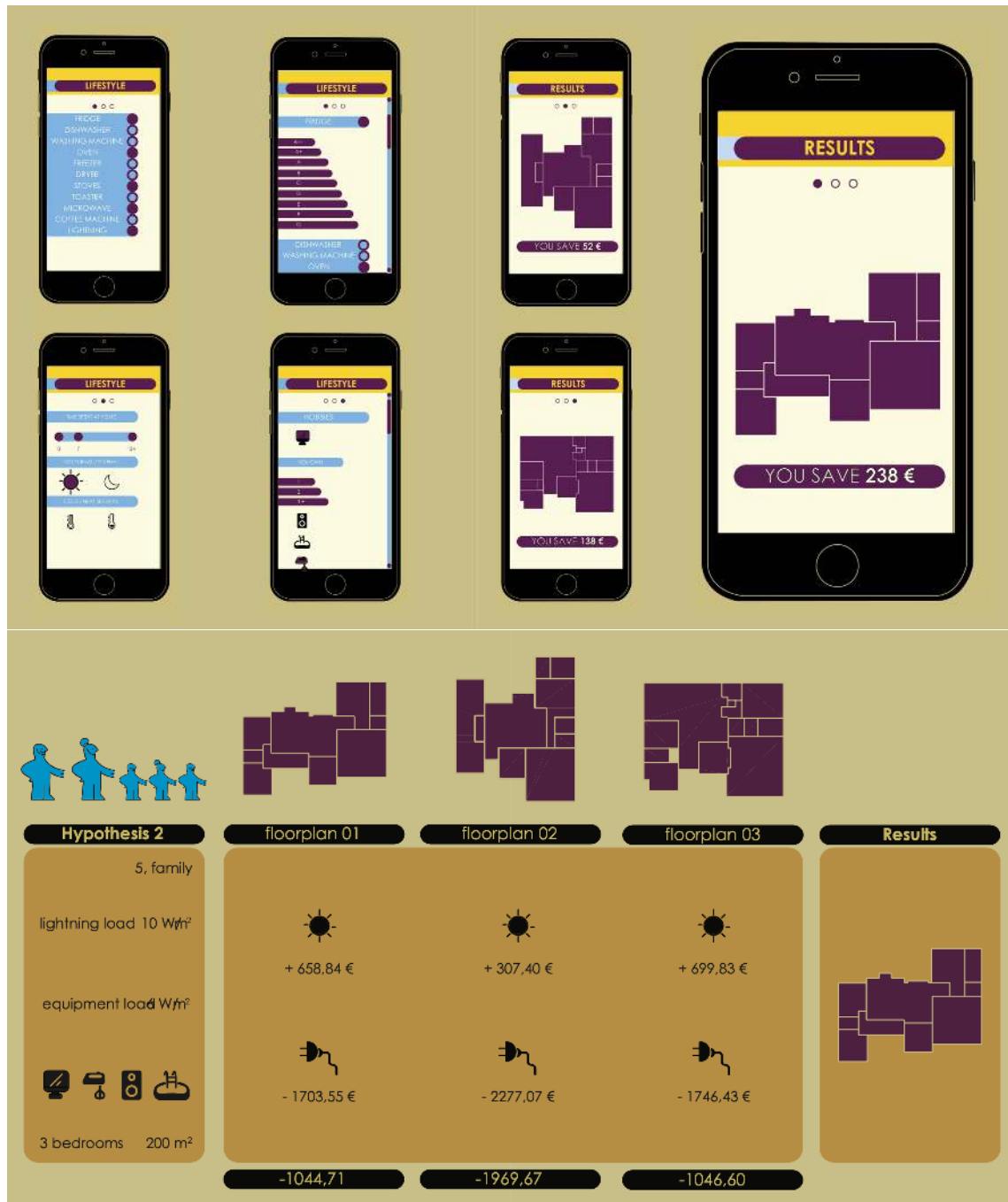


Figure 10.27 – SAVING MONEY app concept. Image credits: Wurm, Jiménez.

of inhabitants, the location, the area of the house, and its orientation. The users define the appliances and their time at home to determine artificial lighting and heating/cooling needs. The floor plan alternatives are presented rated by the amount of savings compared to the current electricity bill of the user.

COSTINATOR by Nicole Klumb and Maximilian Pfaff The authors found out that the building costs for a house are of high importance for the homeowners through a survey. They create an app concept that compares floor plans based on the costs of construction (Figure 10.28). To keep the app as user-friendly as possible, it only calculates the construction costs for the shell. The estimate is

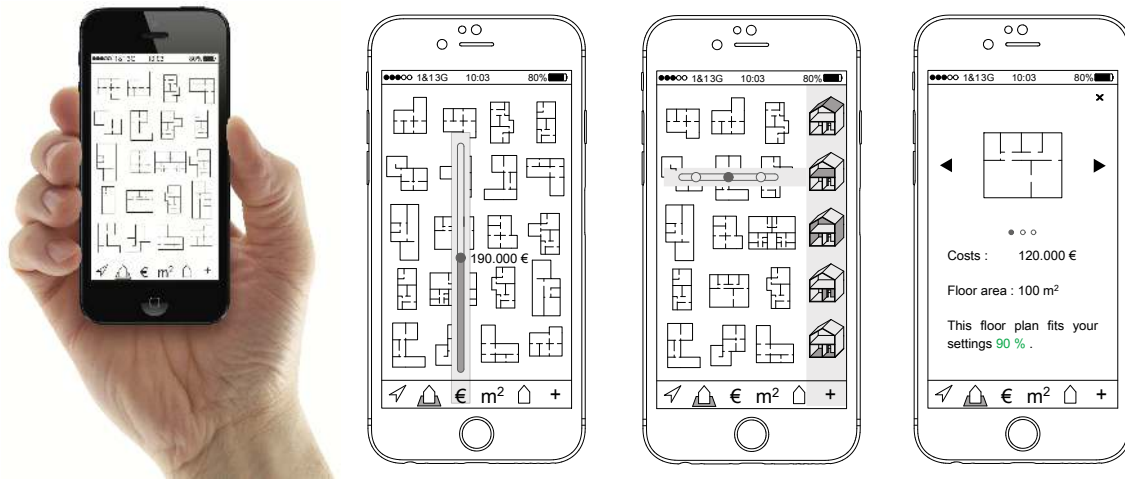


Figure 10.28 – COSTINATOR app concept. It uses the Baukostenindex (German for Construction Costs Index) to help the user find a floor plan based on their preference for budget, area, and construction quality. Image credits: Klumb, Pfaff.

based on a model used by the German Baukostenindex (BKI) with defines three quality levels of construction (low, medium, and high) for the five main element categories (foundation, outer walls, slabs, roof, and inner walls). The user can filter and sort the alternative designs based on the budget and total area settings.

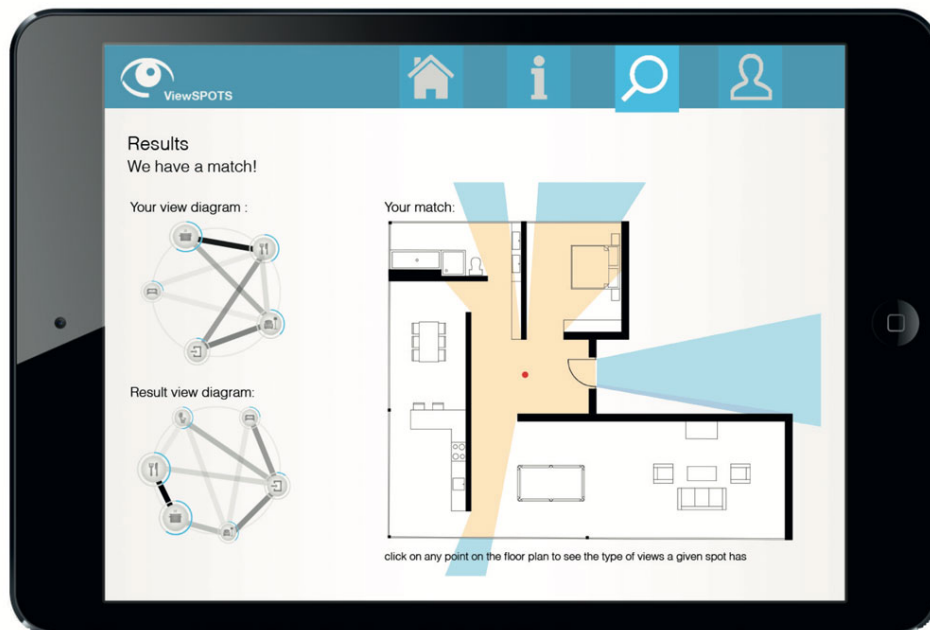
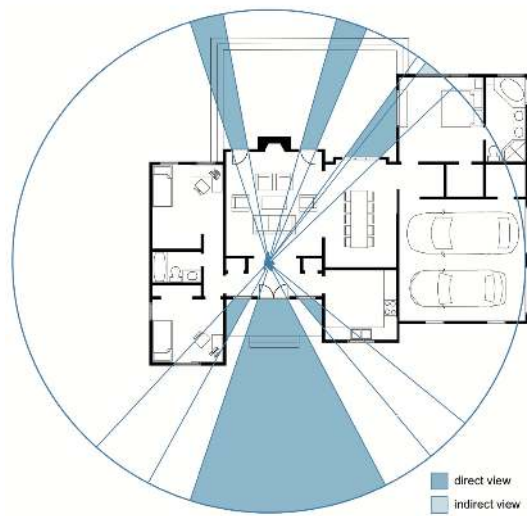


Figure 10.29 – VIEWSPOTS app concept. The app uses the isovist analysis to form a visual connectivity footprint for the user and each floor plan and match them. Image credits: Helbig, Vehrenberg.

VIEWSPOTS by Anjuscha Helbig and Philipp Vehrenberg. ViewSPOTS is an app that helps people find a floor plan based on their view-related preferences (Figure 10.29). It analyzes the view connections to the outside and the visual connections between the different rooms in the inside of the house (Figure 10.30). A grid

Outgoing views: How much can be seen from the inside out?



Interior view connections: what rooms are visually connected? How much of those rooms can be seen?



Figure 10.30 – VIEWSPOTS logic. The app determines the direct and indirect internal and external views from every spot in the houses. Image credits: Helbig, Vehrenberg.

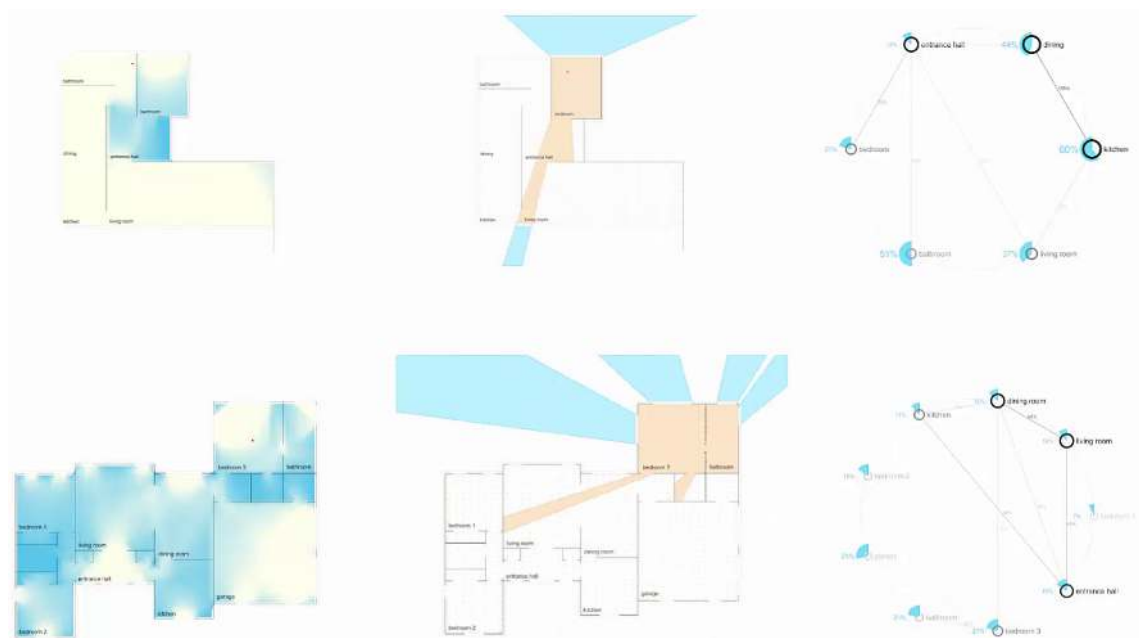


Figure 10.31 – Visual connectivity footprint. The visual connectivity footprint in VIEWSPOTS captures the qualitative differences between open plans (top) and plans with separate rooms (bottom). Image credits: Helbig, Vehrenberg.

of many points analyzes the entire floor plan and creates a visual connectivity footprint (Figure 10.31). The diagram displays the percentage of internal views among the rooms and the external views of each room separately. A heat map is generated from the exterior views, giving an overall impression of the number of building envelope openings. The visual connectivity footprint captures the difference in spatial quality between open-plan houses and houses with separate rooms. Each person has different needs and requirements for their own living spaces. The app helps users find a floor plan that best suits their needs based on their personal view connectivity footprint.

Cultural influences on floor plans

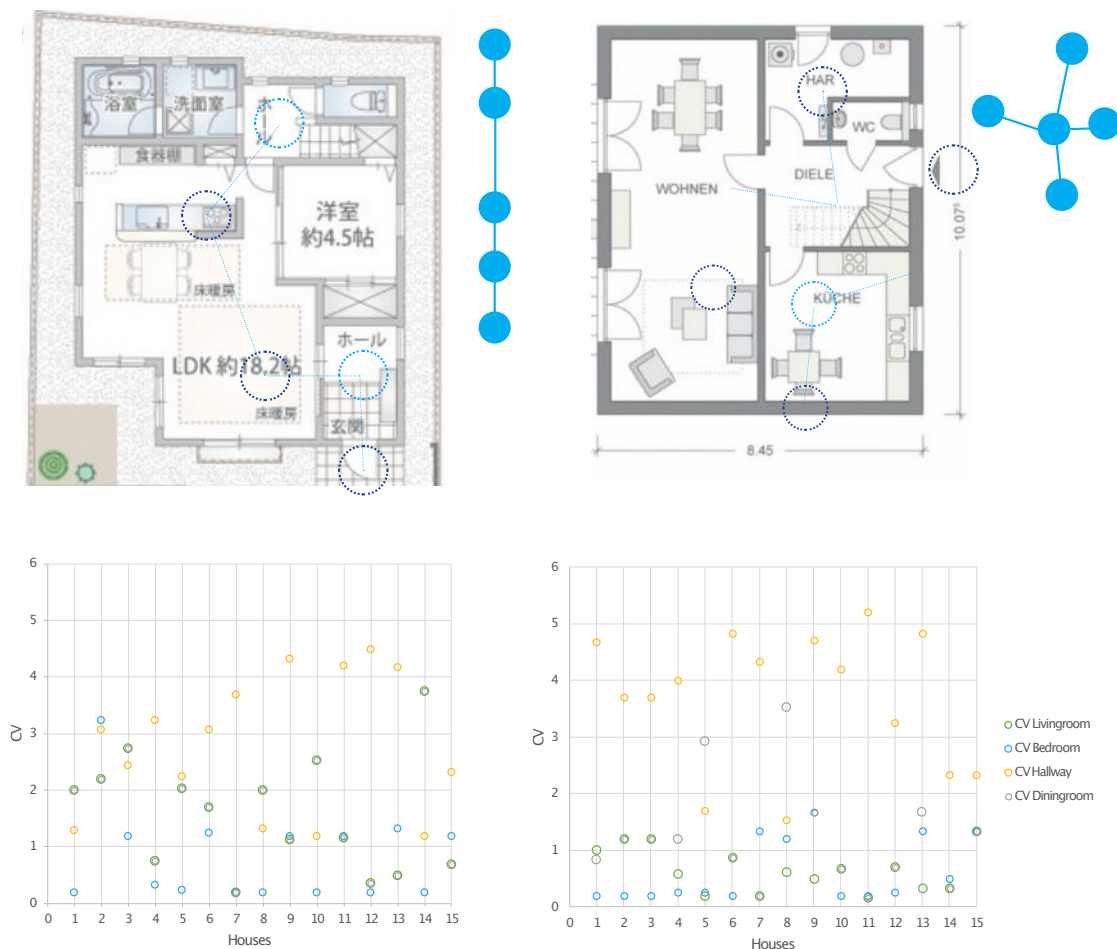


Figure 10.32 – Control value (CV) justified graph analysis. The CV parameter represents how isolated a room is. Lower value means more isolated. The clear separation between the CV value of the hallway in German houses (right) means there is a star-like organization of the spaces. At the same time the smooth blend of CV parameters of all rooms in Japanese houses (left) shows that the organization is more linear. Image credits: Rui Zhi.

The layout of spaces hosting the various daily and nightly activities and the connectivity between depends on the individual lifestyle of the inhabitants. However, it also has a cultural bias. Considering the topological differences can assist in generating and searching for matching floor plans.

The student Rui Zhi used space syntax analysis to compare the floor plans of

15 German (Figure 10.34) and 15 Japanese freestanding single-family houses (Figure 10.35) with area between 100 and 150 m². For each house, Rui Zhi calculated total depth (TD), mean depth (MD), real relative asymmetry (RRA), control value (CV), and the level distance of a single room. The parameters are calculated from the justified graph used in Space Syntax (Hillier 2007). The calculation method is described by Ostwald and Dawes 2018.

The comparison shows that Japanese houses are more linear in their organization than German ones, which are more star-like, with the hallway acting as a connection hub. This can be seen when comparing the CV parameter for the individual rooms in both groups (Figure 10.32).

The analysis also cross-referenced the CV values with the RRA parameter, a measure of depth (Figure 10.33). The results show the rooms in German houses are relatively polarized, especially the isolated bedrooms and the shallow hallways right beside the entrance doors. This shows a clear functional division. On the other hand, in Japanese houses, rooms are blended without apparent single-purpose zones. At the same time, the main rooms in Japan have a higher CV, which means they are more accessible and pleasantly designed despite the functional overlap.

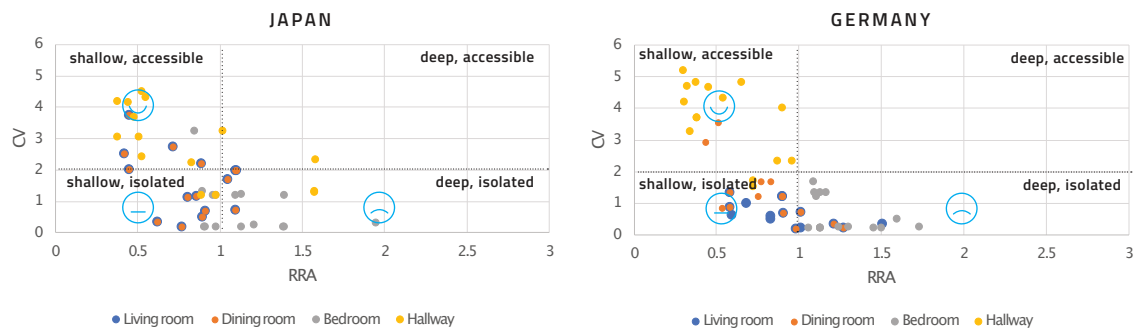
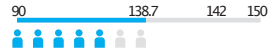


Figure 10.33 – Real relative asymmetry (RRA) analysis. Image credits: Rui Zhi.

INFORMATION

Total Floor Area
Number of Residents



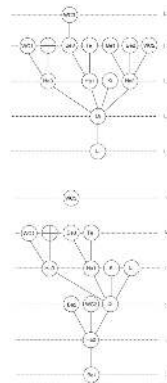
Basement



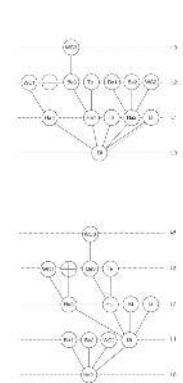
First floor



Second floor



Justified graphs



Persons	Total Floor Area (m²)	Total Floor Area Per Person (m²)	Living room Per Person (m²)	Dining room Per Person (m²)	Kitchen Per Person (m²)	Bedroom Per Person (m²)	Bathroom Per Person (m²)	others Per Person (m²)	Daytime Area Per Person (m²)	Nighttime Area Per Person (m²)
5	138.7	27.74	4.51	2.54	1.50	11.00	3.13	5.06	8.55	11.00

Functional configuration

Carrier	Nodes at each level						Depth			
	level 0	Level 1	Level 2	Level 3	Level 4	Level 5	TD	MD	RRA	CV
Living room	1	1	4	7	1		34	2.62	1.01	0.20
Dining room	1	5	7	1			22	1.69	0.43	2.92
Bedroom	1	1	3	4	4	1	60	3.08	1.30	0.25
Hallway1	1	3	5	5			28	2.15	0.72	1.70

Spatial characteristics

Figure 10.34 – German house. Image credits: Rui Zhi.

INFORMATION

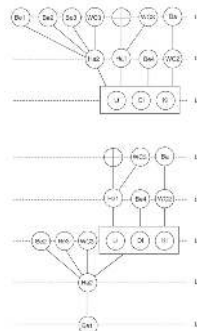
Total Floor Area
Number of Residents



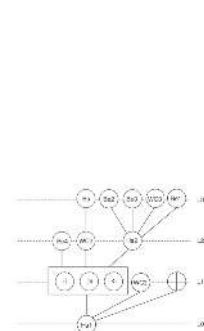
First floor



Second floor



Justified graphs



Persons	Total Floor Area (m²)	Total Floor Area Per Person (m²)	Living room Per Person (m²)	Dining room Per Person (m²)	Kitchen Per Person (m²)	Bedroom Per Person (m²)	Bathroom Per Person (m²)	others Per Person (m²)	Daytime Area Per Person (m²)	Nighttime Area Per Person (m²)
4	142.42	35.61	3.66	2.80	3.61	12.80	4.61	8.14	10.06	12.80

Functional configuration

Carrier	Nodes at each level						Depth			
	level 0	Level 1	Level 2	Level 3	Level 4		TD	MD	RRA	CV
Living room	1	4	7				18	1.64	0.45	2.03
Dining room	1	4	7				18	1.64	0.45	2.03
Bedroom	1	1	4	3	3		30	2.73	1.21	0.25
Hallway1	1	3	3	5			24	2.18	0.83	2.25

Spatial characteristics

Figure 10.35 – Japanese house. Image credits: Rui Zhi.

10.4 Human Agency

The case study explores the program-first type of tasks for the users. That is that users create and guide a design towards a wanted state by defining the components of its architectural program. The users cannot directly edit the geometry of the architectural shapes.

The design paradigm of *Rechteck2BIM* suggests that stakeholders, both non-expert and expert, i.e., inhabitants and architects, perform two main tasks:

1. put together home layouts from programmatic units that represent the rooms in a single-family house and
2. browse and select floor plans ranked based on a specific performance indicator such as energy consumption, solar exposure, etc.

It is important to understand which performance indicators are most important for the non-expert stakeholders, i.e., homeowners. Therefore the students creating the prototypes for computational analysis were asked to survey homeowners to narrow down the focus on the essential aspects to analyze (Figure 10.36). The results showed that (Figure 10.37).

WIE WÄHLST DU UNTER TAUSENDEN GRUNDRISSSEN AUS?

Stell dir vor, Du willst ein neues Haus für Dich und Deine Familie bauen. Das Grundstück ist bereits gekauft und momentan suchst Du nach einem Entwurf, der Deinen Vorstellungen entspricht. Nun findest Du eine App, die Dir Tausende von vorgefertigten Grundrissen zur Auswahl stellt.

Welche 7 Kriterien würdest Du wählen, um eine Auswahl zu erhalten, die Deinen Vorstellungen entspricht?

Angaben zur Person:

Alter:

☐ 0 - 17

☐ 18 - 24

☒ 25 - 34

☐ 35 -

☐ in Rente / Pension

Momentane Lebenssituation (mehrfach möglich):

Antworten möglich:

☐ Alleinstehend

☒ mit Partner

☐ mit Kindern

☐ mit Eltern

☐ andere: _____

Wohnungssituation (mehrfach möglich):

Antworten möglich:

☐ Ich besitze ein Haus/Wohnung.

☒ Ich wohne zur Miete.

☐ Im Elternhaus.

☐ Ich vermiete.

☐ Ich möchte in den nächsten 1-2 Jahren Eigentümer werden.

Werte insgesamt 7 Kriterien aus der folgenden Liste aus. Die Aufteilung der Themenschwerpunkte spielt hierbei keine Rolle. Schreibe die jeweiligen Nummern in die Liste des Smartphones. Sortiere die Kriterien nach Wichtigkeit von oben nach unten.

KRITERIEN ZUR SORTIERUNG

BLICKBEZUG:

☐ 1. Wie viel Himmel ich durch das Fenster sehen kann.

☐ 2. Wie viel Grün und Bäume ich sehen kann, wenn ich im Haus bin.

☐ 3. Wie viel ich von den Nachbarhäusern aus dem Haus sehe.

☒ 4. Wie viel ich vom Horizont sehe, wenn ich im Haus bin.

☐ 5. Die Wahrscheinlichkeit, dass jemand von außen in mein Haus schaut.

ENERGIE

☐ 6. Die Energie, die das Haus über ein Jahr verbraucht.

☐ 7. Der erwartete Energieverlust durch die Fassade.

☒ 8. Die Energie, die ich durch Geräte und Anlagen zur aktiven Reduzierung des Energieverbrauchs im Haus einsparen kann (Solar Kollektoren, Heiz- und Kühlgeräte, automatisierte Fensterläden usw.).

☐ 9. Wie hoch die zusätzlichen Kosten für Geräte und Anlagen sind, die den Energieverbrauch des Hauses aktiv reduzieren.

TAGESLICHT

☐ 10. Wie viel Tageslicht über das Jahr in das gesamte Haus fällt.

☐ 11. Wie viel Tageslicht über das Jahr in das Schlafzimmer fällt.

☒ 12. Wie viel Tageslicht über das Jahr in das Wohnzimmer fällt.

☐ 13. Wie viel Tageslicht über das Jahr in das Badezimmer fällt.

☐ 14. Wie viel Tageslicht über das Jahr in den Garten fällt.

☐ 15. Wie viel Tageslicht über das Jahr in die Küche fällt.

☐ 16. Die Tageszeit, zu der das meiste Sonnenlicht in das Schlafzimmer fällt.

☒ 17. Die Tageszeit, zu der das meiste Sonnenlicht in das Wohnzimmer fällt.

☐ 18. Die Tageszeit, zu der das meiste Sonnenlicht in das Badezimmer fällt.

☐ 19. Die Tageszeit, zu der das meiste Sonnenlicht in den Garten fällt.

☐ 20. Die Tageszeit, zu der das meiste Sonnenlicht in die Küche fällt.

AUSRICHTUNG

☐ 21. Der Winkel zwischen Süden und der Ausrichtung des Schlafzimmers.

☒ 22. Der Winkel zwischen Süden und der Ausrichtung des Wohnzimmers.

☐ 23. Der Winkel zwischen Süden und der Ausrichtung des Badezimmers.

☐ 24. Der Winkel zwischen Süden und der Ausrichtung des Gartens.

☐ 25. Der Winkel zwischen Süden und der Ausrichtung der Küche.

WASSER

☐ 26. Die Menge Regenwasser, die ich wiederverwenden kann.

☐ 27. Die Menge Nutzwasser, die ich aufbereiten und wiederverwenden kann.

☐ 28. Die erwartete durchschnittliche Menge Nutzwasser, die im Haushalt über ein Jahr verbraucht wird.

KOSTEN

☒ 29. Die Rohbaukosten und -materialien (Stein, Holz, Beton usw.).

☐ 30. Die Kosten für den Ausbau (Fenster, Türen, Fassade, Putz usw.).

ANDERE

☒ 31. Die erwartete Bauzeit.

☐ 32. Der Anteil Land, der nach dem Bau des Hauses zur Nutzung übrig bleibt.

☐ 33. Die Anzahl der Geschosse / Split-Level

DIE KRITERIEN, DIE WIR VERGESSEN HABEN

☐ 34. _____

☐ 35. _____

☐ 36. _____

☐ 37. _____

☐ 38. _____

Wähle 7 Kriterien

Grundriss Browser

Ungewiss 1:33.000.000 Bayern/DDR

Kriterien zur Sortierung

1. ☐ 8

2. ☐ 31

3. ☐ 22

4. ☐ 12

5. ☐ 17

6. ☐ 4

7. ☐ 29

Figure 10.36 – The survey in 1000 Plans. Image credits: Helbig, Vehrenberg.

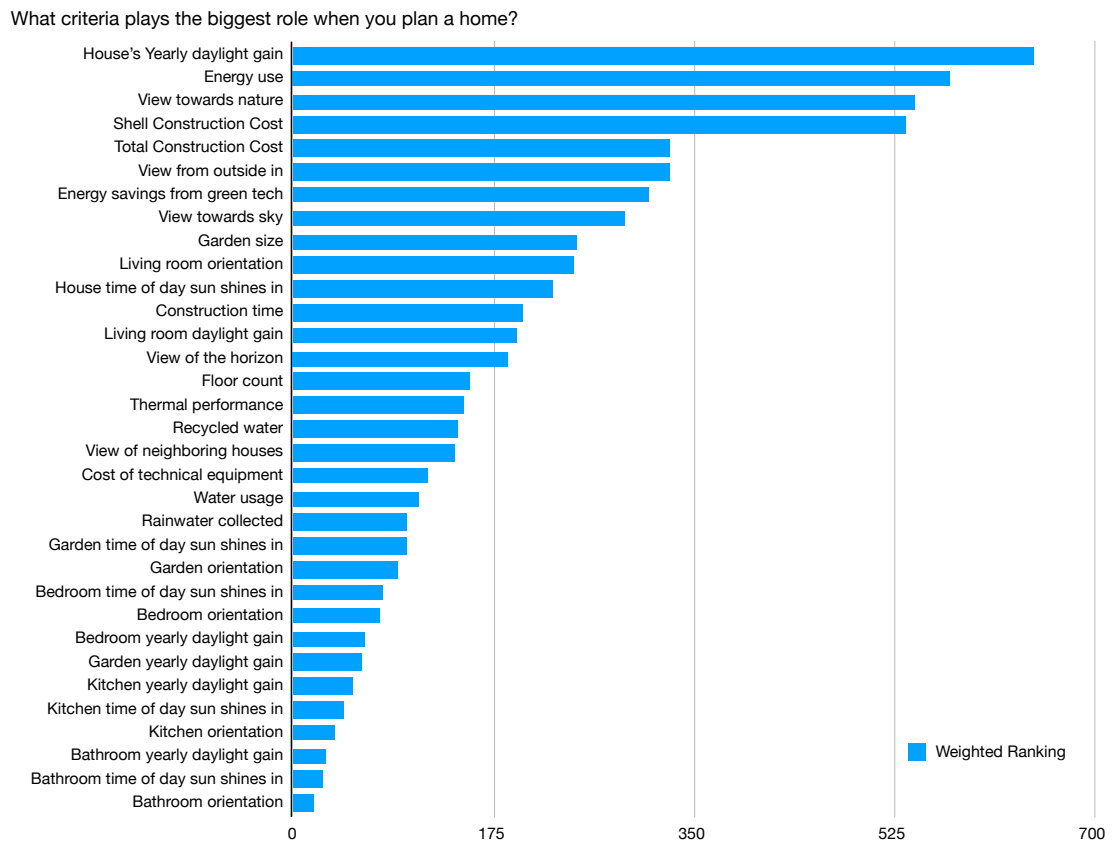


Figure 10.37 – Survey results. 234 respondents. Image credits: the author.

10.5 Take-aways

The primary source of findings is my observations of people interacting with the prototypes and conversations with industry experts (architects and prefab housing manufacturers) and non-experts (representatives of the inhabitants' group and people who played the game prototypes).

Potential use case scenarios for the technology illustrated with the prototypes described above are:

1. Crowdsourcing floor plans for a given brief to the crowd. Architects, designing for a client, use the search functionality to browse submitted designs.
2. Configuring one's own house.
3. Configure one's own house together with an architect.

Architects feel more constrained because the designs are too defined. In conversations with architects, it becomes clear that they find problematic two things:

1. showing one or more floor plan layout alternatives to the inhabitant before they have come in conversation with the architect.
2. having to choose and adapt a ready floor plan layout for their clients instead of gradually deriving it from the client's and site's constraints.

Furthermore, unlike in the approaches from *20.000 BLOCKS* and *Project Reptiles*, which worked with a vocabulary of shapes as input for the algorithms, here the generative technique is based on computational geometry and is not adjustable through geometric modeling but coding. Non-expert stakeholders, such as inhabitants and the crowd, can more easily define the initial input with a set of preferences. However, they have more difficulties editing the result iteratively as the mapping between the programmatic input, and the procedurally generated 3D model is too opaque/abstract.

An open question that arises here is: How much info can we get from the homeowner, and what will be the variety of designs? Will that be limitless or very limited?

The need for extensive feedback on the house's performance is close to the pragmatism that drives non-experts when building their new home. Future homeowners can better understand what they want and adapt their preferences from their future homes if provided with feedback as they explore options for layouts.

Chapter 11

Discussion

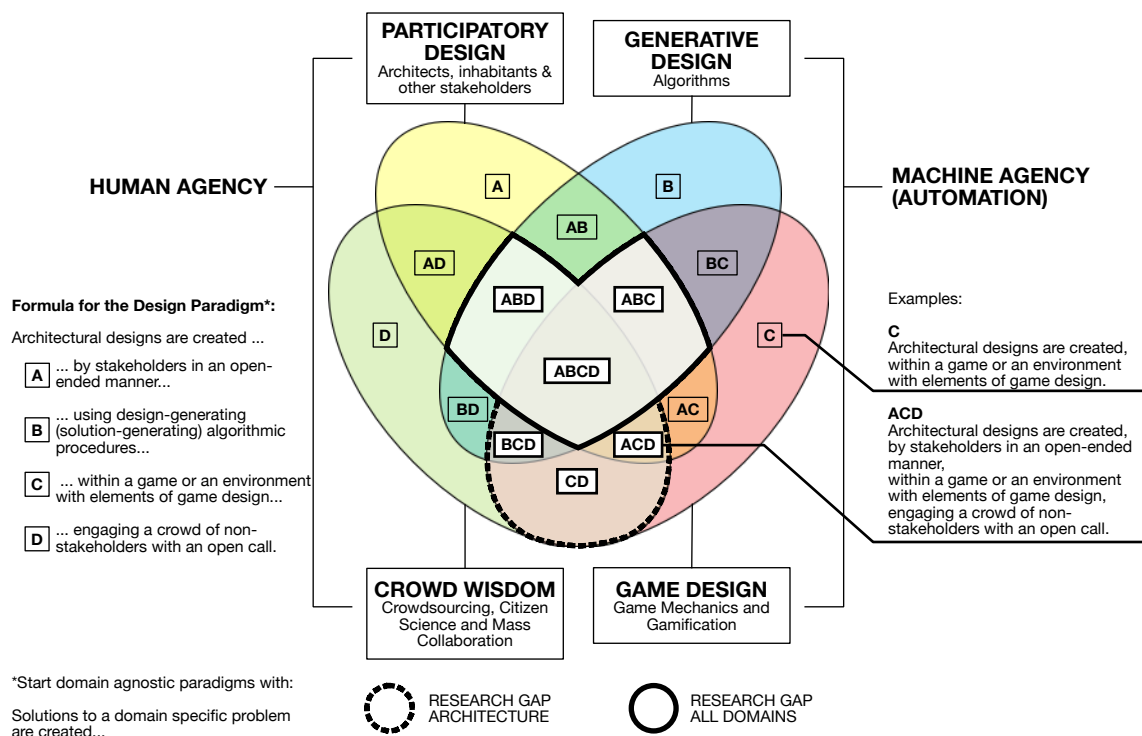


Figure 11.1 – Repeat of Figure 6.2. The Map of Design Paradigms shows the design paradigms at the intersections of the four fields, their descriptions and the identified research gap. Image credits: the author.

I began this work with the thesis that:

Architectural design knowledge can be encoded into generative game worlds where every role — architects, stakeholders, and third-party participants — can contribute, respective to their skills and interests, to creating schematic architectural designs.

The thesis is explored in two research questions at the intersection of *Participatory Design*, *Generative Design*, *Game Design*, and *Crowd Wisdom*. The first research question aims to unpack the possible frameworks or constructs under which this mass contribution occurs. The four case studies are used to explore the *design paradigms* that lie at the intersection of participatory design, generative design,

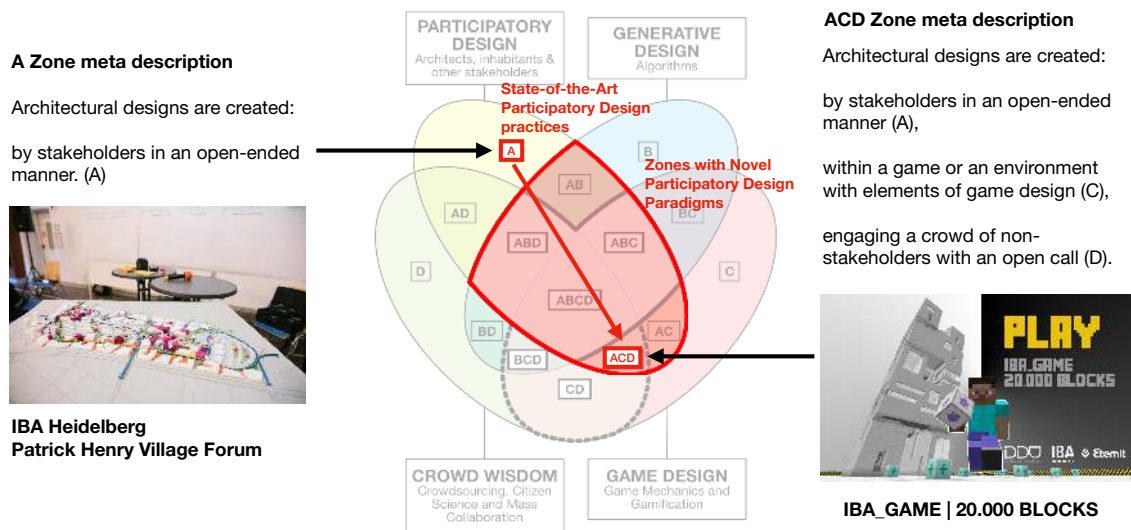


Figure 11.2 – Operational use of the design paradigms map. The *IBA_GAME* example illustrates how the map allows to systematically explore participatory design paradigms outside of the zone of today's state-of-the-art practices. The meta description of the zones serves to specify the modes of engagement of human and machine agency. Image credits: the author.

crowdsourcing, and games. The second research question calls for prototyping of the *roles, tools, and tasks* that can be offered at this intersection for the creative involvement of the various groups — architects, stakeholders, third-party contributors.

11.1 Design paradigms

In response to the first research question, I defined the design paradigms in chapter 6 by systematically intersecting the essential feature of each of the four fields. As Figure 11.1 shows, each design paradigm is the result of a formulaic combination of the properties of the four fields. Within this meta-definition, many possible design processes exist.

To illustrate how this map can be used, I want to bring attention to the main difference between participatory design as it is practiced today and the paradigms lying in these new territories (Figure 11.2). The *IBA_GAME*, developed as part of the *20.000 BLOCKS* case study, provides the opportunity for a direct comparison between the two approaches.

In the case of *IBA Heidelberg's* participatory process for the Patrick Henry Village, a stellar set of architects was commissioned — KCAP, MVRDV, Carlo Ratti Associati, ASTOC, and Ramboll Liveable Cities Lab. After three citizen forums, they produced four visions and one dynamic master plan for the new city quarters. In a participatory design, the stakeholders debate on a list of wants that the architects then process into a design. In this collaborative brief-writing, it is difficult for the non-expert stakeholders to visualize and understand the trade-offs when we satisfy one or another preference. The decision for making the trade-offs lies with the architect. As such, the *authoritative* strategy for dealing with wicked problems

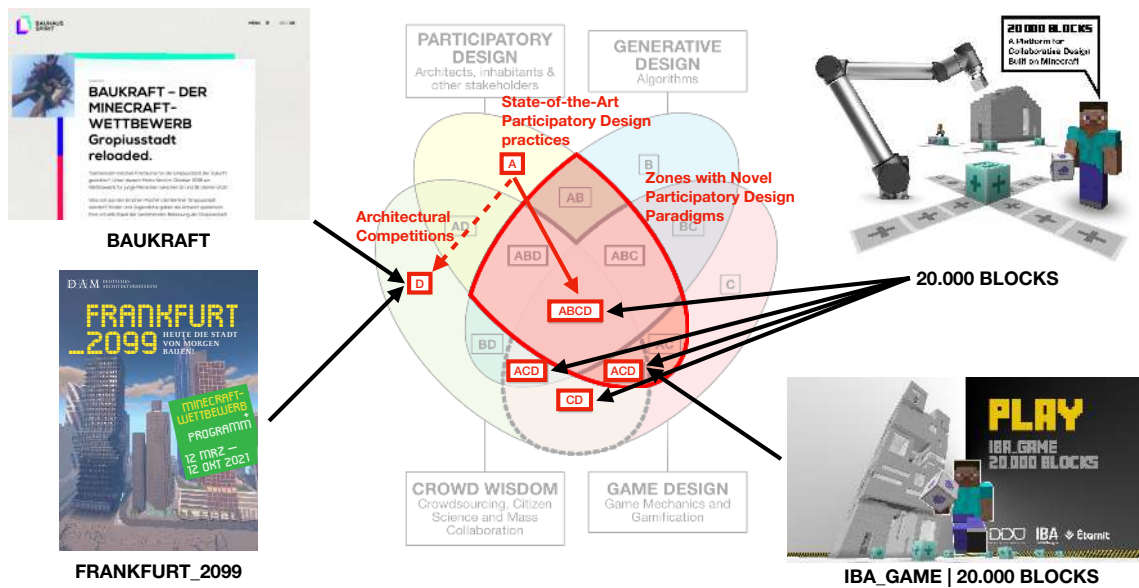


Figure 11.3 – Distinction between popular uses of Minecraft in Architecture and the 20,000 BLOCKS. *Baukraft* (Bauhaus Spirit 2016) and *Frankfurt_2099* (Deutsches Architekturmuseum (DAM) 2021) are examples how game worlds are used merely as widespread 3D modeling software. In contrast, *IBA_GAME*, and all projects in the *20,000 BLOCKS* case study, make use of game design, generative design and engage the stakeholders in a conversation or collaboration instead of in a competition mode. Image credits: the author.

(See) is leading here.

A *collaborative*, or a *competitive* strategy, entails that the power, i.e. the creative agency, is dispersed and often contested (Roberts 2001). To systematically explore these strategies we can refer to the map on Figure 11.2 which provides a meta description of the tools that populate each intersection. The game we developed for *IBA Heidelberg*, *IBA_GAME*, follows this specification. In *IBA_GAME*, players of all ages created neighborhoods in the new Patrick Henry Village by placing buildings and public spaces. They could see how parameters such as height, program diversity, and density interplay with one another. All player-created neighborhoods are accessible online for browsing. When we engage everyone through a game, the trade-offs of various parameters can be made tangible so that stakeholders understand the problem better.

Figure 11.3 shows one more way to use the Design Paradigm Maps. We can clearly see that common uses of Minecraft in architecture are not using the full potential that games and generative design offer. In *Baukraft*, a competition for the redesign of Gropiusstadt in Berlin (Bauhaus Spirit 2016) and *Frankfurt_2099* (Deutsches Architekturmuseum (DAM) 2021), a competition for visionary urban scenarios for the city of Frankfurt, Minecraft is not used as a game but simply as a 3D modeling software that many young people know how to use. Each of the competitions provides a Minecraft map for the participants to download from a website. They can then use Minecraft in Creative mode, i.e., as a 3D environment to build with unlimited resources. No means of automated or human feedback is provided to participants. Neither is there any machine agency such as a generative algorithm or game mechanics. Both *Baukraft* and *Frankfurt_2099* are organized as a conventional architectural competition. As discussed in chapter 4, architectural

competitions are in the pure Crowdsourcing field (D).

As a result of the awareness that the Map of Design Paradigms brought upon, in *IBA_GAME*, I tested the use of Game Mechanics and Generative algorithms. Therefore, in *20.000 BLOCKS*, participants do not model their ideas block by block but can also use predefined modules that they combine to achieve a specific game goal.

11.2 Tools, tasks and roles

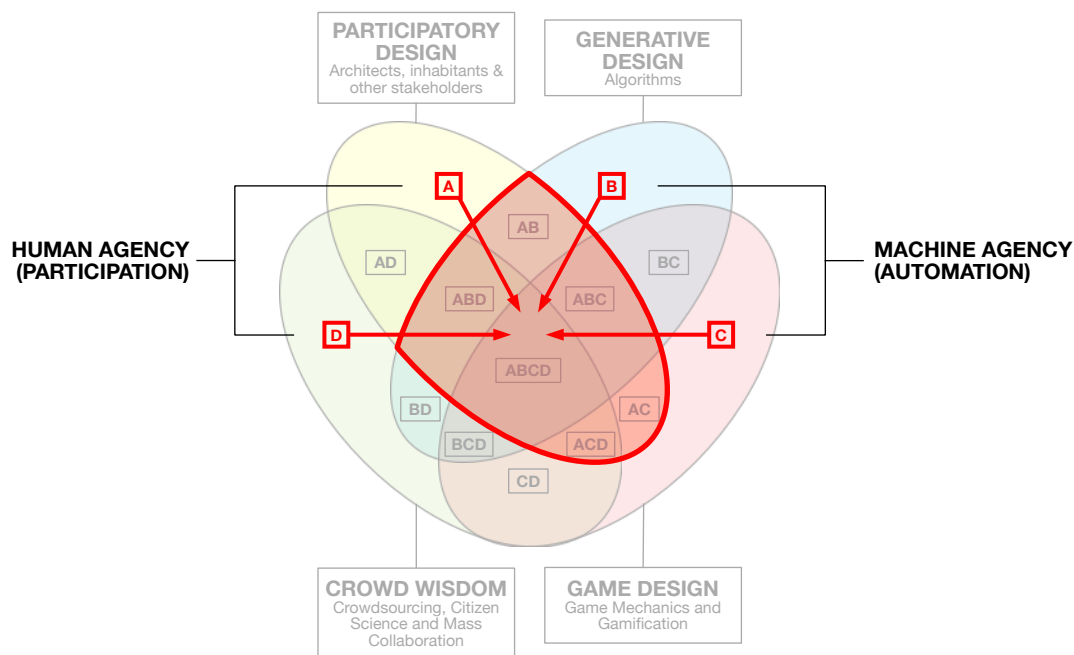


Figure 11.4 – New zones of human-machine interaction. Image credits: the author.

The second research question explores tools, tasks, and roles. In effect, it is concerned with the human agency in a given design paradigm. Human agency is represented by the two fields on the left of Figure 11.1 — *Participatory Design* and *Crowd Wisdom*. At the same time, Machine Agency is represented by the two fields on the right — *Generative Design* and *Game Design*.

Exploring the design paradigms lying at the central zones of the Map on Figure 11.4 requires the staging of human-machine interaction in novel ways. The case studies and the literature review revealed that finding the right balance of power between the two agencies is an ever-present challenge. There is no universal solution, and the choice where the balance lands depends on the subject and goals of the respective project. However, it is essential to identify the various levels at which this balance can be tuned. The following section introduces a three-tier hierarchy of human-machine interaction that emerged throughout the case studies.

The three levels of engagement guide the exploration of how the human agency can be employed in the novel design paradigms. I formulate three scenarios for crowdsourcing in architecture by identifying similarities of task re-distribution among the four case studies.

New modes of human agency in the design process also require us to find new ways of thinking about authorship. Traditionally the lead architect is considered the author of a design. However, in a design process where architects do not design buildings but rather systems for design in the form of games and where stakeholders and third-party contributors alike produce designs, it is clear that single authorship is not the case anymore. The question of authorship is explored in the last section of this chapter.

