



UvA-DARE (Digital Academic Repository)

Languages of games and play

Automating game design & enabling live programming

van Rozen, R.A.

Publication date

2020

Document Version

Final published version

License

Other

[Link to publication](#)

Citation for published version (APA):

van Rozen, R. A. (2020). *Languages of games and play: Automating game design & enabling live programming*.

General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.



Languages of Games and Play

Riemer van Rozen

Languages of Games and Play

Automating Game Design & Enabling Live Programming

Languages of Games and Play

Automating Game Design & Enabling Live Programming

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K. I. J. Maex
ten overstaan van een door het College voor Promoties
ingestelde commissie,
in het openbaar te verdedigen in de Agnietenkapel
op woensdag 19 februari 2020, te 10.00 uur

door

Riemer Andries van Rozen

geboren te Dokkum

Promotiecommissie:

Promotor:	prof. dr. P. Klint	Universiteit van Amsterdam
Copromotor:	prof. dr. T. van der Storm	Rijksuniversiteit Groningen
Overige leden:	dr. ir. A.R.E. Bidarra de Almeida	TU Delft
	prof. dr. ir. C.T.A.M. de Laat	Universiteit van Amsterdam
	prof. dr. R.V. van Nieuwpoort	Universiteit van Amsterdam
	prof. dr. M. de Rijke	Universiteit van Amsterdam
	prof. dr. H.L.M. Vangheluwe	Universiteit Antwerpen
	prof. dr. E.J. Whitehead	University of California Santa Cruz
Faculteit:	Faculteit der Natuurwetenschappen, Wiskunde en Informatica	



Centrum Wiskunde & Informatica



The work in this thesis has been carried out at Centrum Wiskunde & Informatica (CWI), in collaboration with the Amsterdam University of Applied Sciences (AUAS), under the auspices of the research school Institute for Programming research and Algorithmics (IPA) and has been supported by the NWO/SIA grants “Early Quality Assurance in Software Production”, “Automated Game Design” and “Live Game Design”.

CONTENTS

Contents	vii
Acknowledgements	ix
1 Introduction	3
1.1 Perspectives	3
1.2 Research Questions	4
1.3 Origin of the Chapters	10
1.4 Thesis Structure	12
1.5 Conclusion	13
2 Languages of Games and Play: A Systematic Mapping Study	17
2.1 Introduction	17
2.2 Research Vision	19
2.3 Methodology	23
2.4 Review Protocol	25
2.5 Research Areas	30
2.6 Research Perspectives	42
2.7 Challenges and Opportunities	120
2.8 Related Work	126
2.9 Threats to Validity	127
2.10 Conclusion	130
3 Analyzing Game Mechanics	135
3.1 Introduction	135
3.2 Micro-Machinations	136
3.3 MM AiR Framework	147
3.4 Case Study: SimWar	152
3.5 Conclusion	157
4 Adapting Game Mechanics	161
4.1 Introduction	161
4.2 Background	162
4.3 Adapting Game Mechanics	164
4.4 Case Study: AdapTower	170
4.5 Conclusion	178
5 Designing Game Mechanics with Patterns	183

5.1	Introduction	183
5.2	Related Work	184
5.3	Mechanics Design Assistant	188
5.4	Discussion	199
5.5	Conclusion	199
6	Toward Live Domain-Specific Languages	203
6.1	Introduction	203
6.2	From Text Differencing to Live Models at Run Time	205
6.3	TMDiff: Textual Model Diff	206
6.4	RMPatch: Generic Run-time Model Patching	216
6.5	Case Study: Live State Machine Language	219
6.6	Discussion and Related Work	227
6.7	Conclusion	232
7	Measuring Quality of Grammars for Procedural Level Generation	237
7.1	Introduction	237
7.2	Related Work	239
7.3	Grammars for Level Generation	240
7.4	Grammar Analysis and Debugging	244
7.5	Preliminary Evaluation	249
7.6	Discussion	251
7.7	Conclusion	252
	Bibliography	255
	Summary	295
	Samenvatting	299

ACKNOWLEDGEMENTS

Little did Paul Klint know that when he asked me survey “languages for games”, it would take me the better part of a decade to complete the work this entailed. Over the years, challenges were plentiful, and certainly not all related to research. Together, we discussed my teaching activities that happened in parallel and research proposals that kept my research going. We devised communication strategies for building a network and collaborating with industry. We exchanged thoughts on formulating questions, proposing solutions and contributing results. Paul, thank you for your advice and feedback. I have found our conversations highly educational, humorous and tremendously motivating. Of course, I also thank my copromotor Tijs van der Storm. Tijs, our collaboration on live programming has been an amazing experience that continues to inspire me. In addition, we enjoyed many memorable social events together. I mention ‘de Keet’ and the cargo bike.

We have motivated our research and validated our results together with game developers. For this, I thank our industry partners, specifically Loren (Knowingo) who introduced game development challenges to me, Joris (Ludomotion) with whom I collaborated on Micro-Machinations and Chapter 4 and Paul (Firebrush Studios).

For hosting me, and being an amazing work place, I thank the Software Analysis & Transformation group of CWI. Special thanks to group leader Jurgen, and in alphabetical order: Aiko, Ali, Anastasia, Aggelos, Alexander, Anya, Arnold, Ashim, Atze, Bas, Bert, Cleverton, Davy, Floor, Gauthier, Jan, Jeroen, Jouke, Lina, Magiel, Mark, Mauricio, Michael, Mike, Nikos, Pablo, Paul, Piotr, Robert, Rodin, Sunil, Susanne, Thomas, Thomas, Tim, Yanja, Ulyana and Vadim. We had great fun at Polder, Praethuys and ‘game night’. I thank my office mates for interesting conversations on diverse topics and for creating such a pleasant work environment. I enjoyed exchanging perspectives with Pablo on languages and politics, and confining dry humour to our office with Tim. I will remember Atze’s powerful laugh echoing in the halls of CWI.

In addition, I thank my AUAS colleagues of CREATE-IT applied research and Play & Civic Media: Ben, Daniël, Gabriele, Karel, Karin, Marije, Marcel, Maarten, Martijn, Martijn, Mirjam, Monique, Mossa, Nikolai, Paul, Ria, Sabine, Stefan, Svetlana, Tamara, et al. Of course, I thank Anders with whom I faced deadlines and deliverables of several applied research projects. In addition I thank Jacob (Fontys), and the spontaneous collective we call ‘PhD support group’: Saskia, Ahmed, Marije, et al. and our former colleagues Sander and Tim. I will remember the cheerfulness, picnicks on the roof, and to ‘go do things’.

During my PhD I lectured in several teaching groups. I thank my colleagues of HBO-ICT at the AUAS, to name a few: Alexander, Claar, Dennis, Dop, Eric, Ferry,

Frank, Gerke, Huub, Irshad, Jan, John, Karel, Karthik, Kees, Marcio, Marco, Marten, Martijn, Michel, Nico, Reza, Remco, Rosa, Ruud, et al. At the Master of Software Engineering of the University of Amsterdam I thank Ana, Clemens, Simon, et al.

I thank the students and alumni that worked with me, in particular Quinten, who is coauthor of Chapter 7.

For being welcoming and kind, I thank the Software Language Engineering and AI and Games communities. Their feedback has helped my research progress. I thank the committee members for their participation and for reviewing my thesis.

Because it is difficult to concisely and adequately express my thanks to everyone, I share two especially memorable events.

On a bright Summer day, Adelheid[†] surprised me with a wonderful gift from her desk drawer: a bottle of Oude Geuze, an exquisite artisanal beer of spontaneous fermentation. Not only is this my favourite beer, it is simply the best beer in the world.

One late evening, Daria and I were delighted by a surprise visit of Pablo and Geoffrey at our home. We enjoyed good cheese, beers and great conversations together. Pablo, you will always be welcome.

Finally, I thank my family and close friends. In particular, Wicher, Sjoerd and Rien[†], great friends I have known since high school. For her support and friendship I thank Ismênia. For their love and support, I thank my parents Annie and Alwin. Most of all, I am grateful to my love and life partner Daria, and our kids Pieter and Anna. A big hug!

INTRODUCTION

Digital games are a powerful means for creating enticing, beautiful, educational, and often highly addictive interactive experiences that impact the lives of billions of players worldwide. Developing high quality digital games in a time-to-market manner is hard because game design is intrinsically complex.

The motivations for this dissertation are twofold. The first is to empower game designers with languages and tools that automate and speed up game design processes (“automated game design”). The second is to explore how such languages and tools can be created (“metaprogramming”).

Here, we introduce the main concepts, formulate and motivate our research questions, describe our research methods and contributions, and give an outline of the overall structure of this thesis.

1.1 PERSPECTIVES

The contributions of this thesis relate at three levels of abstraction, shown in Figure 1.1, that pose distinct challenges, objectives and solutions. We explain them one by one.

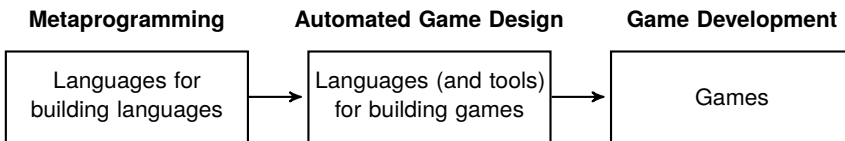


Figure 1.1: A three-tier approach to tackling game development challenges

1.1.1 *Game Development*

Many game development challenges result from game design complexity. Chapter 2.2.1 explains them in some more detail, here we mention a few key challenges.

Games bring about experiences called *gameplay* such as enjoyment, persuasion and learning. Game designers iteratively improve a game’s quality by forming hypotheses, *prototyping* abstract rule systems, and *play testing* how those rules affect players in interactive sessions. Often, the intended and actual gameplay differ. To evolve a game, they have to explore alternatives, constantly requiring changes. Key challenges are:

Iteration time. Game design iterations simply take too much time, which all too often prevents development teams from timely achieving the optimal quality.

Division of labor. Quality improvements fail and time is lost when adjustments best understood by game designers have to be implemented by software engineers.

Software decay. The software quality deteriorates with frequent changes to the source code made to accommodate evolving requirements.

1.1.2 Automated Game Design

This thesis contributes to Automated Game Design (AGD), a research area that aims to automate game design activities. We propose to tackle game development challenges by creating languages and tools that speed up game design processes. Our motivation is detailed in Chapter 2.2.4. Objectives include:

Separation of concerns. Designers express a game’s interactive parts as *source code artifacts*, without the help of engineers who instead maintain the game’s engine.

Live programming. Designers rapidly evolve ‘games and play’ by *making changes* to the source code, and receiving *immediate and continuous feedback* about the effects.

Focused exploration. Intelligent design tools help focus the creative effort and support the design space exploration by suggesting design alternatives (feed forward).

1.1.3 Metaprogramming

In this thesis, we tackle AGD challenges by leveraging and developing generic language technology. In particular, we create *metaprograms*, programs that analyze and transform the source code of other programs. Software language engineers can use metaprogramming techniques to reduce complexity, improve quality, simplify maintenance, and create languages and tools that game designers need to raise their productivity. Objectives include:

Domain-specific languages. We study to what extent Domain-Specific Languages (DSLs) can help automate game design. We introduce DSLs in Chapter 2.2.3.

Generic language technology. We explore how generic language technology can be developed for constructing DSLs for live programming, in particular for AGD.

1.2 RESEARCH QUESTIONS

1.2.1 RQ1: Languages of Games and Play: A Systematic Mapping Study

We wish to learn what informs the design of good games in order to help speed-up the game development process, for creating better games more quickly. In particular, we study to what extent *languages, structured notations, patterns and tools*, can offer designers and developers theoretical foundations, systematic techniques and practical solutions they need to raise their productivity and improve the quality of games and play. We propose the term ‘*languages of games and play*’ for language-centric approaches for tackling challenges and solving problems related to game design and development. Despite the growing number of publications on this topic there is currently no overview describing the state-of-the-art that relates research areas, goals and applications. As a result, efforts and successes are often one-off, lessons learned go overlooked, language reuse remains minimal, and opportunities for collaboration and synergy are lost.

This leads us to the first research question, which is composed of four sub-questions.

Research Question 1 (RQ1)

What is the state-of-the-art in languages of games and play?

RQ1.1 What are the research areas and publication venues where authors have published, and what does a map of the field look like?

RQ1.2 Which languages have been proposed and how can these solutions be characterized in terms of 1) objectives, scope and problems addressed; 2) language design decisions, structure and notable features; 3) applications, show cases or case studies; and 4) implementation, deployment and availability?

RQ1.3 What are similarities and differences between approaches, and common research perspectives sharing similar frames and goals, which languages illustrate them, and what are limitations?

RQ1.4 Which developments and trends can be observed in recent work, and what are the challenges and opportunities for future language research and development?

Method Answering RQ1 in an unbiased, reproducible and systematic way requires a rigorous research methodology for conducting a high quality survey. *Systematic mapping studies* offer a structured framework for retrieving and analyzing publications, and mapping the breadth of related work on a particular topic of interest [KC07].

Results We have answered RQ1 in Chapter 2 by performing a systematic mapping study on ‘languages of games and play’. Its contributions are:

1. A systematic map on languages of games and play that provides an overview of research areas and publication venues, which answers RQ1.1.
2. A set of fourteen complementary research perspectives on languages of games and play synthesized from summaries of over 100 distinct languages we identified in over 1400 publications, which answers RQ1.2 and RQ1.3.
3. An analysis of general trends and success factors, and one unifying perspective on ‘automated game design’, which discusses challenges and opportunities for future research and development, which answers RQ1.4.

1.2.2 RQ2: Domain-Specific Languages for Game Mechanics

Many games have an *internal economy*, an abstract rule-system that determines player choices actions that impact gameplay. Using its mechanisms, players face challenges, enact strategies, and manage trade-offs by accumulating, spending and distributing

in-game resources (e.g., gems, bricks or life essence). Unfortunately, game designers lack a common vocabulary for expressing gameplay, which hampers specification, communication and agreement. We aim to speed up the game development process by improving designer productivity and design quality. The language *Machinations* has introduced a graphical notation for expressing the rules of game economies that is close to a designer's vocabulary [AD12; Dor09]. Crucially, it foregrounds feedback loops associated with patterns of 'emergent' gameplay. However, *Machinations* in its current form is a 'conceptual game design aid' that is not suitable for programming. Because it is not a programming language and lacks a formal semantics, predictions about designs are imprecise, and not about the source code of an eventual game. To apply *Machinations* for software prototyping, designers require a DSL that enables:

- expressing and analyzing game mechanics as source code
- live programming, modifying game mechanics possibly embedded in a game while it is running, and obtaining immediate and continuous (live) feedback about the results, e.g., while play testing
- designing, analyzing and transforming a game's mechanics using a visual tool and an extensible pattern library that expresses common structures

This leads us to the second research question, which has three sub-questions.

Research Question 2 (RQ2)

To what extent can a DSL for game-economic mechanics (game economies) speed up the game development processes, improve the design quality and raise designer productivity?