11.3 Hierarchy of human-machine interaction

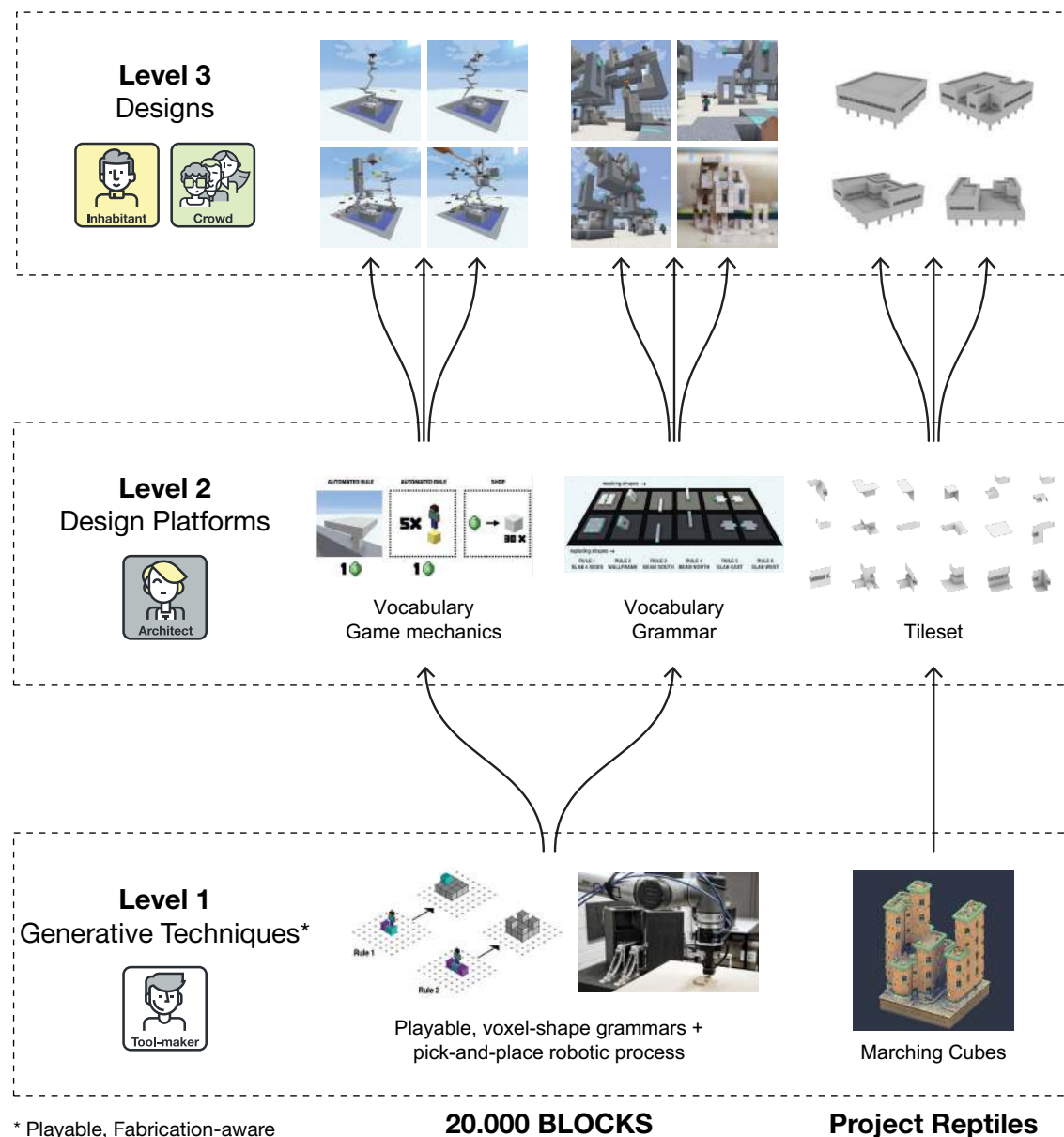


Figure 11.5 – Three levels of crowdsourcing. Generative techniques, design platforms and designs. Image credits: the author.

The proposed hierarchy of human-machine interaction is based on two observations.

First, feedback from architects using the prototypes in the *Rechteck2BIM* case study reveals that they feel creatively disconnected when dealing only with aspects of the building’s architectural program, while the generative system takes control of shape. The ability for architects to modify the design outcomes of the generative system is crucial for them actually using it. In that sense, the architect designs a system from design, not a building directly. This has an added positive side effect. As a new catalog or tileset is created, it can be reused in later projects, effectively crowdsourcing the parts library of the generative system.

The concept of using components with shared combinatorial rules is akin to the idea of *product platforms* from the product manufacturing and the automotive industry. Product platforms emerged with the phenomena of mass-customization and are beginning to find applications also in construction (Jensen, Lidelöw and Olofsson 2015). Meyer and Lehnerd 1997, p.11 define a product platform as “a set of subsystems and interfaces developed to form a common structure from which a stream of derivative products can be efficiently developed and produced.” Drawing on the analogy to product platforms, I here define *design platforms* as follows.

*A **design platform** is a set of design vocabulary, syntax rules, and game mechanics from which designs for buildings can be created.*

Not all three — vocabulary, grammar, and game mechanics — need to be present in a design platform. A platform can be as simple as a vocabulary of shapes that fit together, such as lego.

The second observation comes mainly from systemizing the *20.000 BLOCKS* case study hierarchy as shown on Figure 8.44. The case study *20.000 BLOCKS* is organized into projects created with the framework. Some projects are done within the same overarching brief or topic. Each project is a game defined by the experts. When played, these games produce designs created by the players. A similar approach is possible with the tileset approach used in *Project Reptiles*. Architects could use the tile editor to define catalogs of tiles on their own, giving the designs a bespoke design expression.

Given these two observations, I identify three levels at which human agency can be sought in a crowdsourced design process:

1. A human-in-the-loop **generative technique** is the lowest level of crowdsourcing. It can be playable if game mechanics are the means to ensure human agency. It is fabrication-aware if coupled with an automated process for manufacturing the design outcomes.
2. **Design platforms** are based on such a generative system and include design vocabulary, grammar, and game mechanics defined by the experts. The platforms contain components that carry architectural meaning.
3. **Designs** — the highest level of crowdsourcing — are the result of interacting with the generative technique through a specific design platform.

A parallel to the Lego system can be made. At level 3, a kid builds a lego toy; at level 2, the available shapes of Lego pieces are defined by the designers; level 1 is the knobs and other joinery features on the blocks that allow the assembly plus the injection molding process used to manufacture each piece from plastic. However,

Lego is modular because the design fragments used in the design process and the parts a design is built with are the same.

Wikihouse, as another example, crowdsources the design and the production (Parvin 2013). Its design system, as well as the fabrication method, is predefined. In the Wikihouse example, the only level at which user freedom is provided, i.e., crowdsourcing could happen, is the level of Designs.

However, in architecture, it is needed that fabrication and design are negotiated on a less literal transformation. As explored with *Project Reptiles*, the *iso-surfacing* generative technique delivers this feature. It works similarly to how two drops of water lying on a surface merge into one drop if close enough to each other, and opposed to how two Lego pieces placed next to each other remain two separate parts.

The idea of design platforms becomes central to the hierarchy of human-machine interaction as it opens up new possibilities of fine-tuning the balance of power between human and machine agency. The two aspects that define how design platforms can be specified are the *Generative Technique* which defines vocabulary and grammar, and the *Game Design* which defines the interactions and guides contributors towards the desired design outcome.

11.3.1 Generative technique

The use of generative techniques and game design is a means to frame the design process as crowdsourced combinatorial exploration and fabrication of parts making up buildings. The idea is not new, neither to architecture nor to nature. According to N. Gershenfeld 2006 genes are a generative system creating life designs out of four amino acids. Language is generative as well. Gershenfeld reminds us that all masterpieces are written in a language of words from the dictionary. One does not invent new words to write a novel.

A common feature across all case studies is the use of a *generative technique* to encode design principles and constraints. This allows the definition of a generative alphabet of architectural elements. Architects can design tilesets or vocabularies, and anyone can create buildings out of them.

As mentioned in *section 1.3: Methodology*, generative design techniques provide architecturally specific machine agency that forms the basis for interactivity, participation, and game mechanics in this work. Therefore the field of Generative Design gets a more dominant position in this research than the other three fields. This allowed identifying techniques that fit well with developing game mechanics or mass collaboration that previous work might have overlooked. An example of such a technique is the newly emerging application of Marching Cubes in procedural game worlds used in the *Project Reptiles* case study. Furthermore, it enabled the systematic formulation of a novel, playable generative design technique as a suborder of shape grammars — *playable voxel shape-grammars* (Figure 11.6).

The automatic generation of designs disconnects both expert and non-expert users from the process. Therefore, the particular generative technique of choice must support interaction. As described in chapter 3, I determined several suitable generative techniques (Figure 11.7) which are explored in the case studies: shape grammars, iso-surfacing algorithms, case-based design, and physically-based simulations.

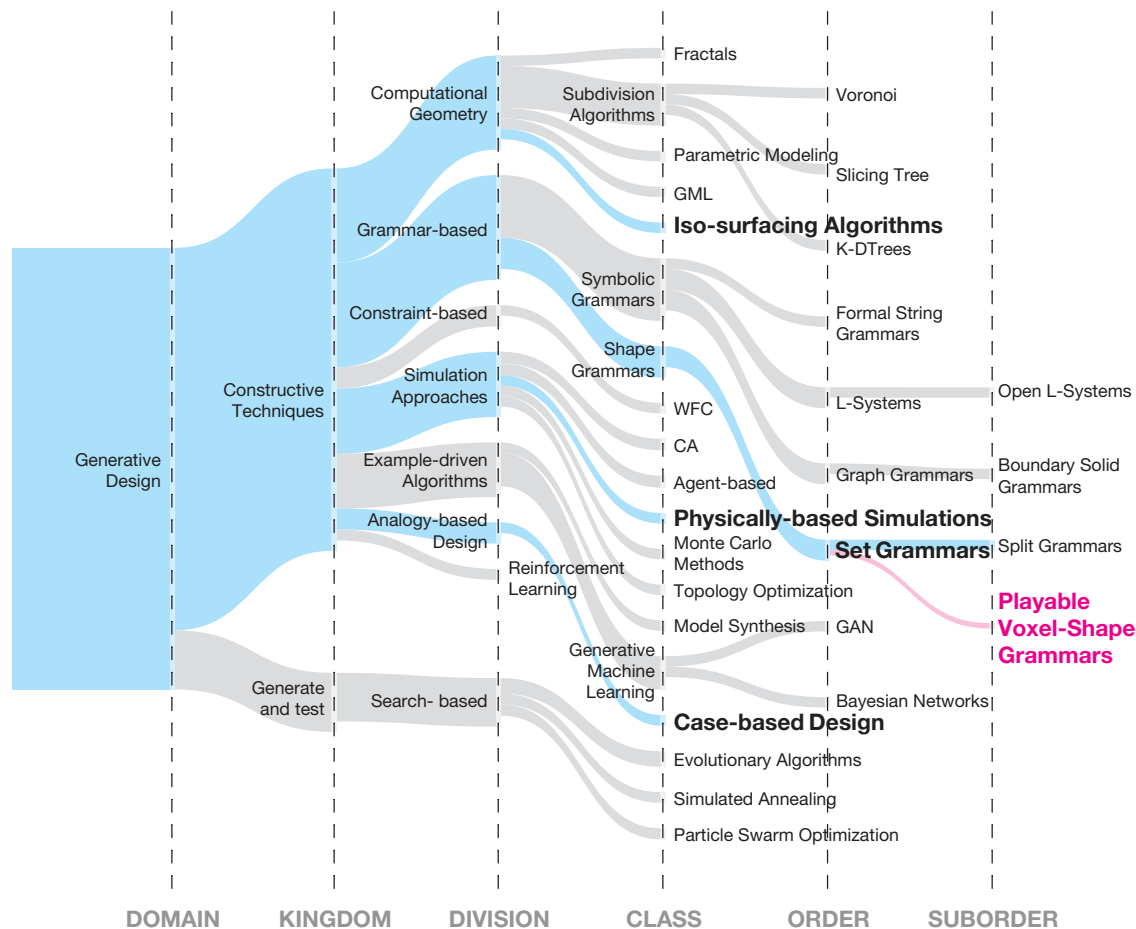


Figure 11.6 – A repeat of Figure 8.43. The *Taxonomy of Generative Design in Architecture*, introduced in section 3.2, extended with the *Playable voxel-shape grammars*. Image credits: the author.

A purely visual proof that the chosen techniques allowed a high degree of freedom for both the architects and the non-expert participant to define and configure design is shown on Figure 11.8 for set shape grammars and Figure 11.9 for iso-surfacing algorithms. Both were tested at the scale of a building and the scale of the city.

Shape-first vs Program-first

The generative techniques available to the participants can be categorized according to how they influence the design outcome. For example, *20.000 BLOCKS* and *Rechteck2BIM* occupy the ABCD design paradigm. However, the tasks of the different roles cannot be more different. While the grammar-based approach of *20.000 BLOCKS* is straightforward and concerns the users with a vocabulary of shapes and their possible combinations, *Rechteck2BIM* expects a more indirect input in the form of programmatic units (living room, kitchen, etc.) and a much stronger dependence on the received automated feedback. *20.000 BLOCKS* uses a *shape-first* approach and *Rechteck2BIM*, a *program-first* approach (Figure 11.10).

These two approaches relate to the two main paradigms of design that have emerged as reiterated by Kalay and Carrara 1996. In the first, architects start with a set of forms that they combine, modify and adapt until the desired formal and

functional qualities of the building are achieved (Alexander, Ishikawa and Silverstein 1977; Alexander 1964; Archea 1987). In the second, architects start with the building's functional program, represented as goals and constraints, and set out to find a form that will support it (Figure 11.11). Kalay points to the work by Nobel laureate Herbert Simon, together with Allen Newell and John Shaw, on the *General Problem Solver* in the 1905s as the theoretical background of the program-first approach (Newell, Shaw and H. A. Simon 1959; H. Simon 1979).

In the *shape-first* approach, the users create the geometry of a building while the machine agency provides an automated evaluation of critical building programmatic parameters. The user input is a combination of adjacent shapes from a predefined vocabulary. The items of the architectural program (rooms, zones, etc.) and their geometric representations are treated as predefined parts, i.e., a vocabulary of *shapes*. A shape in the context of this work means a geometry that carries architectural meaning, e.g., an apartment or a part of a room or few stairs is a shape (see Figure 11.12). The *shape-first approach* is explored in *20.000 BLOCKS* (p.171), *Sensitive Assembly* (p.147), and *Project Reptiles* (p.267). The newly introduced by me *playable voxel-shape grammars* (section 8.3) is a shape-first generative technique. At the same time, *Project Reptiles*'s use of iso-surfacing algorithms is a good example that the shape-first approach carries across various generative techniques and various modeling environments.

In the *program-first* approach, users set the parametric relationships between the items of the architectural program. The composition, i.e. the spatial configuration is then interpreted by a procedural algorithm into a 3D design (see Figure 11.13). The result is computationally analyzed to provide feedback to the user. This approach treats architectural program items more abstractly. The idea is for the experts and stakeholders to define conditions that the placement of items should fulfill but

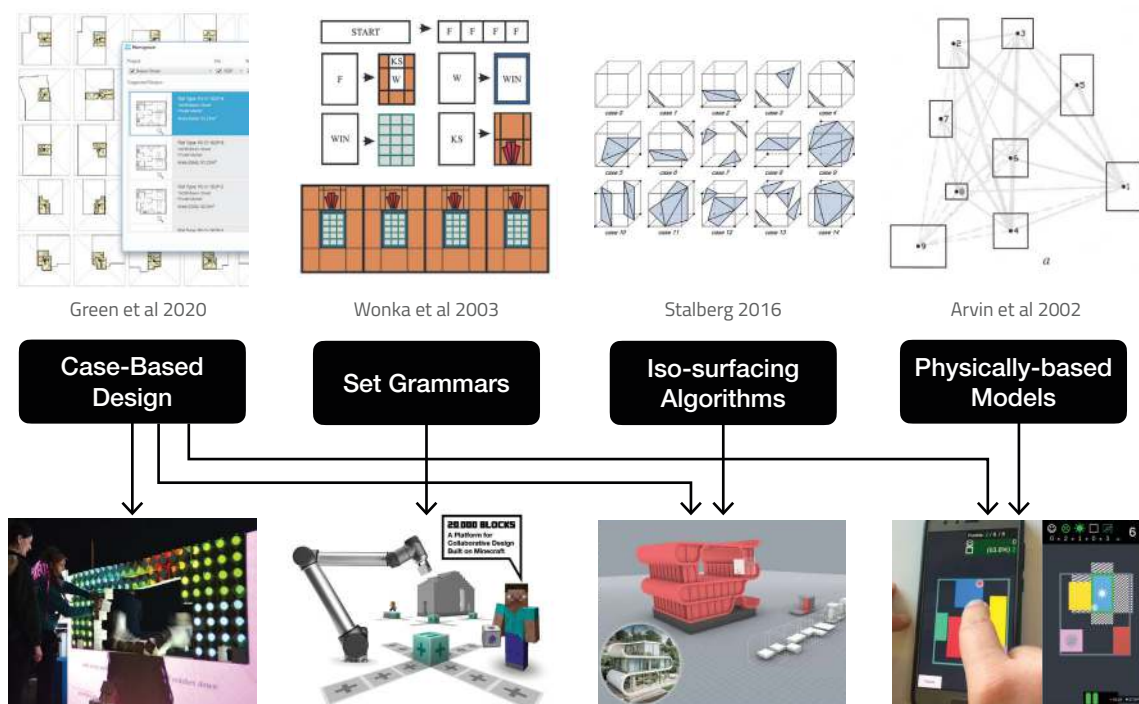


Figure 11.7 – The explored generative techniques per case study. Image credits: the author.



Figure 11.8 – Designs created within 20.000 BLOCKS. The generative technique used here is the newly introduced in this work *playable, voxel-shape grammars*. Diversity stems from the user-created vocabularies, grammars and game mechanics. Image credits: the author.



Figure 11.9 – Designs created within Project Reptiles. The generative techniques used here is the iso-surfacing algorithm Marching Cubes. The diversity stems from user-created tilesets. Image credits: the author.

leave the actual arrangement open to the users. If a condition is fulfilled, reward the player. If not - then also fine but let them know. This happens in a software environment with game-like features – scorekeeping, goals, easy to play, difficult to master — see *chapter 5: Game Design*. *Rechteck2BIM* (p.295) is the case study where I mainly explored this approach.

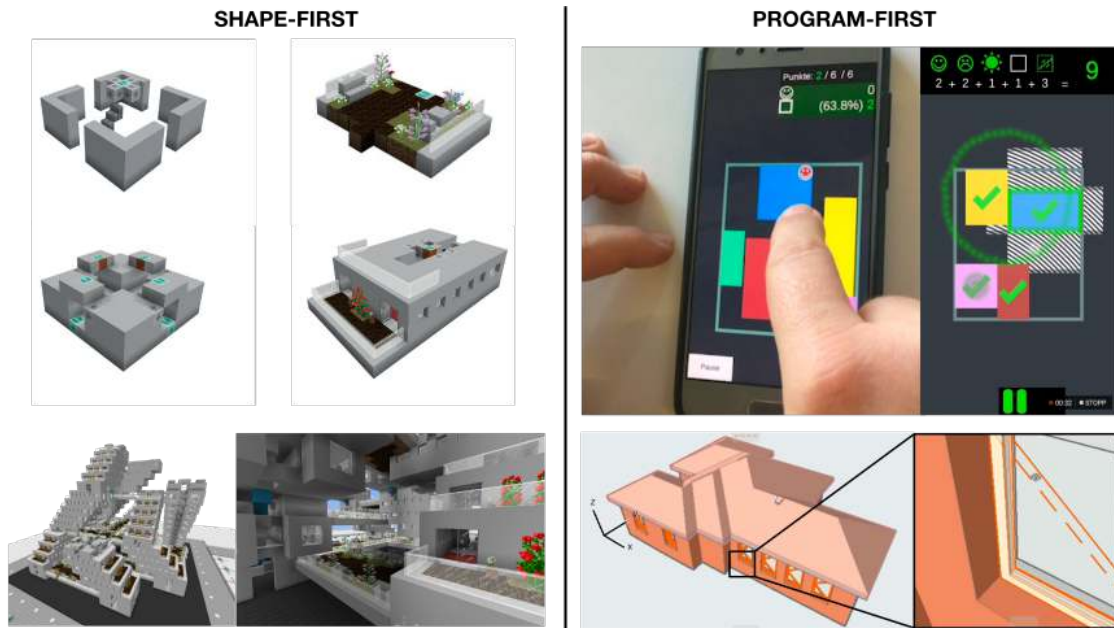


Figure 11.10 – Shape-first vs program-first. Shape-first tasks and tools revolve around the building's geometry, while program-first tasks and tools tackle the building's program. Image credits: Schneider and Stöckli, the author.



Figure 11.11 – An example of the Program-first approach in real practice. Client requirements are turned into a bubble diagram, which is used to generate a floor plan, which in turn is used to create the 3D model. Image credits: P. Merrell, Schkufza and Koltun 2010.

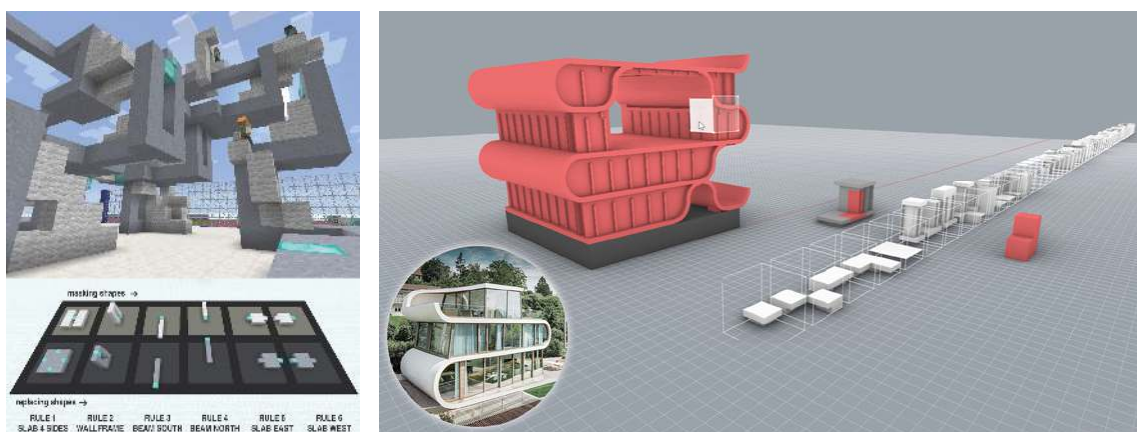


Figure 11.12 – Shape-first generative techniques. Users create the geometry of a building while the machine agency provides an automated evaluation of critical building programmatic parameters. Left: an example of a shape vocabulary from *20.000 BLOCKS*. Right: an example of tile-based shape vocabulary from *Project Reptiles*. Image credits: the author, Winkler.

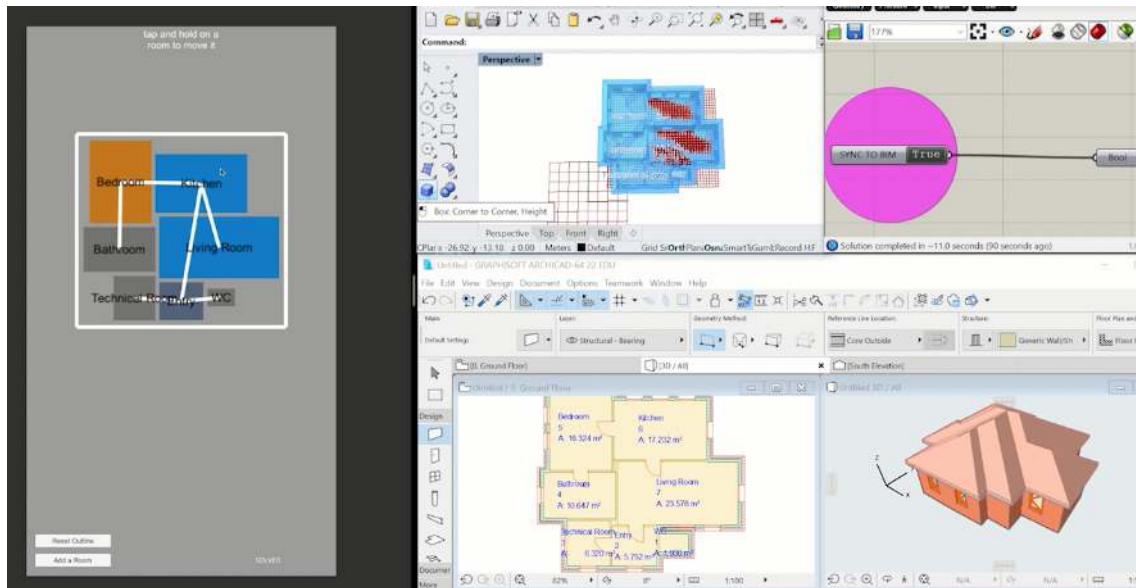


Figure 11.13 – Program-first generative technique. Prototype of a simulation-based floor plan configurator from the case study *Rechteck2BIM*. Left: The user defines the rooms and their arrangement. Room connectivity is shown with white lines. Right: A 3D (top) and a BIM (bottom) model are automatically generated to provide feedback to the user. Sample feedback includes total area, cost estimate, the house's looks, the sun's shine in the morning, etc. Image credits: the author.

The actual design process is integrative and requires both program and shape to be defined and synchronized. However, choosing a shape-first or program-first generative technique for a prototype lets me be explicit about the subject of tasks and tools provided to the participants. Shape-first tasks and tools revolve around the building's geometry, while program-first tasks and tools tackle the building's program. As such, a prototype can be developed with one or the other approach in any of the intersections on the Map of Design Paradigms (Figure 6.2).

The shape-first approach allows architectural expertise to be modeled into a design platform by an architect without programming knowledge. On the other hand, the program-first approach relies on architectural knowledge being encoded into algorithms, which requires special skills to author, so not every architect can contribute, i.e., less crowdsourcing potential.

The shape-first approach expects non-expert participants to choose the shape to place next and where to place it. Since shapes carry architectural meaning, picking and placing them requires an understanding of the repercussions. On the other hand, the program-first approach tasks users to set the relationships between the building's elements while an algorithm produces the actual shape. This can reduce the player input only to straightforward actions. However, relationships between units of the architectural program can be pretty abstract for the non-expert.

In short, shape-first tends to give more control to non-experts but requires skills they might not have, while program-first provides more power to experts but might be abstract for non-experts.

Fabrication-aware designs

Digital materials and discrete assemblies, as discussed in section 3.4 and explored with *Sensitive Assembly* and *20.000 BLOCKS*, hold the highest potential for ensur-



Figure 11.14 – Digital Materials. Left: A sample digital material from W. Langford, Ghassaei and N. Gershenfeld 2016. Middle: A wall block being removed from *Sensitive Assembly*. Right: A close up photo of the robot gripper placing a block in *20.000 BLOCKS*. Image credits: W. Langford, Ghassaei and N. Gershenfeld 2016, the author, Rui Nong.

ing manufacturability of models produced through a playable generative technique. This comes from the use of vocabularies or tilesets in these techniques that map nicely to the physical parts of a digital material (N. Gershenfeld, Carney, et al. 2015; W. Langford, Ghassaei and N. Gershenfeld 2016; Rossi and Tessmann 2017b,c, 2018).

Gramazio, Kohler and Willmann 2014 point out that with the introduction of robotic fabrication to the architectural practice “the modern division between intellectual work and manual production, between design and realisation, is being rendered obsolete”. Therefore the design paradigms that I explore in the case studies allow the fabrication method to be chosen in advance so that the design exploration is within its constraints and possibilities. This was illustrated by embedding a robotic assembly process in *20.000 BLOCKS*.

The proof of principle is carried out on a 1:100 scale since, according to Budig, Lim and Petrovic 2014, a physical model is enough for this exploration, i.e., a 1:1 prototyping is not needed at the beginning.

In *Sensitive Assembly*, players interact directly with the digital material, i.e., the cardboard cubes from which the wall is made, producing a wall design as they play. In *20.000 BLOCKS* models, created by players in the Minecraft component of the framework, are post-processed and fabricated from wooden cubes by a robot. At a speed of 100 blocks/minute, an average model takes 8-12 hours to manufacture. This meant that construction automation was not yet entirely a part of the design process. One thing to explore in the future is ideas on how to connect the robot to the gameplay in a meaningful way, whereby digital models created by players are automatically output to a digital fabrication machine (robot, 3D printer, CNC cutter, etc.) for construction.

The case studies *Sensitive Assembly* and *20.000 BLOCKS* helped verify that digital materials enable the fabrication awareness of a crowdsourced, game-based design framework. After that, I decided to exclude the robot from future case studies because it proved slow and expensive to produce a model. Engaging people under unexplored design paradigms is the priority, so I focused on the generative, participatory, and game features instead.

11.3.2 Game design

All projects were successful in generating the expected design results. However, it was clear that participant engagement is where many improvements can be made. One of the aims of game design is to incentivize the players to stay in the digital

environment and create. Game design makes sure that any user action is (i) incentivized by a goal, (ii) can lead to new challenges posed by the game or other users, and (iii) triggers feedback on the player's progress. In the use of games in architecture, players who are more engaged create more design options with better quality. In addition, good game design creates a feeling of immersion, the state of flow. Immersion enhances engagement (Dede 2009). Dede 2009 defines *immersion* as “the subjective impression that one is participating in a comprehensive, realistic experience.”

Two main strategies of using game design to engage participants in the design process are explored in this work — the byproduct strategy and the direct engagement strategy.

Direct engagement vs. byproduct

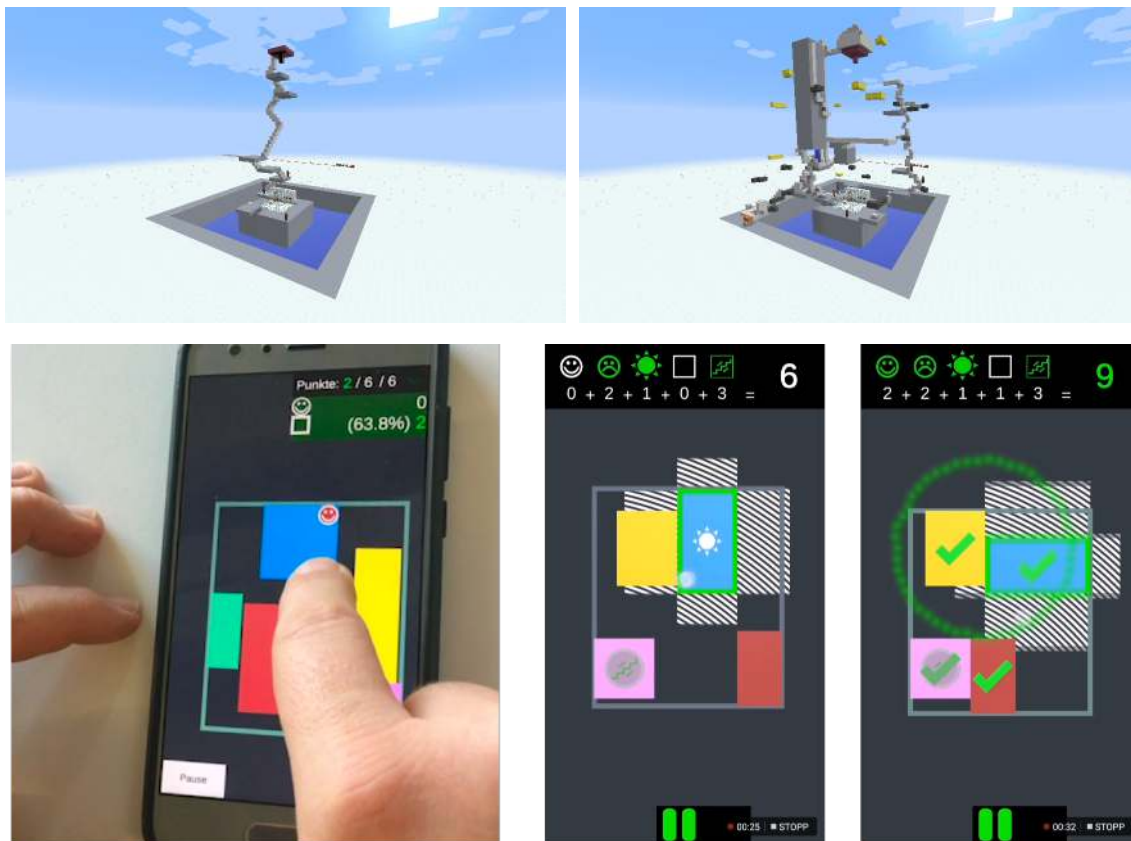


Figure 11.15 – Direct engagement vs byproduct. In the byproduct strategy (top), the game goal and game mechanics are not related to the architectural design. Secondary need to be introduced to break players away from optimization-driven gameplay. In the direct engagement strategy (bottom), the design of the building is a conscious task of the player. Various metrics visualize the architectural tradeoffs between game actions. Image credits: the author.

In the byproduct strategy, the game goal and game mechanics are not related to the architectural design. In *20.000 BLOCKS*, for example, the players' experience is tightly controlled. The Experts decide which blocks they can place and break and where they can walk. The players are given a goal but insufficient resources to achieve it. To progress, players build shapes out of Minecraft blocks, choosing from architectural elements defined by the Experts. Players are rewarded with resources

for creating one of the shapes. While players compete to reach the goal, a building emerges out of the shapes that they have built.

The byproduct strategy is excellent for design tasks with clear problem solutions subject to optimization. We observed a similar result in the project *Sensitive Assembly* where topologically optimized wall designs emerged as a byproduct of players' actions. In that case, it is most likely that an algorithm can be written to perform the task faster and better. However, the benefit of using games is in introducing serendipity to otherwise optimizable problems, such as shortest path, as can be seen on Figure 11.15. We introduce incentives to move away from optimized solutions, such as the gold platforms and the expensive water item.

As pointed out in the problem statement, architectural design is a wicked problem with no clear solution. We need to keep the participating stakeholders focused on the design challenge for them to be able to form an understanding of the criteria for good designs.

Therefore in the direct engagement strategy, the game mechanics are such that the design of the building is a conscious activity of the player. This is the main principle in the project *Rechteck2BIM*, where the space allocation for rooms in a floor plan is represented as 2D game mechanics.

This approach has high educational potential, suitable for understanding the tradeoffs in a design. We introduce measurements to visualize tradeoffs, such as the five subscores in the Rechteck game (Figure 11.15 right).

Gradient of control

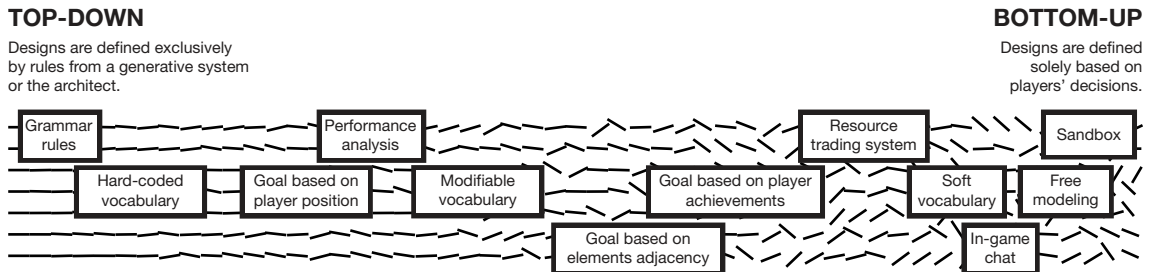


Figure 11.16 – The gradient of control. This diagram is extendable. Any new tool can be plotted on it in relation to how strong it moves the design outcomes to one or the other end of the spectrum when applied. Note that in a given prototype, it is the combination of tools that determines where the design outcomes tend towards. Image credits: the author.

The case study *20.000 BLOCKS* reveals that calibrating to what extent do Experts and Players have control over the design outcomes of the games is an ongoing challenge and the main focus of this research.

As Brabham 2013 states, for a well-functioning crowdsourcing model, the locus of control regarding the creative production of goods must be between the organization and the crowd. If the locus of control is closer to the community, such as in the case of open-source software or Wikipedia, or if the power is mainly in the organization's hands, such as when a company enlists the community to merely vote for the color of a product, we are not seeing an actual crowdsourcing model (Brabham 2013).

Therefore, I consider the balance of control that players and experts had when analyzing the results. Tools are the means for participants to affect the design

TOP-DOWN

Designs are defined exclusively by rules from a generative system or the architect.

BOTTOM-UP

Designs are defined solely based on players' decisions.

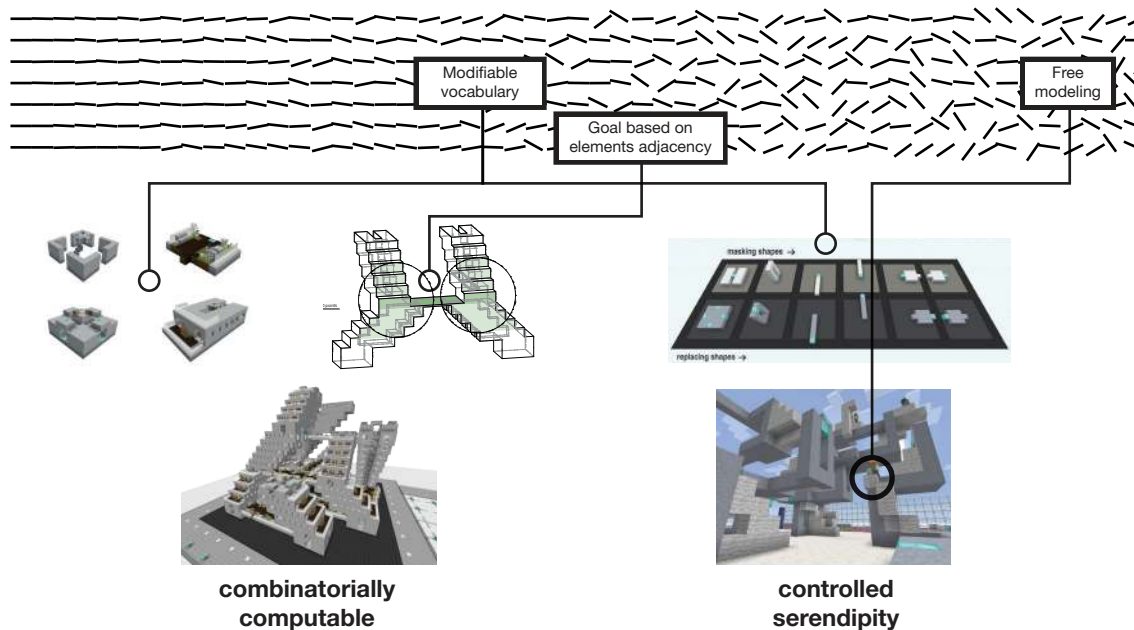


Figure 11.17 – Pure vocabulary combinations vs free modeling on the gradient of control.
Image credits: the author.

outcome, i.e., exercise control. And game mechanics are the means to match tools to tasks and vice versa automatically. Therefore I imagine game mechanics on a gradient scale of hard to soft, where the hard end of the spectrum gives more control to the *Experts* and the soft end of the spectrum, more power to the *Players* (Figure 11.16).

At the very extreme on the hard end, we can find pure generative techniques that aim to describe all design aspects in a system. Designs produced by tools and game mechanics from this end of the spectrum will be more likely to be combinatorially computable (Figure 11.17). As the literature review shows, this treats design as a tame problem and ignores its true, ill-defined nature.

At the other extreme, the soft end, we find open game worlds such as Second Life, where there is no underlying grid, no voxel, or different organizing structures. Freeform modeling and any means for players to self-organize, such as chats and forums, are in this end. Secondary-goal game mechanics, also at this end of the spectrum, used in design can introduce a level of controlled serendipity (Figure 11.17) as discussed in the byproduct game design strategy.

The type of vocabulary — soft, hard-coded, or modifiable — influences the degree to which different roles can influence the outcome and affects the balance between them.

Each feature of a generative technique, each game mechanic, each means of communication between participants, and each category of computational feedback can be positioned on this spectrum according to how it influences the design outcomes. Combining these variables into a design paradigm defines how open or rigid the design space will be.

Finding the right balance between hard and soft requires constant testing. The

TOP-DOWN

Designs are defined exclusively by rules from a generative system or the architect.

BOTTOM-UP

Designs are defined solely based on players' decisions.

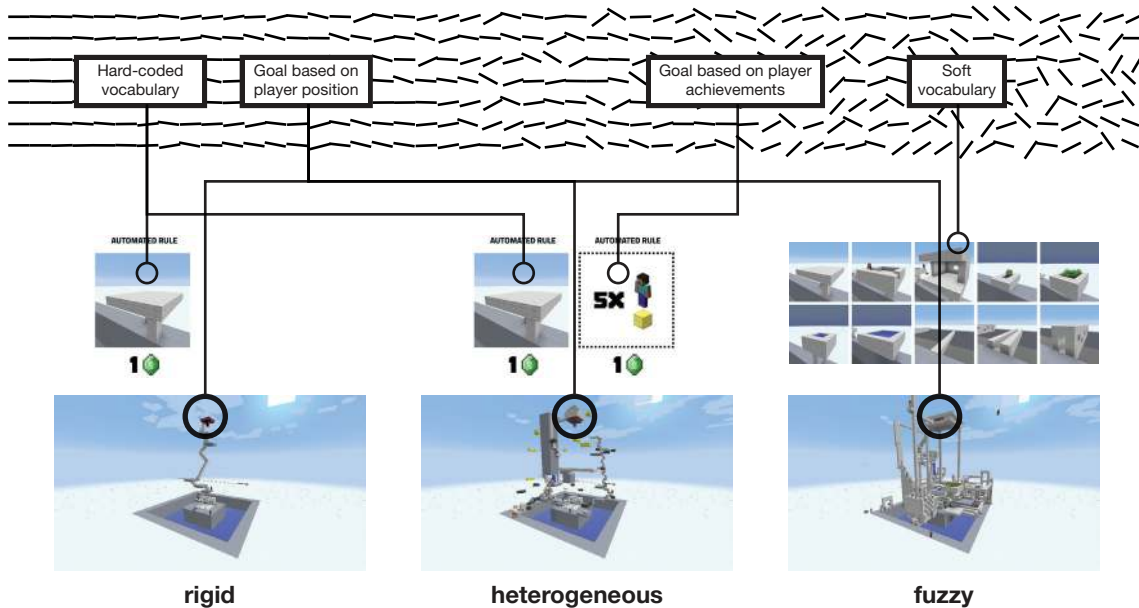


Figure 11.18 – Goal types on the gradient of control. Image credits: the author.

aim for the game designer will be to create a game that offers a heterogeneous design space (Figure 11.18). A rigid design space will not capture the creativity of players. On the other hand, a fuzzy design space captures players' imagination in a form that is challenging for the experts to use in further project stages. We can calibrate the balance better with every project iteration and every game played.

11.4 Three scenarios for the use of crowdsourcing in architecture

It is essential to acknowledge that problems of schematic architectural design are wicked problems, not tame ones. As such, it is not possible to chop the problem-solving process into micro tasks that can be distributed to the online crowd like *Re-Captcha* did (von Ahn 2011). Instead, architects and stakeholders typically perform tasks that depend on each other's outcomes. So we need concurrency of execution and means of exchange and feedback between the participants. A strategy similar to the one used in the protein folding chain *Foldit* is more fitting for applications of games and crowdsourcing in architecture. In *Foldit*, players contribute a fully-fledged solution to the protein puzzle, and others can learn from it and extend it to an alternative one (S. Cooper, Baker, et al. 2010).

At the start of this work I identified four main relevant roles of participants: the *inhabitants* (non-expert stakeholders), the *architects* (expert stakeholders), the *crowd* (third-party contributors), and the *tool-makers*. Within the novel design paradigms explored in this work, the set of tasks performed by the four different roles is expanded by new ones (Figure 11.19).

It is obvious that one of the uses of games and crowdsourcing in architecture is

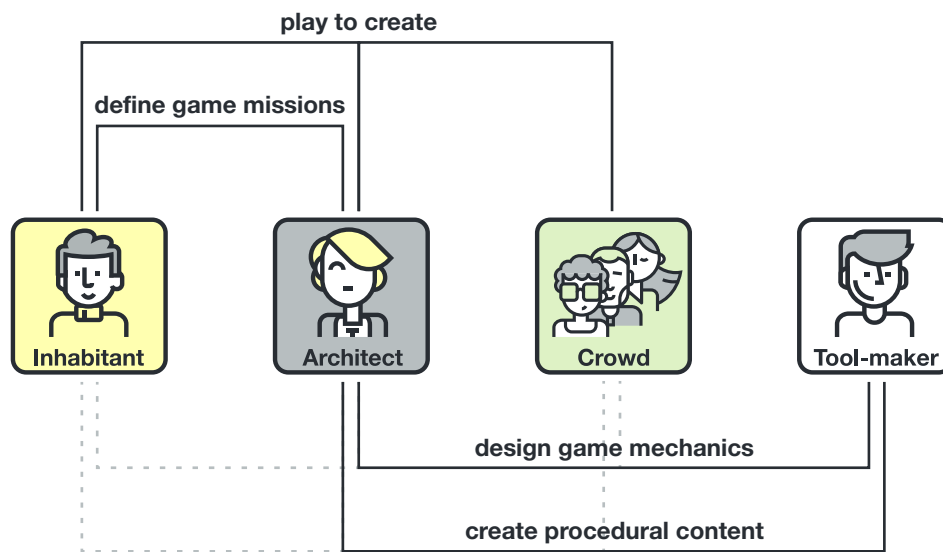


Figure 11.19 – Roles with new tasks. Image credits: the author.

to ask the crowd to create designs. Tasks traditionally performed by expert roles, architect and tool-maker, are crowdsourced to non-experts, inhabitants, and crowd. This must happen within a framework to ensure the designs respond to the given project brief and are architecturally and structurally feasible.

However, it is the feeding of content into that framework where novel and unexpected possibilities for crowdsourcing occurs. Furthermore, and much more importantly, the architect's role, traditionally considered a single agency, can now be seen as a collective role that allows for novel ways of task assignment.

In the four case studies, I identified three main scenarios for the use of crowdsourcing (Figure 11.20).

1. **Unsupervised phase-zero** — here crowdsourcing serves to enable non-expert stakeholders to formulate their preferences prioritize them without the need for synchronous engagement of an architect.
2. **Crowd-optioneering** — here, similar to the heavily algorithmic optioneering in generative design, crowdsourcing enables design iteration to cover a given, non-algorithmically explorable design space.
3. **Crowdsourced design platforms** — here, crowdsourcing enables the expansion of the design space of a playable generative design technique.

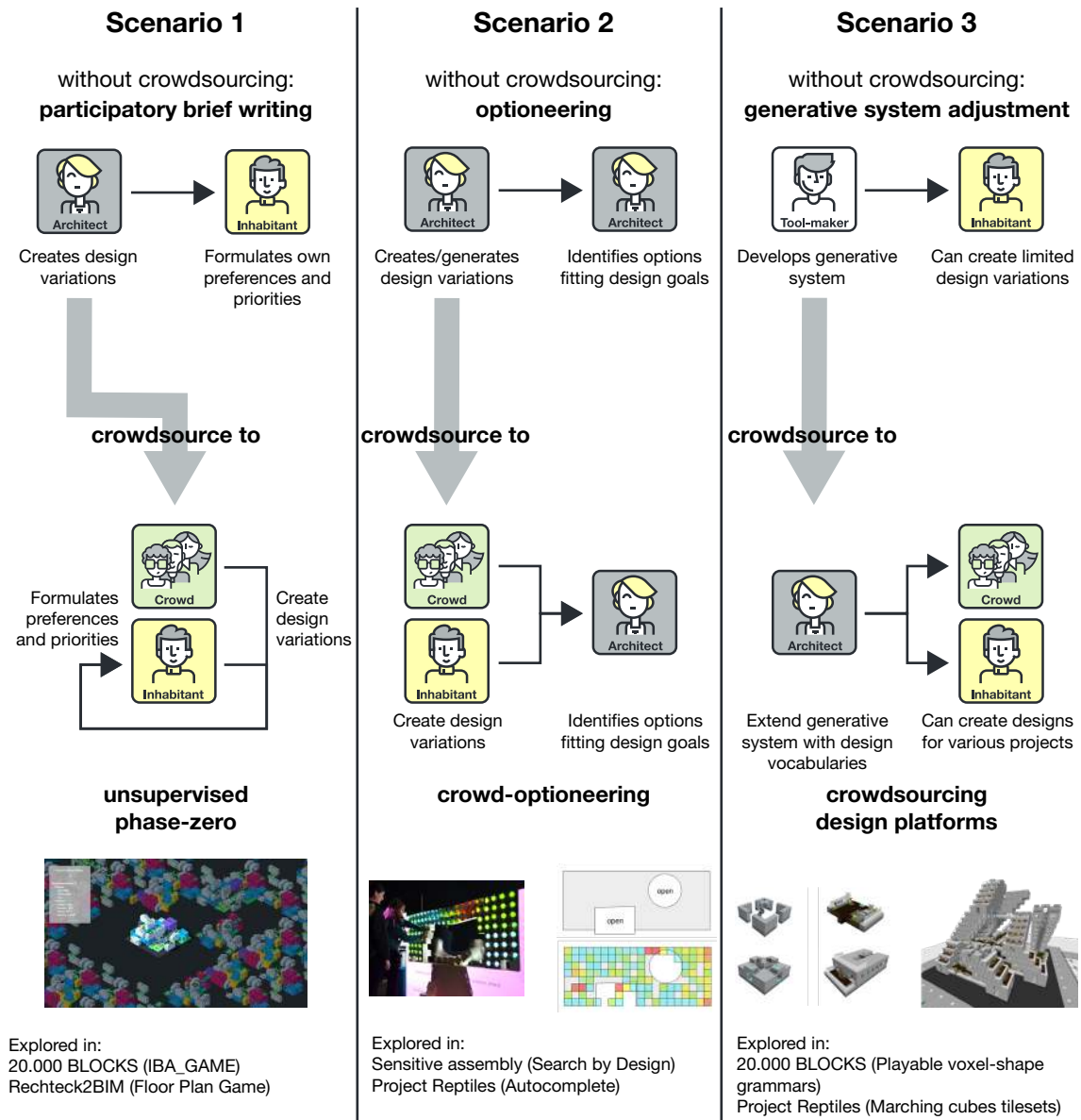


Figure 11.20 – Three crowdsourcing scenarios. Image credits: the author.

11.4.1 Scenario 1: Unsupervised phase-zero

In today's typical early design stage workflow, the architect will create design alternatives and then present them to the stakeholders. The stakeholders' priorities are established in these discussions, and the project's tradeoffs become apparent. This is done to formulate the project brief as precisely as possible. This includes the stakeholders' preferences and their prioritization. This is the activity that is most participatory at the moment. As laid out in chapter 2, it is often called Phase-Zero, and we can think of it as collaborative brief-writing.

This scenario is illustrated with the *IBA_GAME* from the *20.000 BLOCKS* case study. Here, communicating and visualizing the complex relationships between the parameters driving the project, typically performed by the architect, is crowdsourced to the inhabitants and the crowd. The crowd and the non-expert stakeholders create designs in a game format so that the non-expert stakeholders can understand the tradeoffs of the project. At the urban scale, this scenario is not necessarily

new, as the work of Ekim Tan on *Playing the City* shows (See Figure 2.8 in chapter 2). The novel application I introduce here is at the scale of the building as in the *Rechteck* game from the *Rechteck2BIM* case study. The game asks the player to arrange rooms given a particular preference for their adjacency and size. Points are assigned to fulfilled adjacency requirements, minimizing footprint, and others. As players attempt to fulfill all of their choices, conflicting requirements become apparent. The architect does not have to be involved in the stakeholders' journey to self-awareness regarding the project's requirements. They can complete this journey having themselves, the crowd, the generative system, and its feedback as the available resources.

11.4.2 Scenario 2: Crowd-optioneering

Another task the architect performs in the design process is evaluating different options to solve a specific problem. Lately, at least in the computational design field, when this activity is done with the help of algorithms, it is called optioneering. According to Autodesk 2021, optioneering “can be used to explore a design space quickly when you might not know what metrics you want to optimize for yet.” Architects create multiple design variations for their personal, i.e., internal use. They use these designs to understand the problem space and iteratively form an idea for the direction they would like to take the architectural design. Optioneering interfaces enable architects to browse computer-generated massing models in computational design software such as Spacemaker (Figure 9.14). The speed gained by the algorithmic generation and human exploration reduces the explored design space.

Creating options can be crowdsourced to the non-expert stakeholder or the third party contributor, i.e., the crowd. The expert then can filter, reuse, mix and extend these crowd-created design variations. Special tools are needed for that as the prototyped ones in *Sensitive Assembly* for filtering player-created wall designs (Figure 7.19) and the one in *Project Reptiles* for autocompleting massing models (Figure 9.17).

The reintroduction of human agency in the generation phase leads to increased explored design space. The individual peculiarities of each contributor and other serendipitous events and choices increase the diversity of design and are not encodable in an algorithm.

This scenario might seem very similar to the first scenario. However, the focus is different. Unlike scenario 1, there is no need for direct feedback to the contributors. The purpose is to collect the data, e.g., the designs they create and make it available to the architect. Despite the subtle difference, in an overall application of crowdsourcing in the design, the same crowdsourcing platform can allow both scenarios to happen, making double use of the crowd's input.

11.4.3 Scenario 3: Crowdsourcing design platforms

The third significant task to crowdsource that I explored in my case studies is modifying the generative system to fit a design expression by defining design platforms. I find this the most intriguing scenario of all three because it allows the productization of architectural expertise. This can open up a plethora of new design workflows

and business models.

A generative system with a large enough design space can enable non-expert stakeholders and the crowd to create project-specific designs. If the design space is too narrow, i.e., only a limited set of options can be generated by the system. The crowd creating the designs feels constrained and unable to express their genuine design intent. They begin adapting their preferences to the possibilities of the generative system. As P. Merrell 2007 points out, “A procedural modeling technique that can only model a specific type of object often has limited value, because as soon as a significantly different object is needed, the technique must be reprogrammed.”

Currently, as discussed in detail in section 3.2, the creation of content for a generative system is done in a proprietary modeling environment explicitly developed for this purpose. Examples are the CityEngine that uses the Split Grammars approach from Wonka et al. 2003 or the various shape grammar interpreters such as the general 2D grammar interpreter by Trescak, Esteva and Rodriguez 2009, or the MALAG interpreter by Correia, Duarte and A. M. Leitão 2010.

In a typical workflow today, the *tool-maker* will carry out that work. However, “3D artists will find that creating a new example model is often easier than adjusting an existing procedural modeling technique to suit their needs.” (P. Merrell 2007).

In my case studies, I explored how the task of seeding a generative design technique with project-specific design content is crowdsourced to a large group of architects or anyone else with the ability to think up 3D combinatorial systems. Each contributor contributes vocabularies, grammars, and game mechanics that the generative algorithm can use to generate designs. The contribution can be holistic or start by modifying a catalog submitted by another contributor.

This scenario was prototyped using the playable voxel shape grammars in *20.000 BLOCKS* and the marching cubes tilesets in *Project Reptiles*. In *20.000 BLOCKS*, game creators can model the vocabulary and the rules visually directly in the Minecraft world, requiring very little previous knowledge in 3D modeling. In *Project Reptiles*, experts can model a tileset that carries their design expression and intent in Rhino. In both ways, the underlying generative system does not need to be modified by the content creators, thus opening the possibility for widening the design space by crowdsourcing various catalogs of elements that inform the system.

In the end, all platforms taken together increases significantly the design space the generative system can cover. A task too big if left simply to the tool-maker. Not just because of the time it would take a single person to model multiple diverse catalogs, but also because creative thinking and creative insights are largely repetitive if left to the same author. We need various authors for a true diversity of ideas encoded as catalogs.

It is important to note that for scenarios 1 and 2 to be meaningfully applied, a generative system extendable through design platforms as described in scenario 3 will be needed.

As an outlook, the ability to automatically derive tilesets from example designs can further increase the accessibility and application of the third scenario for using crowdsourcing in architecture. In the case studies presented here, architects had to model the elements of a design platform individually. While much more straightforward than reprogramming a generative algorithm, this still requires developing new skills. The work by (P. Merrell 2007) on Model Synthesis shows strategies to derive tilesets out of holistic models created by the contributors as a sampling source.

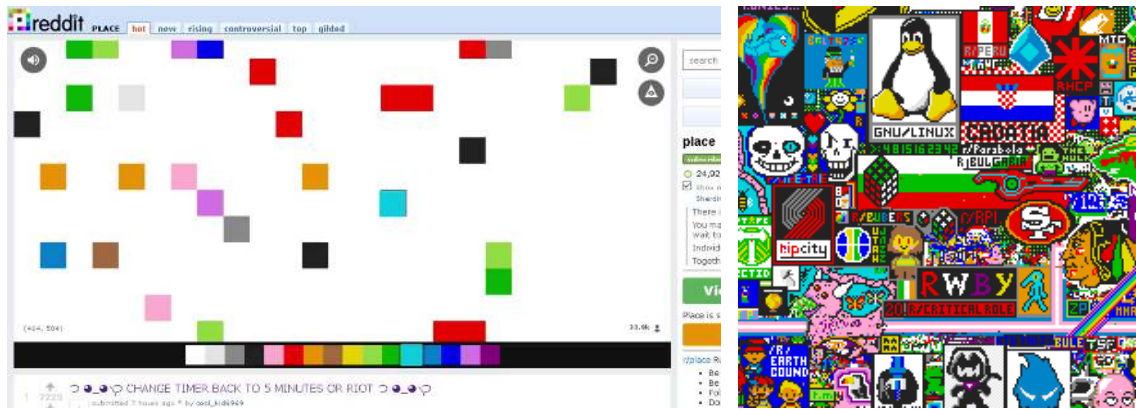


Figure 11.21 – Granular authorship in r/Place. Left: A contributor’s UI as they paint one pixel every five minutes. Right: a fragment of the final state of the r/Place canvas after 72 hours. Image credits: Reddit.

11.5 Granular authorship

The ill-defined nature of design problems makes the authorship question in architectural design important. Had design problems been tame instead of wicked, we could hand authorship over to a method that the professional community has widely accepted. For example, authorship is much less relevant in the case of a structural analysis of a building since it is done following an industry-wide accepted procedure, i.e., FEM, graphic statics, or others. However, there is no calculable solution to design problems. The challenge in front of the author, or authors, of an architectural project is to stage a platform where conflicts between the stakeholders’ understanding of the problem can emerge. The solution is less relevant than the need for stakeholders to agree on a shared problem definition.

Traditionally the lead architect is considered the author of a design. However, given the three scenarios for crowdsourcing tasks of architectural design discussed above, we can envision new, much less authoritative, modes of human agency in the design process. Roberts 2001 states that in the current times of increasing awareness and technological augmentation of all aspects in life, the go-to Authoritative strategy for dealing with wicked problems loses its power (Figure 11.22). Except for project teams and architectural competitions, the alternative strategies, collaborative and competitive, are relatively unexplored in architecture. The *rPlace* project by Reddit shows the engaging and creative power of staging a simultaneously competitive and collaborative design environment (Figure 11.21). A single person could not create anything independently due to the limit to paint one pixel every five minutes. Participants are forced to act in coordination.

Post-occupancy building extensions are probably the most common form of shared authorship in architecture. A strong recent example of this is the project Quinta Monroy by Elemental and the architect Alejandro Aravena (Figure 11.23 right). The architect provided a predefined architectural form that is half a house covering the essential habitable functions. As inhabitants’ lives unfolded in the new settlement, they filled the intentionally left gap with rooms of their choice.

The uses of games and crowdsourcing presented in this work can enable such co-authorship to occur already during the design phase. The mix between architect-defined design vocabulary and freely built shapes by the players in *20.000 BLOCKS*

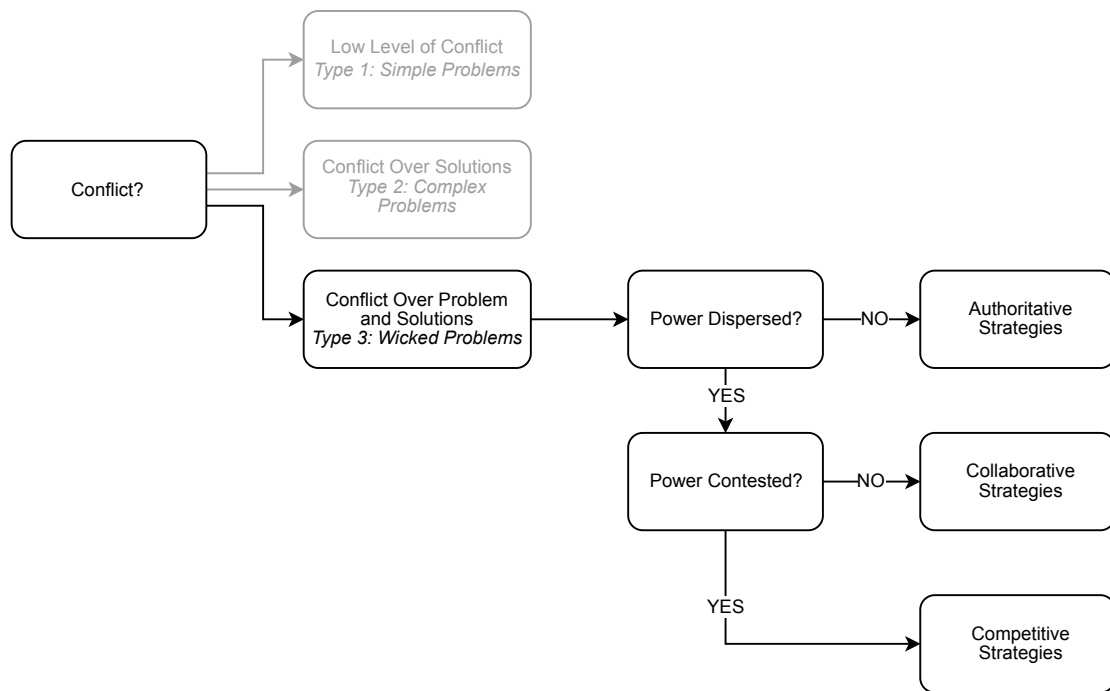


Figure 11.22 – A repeat of Figure 11.1. The three strategies for solving wicked problems — Authoritative, Collaborative and Competitive (Roberts 2001). Image credits: the author after Roberts 2001.



Figure 11.23 – Granular authorship in architecture. Similar to the project Quinta Monroy by Elemental (right), the combination of predefined architectural forms and free-built spaces in *20.000 BLOCKS* (left) results in granular authorship. White material is placed by players, gray material comes from the architect-defined shape grammar. Image credits: the author, Elemental.

is an example of how this can be staged (Figure 11.23 left).

According to Michalatos 2016, granular data structures of the 3D models, such as the one used in Minecraft, i.e., *20.000 BLOCKS* or the ones used in *Project Reptiles*,

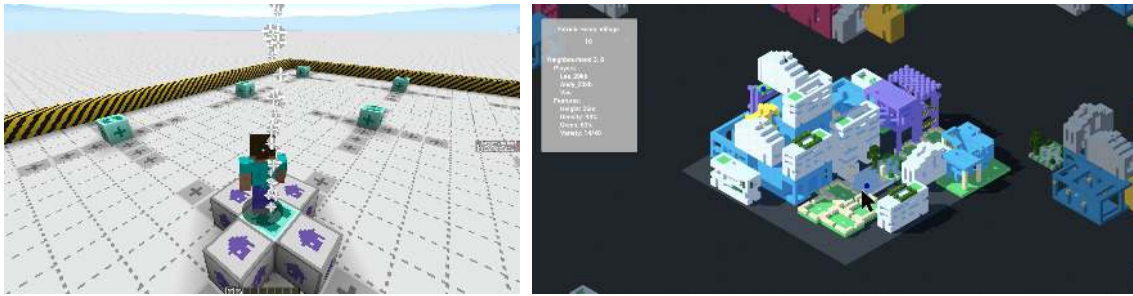


Figure 11.24 – Granular authorship in IBA_GAME. Left: A player placing a building element. Right: A design of a neighborhood with the three players that created it listed. Image credits: the author.

inherently supports concurrent authorship (Figure 11.24). Michalatos speculates further that tracking of every singular user action that creates and changes the design can enable new explorative and generative design techniques.

The ability to track the provenance of contribution to a design leads to a shared intellectual property of the architectural product, which opens up opportunities for new business models in the design profession. Multiuser or multiplayer participatory environments enable the stepwise distribution of the finite resources that architecture deals such as space, views, exposure to sunlight or shade. This granular form authorship could eventually lead to granular ownership over the same resources being allotted in the design process.

Chapter 12

Conclusion

12.1 Summary

In this work, I explored the use of games and crowdsourcing for the schematic architectural design of residential buildings. This research steps on two technological pillars — *network-enabled participation* and *ubiquitous digital fabrication* — that by 2050 will define how the built environment will be created. My research investigates exploratively two sets of questions. First, if everyone can participate in the *network-enabled creation* of the built environment, what role will they play? And second, if anyone can use *digital fabrication* to build any building, what paradigms will govern the design process? The research questions concern the balance of design power between different roles and the range of their tools and tasks. To avoid entangling myself in a discussion about feasibility or efficiency, I begin with the premise that all construction is entirely automated, making it is possible to fabricate any structurally sound building design digitally.

I identified a research gap at the intersection of the four fields that pertain to the research questions: *Participatory Design*, *Generative Design*, *Game Design* and *Crowd Wisdom*. State-of-the-art participatory design practices acknowledge the actual, ill-defined nature of design problems, taking stakeholders' values and preferences into account. However, it overburdens the architect, who has to synthesize more constraints into a one-of-a-kind design. Generative Design promises to equip architects with great power to standardize and systemize the design process. However, the common trap of generative design is treating architecture simply as a tame problem. Game Design and Crowd Wisdom, as observed in Citizen Science and crowdsourcing platforms such as ReCaptcha and Foldit, hold the potential to provide the level of content creation, options exploration, and automation of guidance that can break participation and design computation out of their respective confines and integrate them.

In the presented four case studies — *Sensitive Assembly*, *20.000 BLOCKS*, *Project Reptiles*, and *Rechteck2BIM*— I explored various techniques to employ the practices from the four fields in service of architecture. In response to the first research question, which paradigm will govern the design process, I propose a Map of Design Paradigms that prescribes how to build tools at the intersections of the four fields. The exploration of the second research question revealed a layered structure of entry points for crowd-contributed content and the granular nature of authorship among the four different roles — inhabitants, architects, the crowd, and tool-makers.

In my research, prototypical models and processes were developed and investigated to provide architects with crowdsourcing as a new tool in the design process. When developing the projects in the case studies, many decisions to pursue this goal were taken intuitively. Others were based on feedback from the users of the developed tools and prototypes. Even though I always intended to include architects as designers using the generative tools I developed for crowdsourcing, a significant finding is that tooling needs to be developed so architects can contribute to the generative system's design. I called this the crowdsourcing of design platforms.

The approaches presented in this work are well suited to solve problems of the spatial and programmatic organization of a building. As the creation process happens entirely in a digital medium, it can be easily integrated with tools and machines used in the later stages of design, specification (BIM), and construction (Construction Robotics).

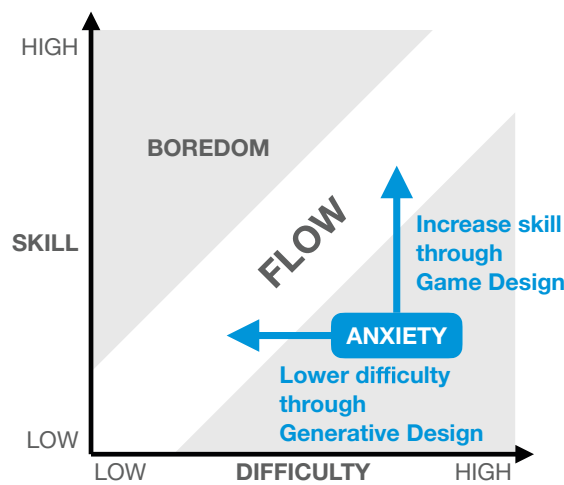


Figure 12.1 – A repeat of Figure 1.7. Image credits: the author.

The main challenge I perceived in creating the conditions for crowdsourcing within a digital environment is illustrated on Figure 12.1. It is to move a participant with a particular set of skills from the zone of anxiety into the zone of flow. The challenge is exacerbated by the fact that this transition needs to be automated to scale participation up. It cannot happen under the guidance of an expert user. On the one hand, I employed algorithms to assist the participants, i.e., make the task easier. On the other hand, the game design helped slowly bring participants up to a sufficient skill level to take on the often challenging task of participation.

12.2 Relevance for the architectural discipline

There are views on architecture that present it as a communication system of design principles and drafted building manuals such as Alberti's understanding of the architect as the single project Master (Carpo 2011, p.23). Such a definition of architecture explains and secures the architect's role as a mediator for large groups of clients. Developing participatory computational design systems for a wider professional community of architects is a key enabler to explore the benefits of massively-collaborative creative design search and discovery (Aish 2016).

At the same time, people, in general, excel at spatial reasoning and creativity, i.e., the ability to use and hack design systems to express themselves (S. Cooper 2014). Opening up the design process to non-experts could help scale up architecture's inclusivity, innovate on architectural typologies and business models and find a much-needed revalidation of the discipline by creating value for the middle of the bell curve (Bryson 2017).

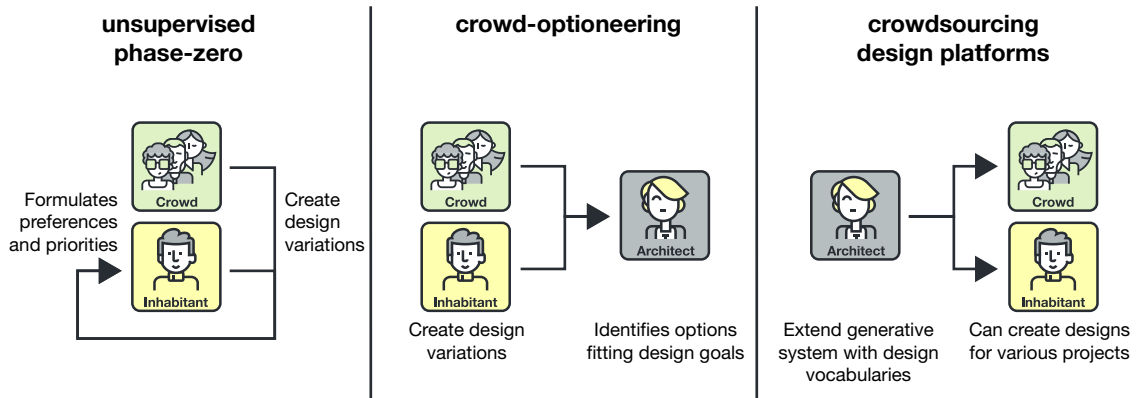


Figure 12.2 – Three crowdsourcing scenarios. Image credits: the author.

The use of crowdsourcing in architecture can mean that thousands of architects can create content to inform a generative design system, thus expanding its design space, opening it up, and softening its deterministic qualities (Figure 12.2). I explored this idea with the creation of shape catalogs and tilesets in the projects *20.000 BLOCKS* (section 8.3) and *Project Reptiles* (chapter 9).

Crowdsourcing can also be about millions of non-experts creating designs so that an architect equipped with computational tools can browse and find patterns for inspiration or use them to autocomplete a design they have started sketching out. I explored this in *Project Reptiles* (chapter 9), *IBA_GAME* (subsection 8.4.3), *Rechteck2BIM* (chapter 10) and *Sensitive Assembly* (chapter 7). Or crowdsourcing can mean that the stakeholders can browse through options generated by the crowd to increase their understanding of the complex issues at hand in a given design project and establish a common language of discussion. I explored this in *20.000 BLOCKS*, more specifically *IBA_GAME* (subsection 8.4.3).

At the same time, the adoption of game design in architecture can be about finding the balance between exploring algorithmically pre-defined design alternatives and open-ended design. I explored this in the *20.000 BLOCKS* projects such as the *Platform game* (subsection 8.4.1). Games can also provide an immersive environment where conflicts between stakeholders' interests can be simulated and explored. I explored this in the *20.000 BLOCKS* projects (section 8.3). And, games can be about providing self-reinforcing mechanisms of motivation of participants for finding designs that an algorithm would be incapable of generating. I explored this in *Sensitive Assembly* (chapter 7), the *20.000 BLOCKS* projects (section 8.3) and *Rechteck2BIM* (chapter 10).

12.3 Thesis contributions

There is no practical precedence of how the multitude of non-experts enters architectural design. This dissertation's contributions:

-
- A proposed Map of Design Paradigms at the intersection of *Participatory Design*, *Generative Design*, *Game Design* and *Crowd Wisdom* (See Figure 6.2) which reveals a significant research gap in architecture, as well as across all disciplines (See Figure 6.3);
 - The introduction of a unified taxonomy of generative design across the disciplines of architecture, computer science, and computer games (See section 3.2);
 - The introduction of a new type of Shape Grammars, namely *Playable Voxel Shape Grammars* (See section 8.3);
 - The identification of three use cases for games and crowdsourcing for schematic architectural design (See Figure 11.20);
 - Systematizing various game mechanics according to the balance of control between experts and non-experts over the design outcome of a crowdsourcing design environment (See Figure 11.16).

12.4 Future Work

With respect to the first research question, which design paradigms exist at the intersection of the four fields, some of the paradigms are not explored in this work. The two areas in the research gap that remain to be explored in the future:

1. the intersection of participatory design, generative design, and game design, excluding the participation of the crowd (ABC) and
2. the intersection of participatory design, game design, and crowd wisdom, excluding generative design (ACD).

Concerning the second research question, the roles of the crowd can be explored in much greater depth. However, primarily reasons of the technical implementation prevented me from doing so in this work. Implementing a well-working, fool-proof online digital application requires resources, expertise, and time that neither I nor the people I had the pleasure to work with possessed.

The 20.000 BLOCKS framework technically supported a mix between collaboration and competition, but the games created with it mainly focused on single-player or competitive mechanics. This was because a Minecraft server with the 20.000 BLOCKS framework could have about 4-6 players simultaneously, which is insufficient for group dynamics to emerge. Modifying Minecraft to suit the needs of 20.000 BLOCKS required users to install it from the *Technic Launcher* which requires advanced skills. A better experience in the programming language JAVA could have allowed the distribution of 20.000 BLOCKS via the vanilla Minecraft, using the Minecraft protocol as the research by Gray et al. 2019 did.

Furthermore, given sufficient resources to manufacture physical prototypes, integrating automated construction processes in the crowdsourcing framework can be revisited. This would require close collaboration with construction experts and developing a prototypical product platform that enables a diversity of designs to be defined in the form of fabrication-aware vocabularies, grammars, and tilesets.

12.5 Outlook

Several recurring topics specific to the discipline of architecture reemerged within the research presented here. The shape-first vs. program-first, the additive vs. subtractive, master architect vs. granular authorship, automation vs. augmentation, crowd wisdom vs. crowd stupidity debates have defined many aspects of architectural discourse in the past and the present. My work does not offer a resolution but a platform for these debates to continue in a new medium and with new tools.

If the design process is sometimes considered a black box, holding the secret of the master architect, the approach I present can help peek into this black box and increase the discipline's understanding of it. Tracking the provenance of design contributions, concurrent authorship, and the hierarchy of human-machine interaction can make many intuitive processes in the profession more explicit.

At first, this will be interesting for researchers but later for practitioners too, since it can open up possibilities for new business models that use the granular data of projects and the crowd as a designer. The regulatory framework that defines the fee structure for architects, for example, in Germany today, has not seen significant changes since the 19th century (1871). The newly emerged split between planning and construction that started with Alberti in the 15th century resulted in more and more free architects or artistic architects (*Künstlerarchitekten*), which began working for an honorarium that is a percentage of the construction cost. And to this day, this is the business model of architects. In his book *Who Owns the future?*, Jaron Lanier suggests how we can reward contributions to code and content with micropayments similar to how royalties used to work in the music industry (Lanier 2013). In the future, architects will partner up with their clients and build business models into the designs.

Ultimately, the integration of *participation* and *computation by crowd wisdom* and *game design* can revisit a paradigm of architectural production that has been long lost. The paradigm of cumulative design evolution and self-organization within an ever adapting hierarchical system that can be observed in informal settlements like the Logone-Birni in Cameroun (Figure 12.3 left) (Rudofsky 1964; Schaur 1991). More specifically, if the pixel canvas of *r/Place* is a map of the territory we want to build something on, what is the size and type of the grid, what does one place in each cell, how do neighboring cells merge to create a continuous space or disjoin into distinct dwellings? What digital tools are given to participants to organize themselves and create? Who will be given the status of an architectural expert, and what degree of influence will they have compared to the other roles within the crowd?

Novel architectural business models might not go the way of Google, Amazon, and Facebook. Architecture is physical, it is personal, and it is local. It is one of the few disciplines where, as Neil Gershenfeld would put it, going from bits to atom is essential for the digital to permeate it (N. Gershenfeld 2012). The strong connection between ideas and materiality in Architecture removes the strong gravitational force of software and data that centralized other disciplines. The architecture will most likely end up with a much more distributed solution to network-enabled participation and digital fabrication. For example, the Grasshopper community, which is linked to the parametric design movement, shows that we can open ourselves to each other and create platforms for exchange within the discipline of architecture. This trend needs

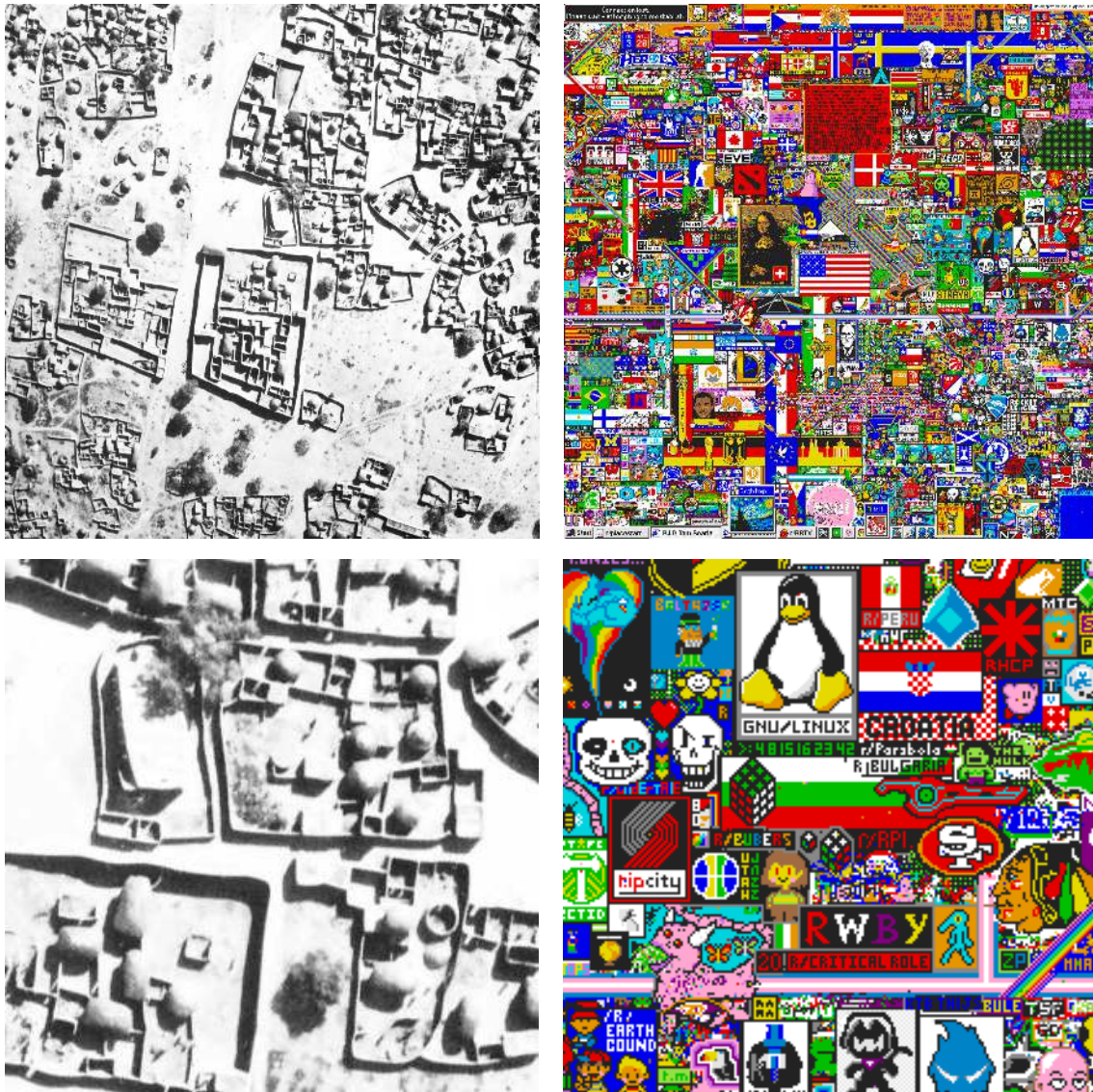


Figure 12.3 – Self-organization. Left: the village Logone-Birni in Cameroun, thousands of participants across multiple generations to evolve a system and the content (Rudofsky 1964). Right: Reddit r/Place, 250.000 participants, 72 hours, the underlying system is predefined, content evolved within it. Image credits: Rudofsky 1964, Reddit.

to continue until the exchange platform engages not only architects and engineers but also everyone else.

Technology lowers the barrier to participation and fosters collaboration. *Network-enabled participation* and *ubiquitous digital fabrication* will enable us all to shape the environment around us. With the proper thought and technological frameworks that allow us to tune the design paradigms under which this happens, we can tackle the significant challenges of this century: housing billions of additional people in cities worldwide, responding to the effects of climate change and creating buildings that help reverse them.

Part III

APPENDICES

Appendix A

Interview with the Game Designer Ben Buckton

Ben Buckton is a game designer born in Australia. Currently, he lives in Frankfurt, and work sees him commute to Prague, where he is the *Content and Player Experience Designer* for the game Factorio. Between 2012 and 2019, he was a partner at, Invent The World, Australia, offering game-based courses for kids focusing on technology and play. Since 2014, Ben has organized community events for Indie game developers in Frankfurt. In 2015, I approached Ben to collaborate on the project *20.000 BLOCKS* in Minecraft. Here we discuss the more abstract concepts from the field of games, such as game mechanics, gamification, and the subject of game design. These are important for the game design in the four case studies. A more academic definition can be found in chapter 5.

Frankfurt, 24 May 2019



Figure A.1 – Game examples. Left: Dwarf Fortress; Right: Subnautica. Image sources: <http://www.bay12games.com/dwarves/screens/dwf5.html>, accessed, 10. July, 2019; <https://steamcommunity.com/sharedfiles/filedetails/?id=1798035536>, accessed: 10 July, 2019. Image credits: Bay12Games; Sophon Relax.

Anton Hi Ben, thank you for taking the time for this interview. The first question I wanna ask you is what are some of your favorite games? I remember you once showed me how to play the game Dwarf Fortress (Figure A.1 left). We also did a project together using Minecraft.

Ben I'd say the most important thing about a game for me is that it has some challenge and the player experiences something interesting. Dwarf Fortress is,

of course, a good example. Minecraft lets people create their own story, but I think as a good game, I would say Minecraft is not a good game. I would say Minecraft is a good platform for creating games or for storytelling. A very good game gives a player the feeling of discovery and being able to choose their own path. I would say a good game would be Subnautica¹ (Figure A.1 right). That's an exploration game. There's no on-boarding. The player is put in an environment where they're playing a character who is not very skilled in that environment. They don't know what's going on and then the player and the character are both learning together. You get put in situations that are uncomfortable for the character and they are also uncomfortable for you as the player. You feel something when you play this game in the true sense of discovering how the game works. Some of this is also in Dwarf Fortress as well. You learn by failing, you learn by overstepping your boundaries and the game has a soft border, so it lets you overstep it. This is what I would call a good game. Title-wise, there's many, but I would say this is what I love most about games.

Anton Is that what you look for in a good game — the player can make a discovery?

Ben Yes. And make the choices for themselves.

Anton What is a game?

Ben Any experience where you have some sort of goal, there is a challenge to overcome and there is a set of tools that are given by the designer. That's a very broad description, I guess, of what a game is. It has to be interactive and that is where, as a designer, I think of interaction as the tools that are given to overcome any particular challenge in a game. If there's no challenge, it's not a game. If there's no tools to overcome the challenge, it's not a game. A movie is not a game. Because there's no tools. There's a challenge, and it gets overcome regardless of what you do in your seat watching it. There's no tools. But there was a challenge. There's some conflict.

Anton And toys would be then the opposite. Toys are tools and there's no challenge posed. You need to invent your own challenges.

Ben Exactly. That would be my distinction of toys and games.

Anton When you talk about a designer you mean a *game designer* and that relates to what you do. What does a game designer do?

Ben Well, I would say it's designing this experience that the player has. I guess a lot of designers might answer this question with: "Well, the game designer writes the narrative of the player rather than the character." But I would probably see it more that the designer is creating the challenges and the tools to solve the challenges. And, of course if you design those two things together, so it's interesting to use this tool to solve this problem, then you have an interesting experience. Sometimes if those things are designed separately, it can also result in a good outcome. For example, in Dwarf Fortress, which is a sandbox game², there is a set of tools and there's a set of challenges. They're

¹Subnautica is a game by Unknown Worlds from 2019 and brief description.

²Sandbox game is a game where the player has the ability to create, modify, or destroy their environment, i.e. a game that includes some form of a game creation system. The term alludes to

not really designed to counter each other, it just happens that the players figure out how to counter those challenges with those tools.

Anton There is a concept which is very difficult to explain to someone who's outside the games industry — 'Game Mechanic'. What is a game mechanic? Can you give a couple of examples?

Ben It's hard to say if the game mechanic is the tool or is that the conflict or is it the interface between the two? I would probably call the *mechanics* the things that the player doesn't directly interact with and the tools are the interfaces to the *mechanics*. You can for example have a mechanic of driving a car but you only expose the steering wheel to the player, not the pedals. So, car driving and the car physics are part of the car mechanic but the steering wheel is the tool that's given to the player to overcome the challenge of driving through an obstacle course.

Anton Does Minecraft have game mechanics?

Ben (*pauses*) Maybe yes. It definitely has simulation-based mechanics. There's not so much that the player interacts with directly. Creature AI is a game mechanic.

Anton Or maybe the day/night cycle.

Ben Day/night cycle is a good game mechanic. It's something a player can't really affect. It's interesting to think of it in this way. There's definitely some mechanics in Minecraft, but as a sandbox game, they're very disconnected from the player, so the game world will just do its thing without you regardless. Whereas if you think about something like the game Quake (or Doom)³, or any first-person shooter game, where nothing happens unless the player interacts with it. If you stay in one room, then the enemies in the other room just don't exist. They don't interact with each other.

Anton Can we think of the game mechanics as the game agency, the automated force that acts against or in support of the player?

Ben Yes, I guess, the mechanics are the individual parts of the machine that is creating the conflict for the player.

Anton Earlier, you mentioned that in a game, on the one hand, there are the players who can make choices and take actions, and that gives them the sense of control over the game, and on the other hand there are the tools or the agency of the game given by the game designer, which assigns rewards to these actions and constrains players in their choices. And that gives game designers

a child's sandbox where the child can create and destroy with no given objective. While *open world* and *sandbox* are sometimes used interchangeably, the terms refer to different concepts and are not synonymous. Source: Wikipedia: https://en.wikipedia.org/wiki/Sandbox_game, accessed 10 July 2019

³*Quake* is a first-person shooter video game developed by *id Software* and first published in 1996. *Doom*, also by *id Software*, is considered one of the pioneering first-person shooter games, introducing to IBM-compatible computers features such as 3D graphics, third-dimension spatiality, networked multiplayer gameplay, and support for player-created modifications with the Doom WAD format. Since its debut in 1993, over 10 million copies of games in the Doom series have been sold; the series has spawned numerous sequels, novels, comic books, board games, and film adaptations. Source [https://en.wikipedia.org/wiki/Doom_\(franchise\)](https://en.wikipedia.org/wiki/Doom_(franchise)), accessed 11 June 2021

a sense of control over the player's experience. What is the importance of the balance between this choice and agency? Are there any best practices that you encounter of how this can be measured by something other than the game becoming a hit? Can it be tuned or balanced?

Ben I think this is definitely one of the *unmeasurables*. There's this saying that good design is invisible. So when players are happy and they feel like they've been given a lot of choices, quite often that is not true, but they just feel like that. There are a lot of successful games with that structure, but there are also a lot of successful games where **the player has complete knowledge of the mechanics** — nothing is hidden from them or they have access to a wiki. I think the player definitely needs some agency. If they feel they are railroaded, just on a path, clicking the button, some people might say it's not a game. A recent example that came up is *Metal Gear Solid 4*⁴, which is a third-person shooter game, but realistically it doesn't matter what you do. It's basically almost a movie. There are so many cut scenes. There's lots of gameplay but **none of your choices really affect the outcome in the end**. Whereas you have the branching story of a game like *Dragon Age*, or any other RPG⁵ game, where your choices do matter and the game designers have just accommodated for the player making many different choices. You feel you've made choices because you heard from a friend that they went a different way. And then you have the kind of **meta choices**. For example, the game *Subnautica* is designed without a map. You don't know how to navigate in the environment. It's an underwater environment. However, you can just go on the internet and download a map. That is not a choice the game gives you, whether you want a map or not. But if you take that meta choice, you've effectively broken the designed experience for yourself. So, I would group them into these three categories of choices. And I think it's important that you consider all of them. Best practices-wise, you can always consider what the player will do to ruin their own fun and try to limit that. If there is a button that gives them one coin and they can press it once per hour, they will press it once per hour and they will say that your game is boring. If you take that away, they will engage with the rest of the mechanics. People always take the path of least resistance. So, as a game designer, you need to know what that path is and ask yourself, 'Is it fun?'. And if it's not, you need to modify the path by making things harder or easier.

Anton We've collaborated on few game prototypes where players had to co-create architectural designs. What were the challenges with using games in that context from the perspective of a game designer?

Ben I think the challenges here were that anything that's generated in game design rather than handmade, brings you to this situation where everything is unique,

⁴Metal Gear Solid 4: Guns of the Patriots is a 2008 action-adventure game developed by Kojima Productions and published by Konami for the PlayStation 3 console. Source: https://en.wikipedia.org/wiki/Metal_Gear_Solid_4:_Guns_of_the_Patriots, accessed 11 June 2021

⁵A role-playing game (RPG) is a game in which players assume the roles of characters in a fictional setting. Players take responsibility for acting out these roles within a narrative, either through literal acting or through a process of structured decision-making regarding character development. Source: https://en.wikipedia.org/wiki/Role-playing_game, accessed 11 June 2021

but because everything is unique, nothing is unique. There's no pattern, just a field of random noise. We see this in a lot of other games which have generated content — most of the time the content is uninteresting. But every now and then someone posts a screenshot and says: "I found this planet in the middle of nowhere where if you go to this coordinate and you look in that direction, there's this amazing location that looks perfect, as if it was handcrafted." But then that's out of 14 billion different locations, so...

Anton You're actually touching upon a very important aspect that surfaces not only in procedurally generated worlds but also in participatory design, where the world is created by a crowd of people. We are always running into this issue that the results look aesthetically and compositionally weak — they're like noise, because pretty much all evens out. It's full entropy.

Ben I think you have this diagram of full entropy and of full control and neither is good, right?

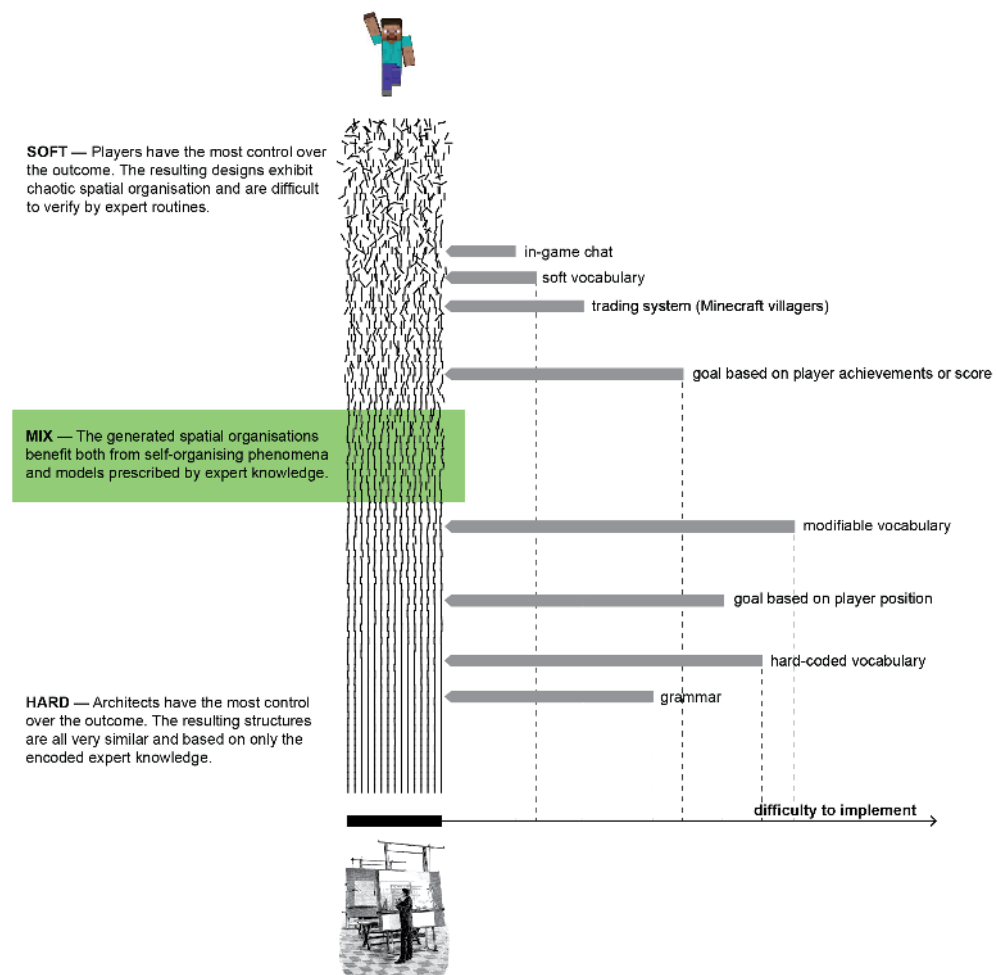


Figure A.2 – Centralized vs decentralized control. Image credits: the author.

Anton Yes, at the one end, we have the bottom-up approach, the little agents, whether they are algorithms or people, which create design, and at the other end is the single figure of a designer, the author, almost like a dictator that says, "This is how it should be" (Figure A.2). Could games be a medium to

navigate this path?

Ben Most definitely. I think we have examples in game design of total chaos, like a game where the player is dropped into a system that is complete anarchy and they need to just figure out what to do.



Figure A.3 – Unbalanced vs balanced games. Left: A typical procedurally generated game level with three rooms linked with corridors, the player's character (the @ symbol) and an enemy (represented by the K) from the 1980 game Rogue; Right: A screenshot from FarmVille, 2009, by ZYNGA. Image credits: Barton and Loguidice 2009; ZYNGA.

Anton What would be a game that does that?

Ben Older Rogue-like⁶ games were just like this (Figure A.3 left). Every time you load the game, it's randomized. You might move one step and a creature kills you that's invisible and is 10 times more powerful than you. And there's nothing you can do so you restart the game. Generally in all the Rogue-likes the idea of game balance is not considered. It's supposed to be hard and unfair. If you play 10 games, in one of those 10, you'll have a run that is at your skill level. There's this idea that, as a game designer, you have to balance your game for players of low, medium and high skill. There's always someone who can beat your game every time. There's always someone who can never beat your game. And if you don't balance this, you leave it up to the whims of fate. Then actually if a player is persistent they will have that perfect run through the game where everything is just hard enough to challenge them, but, at the same time, just easy enough so they can beat everything. The eureka moment for a game designer is if they intentionally design something like this where a player has that experience. It's almost like a flow state. And making something that gives this feeling to enough people in the market, enough different people, is The Holy Grail. People want that and we see this with a lot of free-to-play play games, such as Farmville (Figure A.3 right) and Candy Crush, where the game designers do a lot of data-driven design. These games try to abstract and narrow down what it is that makes people go into this state and find an algorithm that makes the levels hard enough and easy enough so that everyone is just constantly hooked. I don't consider that particularly valuable, but, I would say, that is the best example of achieving

⁶Roguelike is a subgenre of role-playing video game characterized by a dungeon crawl through procedurally generated levels, turn-based gameplay, tile-based graphics, and permanent death of the player character. Most roguelikes are based on a high fantasy narrative, reflecting their influence from tabletop role playing games such as Dungeons & Dragons. Source: wikipedia: <https://en.wikipedia.org/wiki/Roguelike>, accessed, 10 July 2019

the goal of every player having an optimal, procedurally generated, experience.