- RQ2.1 How can meta-programming and model-checking technology be used to formalize a DSL for game mechanics, and analyze and predict qualities of game software?
- RQ2.2 How can the game development process be accelerated and feedback on game design quality be improved by adapting and improving the game mechanics of running game software?
- RQ2.3 How can patterns be used for constructing an interactive game design assistant (a tool) that statically analyzes and generates game mechanics?

Method We apply *design research* [HRS⁺04], and propose solutions that are iteratively tested and improved in practice. Over the years, we have collaborated with industry and academic partners in the following applied research projects:

- Early Quality Assurance in Software Production (EQuA-RAAK-PRO). In this project led by the Fontys University of Applied Sciences, I was one of three lecturer/researchers (PhD candidates). *May 2011 — February 2015*

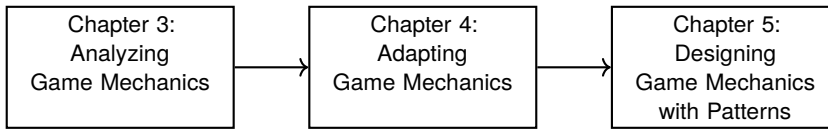


Figure 1.2: The evolution of Machinations: from informal notation for game-economies to visual DSL for live programming

- Automated Game Design (AGD RAAK-MKB). In this project, led by the AUAS, I worked as a researcher. *Feb 2015 — November 2015*
- Live Game Design (LGD RAAK-MKB). I was the main author of the research proposal, principal investigator and project leader. *May 2016 — March 2019*

For each sub-question we have carried out a case study to validate the contributions and evaluated its strengths, weaknesses and threats to validity. Our main industry partners in the work described here have been:

- IC3D Media (Loren Roosendaal)
- Ludomotion (Joris Dormans)
- Firebrush Studios (Paul Brinkkemper)

Results We have demonstrated how Micro-Machinations (MM), a DSL for game-economic mechanics, can speed-up the game development process by improving designer productivity and game design quality. MM has evolved from Machinations [AD12], and addresses its main technical limitations. Figure 1.2 shows the evolution of MM from an informal notation to a DSL for live programming.

In Chapter 3 we answer research question RQ2.1 by demonstrating the feasibility of using RASCAL and PROMELA for meta-programming Micro-Machinations, a DSL for game economies. In particular, we present a programming environment called Micro-Machinations Analysis in Rascal (MM AiR) that enables model-checking small programs against simple invariant properties, visually simulating models, and inspecting event traces. In a case study on SimWar, a conceptual example used to explain poor balancing, we prove that an approximation in MM is poorly balanced.

In Chapter 4 we answer research question RQ2.2 by proposing a novel game design approach for modifying mechanics of running games. In particular, we present an embeddable Micro-Machinations Library (MM Lib) that facilitates runtime modifications of game mechanics, e.g., during play testing. At the cost of development effort during software prototyping and maintenance, our method enables modifying mechanics with the flexibility and ease of paper-prototyping. In a case study on AdapTower, a prototype tower defense game, we demonstrate that we can flexibly change the rules of the game (and the gameplay) while the game is running.

In Chapter 5, we answer research question RQ2.3. We propose a pattern-based approach for analyzing existing mechanics and for modifying them step by step, introducing new challenges, trade-offs and strategies. Applying this approach is a benefit one gets for free when factoring out game mechanics as separate models. We validate the approach on the mechanics of Johnny Jetstream, a 2D fly-by shooter developed at IC3D Media.

Current work The research on MM continues. Our current work entails a new version of the MM library written in C# and a visual live programming environment created in Unity. This new version is more maintainable and understandable, because it leverages a novel technique for run-time state migration, which is based on Chapter 6.

1.2.3 RQ3: *Developing Domain-Specific Languages for Live Programming*

Live programming is a style of development characterized by incremental change and immediate feedback. Instead of long edit-compile cycles, developers modify a running program by changing its source code, receiving immediate feedback as it instantly adapts in response. However, little is known about how the benefits of live programming can be obtained for DSLs. This brings us to the third research question.

Research Question 3 (RQ3)

How can generic language technology be developed for constructing DSLs for live programming?

RQ3.1 How can origin tracking and text differencing be leveraged for textual model differencing that produces model-based deltas?

RQ3.2 How can model-based deltas be leveraged in the design of DSLs that migrate an application's run-time state?

Method As before, we apply Design Research [HRS⁺04]. We propose a novel technique, create a prototype, and study its qualities to assess its feasibility.

Results Chapter 6 answers research question RQ3. We propose two novel techniques and its implementation as generic language technology for developing textual DSLs for live programming. The first, Textual Model Differencing (TMDIFF), produces a model-based difference (or delta) for two textual models. The second, Run-time Model Patching (RMPATCH), applies this delta to the run-time model of a running program, and migrates the domain-specific run-time state. We demonstrate the feasibility of the approach with a language for simultaneously programming and interpreting state machines. Scaling to more complex languages is future work.

Current work We address the following challenges. First, we abstract from concrete domain-specific state migration scenarios for *scaling to complex languages* that have more elaborate migrations. Second, we devise modular high-level language constructs for *programming understandable and maintainable migrations*. Finally, we provide feedback about cause and effect for forming more accurate mental models and making better predictions. The new version of the MM library will demonstrate these principles.

1.2.4 RQ4: Measuring Quality of Grammars for Procedural Level Generation

Grammar-based procedural level generation raises the productivity of level designers for games such as dungeon crawl and platform games. However, the improved productivity comes at the cost of level quality assurance. Authoring, improving and maintaining grammars is difficult because it is hard to predict how each grammar rule impacts the overall level quality, and tool support is lacking. This leads us to the fourth and final research question.

Research Question 4 (RQ4)

How can the quality of grammars for procedural level generation be improved?

Method As before, we apply design research [HRS⁺04]. We have worked with Ludomotion on an iterative improvement for grammar-based level generation technology [DB11], in the context of the Live Game Design project.

We have tackled quality issues by applying common techniques from the research area of software evolution. In particular, we trace model transformations (using *origin tracking*) and analyze level characteristics over time (using *metrics*).

Results In Chapter 7 we have answered research question RQ4 and addressed the lack of tool support by proposing two novel techniques. The first is a novel metric called Metric of Added Detail (MAD) that indicates if a rule adds or removes detail with respect to its phase in the transformation pipeline. The second, Specification Analysis Reporting (SAnR) enables expressing level properties in a simple DSL, and analyzing how qualities evolve in level generation histories. We demonstrate MAD and SAnR using a prototype of a level generator called Ludoscope Lite.

Our preliminary results show that problematic rules tend to break SAnR properties and that MAD intuitively raises flags. MAD and SAnR augment existing approaches, and can ultimately help designers make better levels and level generators. Of course, these techniques are academic prototypes. A more extensive validation and empirical evaluation is part of future work.

Current work The approach opens a research area for leveraging metaprogramming techniques to address a lack of tool support, in this case for grammar-based level

Table 1.1: Relating questions, chapters, publications (P) and tools (T)

question	chapter	publication	venue	tool
RQ1 (1–4)	2	U1	submitted to CSUR	
RQ2.1	3	P1	SLE 2013	T1
RQ2.2	4	P2	FDG 2014	T2
RQ2.3	5	P3	FDG 2015	T3
RQ3.1	6	P4	ICMT 2015	T4
RQ3.2	6	P5	SOSYM	T4
RQ4	7	P6	PCG 2018	T5

design. The chapter was written as an introductory example, and has the purpose of engaging students as collaborators in future applied research projects to continue this work.

1.3 ORIGIN OF THE CHAPTERS

Here we relate thesis chapters to research contributions, shown in Table 1.1. Each chapter is composed of one or more peer-reviewed publications (P), of which I was the primary author. A reduced version of Chapter 2 is under submission (U) at ACM Computing Surveys (CSUR). In addition, we list languages, libraries and tools (T).