Anton Mihaly Csikszentmihalyi who is a psychologist, has a very good diagram to depict the state of flow. The X-axis represents the difficulty of the challenge and the Y-axis the skill level. If the challenge is too high and the skills are too low, you're in anxiety. And if the challenge is too low and your skills are too high, you're bored. There's this diagonal area which is the flow (Figure A.4).

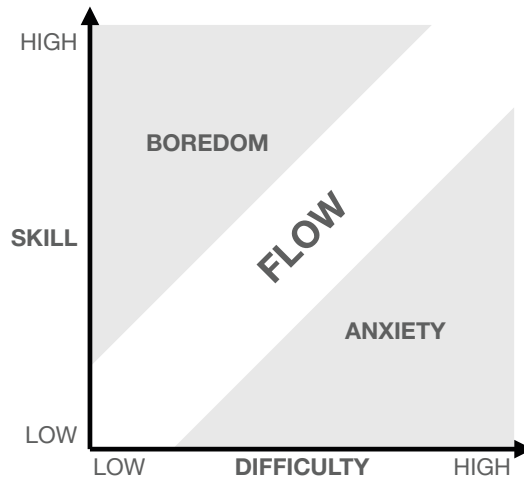


Figure A.4 – The concept of Flow. Adapted from Csikszentmihalyi 2008. Image credits: the author.

Ben Yes, it's interesting to think with this idea that you can design your game to fit in this pipe. If you give the player the opportunity to play optimally, it leads them to anxiety or boredom. If it's the fastest way for them to reach the goal, they'll take that way. Doesn't matter if they're having fun or not.

Anton Games are considered part of the entertainment industry. However, I believe they will affect or are affecting the non-entertainment industries in the near future and in the long run. What are your thoughts on that?

Ben I definitely think that this mindset that people currently have, that games are entertainment and entertainment is not productive, will change. I think there's a lot of resistance to this idea. If you're not at work, on the grind stone, making dollars, making the economy better, that thing that is stopping you from doing that, is the opposite of productive. But I think people are much more productive when they are engaged and I think at the moment, that is not a value that is measured by our economy. And therefore, the value of happy people, of people playing games, being motivated and engaged in their work is just not measurable in a dollar value yet. I think once people realize that there is an inherent value there that is immeasurable, we will see more and more people pushing towards integrating games as a way to engage people into normal life. We've seen there was obviously a big boom with gamification, which I don't consider the same as games...

Anton *Gameful* is a term that emerged in the literature as a reaction to *gamification*⁷.

Ben *Gameful* is a good way to put it. While gamification is more about the psy-

⁷See chapter 5

chology and controlling people's engagement through psychological rewards, gameful approaches are basically removing the risk of trying, the risk of failure and we already see that this is a better way to go and that is becoming more and more important. Just last night I was talking with a bunch of engineers about using Virtual Reality to train firefighters because you can't send them into a fire without skills. Because they won't learn, they will die, right? And I think in a lot of ways, what is currently holding us back as a society, is that people don't want to fail, they don't want to be liable for someone else's failure and games give us a way of trying, of putting yourself in a situation where you basically know that you can't win because you have no skills and you can find out how and you can try through iteration. That's the strength that I see we're going to get out of games in the future, especially with education, with on-the-job training. Hell, even with relationships. I think in terms of social interactions, people can get a lot out of games.

Anton Now that we're talking, I remember that *gameful* was actually introduced as a term to balance the bad connotations of gamification on one side, where it almost feels like manipulation and, on the other side, the concept of playful where it's not taken seriously.

Ben Yes. I think that's really the two ends of the spectrum. And play is actually the important thing because play is pretend. *Playful* means pretend. It's not real, the consequences aren't there. But if you take *playful* and you make it a simulation, this can become a way to experience something real without the consequences and still get something out of it. As long as people understand that it's serious.

Anton Looking at the non-entertainment industries and how they can benefit from games, I have a more particular question. Who do you think can benefit the most if we consider games as a medium to design and to create architecture?

Ben Here I'm a bit split. Maybe it comes back to this immeasurable thing. You know, there's this saying: In the economy, if you find a need, you fill a need. You find a need that someone has and you invent something to fill it. The lack of design thinking in the world at the moment is, I think, causing a lot of solutions to be based on just how much money does it cost and how much can people pay for it? And design is immeasurable. Unfortunately, today, the word design just means it's expensive. It doesn't mean that the user experience was considered. While actually design is about thinking about the user. The answer to the question "What is the experience of the person using this thing?" is, I think, where you see good design. Well, you don't see it, because it's invisible. But if it's a good design, the person using it just has a good experience and doesn't know why. I've gone off track a little bit, but I think teaching game design teaches design thinking. And, in my experience, architecture is the design thinking element of Engineering, right? It's easy to build a house that doesn't fall down but it's much harder to think about the people in the house and how they are going to use this space. Game design is interactive so you can really teach design thinking and make people think about psychology and about the physical space all at the same time. And I think that is analogous to architecture in my mind as a non-architect.
(laughs)

You can tell I have a positive outlook and I think that's a big thing. A lot of people have this negative idea, this kind of feeling that nothing has come out of games yet, therefore nothing will. We've been through several hype cycles with gameful hybridization or transdisciplinary projects and sure, we've had a lot of ups and downs. But I think the game industry as a whole is just going to get more and more diverse. We will see game design everywhere.

Anton A great sentence to end on! Thank you!

Appendix B

Getting Started with 20.000 BLOCKS in Minecraft

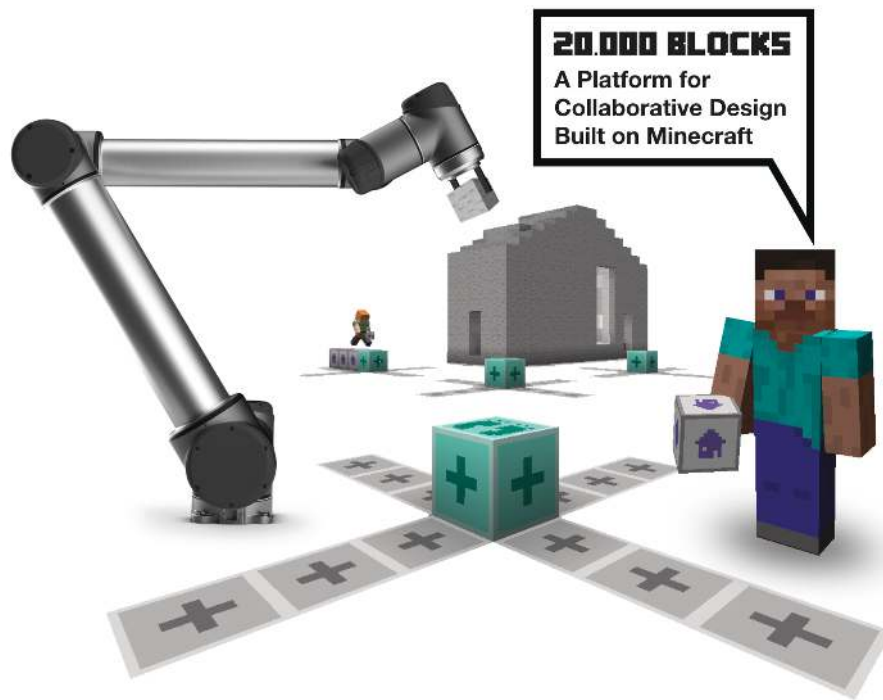


Figure B.1 – 20.000 BLOCKS. Image credits: the author.

This manual helps you get started using the Minecraft components of 20.000 BLOCKS. It was provided to participants in the various *20.000 BLOCKS* projects described in this dissertation.

B.1 SETUP IN 5 STEPS

To run 20.000 BLOCKS you need to:

- Have your own Minecraft Java Edition,

- Install TechnicLauncher (requires Java) which is used to play custom Minecraft modpacks.
- Install the 20.000 BLOCKS custom modpack in TechnicLauncher.

If you already have installed both Minecraft and TechnicLauncher, proceed with STEP 4.

STEP 1: Install Minecraft

1. To run 20.000 BLOCKS, you need to own Minecraft Java Edition for Windows, Mac or Linux. Buy and download it here: <https://www.minecraft.net/en-us/download>
2. Install and run Minecraft at least once.

STEP 2: Install Java

On macOS (tested on macOS Catalina 10.15.4)

1. To run TechnicLauncher you need to install the Java for Mac OS X Version 8. DO NOT INSTALL NEWER VERSIONS such as Java SE 11 or Java SE 14, they will not work with TechnicLauncher.
2. You need Java JRE (Java runtime Engine), not Java JDK (Java development kit). At the time of this writing the latest update to version 8 is Java JRE 8 update 241.
3. Go to: https://www.java.com/en/download/mac_download.jsp
4. Download the Java JRE 8 and install it.

On Windows

- not tested yet

STEP 3: Install the Technic Launcher

1. Go to: <https://www.technicpack.net/download>
2. Select and download the appropriate version for your operating system (Windows, Mac, Linux)
3. On macOS you will end up with a `TechnicLauncher.jar` file
4. Double-click on the `TechnicLauncher.jar` file to run it and give the permissions to Java to access the folder it is in
5. Once it runs it will ask you to install. Use the STANDARD INSTALL and click Install.
6. It will download the needed files from the net and install itself.

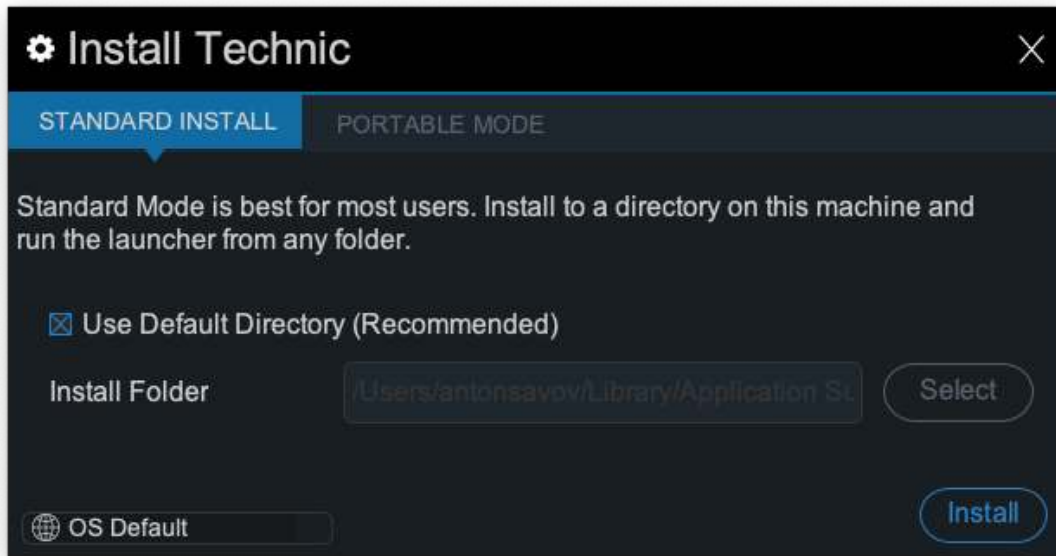


Figure B.2 – Install Technic Launcher. Image credits: the author.

7. Then it will ask you for your Mojang account. This is your account for Minecraft. Enter it.



Figure B.3 – Log in with Mojang account. Image credits: the author.

8. Now you should see the home screen of TechnicLauncher
9. Proceed with adding the 20.000 BLOCKS modpack



Figure B.4 – Home screen of Technic Launcher. Image credits: the author.

STEP 4: Add the 20.000 BLOCKS Modpack

1. Open up the Technic Pack Launcher by double-clicking the TechnicLauncher.jar (on macOS)
2. Select the MODPACKS Tab in the TechnicLauncher home screen.
3. Search for the 20.000 BLOCKS modpack by typing 20.000 BLOCKS in the search field.
4. Once you have found the 20.000 BLOCKS modpack, click on the Install button.
5. It will take a while for the modpack to download. You will know it is complete when a Play button replaces the Install button.

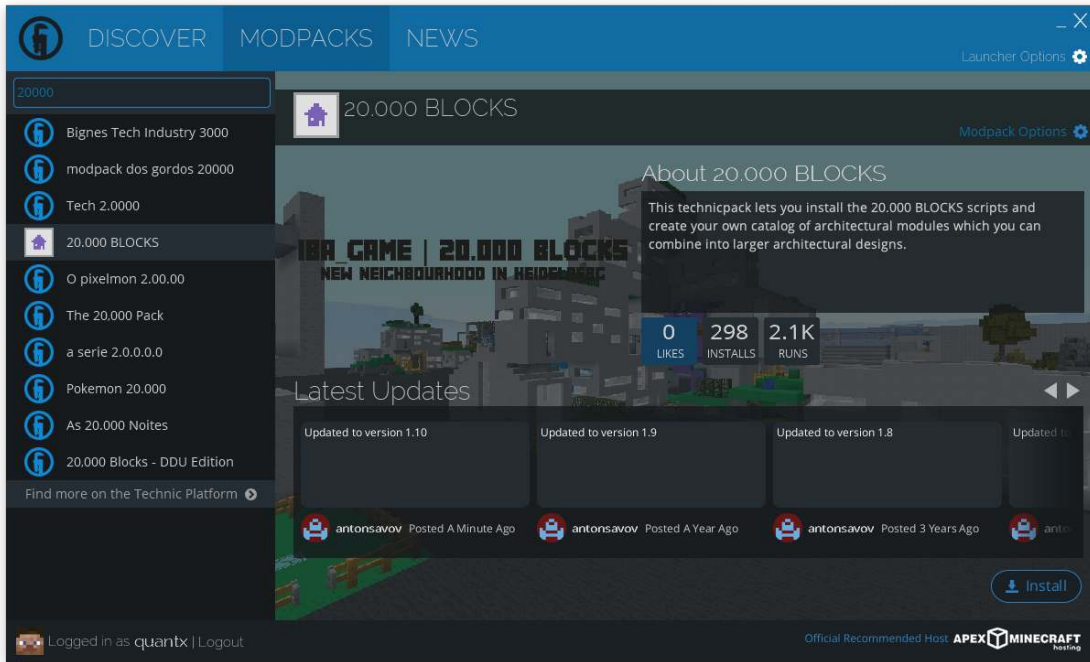


Figure B.5 – Modpacks screen. Image credits: the author.

STEP 5: Run the 20.000 BLOCKS Modpack

1. Open up the Technic Pack Launcher by double-clicking the TechnicLauncher.jar
2. Go to MODPACKS and choose 20.000 BLOCKS.
3. Press Play
4. It will take a while for the modpack to load. You will know it is complete when the Mojang logo is gone and replaced with the Minecraft Main Menu.
5. Great! You are all set. Proceed to creating and playing in 20.000 BLOCKS



Figure B.6 – Minecraft home screen. Image credits: the author.

B.2 Get going with 20.000 BLOCKS

To follow these instructions you need to have set up 20.000 BLOCKS properly.

The main principle behind 20.000 BLOCKS combinatorial logic is that you can build a complex shape and a much more simple matching key. If the player models the key during a game and activates it by standing on it, then it gets replaced by the corresponding shape. This is powerful! (need image here)

Create a new World

1. Open the 20.000 BLOCKS Modpack from TechnicLauncher.
2. In the Minecraft Main Menu click on Singleplayer
3. In the next window click on Create New World
4. Give your world a name and set the game mode to Creative.

Optional: Create a superflat world (Recommended)

The default World settings will create a classic, nature-like Minecraft world with random distribution of grass, water, forests, etc. If you want to create a flat world with the white or gray grid textures do the following.

1. When you are creating the new world, click on “More World Options...”
2. Set Generate Structures to OFF, keep Allow Cheats to ON



Figure B.7 – Create a new world. Image credits: the author.



Figure B.8 – Classic, nature-like world in Minecraft. Image credits: the author.

3. set World Type to Superflat and then click on Customize.
4. Click on Presets
5. In the text field paste:
 - (a) for the white grid world: `3;55*minecraft:sandstone;1;`
 - (b) for the gray grid world: `3;55*minecraft:snow;1;`

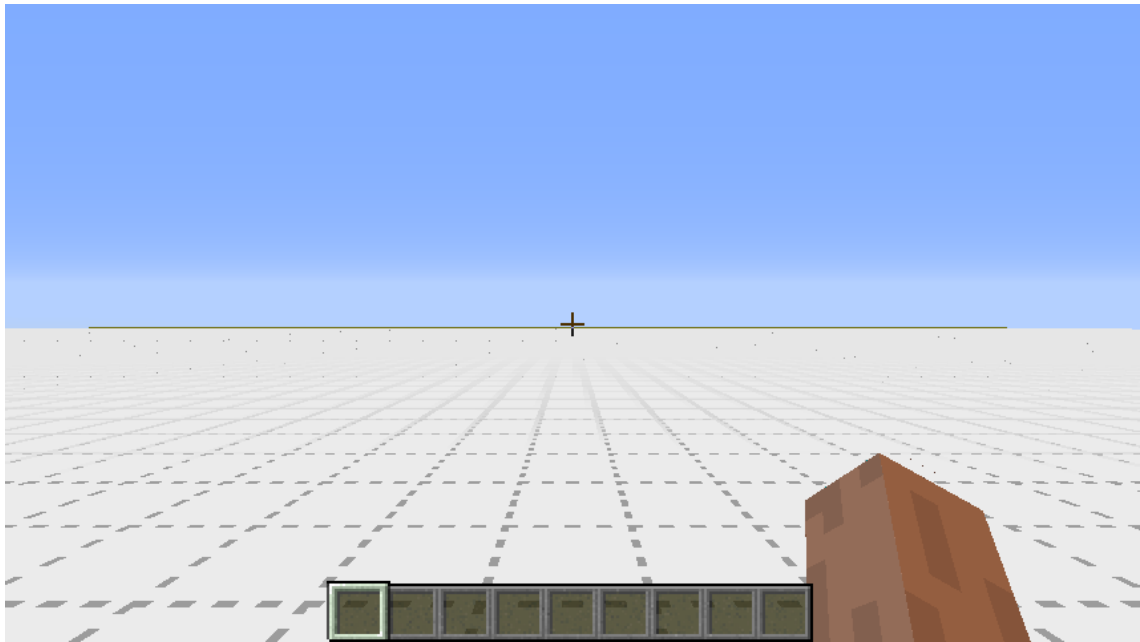


Figure B.9 – Superflat world with white grid. Image credits: the author.



Figure B.10 – Superflat world with gray grid. Image credits: the author.

Tip: To create custom presets you can use this superflat world preset generator:
<http://www.minecraft101.net/superflat/>

6. Click on Use this Preset.
7. Click Done
8. Click Create New World to start
9. Once you are in the world press Escape, click on Options... and then set Difficulty to Peaceful

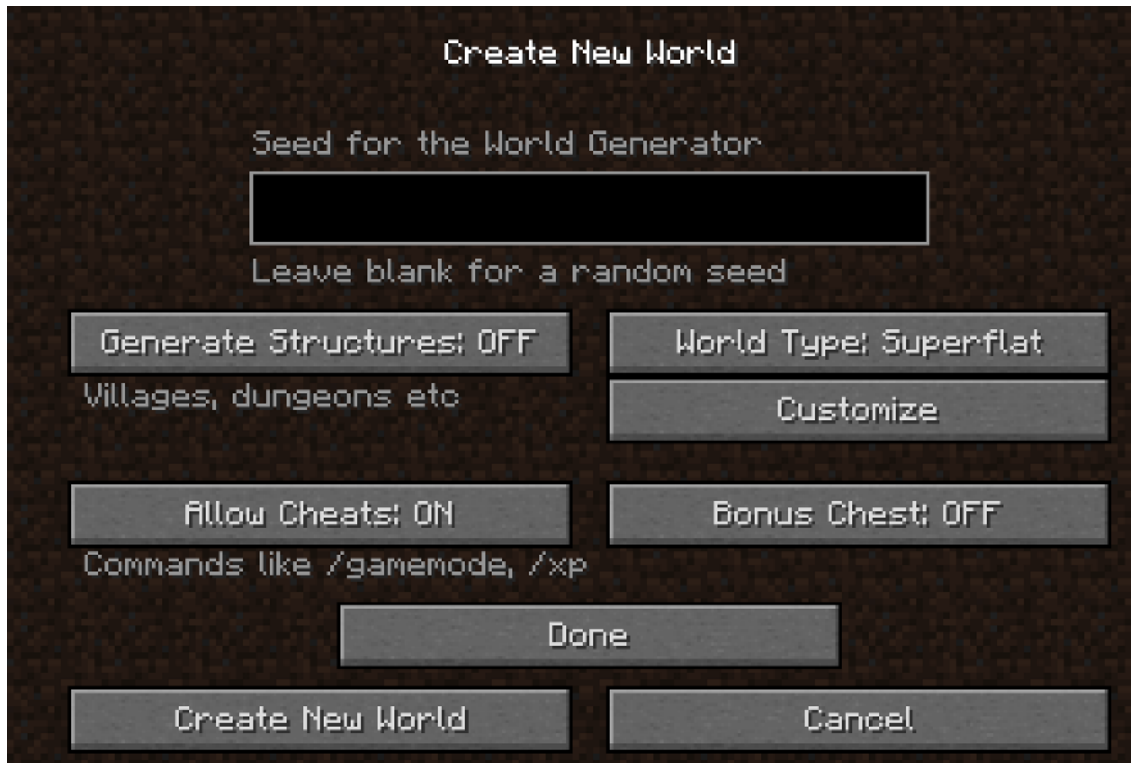


Figure B.11 – World settings. Image credits: the author.



Figure B.12 – Set the difficulty level to peaceful. Image credits: the author.

Build a Command Computer to run 20.000 BLOCKS

1. Press 'e' to open inventory.
2. Click on the Compass.
3. Search for 'Command Computer'.
4. Click on the command computer and move it to your hotbar.
5. Press Esc to close the inventory.
6. Place the Command Computer using the Right Mouse button. Hint: Destroy items with the Left Mouse button.
7. Disable commandBlock output with the following command:

```
/gamerule commandBlockOutput false
```



Figure B.13 – Command computer in the inventory. Image credits: the author.

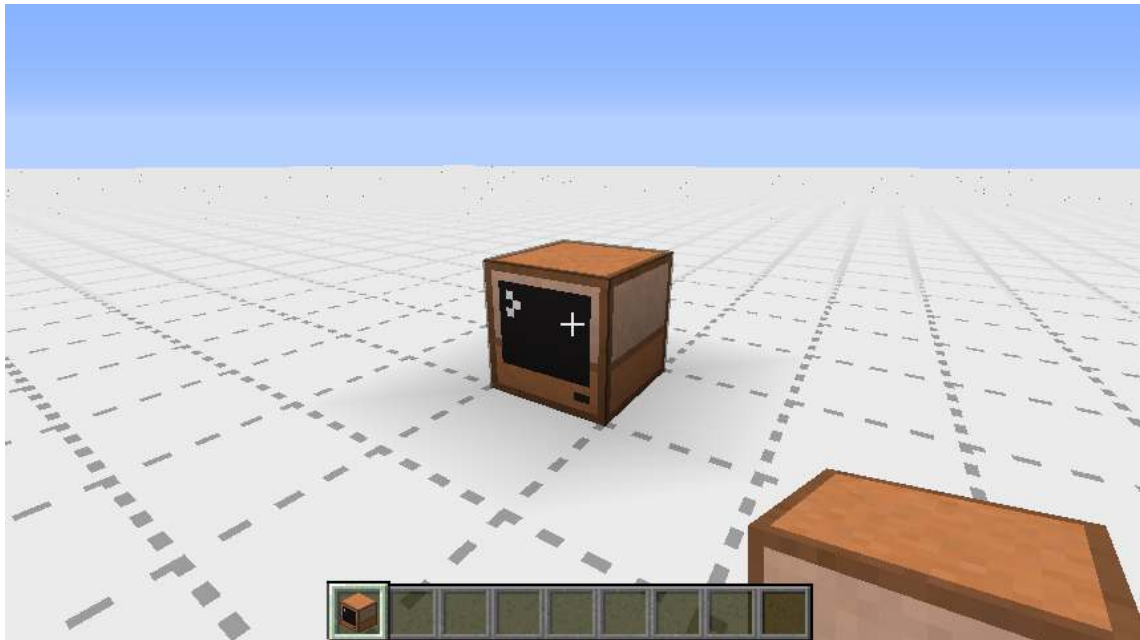


Figure B.14 – A placed command computer. Image credits: the author.

Install the 20.000 BLOCKS on the Command Computer

1. Right click on the Command Computer to open its Terminal. You see Figure B.15.
2. Run the following commands to install the game scripts (you need to be connected to the internet for this to work):

```
pastebin get VbZ5tvq9 update
```

Tip: on macOS the shortcut for pasting in the terminal of a ComputerCraft computer is CTRL+V, not CMD+V as usual for macOS.

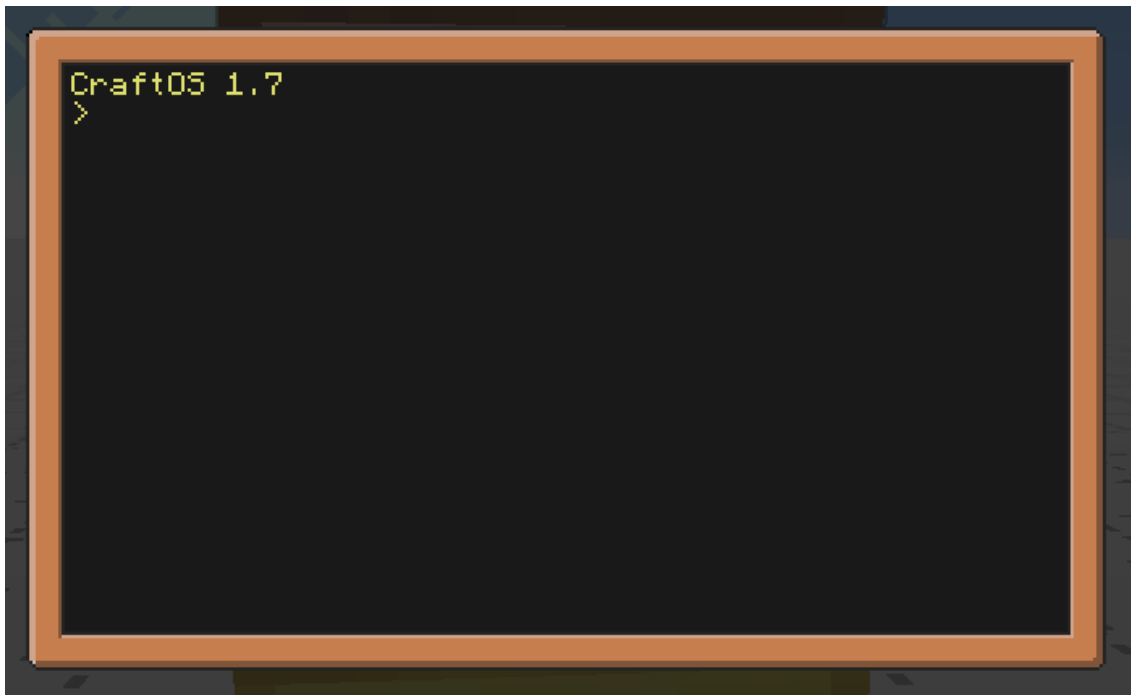


Figure B.15 – CraftOS home screen. Image credits: the author.

Tip: on macOS if shortcuts such as paste (CTRL+V) or select all (CMD+A) don't work, pressing OPTION+SHIFT (ALT+SHIFT) once might solve the problem.

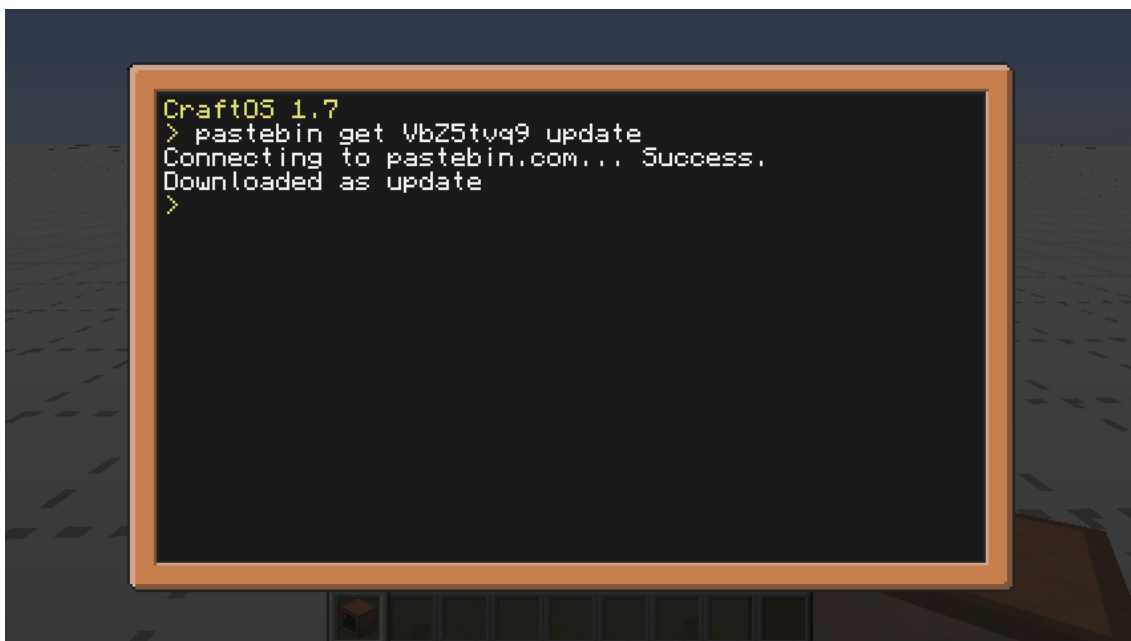
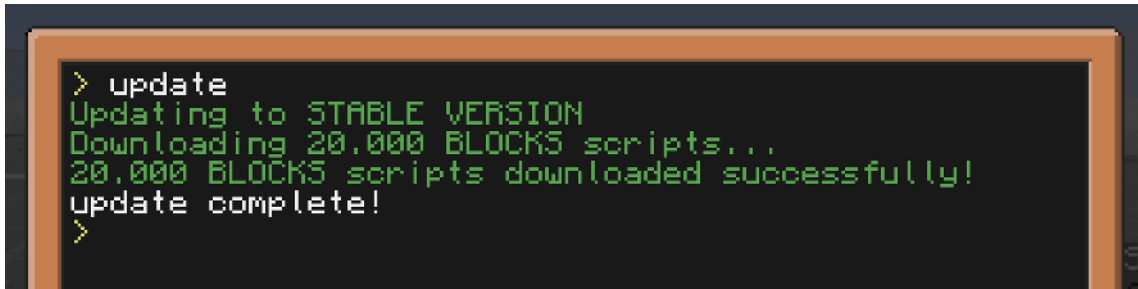


Figure B.16 – Load the 20.000 BLOCKS scripts from Pastebin. Image credits: the author.

3. Then run update

All game scripts from the current stable version should download to the computer and the computer will reboot.



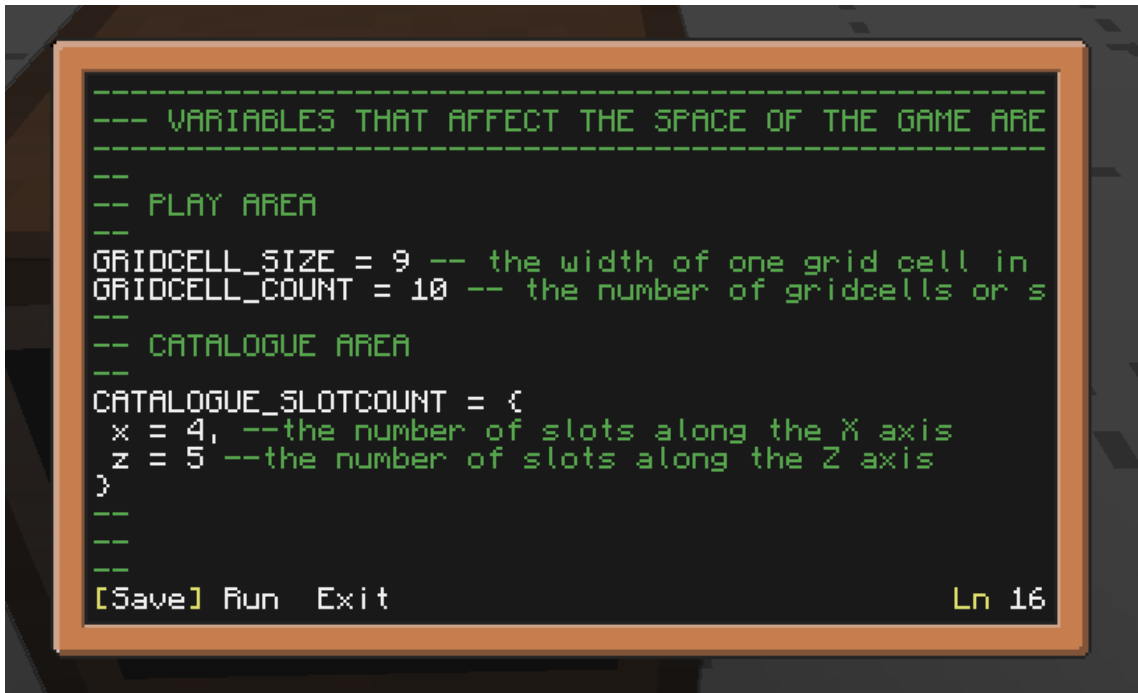
```
> update
Updating to STABLE VERSION
Downloading 20.000 BLOCKS scripts...
20.000 BLOCKS scripts downloaded successfully!
update complete!
>
```

Figure B.17 – Update scripts. Image credits: the author.

Tip: if the update command or later the play command hangs, press and hold CTRL+T to terminate the program.

Editing the project settings (Catalog size, grid size, etc.)

- Run `edit project_settings`



```
-----
--- VARIABLES THAT AFFECT THE SPACE OF THE GAME ARE
-----
--
-- PLAY AREA
--
GRIDCELL_SIZE = 9 -- the width of one grid cell in
GRIDCELL_COUNT = 10 -- the number of gridcells or s
--
-- CATALOGUE AREA
--
CATALOGUE_SLOTCOUNT = {
  x = 4, --the number of slots along the X axis
  z = 5 --the number of slots along the Z axis
}
--
--
--
[Save] Run Exit Ln 16
```

Figure B.18 – 20.000 BLOCKS project settings. Image credits: the author.

There are four values that can to be changed.

1. The first one `gridcell_size` means the size of your element, i.e. 9 means the maximal size of your element is 9*9 blocks.
2. The second one `gridcell_count` means the size of your buildzone (the number of cells along each axis).
3. The last two values `catalog_slotcount` define the size of your catalog. X means the number of elements along the x axis, and z means the number of elements along the z axis.

After changing, press ctrl choose Save and press Enter, then press ctrl choose Exit and press Enter.

- Run play to start the game with the new settings

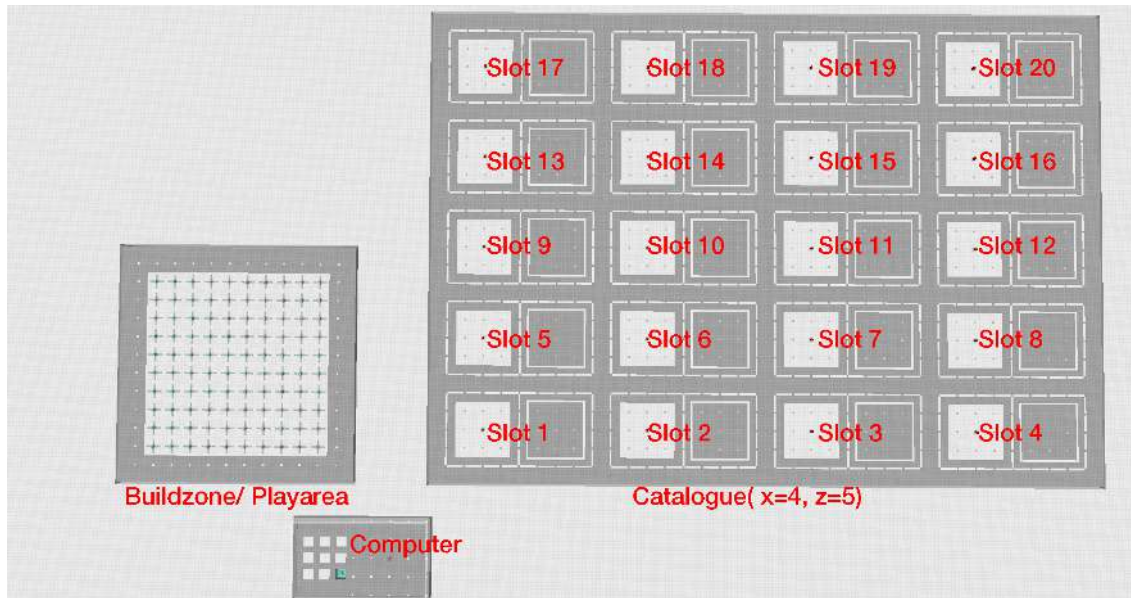


Figure B.19 – The zones in the game area. Image credits: the author.

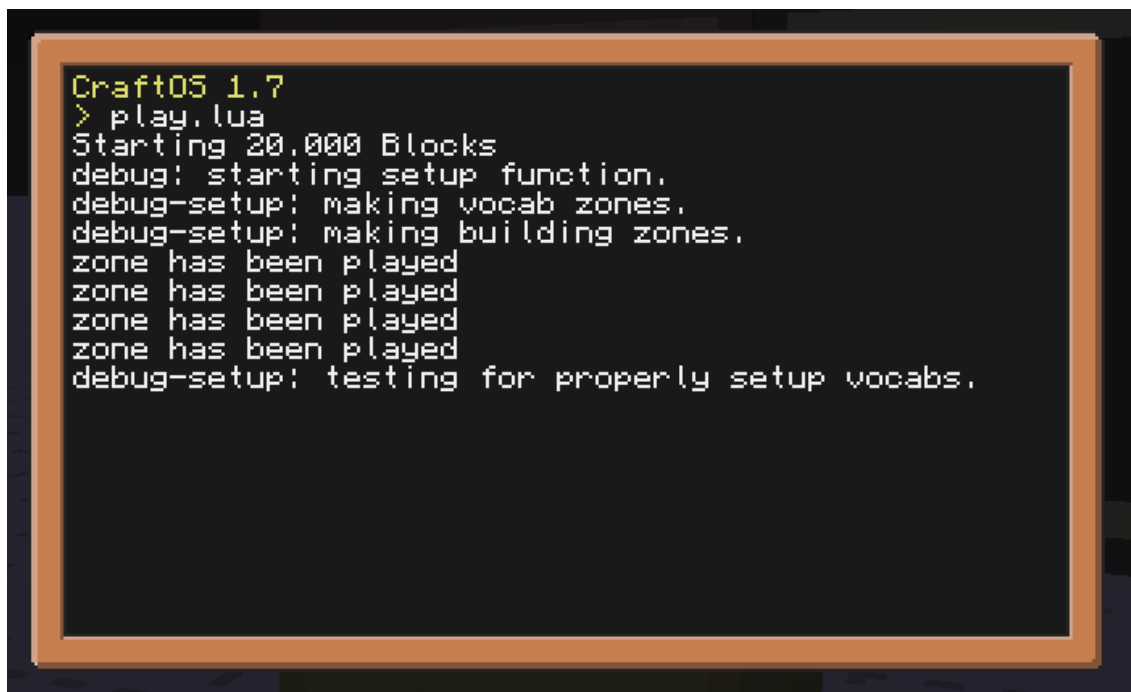


Figure B.20 – Start the game. Image credits: the author.

Game components generation

It will take about 45 seconds to place all of the required game components.

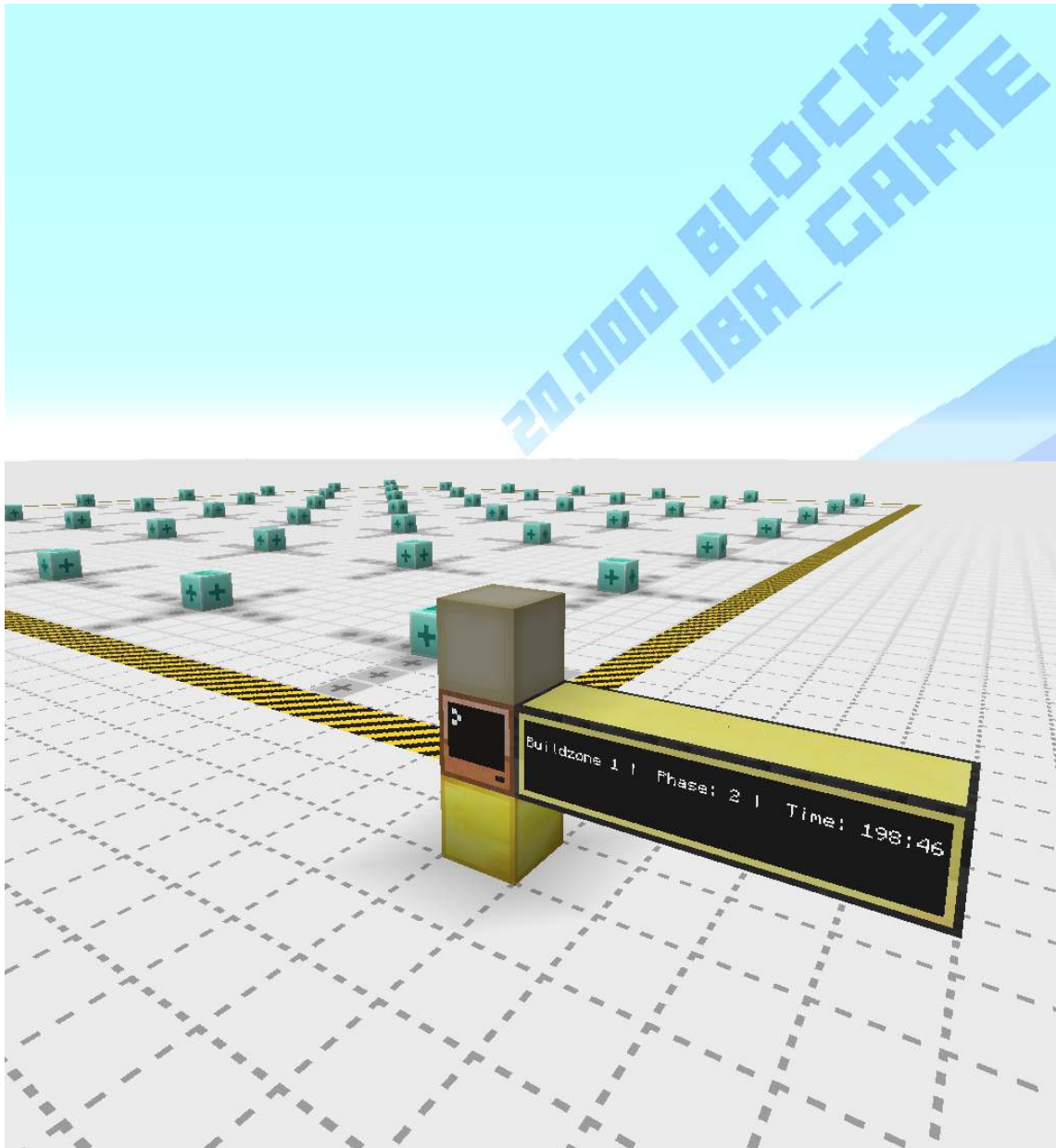


Figure B.21 – A freshly started game script with the buildzone. Image credits: the author.

Build and Test the Catalog

Place a simple test element in one of the slots in the catalog. Be sure to remove the Obsidian block only from the catalog slots that you intend to use. The script is designed to ignore any slot which still has the Obsidian block in the center.

The white area is for the Key (what the player must build) and the gray area for the element, which will be generated when player places and activates the key in the Play Area later.

Once you have made this, go back to the Play Area and build the Key Shape. Stand on the Activator Block and then the element will be generated around where the Activator Block was.

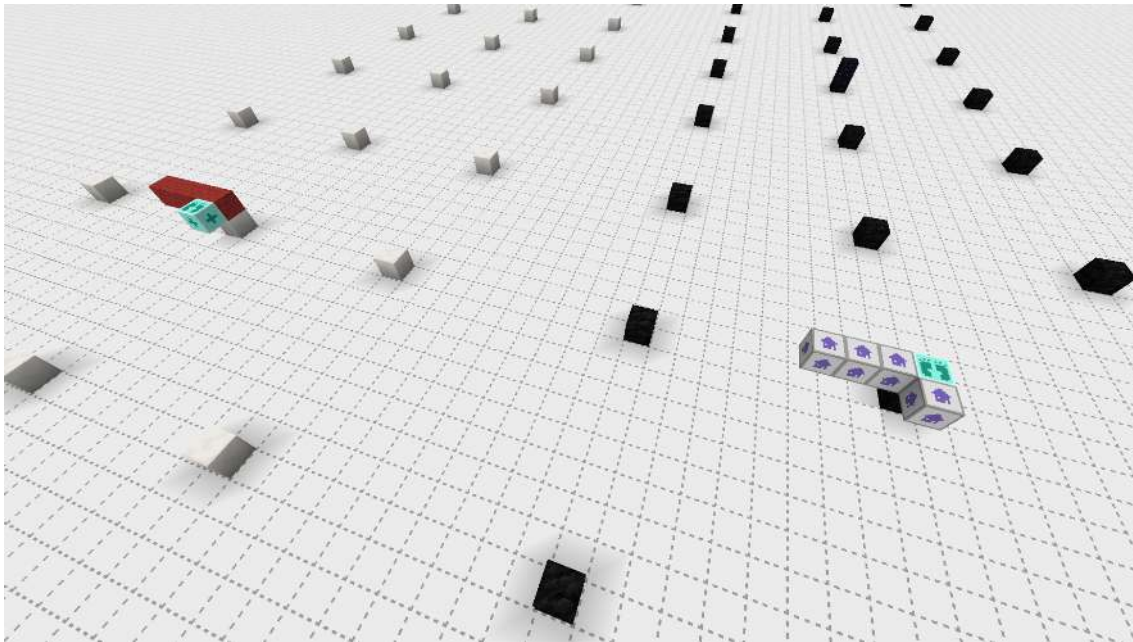


Figure B.22 – The catalogue building area. Image credits: the author.

Open to LAN

You can open your game to others on the network. Press escape and click the Open to lan option. If you want other players to only build in the Play Area then set the gamemode to Adventure. Keep cheats enabled only if you trust those you are inviting to your server. If you would like to have help building vocabulary then set the gamemode to Creative.

Reset Play Area

1. Go to the computer and enter it
2. Press and hold CTRL + T
3. The running script will terminate
4. Run `play.lua true`
5. The game will start again and the play area will be cleaned

B.3 Tips for the Catalog

1. You can use any blocks in your creations.
2. Only place new Detector Blocks in line with the existing Detection Grid
3. In the Element Zone of each slot, place the smallest amount of blocks to prove that the elements vocabulary will fit together before decorating. Framework -> Massing Model -> Facades is a good way to build, testing many times between each step.

4. Barrier block (11.24.2017 updated)

- What is a barrier block  ?

A barrier block displays as a red box with a slash through it. They are visible only when you hold one in your hand, otherwise they stay invisible. You can imagine them as solid air: they are transparent to light and they has solid volume, so you can place them, break them but you can not go through them.

- How to get barrier blocks?

Barrier blocks are not in the inventory. To get them or give them to your group members, you need to be an op and use the following command:

```
/give <name> barrier
```

<name> is your name or your group member's name in Minecraft. Using the first letter + "tab" can save you time for entering names. If you do not get barrier block in your hand, press "E" to find it in your survival inventory which is at the lower right corner.

- How is barrier block useful in 20.000 BLOCKS?

When the element does not replace all the blocks of the key which generated it, the barrier block is very useful to "delete" the key, to make sure the buildzone is clean.

For example, in Figure B.23 and Figure B.24 the key still exists after it was detected. As a result, we want the element while the key is no longer useful.

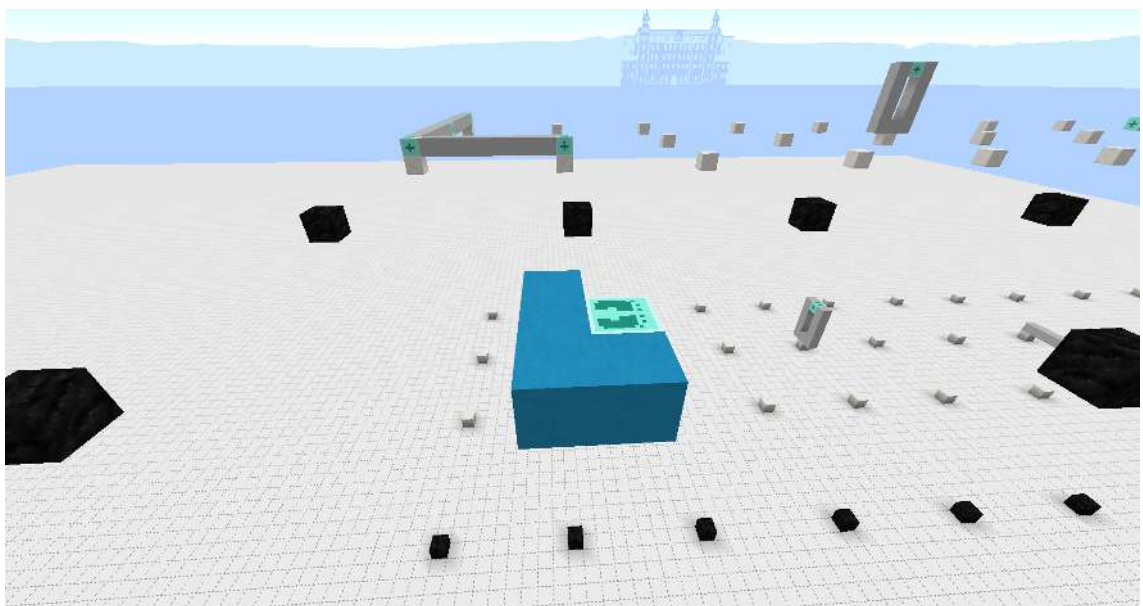


Figure B.23 – Key. Image credits: the author.

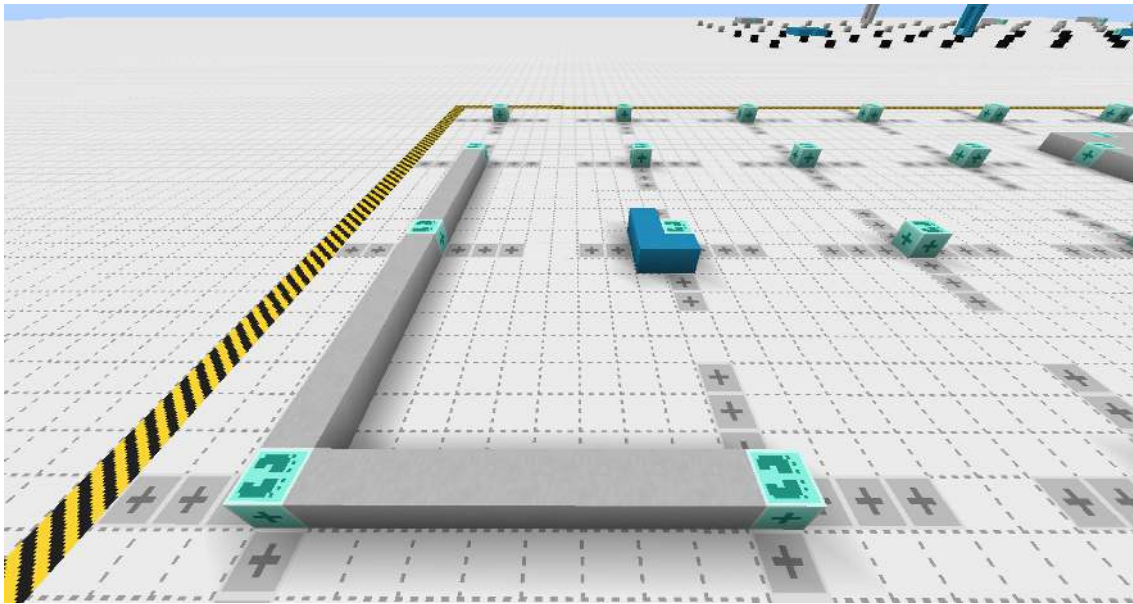


Figure B.24 – Element is placed but key remains. Image credits: the author.



Figure B.25 – An element with barrier blocks to erase its key. Image credits: the author.

In this case, barrier blocks can be placed into the catalog, to replace the key with air, so as it is shown in Figure B.25 and Figure B.26, the key disappears after being detected.

5. Fill command

`/fill X1 Y1 Z1 X2 Y2 Z2 Blockname`

In Minecraft X,Y,Z is used to locate a player or a block. X(red axis) and Z(blue axis) is horizontal, Y(green axis) is vertical. This command is to fill the blocks from (X1,Y1,Z1) to (X2, Y2, Z2) with a new block, no matter if they are occupied or not.

You can try the following command and see how it works.

`/fill ~ ~ ~ ~4 ~ ~-4 snow`

~ (tilde) means your current location. In the command above, the first ~ means your X, the second one your Y and the third - your Z. The fourth value ~4 means

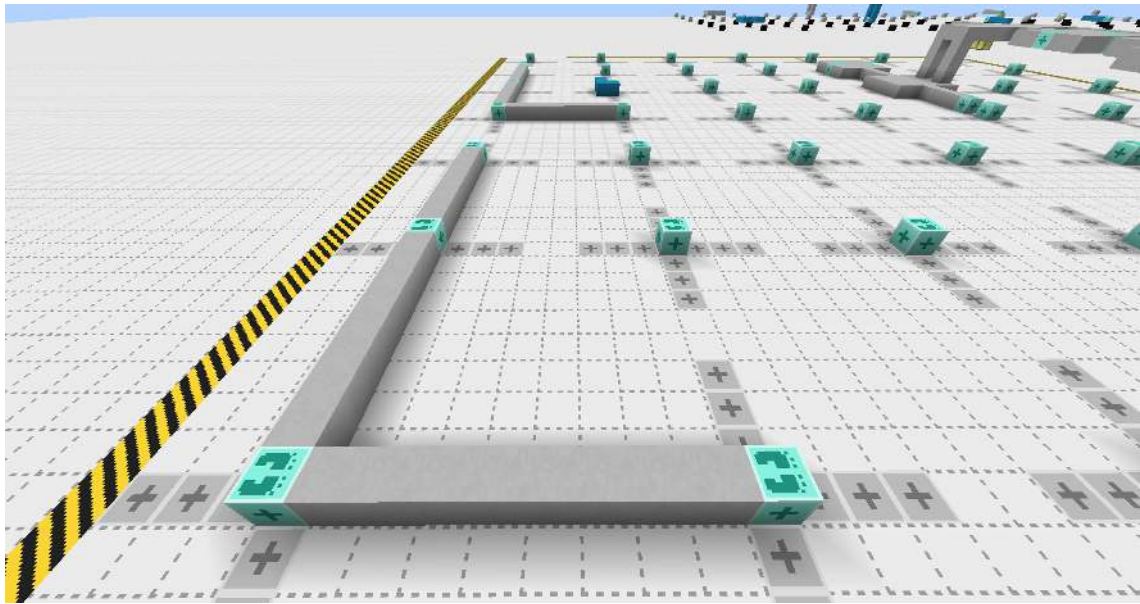


Figure B.26 – The same element placed, the is removed upon placing. Image credits: the author.

your X plus 4. The sixth value `~-4` means your Z minus 4. It counts your current location as beginning so a cube of 5x5x1 blocks is filled.

If you want to replace a certain type of blocks with a new type, the replace option is quite useful.

```
/fill X1 Y1 Z1 X2 Y2 Z2 <newblock_name> <newblock_tynumber> replace
<oldblock_name>
```

- Press F3 and point at any block, the name of the block will be shown in the top right corner.

You can try the following commands and see how it works.

```
/fill ~ ~ ~ ~7 ~1 ~7 minecraft:stained_hardened_clay 1 replace air
```

```
/fill ~ ~ ~ ~4 ~ ~4 minecraft:gold_block 0
replace minecraft:stained_hardened_clay
```

- Use the Tab key for autocompletion of a material's name as you type.
- The new type number (in this case 0 and 1) is always needed, you can google the one you need or find it in Minecraft Wiki or Minecraftinfo.

B.3.1 Making your catalog as compact as possible

The smallest Area that your catalog takes up the faster the game loads and runs. Here are few tips to move things around so you reduce the area of the catalog.

B.3.2 Swapping catalog elements from one slot to another

1. First update your game by typing the following three command in the computer:

-
- (a) `delete setup_2.5` (in case it already exists)
 - (b) `pastebin get FcGRrZgr setup_2.5`
 - (c) `setup_2.5`
2. Then run the game once typing “play”.
 3. When it loads hold CTRL-T to stop it
 4. Go to your catalog area and you should see numbers for each slot

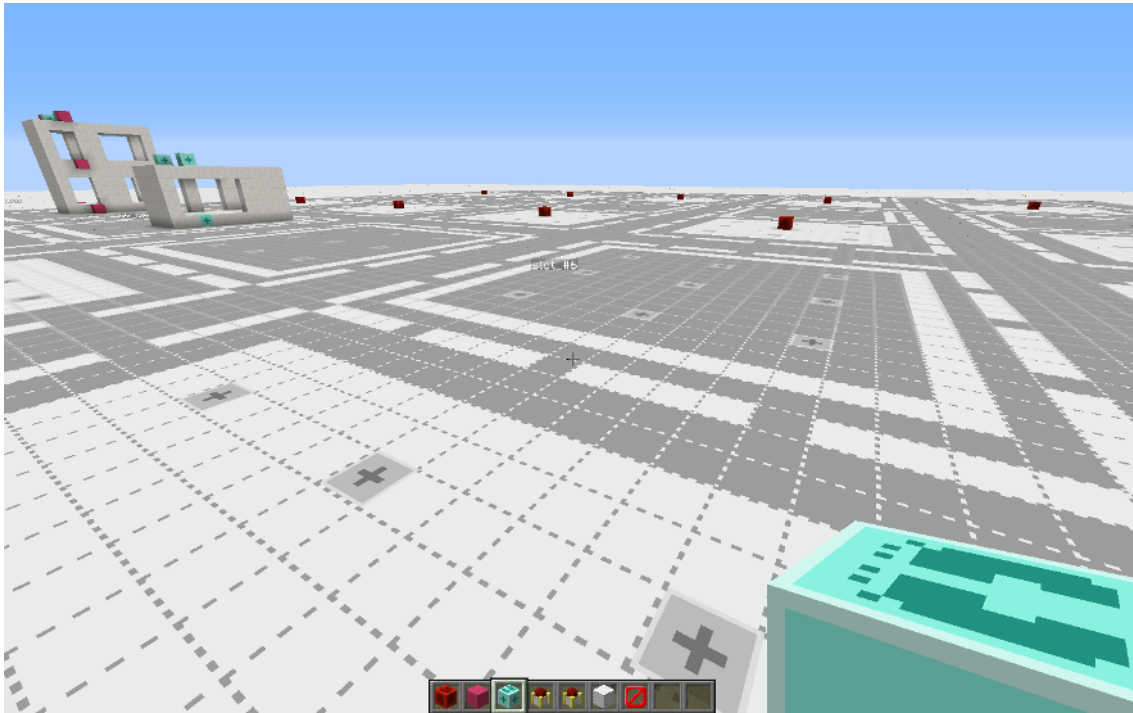


Figure B.27 – Slot number. Slot #6. Image credits: the author.

5. Note the numbers of two slots you want to swap. In this example 6 and 14
6. Then go back to the computer and type:
`swap_slots <number_of_slot_1> <number_of_slot_2>`

Done

Reducing the number of slots in the catalog

In the example the catalog has 4x5 slots but only 2x3 are used. It will speed up the game if we change that in the settings

1. Go to the computer and type: `edit settings`
2. The settings file will open. Scroll to find the “`catalogue_slot_count`” and change it to your desired values. In the example here we are changing from x=4 and z=5 to x=2 and z=3



Figure B.28 – Another slot number. Slot #14. Image credits: the author.

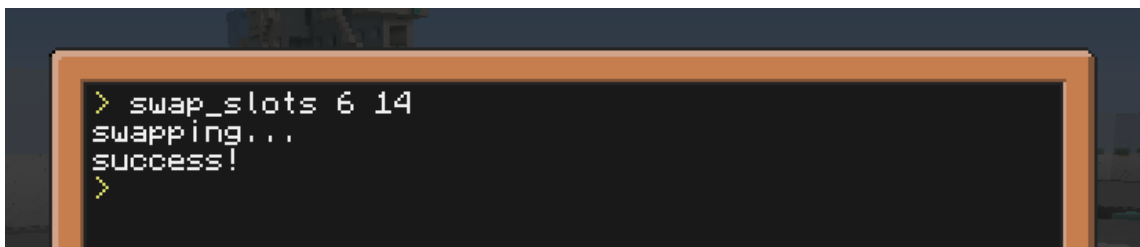


Figure B.29 – Swap slots 6 and 14. Image credits: the author.

3. press CTRL key and choose SAVE and press ENTER
4. press CTRL key and choose EXIT and press ENTER then press ESC to close the computer console
5. Run your game again with “play”
6. You will notice a thin, one-block-wide gap where the new border of the active catalog is drawn



Figure B.30 – Catalog settings. Image credits: the author.



Figure B.31 – The border of the new catalog area. Image credits: the author.

B.4 Changing the game rules

B.4.1 Setting the items that players get at the start of a game

1. Check which items you want to give to players at the start of a game. Usu-

ally those are the materials which you used to define the key shapes in your catalog. Go to the key area and with F3 toggle the display of debug info, point to a material and note its name and variant: in the example the name is “stained_hardened_clay” and the variant is “cyan”.



Figure B.32 – Use the Debug info mode to see material names. Image credits: the author.

2. Find the number corresponding to the variant from the minecraft.gamepedia.com. In this example “cyan” is 9
3. Go to the command computer and type “edit starting items” (NB: the game must have been started at least once before for that file to exist. If you haven’t started it then use “play” to do so)
4. By default the starting items are two materials giving 16 blocks each (the left picture below). Edit the file with the material information you collected in steps 1. and 2. In the example we change the first material name from “wool” to “stained_hardened_clay” and the variant from 0 to 9 (the picture on the right). We also delete the information for the second item including the separating commas and enclosing curly brackets.
5. Then press CTRL and choose “Save”. Press CTRL again and choose “Exit”
6. Run the game with “play” and go to a portal to test if the items you are given at the start are those you need.

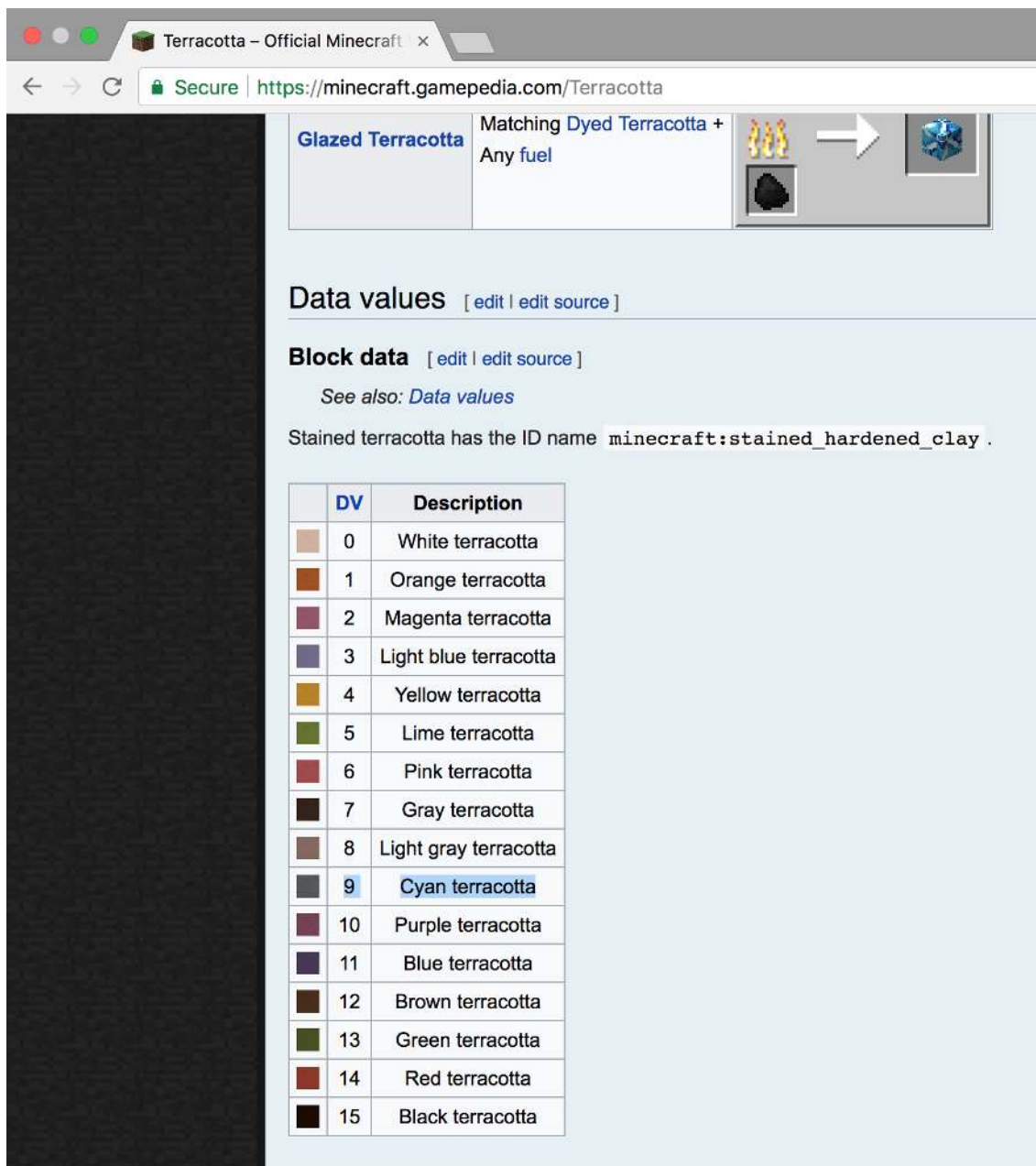


Figure B.33 – Online list of Minecraft materials. Image credits: Minecraft Gamepedia.

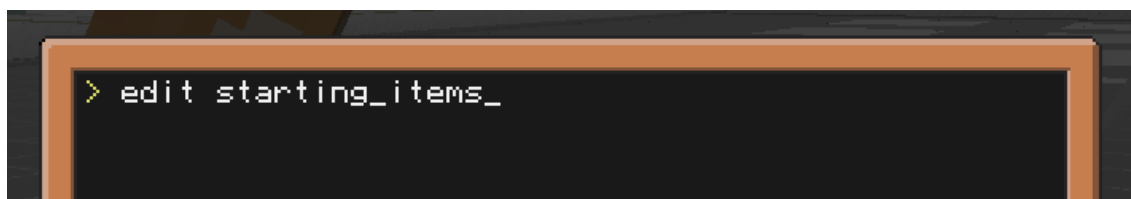


Figure B.34 – Edit starting items. Image credits: the author.

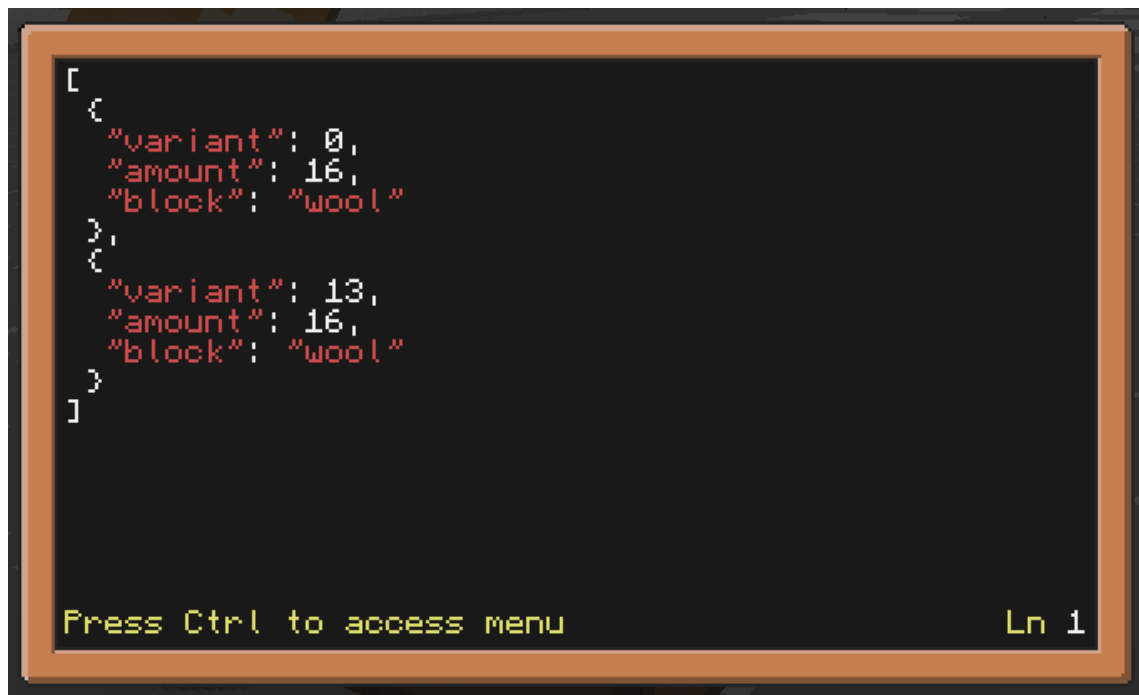


Figure B.35 – Default starting items settings. Image credits: the author.

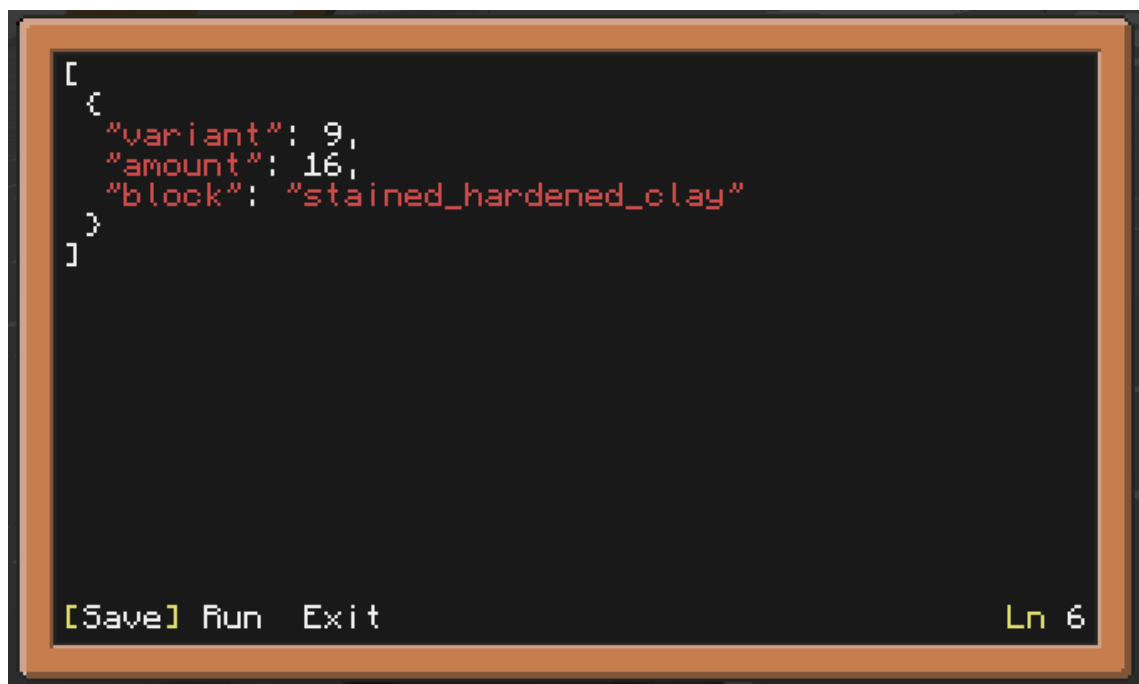


Figure B.36 – Modified starting items settings. Image credits: the author.

B.5 Playing

B.5.1 Starting a game as a player

1. Go to one of the active portals on the side of the computer. The portals are the 9 squares on the ground (3x3). If a portal is active it will be represented with blue activator blocks.
2. Step on the portal and you will be:
 - (a) automatically teleported to its corresponding buildzone,
 - (b) changed to adventure mode
 - (c) and given the starting items defined for this game.

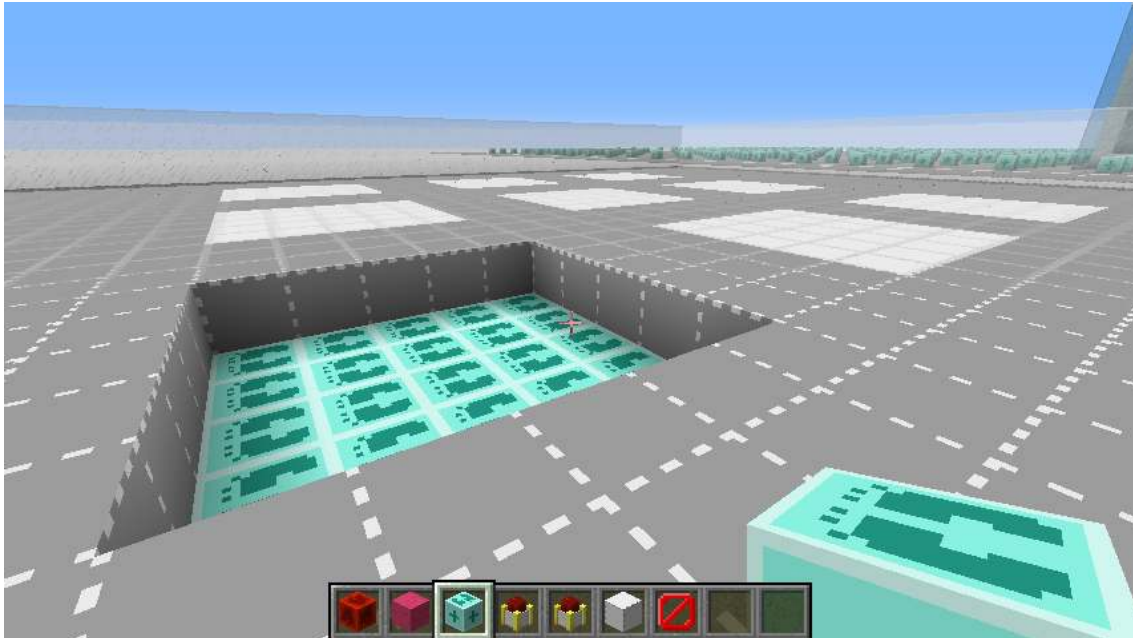


Figure B.37 – A portal to start a new game. Image credits: the author.

B.5.2 Returning to the spawn area from a game in progress

Throw the egg which is named “HOMECOMER” on the ground in front of you.

You will be automatically teleported back to the spawn area.



Figure B.38 – The Homecomer. Image credits: the author.

B.6 Setting up a your own 20.000 BLOCKS Server

B.6.1 Installing the Server Files

The first step is to visit the Minecraft forge website and download the Latest Minecraft Forge Installer package for Minecraft 1.8.9. This link will take you directly to the suitable versions: https://files.minecraftforge.net/maven/net/minecraftforge/forge/index_1.8.9.html

It does not matter where you download the file. Once it is finished, run the installer file.

Select “Install Server” and choose a folder using the button with the three dots. Then click OK.



Figure B.39 – Forge installation. Image credits: the author.

Once it is complete, navigate to the folder where you installed the server and run the `minecraft_forge.jar` file. The server will run but then immediately close.

A new file, called `EULA.txt`, should have appeared in the folder. Open it and change the word “true” to “false” to agree to the Minecraft End User License Agreement.

Once again, run the `minecraft_forge.jar` file. The server will start up this time, and many more files will be placed in your server folder.

A new window will open:

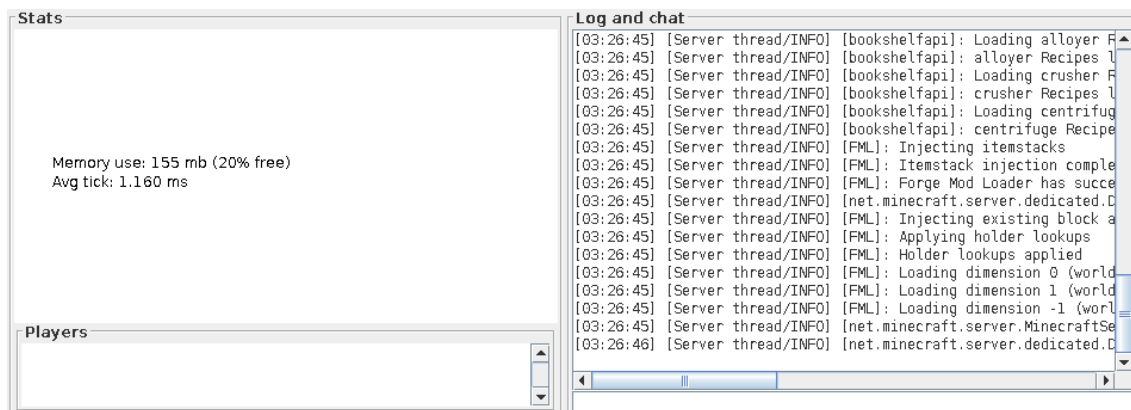


Figure B.40 – Server console. Image credits: the author.

This is the server console.

The first thing you should do is type the command:

`/op [username]`

Replacing [username] with the Minecraft account name you want to give Operator and Admin privileges.

NOTE: DO NOT GIVE UNTRUSTED ACCOUNTS THESE PRIVILEGES!

Now type the following command to turn off the server:

`/stop`

Next we want to install the 20kb Modpack on your new server. Download the Modpack from TechnicLauncher, or if you already have the modpack installed on your computer, you can take it from your Technic Launcher folder. The mods folder in the modpack archive contains the following files:

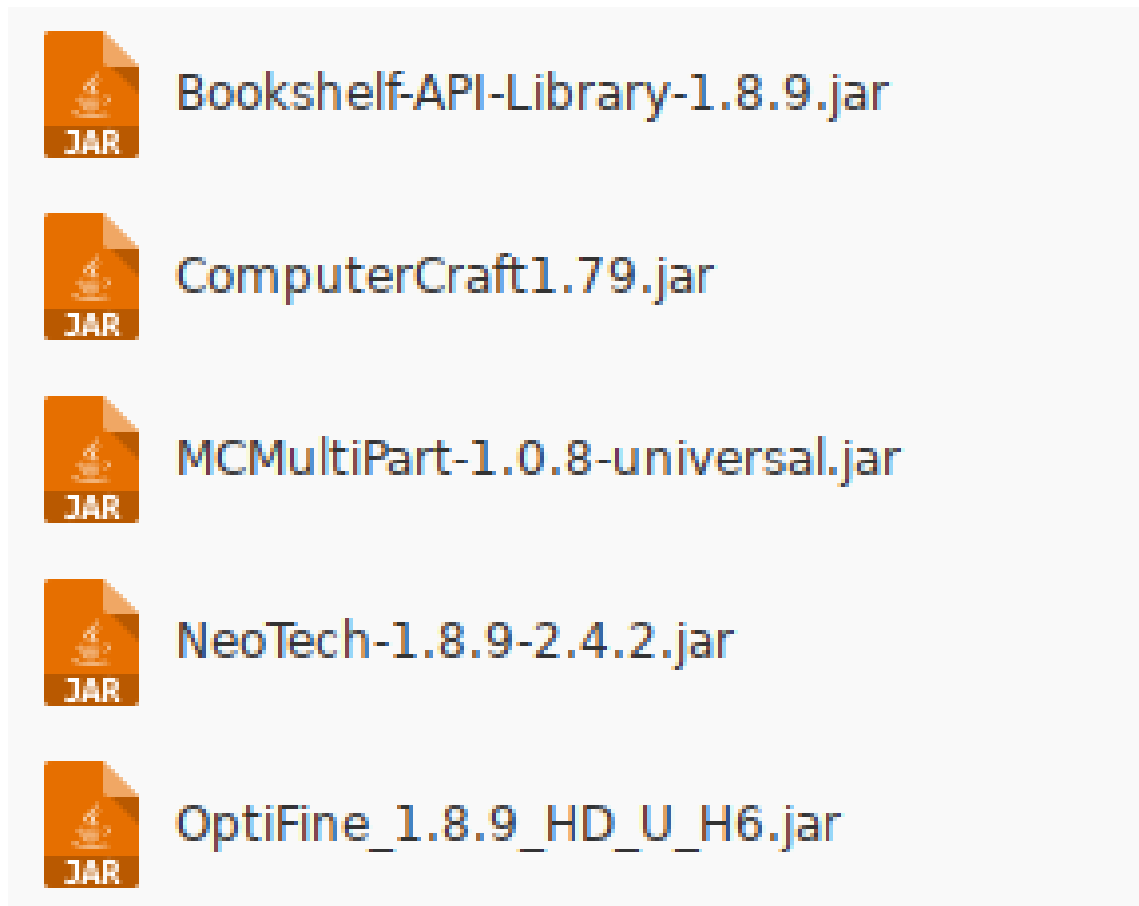


Figure B.41 – Mod jar files. Image credits: the author.

Copy these files into the /mods folder of your server.

B.6.2 Server Settings

Now we want to configure your server. In the server folder find and open the server.properties file in any common text editor software (Notepad, etc.).

There are a lot of setting here but we want to make the following changes:

- Spawn-protection = 0
- Gamemode = 2
- Difficulty = 0

-
- Spawn-monsters = false
 - Pvp = false
 - Generate-structures = false
 - Enable-command-block =true
 - Max-players = 5
 - Motd = Your greeting message

You can increase the number of max players, however it is important to remember that Minecraft uses some extra resources just to keep player slots open. Setting this to a high number will make your server run slow, even when very few players are actually playing.

You can also set a whitelist here, to allow you to control who is able to join your server. Change the following setting if you want a whitelist:

- Whitelist = true

To add players to the whitelist use the following command from either an Operator in game, or the Server Console:

```
/whitelist add [username]
```

B.6.3 Choose a World Type

You now have a few options on how you want your world to look.

Use the demo world

If you have downloaded our demo world, you can use it on the server. First make sure your server is off by using this command:

```
/stop
```

Then navigate to the /world folder of your server, and delete all the contents.

Now copy the contents of the demo world folder into your /world folder of your server.

Start your server and you will see the demo world is running.

Use a Survival World

A survival world is a very picturesque place, and can make a good home for your 20,000 BLOCKS players.

By default your server will create a survival world.

Explore your world and find a place where you wish to set up your 20,000 BLOCKS Computer as was shown above.

Be aware that some blocks which have special effects and textures in 20,000 BLOCKS will spawn naturally in a Survival world. This means that some areas of the world will look quite strange.

Use a Flat world

To use a flatworld, make the following changes to your server.properties file:

- level-type=FLAT
- level-name=flatworld
- generator-settings=3;1*minecraft\:bedrock,4*minecraft\:sandstone;0;

Then navigate to your server folder and delete the /world folder. When you start the server, a new folder called /flatworld will be created.

The Generator Settings option allows you to customize what your flatworld is made of. Visit this website to create your own style: <http://www.minecraft101.net/superflat/>

B.6.4 Joining your server

If you are joining the server from the same computer, you can access it by Adding a Server from the Minecraft Multiplayer server menu like so:



Figure B.42 – Joining a server. Image credits: the author.

If you are joining from a different computer then you will need to know the IP address of the server, and use that address instead of “localhost”.

Finding your IP address is different on every server type. Consult a search engine to discover the best way to find your IP address.

B.6.5 Inviting others to Join your 20.000 BLOCKS server

20.000 BLOCKS is designed for multiplayer! Here are some tips and tricks for having others in your server.

What info you need to send to people who want to join your server

- IP address
- This Manual how to install and setup 20.000 BLOCKS

B.6.6 Understanding Game Modes

Minecraft has 4 different game modes. They each have different uses in 20.000 BLOCKS. You can change a player's gamemode using these commands:

```
/gamemode survival [playername]
```

```
/gamemode creative [playername]
```

```
/gamemode spectator [playername]
```

```
/gamemode adventure [playername]
```

Adventure

Adventure mode players are bound by strict block placement and breaking rules, which are the foundation of 20.000 BLOCKS game logic.

This game mode will be used by the majority of your players. Any player you want to be building in the Campaign area and not modifying the Catalog should be in Adventure mode. New players joining your server will be set to Adventure mode by default.

Creative

This game mode is best used when building and testing your Catalog. Creative mode users have unlimited blocks of all types, can fly and instantly destroy (not gather) blocks they hit.

Creative mode users are also invincible, except to other Creative mode users who are carrying swords.

It is inadvisable to grant untrusted players Creative mode status.

Spectator

This mode is good for making videos or taking screenshots of your creations. Spectators can fly through walls and track players, but cannot interact with the game world.

Spectator mode can also be an effective way to control unwanted or destructive players, as they can do no damage while in this game mode.

Spectators are also unable to communicate with other players through text chat.

Survival

Survival gamemode is the main way to play vanilla Minecraft. We do not recommend using 20.000 BLOCKS in this gamemode as it allows players to circumvent much of the special game logic that 20.000 BLOCKS is built around.

B.6.7 Teleporting Players

If players become lost any Operator player may teleport them back using this command:

```
/tp [player_from] [player_to]
```

Or you can give absolute coordinates:

```
/tp [player] X Y Z
```

If you yourself become lost, you can return to the world spawn by using this command:

```
/kill [player]
```

B.6.8 Setting the World Spawn

If you have selected a location for your Buildzone that is not near the default spawn area, you can set it by using this command:

```
/setworldspawn
```

You can also set a player's individual spawnpoint using this command

```
/spawnpoint [player]
```

Which will set their spawn to their current location.

B.6.9 Dealing with “those” Players

If players become destructive or unruly, it is always better to first request that they stop, and warn them of possible consequences first. Escalate slowly and try to deal with these players as nicely as possible. If they do not listen you can kick them with this simple command:

```
/kick [player] [reason]
```

This does not tend to be very effective, as they may immediately rejoin the server.

The best way is instead to set them to spectator mode, which mutes their text chat and stops them damaging any blocks. Let them cool off and change them back if they agree to settle down.

```
/gamemode spectator [username]
```

If these players repeatedly break the rules, it is better to ban them with this command:

```
/ban [player] [reason]
```

Remember that it is commonly communicated through Youtube and other influences that it is “ok to grief players if you don’t get caught”. The Minecraft community culture includes a lot of “pranking” which borders on terrorism. This does not excuse these actions and we suggest that on your servers you maintain a “No Griefing, No Trolling, No Pranking” policy.

B.7 Taking screenshots

B.7.1 How to take a screenshot in Minecraft and where to find it?

Press “F2” to take a screenshot in game. You can also try press “F1” to hide user interface before taking a screenshot. Or try press “F5” to have a nice group photo.

The screenshots are automatically saved in the “screenshots” folder within the 20000-blocks directory. Press “Esc” in the game and go to “options”==>”resource

packs”==>”open resource folder”, then the “resourcepacks folder” under the 20000-blocks folder is opened.

B.7.2 How to take a screenshot with the same viewpoint?

- Find your certain point and press “F3” to show the debug screen.
- At the top left you can see your location (XYZ) and which direction you are facing (Facing) and remember these five numbers.



Figure B.43 – Using the Debug info view mode to get location and orientation coordinates.
Image credits: the author.

- Use the following command to teleport back to your chosen point:

```
/tp <name> X Y Z Facing1 Facing2
```

If you want to teleport yourself, you do not need to enter <name>. Press “/” and then press “UP” you can get the last commands you just used.

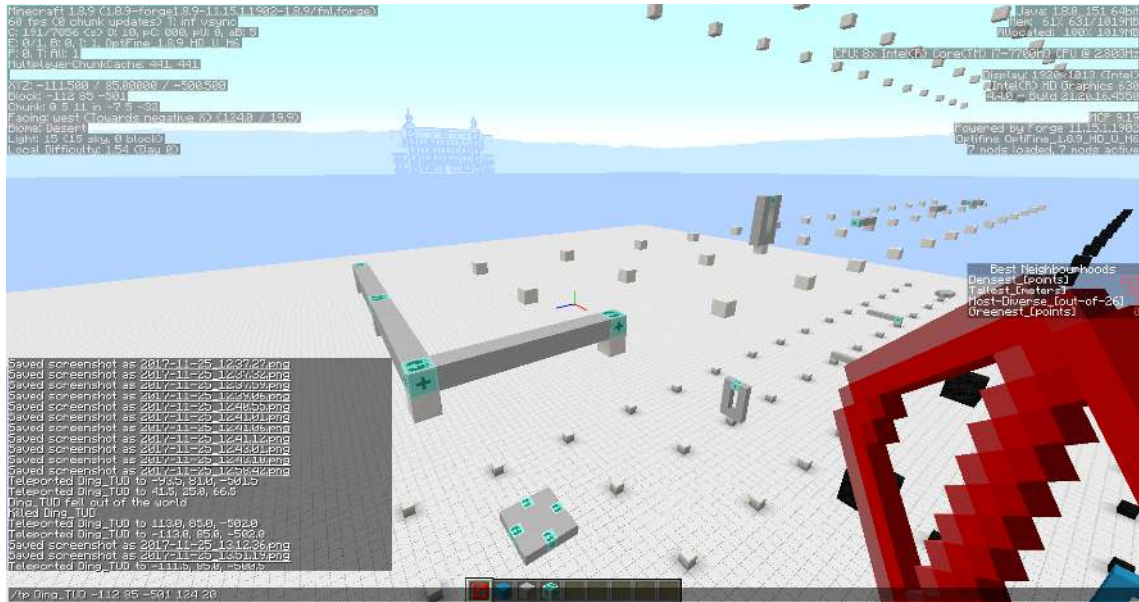


Figure B.44 – Teleport to predefined location and view angles. Image credits: the author.

B.7.3 How to take screenshots with the same viewpoint on a series of objects?

- If you want to take screenshots for each element in your catalog with the same perspective, first you need to find the reference blocks, which should be in relatively certain position for each element. For example the detect block in the key can be the reference block.
- Jump on a reference block and remember your X1 Y1 Z1.



Figure B.45 – Relative positions. Image credits: the author.

- Go to your chosen screenshot point and remember your X2 Y2 Z2 and Facing angles.



Figure B.46 – Adjusted relative position. Image credits: the author.

- Jump on the reference block of the element you want to take photo, press “Space” two times to fly up just one block, then use the following command:

`/tp <name> ~X ~Y ~Z Facing1 Facing2`

(X=X2-X1, Y=Y2-Y1, Z=Z1-Z2. ~ means the current location in game)



Figure B.47 – Teleport command with relative coordinates. Image credits: the author.

Then the player is teleported to the screenshot point (and also flying) of the element.

- Repeating the steps for each element, and takes screenshots for them all.

B.7.4 Making high-res axonometric screenshots using the Mineshot mod

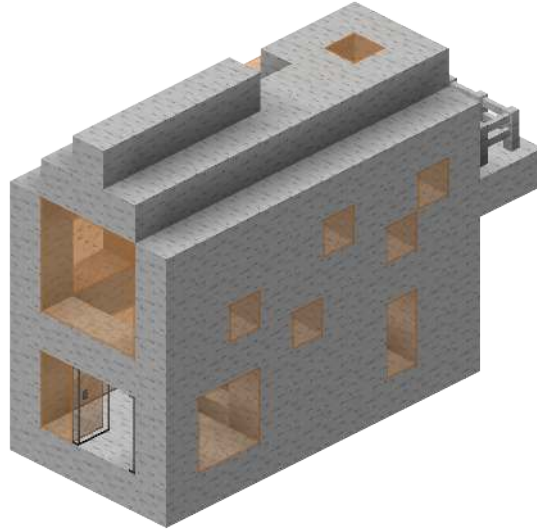


Figure B.48 – Axonometric screen shot. Image credits: the author.

Mineshot adds a build-in orthographic camera, which allows you to create nice high-res isometric screenshots directly in-game (singleplayer only).

Link to get the 1.8.9 version of the mod and instructions how to use it: <http://www.minecraftforum.net/forums/mapping-and-modding-java-edition/minecraft-mods/1282034-mineshot-1-7-high-resolution-screenshot-capturing>

How to install the Mineshot mod:

1. Download Minecraft Forge(for version 1.8.9) <http://file-minecraft.com/minecraft-forge-api-minecraft/>
2. Right click, Run as Administrator and press OK to install Forge. (You can skip this step if you've installed Minecraft Forge)
3. Download Mineshot(for version 1.8.9) <http://www.minecraftforum.net/forums/mapping-and-modding-java-edition/minecraft-mods/1282034-mineshot-1-7-high-resolution-screenshot-capturing>
4. Go to folder ...\.technic\modpacks\20000-blocks\mods. Put the jar file of Mineshot Mod into mods folder.
5. Run 20,000 Blocks and go to “Mod Options” in the main menu, select Mineshot in the mod list and click on “Config”, change the capturing size to fit your screen resolution.
6. Use F9 to save screenshot with Mineshot or use F2 to save normal screenshot.