1.3.1 Chapter 2: Languages of Games and Play

- U1 R. van Rozen. “Languages of Games and Play: A Systematic Mapping Study”. Under submission to ACM Computing Surveys. 2019

1.3.2 Chapter 3: Analyzing Game Mechanics

- P1 P. Klint and R. van Rozen. “Micro-Machinations: a DSL for Game Economies”. In: *Software Language Engineering – Proceedings of the 6th International Conference on Software Language engineering, SLE 2013, Indianapolis, IN, USA, October 26–28, 2013*. Ed. by M. Erwig, R. F. Paige, and E. Van Wyk. Vol. 8225. LNCS. Springer, 2013, pp. 36–55. ISBN: 978-3-319-02654-1. DOI: 10.1007/978-3-319-02654-1_3

1.3.3 Chapter 4: Adapting Game Mechanics

- P2 R. van Rozen and J. Dormans. “Adapting Game Mechanics with Micro-Machinations”. In: *Proceedings of the 9th International Conference on the Foundations of Digital Games, FDG 2014, Liberty of the Seas, Caribbean, April 3–7, 2014*.

Ed. by M. Mateas, T. Barnes, and I. Bogost. Society for the Advancement of the Science of Digital Games, 2014. ISBN: 978-0-9913982-2-5

1.3.4 Chapter 5: Designing Game Mechanics with Patterns

- P3 R. van Rozen. “A Pattern-Based Game Mechanics Design Assistant”. In: *Proceedings of the 10th International Conference on the Foundations of Digital Games, FDG 2015, Pacific Grove, CA, USA, June 22–25, 2015*. Ed. by J.P. Zagal, E. MacCallum-Stewart, and J. Togelius. Society for the Advancement of the Science of Digital Games, 2015

1.3.5 Chapter 6: Towards Domain-Specific Languages for Live Programming

The first paper (P4) introduces TMDIFF. Chapter 6 is based on an extended journal version (P5) that adds RMPATCH, but which omits an evaluation of TMDIFF on Derric, a DSL for digital forensics.

- P4 R. van Rozen and T. van der Storm. “Origin Tracking + Text Differencing = Textual Model Differencing”. In: *Theory and Practice of Model Transformations – Proceedings of the 8th International Conference on Model Transformation, ICMT 2015, L’Aquila, Italy, July 20–21, 2015*. Ed. by D. Kolovos and M. Wimmer. Vol. 9152. LNCS. Springer, 2015, pp. 18–33. ISBN: 978-3-319-21155-8. DOI: 10.1007/978-3-319-21155-8_2

- P5 R. van Rozen and T. van der Storm. “Toward Live Domain-Specific Languages: From Text Differencing to Adapting Models at Run Time”. In: *Software & Systems Modeling* 18.1 (Feb. 2019). Special Section Paper on STAF2015. Received June 27th 2016. Revised May 26th 2017. Accepted June 20th 2017. First Online August 14th 2017, pp. 195–212. ISSN: 1619-1374. DOI: 10.1007/s10270-017-0608-7

1.3.6 Chapter 7: Measuring Quality of Grammars for Procedural Level Generation

The following paper, of which I am the main author, is based on the master project of Quinten Heijn. His thesis elaborates on this work, and adds a case study [Hei18].

- P6 R. van Rozen and Q. Heijn. “Measuring Quality of Grammars for Procedural Level Generation”. In: *Proceedings of the 13th International Conference on Foundations of Digital Games, FDG 2018, as part of the 9th Workshop on Procedural Content Generation, PCG 2018, Malmö, Sweden, August 7–10, 2018*. Ed. by S. Dahlskog, S. Deterding, J. Font, M. Khandaker, C. M. Olsson, S. Risi, and C. Salge. ACM, 2018, pp. 1–8. ISBN: 978-1-4503-6571-0. DOI: 10.1145/3235765.3235821

1.3.7 Other contributions

The following paper, of which I am the main author, is not included in this thesis. It addresses identifying quality issues of Lua source code.

- P7 P. Klint, L. Roosendaal, and R. van Rozen. “Game Developers Need Lua AiR: Static Analysis of Lua Using Interface Models”. In: *Entertainment Computing – Proceedings of the 11th International Conference on Entertainment Computing, ICEC 2012, as part of the 2nd Workshop on Game Development and Model-Driven Software Development, GD&MDS 2012, Bremen, Germany, September 26–29, 2012*. Ed. by M. Herrlich, R. Malaka, and M. Masuch. Vol. 7522. LNCS. Springer, 2012, pp. 530–535. ISBN: 978-3-642-33542-6. DOI: 10.1007/978-3-642-33542-6_69

1.3.8 Languages, Libraries and Tools

The proposed languages, libraries and tools are released as open source software.

- T1 R. van Rozen. *MM AiR: Micro-Machinations Analysis in Rascal*. <https://github.com/vrozen/MM-AiR>. Eclipse License. 2013
- T2 R. van Rozen. *MM Lib: Micro-Machinations Library*. <https://github.com/vrozen/MM-Lib>. 3-Clause BSD License. 2014
- T3 R. van Rozen. *MeDeA: Mechanics Design Assitant*. <https://github.com/vrozen/MeDeA>. Eclipse License. 2015
- T4 T. van der Storm and R. van Rozen. *TMDiff: Textual Model Differencing and RMPatch: Run-time Model Patching*. <https://github.com/cwi-swat/textual-model-diff>. Eclipse License. 2015
- T5 Q. Heijn and R. van Rozen. *LudoScope Lite*. Includes and demonstrates the Metric of Added Detail (MAD) and Specification Analysis Reporting (SAnR). URL: <https://github.com/visknut/LudoscopeLite>. URL: <https://github.com/vrozen/MAD-Level-Design>. Eclipse License. Aug. 2018

1.4 THESIS STRUCTURE

Here, we give an outline of the chapter structure. Figure 1.3 shows the general structure of chapters (boxes) and possible reading directions (arrows). Chapter 2 answers research question RQ1 and its sub-questions. The next three chapters are on *game mechanics* in particular. Chapters 3, 4 and 5 respectively answer research questions RQ2.1, RQ2.2 and RQ2.3. Together they answer research question RQ2. Chapter 6 is on *live programming* in general. It answers research question RQ3, and its sub-questions. Finally, Chapter 7 answers research question RQ4.

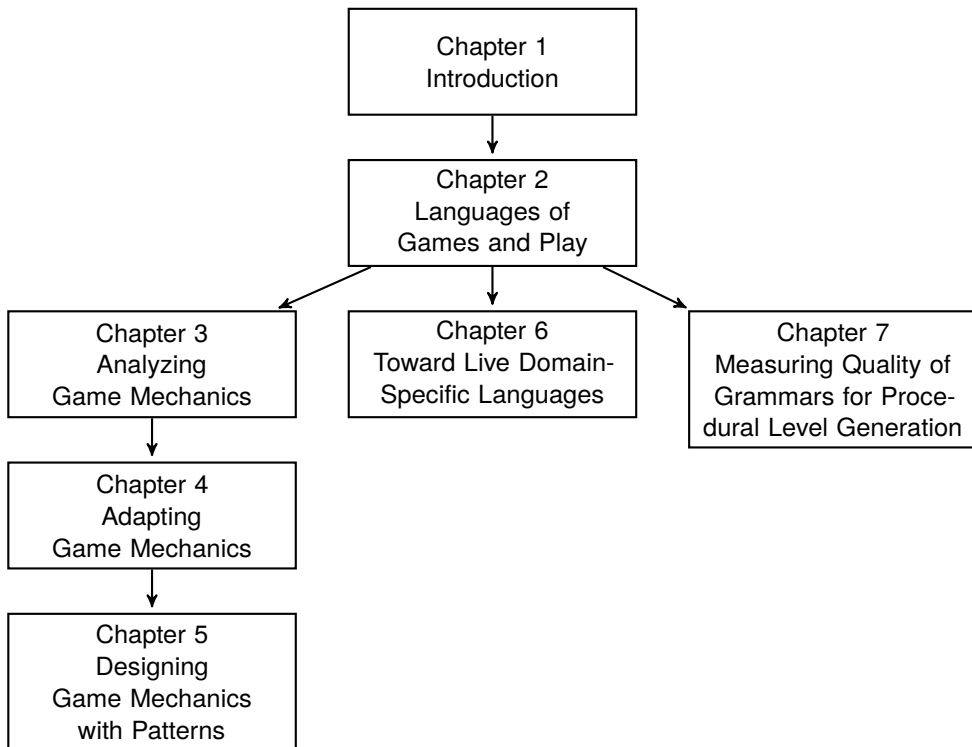


Figure 1.3: Thesis Chapter Structure

1.5 CONCLUSION

Because this thesis has an open-ended structure, we refer to the individual chapters for reflections on future work about their respective topics, in particular Chapters 2 and 6.

Abstract

Digital games are a powerful means for creating enticing, beautiful, educational, and often highly addictive interactive experiences that impact the lives of billions of players worldwide. We explore what informs the design and construction of good games in order to learn how to speed-up game development. In particular, we study to what extent *languages, notations, patterns* and *tools*, can offer experts theoretical foundations, systematic techniques and practical solutions they need to raise their productivity and improve the quality of games and play.

Despite the growing number of publications on this topic there is currently no overview describing the state-of-the-art that relates research areas, goals and applications. As a result, efforts and successes are often one-off, lessons learned go overlooked, language reuse remains minimal, and opportunities for collaboration and synergy are lost. We present a systematic map that identifies relevant publications and gives an overview of research areas and publication venues. In addition, we categorize research perspectives along common objectives, techniques and approaches, illustrated by summaries of selected languages. Finally, we distill challenges and opportunities for future research and development.

2.1 INTRODUCTION

In the past decades, digital games have become a main podium for creative expression enabling new forms of play and interactive experiences that rival the works of great historical writers, painters, artists and composers. The game development industry is a vast and lucrative branch of business that eclipses traditional arts and entertainment sectors, outgrowing even the movie industry [vGOK19]. Games reach audiences around the world, unite players in common activities and give rise to subcultures and trends that impact pass-time, awareness and policies of modern societies.

However, for every outstanding success exist many games with unrealized potential and failures that preceded bankruptcy. Developing high quality games is dreadfully complicated because game design is intrinsically complex. We wish to learn what informs the design of good games in order to help speed-up the game development process for creating better games more quickly. In particular, we study to what extent *languages, structured notations, patterns* and *tools*, can offer designers and developers

This chapter is under submission at ACM Computing Surveys (CSUR)

theoretical foundations, systematic techniques and practical solutions they need to raise their productivity and improve the quality of games and play.

We propose the term '*languages of games and play*' for language-centric approaches for tackling challenges and solving problems related to game design and development. Despite the growing number of researchers and practitioners that propose and apply these languages, there is currently no overview of publications that relates languages, goals and applications. As a result, publications on the topic lack citations of relevant related work. In addition, lessons learned are overlooked and available methods and techniques for language development often remain unused. As a consequence, it remains difficult to compare and study games, designs and research contributions in order to build bodies of knowledge that describe best practices and industry standards.

We aim to map the state-of-the-art of languages of games and play in an understandable way, such that it is accessible to a wide audience. Our goal is to provide a means for 1) informing practitioners and researchers about the breadth of related work; 2) sharing knowledge between research areas and industry for improved results and collaboration; 3) enabling the application of available techniques; and 4) identifying opportunities for future research and development.

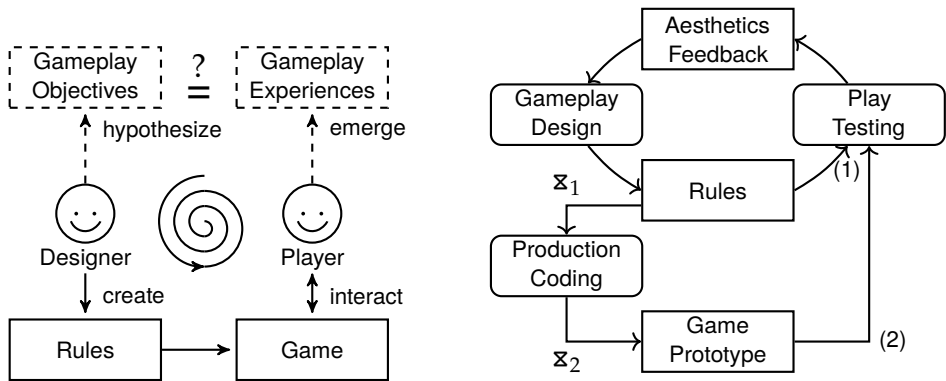
We pose the following research questions:

- Which publication venues include papers on languages of games and play?
- How do the various approaches compare?
- What are open research challenges and opportunities for future work?

For answering these questions we conduct a survey of languages of games and play called a *systematic mapping study*. Mapping studies provide a wide overview of a research area by identifying, categorizing and summarizing all existing research evidence that supports broad hypotheses and research questions [KC07]. In contrast, systematic literature reviews usually have a more narrow focus, and instead perform in-depth analyses to answer particular research questions. Both enjoy the benefits of a well-defined methodology for (re)producing high quality results and reducing bias.

We identify and analyze relevant publications on languages of games and play. First, we motivate the need for this study by describing its scope in Section 2.2. Next, we describe the methodology with research questions, sources, queries and inclusion criteria, and a review protocol in Sections 2.3 and 2.4. We contribute the following:

1. A systematic map on languages of games and play that provides an overview of research areas and publication venues, presented in Section 2.5.
2. A set of fourteen complementary research perspectives on languages of games and play synthesized from summaries of over 100 distinct languages we identified in over 1400 publications, presented in Section 2.6.
3. An analysis of general trends and success factors, and one unifying specific perspective on '*automated game design*', which discusses challenges and opportunities for future research and development, presented in Section 2.7.



(a) Game designers form hypotheses about how a game’s rules realize gameplay goals, but usually repeatedly find intended and actual experiences differ

(b) Developing a high quality game entails iteratively designing, playtesting and improving its rules as a paper- (1) or a software- prototype (2)

Figure 2.1: Game development aims for games with high quality player experiences

We describe related work in Section 2.8, discuss threats to validity in Section 2.9, and conclude in Section 2.10. Our map provides a good starting point for anyone who wishes to learn more about the topic.

2.2 RESEARCH VISION

A mapping study on games and play can be approached from different research perspectives, each with different goals and needs. We introduce challenges of digital game design in Section 2.2.1, and formulate two general hypotheses that drive this study in Section 2.2.2. Our specific motivation is to automate game design and investigate how domain-specific language technology, introduced in Section 2.2.3, can offer solutions. We clarify our position and motivate this study in Section 2.2.4.

2.2.1 Games and Play

Games and play are inextricably intertwined concepts. Games bring about experiences such as enjoyment, persuasion and learning. There exist many different view points, explanations and definitions. We share a well-known definition of Juul, who studies games, examines similarities between them and proposes:

“A game is (1) a rule-based formal system with (2) a variable and quantifiable outcome, where (3) different outcomes are assigned different values [good/bad], (4) the player exerts effort in order to influence the outcome [challenge], (5) the player feels attached to the outcome, and (6) [real-world] consequences of the activity are negotiable.” [Juul11].