Try following to get perfect isometric screenshots:

-
- Numpad 5: Switch between perspective and axonometric projection
 - Ctrl + Numpad 5: Switch between fixed and free camera
 - Numpad 4/ 6: Rotate left/ right
 - Numpad 8/ 2: Rotate up/ down
 - Numpad 7/ 1/ 3: Top view/ Front view/ Side view
 - Plus/Minus: Zoom in/Zoom out

List of Tables

3.1	Strategies for encoding design content (by the author)	40
3.2	Existing surveys and reviews of generative design techniques (by the author)	48
3.3	Search phrases for the identified generative techniques (by the author)	56
3.4	Taxonomy ranks with examples (by the author)	59
3.5	Generative techniques rated on their suitability to generate buildings and encode design intent (by the author)	81
3.6	Potential for human-in-the-loop (by the author)	85
3.7	Human-in-the-loop potential of Evolutionary Algorithms (by the author)	90
3.8	Human-in-the-loop potential of Grammar-based techniques (by the author)	92
3.9	Human-in-the-loop potential of Constraint-based techniques (by the author)	95
3.10	Human-in-the-loop potential of Case-based Design techniques (by the author)	97
3.11	Human-in-the-loop potential of Computational Geometry (by the author)	98
3.12	Human-in-the-loop potential of Simulation Approaches (by the author)	100
3.13	Human-in-the-loop potential of Example-driven techniques (by the author)	101
9.1	Ranking (Roger Winkler)	293

List of Figures

1	Project Avocado (by the author)	xi
1.1	Block by Block and Robotic Construction (Block by Block, Leder et al. 2019)	2
1.2	Two types of tools for the non-experts (BARArchitekten, Mr.&Mrs. Homes, Foldit)	4
1.3	The research gap (by the author)	5
1.4	Reddit <i>r/Place</i> (Reddit)	6
1.5	A map of the fields investigated in this thesis (by the author)	7
1.6	The concept of Flow (by the author)	9
1.7	From Anxiety into Flow (by the author)	9
1.8	The Paulson curve (Hendrickson 2008)	11
1.9	Spatial configuration parameters (by the author)	12
1.10	Participant roles (by the author)	13
1.11	Thesis outline (by the author)	15
1.12	Repeat of Figure 1.5 (by the author)	18
2.1	The three strategies for solving wicked problems (the author after Roberts 2001)	22
2.2	The dimensions of participation (Cohen, Uphoff, et al. 1977; Cohen and Uphoff 1980)	23
2.3	The <i>Ladder of Citizen Participation</i> (Arnstein 1969)	24
2.4	The R50 project finished (Heide & von Beckerath)	26
2.5	Formalized participation in R50 (Heinemann 2011)	27
2.6	Open-ended participation in R50 (Heinemann 2011)	28
2.7	Design kits (the author; BARArchitekten)	28
2.8	Play the City, components of the game (Tan 2014)	30
2.9	Play the City, game result and architectural interpretation (Tan 2014)	30
2.10	The Baupiloten Method (Hofmann 2018)	30
2.11	The <i>Negotiating Game for School Development</i> (Baupiloten)	31
2.12	Block by Block (Joakim Formo/Ericsson; Block by Block)	32
2.13	Fès, a non-planned settlement in Morocco (Georg Gerster)	33
3.1	Design systems (Durand, Gropius)	36
3.2	Flatwriter by Yona Friedman (Y. Friedman 1980)	37
3.3	Design process in Flatwriter (Y. Friedman 1980)	37
3.4	Generator (Cedric Price, Frazer 1995)	39
3.5	Online house configurator (<i>MR + MRS HOMES</i> 2020)	41
3.6	Part Assembly (Traumgarten, Lego)	42

3.7	Examples from the <i>Pattern Language</i> (Alexander, Ishikawa and Silverstein 1977)	43
3.8	<i>Taxonomy of Generative Design in Architecture</i> (by the author) . . .	44
3.9	Generative Design (Caetano, Santos and A. Leitão 2020)	46
3.10	Early examples of generative design in planning and architecture (Armour and Buffa 1963, Miller 1970, Negroponte 1970)	46
3.11	Early examples of generative design in computer graphics and games (Tobler 1970, Barton and Loguidice 2009)	47
3.12	Computer-based form generation (Grobman, Yezioro and Capeluto 2009)	50
3.13	Taxonomy of common methods for generating game content (Hendrikx et al. 2013)	51
3.14	Comparison of evolutionary approaches to generative design (Rodrigues 2014)	52
3.15	Levels of algorithmic assistance on designer's actions (Bernal, Haymaker and Eastman 2015)	53
3.16	Assistive generative techniques in games based on the tasks (Yannakakis and Togelius 2015)	53
3.17	The evolution of Shape Grammars ()	54
3.18	The taxonomy ranks (Wikimedia)	59
3.19	The steps in an evolutionary algorithm (Harding 2014)	60
3.20	Fractal-based form generation (Ediz and Çağdaş 2007)	61
3.21	Slice Tree Structure (Kado 1995)	62
3.22	K-d trees used to subdivide layouts (Knecht and Koenig 2010)	62
3.23	Computational Geometry used to subdivide form and layout (Menges 2012)	63
3.24	A custom pipeline of constructive geometric operations to model building massing (Greuter, Parker, et al. 2003)	63
3.25	Generative Modeling Language (Havemann 2005)	64
3.26	Justified plan graphs (J. H. Lee, Ostwald and Gu 2018)	65
3.27	Discursive Shape Grammars (Duarte 2001)	65
3.28	Split Grammars (Wonka et al. 2003)	66
3.29	Boundary Solid Grammars (J. Heisserman 1994; J. A. Heisserman 1990)	66
3.30	Applications of shape grammars (Chau et al. 2004)	67
3.31	Wave Function Collapse (Tigas and Hosmer 2021)	68
3.32	Generating architectural form with 1-D Cellular Automata (Silver 2006) .	69
3.33	Physically-based simulation (Arvin and House 2002)	70
3.34	Floor plan optimization using the Metropolis algorithm (P. Merrell, Schkufza and Koltun 2010)	70
3.35	Topology optimization (Sasaki, Itō and Isozaki 2007; Dapogny et al. 2017)	71
3.36	Design interpretations of topologically optimized voxel fields (Rossi and Tessmann 2017a)	71
3.37	Model Synthesis (P. Merrell 2007.)	72
3.38	Data-driven Generative Machine Learning (Hu et al. 2020)	73
3.39	Case-based Design (CBD), IDIOM (I. Smith, Lottaz and Faltings 1995) .	74
3.40	Case-based design, GPRS (R. Oxman 1990)	75

3.41	An example of case adaptation in CADRE (Dave et al. 1994, Hua and Faltings 1993)	75
3.42	Case adaptation in IDIOM (I. Smith, Stalker and Lottaz 1996)	76
3.43	Contemporary case-based adaptation in practice (Green 2020)	76
3.44	Search-based approaches (Dréo and Candan 2011)	77
3.45	Types of evolutionary algorithms (Yu and Gen 2010)	78
3.46	The Space of possible designs (Gero 1990)	84
3.47	Cell-based user input (D. Anderson et al. 2000)	86
3.48	Requirements and objects lists input (R. Smelik et al. 2010)	87
3.49	Block'hood by Jose Sanchez (Plethora-project)	87
3.50	Types of design evaluation (Liapis, Yannakakis and Togelius 2012) . .	89
3.51	Subjective and objective evaluation in Interactive Genetic Algorithms (IGA) (Quiroz et al. 2009)	90
3.52	Tasks for the user in grammar-based techniques (Chase 2002)	91
3.53	Scenarios for user engagement in grammar-based generative techniques (Chase 2002)	92
3.54	Node-based shape grammar rules specification (Silva et al. 2013; Patow 2012)	93
3.55	Visual, drag-and-drop grammar rule editing (Lipp, Wonka and Wimmer 2008)	93
3.56	Guided procedural generation using Open L-systems (Beneš et al. 2011)	94
3.57	User interaction in Case-based design (Gero 1990)	96
3.58	To guide the design outcome the user changes the environment conditions in simulation generative techniques (Herr and Kvan 2007) . .	99
3.59	Visually introduced disturbance in the environment (Vanegas et al. 2009)	100
3.60	Closed vs open Topologies (Sanchez 2021)	103
3.61	WikiHouse (Parvin 2013)	104
3.62	Informed Wall (Bonswetch 2006)	104
3.63	The BUGA Wood Pavilion by ICD Stuttgart (Wagner, Alvarez, Kyjanek, et al. 2020)	105
3.64	Modularization of a parametric model (Bianconi, Filippucci and Buffi 2019)	105
3.65	Parametric fabrication-aware scaled model of towers (Budig, Lim and Petrovic 2014)	105
3.66	Project Avocado (by the author)	106
3.67	Digital materials (W. Langford, Ghassaei and N. Gershenfeld 2016) .	107
3.68	Discrete design (Retsin 2020)	107
3.69	Robotic discreet assemblies (Tessmann and Rossi 2019)	107
3.70	The selected generative techniques (by the author)	108
4.1	ReCaptcha (ReCaptcha)	109
4.2	Micro-task crowdsourcing model (by the author)	112
4.3	Steps in a crowdsourcing framework (by the author)	112
4.4	Threadless (Threadless)	113
4.5	Galaxy Zoo (Lintott et al. 2008)	114
4.6	Foldit (Foldit)	115
4.7	Wikipedia Content curation (Mark Fidelman)	116

4.8	Reddit <i>r/Place</i> (Reddit)	117
4.9	BuildTheEarth interactive map (BuildTheEarth)	119
4.10	BuildTheEarth in Minecraft (BuildTheEarth)	119
4.11	Hausbau-forum (www.hausbau-forum.de)	120
4.12	Distribution of contributors per thread on hausbau-forum (by the author)	120
4.13	Distribution of experts on hausbau-forum (by the author)	121
4.14	Different User Types Identified in a Community (Hutter et al. 2011)	123
4.15	Types of participants (G. Fischer 2016)	124
4.16	Crowd wisdom precedents (by the author)	127
5.1	Simcity (Electronic Arts)	130
5.2	<i>Project Discovery</i> (CCP)	133
5.3	Architectural games (MVRDV, Plethora-Project)	135
5.4	Gamification examples (Stack Overflow, Foursquare, Apple, Inc)	136
5.5	Game, Gamification or Toy (Deterding et al. 2011)	137
6.1	Intersections of the four fields (by the author)	138
6.2	Map of Design Paradigms (by the author)	139
6.3	Precedents and own case studies (by the author)	139
6.4	Examples of Generative Design and Game Design (Stalberg; Namco)	141
6.5	CraftAssist by Facebook AI Research (Gray et al. 2019)	142
6.6	Picbreeder (Secretan, Beato, D'Ambrosio, et al. 2011)	142
6.7	The four case studies (by the author)	146
7.1	The installation <i>Sensitive Assembly</i> (Oliver Tessmann)	147
7.2	The case study <i>Sensitive Assembly</i> on the map of design paradigms (by the author)	148
7.3	<i>Sensitive Assembly</i> plan (by the author)	150
7.4	The components of the <i>Sensitive Assembly</i> set-up (by the author)	151
7.5	A sequence of subtractive wall operations (by the author)	151
7.6	The game Jenga consists of 54 blocks (by the author)	152
7.7	The wall is made from interlocking blocks (by the author)	152
7.8	The wall block design (by the author)	153
7.9	The assembly sequence of a wall block (by the author)	153
7.10	Grip and pull (by the author)	154
7.11	Wall state reconstruction (by the author)	156
7.12	Structural analysis (by the author)	157
7.13	Projection mapping textures (by the author)	158
7.14	Wall sides (by the author)	158
7.15	The flowchart of the feedback loop (by the author)	159
7.16	Collapse prediction feedback (Stig Anton Nielsen)	159
7.17	Edginess, porosity and coherency (Stig Anton Nielsen)	160
7.18	Searching by a number of blocks (by the author)	162
7.19	Searching by design (by the author)	162
7.20	Instructions for the Sunlight play mode (by the author)	164
7.21	Sunlight play mode (by the author)	165
7.22	Instructions for the Jenga play mode (by the author)	165
7.23	Jenga play mode (Oliver Tessmann)	165

7.24	Competition and collaboration (Oliver Tessmann)	166
7.25	Fuzziness in the player created designs (by the author)	167
7.26	Topological optimization games (by the author)	167
8.1	<i>20.000 BLOCKS</i> , framework illustration (by the author)	171
8.2	The projects in <i>20.000 BLOCKS</i> on the map of Design Paradigm (by the author)	172
8.3	The main algorithmic components of the <i>20.000 BLOCKS</i> framework (by the author)	173
8.4	<i>20.000 BLOCKS</i> Game and Robot shown at CEBIT 2017 (Futurium, Rottmann 2017)	174
8.5	Hardware and software components of the <i>20.000 BLOCKS</i> framework (by the author)	175
8.6	The layout of a <i>20.000 BLOCKS</i> game space in Minecraft (by the author)	177
8.7	Screenshots of <i>20.000 BLOCKS</i> running on a ComputerCraft computer in Minecraft (by the author)	177
8.8	Villagers are used in <i>20.000 BLOCKS</i> as material shops (by the author)	178
8.9	Balancing the material costs in the Platform game (by the author)	178
8.10	Structural verification of Minecraft shapes in real-time (by the author)	179
8.11	Minecraft systems used for player feedback in <i>20.000 BLOCKS</i> (by the author)	180
8.12	Shape activation in <i>20.000 BLOCKS</i> (by the author)	180
8.13	The robotic process in <i>20.000 BLOCKS</i> (Rui Nong 2018)	181
8.14	The plan and side view of the latest iteration of the robot process setup in <i>20.000 BLOCKS</i> (by the author)	182
8.15	The robot setup (Tabita Cargnel 2016)	183
8.16	The components of the robot process (Rui Nong 2018)	183
8.17	The <i>20.000 BLOCKS</i> robot installation at CEBIT 2017 (Futurium, Rottmann 2017)	184
8.18	The robot setup of <i>20.000 BLOCKS</i> installed at an info event of Technical University Darmstadt (Tabita Cargnel 2016)	184
8.19	The sequence of robot actions in the pick-and-place routine used to build the models in <i>20.000 BLOCKS</i> (Rui Nong 2018)	185
8.20	The robot gripper and the three types of cubes used to produce the robotically assembled models in <i>20.000 BLOCKS</i> (by the author)	185
8.21	The components making up the custom-designed suction gripper in <i>20.000 BLOCKS</i> (by the author)	186
8.22	Close-up photos of the robot process components (Rui Nong 2018)	186
8.23	The simulation of the path planning using the Robots add-on for Grasshopper (by the author)	186
8.24	The robotic fabrication process in <i>20.000 BLOCKS</i> (Jörg Hartmann 2016)	187
8.25	Models produced by the robot (the author, Rui Nong)	187
8.26	The online visualizer set to campaign view (by the author)	188
8.27	The design view of the online visualizer (by the author)	188
8.28	The element view of the online visualizer (by the author)	189
8.29	Components of the <i>playable, voxel-shape grammars</i> (by the author)	189

8.30	Voxel shapes in the language defined by the voxel-shape grammar shown in Figure 8.35 (by the author)	190
8.31	Steps in the development and application of a grammar (Chase 2002)	191
8.32	Combinatorial design with shape grammars (Stiny 1980a)	193
8.33	A shape and a voxel shape (by the author)	195
8.34	Shape grammar specification (Stiny 1980b)	196
8.35	Voxel-shape grammar specification (by the author)	197
8.36	Playable voxel-shape grammar specification (by the author)	198
8.37	Game mechanics implemented with playable voxel-shape grammars (by the author)	198
8.38	An example of a voxel-shape grammar at the urban scale (by the author)	200
8.39	Two resources that describe a set of metadesign features of the voxel-shapes in the grammar (by the author)	201
8.40	Design variations based on shape orientation (by the author)	201
8.41	A schematic design for a residential building created with a 6-rule voxel-shape grammar (by the author)	202
8.42	Limiting player choices with design (by the author)	204
8.43	Taxonomy extension (by the author)	205
8.44	Structure of the case study <i>20.000 BLOCKS</i> (by the author)	207
8.45	Results from Platform game (by the author)	207
8.46	Platform game screenshots (by the author)	208
8.47	The game logic in an early-version <i>20.000 BLOCKS</i> world (by the author)	208
8.48	Initial state of the Platform game (by the author)	209
8.49	The soft vocabulary (by the author)	209
8.50	A typical result from a game session using the soft vocabulary (by the author)	210
8.51	Secondary goals (by the author)	211
8.52	Game loops (by the author)	211
8.53	Structural simulation of game results (by the author)	212
8.54	The grammar from the intro game in <i>Play-Design-Build</i> (by the author)	213
8.55	The <i>20.000 BLOCKS</i> team and workshop participants at SmartGeometry 2016 (by the author)	214
8.56	The <i>20.000 BLOCKS</i> setup at SmartGeometry 2016 (by the author)	214
8.57	Photos from the <i>20.000 BLOCKS</i> cluster at SmartGeometry 2016 (Jörg Hartmann, the author, Andrea Quartara)	215
8.58	Isovist analysis (Quartara, Emmanuelli, Montnemery, the author)	215
8.59	A <i>20.000 BLOCKS</i> game map titled <i>WOULD YOU MIND YONA FRIEDMAN?</i> (Messios, Rudolph)	216
8.60	Architectural rendering (Messios, Rudolph)	217
8.61	The vocabulary and replacing rules in <i>Would you mind Yona Friedman?</i> (Messios, Rudolph)	217
8.62	A typical game progression in the project <i>Would you Mind Yona Friedman?</i> (Messios, Rudolph)	218
8.63	In-game screenshot from the two selected game results (Messios, Rudolph)	218
8.64	Programmatic composition of the two game results (Messios, Rudolph)	219
8.65	Structural analysis of the two game results (Messios, Rudolph)	219

8.66	Model photo (by the author)	219
8.67	Architectural rendering (Messios, Rudolph)	220
8.68	A <i>20.000 BLOCKS</i> game map titled <i>HOW HIGH</i> (Quartara, Emmanuelli, Montnemery)	221
8.69	The grammar in the <i>20.000 BLOCKS</i> project <i>How High</i> (Quartara, Emmanuelli, Montnemery)	222
8.70	A game sequence (Quartara, Emmanuelli, Montnemery)	222
8.71	Grammar evolution (by the author)	223
8.72	Structural analysis of one on the player created designs in the <i>How High</i> project (Quartara, Emmanuelli, Montnemery)	223
8.73	Model photo (by the author)	224
8.74	The <i>20.000 BLOCKS</i> project <i>Bridging</i> (Gösta, Eliasson, Choudhury)	225
8.75	Game result from the <i>Bridging</i> game (Jörg Hartmann)	226
8.76	The vocabulary from the project <i>Bridging</i> (Gösta, Eliasson, Choudhury)	226
8.77	Vocabulary-based game rewards (Gösta, Eliasson, Choudhury)	227
8.78	Two designs created by players of the <i>Bridging</i> game (by the author)	227
8.79	Structural analysis of a player-created design in the project <i>Bridging</i> (Gösta, Eliasson, Choudhury)	228
8.80	Model photo (Eliasson)	228
8.81	<i>IBA_GAME</i> poster (by the author)	229
8.82	Aerial photo of the Patrick Henry Village (Stadt Heidelberg, Kai Sommer, 2010)	230
8.83	The participatory process for Planning Phase Zero of PHV (IBA Heidelberg, Christian Buck, 2017)	231
8.84	The dynamic masterplan for PHV by KCAP (Carlo Ratti Associati, IBA Heidelberg, Christian Buck 2016)	232
8.85	Photos from <i>IBA_GAME</i> play sessions (IBA Heildeberg, Christian Buck, DDU, Tabita Cargnel, 2016)	232
8.86	Neighborhoods created by players in <i>IBA_GAME</i> (by the author)	233
8.87	The Patrick Henry village divided into neighborhoods in the <i>IBA_GAME</i> (Mar- ios Messios)	233
8.88	<i>IBA_GAME</i> tutorial area (by the author)	234
8.89	The three stages of developing a new vocabulary iteration (Ben Buck- ton)	234
8.90	Grammar and vocabulary evolution (the author, Messios, Rudolph)	235
8.91	A grammar logic for extending a house (Max Rudolph)	236
8.92	The grammar logic for extending the buildings in <i>IBA_GAME</i> (Max Rudolph)	236
8.93	Pareto front of residential designs (by the author)	237
8.94	The method of the course <i>Combinatorial Design</i> (by the author)	238
8.95	Habitat 67 in <i>20.000 BLOCKS</i> (n.n.)	238
8.96	A residential unit from Habitat 67 (Mingquan Ding)	239
8.97	Density as score (Mingquan Ding)	239
8.98	Access rule (Mingquan Ding)	240
8.99	Designs from the Habitat67-inspired grammar (Mingquan Ding)	240
8.100	The Crave (A-Lab, Schneider, Stöckli)	241

8.101	The organizational principle encoded in the grammar (Schneider, Stöckli)	242
8.102	Game mechanics (Schneider, Stöckli)	242
8.103	Iterations from the grammar development of <i>The Carve</i> (Schneider, Stöckli)	243
8.104	The elements in the grammar of <i>The Carve</i> with their keys and activator blocks (Schneider, Stöckli)	243
8.105	A top view of the catalog area in Minecraft (Schneider, Stöckli) . . .	244
8.106	The game area (Schneider, Stöckli)	244
8.107	Variety of player-created design in <i>The Carve</i> (Schneider, Stöckli) . .	245
8.108	Pareto Front (Schneider, Stöckli)	245
8.109	The eight designs created by players in <i>The Carve</i> (Schneider, Stöckli)	246
8.110	Juvika (Kazuhiro Kojima, Schäfer, Abu-salha, Mayer)	247
8.111	The game field of <i>Juvika</i> (Schäfer, Abu-salha, Mayer)	248
8.112	The vertical voids rule in the grammar of <i>Juvika</i> (Kazuhiro Kojima, Schäfer, Abu-salha, Mayer)	248
8.113	The stairs game mechanic in <i>Juvika</i> (Kazuhiro Kojima, Schäfer, Abu-salha, Mayer)	249
8.114	The shape grammar in <i>Juvika</i> (Schäfer, Abu-salha, Mayer)	249
8.115	The point system in <i>Juvika</i> (Schäfer, Abu-salha, Mayer)	249
8.116	Density vs Quality (Schäfer, Abu-salha, Mayer)	250
8.117	All 18 designs collected in <i>Juvika</i> (Schäfer, Abu-salha, Mayer)	250
8.118	Inakasa (Acosta, Martín, Sajak, Markgraf)	251
8.119	Inakasa Grammar (Sajak, Markgraf)	252
8.120	The breakdown of apartment components in the Inakasa building (Sajak, Markgraf)	252
8.121	Grammar combinations 1 (Sajak, Markgraf)	253
8.122	Material sourcing for free building (Sajak, Markgraf)	253
8.123	Grammar combinations 2 (Sajak, Markgraf)	253
8.124	Grammar iteration 1 (Sajak, Markgraf)	254
8.125	Grammar iteration 2 (Sajak, Markgraf)	254
8.126	Grammar iteration 3 (Sajak, Markgraf)	254
8.127	The Mountain (PLOT, Anjadini, Krzywik, Diduch)	255
8.128	Building analysis (PLOT, Anjadini, Krzywik, Diduch)	256
8.129	Building elements (Anjadini, Krzywik, Diduch)	256
8.130	The vocabulary catalog in <i>The Mountain</i> with its activation keys (Anjadini, Krzywik, Diduch)	257
8.131	The reward system in <i>The Mountain</i> (Anjadini, Krzywik, Diduch) . .	257
8.132	Placement rules (Anjadini, Krzywik, Diduch)	257
8.133	A sample game sequence and the points it brings to the player (Anjadini, Krzywik, Diduch)	258
8.134	The two game phases in <i>The Mountain</i> (Anjadini, Krzywik, Diduch)	259
8.135	Designs 1 nad 2 from <i>The Mountain</i> game (Anjadini, Krzywik, Diduch)	260
8.136	Designs 3 nad 4 from <i>The Mountain</i> game (Anjadini, Krzywik, Diduch)	261
8.137	Vocabulary types explored in <i>20.000 BLOCKS</i> (by the author)	262
8.138	Gradient from soft to hard (by the author)	265
8.139	Design style guide (Marios Messios)	266

9.1	A tileset from <i>Project Reptiles</i> in use (Roger Winkler)	267
9.2	<i>Project Reptiles</i> on the map of design paradigms (by the author)	268
9.3	Flow through assistive algorithms (by the author)	269
9.4	Brick Block tileset (Roger Winkler)	270
9.5	Example tileset modeled by an architecture student (Roger Winkler)	270
9.6	User interface in the sculpting mode (Roger Winkler)	271
9.7	User interface in the tileset editing mode (Roger Winkler)	272
9.8	Online configurator (Roger Winkler)	273
9.9	Site 3D model (Roger Winkler)	274
9.10	Tools available to the user in <i>Project Reptiles</i> (Roger Winkler)	274
9.11	Design metrics (Roger Winkler)	275
9.12	Data representation (Roger Winkler)	275
9.13	Flights filtering interface (Opodo)	276
9.14	Spacemaker Optioneering Interface (Spacemaker)	277
9.15	<i>Project Reptiles</i> filtering interface (Roger Winkler)	277
9.16	Google autocomplete (Google, Inc)	278
9.17	Autocomplete in <i>Project Reptiles</i> (Roger Winkler)	278
9.18	Similarity metric (Roger Winkler)	279
9.19	Marching cubes logic on architectural tilesets (Roger Winkler)	281
9.20	Marching squares (Roger Winkler)	282
9.21	Brick Block (Oskar Stalberg)	283
9.22	Design options (Roger Winkler)	283
9.23	Tilesets catalog (Roger Winkler)	284
9.24	Connectivity tileset (Roger Winkler)	285
9.25	Project by Oswald Mathias Ungers for the Roosevelt Island Housing competition, 1975 (Ungers)	287
9.26	User study setup (by the author)	288
9.27	<i>Project Reptiles</i> ad on Facebook (Roger Winkler)	288
9.28	Crowdsourced designs (Roger Winkler)	289
9.29	Photos from the mock design competition (Roger Winkler)	290
9.30	Normalization step (Roger Winkler)	291
9.31	Model photos (Roger Winkler)	291
9.32	Site plans (Roger Winkler)	292
9.33	Photo from the ranking process (Roger Winkler)	292
10.1	<i>Rechteck2BIM</i> on the four fields map (by the author)	295
10.2	Spring-based prototype for <i>Rechteck2BIM</i> (by the author)	297
10.3	A spring-based simulation approach to floor plan generation ((Arvin and House 2002))	300
10.4	3D model generated from room arrangement (by the author)	301
10.5	Prototype of a simulation-based floor plan configurator from the case study <i>Rechteck2BIM</i> (by the author)	301
10.6	Constraint-based floor plan game prototype (by the author)	302
10.7	Adjacency mechanic (by the author)	302
10.8	Separation mechanic (by the author)	302
10.9	Daylight mechanic (by the author)	303
10.10	Footprint mechanic (by the author)	303
10.11	Stairs mechanic (by the author)	303

10.12	Case-based floor plan suggesting tool (Roger Winkler)	303
10.13	Floor plan attributes (Meizi Ren)	304
10.14	Sample question (Meizi Ren)	304
10.15	Attribute-ranking matrix (Meizi Ren)	305
10.16	Survey results (Meizi Ren)	305
10.17	Respondent histogram (Meizi Ren)	306
10.18	Floor plan matching (Meizi Ren)	306
10.19	Top matches (Meizi Ren)	306
10.20	Graph-similarity matching (by the author)	307
10.21	Procedural 3D model generation (by the author)	307
10.22	Automated BIM model generation (by the author)	308
10.23	Computational floor plan analysis (by the author)	309
10.24	CHASELIGHT App concept (Carrion, Hamel, Hamot)	310
10.25	COSY app concept (Hartmann, Joachim, Sand)	311
10.26	ENERGETIC ASSESSMENT App logic (Ruffertshöfer, Ritz, Walser)	311
10.27	SAVING MONEY app concept (Wurm, Jiménez)	312
10.28	COSTINATOR app concept (Klumb, Pfaff)	313
10.29	VIEWSPOTS app concept (Helbig, Vehrenberg)	313
10.30	VIEWSPOTS logic (Helbig, Vehrenberg)	314
10.31	Visual connectivity footprint (Helbig, Vehrenberg)	314
10.32	Control value (CV) justified graph analysis (Rui Zhi)	315
10.33	Real relative asymmetry (RRA) analysis (Rui Zhi)	316
10.34	German house (Rui Zhi)	317
10.35	Japanese house (Rui Zhi)	317
10.36	The survey in 1000 Plans (Helbig, Vehrenberg)	319
10.37	Survey results (by the author)	320
11.1	Repeat of Figure 6.2 (by the author)	322
11.2	Operational use of the design paradigms map (by the author)	323
11.3	Distinction between popular uses of Minecraft in Architecture and the <i>20.000 BLOCKS</i> (by the author)	324
11.4	New zones of human-machine interaction (by the author)	325
11.5	Three levels of crowdsourcing (by the author)	326
11.6	A repeat of Figure 8.43 (by the author)	329
11.7	The explored generative techniques per case study (by the author)	330
11.8	Designs created within <i>20.000 BLOCKS</i> (by the author)	331
11.9	Designs created within <i>Project Reptiles</i> (by the author)	331
11.10	Shape-first vs program-first (Schneider and Stöckli, the author)	332
11.11	An example of the Program-first approach in real practice (P. Merrell, Schkufza and Koltun 2010)	332
11.12	Shape-first generative techniques (the author, Winkler)	332
11.13	Program-first generative technique (by the author)	333
11.14	Digital Materials (W. Langford, Ghassaei and N. Gershenfeld 2016, the author, Rui Nong)	334
11.15	Direct engagement vs byproduct (by the author)	335
11.16	The gradient of control (by the author)	336
11.17	Pure vocabulary combinations vs free modeling on the gradient of control (by the author)	337

11.18	Goal types on the gradient of control (by the author)	338
11.19	Roles with new tasks (by the author)	339
11.20	Three crowdsourcing scenarios (by the author)	340
11.21	Granular authorship in r/Place (Reddit)	343
11.22	A repeat of Figure 11.1 (the author after Roberts 2001)	344
11.23	Granular authorship in architecture (the author, Elemental)	344
11.24	Granular authorship in <i>IBA_GAME</i> (by the author)	345
12.1	A repeat of Figure 1.7 (by the author)	347
12.2	Three crowdsourcing scenarios (by the author)	348
12.3	Self-organization (Rudofsky 1964, Reddit)	351
A.1	Game examples (Bay12Games; Sophon Relaxs)	354
A.2	Centralized vs decentralized control (by the author)	358
A.3	Unbalanced vs balanced games (Barton and Loguidice 2009; ZYNGA)	359
A.4	The concept of Flow (by the author)	360
B.1	<i>20.000 BLOCKS</i> (by the author)	363
B.2	Install Technic Launcher (by the author)	365
B.3	Log in with Mojang account (by the author)	365
B.4	Home screen of Technic Launcher (by the author)	366
B.5	Modpacks screen (by the author)	367
B.6	Minecraft home screen (by the author)	368
B.7	Create a new world (by the author)	369
B.8	Classic, nature-like world in Minecraft (by the author)	369
B.9	Superflat world with white grid (by the author)	370
B.10	Superflat world with gray grid (by the author)	370
B.11	World settings (by the author)	371
B.12	Set the difficulty level to <i>peaceful</i> (by the author)	371
B.13	Command computer in the inventory (by the author)	372
B.14	A placed command computer (by the author)	372
B.15	CraftOS home screen (by the author)	373
B.16	Load the <i>20.000 BLOCKS</i> scripts from Pastebin (by the author)	373
B.17	Update scripts (by the author)	374
B.18	<i>20.000 BLOCKS</i> project settings (by the author)	374
B.19	The zones in the game area (by the author)	375
B.20	Start the game (by the author)	375
B.21	A freshly started game script with the buildzone (by the author)	376
B.22	The catalogue building area (by the author)	377
B.23	Key (by the author)	378
B.24	Element is placed but key remains (by the author)	379
B.25	An element with barrier blocks to erase its key (by the author)	379
B.26	The same element placed, the is removed upon placing (by the author)	380
B.27	Slot number (by the author)	381
B.28	Another slot number (by the author)	382
B.29	Swap slots 6 and 14 (by the author)	382
B.30	Catalog settings (by the author)	383
B.31	The border of the new catalog area (by the author)	383
B.32	Use the Debug info mode to see material names (by the author)	384

B.33 Online list of Minecraft materials (Minecraft Gamepedia)	385
B.34 Edit starting items (by the author)	385
B.35 Default starting items settings (by the author)	386
B.36 Modified starting items settings (by the author)	386
B.37 A portal to start a new game (by the author)	387
B.38 The Homecomer (by the author)	388
B.39 Forge installation (by the author)	389
B.40 Server console (by the author)	389
B.41 Mod jar files (by the author)	390
B.42 Joining a server (by the author)	392
B.43 Using the Debug info view mode to get location and orientation co- ordinates (by the author)	395
B.44 Teleport to predefined location and view angles (by the author) . . .	396
B.45 Relative positions (by the author)	396
B.46 Adjusted relative position (by the author)	397
B.47 Teleport command with relative coordinates (by the author)	397
B.48 Axonometric screen shot (by the author)	398

Bibliography

Papers, Books and Articles

- Afacan, Y. and H. Demirkan (May 1, 2011), “An Ontology-Based Universal Design Knowledge Support System”, in: *Knowledge-Based Systems* 24.4, pp. 530–541 (cit. on pp. 20, 73).
- Aish, R. (2016), “Design Research and Design Participation”, in: *Design Research Society Conference 2016* (cit. on p. 347).
- Akin, Ö. and H. Moustapha (Jan. 1, 2004), “Strategic Use of Representation in Architectural Massing”, in: *Design Studies* 25.1, pp. 31–50 (cit. on p. 11).
- Akizuki, Y., M. Bernhard, R. Kakooee, M. Kladeftira and B. Dillenburger (2020), “Generative Modelling with Design Constraints”, in: *Anthropocene: Proceedings of the 25th International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA)*, vol. 1, pp. 445–454 (cit. on p. 77).
- Alexander, C., S. J. A. Ishikawa and M. J. A. Silverstein (1977), *A Pattern Language: Towns, Buildings, Construction* (cit. on pp. 35, 36, 42, 43, 78, 330).
- Alexander, C. (1964), *Notes on the Synthesis of Form*, Cambridge: Harvard University Press (cit. on pp. 19, 21, 22, 34, 36, 330).
- Alfonseca, M. and A. Ortega (June 15, 1996), “Representation of Fractal Curves by Means of L Systems”, in: *ACM SIGAPL APL Quote Quad* 26.4, pp. 13–21 (cit. on pp. 63, 64).
- Anderson, C. (2008), “The End of Theory: The Data Deluge Makes the Scientific Method Obsolete”, in: *Wired magazine* 16.7, pp. 16–07 (cit. on p. 114).
- Anderson, D., J. L. Frankel, J. Marks, A. Agarwala, P. Beardsley, J. Hodgins, D. Leigh, K. Ryall, E. Sullivan and J. S. Yedidia (July 1, 2000), “Tangible Interaction + Graphical Interpretation: A New Approach to 3D Modeling”, in: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, USA: ACM Press/Addison-Wesley Publishing Co., pp. 393–402 (cit. on p. 86).
- Anderson, D. P., J. Cobb, E. Korpela, M. Lebofsky and D. Werthimer (Nov. 1, 2002), “SETI@home: An Experiment in Public-Resource Computing”, in: *Communications of the ACM* 45.11, pp. 56–61 (cit. on p. 114).
- Apt, K. R. (2003), *Principles of Constraint Programming* (cit. on p. 68).
- Archea, J. (1987), “Puzzle-Making : What Architects Do When No One Is Looking”, in: *New York: Wiley-Interscience, 1987. Pp. 37-52. Includes Bibliography, CUMINCAD* (cit. on pp. 19, 330).
- Armour, G. C. and E. S. Buffa (Jan. 1, 1963), “A Heuristic Algorithm and Simulation Approach to Relative Location of Facilities”, in: *Management Science* 9.2, pp. 294–309 (cit. on p. 46).

- Arnstein, S. R. (1969), “A Ladder Of Citizen Participation”, in: *Journal of the American Planning Association* 35.4, pp. 216–224 (cit. on pp. 23, 24).
- Arvin, S. A. (2004), “Physically Based Mechanical Metaphors in Architectural Space Planning”, PhD thesis, Texas A&M University (cit. on p. 297).
- Arvin, S. A. and D. H. House (1999), “Modeling Architectural Design Objectives in Physically Based Space Planning”, in: *Acadia 99*, pp. 192–205 (cit. on pp. 86, 297).
- Arvin, S. A. and D. H. House (2002), “Modeling Architectural Design Objectives in Physically Based Space Planning”, in: *Automation in Construction* 11.2, pp. 213–225 (cit. on pp. 69, 70, 297, 300).
- Aurenhammer, F. (Sept. 1, 1991), “Voronoi Diagrams — a Survey of a Fundamental Geometric Data Structure”, in: *ACM Computing Surveys* 23.3, pp. 345–405 (cit. on p. 61).
- Backlund, P., H. Engström, C. Hammar, M. Johannesson and M. Lebram (2007), “Sidh - A Game Based Firefighter Training Simulation”, in: *Proceedings of the International Conference on Information Visualisation* (May 2014), pp. 899–907 (cit. on p. 134).
- Ball, P. (2004), *Critical Mass: How One Thing Leads to Another*, 1st American ed, New York: Farrar, Straus and Giroux, 520 pp. (cit. on p. 109).
- Banerjee, A., J. C. Quiroz and S. J. Louis (2008), “A Model of Creative Design Using Collaborative Interactive Genetic Algorithms”, in: *Design Computing and Cognition '08*, ed. by Gero, J. S. and Goel, A. K., Dordrecht: Springer Netherlands, pp. 397–416 (cit. on p. 88).
- Batty, M. (2005), *Cities and Complexity: Understanding Cities with Cellular Automata, Agent-Based Models, and Fractals*, Cambridge, Mass: MIT Press, 565 pp. (cit. on p. 69).
- Becker, A., ed. (2015), *Bauen und wohnen in Gemeinschaft: Ideen, Prozesse, Architektur = Building and living in communities: ideas, processes, architecture*, Basel: Birkhäuser, 239 pp. (cit. on pp. 4, 26).
- Bell, K. (2018), *Game on! Gamification, Gameful Design, and the Rise of the Gamer Educator*, Tech.Edu : A Hopkins Series on Education and Technology, Baltimore: Johns Hopkins University Press, 203 pp. (cit. on p. 136).
- Beneš, B., O. Št'ava, R. Měch and G. Miller (2011), “Guided Procedural Modeling”, in: *Computer Graphics Forum* 30.2, pp. 325–334 (cit. on pp. 93, 94, 102).
- Benros, D. and J. P. Duarte (2009), “An Integrated System for Providing Mass Customized Housing”, in: *Automation in Construction* 18.3, pp. 310–320 (cit. on p. 4).
- Bernal, M., J. R. Haymaker and C. Eastman (Nov. 1, 2015), “On the Role of Computational Support for Designers in Action”, in: *Design Studies* 41, pp. 163–182 (cit. on pp. 48, 51, 53).
- Best, S. and D. Kellner (June 1, 1999), “Kevin Kelly’s Complexity Theory: The Politics and Ideology of Self-Organizing Systems”, in: *Organization & Environment* 12.2, pp. 141–162 (cit. on p. 110).
- Bhatta, S. R. and A. K. Goel (2002), “Design Patterns and Creative Design”, in: *Engineering Design Synthesis: Understanding, Approaches and Tools*, ed. by Chakrabarti, A., London: Springer, pp. 271–284 (cit. on p. 78).

-
- Bianconi, F., M. Filippucci and A. Buffi (2019), “Automated Design and Modeling for Mass-Customized Housing. A Web-Based Design Space Catalog for Timber Structures”, in: *Automation in Construction* 103, pp. 13–25 (cit. on pp. 4, 105).
- Bishop, C. (2012), *Artificial Hells: Participatory Art and the Politics of Spectatorship*, London ; New York: Verso Books, 382 pp. (cit. on p. 24).
- Bohm, M. R., J. P. Vucovich and R. B. Stone (Feb. 28, 2008), “Using a Design Repository to Drive Concept Generation”, in: *Journal of Computing and Information Science in Engineering* 8.014502 (cit. on p. 79).
- Bonney, R. (1996), “Citizen Science: A Lab Tradition”, in: *Living Bird* 15.4, pp. 7–15 (cit. on p. 114).
- Bonswetch, T. (2006), “The Informed Wall: Applying Additive Digital Fabrication Techniques on Architecture”, in: *Synthetic Landscapes [Proceedings of the 25th Annual Conference of the Association for Computer-Aided Design in Architecture] Pp. 489-495*, CUMINCAD (cit. on pp. 103, 104).
- Börner, K., E. Pippig, E.-C. Tammer and C.-H. Coulon (1996), “Structural Similarity and Adaptation”, in: *Advances in Case-Based Reasoning*, ed. by Smith, I. and Faltings, B., Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, pp. 58–75 (cit. on p. 73).
- Borries, F. von, ed. (2007), *Space Time Play: Computer Games, Architecture and Urbanism: The next Level*, Basel: Birkhäuser, 495 pp. (cit. on p. 131).
- Box, G. E. P. (Jan. 1, 1979), “Robustness in the Strategy of Scientific Model Building”, in: *Robustness in Statistics*, ed. by Launer, R. L. and Wilkinson, G. N., Academic Press, pp. 201–236 (cit. on p. 147).
- Brabham, D. C. (2013), *Crowdsourcing*, 176 pp. (cit. on pp. 110, 126, 264, 336).
- Brawne, M. (2005), *Architectural Thought: The Design Process and the Expectant Eye*, Amsterdam ; Boston: Elsevier : Architectural Press, 190 pp. (cit. on p. 19).
- Brintrup, A. M., J. Ramsden and A. Tiwari (2007), “An Interactive Genetic Algorithm-Based Framework for Handling Qualitative Criteria in Design Optimization”, in: *Computers in Industry* 58.3, pp. 279–291 (cit. on pp. 88, 89, 101, 194).
- Budig, M., J. Lim and R. Petrovic (May 2014), “Integrating Robotic Fabrication in the Design Process”, in: *Architectural Design* 84.3, pp. 22–43 (cit. on pp. 105, 334).
- Burnap, A., Y. Ren, R. J. Gerth, G. Papazoglou, R. Gonzalez and P. Y. Papalambros (Mar. 1, 2015), “When Crowdsourcing Fails: A Study of Expertise on Crowdsourced Design Evaluation”, in: *Journal of Mechanical Design* 137.031101 (cit. on p. 124).
- Cabanes, G. and Y. Bennani (Apr. 1, 2010), “Learning the Number of Clusters in Self Organizing Map”, in: *Self-Organizing Maps*, IntechOpen (cit. on p. 158).
- Caetano, I., L. Santos and A. Leitão (2020), “Computational Design in Architecture: Defining Parametric, Generative, and Algorithmic Design”, in: *Frontiers of Architectural Research* 9.2, pp. 287–300 (cit. on pp. 46–48, 54, 62, 101).
- Calixto, V. and G. Celani (2015), “A Literature Review for Space Planning Optimization Using an Evolutionary Algorithm Approach: 1992-2014”, in: *Blucher Design Proceedings*, XIX Congresso Da Sociedade Ibero-americana de Gráfica Digital 2015, vol. 2, 3, pp. 662–671 (cit. on pp. 47, 48, 52, 77).
- Carmo, M. (2011), *The Alphabet and the Algorithm*, Cambridge, MA: MIT Press, 169 pp. (cit. on pp. 3, 347).

- Carrara, G., Y. E. Kalay and G. Novembri (July 1, 1994), “Knowledge-Based Computational Support for Architectural Design”, in: *Automation in Construction*, Special Issue: Knowledge-based, Computer-aided Architectural Design 3.2, pp. 157–175 (cit. on p. 73).
- Chakrabarti, A., K. Shea, R. Stone, J. Cagan, M. Campbell, N. V. Hernandez and K. L. Wood (June 15, 2011), “Computer-Based Design Synthesis Research: An Overview”, in: *Journal of Computing and Information Science in Engineering* 11.021003 (cit. on pp. 46–49, 63, 65, 67, 72–74, 79, 93).
- Charman, P. (1993), *Solving Space Planning Problems Using Constraint Technology*, Tallinn, Estonia: Institute of Cybernetics, Estonian Academy of Sciences (cit. on p. 68).
- Chase, S. C. (Feb. 1, 2002), “A Model for User Interaction in Grammar-Based Design Systems”, in: *Automation in Construction*, ACADIA '99 11.2, pp. 161–172 (cit. on pp. 91, 92, 191).
- Chase, S. C. (2010), “Shape Grammar Implementations”, in: p. 55 (cit. on pp. 191, 192).
- Chase, S. C. (2003), “Revisiting the Use of Generative Design Tools in the Early Stages of Design Education”, in: *Digital Design: Proceedings of the 21st Conference on Education in Computer Aided Architectural Design in Europe, 17-20 September 2003*, Graz, Austria, pp. 465–472 (cit. on pp. 47, 48).
- Chau, H. H., X. Chen, A. McKay and A. de Pennington (2004), “Evaluation of a 3D Shape Grammar Implementation”, in: *Design Computing and Cognition '04*, ed. by Gero, J. S., Dordrecht: Springer Netherlands, pp. 357–376 (cit. on pp. 47, 54, 67).
- Cheung, K. C.-W. (2012), “Digital Cellular Solids : Reconfigurable Composite Materials”, PhD thesis, Massachusetts Institute of Technology (cit. on p. 106).
- Chomsky, N. (June 1, 1959), “On Certain Formal Properties of Grammars”, in: *Information and Control* 2.2, pp. 137–167 (cit. on p. 63).
- Chowdhary, K. R. (2020), “Rule Based Reasoning”, in: *Fundamentals of Artificial Intelligence*, ed. by Chowdhary, K., New Delhi: Springer India, pp. 89–109 (cit. on p. 78).
- Coates, P., C. Derix, I. S. P. Krakhofer and A. Karanouh (Dec. 2005), “Generating Architectural Spatial Configurations. Two Approaches Using Voronoi Tessellations and Particle Systems”, in: (cit. on p. 61).
- Coates, P., N. Healy, C. Lamb and W. L. Voon (Mar. 1996), “The Use of Cellular Automata to Explore Bottom up Architectonic Rules”, in: (cit. on p. 69).
- Cohen, J. M., N. T. Uphoff, et al. (1977), “Rural Development Participation: Concepts and Measures for Project Design, Implementation and Evaluation.”, in: *Rural development participation: concepts and measures for project design, implementation and evaluation*. 2 (cit. on p. 23).
- Cohen, J. M. and N. T. Uphoff (Mar. 1, 1980), “Participation’s Place in Rural Development: Seeking Clarity through Specificity”, in: *World Development* 8.3, pp. 213–235 (cit. on p. 23).
- Cooper, C. B. and B. V. Lewenstein (2016), “Two Meanings of Citizen Science”, in: *Citizen Science the Rightful Place of Science*, ed. by Kennedy, E. B. and Cavalier, D., Arizona State University Press (cit. on p. 114).
- Cooper, S. (2014), “A Framework for Scientific Discovery through Video Games”, PhD thesis, 133 pp. (cit. on p. 348).

-
- Cooper, S., D. Baker, Z. Popović, A. Treuille, J. Barbero, A. Leaver-Fay, K. Tuite, F. Khatib, A. C. Snyder, M. Beenen and D. Salesin (2010), “The Challenge of Designing Scientific Discovery Games”, in: *Proceedings of the Fifth International Conference on the Foundations of Digital Games - FDG '10*, New York, New York, USA: ACM Press, pp. 40–47 (cit. on p. 338).
- Cooper, S., F. Khatib, A. Treuille, J. Barbero, J. Lee, M. Beenen, A. Leaver-Fay, D. Baker, Z. Popović and F. Players (Aug. 5, 2010), “Predicting Protein Structures with a Multiplayer Online Game”, in: *Nature* 466.7307, pp. 756–760 (cit. on pp. 5, 115).
- Correia, R. C., J. P. Duarte and A. M. Leitão (2010), “MALAG: A Discursive Grammar Interpreter for the Online Generation of Mass Customized Housing”, in: *Proceedings of the workshop in 4th Conference Design Computing and Cognition* (August) (cit. on p. 342).
- Cress, U., H. Jeong and J. Moskaliuk, eds. (2016a), *Mass Collaboration and Education*, 1st ed. 2016, Computer-Supported Collaborative Learning Series 16, Cham: Springer International Publishing : Imprint: Springer, 1 p. (cit. on p. 118).
- Cress, U., H. Jeong and J. Moskaliuk (2016b), “Mass Collaboration as an Emerging Paradigm for Education? Theories, Cases, and Research Methods”, in: *Mass Collaboration and Education*, Springer International Publishing, pp. 3–27 (cit. on pp. 116, 118).
- Creswell, J. W. and C. N. Poth (2018), *Qualitative Inquiry & Research Design: Choosing among Five Approaches*, Fourth edition, Los Angeles: SAGE, 459 pp. (cit. on p. 8).
- “Design Participation: Proceedings of the Design Research Society’s Conference, Manchester, September 1971” (1971), in: *Proceedings of the Design Research Society’s Conference*, ed. by Cross, N., London: Academy Editions (cit. on pp. 24, 25).
- Cross, N. (1982), “Designerly Ways of Knowing”, in: *Design Studies* 3.82, pp. 221–227 (cit. on pp. 19, 20, 73).
- Cross, N. (2011), *Design Thinking: Understanding How Designers Think and Work*, Engl. ed, Oxford: Berg, 163 pp. (cit. on p. 19).
- Cross, N. (2013), *Designerly Ways of Knowing*, vol. 53, 9, 1689–1699 (cit. on p. 19).
- Crumley, Z., P. Marais and J. Gain (2012), “Voxel-Space Shape Grammars”, in: *WSCG’2012 Conference Proceedings*, ed. by Skala, V., vol. Part I, Pilsen: Union Agency, pp. 113–122 (cit. on pp. 194, 195).
- Csikszentmihalyi, M. (2008), “Flow: The Psychology of Optimal Experience (Harper Perennial Modern Classics)”, in: (cit. on pp. 8, 9, 268, 360).
- Damski, J. C. and J. S. Gero (1997), “An Evolutionary Approach to Generating Constraint-Based Space Layout Topologies”, in: *CAAD Futures 1997*, ed. by Junge, R., Dordrecht: Springer Netherlands, pp. 855–864 (cit. on p. 61).
- Dapogny, C., A. Faure, G. Michailidis, G. Allaire, A. Couvelas and R. Estevez (June 1, 2017), “Geometric Constraints for Shape and Topology Optimization in Architectural Design”, in: *Computational Mechanics* 59.6, pp. 933–965 (cit. on pp. 70, 71).
- Dave, B., G. Schmitt, B. Faltings and I. Smith (1994), “Case Based Design in Architecture”, in: *Artificial Intelligence in Design '94*, ed. by Gero, J. S. and Sudweeks, F., Dordrecht: Springer Netherlands, pp. 145–162 (cit. on pp. 73, 75).

- De Landa, M. (2000), *A Thousand Years of Nonlinear History*, Zone Books (cit. on p. 122).
- De Berg, M., M. van Kreveld, M. Overmars and O. Schwarzkopf (1997), “Computational Geometry”, in: *Computational Geometry: Algorithms and Applications*, ed. by de Berg, M., van Kreveld, M., Overmars, M. and Schwarzkopf, O., Berlin, Heidelberg: Springer, pp. 1–17 (cit. on pp. 60, 97).
- Dede, C. (Jan. 2, 2009), “Immersive Interfaces for Engagement and Learning”, in: *Science* 323.5910, pp. 66–69 (cit. on p. 335).
- De Mul, J. (2016), “Possible Printings: On 3D Printing, Database Ontology, and Open (Meta)Design”, in: *3D Printing: Legal, Philosophical and Economic Dimensions*, ed. by van den Berg, B., van der Hof, S. and Kosta, E., The Hague: T.M.C. Asser Press, pp. 87–98 (cit. on pp. 1, 2, 13, 124).
- Deterding, S., D. Dixon, R. Khaled and L. Nacke (2011), “From Game Design Elements to Gamefulness: Defining ”Gamification””, in: *Proceedings of the 15th International Academic MindTrek Conference on Envisioning Future Media Environments - MindTrek ’11*, Tampere, Finland: ACM, pp. 9–11 (cit. on pp. 135–137).
- De Villiers, M. and N. Naicker (Nov. 2006), *A Sketching Interface for Procedural City Generation*, Department of Computer Science, University of Cape Town, p. 7 (cit. on p. 102).
- Dillenbourg, P. (1999), “What Do You Mean by ’Collaborative Learning’?”, in: *Collaborative-learning: Cognitive and Computational Approaches* Vol. 1 (cit. on p. 116).
- Dillenburg, B., M. Braach, H. Hovestadt and L. Hovestadt (2009), “Building Design as an Individual Compromise between Qualities and Costs: A General Approach for Automated Building Generation under Permanent Cost and Quality Control”, in: *CAAD Futures 2009*, pp. 458–471 (cit. on p. 4).
- Do, E. Y.-L. (June 2002), “Drawing Marks, Acts, and Reacts: Toward a Computational Sketching Interface for Architectural Design”, in: *AI EDAM* 16.3, pp. 149–171 (cit. on p. 20).
- Dodig, M. B. and L. N. Groat (Dec. 12, 2019), *The Routledge Companion to Games in Architecture and Urban Planning: Tools for Design, Teaching, and Research*, New York: Routledge, 274 pp. (cit. on p. 29).
- Donath, D. and L. F. G. Böhme (Jan. 1, 2008), “Constraint-Based Design in Participatory Housing Planning”, in: *International Journal of Architectural Computing* 6.1, pp. 97–117 (cit. on pp. 94, 95).
- Dréo, J. ” and C. Candan (Aug. 28, 2011), *Different Classifications of Metaheuristics Shown as a Euler Diagram* (cit. on pp. 75, 77).
- Duarte, J. P. (2005), “A Discursive Grammar for Customizing Mass Housing: The Case of Siza’s Houses at Malagueira”, in: *Automation in Construction* 14 (2 SPEC. ISS.), pp. 265–275 (cit. on pp. 4, 192, 295, 296).
- Duarte, J. P. (2001), “Customizing Mass Housing : A Discursive Grammar for Siza’s Malagueira Houses”, PhD thesis, Massachusetts Institute of Technology (cit. on pp. 64–67, 191).
- Dunbar, R. (June 1992), “Neocortex Size as a Constraint on Group Size in Primates”, in: *Journal of Human Evolution* 22.6, pp. 469–493 (cit. on pp. 32, 110).

-
- Durand, J.-N.-L. (2000), *Précis of the Lectures on Architecture : With, Graphic Portion of the Lectures on Architecture*, Los Angeles, CA: Getty Research Institute (cit. on pp. 4, 36).
- Dutta, K. and S. Sarthak (May 1, 2011), “Architectural Space Planning Using Evolutionary Computing Approaches: A Review”, in: *Artificial Intelligence Review* 36.4, p. 311 (cit. on pp. 47, 54).
- Ebert, D. S., F. K. Musgrave, D. Peachey, K. Perlin, S. Worley, W. R. Mark and J. C. Hart (Dec. 2002), *Texturing and Modeling: A Procedural Approach: Third Edition*, Elsevier Inc. (cit. on p. 47).
- Ediz, Ö. and G. Çağdaş (Jan. 1, 2007), “A Computational Architectural Design Model Based on Fractals”, in: *Open House International* 32.2, pp. 36–45 (cit. on p. 61).
- Eimler, S. C., G. Neubaum, M. Mannsfeld and N. C. Krämer (2016), “Altogether Now! Mass and Small Group Collaboration in (Open) Online Courses: A Case Study”, in: *Mass Collaboration and Education*, pp. 285–304 (cit. on p. 123).
- Eitzel, M. V., J. L. Cappadonna, C. Santos-Lang, R. E. Duerr, A. Virapongse, S. E. West, C. C. M. Kyba, A. Bowser, C. B. Cooper, A. Sforzi, A. N. Metcalfe, E. S. Harris, M. Thiel, M. Haklay, L. Ponciano, J. Roche, L. Ceccaroni, F. M. Shilling, D. Dörler, F. Heigl, T. Kiessling, B. Y. Davis and Q. Jiang (June 5, 2017), “Citizen Science Terminology Matters: Exploring Key Terms”, in: *Citizen Science: Theory and Practice* 2.1 (1), p. 1 (cit. on pp. 114, 122).
- Elezkurtaj, T. and G. Franck (2002), “Algorithmic Support of Creative Architectural Design”, in: *Organization*, pp. 1–16 (cit. on pp. 3, 20, 128).
- Elliott, M. (2016), “Stigmergic Collaboration: A Framework for Understanding and Designing Mass Collaboration”, in: *Mass Collaboration and Education*, pp. 65–84 (cit. on p. 122).
- Estellés-Arolas, E. and F. González-Ladrón-de-Guevara (2012), “Towards an Integrated Crowdsourcing Definition”, in: *Journal of Information Science* 38.2, pp. 189–200 (cit. on pp. 110, 111).
- Faltings, B. (Jan. 1, 1996), “Working Group in Model-Based Design and Reasoning. Part II: Design”, in: *AI Communications* 9.2, pp. 65–73 (cit. on p. 73).
- Fischer, G. (2016), “Exploring, Understanding, and Designing Innovative Socio-Technical Environments for Fostering and Supporting Mass Collaboration”, in: *Mass Collaboration and Education*, pp. 43–63 (cit. on pp. 123, 124).
- Fischer, T. (2008), “Designing (Tools (for Designing (Tools (For ...))”, PhD thesis, University of Kassel (cit. on p. 111).
- Flemming, U. (July 1, 1994), “Case-Based Design in the SEED System”, in: *Automation in Construction*, Special Issue: Knowledge-based, Computer-aided Architectural Design 3.2, pp. 123–133 (cit. on p. 73).
- Flemming, U. (1999), *SEED-Layout Tutorial* (cit. on p. 73).
- Flemming, U. and Z. Aygen (Aug. 1, 2001), “A Hybrid Representation of Architectural Precedents”, in: *Automation in Construction*, Design Representation 10.6, pp. 687–699 (cit. on p. 73).
- Flemming, U. and S.-F. Chien (1995), “Schematic Layout Design in SEED Environment”, in: *Journal of Architectural Engineering* 1.4, pp. 162–169 (cit. on p. 73).
- Flemming, U. and R. Woodbury (Dec. 1, 1995), “Software Environment to Support Early Phases in Building Design (SEED): Overview”, in: *Journal of Architectural Engineering* 1.4, pp. 147–152 (cit. on p. 73).

- Frazer, J. (1995), *An Evolutionary Architecture*, London: Architectural Association (cit. on p. 39).
- Friedman, S. and I. Stamos (2013), “Automatic Procedural Modeling of Tree Structures in Point Clouds Using Wavelets”, in: *Proceedings - 2013 International Conference on 3D Vision, 3DV 2013*, pp. 215–222 (cit. on p. 194).
- Friedman, Y. (1980), *Toward a Scientific Architecture*, Cambridge MA: MIT Press, 181 pp. (cit. on pp. 35–37).
- Frith, J. (May 1, 2013), “Turning Life into a Game: Foursquare, Gamification, and Personal Mobility”, in: *Mobile Media & Communication* 1.2, pp. 248–262 (cit. on p. 135).
- Fröst, P. (2003), “A Real Time 3D Environment for Collaborative Design”, in: *CAAD Futures*, Tainan, Taiwan, pp. 203–212 (cit. on p. 176).
- Fröst, P. and P. Warren (2000), “Virtual Reality Used in a Collaborative Architectural Design Process”, in: *2000 IEEE Conference on Information Visualization. An International Conference on Computer Visualization and Graphics*, IEEE Comput. Soc, pp. 568–573 (cit. on p. 175).
- Fuller, R. B. (1969), “THE WORLD GAME”, in: *Ekistics* 28.167, pp. 286–292 (cit. on p. 29).
- Gaglione, C. (2018), *Amsterdam Affordable Housing Game Report* (cit. on p. 29).
- Garcia, S. (2017), “Classifications of Shape Grammars”, in: *Design Computing and Cognition '16*, ed. by Gero, J. S., Cham: Springer International Publishing, pp. 229–248 (cit. on pp. 47, 68).
- Gentner, D. (1983), “Structure-Mapping: A Theoretical Framework for Analogy”, in: *Cognitive Science* 7.2, pp. 155–170 (cit. on p. 73).
- Gerber, D. J. and S.-H. E. Lin (Aug. 1, 2014), “Designing in Complexity: Simulation, Integration, and Multidisciplinary Design Optimization for Architecture”, in: *SIMULATION* 90.8, pp. 936–959 (cit. on p. 59).
- Gero, J. S. (Dec. 15, 1990), “Design Prototypes: A Knowledge Representation Schema for Design”, in: *AI Magazine* 11.4 (4), pp. 26–26 (cit. on pp. 84, 96).
- Gershenfeld, N., director (2006), *Unleash Your Creativity in a Fab Lab* (cit. on pp. 2, 328).
- Gershenfeld, N. (2012), “How to Make Almost Anything: The Digital Fabrication Revolution”, in: *Foreign Affairs* 91.6, pp. 43–57 (cit. on pp. 2, 103, 350).
- Gershenfeld, N., M. Carney, B. Jenett, S. Calisch and S. Wilson (2015), “Macro-fabrication with Digital Materials: Robotic Assembly”, in: *Architectural Design* 85.5, pp. 122–127 (cit. on pp. 103, 106, 151, 334).
- Gershenfeld, N., A. Gershenfeld and J. Cutcher-Gershenfeld, eds. (2017), *Designing Reality: How to Survive and Thrive in the Third Digital Revolution*, New York: Basic Books (cit. on p. 2).
- Gerth, R. J., A. Burnap and P. Y. Papalambros (Aug. 2012), “Crowdsourcing: A Primer and Its Implications for Systems Engineering”, in: 2012 NDIA Ground Vehicle Systems Engineering and Technology Symposium (cit. on p. 111).
- Gips, J. (1975), *Shape Grammars and Their Uses*, Basel: Birkhauser (cit. on p. 66).
- Gips, J. (1999), “Computer Implementation of Shape Grammars”, in: *Workshop on Shape Computation*, MIT, p. 12 (cit. on pp. 91, 191, 192, 194).
- Goel, A. K. and S. R. Bhatta (Apr. 1, 2004), “Use of Design Patterns in Analogy-Based Design”, in: *Advanced Engineering Informatics, Ontology and It’s Applications to Knowledge-Intensive Engineering* 18.2, pp. 85–94 (cit. on p. 78).

-
- Goel, A. K. and S. Craw (Sept. 2005), “Design, Innovation and Case-Based Reasoning”, in: *The Knowledge Engineering Review* 20.3, pp. 271–276 (cit. on pp. 48, 49, 78).
- Gramazio, F., M. Kohler and J. Willmann (May 2014), “Authoring Robotic Processes”, in: *Architectural Design* 84.3, pp. 14–21 (cit. on p. 334).
- Gramazio, F., M. Kohler, J. Willmann, S. Leittle, R. Jaeger, E. T. H. Zürich and G. bibinitperiod Kohler, eds. (2014), *The Robotic Touch: How Robots Change Architecture*, Zürich: Park Books, 488 pp. (cit. on p. 1).
- Granath, J. A. (2001), “Architecture–Participation of Users in Design Activities”, in: *Chalmers Tekniska Högskola. Göteborg* (<http://www.fm.chalmers.se>) (cit. on pp. 3, 24, 25, 33).
- Gray, J., K. Srinet, Y. Jernite, H. Yu, Z. Chen, D. Guo, S. Goyal, C. L. Zitnick and A. Szlam (2019), *CraftAssist : A Framework for Dialogue-enabled Interactive Agents*, Facebook Research (cit. on pp. 141, 142, 349).
- Green, O. (Oct. 7, 2020), “An Introspective Approach to Apartment Design”, in: *DC I/O 2020*, Design Computation Ltd. (cit. on pp. 73, 74, 76, 295, 296).
- Greuter, S., J. Parker, N. Stewart and G. Leach (Feb. 11, 2003), “Real-Time Procedural Generation of ‘pseudo Infinite’ Cities”, in: *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, GRAPHITE ’03, New York, NY, USA: Association for Computing Machinery, 87–ff (cit. on pp. 61, 63).
- Greuter, S., N. Stewart and G. Leach (2004), “Beyond the Horizon: Computer-generated, Three-Dimensional, Infinite Virtual Worlds without Repetition”, in: *Image Text and Sound Conference* (cit. on p. 61).
- Greuter, S., N. Stewart, J. Parker and G. Leach (2003), “Undiscovered Worlds – towards a Framework for Real-Time Procedural World Generation”, in: *Fifth International Digital Arts and Culture Conference, Melbourne, Australia*, vol. 5, p. 5 (cit. on p. 61).
- Groat, L. N. and D. Wang (2013), *Architectural Research Methods*, Architectural Research Methods, Wiley (cit. on p. 8).
- Grobman, Y. J. (2008), “Building the Digital World – Architectural Design Methods Based on the Use of Digital Tools – Performance Based Form Generation and Optimization”, PhD thesis, Technion – Israel Institute of Technology, 194 pp. (cit. on pp. 72–74, 77).
- Grobman, Y. J., A. Yezioro and I. G. Capeluto (Dec. 1, 2009), “Computer-Based Form Generation in Architectural Design — A Critical Review”, in: *International Journal of Architectural Computing* 7.4, pp. 535–553 (cit. on pp. 47–50).
- Gropius, W. (1965), *The New Architecture and the Bauhaus* (cit. on pp. 4, 36).
- Gu, N. and P. A. Behbahani (2018), “Shape Grammars: A Key Generative Design Algorithm”, in: *Handbook of the Mathematics of the Arts and Sciences*, ed. by Sriraman, B., Cham: Springer International Publishing, pp. 1–21 (cit. on pp. 47, 48, 52, 54, 65, 66, 92).
- Gumin, M. (2016), *WaveFunctionCollapse* (cit. on pp. 68, 72).
- Gürsimsek, R. A. (2012), “Multimodal Semiotics and Collaborative Design: A Social Semiotic Approach to the Co-Production of Virtual Places and Artifacts in Second Life”, PhD thesis, Roskilde Universitetsforlag (cit. on pp. 126, 176).

- UN-Habitat (2015), *Using Minecraft for Youth Participation in Urban Design and Governance*, HS/088/15E, United Nations Human Settlements Programme (UN-HABITAT) (cit. on p. 3).
- UN-Habitat (2016), *Using Minecraft for Community Participation*, UN-Habitat (cit. on p. 3).
- Habraken, N. J. and J. Teicher (2000), *The Structure of the Ordinary: Form and Control in the Built Environment*, 1. paperback ed, Cambridge, Mass.: MIT Press, 359 pp. (cit. on p. 2).
- Habraken, N. J. (July 1, 1987), “The Control of Complexity”, in: *Places* 4.2 (cit. on p. 19).
- Hamers, D., N. Bueno de Mesquita, A. Vaneycken and J. Schoffelen, eds. (2017), *Trading Places: Practices of Public Participation in Art and Design Research*, Barcelona: dpr-barcelona, 176 pp. (cit. on p. 24).
- Hardin, G. (Dec. 13, 1968), “The Tragedy of the Commons”, in: *Science* 162.3859, pp. 1243–1248 (cit. on p. 2).
- Harding, J. (2014), “Meta-Parametric Design Developing a Computational Approach for Early Stage Collaborative Practice”, in: (cit. on pp. 60, 88).
- Harding, J. and C. Derix (2011), “Associative Spatial Networks in Architectural Design: Artificial Cognition of Space Using Neural Networks with Spectral Graph Theory”, in: *Design Computing and Cognition '10*, ed. by Gero, J. S., Dordrecht: Springer Netherlands, pp. 305–323 (cit. on p. 61).
- Havemann, S. (2005), “Generative Mesh Modeling”, PhD thesis, Technischen Universität Braunschweig (cit. on pp. 47–49, 62, 64).
- Hays, K. M., ed. (1998), *Architecture Theory since 1968*, Cambridge, Mass: The MIT Press, 807 pp. (cit. on p. 19).
- Heinemann, C. (2011), “Situative Standards: What Co-Housing Looks like - inside Berlin’s Radical R50 Baugruppen Project” (London) (cit. on pp. 27, 28).
- Heisserman, J. (Mar. 1994), “Generative Geometric Design”, in: *IEEE Computer Graphics and Applications* 14.2, pp. 37–45 (cit. on pp. 66, 192).
- Heisserman, J., R. Mattikalli and S. Callahan (2004), “A Grammatical Approach to Design Generation and Its Application to Aircraft Systems”, in: *Proc. Generative CAD Systems Symp*, vol. 4 (cit. on p. 66).
- Heisserman, J. and R. Woodbury (June 1, 1993), “Generating Languages of Solid Models”, in: *Proceedings on the Second ACM Symposium on Solid Modeling and Applications*, SMA '93, New York, NY, USA: Association for Computing Machinery, pp. 103–112 (cit. on p. 66).
- Heisserman, J. A. (Jan. 1, 1990), “Generative Geometric Design and Boundary Solid Grammars”, Thesis Proposal, Carnegie Mellon University, Engineering Design Research Center (cit. on p. 66).
- Hendrickson, C. (2008), “Organizing for Project Management”, in: *Project Management for Construction: Fundamental Concepts for Owners, Engineers, Architects and Builders*, version 2., Pittsburgh: Department of Civil and Environmental Engineering, Carnegie Mellon University (cit. on p. 11).
- Hendrikx, M., S. Meijer, J. Van Der Velden and A. Iosup (Feb. 19, 2013), “Procedural Content Generation for Games: A Survey”, in: *ACM Transactions on Multimedia Computing, Communications, and Applications* 9.1, 1:1–1:22 (cit. on pp. 48, 50, 51, 60).

-
- Herr, C. M. and T. Kvan (2005), “Using Cellular Automata to Generate High-Density Building Form”, in: *Computer Aided Architectural Design Futures 2005*, ed. by Martens, B. and Brown, A., Dordrecht: Springer Netherlands, pp. 249–258 (cit. on p. 69).
- Herr, C. M. and T. Kvan (Jan. 1, 2007), “Adapting Cellular Automata to Support the Architectural Design Process”, in: *Automation in Construction, CAAD Futures*, 2005 16.1, pp. 61–69 (cit. on pp. 69, 98, 99).
- Herrmann, T. (2016), “Socio-Technical Procedures of Facilitated Mass Collaboration for Creative E-Participation”, in: *Mass Collaboration and Education*, ed. by Cress, U., Moskaliuk, J. and Jeong, H., Computer-Supported Collaborative Learning Series, Cham: Springer International Publishing, pp. 305–328 (cit. on p. 110).
- Heylighen, A. and H. Neuckermans (Dec. 1, 2001), “A Case Base of Case-Based Design Tools for Architecture”, in: *Computer-Aided Design* 33.14, pp. 1111–1122 (cit. on pp. 48, 54, 73, 74, 82).
- Heylighen, A. (2000), “In Case of Architectural Design. Critique and Praise of Case-Based Design in Architecture”, PhD thesis, KU Leuven, Fac. Toegepaste wetenschappen, Dep. architectuur, stedenbouw en ruimtelijke ordening: CUMINCAD (cit. on p. 78).
- Heylighen, A. and H. Neuckermans (Jan. 1, 2003), “(Learning from Experience)? Promises, Problems and Side-effects of Case-Based Reasoning in Architectural Design”, in: *International Journal of Architectural Computing* 1.1, pp. 60–70 (cit. on pp. 73, 78).
- Hillier, B. (Jan. 2007), *Space Is The Machine: A Configurational Theory Of Architecture* (cit. on pp. 65, 316).
- Hofmann, S. (2018), “Participative Architecture: The Way to More Environmental Justice”, in: *Architectural Design* 88.5, pp. 116–121 (cit. on pp. 3, 29, 30, 33).
- Hofmann, S. (2019), “Room for Play in Architecture: Participatory Games and Game-Like Tools in Architecture Developed by Die Baupiloten”, in: *The Routledge Companion to Games in Architecture and Urban Planning*, Routledge (cit. on pp. 3, 29, 33).
- Hoisl, F. and K. Shea (Nov. 2011), “An Interactive, Visual Approach to Developing and Applying Parametric Three-Dimensional Spatial Grammars”, in: *AI EDAM* 25.4, pp. 333–356 (cit. on pp. 47, 54, 67).
- Holland, J. H. (1975), *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, Ann Arbor: University of Michigan Press, 183 pp. (cit. on pp. 60, 77).
- Hoskins, B., D. Kerr, H. J. Abs, J. G. Janmaat, J. Morrison, R. Ridley and J. Sizmur (2012), *Participatory Citizenship in the European Union Institute of Education*, 2, European Commission, Europe for Citizens Programme, p. 103 (cit. on p. 3).
- Howe, A. S., I. Ishii and T. Yoshida (2017), “Kit-of-Parts: A Review of Object-Oriented Construction Techniques”, in: *Proceedings of the 16th IAARC/IFAC/IEEE International Symposium on Automation and Robotics in Construction*, pp. 165–171 (cit. on p. 4).
- Hsu, Y. C. and R. J. Krawczyk (2003), “New Generation of Computer Aided Design in Space Planning Methods: A Survey and a Proposal”, in: *Proceedings of the 8th International Conference on Computer Aided Architectural Design Research in Asia* 1, pp. 101–115 (cit. on p. 54).

- Hu, R., Z. Huang, Y. Tang, O. V. Kaick, H. Zhang and H. Huang (2020), “Graph2Plan: Learning Floorplan Generation from Layout Graphs”, in: *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2020)* 39.4 (cit. on pp. 72, 73, 77, 80, 100, 295, 296).
- Hua, K., B. Fairings and I. Smith (Jan. 1, 1996), “CADRE: Case-Based Geometric Design”, in: *Artificial Intelligence in Engineering* 10.2, pp. 171–183 (cit. on p. 73).
- Hua, K. and B. Faltings (May 1993), “Exploring Case-Based Building Design—CADRE”, in: *AI EDAM* 7.2, pp. 135–143 (cit. on p. 75).
- Hunicke, R., M. LeBlanc and R. Zubek (2004), “MDA: A Formal Approach to Game Design and Game Research”, in: *Proceedings of the AAAI Workshop on Challenges in Game AI*, vol. 4, 1, San Jose, CA, p. 1722 (cit. on p. 134).
- Hutter, K., J. Hautz, J. Füller, J. Mueller and K. Matzler (Mar. 2011), “Communitition: The Tension between Competition and Collaboration in Community-Based Design Contests”, in: *Creativity and Innovation Management* 20.1, pp. 3–21 (cit. on pp. 122, 123).
- Iľčík, M. and M. Wimmer (2016), “Collaborative Modeling with Symbolic Shape Grammars”, in: *Proceedings of eCAADe 2016*, pp. 417–426 (cit. on p. 64).
- Irwin, A. (1995), *Citizen Science: A Study of People, Expertise and Sustainable Development*, London: Routledge, 216 pp. (cit. on p. 114).
- Jackson, H. (Jan. 1, 2002), “Chapter 11 - Toward a Symbiotic Coevolutionary Approach to Architecture”, in: *Creative Evolutionary Systems*, ed. by Bentley, P. J. and Corne, D. W., The Morgan Kaufmann Series in Artificial Intelligence, San Francisco: Morgan Kaufmann, pp. 299–313 (cit. on p. 82).
- Jansen, S. J. T., H. C. C. H. Coolen and R. W. Goetgeluk, eds. (2011), *The Measurement and Analysis of Housing Preference and Choice*, Springer Netherlands (cit. on p. 298).
- Järvinen, A. (2008), “Games without Frontiers: Theories and Methods for Game Studies and Design”, PhD thesis, Tampere University (cit. on p. 134).
- Jensen, P., T. Olofsson, E. Smiding and R. Gerth (2014), “Developing Products in Product Platforms in the AEC Industry”, in: *Computing in Civil and Building Engineering - Proceedings of the 2014 International Conference on Computing in Civil and Building Engineering* (June), pp. 1062–1069 (cit. on p. 4).
- Jensen, P., H. Lidelöw and T. Olofsson (2015), “Product Configuration in Construction”, in: *International Journal of Mass Customisation* 5.1, p. 73 (cit. on pp. 4, 327).
- Jobs, S., director (1990), *Steve Jobs Talks about the Library of Congress 1990* (cit. on pp. 29, 130).
- Johnson, B. R. (Nov. 18, 2016), *Design Computing: An Overview of an Emergent Field*, Taylor & Francis, 221 pp. (cit. on pp. 36, 83).
- Johnson, B. R. (2017), *Design Computing: An Overview of an Emergent Field* (cit. on p. 19).
- Jowers, I., D. C. Hogg, A. McKay, H. H. Chau and A. de Pennington (Oct. 1, 2010), “Shape Detection with Vision: Implementing Shape Grammars in Conceptual Design”, in: *Research in Engineering Design* 21.4, pp. 235–247 (cit. on p. 67).
- Juul, J. (2005), *Half-Real: Video Games between Real Rules and Fictional Worlds*, Cambridge, Mass.: MIT Press, 233 pp. (cit. on p. 131).

-
- Kado, K. (1995), “An Investigation of Genetic Algorithms for Facility Layout Problems”, MA thesis, University of Edinburgh (cit. on pp. 61, 62, 82).
- Kalay, Y. E. (May 1, 2006), “The Impact of Information Technology on Design Methods, Products and Practices”, in: *Design Studies*, Digital Design 27.3, pp. 357–380 (cit. on p. 35).
- Kalay, Y. E. and G. Carrara (1996), “A Performance-Based Paradigm of Design”, in: *Advances in Formal Design Methods for CAD: Proceedings of the IFIP WG5.2 Workshop on Formal Design Methods for Computer-Aided Design, June 1995*, ed. by Gero, J. S. and Sudweeks, F., IFIP — The International Federation for Information Processing, Boston, MA: Springer US, pp. 107–135 (cit. on pp. 19, 73, 329).
- Karth, I. and A. M. Smith (2017), “WaveFunctionCollapse Is Constraint Solving in the Wild”, in: *Proceedings of the 12th International Conference on the Foundations of Digital Games*, FDG ’17, New York, NY, USA: ACM, 68:1–68:10 (cit. on pp. 68, 72, 82, 94, 296).
- Kelly, G. and H. McCabe (2006), “A Survey of Procedural Techniques for City Generation”, in: *The ITB Journal* 7.2 (cit. on pp. 47–49, 61, 78).
- Kelly, K. (1994), *Out of Control: The New Biology of Machines, Social Systems and the Economic World*, 1., paperback printing, March, Reading, Mass.: Addison-Wesley, 521 pp. (cit. on p. 109).
- Kelly, T. (2013), “Unwritten Procedural Modeling with the Straight Skeleton”, PhD thesis, University of Glasgow (cit. on pp. 47, 48, 51, 62–64, 67, 69–71).
- Kelty, C. M. (2019), *The Participant: A Century of Participation in Four Stories*, Chicago ; London: The University of Chicago Press, 326 pp. (cit. on pp. 1, 13, 14, 23–25, 33).
- Ketteler, G. and M. Lenart (1992), “The Design of Building Parts by Using Knowledge Based Systems”, in: *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, ed. by Belli, F. and Radermacher, F. J., Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, pp. 655–665 (cit. on p. 73).
- Khokhlov, M., I. Koh and J. Huang (2019), “Voxel Synthesis for Generative Design”, in: *Design Computing and Cognition ’18*, ed. by Gero, J. S., Cham: Springer International Publishing, pp. 227–244 (cit. on p. 296).
- Kim, H. S. and S. B. Cho (2000), “Application of Interactive Genetic Algorithm to Fashion Design”, in: *Engineering Applications of Artificial Intelligence* 13.6, pp. 635–644 (cit. on p. 88).
- Kittur, A. and R. E. Kraut (2008), “Harnessing the Wisdom of Crowds in Wikipedia: Quality through Coordination”, in: *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work*, CSCW ’08, New York, NY, USA: Association for Computing Machinery, pp. 37–46 (cit. on p. 118).
- Knecht, K. and R. Koenig (2010), “Generating Floor Plan Layouts with k-d Trees and Evolutionary Algorithms”, in: *13th Generative Art Conference GA2010*, pp. 238–253 (cit. on pp. 61, 62, 82).
- Knight, T. (1999), *Applications in Architectural Design, and Education and Practice* (cit. on pp. 47, 192).
- Knight, T. (Feb. 1, 2003), “Computing with Emergence”, in: *Environment and Planning B: Planning and Design* 30.1, pp. 125–155 (cit. on p. 67).

- Knöll, M. (2018), “Mobile Partizipation in der gesundheitsfördernden Stadtgestaltung: Zwei Fallbeispiele zu Datenerfassung und Interaktion im Stadtraum”, in: *Planung für gesundheitsfördernde Städte*, ed. by Baumgart, S., Köckler, H., Ritzinger, A. and Rüdiger, A., Hannover: Verlag der ARL - Akademie für Raumforschung und Landesplanung, pp. 387–401 (cit. on pp. 3, 134).
- Koch, H. von (1904), “Sur une courbe continue sans tangente, obtenue par une construction géométrique élémentaire”, in: *Arkiv för Matematik, Astronomi och Fysik* 1, pp. 681–702 (cit. on p. 61).
- Koile, K. (1997), “Design Conversations with Your Computer: Evaluating Experiential Qualities of Physical Form”, in: *CAAD Futures 1997*, ed. by Junge, R., Dordrecht: Springer Netherlands, pp. 203–218 (cit. on p. 95).
- Kolodner, J. L. (Mar. 1, 1992), “An Introduction to Case-Based Reasoning”, in: *Artificial Intelligence Review* 6.1, pp. 3–34 (cit. on p. 73).
- Koltsova, A., G. Schmitt, P. Schumacher, T. Sudo, S. Narang and L. Chen (2011), “A Case Study of Script-Based Techniques in Urban Planning”, in: *Design Computing and Cognition '10*, ed. by Gero, J. S., Dordrecht: Springer Netherlands, pp. 681–700 (cit. on p. 61).
- Koster, R. and W. Wright (2013), *A Theory of Fun for Game Design*, 2nd ed, Sebastopol, CA: O'Reilly, 279 pp. (cit. on p. 131).
- Krawczyk, R. (2002a), “Architectural Interpretation of Cellular Automata”, in: *GENERATIVE ART 2002, 5th International Conference GA2002, Milan, 11-12-13 December 2002*, Generative Art 2002, ed. by Soddu, C., Politecnico di Milano University, Italy (cit. on p. 69).
- Krawczyk, R. (Oct. 8, 2002b), “Experiments in Architectural Form Generation Using Cellular Automata”, in: (cit. on p. 69).
- Krishnamurti, R. a. S. (1993), “Spatial Grammars: Motivation, Comparison, and New Results”, in: *CAAD Futures '93 [Conference Proceedings / ISBN 0-444-89922-7] (Pittsburgh / USA), 1993, Pp. 57-74*, CUMINCAD (cit. on pp. 64–66).
- Lange, T. (2016), “Rittel’s Riddles: Design Education and “Democratic” Planning in the Age of Information”, in: *Re-Scaling the Environment* (cit. on pp. 3, 4, 13, 14, 20, 21, 25, 35, 42, 83, 84).
- Langford, W., A. Ghassaei and N. Gershenfeld (Sept. 27, 2016), “Automated Assembly of Electronic Digital Materials”, in: ASME 2016 11th International Manufacturing Science and Engineering Conference, American Society of Mechanical Engineers Digital Collection (cit. on pp. 107, 334).
- Langford, W. K. (2019), “Discrete Robotic Construction”, PhD thesis, Massachusetts Institute of Technology (cit. on p. 106).
- Lanier, J. (2011), *You Are Not a Gadget: A Manifesto*, 1st Vintage Books ed, New York: Vintage Books, 223 pp. (cit. on p. 124).
- Lanier, J. (2013), *Who Owns the Future?*, First Simon & Schuster hardcover edition, New York: Simon & Schuster, 396 pp. (cit. on p. 350).
- Lara-Cabrera, R., C. Cotta and A. J. Fernandez-Leiva (2013), “A Review of Computational Intelligence in RTS Games”, in: *Proceedings of the 2013 IEEE Symposium on Foundations of Computational Intelligence, FOCI 2013 - 2013 IEEE Symposium Series on Computational Intelligence, SSCI 2013* (September 2014), pp. 114–121 (cit. on pp. 47, 48, 51, 70, 74, 77).

-
- Larsen, M. S. S., S. M. Lindhard, T. D. Brunoe, K. Nielsen and J. K. Larsen (2019), “Mass Customization in the House Building Industry: Literature Review and Research Directions”, in: *Frontiers in Built Environment* 5 (October), pp. 1–12 (cit. on p. 55).
- Lawson, B. (2006a), *How Designers Think*, Routledge (cit. on p. 19).
- Lawson, B. (2006b), “The Changing Role of the Designer”, in: *How Designers Think: The Design Process Demystified*, How Designers Think: The Design Process Demystified, Elsevier/Architectural, pp. 17–30 (cit. on p. 19).
- Lechner, T., B. Watson, U. Wilensky and M. Felsen (2003), *Procedural City Modeling*, NWU-CS-04-38, Evanston, IL: Northwestern University, Computer Science department (cit. on p. 69).
- Leder, S., R. Weber, O. Bucklin, D. Wood and A. Menges (Dec. 16, 2019), “Towards Distributed In Situ Robotic Timber Construction”, in: *Research Culture in Architecture*, Birkhäuser, pp. 67–76 (cit. on pp. 1, 2).
- Lee, J.-H. (2002), “Integrating Housing Design and Case-Based Reasoning”, PhD thesis, School of Architecture and Institute for Complex Engineered Systems (ICES), Carnegie Mellon University (cit. on p. 73).
- Lee, J. H., M. J. Ostwald and N. Gu (Jan. 1, 2018), “A Justified Plan Graph (JPG) Grammar Approach to Identifying Spatial Design Patterns in an Architectural Style”, in: *Environment and Planning B: Urban Analytics and City Science* 45.1, pp. 67–89 (cit. on pp. 64, 65).
- Leimeister, J. M. (2012), “Crowdsourcing: Crowdfunding, Crowdvoting, Crowdcreation”, in: *Zeitschrift für Controlling und Management (ZFCM)* 56.2012, pp. 388–392 (cit. on p. 111).
- Levy, Y. and T. J. Ellis (Sept. 2006), “A Systems Approach to Conduct an Effective Literature Review in Support of Information Systems Research”, in: (cit. on p. 55).
- Liapis, A., G. N. Yannakakis and J. Togelius (Sept. 2012), “Adapting Models of Visual Aesthetics for Personalized Content Creation”, in: *IEEE Transactions on Computational Intelligence and AI in Games* 4.3, pp. 213–228 (cit. on pp. 88, 89).
- Liapis, A., G. N. Yannakakis and J. Togelius (June 2014), “Computational Game Creativity”, in: (cit. on p. 131).
- Liggett, R. S. (Mar. 1, 2000), “Automated Facilities Layout: Past, Present and Future”, in: *Automation in Construction* 9.2, pp. 197–215 (cit. on pp. 47, 48, 68, 298).
- Lin, Z. and Z. Yingjie (Apr. 2019), “Solving the Facility Layout Problem with Genetic Algorithm”, in: *2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA)*, 2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA), pp. 164–168 (cit. on p. 298).
- Lintott, C. J., K. Schawinski, A. Slosar, K. Land, S. Bamford, D. Thomas, M. J. Raddick, R. C. Nichol, A. Szalay, D. Andreescu, P. Murray and J. Vandenberg (Sept. 21, 2008), “Galaxy Zoo: Morphologies Derived from Visual Inspection of Galaxies from the Sloan Digital Sky Survey*”, in: *Monthly Notices of the Royal Astronomical Society* 389.3, pp. 1179–1189 (cit. on pp. 114, 115).
- Lipp, M., P. Wonka and M. Wimmer (2008), “Interactive Visual Editing of Grammars for Procedural Architecture”, in: *SIGGRAPH’08: International Conference*

- on Computer Graphics and Interactive Techniques, ACM SIGGRAPH 2008 Papers 2008* (cit. on p. 93).
- Lobos, D. and D. Donath (2010), “The Problem of Space Layout in Architecture: A Survey and Reflections”, in: *Arquitetura Revista* 6.2, pp. 136–161 (cit. on pp. 47–49, 68, 72).
- Lorensen, W. E. and H. E. Cline (1987), “Marching Cubes: A High Resolution 3D Surface Construction Algorithm”, in: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’87, New York, NY, USA: ACM, pp. 163–169 (cit. on pp. 63, 195, 280).
- Lubart, T. (Oct. 1, 2005), “How Can Computers Be Partners in the Creative Process: Classification and Commentary on the Special Issue”, in: *International Journal of Human-Computer Studies*, Computer Support for Creativity 63.4, pp. 365–369 (cit. on pp. 83, 84).
- Lynch, K. (1960), *The Image of the City*, MIT Press, 204 pp. (cit. on p. 21).
- Maas, W. (2007), *Spacefighter. The Evolutionary City (Game)*. MVRDV/DSD, New York: Actar-D (cit. on p. 134).
- Mackworth, A. K. (Feb. 1, 1977), “Consistency in Networks of Relations”, in: *Artificial Intelligence* 8.1, pp. 99–118 (cit. on p. 68).
- Macy, M. W. and R. Willer (Aug. 2002), “From Factors to Actors: Computational Sociology and Agent-Based Modeling”, in: *Annual Review of Sociology* 28.1, pp. 143–166 (cit. on p. 69).
- Mahlmann, T., J. Togelius and G. N. Yannakakis (2012), “Spicing Up Map Generation”, in: *Applications of Evolutionary Computation*, ed. by Di Chio, C., Agapitos, A., Cagnoni, S., Cotta, C., de Vega, F. F., Di Caro, G. A., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A. I., Farooq, M., Langdon, W. B., Merelo-Guervós, J. J., Preuss, M., Richter, H., Silva, S., Simões, A., Squillero, G., Tarantino, E., Tettamanzi, A. G. B., Togelius, J., Urquhart, N., Uyar, A. Ş. and Yannakakis, G. N., Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, pp. 224–233 (cit. on p. 82).
- Majchrzak, A. and A. Malhotra (Dec. 1, 2013), “Towards an Information Systems Perspective and Research Agenda on Crowdsourcing for Innovation”, in: *The Journal of Strategic Information Systems* 22.4, pp. 257–268 (cit. on p. 4).
- Malmgren, L., P. Jensen and T. Olofsson (2011), “Product Modeling of Configurable Building Systems - A Case Study”, in: *Electronic Journal of Information Technology in Construction* 16 (October), pp. 697–712 (cit. on p. 4).
- Mandelbrot, B. B. (1977), *Fractal: Form, Chance and Dimension* (cit. on p. 61).
- Mandelbrot, B. B. (1982), *The Fractal Geometry of Nature*, San Francisco: W.H. Freeman, 460 pp. (cit. on p. 61).
- Marsh, L. and C. Onof (Mar. 1, 2008), “Stigmergic Epistemology, Stigmergic Cognition”, in: *Cognitive Systems Research*, Perspectives on Social Cognition 9.1, pp. 136–149 (cit. on p. 122).
- Martin, J. (2006), “Procedural House Generation : A Method for Dynamically Generating Floor Plans”, in: *Symposium on Interactive 3D Graphics and Games* (cit. on p. 192).
- Matheson, J. A. L. (1971), *Hyperstatic Structures: An Introduction to the Theory of Statically Indeterminate Structures*, 2nd ed, London: Butterworths, 500 pp. (cit. on p. 152).

-
- May, F., S. Ullrich and K. Steiger (2017), *Gemeinsam bauen: Baugruppen, Baugemeinschaften, Wege und Erfahrungen*, Das Gebäude, Berlin Offenbach: VDE Verlag GmbH, 122 pp. (cit. on p. 25).
- Mayer, I. S. (Dec. 1, 2009), “The Gaming of Policy and the Politics of Gaming: A Review”, in: *Simulation & Gaming* 40.6, pp. 825–862 (cit. on p. 29).
- McCormack, J. P. and J. Cagan (Dec. 1, 2002), “Supporting Designers’ Hierarchies through Parametric Shape Recognition”, in: *Environment and Planning B: Planning and Design* 29.6, pp. 913–931 (cit. on pp. 66, 67, 191, 193).
- McCormack, J. P. and J. Cagan (Aug. 1, 2006), “Curve-Based Shape Matching: Supporting Designers’ Hierarchies through Parametric Shape Recognition of Arbitrary Geometry”, in: *Environment and Planning B: Planning and Design* 33.4, pp. 523–540 (cit. on pp. 66, 67, 191–193).
- McCormack, J., A. Dorin and T. Innocent (Nov. 17, 2004), “Generative Design: A Paradigm for Design Research.”, in: *Redmond, J., Durling, D. and de Bono, A (Eds.), Futureground - DRS International Conference 2004, 17-21 November, Melbourne, Australia* (cit. on p. 101).
- McDaniel, T. (Apr. 2018), “Block by Block: The Use of the Video Game ”Minecraft” as a Tool to Increase Public Participation”, MA thesis, San Marcos, TX: Texas State University (cit. on p. 31).
- McGonigal, J. (2011), *Reality Is Broken: Why Games Make Us Better and How They Can Change the World*, New York: Penguin Press, 388 pp. (cit. on p. 136).
- McKay, A., S. Chase, K. Shea and H. H. Chau (May 2012), “Spatial Grammar Implementation: From Theory to Useable Software”, in: *AI EDAM* 26.2, pp. 143–159 (cit. on p. 67).
- McNair, M. P. and A. C. Hersum (1954), *The Case Method at the Harvard Business School: Papers by Present and Past Members of the Faculty and Staff*, McGraw-Hill (cit. on p. 29).
- Měch, R. and P. Prusinkiewicz (Aug. 1, 1996), “Visual Models of Plants Interacting with Their Environment”, in: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’96, New York, NY, USA: Association for Computing Machinery, pp. 397–410 (cit. on p. 64).
- Mees, C. (2017), *Participatory Design and Self-building in Shared Urban Open Spaces: Community Gardens and Casitas in New York City*, 1st ed. 2017, Urban Agriculture, Cham: Springer International Publishing : Imprint: Springer, 1 p. (cit. on p. 32).
- Meller, R. D. and K.-Y. Gau (Jan. 1, 1996), “The Facility Layout Problem: Recent and Emerging Trends and Perspectives”, in: *Journal of Manufacturing Systems* 15.5, pp. 351–366 (cit. on pp. 47, 54).
- Menges, A. (Feb. 2012), “Biomimetic Design Processes in Architecture: Morphogenetic and Evolutionary Computational Design”, in: *Bioinspiration & Biomimetics* 7.1, p. 015003 (cit. on pp. 61, 63, 77).
- Meniru, K., H. Rivard and C. Bédard (Jan. 1, 2003), “Specifications for Computer-Aided Conceptual Building Design”, in: *Design Studies* 24.1, pp. 51–71 (cit. on p. 19).
- Merrell, P. (Apr. 30, 2007), “Example-Based Model Synthesis”, in: *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D ’07, New York, NY, USA: Association for Computing Machinery, pp. 105–112 (cit. on pp. 72, 141, 296, 342).

- Merrell, P. and D. Manocha (Dec. 1, 2008), “Continuous Model Synthesis”, in: *ACM SIGGRAPH Asia 2008 Papers*, SIGGRAPH Asia ’08, New York, NY, USA: Association for Computing Machinery, pp. 1–7 (cit. on pp. 72, 296).
- Merrell, P. and D. Manocha (2009), “Constraint-Based Model Synthesis”, in: (January), p. 101 (cit. on p. 296).
- Merrell, P., E. Schkufza and V. Koltun (2010), “Computer-Generated Residential Building Layouts”, in: *ACM Transactions on Graphics* 29.6, p. 1 (cit. on pp. 70, 82, 86, 295, 296, 332).
- Merrell, P. C. (2009), “Model Synthesis”, PhD thesis, University of North Carolina (cit. on pp. 47, 54, 72).
- Meyer, M. H. and A. P. Lehnerd (1997), *The Power of Product Platforms: Building Value and Cost Leadership*, New York: Free Press (cit. on p. 327).
- Michalatos, P. (2016), “Design Signals: The Role of Software Architecture and Paradigms in Design Thinking and Practice”, in: *Architectural Design* 86.5, pp. 108–115 (cit. on pp. 175, 176, 206, 344).
- Michalek, J. J., R. Choudhary and P. Y. Papalambros (2002), “Architectural Layout Design Optimization”, in: *Engineering Optimization* 34.5, pp. 461–484 (cit. on p. 11).
- Michalek, J. J. and P. Y. Papalambros (2002), “Interactive Design Optimization of Architectural Layouts”, in: *Engineering Optimization* 34.5, pp. 485–501 (cit. on p. 69).
- Miller, W. R. (June 22, 1970), “Computer-Aided Space Planning”, in: *Proceedings of the 7th Design Automation Workshop*, DAC ’70, New York, NY, USA: Association for Computing Machinery, pp. 28–34 (cit. on p. 46).
- Mitchell, W. J. (Jan. 1, 1975), “Techniques of Automated Design in Architecture: A Survey and Evaluation”, in: *Computers & Urban Society* 1.1, pp. 49–76 (cit. on pp. 45, 47, 48).
- Moore, B. (Feb. 8, 2014), “Inside the Epic Online Space Battle That Cost Gamers \$300,000”, in: *Wired* (cit. on p. 132).
- Mountstephens, J. and J. Teo (2020), “Progress and Challenges in Generative Product Design: A Review of Systems”, in: *Computers* 9.4, pp. 1–23 (cit. on pp. 44–48, 54, 77, 83, 84).
- Movshovitz-Attias, D., Y. Movshovitz-Attias, P. Steenkiste and C. Faloutsos (2013), “Analysis of the Reputation System and User Contributions on a Question Answering Website: StackOverflow”, in: *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2013*, pp. 886–893 (cit. on p. 135).
- Müller, P., P. Wonka, S. Haegler, A. Ulmer and L. Van Gool (2006), “Procedural Modeling of Buildings”, in: *ACM SIGGRAPH 2006 Papers, SIGGRAPH ’06*, pp. 614–623 (cit. on pp. 64, 191, 192, 194).
- Müller, P., G. Zeng, P. Wonka and L. Van Gool (July 29, 2007), “Image-Based Procedural Modeling of Facades”, in: *ACM Transactions on Graphics* 26.3, 85–es (cit. on p. 72).
- Nacke, L. E. and S. Deterding (2017), “The Maturing of Gamification Research”, in: *Computers in Human Behavior* 71, pp. 450–454 (cit. on p. 135).
- Nagl, M. (Mar. 1, 1976), “Formal Languages of Labelled Graphs”, in: *Computing* 16.1, pp. 113–137 (cit. on p. 65).