Game design [FSHo8; Sch14b], the discipline and process of iteratively designing and improving games, is an instance of a so called *wicked problem*, a problem that is “difficult to solve in general due to incomplete, contradicting and evolving requirements” [Coy05; EO12; MS05a]. We highlight challenges that illustrate its inherent complexity.

Improving a game’s qualities depends on gradually improving insight, as illustrated by Figure 2.1(a). Game designers use *paper prototyping* to explore and understand the problem at hand, abstracting away a game’s details until what remains is essential. They form hypotheses about play, experiment with rules and objectives to evolve a game’s design and learn what the solution can become. Designers create interaction mechanisms (a.k.a. game mechanics, or rules) offering playful affordances [SZ03].

Players interact with games via these mechanisms during a game’s execution. Playful acts result in dynamic interaction sequences. Ideally, these also represent aesthetically pleasing experiences called *gameplay*, e.g., fellowship, challenge, fantasy, narrative, discovery or self-expression [HLZ04]. However, opinions on a game’s quality differ from person to person, e.g., with age, gender and beliefs.

For game designers *playtesting* is essential for verifying assumptions and learning if a game meets its objectives. More often than not, designers discover that the realized and intended gameplay differ. Unfortunately, even well prototyped games may fail to meet expectations as fully developed software. In general, it is hard to predict the outcome of modifying a game’s parts, e.g., how changing the rules affects the dynamics and aesthetics of play. As a result, steering towards new goals is difficult.

Improving a game is never truly done. The maximum number of game design iterations determines the achievable quality. Efforts on balancing, fine-tuning and polishing are limited only by time and money. Resource-wise, AAA studios have a competitive advantage over indie game developers. However, developing novel high quality games in a time-to-market manner is universally hard because game design iterations take simply too much time. Figure 2.1(b) shows an abstract game development process that illustrates the root causes of delay (shown as \mathbf{x}).

Game designers and software engineers usually live on opposite sides of the fence [KvR13]. Both lose time when adjustments best understood by designers have to be implemented by software engineers (\mathbf{x}_1). To evolve a game, designers have to explore alternative gameplay scenarios, constantly requiring changes.

As time progresses, more and more choices become fixed, and frequent changes to the source code become more difficult, time-consuming and error-prone (\mathbf{x}_2). The evolution of digital games, like other software, suffers from a well-known phenomenon called *software decay* [Men08]. The software quality deteriorates with frequent changes to the source code made to accommodate evolving requirements.

As a consequence, game designers have precious few chances to experiment with design alternatives. This seriously compromises their ability to design, prototype

and playtest. Unfortunately, the complexity of game design all too often prevents development teams from timely achieving the optimal quality.

These challenges urgently require solutions. Our brief discussion indicates that rules, objectives and gameplay assumptions are artifacts that require appropriate notations for constructing high quality digital games. However, game designers lack a common vocabulary for expressing gameplay. Next, we address this need.

2.2.2 *Languages of Games and Play*

Languages of games and play are language-centric approaches for tackling challenges and solving problems related to game design and development. We propose studying existing languages and creating new ones. Two central hypotheses drive this study. We formulate a general and a specific hypothesis:

1. *Languages, structured notations, patterns and tools* can offer designers and developers theoretical foundations, systematic techniques and practical solutions they need to raise their productivity and improve the quality of games and play.
2. “Software” languages (and specifically domain-specific languages) can help automate and speed-up game design processes.

Languages of games and play exist in many shapes and forms. The next section describes one specific technical point of view that represents the departure point of this study, which also details and motivates the second more specific hypothesis.

2.2.3 *Domain-Specific Languages*

We aim to deliver solutions that automate game design and speed-up game development with so-called Domain-Specific Languages (DSLs), an approach originating in the field of Software Engineering. Van Deursen et al. define the term as follows:

“A Domain-Specific Language is a programming language or executable specification language that offers, through appropriate abstractions and notations, expressive power focussed on, and usually restricted to, a particular problem domain.” [vDKV00].

DSLs have several compelling benefits. They have been successfully created and applied to boost the productivity of domain-experts and raise the quality of software solutions. For instance, in areas like carving data in digital forensics [vd-BvdS11], engineering financial products [vDeu97], and controlling lithography machines [TMvdB⁺13], to name a few. DSLs divide work and separate concerns by offering domain-experts ways to independently evolve and maintain a system’s parts. Typically, DSLs raise the abstraction level and incorporate domain-specific terminology that is more recognizable to its users. Powerful language work benches enable analyses, optimizations, visualizations [EVV⁺13], and foreground important trade-offs, e.g., between speed and accuracy in file carving.

Naturally, there are also costs. DSLs are no silver bullet for reducing complexity. Time and effort go into developing the right language with features that are both necessary and sufficient for its users. In addition, a DSL may have steep learning curve and users require training [vDKVoo]. While DSLs help users maintain products, DSLs themselves also demand maintenance and must evolve to accommodate new requirements, usage scenarios, restrictions and laws, such as new legislation on financial transparency or privacy.

2.2.4 *Need for a Mapping Study*

There are many compelling reasons to perform a mapping study on languages of games and play. This study can be approached from different research perspectives with distinct research needs and goals. Here we describe our position and motivation.

We aim to empower game designers with DSLs that automate and speed-up the game design process. We wish to learn how to facilitate the design space exploration and reduce design iteration times. We envision a set of complementary visual languages, techniques and tools that help designers boost their productivity and raise the quality of games and play. Challenges include providing abstractions and affordances for:

1. expressing a game's parts as source code artifacts, especially interaction-bound game elements, and modifying these at any given moment
2. evolving 'games and play' by steering changes in the source code towards new gameplay goals prototyping, play testing, balancing, fine-tuning or polishing
3. obtaining immediate and continuous (live) feedback on a game's quality by continuously play testing the effect of changes on quantified gameplay hypotheses
4. obtaining feed forward suggestions that focus creative efforts and assist in exploring alternative design decisions in a targeted way
5. forming better mental models for learning to better predict the outcome on play

To know where to start automating game design, we need an extensive analysis on existing approaches. However, these efforts are currently not mapped, and opportunities and limitations are not yet well understood. As a result it is unclear which game facets are amenable to DSL development, which features can express game designs, and what the limits of formalism are. There is no telling if DSLs can deliver, and how the tradeoff between costs and benefits applies to game development.

We perform this mapping study on languages of games and play to obtain evidence to support our hypotheses in general, and suit our own specific research needs by scouting for opportunities for developing DSLs in particular.

2.3 METHODOLOGY

A systematic mapping study requires a precise description of its scope, research questions, search queries and databases for accurate and reproducible information extraction, categorization and comparison [KC07]. We apply the following methodology.

2.3.1 *Scope*

Games have been studied from different perspectives. Language-oriented approaches have been proposed by authors who published in separate fields of research using distinct vocabularies. As a result, language-centric solutions, intended for diverse domain experts and novices solve differently scoped problems related to a game's design, development, and applications. We survey the full breadth of related work.

2.3.2 *Research questions*

The research questions addressed by this study on languages of games and play are:

- RQ1 What are the research areas and publication venues where authors have published, and what does a map of the field look like?
- RQ2 Which languages have been proposed and how can these solutions be characterized in terms of 1) objectives, scope and problems addressed; 2) language design decisions, structure and notable features; 3) applications, show cases or case studies; and 4) implementation, deployment and availability?
- RQ3 What are similarities and differences between approaches, and common research perspectives sharing similar frames and goals, which languages illustrate them, and what are limitations?
- RQ4 Which developments and trends can be observed in recent work, and what are the challenges and opportunities for future language research and development?