-
- Nam, J.-H. and T.-H. Le (Dec. 1, 2012), “Automatic Interior Space Arrangement of Mid-Sized Superyachts Using a Constraint-Based Genetic Algorithm”, in: *Journal of Marine Science and Technology* 17.4, pp. 481–492 (cit. on p. 45).
- Negroponte, N. (1970), *The Architecture Machine* (cit. on pp. 35, 46, 84).
- Neuckermans, H. M. W. (2007), “Data and Metadata in Architectural Repositories”, in: *CAADRIA 2007 [Proceedings of the 12th International Conference on Computer Aided Architectural Design Research in Asia] Nanjing (China) 19-21 April 2007*, CUMINCAD (cit. on p. 73).
- Newell, A., J. C. Shaw and H. A. Simon (1959), “Report on a General Problem Solving Program”, in: *IFIP Congress*, vol. 256, Pittsburgh, PA, p. 64 (cit. on p. 330).
- Nisztuk, M. and P. B. Myszkowski (2018), “Usability of Contemporary Tools for the Computational Design of Architectural Objects: Review, Features Evaluation and Reflection”, in: *International Journal of Architectural Computing* 16.1 (cit. on pp. 47, 48, 53, 295).
- Oeberst, A., U. Cress, M. Back and S. Nestler (2016), “Individual Versus Collaborative Information Processing: The Case of Biases in Wikipedia”, in: *Mass Collaboration and Education*, pp. 165–185 (cit. on p. 122).
- Oosterhuis, K. (Jan. 1, 2001), “Game, Set, and Match”, in: *Oz* 23.1 (cit. on p. 131).
- Oosterhuis, K. and L. Feireiss, eds. (2006), *The Architecture Co-Laboratory: Game Set And Match II. On Computer Games, Advanced Geometries, and Digital Technologies*, Rotterdam: Episode Publishers, 615 pp. (cit. on p. 131).
- Oosterhuis, K., I. J. Hubers, M. van Veen, C. Kievid, M. Fox and M. Engeli (2003), *Game Set and Match I (Conference)* (cit. on p. 131).
- Open Systems Lab (2019), “The DfMA Housing Manual”, in: p. 50 (cit. on p. 1).
- Opoku, A. (2016), “SDG2030: A Sustainable Built Environment’s Role in Achieving the Post-2015 United Nations Sustainable Development Goals”, in: *Proceedings of the 32nd Annual ARCOM Conference*, vol. 2, Association of Researchers in Construction Management Manchester, UK, pp. 1149–1158 (cit. on p. 1).
- Orsborn, S., J. Cagan and P. Boatwright (Jan. 1, 2008), “A Methodology for Creating a Statistically Derived Shape Grammar Composed of Non-Obvious Shape Chunks”, in: *Research in Engineering Design* 18.4, pp. 181–196 (cit. on p. 68).
- Ostwald, M. J. and M. J. Dawes (2018), *The Mathematics of the Modernist Villa*, vol. 3 (cit. on p. 316).
- Otten, R. H. J. M. (June 1982), “Automatic Floorplan Design”, in: *19th Design Automation Conference*, 19th Design Automation Conference, pp. 261–267 (cit. on p. 61).
- Oxman, R. (1990), “Architectural Knowledge Structures as ”Design Shells”: A Knowledge-Based View of Design and CAAD Education”, in: *The Electronic Design Studio: Architectural Knowledge and Media in the Computer Era [CAAD Futures ’89 Conference Proceedings / ISBN 0-262-13254-0] Cambridge (Massachusetts / USA), 1989, Pp. 187-199*, CUMINCAD (cit. on pp. 73, 75).
- Oxman, R. E. (1992), “Multiple Operative and Interactive Modes in Knowledge-Based Design Systems”, in: *New York: John Wiley & Sons, 1992. Pp. 125-143 : Ill. Includes Bibliography*, CUMINCAD (cit. on p. 96).
- Paoluzzi, A., V. Pascucci and M. Vicentino (July 1, 1995), “Geometric Programming: A Programming Approach to Geometric Design”, in: *ACM Transactions on Graphics* 14.3, pp. 266–306 (cit. on p. 62).

- Papoutsoglou, M., G. M. Kapitsaki and L. Angelis (Dec. 1, 2020), “Modeling the Effect of the Badges Gamification Mechanism on Personality Traits of Stack Overflow Users”, in: *Simulation Modelling Practice and Theory* 105, p. 102157 (cit. on p. 135).
- Parish, Y. I. H. and P. Müller (2001), “Procedural Modeling of Cities”, in: *In Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, Press, pp. 301–308 (cit. on pp. 64, 80, 91, 102, 192).
- Parvin, A. (2013), “Architecture (and the Other 99%): Open-Source Architecture and Design Commons”, in: *Architectural Design* 83.6, pp. 90–95 (cit. on pp. 1, 2, 104, 328).
- Patow, G. (Mar. 1, 2012), “User-Friendly Graph Editing for Procedural Modeling of Buildings”, in: *IEEE Computer Graphics and Applications* 32.02, pp. 66–75 (cit. on p. 93).
- Pavlidis, T. (Jan. 1972), “Linear and Context-Free Graph Grammars”, in: *Journal of the ACM* 19.1, pp. 11–22 (cit. on p. 65).
- Peplow, M. (May 1, 2016), “Citizen Science Lures Gamers into Sweden’s Human Protein Atlas”, in: *Nature Biotechnology* 34.5 (5), pp. 452–453 (cit. on p. 132).
- Pérez-Gosende, P., J. Mula and M. Díaz-Madroñero (2021), “Facility Layout Planning. An Extended Literature Review”, in: *International Journal of Production Research* 0.0, pp. 1–40 (cit. on pp. 47, 54, 298).
- Perlin, K. (July 1, 1985), “An Image Synthesizer”, in: *ACM SIGGRAPH Computer Graphics* 19.3, pp. 287–296 (cit. on p. 60).
- Petersen, K. H., R. Nagpal and J. K. Werfel (2011), “TERMES: An Autonomous Robotic System for Three-Dimensional Collective Construction”, in: *Robotics: Science and Systems VII*, ed. by Durrant-Whyte, H., Roy, N. and Abbeel, P., MIT Press (cit. on p. 106).
- Pfaltz, J. L. and A. Rosenfeld (May 7, 1969), “Web Grammars”, in: *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, IJCAI’69, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 609–619 (cit. on p. 65).
- Picon, A. (2014), “Robots and Architecture: Experiments, Fiction, Epistemology”, in: *Architectural Design* 84.3, pp. 54–59 (cit. on p. 1).
- Pine, B. J. (1993), “Mass Customizing Products and Services”, in: *Planning Review* 21.4, pp. 6–55 (cit. on pp. 41, 140).
- Popescu, G. A. (2007), “Digital Materials for Digital Fabrication”, MA thesis, Massachusetts Institute of Technology (cit. on p. 106).
- Poplin, A. (May 1, 2012), “Playful Public Participation in Urban Planning: A Case Study for Online Serious Games”, in: *Computers, Environment and Urban Systems* 36.3, pp. 195–206 (cit. on pp. 3, 33).
- Quiroz, J. C., S. J. Louis, A. Banerjee and S. M. Dascalu (2009), “Towards Creative Design Using Collaborative Interactive Genetic Algorithms”, in: *2009 IEEE Congress on Evolutionary Computation*, pp. 1849–1856 (cit. on pp. 88, 90).
- Raddick, M. J., G. Bracey, P. L. Gay, C. J. Lintott, P. Murray, K. Schawinski, A. S. Szalay and J. Vandenberg (Dec. 2010), “Galaxy Zoo: Exploring the Motivations of Citizen Science Volunteers”, in: *Astronomy Education Review* 9.1 (cit. on p. 100).

-
- Rani, M., R. U. Haq and N. Sulaiman (July 23, 2011), “Koch Curves: Rewriting System, Geometry and Application”, in: *Journal of Computer Science* 7.9 (9), pp. 1330–1334 (cit. on p. 61).
- Rappaz, J., M. Catasta, R. West and K. Aberer (June 15, 2018), “Latent Structure in Collaboration: The Case of Reddit r/Place”, in: *Proceedings of the International AAAI Conference on Web and Social Media* 12.1 (1) (cit. on pp. 5, 117).
- Ratti, C. and M. Claudel (2015), “Why It Did Not Work: A Horse Designed by Committee”, in: *Open Source Architecture*, Thames & Hudson, pp. 45–61 (cit. on p. 124).
- Reeves, B. and J. L. Read (2009), *Total Engagement: Using Games and Virtual Worlds to Change the Way People Work and Businesses Compete* (cit. on p. 137).
- Retsin, G. (2016), “Discrete Assembly and Digital Materials in Architecture”, in: *Ecaade 2016*, vol. 1, pp. 143–151 (cit. on p. 106).
- Retsin, G. (2020), “Discrete Timber Assembly”, in: *Fabricate 2020*, pp. 264–271 (cit. on pp. 103, 106, 107).
- Reynolds, C. W. (Aug. 1, 1987), “Flocks, Herds and Schools: A Distributed Behavioral Model”, in: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’87, New York, NY, USA: Association for Computing Machinery, pp. 25–34 (cit. on p. 69).
- Richter, K. H. (2007), “Looking Back to the Future: An Updated Case Base of Case-Based Design Tools for Architecture”, in: *Predicting the Future [25th eCAADe Conference Proceedings / ISBN 978-0-9541183-6-5] Frankfurt Am Main (Germany) 26-29 September 2007, Pp. 285-292*, CUMINCAD (cit. on pp. 48, 73, 74, 82).
- Rifkin, J. (2015), *The Zero Marginal Cost Society: The Internet of Things, the Collaborative Commons, and the Eclipse of Capitalism*. New York: Palgrave Macmillan (cit. on pp. 1, 14).
- Ring, K., ed. (2007), *Auf.Einander.Bauen: Baugruppen in Der Stadt*, Berlin: Jovis, 95 pp. (cit. on p. 25).
- Ripperda, N. and C. Brenner (2009), “Application of a Formal Grammar to Facade Reconstruction in Semiautomatic and Automatic Environments”, in: *12th AG-ILE International Conference on Geographic Information Science 2009*, Leibniz Universität Hannover, Germany, p. 12 (cit. on p. 72).
- Rittel, H. W. J. and M. M. Webber (1973), “Dilemmas in a General Theory of Planning”, in: *Policy Sciences* 4.2, pp. 155–169 (cit. on pp. 3, 4, 20, 25, 83, 84).
- Roberts, N. (2001), “Wicked Problems and Network Approaches to Resolution”, in: *International Public Management Review* 1 (cit. on pp. 1, 3–5, 20–22, 324, 343, 344).
- Rodrigues, E. (2014), “Automated Floor Plan Design: Generation, Simulation, and Optimization”, PhD thesis, FACULTY OF SCIENCES AND TECHNOLOGY, UNIVERSITY OF COIMBRA (cit. on pp. 47, 48, 51, 52).
- Rossi, A. and O. Tessmann (Sept. 28, 2017a), “Aggregated Structures: Approximating Topology Optimized Material Distribution with Discrete Building Blocks”, in: *Proceedings of IASS Annual Symposia* 2017.23, pp. 1–10 (cit. on p. 71).
- Rossi, A. and O. Tessmann (2017b), “Collaborative Assembly of Digital Materials”, in: *ACADIA 2017: DISCIPLINES & DISRUPTION [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture*

- (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017), Pp. 512- 521, CUMINCAD (cit. on p. 334).
- Rossi, A. and O. Tessmann (2017c), “Designing with Digital Materials”, in: *Protocols, Flows and Glitches; Proceedings of the 22nd International Conference of the Association for Computer-Aided Architectural Design Research in Asia CAADRIA 2017*, ed. by Janssen, P. L., Raonic, A. and Schnabel, M. A., Hong Kong: The Association for Computer-Aided Architectural Design Research in Asia (CAADRIA), pp. 37–42 (cit. on pp. 106, 206, 334).
- Rossi, A. and O. Tessmann (2017d), “Geometry as Assembly”, in: *eCAADe Conference — Sharing of Computable Knowledge*, ed. by Fioravanti, A., Elahmar, S., Gargaro, S., Loffreda, G., Novembri, G., Trento, A. and Cursi, S., vol. 2, Rome: eCAADe (Education and Research in Computer Aided Architectural Design in Europe) and Dep. of Civil, Building and Environmental Engineering, Faculty of Civil and Industrial Engineering, Sapienza University of Rome, p. 9 (cit. on pp. 106, 192).
- Rossi, A. and O. Tessmann (2018), “From Voxels to Parts: Hierarchical Discrete Modeling for Design and Assembly”, in: *ICGG 2018 - Proceedings of the 18th International Conference on Geometry and Graphics*, ed. by Cocchiarella, L., Advances in Intelligent Systems and Computing, Cham: Springer International Publishing, pp. 1001–1012 (cit. on pp. 106, 192, 334).
- Ruales, M. S. (2017), “Collective Intelligence Is the Architect: Automatized Space Layout Planning from Preceding Typologies”, MA thesis, Bartlett School of Graduate Studies, UCL (cit. on pp. 47, 54).
- Rudofsky, B. (1964), *Architecture without Architects: An Introduction to Nonpedigreed Architecture*, Architecture Without Architects: A Short Introduction to Non-pedigreed Architecture, Museum of Modern Art; distributed by Doubleday, Garden City, N.Y. (cit. on pp. 32, 122, 350, 351).
- Ruiz-Montiel, M., J. Boned, J. Gavilanes, E. Jiménez, L. Mandow and J.-L. Pérez-de-la-Cruz (Apr. 2013), “Design with Shape Grammars and Reinforcement Learning”, in: *Advanced Engineering Informatics* 27.2, pp. 230–245 (cit. on pp. 191, 194, 203).
- Russell, S. J. and P. Norvig (1995), *Artificial Intelligence: A Modern Approach*, Prentice Hall Series in Artificial Intelligence, Englewood Cliffs, N.J: Prentice Hall, 932 pp. (cit. on p. 76).
- Sanchez, J. (2013), “Gamescapes”, in: *ACADIA 2013 Adaptive Architecture : Proceedings of the 33rd Annual Conference of the Association for Computer Aided Design in Architecture*, ed. by Beesley, P., Stacey, M. and Khan, O., Toronto: Riverside Architectural Press, pp. 207–216 (cit. on p. 128).
- Sanchez, J. (2015), “Block’hood - Developing an Architectural Simulation Video Game”, in: *Real Time - Proceedings of the 33rd eCAADe Conference - Volume 1*, pp. 89–97 (cit. on pp. 85, 134).
- Sanchez, J. (2021), *Architecture for the Commons: Participatory Systems in the Age of Platforms*, Routledge (cit. on pp. 1, 2, 103).
- Sanfeliu, A. and K.-S. Fu (May 1983), “A Distance Measure between Attributed Relational Graphs for Pattern Recognition”, in: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.3, pp. 353–362 (cit. on p. 307).
- Sanoff, H. (1990), *Participatory Design: Theory & Techniques*, North Carolina: North Carolina State University (cit. on pp. 3, 25, 33).

-
- Sanoff, H. (Apr. 1, 2011), “Multiple Views of Participatory Design”, in: *Focus* 8.1 (cit. on pp. 3, 25, 33).
- Sanz-Blas, S., S. Tena-Monferrer and J. Sánchez-García (2015), “Crowdsourcing: An Application of Promotional Marketing”, in: *Advances in Crowdsourcing*, ed. by Garrigos-Simon, F. J., Gil-Pechuán, I. and Estelles-Miguel, S., Cham: Springer International Publishing, pp. 147–161 (cit. on p. 111).
- Sasaki, M., T. Itō and A. Isozaki (2007), *Morphogenesis of Flux Structure*, London: AA Publ, 111 pp. (cit. on p. 71).
- Savov, A., B. Buckton and O. Tessmann (2016), “20,000 Blocks: Can Gameplay Be Used to Guide Non-Expert Groups in Creating Architecture?”, in: *ACADIA 2016: POSTHUMAN FRONTIERS: Data, Designers, and Cognitive Machines [Proceedings of the 36th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)]*, Ann Arbor, pp. 24–33 (cit. on p. iii).
- Savov, A. and O. Tessmann (2017), “Introduction to Playable Voxel-Shape Grammars”, in: *Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, Cambridge, pp. 534–543 (cit. on pp. iii, 67).
- Savov, A., O. Tessmann and S. A. Nielsen (2016), “Sensitive Assembly: Gamifying the Design and Assembly of Facade Wall Prototypes”, in: *International Journal of Architectural Computing* 14.1, pp. 30–48 (cit. on p. iii).
- Savov, A., R. Winkler and O. Tessmann (2020), “Encoding Architectural Designs as Iso-surface Tilesets for Participatory Sculpting of Massing Models”, in: *Impact: Design With All Senses*, ed. by Gengnagel, C., Baverel, O., Burry, J., Ramsgaard Thomsen, M. and Weinzierl, S., Cham: Springer International Publishing, pp. 199–213 (cit. on pp. iii, 82).
- Schall, D. (2012), *Service-Oriented Crowdsourcing*, SpringerBriefs in Computer Science, New York, NY: Springer New York (cit. on p. 111).
- Schank, R. C. and R. P. Abelson (1988), “Scripts, Plans, Goals, and Understanding”, in: *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*. San Mateo, CA, US: Morgan Kaufmann, p. 223 (cit. on p. 73).
- Schaur, E. (1991), *Non-Planned Settlements*, Institut für Leichte Flächentragwerke (cit. on pp. 32, 122, 350).
- Schell, J. (2008), *The Art of Game Design: A Book of Lenses*, Amsterdam ; Boston: Elsevier/Morgan Kaufmann, 489 pp. (cit. on p. 131).
- Schindler, S. (2017), “1966 CAN BE THE YEAR OF REBIRTH FOR AMERICAN CITIES”, in: *San Rocco* 66 (cit. on p. 25).
- Schindler, S. (Nov. 2020), “The Institutions Must Be Designed Before the Buildings”, in: *Onus* 53, pp. 110–134 (cit. on p. 25).
- Schramm, H. (2008), *Low Rise - High Density : Horizontale Verdichtungsformen Im Wohnbau*, Wien: Springer, 178 pp. (cit. on p. 12).
- Schrandt, R. G. and S. M. Ulam (1967), *Recursively Defined Geometrical Objects and Patterns of Growth*, Los Alamos Scientific Lab., N. Mex. (cit. on p. 69).
- Schreiner, K. (2008), “Digital Games Target Social Change”, in: *{IEEE} Computer Graphics and Applications* 28.1, pp. 12–17 (cit. on p. 134).
- Schulz, A., H. Wang, E. Grinspun, J. Solomon and W. Matusik (July 30, 2018), “Interactive Exploration of Design Trade-Offs”, in: *ACM Transactions on Graphics* 37.4, 131:1–131:14 (cit. on pp. 83, 84).

- Seaborn, K. and D. I. Fels (July 1, 2015), “Gamification in Theory and Action: A Survey”, in: *International Journal of Human Computer Studies* 74, pp. 14–31 (cit. on p. 136).
- Secretan, J., N. Beato, D. B. D Ambrosio, A. Rodriguez, A. Campbell and K. O. Stanley (Apr. 6, 2008), “Picbreeder: Evolving Pictures Collaboratively Online”, in: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, New York, NY, USA: Association for Computing Machinery, pp. 1759–1768 (cit. on p. 141).
- Secretan, J., N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, J. T. Folsom-Kovarik and K. O. Stanley (Sept. 1, 2011), “Picbreeder: A Case Study in Collaborative Evolutionary Exploration of Design Space”, in: *Evolutionary Computation* 19.3, pp. 373–403 (cit. on p. 142).
- Segard, A., T. Moleta and J. Moloney (2013), “Open Communitition – Competitive Design in a Collaborative Virtual Environment”, in: (cit. on p. 176).
- Shang, Y. (Jan. 1, 2005), “5 - Expert Systems”, in: *The Electrical Engineering Handbook*, ed. by Chen, W.-K., Burlington: Academic Press, pp. 367–377 (cit. on p. 79).
- Shapiro, R. B. (2016), “Toward Participatory Discovery Networks: A Critique of Current Mass Collaboration Environments and a Possible Learning-Rich Future”, in: *Mass Collaboration and Education*, pp. 187–207 (cit. on pp. 116, 126).
- Sharma, J. and R. S. Singhal (2014), “Genetic Algorithm and Hybrid Genetic Algorithm for Space Allocation Problems - A Review”, in: *International Journal of Computer Applications* 95.4, pp. 33–37 (cit. on pp. 47, 48, 51).
- Sicart, M. (Dec. 2008), “Defining Game Mechanics”, in: *Game Studies* 8.2 (cit. on p. 134).
- Silva, P. B., P. Müller, R. Bidarra and A. Coelho (2013), “Node-Based Shape Grammar Representation and Editing”, in: *Proceedings of the Workshop on Procedural Content Generation in Games (PCG'13)*, pp. 1–8 (cit. on p. 93).
- Silver, M. (2006), “Building without Drawings: Automason Ver 1.0”, in: *Architectural Design* 76.4, pp. 46–51 (cit. on p. 69).
- Simon, H. (1979), *Models of Thought*, New Haven, Ct: Yale Univ. Pr, 524 pp. (cit. on p. 330).
- Simonsen, J. and T. Robertson, eds. (2013), *Routledge International Handbook of Participatory Design*, New York: Routledge, 294 pp. (cit. on pp. 3, 25, 33).
- Sinclair, D., I. Davies, M. Davys, R. Fairhead, P. Holden, A. Kell, J. Orrell, B. Gething, P. Fletcher, S. Chalmers and A. Dobson (2013), *RIBA Plan of Work 2013* (cit. on p. 11).
- Singh, V. and N. Gu (Mar. 2012), “Towards an Integrated Generative Design Framework”, in: *Design Studies* 33.2, pp. 185–207 (cit. on pp. 19, 47–49, 64, 69, 74, 77, 82, 88, 98).
- Smelik, R., T. Tutenel, K. J. de Kraker and R. Bidarra (June 18, 2010), “Integrating Procedural Generation and Manual Editing of Virtual Worlds”, in: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, PCGames '10, New York, NY, USA: Association for Computing Machinery, pp. 1–8 (cit. on p. 87).
- Smelik, R. M., T. Tutenel, R. Bidarra and B. Benes (2014), “A Survey on Procedural Modelling for Virtual Worlds”, in: *Computer Graphics Forum* 33.6, pp. 31–50 (cit. on pp. 47, 48, 51, 61, 63, 66, 68, 69, 71, 72, 82, 87).

-
- Smith, C. and G. Levermore (Dec. 1, 2008), “Designing Urban Spaces and Buildings to Improve Sustainability and Quality of Life in a Warmer World”, in: *Energy Policy*, Foresight Sustainable Energy Management and the Built Environment Project 36.12, pp. 4558–4562 (cit. on p. 1).
- Smith, I., C. Lottaz and B. Faltings (1995), “Spatial Composition Using Cases: IDIOM”, in: *Case-Based Reasoning Research and Development*, ed. by Veloso, M. and Aamodt, A., Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, pp. 88–97 (cit. on pp. 73, 74).
- Smith, I., R. Stalker and C. Lottaz (1996), “Creating Design Objects from Cases for Interactive Spatial Composition”, in: *Artificial Intelligence in Design '96*, ed. by Gero, J. S. and Sudweeks, F., Dordrecht: Springer Netherlands, pp. 97–116 (cit. on pp. 73, 76).
- Snyder, J. M. and J. T. Kajiya (July 1, 1992), “Generative Modeling: A Symbolic System for Geometric Modeling”, in: *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '92, New York, NY, USA: Association for Computing Machinery, pp. 369–378 (cit. on p. 62).
- Sobey, A., P. Grudniewski and T. Savasta (2021), “Genetic Algorithm Selection for Ship Concept Design”, in: *Practical Design of Ships and Other Floating Structures*, ed. by Okada, T., Suzuki, K. and Kawamura, Y., Lecture Notes in Civil Engineering, Singapore: Springer, pp. 156–172 (cit. on p. 45).
- Soler, V. (July 28, 2021), *Visose/Robots* (cit. on p. 182).
- Speller, T. H., D. Whitney and E. Crawley (2007), “Using Shape Grammar to Derive Cellular Automata Rule Patterns”, in: *Complex Systems* 17 (cit. on p. 82).
- Srinet, K., Y. Jernite, J. Gray and A. Szlam (July 2020), “CraftAssist Instruction Parsing: Semantic Parsing for a Voxel-World Assistant”, in: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online: Association for Computational Linguistics, pp. 4693–4714 (cit. on p. 141).
- Stalberg, O. (2016), “Voxel House: Breaking down the Voxel House Demo”, in: *Vertex 3*, pp. 170–175 (cit. on pp. 269, 283).
- Stiny, G. (Dec. 1, 1980a), “Kindergarten Grammars: Designing with Froebel’s Building Gifts”, in: *Environment and Planning B: Planning and Design* 7.4, pp. 409–462 (cit. on pp. 66, 94, 190, 192, 193).
- Stiny, G. (Mar. 1, 1982), “Spatial Relations and Grammars”, in: *Environment and Planning B: Planning and Design* 9.1, pp. 113–114 (cit. on pp. 66, 192, 193).
- Stiny, G. (Aug. 1, 1992), “Weights”, in: *Environment and Planning B: Planning and Design* 19.4, pp. 413–430 (cit. on p. 63).
- Stiny, G. and W. J. Mitchell (1978), “The Palladian Grammar”, in: *Environment and Planning B: Planning and Design* 5.1, pp. 5–18 (cit. on pp. 66, 86, 191, 192, 295, 296).
- Stiny, G. (1980b), “Introduction to Shape and Shape Grammars”, in: *Environment and planning B* 7.3, pp. 343–351 (cit. on pp. 67, 91, 192, 194–196).
- Stiny, G. and J. Gips (1971), “Shape Grammars and the Generative Specification of Painting and Sculpture”, in: *Information Processing 71 Proceedings of the IFIP Congress 1971. Volume 2* 71, pp. 1460–1465 (cit. on pp. 59, 60, 66, 191, 192, 206).
- Stone, R. B. and K. L. Wood (1999), “Development of a Functional Basis for Design”, in: *Proceedings of DETC99 1999 ASME Design Engineering Technical*

- Conferences*, ASME 1999 Design Engineering Technical Conferences, American Society of Mechanical Engineers Digital Collection, pp. 261–275 (cit. on p. 79).
- Strahringer, S., C. L. Hrsg, J. Hofmann and M. Knoll (2017), *Gamification Und Serious Games*, Springer (cit. on p. 134).
- Sun, J., X. Yu, G. Baciú and M. Green (Nov. 11, 2002), “Template-Based Generation of Road Networks for Virtual City Modeling”, in: *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST ’02, New York, NY, USA: Association for Computing Machinery, pp. 33–40 (cit. on p. 78).
- Surowiecki, J. (2004), *The Wisdom of Crowds: Why the Many Are Smarter than the Few and How Collective Wisdom Shapes Business, Economies, Societies, and Nations*, 1st ed, New York: Doubleday, 296 pp. (cit. on p. 122).
- Tan, E. (Aug. 13, 2014), “Negotiation and Design for the Self-Organizing City: Gaming as a Method for Urban Design”, in: *A+BE — Architecture and the Built Environment* 11 (11), pp. 1–454 (cit. on pp. 29, 30).
- Tan, E. (2017), *Play the City: Games Informing the Urban Development*, Heijningen: Jap Sam Books, 392 pp. (cit. on p. 29).
- Tching, J., J. Reis and A. Paio (Jun-2016), “A Cognitive Walkthrough towards an Interface Model for Shape Grammar Implementations”, in: *Computer Science and Information Technology* 4.3, pp. 92–119 (cit. on p. 94).
- Tessmann, O. and A. Rossi (Sept. 17, 2019), “Geometry as Interface: Parametric and Combinatorial Topological Interlocking Assemblies”, in: *Journal of Applied Mechanics* 86.111002 (cit. on pp. 106, 107).
- Tichelmann, K. U., D. Blome, M. T. Ringwald, M. Günther and K. Groß (2019), *Wohnraumpotenziale in Urbanen Lagen: Aufstockung Und Umnutzung von Nichtwohngebäuden*, Technische Universität Darmstadt (cit. on p. 12).
- Tobler, W. R. (1970), “A Computer Movie Simulating Urban Growth in the Detroit Region”, in: *Economic Geography* 46, pp. 234–240 (cit. on p. 47).
- Togelius, J., M. Preuss, N. Beume, S. Wessing, J. Hagelbäck and G. N. Yannakakis (Aug. 2010), “Multiobjective Exploration of the StarCraft Map Space”, in: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, pp. 265–272 (cit. on p. 47).
- Togelius, J., N. Shaker and J. Dormans (2016), “Grammars and L-systems with Applications to Vegetation and Levels”, in: *Procedural Content Generation in Games*, ed. by Shaker, N., Togelius, J. and Nelson, M. J., Computational Synthesis and Creative Systems, Cham: Springer International Publishing, pp. 73–98 (cit. on p. 91).
- Togelius, J., G. N. Yannakakis, K. O. Stanley and C. Browne (2010), “Search-Based Procedural Content Generation”, in: *Applications of Evolutionary Computation*, ed. by Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcazar, A. I., Goh, C.-K., Merelo, J. J., Neri, F., Preuß, M., Togelius, J. and Yannakakis, G. N., red. by Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y. and Weikum, G., vol. 6024, Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 141–150 (cit. on pp. 47–49, 59, 60, 76, 88).
- Togelius, J., G. N. Yannakakis, K. O. Stanley and C. Browne (Sept. 2011), “Search-Based Procedural Content Generation: A Taxonomy and Survey”, in: *IEEE*

-
- Transactions on Computational Intelligence and AI in Games* 3.3, pp. 172–186 (cit. on pp. 48, 58, 74, 76, 82, 85, 88, 101, 194).
- Tomoko, S., ed. (2010), *Total Housing: Alternatives to Urban Sprawl*, Barcelona: Actar, 396 pp. (cit. on p. 12).
- Trescak, T., M. Esteva and I. Rodriguez (Aug. 2009), “General Shape Grammar Interpreter for Intelligent Designs Generations”, in: *Imaging and Visualization 2009 Sixth International Conference on Computer Graphics*, Imaging and Visualization 2009 Sixth International Conference on Computer Graphics, pp. 235–240 (cit. on p. 342).
- Tromba, P. (2013), “Build Engagement and Knowledge One Block at a Time with Minecraft”, in: *Learning & Leading with Technology* 40.8, pp. 20–23 (cit. on p. 29).
- Tyflopoulos, E., F. D. Tollnes, M. Steinert and A. Olsen (2018), “State of the Art of Generative Design and Topology Optimization and Potential Research Needs”, in: *DS 91: Proceedings of NordDesign 2018, Linköping, Sweden, 14th - 17th August 2018*, NordDesign 2018 (cit. on pp. 47, 54, 70).
- Ulam, S. (1962), “On Some Mathematical Problems Connected with Patterns of Growth of Figures”, in: *Proceedings of Symposia in Applied Mathematics*, vol. 14, Am. Math. Soc. Vol. 14, Providence, pp. 215–224 (cit. on p. 69).
- Umeda, Y., M. Ishii, M. Yoshioka, Y. Shimomura and T. Tomiyama (Sept. 1996), “Supporting Conceptual Design Based on the Function-Behavior-State Modeler”, in: *AI EDAM* 10.4, pp. 275–288 (cit. on p. 79).
- Vachher, P., Z. Levonian, H.-F. Cheng and S. Yarosh (Oct. 17, 2020), “Understanding Community-Level Conflicts Through Reddit r/Place”, in: *Conference Companion Publication of the 2020 on Computer Supported Cooperative Work and Social Computing*, CSCW ’20 Companion, New York, NY, USA: Association for Computing Machinery, pp. 401–405 (cit. on pp. 6, 118).
- Vähä, P., T. Heikkilä, P. Kilpeläinen, M. Järviluoma and E. Gambao (Dec. 1, 2013), “Extending Automation of Building Construction — Survey on Potential Sensor Technologies and Robotic Applications”, in: *Automation in Construction* 36, pp. 168–178 (cit. on p. 106).
- Vanegas, C. A., D. G. Aliaga, B. Benes and P. A. Waddell (Dec. 1, 2009), “Interactive Design of Urban Spaces Using Geometrical and Behavioral Modeling”, in: *ACM Transactions on Graphics* 28.5, pp. 1–10 (cit. on pp. 98, 100).
- Vierra, D. (2014), *Mcedit/Pymclevel*, MCEdit, Minecraft World Editor (cit. on p. 179).
- Von Ahn, L. (Mar. 9, 2011), “Three Human Computation Projects”, in: *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, SIGCSE ’11, New York, NY, USA: Association for Computing Machinery, pp. 691–692 (cit. on pp. 111, 338).
- Von Neumann, J. (1951), “The General and Logical Theory of Automata”, in: *1951*, pp. 1–41 (cit. on p. 69).
- Von Neumann, J. (1966), *Theory of Self-Reproducing Automata*, ed. by Burks, A. W., vol. 1102024, University of Illinois press Urbana (cit. on p. 69).
- Voronoi, G. (1908), “Nouvelles Applications Des Paramètres Continus à La Théorie Des Formes Quadratiques. Premier Mémoire. Sur Quelques Propriétés Des Formes Quadratiques Positives Parfaites.”, in: (cit. on p. 61).

- Wagner, H. J., M. Alvarez, A. Groenewolt and A. Menges (Dec. 1, 2020), “Towards Digital Automation Flexibility in Large-Scale Timber Construction: Integrative Robotic Prefabrication and Co-Design of the BUGA Wood Pavilion”, in: *Construction Robotics* 4.3, pp. 187–204 (cit. on p. 103).
- Wagner, H. J., M. Alvarez, O. Kyjanek, Z. Bhiri, M. Buck and A. Menges (Dec. 1, 2020), “Flexible and Transportable Robotic Timber Construction Platform – TIM”, in: *Automation in Construction* 120, p. 103400 (cit. on pp. 1, 103, 105).
- Walz, S. P. (2010), *Toward a Ludic Architecture: The Space of Play and Games*, Pittsburgh, PA: ETC Press, 382 pp. (cit. on p. 131).
- Wang, Z., Q. She and T. E. Ward (2019), “Generative Adversarial Networks: A Survey and Taxonomy”, in: *arXiv* (November), pp. 1–41 (cit. on p. 77).
- Ward, J. (2010), “Additive Assembly of Digital Materials”, MA thesis, Massachusetts Institute of Technology (cit. on p. 106).
- Waters, C. N., J. Zalasiewicz, C. Summerhayes, A. D. Barnosky, C. Poirier, A. Gałuszka, A. Cearreta, M. Edgeworth, E. C. Ellis, M. Ellis, C. Jeandel, R. Leinfelder, J. R. McNeill, D. deB. Richter, W. Steffen, J. Syvitski, D. Vidas, M. Wapreisch, M. Williams, A. Zhisheng, J. Grinevald, E. Odada, N. Oreskes and A. P. Wolfe (Jan. 8, 2016), “The Anthropocene Is Functionally and Stratigraphically Distinct from the Holocene”, in: *Science* 351.6269, aad2622 (cit. on p. 2).
- Watson, I. and S. Perera (Jan. 1997), “Case-Based Design: A Review and Analysis of Building Design Applications”, in: *AI EDAM* 11.1, pp. 59–87 (cit. on pp. 47, 48, 54, 73).
- Wei, L.-Y., S. Lefebvre, V. Kwatra and G. Turk (2009), “State of the Art in Example-Based Texture Synthesis”, in: *Eurographics 2009, State of the Art Report, EG-STAR*, Eurographics Association (cit. on p. 72).
- Wender, S. and I. Watson (2014), “Combining Case-Based Reasoning and Reinforcement Learning for Unit Navigation in Real-Time Strategy Game AI”, in: *Case-Based Reasoning Research and Development*, ed. by Lamontagne, L. and Plaza, E., Lecture Notes in Computer Science, Cham: Springer International Publishing, pp. 511–525 (cit. on pp. 74, 82).
- Wibranek, B. (2021), “Robotic Digital Reassembly: Towards Physical Editing of Dry Joined Architectural Aggregations”, PhD thesis, TU Darmstadt (cit. on p. 8).
- Wiener, N. (1950), *The Human Use of Human Beings, Cybernetics and Society. Norbert Wiener...* Da Capo Press, Incorporated, 241 pp. (cit. on p. 109).
- Wikipedia (Jan. 18, 2019), *List of 3D Modeling Software*, in: *Wikipedia* (cit. on p. 40).
- Wilken, R. (2016), “The De-Gamification of Foursquare”, in: *Social, Casual and Mobile Games: The Changing Gaming Landscape*, ed. by Leaver, T. and Willson, M. A., New York: Bloomsbury Academic (cit. on p. 136).
- Woetzel, J., S. Ram, J. Mischke, N. Garemo and S. Sankhe (Oct. 2014), *A Blueprint for Addressing the Global Affordable Housing Challenge: Executive Summary*, McKinsey Global Institute (cit. on p. 1).
- Wolfram, S. (July 1, 1983), “Statistical Mechanics of Cellular Automata”, in: *Reviews of Modern Physics* 55.3, pp. 601–644 (cit. on p. 69).
- Wonka, P., M. Wimmer, F. Sillion and W. Ribarsky (2003), “Instant Architecture”, in: *ACM Transactions on Graphics* 22.3, pp. 669–677 (cit. on pp. 65–67, 82, 83, 91, 102, 191, 192, 194, 342).

-
- Woodbury, R. (Jan. 1, 2016), “An Introduction to Shape Schema Grammars”, in: *Environment and Planning B: Planning and Design* 43.1, pp. 152–183 (cit. on p. 192).
- Wortmann, T. and R. Stouffs (2018), “Algorithmic Complexity of Shape Grammar Implementation”, in: *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM* 32.2 (cit. on p. 192).
- Wright, F. L. (1927), “In the Cause of Architecture: The Architect and the Machine”, in: *Architectural Record* (cit. on p. 35).
- Wunderlich, J. T. and J. J. Wunderlich (2011), “CrowdSourced Architecture and Environmental Design”, in: (cit. on p. 132).
- Wynn, M. (1985), *Planning Games: Case Study Simulations in Land Management and Development*. London: E. & F. N. Spon Ltd. (cit. on pp. 29, 128).
- Yaneva, A. (2009), *Made by the Office for Metropolitan Architecture: An Ethnography of Design*, Rotterdam: 010 Publishers, 111 pp. (cit. on p. 19).
- Yannakakis, G. N. and J. Togelius (July 2011), “Experience-Driven Procedural Content Generation”, in: *IEEE Transactions on Affective Computing* 2.3, pp. 147–161 (cit. on p. 47).
- Yannakakis, G. N. and J. Togelius (Dec. 2015), “A Panorama of Artificial and Computational Intelligence in Games”, in: *IEEE Transactions on Computational Intelligence and AI in Games* 7.4, pp. 317–335 (cit. on pp. 48, 52, 53).
- Yin, R. K. (2014), *Case Study Research: Design and Methods*, Fifth edition, Los Angeles: SAGE, 282 pp. (cit. on p. 8).
- Yin, X., H. Liu, Y. Chen and M. Al-Hussein (May 1, 2019), “Building Information Modelling for Off-Site Construction: Review and Future Directions”, in: *Automation in Construction* 101, pp. 72–91 (cit. on p. 79).
- Yogev, O., A. A. Shapiro and E. K. Antonsson (Apr. 2010), “Computational Evolutionary Embryogeny”, in: *IEEE Transactions on Evolutionary Computation* 14.2, pp. 301–325 (cit. on p. 68).
- Yoon, D. and K.-J. Kim (Nov. 26, 2012), “3D Game Model and Texture Generation Using Interactive Genetic Algorithm”, in: *Proceedings of the Workshop at SIGGRAPH Asia, WASA '12*, New York, NY, USA: Association for Computing Machinery, pp. 53–58 (cit. on p. 88).
- Yu, X. and M. Gen (2010), *Introduction to Evolutionary Algorithms*, red. by Roy, R., vol. 0, Decision Engineering, London: Springer London (cit. on pp. 77, 78).
- Zawadzki, M. and J. Szklarski (2020), “Multi-Objective Optimization of the Floor Plan of a Single Story Family House Considering Position and Orientation”, in: *Advances in Engineering Software* 141 (January), p. 102766 (cit. on pp. 295, 296).
- Zboinska, M. A. (Feb. 1, 2015), “Hybrid CAD/E Platform Supporting Exploratory Architectural Design”, in: *Computer-Aided Design* 59, pp. 64–84 (cit. on p. 97).
- Zhao, Z., S. Ali Etemad, A. Arya and A. Whitehead (2016), “Usability and Motivational Effects of a Gamified Exercise and Fitness System Based on Wearable Devices”, in: *Design, User Experience, and Usability: Novel User Experiences*, ed. by Marcus, A., Lecture Notes in Computer Science, Cham: Springer International Publishing, pp. 333–344 (cit. on p. 135).
- Zichermann, G. and C. Cunningham (2011), *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*, 1st. ed, Sebastopol, Calif: O’Reilly Media, 182 pp. (cit. on p. 135).

Online Sources

- ArchDaily (2015), *'Low Rise High Density' Exhibition*, URL: <http://www.archdaily.com/364494/low-rise-high-density-exhibition/> (cit. on p. 12).
- Autodesk (2021), *Optioneering*, Generative Design Primer, URL: https://www.generativedesign.org/02-deeper-dive/02-02_optioneering (visited on 07/30/2021) (cit. on p. 341).
- Barton, M. and B. Loguidice (May 5, 2009), *The History of Rogue: Have @ You, You Deadly Zs*, Gamasutra, URL: https://www.gamasutra.com/view/feature/132404/the_history_of_rogue_have_you.php (visited on 05/02/2021) (cit. on pp. 47, 359).
- Bauhaus Spirit (2016), *BAUKRAFT – DER MINECRAFT-WETTBEWERB Gropiusstadt reloaded*. URL: <https://bauhaus-spirit.com/de/b-lab.html> (visited on 01/22/2022) (cit. on p. 324).
- BBC News (2012), *Minecraft to Aid UN Regeneration Projects*, URL: <http://www.bbc.com/news/technology-20492908> (cit. on p. 128).
- Block by Block team (2021), *The Block by Block Methodology*, Block by Block, URL: <https://www.blockbyblock.org/resources> (visited on 06/01/2021) (cit. on pp. 3, 31, 33).
- Brown, M. (Mar. 6, 2014), *Kick the Architectural Competition Habit*, The Architect's Newspaper, URL: <https://www.archpaper.com/2014/03/kick-the-architectural-competition-habit/> (visited on 06/28/2020) (cit. on p. 125).
- Bryson, B. (2017), *Future of Architects: Extinction or Irrelevance*, URL: <https://www.di.net/articles/future-architects-extinction-irrelevance/> (visited on 06/28/2020) (cit. on p. 348).
- Chiang, H. (May 18, 2020), *Minecraft: Connecting More Players Than Ever Before*, Xbox Wire, URL: <https://news.xbox.com/en-us/2020/05/18/minecraft-connecting-more-players-than-ever-before/> (visited on 07/23/2021) (cit. on p. 175).
- Cook, G. (2011), *How Crowdsourcing Is Changing Science*, URL: <http://garethcook.net/how-crowdsourcing-is-changing-science/> (visited on 07/01/2020) (cit. on pp. 114, 125).
- CyLEDGE Media (2018), *Configurator Database*, URL: <https://www.configurator-database.com/> (visited on 12/16/2018) (cit. on p. 40).
- Davis, D. (2011), *The MacLeamy Curve*, URL: <https://www.danieldavis.com/macleamy/> (visited on 09/13/2020) (cit. on p. 11).
- Deutsches Architekturmuseum (DAM) (2021), *Frankfurt_2099 – Wettbewerb und Rahmenprogramm*, URL: https://dam-online.de/veranstaltung/frankfurt_2099/, %20https://dam-online.de/veranstaltung/frankfurt_2099/ (visited on 01/22/2022) (cit. on p. 324).
- Florian Köhl (2012), *BAUGEMEINSCHAFT / Fatkoehl Architekten*, URL: <https://fatkoehl.com/concept/baugemeinschaft/> (visited on 06/01/2021) (cit. on p. 25).
- Foldit Wiki Contributors (2021), *Foldit Wiki*, URL: https://foldit.fandom.com/wiki/Foldit_Wiki (visited on 06/14/2021) (cit. on p. 143).
- Fontan, J. (2019), *Architecture Phases Of Design Explained*, URL: <https://jorgefontan.com/architectural-design-phases/> (visited on 02/09/2020) (cit. on p. 11).

-
- Gershenfeld, N., A. Gershenfeld and J. Cutcher-Gershenfeld (Apr. 17, 2018), *Soon You'll Be Able to Make Anything. It'll Change Politics Forever*. POLITICO Magazine, URL: <https://politi.co/2H9gDrG> (visited on 06/21/2021) (cit. on p. 2).
- Howe, J. (2006), *Crowdsourcing: A Definition*, URL: https://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html (visited on 07/09/2020) (cit. on p. 110).
- Jefferies, D. (Feb. 8, 2018), *'Serious' Games Finally Find Their Fun*, Medium, URL: <https://howwegettonext.com/serious-games-finally-find-their-fun-f1fa6b39b1e7> (visited on 06/04/2021) (cit. on p. 134).
- Kockelkorn, A. and S. Schindler (2019), *Cooperative Conditions: Introduction Housing and the Agency of Nonspeculation*, URL: <https://cooperativeconditions.net/home/introduction> (visited on 05/31/2021) (cit. on p. 25).
- Kollar, P. (Aug. 12, 2016), *No Man's Sky Review*, Polygon, URL: <https://www.polygon.com/2016/8/12/12461520/no-mans-sky-review-ps4-playstation-4-pc-windows-hello-games-sony> (visited on 02/20/2022) (cit. on p. 83).
- Lalonde, L. (2021), *What Is the Difference Between Classification & Taxonomy? — Synonym*, URL: <https://classroom.synonym.com/difference-between-classification-taxonomy-10074596.html> (visited on 05/02/2021) (cit. on p. 58).
- MR + MRS HOMES* (2020), URL: <https://www.mrmrshomes.de> (visited on 08/17/2020) (cit. on p. 41).
- Newswatch Times (2013), *UN-Habitat Predicts Upsurge in City Dwellers by 2050*, URL: <http://www.mynewswatchtimesng.com/un-habitat-predicts-upsurge-city-dwellers-2050/> (cit. on pp. 1, 12).
- Nintendo WiiFit* (2015), URL: <http://www.nintendo.com/wiifit/> (visited on 08/05/2015) (cit. on p. 134).
- Pertigkiozoglou, E. (May 1, 2017), *The Generator Project: Cedric Price & John Frazer*, Medium, URL: <https://eliza-pert.medium.com/1976-22121bb498c4> (visited on 02/17/2020) (cit. on p. 38).
- Sauerbrei, C. (Aug. 31, 2015), *Baugruppenprojekt R50 in Berlin-Kreuzberg*, db deutsche bauzeitung, URL: <https://www.db-bauzeitung.de/architektur/baugruppenprojekt-r50-berlin-kreuzberg/> (visited on 05/29/2021) (cit. on p. 26).
- Sawako, K. and M. Panagiotis (2014), *Millipede*, URL: <http://sawapan.eu> (cit. on p. 179).
- The 2012 ACM Computing Classification System* (2012), URL: <https://www.acm.org/publications/class-2012> (visited on 05/02/2021) (cit. on p. 45).
- Tigas, P. and T. Hosmer (2021), *Spatial Assembly: Generative Architecture With Reinforcement Learning, Self Play and Tree Search*, URL: <http://arxiv.org/abs/2101.07579> (cit. on pp. 68, 296).
- UpsideLearning (2015), *Games vs Game-based Learning vs Gamification — Infographic*, UpsideLearning, URL: <https://www.upsidelearning.com/infographics/games-vs-game-based-learning-vs-gamification/> (visited on 06/13/2021) (cit. on p. 136).
- Warren, T. (2016), *Minecraft Sales Top 100 Million*, The Verge, URL: <http://www.theverge.com/2016/6/2/11838036/minecraft-sales-100-million> (cit. on p. 175).