2.3.3 *Sources*

We use the meta-repository Google Scholar (GS) to obtain primary sources because it maps repositories in which we expect to find relevant publications. GS includes traditional sources of publications such as the Association of Computing Machinery (ACM), Institute of Electrical and Electronics Engineers (IEEE), Springer and Elsevier. In addition, GS includes less-traditional sources such as games conferences that operate independently, influential books, blog posts, and dissertations. Limiting the search to fewer sources would likely make the study more easily reproducible but also reduce its relevance. A wide search over many sources is necessary for answering our research questions, and GS fits this criterion. Note that we exclusively focus on written sources, which excludes games and commercial development products.

Table 2.1: Google Scholar search parameters

URL segment	Description
q=[query]	Queries [query], where AND is a space and NOT is a minus sign
hl=en	Displays Google tips and messages in English (en)
lr=lang_en	Reports results only in English (lang_en)
as_sdt=1,5	Excludes patents (1,5 = exclude and 0,5 = include)
as_vis=1	Excludes citations, (1 = hide and 0 = show)

2.3.4 Queries

Starting with a limited view of the field, we begin with a query to find domain-specific languages for game design and game development. We call this our *narrow query*.

```
"domain_specific_language" AND ("game_development" OR "game_design")
```

GS returns approximately 400 results, mainly in the field of software engineering and although many publications seem relevant, few articles focus on game design. Clearly, the narrow query is biased towards one specific research area and is too restricted for answering our research questions.

Part of a mapping study is identifying the distinct vocabularies experts in separate research areas use for describing similar approaches, and a more general term is “language”. We widen the scope accordingly but unfortunately we now find many results on subjects that are off-topic. We therefore attempt to filter out irrelevant publications by formulating a *wide query*.

```
language AND ("game_development" OR "game_design")
AND NOT ("sign_language" OR "second_language" OR "language_acquisition" OR "body_
language" OR "game_based_learning" OR "beer_game" OR gamification OR gamify)
```

GS reports approximately 17.5 thousand results, more than is feasible for us to analyze. We now realize that given the wicked nature of game design and game development, no single query exists that captures all relevant works. We therefore propose a compromise that combines the results of the narrow query with the first 1000 results of the wide query¹.

We restrict the language to English. We exclude patents and citations, results for which GS typically has not seen the full source. Part of the search parameters are stored in a cookie, accessible via GS’s web interface, e.g., bibliographical data. The cookie also stores a unique id, which GS uses to track the query and block requests with a captcha if its algorithms indicate restrictions are violated. We construct the following GS search URL for the queries. Table 2.1 explains the URL segments.

¹GS limits the number of results to one thousand, but one can obtain more when filtering by year.

[https://scholar.google.com/scholar?q=domain-specific_language" \("game_design" OR "game_development"\)&hl=en&lr=lang_en&as_sdt=1,5&as_vis=1](https://scholar.google.com/scholar?q=domain-specific_language)

[https://scholar.google.com/scholar?q=language \("game_development" OR "game_design"\) -"sign_language" -"second_language" -"language_acquisition" -"body_language" -"game_based_learning" -"beer_game" -gamification -gamify&hl=en&lr=lang_en&as_sdt=1,5&as_vis=1](https://scholar.google.com/scholar?q=language ()

2.3.5 *Inclusion and exclusion criteria*

We select publications according to the following criteria. The inclusion criterion is: The publication describes a structured language-oriented approach for solving problems related to the design or the development of digital games. For instance, we include programming languages, modeling languages, DSLs, pattern languages, ontologies and structured vocabularies. Digital games (or digital representations) include computer games, videogames, and applied games (a.k.a. serious games), etc.

The exclusion criterion is: Language features with a fixed structure and notation are not described in the paper, or the language does not relate directly to games, and as such does not inform the game design process. Therefore, we exclude the mathematics subject of game theory and the general theme of high performance computing. Networking and audio are not excluded a-priori.

2.4 REVIEW PROTOCOL

We identify and analyze relevant publications and categorize them according to the review protocol described in the following section. Each section of the protocol addresses a research question. Section 2.4.1 addresses RQ1, Section 2.4.2 addresses RQ2. The remaining questions RQ3 and RQ4 are outside the scope of this protocol, and are addressed respectively in Section 2.6 and Section 2.7.

2.4.1 *Research areas and publication venues*

For each included publication we record the available bibliographical information in BibTeX, e.g., about its authors, title, editors, year, publication venue, acronym, publisher, journal, volume, number, ISBN, ISSN and DOI. When records are incomplete or missing, as is often the case, we insert the information by hand.

In addition, we add a mapping study identifier that denotes its rank in the query results, a number or not found (–), and appears in the narrow (n) or wide (w) query results. For instance, 12n indicates the publication ranked 12 in the GS results of the narrow query, and –w indicates a publication that conforms to the wide query, but is not ranked in the top one thousand results. In some cases, we include publications

Table 2.2: Categories of publications

Category	Description
Paper or article	Peer-reviewed papers appearing in the proceedings of a symposium, workshop or conference, or a journal article
Thesis	Describes the content of a Bachelor's, Master's or PhD thesis. May contain chapters based on previously published peer-reviewed work
Textbook	Explains a subject such that it can be easily studied chapter by chapter, typically written by one or more established experts
Non-fiction	Describes topic of interest, not primarily intended to be studied
Technical report	Reports on what are usually technical challenges and solutions
Manual	Explains how to perform steps, use solutions or apply concepts
Blog post	Shares an opinion, problem or technical approach on a web page
Presentation	Introduces or explains challenges on a topic that are noteworthy or inspirational during a keynote presentation or invited talk

Table 2.3: Categories of research (adapted from Wieringa et al. [WMM⁺06])

Category	Description
Evaluation research	Investigates a problem or implementation of a technique <i>in practice</i> for gaining <i>empirical</i> knowledge about causal relationships between phenomena or logical relationships among propositions
Proposal of solution	Proposes a novel solution technique and argues for its relevance without a thorough validation.
Validation research	Investigates properties of a proposed solution that has not yet been used in practice
Philosophical paper	Sketches a new way of looking at a problem, a new conceptual framework, etc.
Opinion paper	Provides an author's opinion about what is wrong or good about a topic of interest
Experience report	Explains steps taken and lessons learned from experiences gained during a project
Tutorial	Explains and demonstrates how something works, usually by means of illustrative examples

for clarity that conform to neither query, stating which keyword is missing, e.g., `gd` indicates the keywords “game design” and “game development” are both missing.

Each publication is of a certain type, as shown in Table 2.2. In addition, we analyze research categories shown in Table 2.3, like Petersen et al. [PFM⁺08]. This table extends categories proposed by Wieringa et al. for categorizing peer reviewed research in requirements engineering with the last two [WMM⁺06]. These are general categories that indicate how reliable and mature a source is, without going into detail.

Table 2.4: Languages facets to summarize and analyze

Facet	Element	Description
Brief description	Problem	Problem statement, game topic
	Objectives	Goals the authors formulate, challenges addressed
	Solution	Solutions proposed, claims on language application and scope, game genre
	Category	Solution category and application area (Table 2.5)
Design	Pattern	Language design pattern (Table 2.6)
	Features	Language features (elements shown in Table 2.7)
	Examples	Snippets of text, code, diagrams or models
Implementation		How is the language implemented, e.g., interpreter, compiler (these details are usually not described)
Validation	Products	Games, prototypes and show cases that are constructed using the language
Availability	Web site	URL of a web site providing information on the language, notation or toolset
	Distribution	URL of a binary distribution or source code repository
	Source license	License under which the source code is available

We construct a visual map of the field by leveraging citation data that relates publications, and categorize publication venues to research areas. First, we extract citation information from the GS research results. Next, we generate a citation graph whose nodes are publications and edges are citations between them. Finally, we visualize this graph using Gephi, an interactive graph visualization framework². Gephi's force map algorithm draws together publications with citations between them, forming clusters that roughly correspond to research areas.

We count the number of publications in different journals, conferences, workshops and symposia. We identify research areas by grouping venues according to disciplines and shared topics. We briefly describe each area, and zoom in on the related section of the map for illustration. We summarize venues with two or more publications.

2.4.2 Language analysis and summary

We wish to learn how languages compare, what they have in common, what separates them, and what makes them unique. For each included publication we extract the name of the language, or a description in case no name is provided. We summarize each language concisely by analyzing related publications in a style similar to an annotated bibliography. Table 2.4 highlight the facets we analyze.

²<https://gephi.org> (visited June 6th 2019)

Table 2.5: Language objectives – solution scope, category and application area

Dimension	Category	Description of intent
Scope	Application-specific	Solution is specific for a game or application
	Genre-specific	Solution that is reusable for a specific game genre
	Generic	Generic solution or separated concern
Solution	Framework	Analysis or mental framework for studying, understanding, comparing, categorizing games that does not directly support game development, e.g., ontologies, design patterns, or simulations
	Tool	Authoring tool that facilitates creating a game's parts as models or programs for design or development , e.g., visual environments, programming languages or DSLs
	Engine	Game engine, reusable building block or software library that integrates models fully into game software
Area	Research	Research vehicle primary intended for performing research in a specific area
	Educative	Platform primarily intended for teaching a subject to a group of people or example meant to illustrate, educate or inform
	Practice	Solution primarily intended for practitioners, supporting game design or game development

Claims regarding the scope and applications typically refer to game genres, such as First Person Shooter (FPS), Role Playing Game (RPG) or 2D Platform Game. Although game genre qualifiers are course grained, and not suitable for comparing games in detail [ASS03], they do offer authors ways to indicate the topic of the solution and sketch contours of its scope. In addition, we categorize languages objectives using the categories shown in Table 2.5. Non-exclusive objectives position languages as:

- Communication means for sharing knowledge between experts,
- Illustration means for explaining or clarifying problems or solution by example,
- Maintenance tool for maintaining and modifying a game's parts over time,
- Productivity raiser for increasing the productivity of its users,
- Quality raiser for improving a game's quality,
- Reuse promotor for making parts of a game's code or design reusable.

We highlight language design decisions and notable features, including mentions of language reuse and formal semantics. When possible, we use the language design patterns for DSLs proposed by Mernik et al. shown in Table 2.6 in our description [MHS05]. We analyze language features related to notation, elements and

Table 2.6: Language design patterns (adapted from Mernik et al. [MHS05])

Dimension	Category	Description
Reuse	Piggyback	Partially uses an existing language, a form of exploitation
	Specialization	Restricts an existing language, a form of exploitation
	Extension	Extends an existing language, a form of exploitation
	Invention	Designs a language from scratch without language reuse
Description	Formal	Formally describes a language using an existing semantics definition method such as attribute grammars, rewrite rules, or abstract state machines
	Informal	Informally explains a language without formal methods

Table 2.7: Language features (these features are not mutually exclusive)

Dimension	Feature	Description
Notation	Textual	The language has a textual notation
	Visual	The language has a visual notation
Elements	Scopes	Scopes and bounds may be used to separate elements and limit their valid context
	Conditionality	Conditionality features enable or disable other language elements or events
	Recurrence	Recurrence features are elements that can happen again, e.g., in iterations
	Modularity	Modularity features enable composition and/or reuse of language elements
	Domain-specific	Domain-specific features may be especially created for a special purpose are unique to the language
User Interface	Feedback	Provides a feedback feature enabling understanding
	Mixed-initiative	Provides feedback & feed-forward, alternating between user input and computer generated alternatives
	Live	Provides immediate and continuous feedback, e.g., a live programming environment

user interface described in Table 2.7. We record how a language is implemented, e.g. as an interpreter or a compiler, and what the host language or formalism is.

Furthermore, we assess applications and availability to form an idea about its status, deployment and maturity. We list notable applications, show cases and case studies that have been used to validate or evaluate the language in practice. Finally, we report which languages are actually available, and if applicable, we provide links to manuals, teaching materials, source repositories and license agreements.

This concludes the protocol. We present the results in the following section.

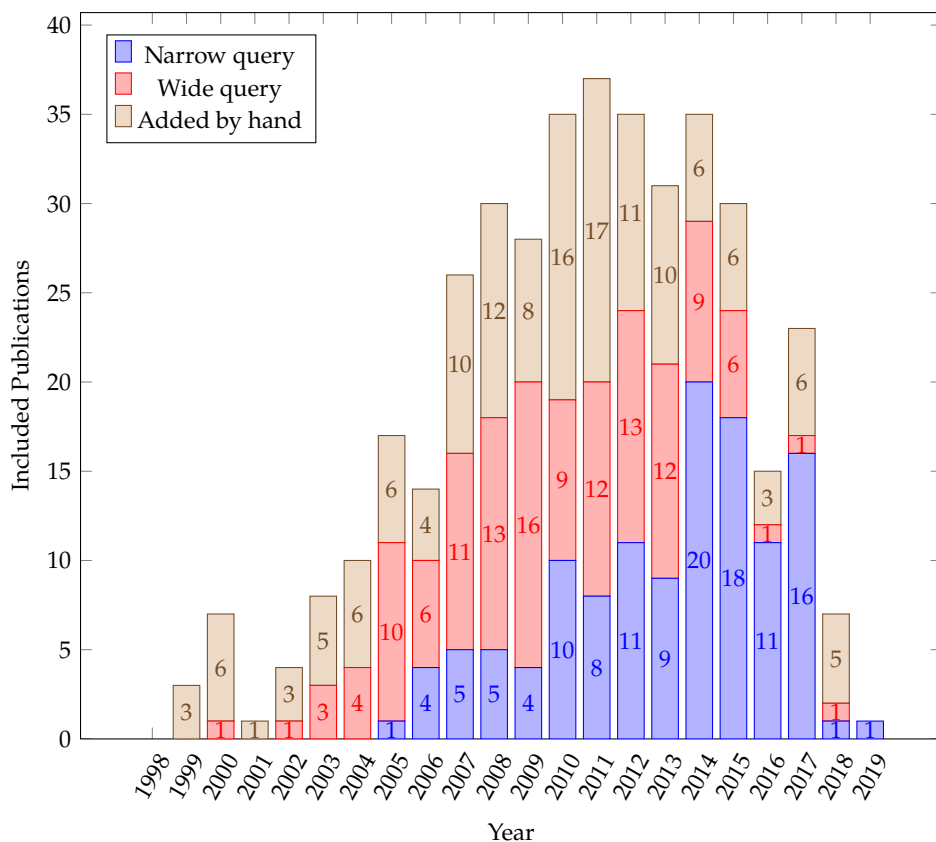


Figure 2.2: Amount of publications included in the study per year

2.5 RESEARCH AREAS

Here we present a systematic map of languages of games and play. We performed a search on GS with our narrow and wide queries between the 2nd and 8th of March 2018, and obtained citation data until March 21st. Figure 2.2 shows a year-by-year count of papers included in this study from both queries and publications we added by hand. Figure 2.3 shows the citation graph of publications in the search results. Each dot with text represents a publication shown with first author name and year. Publications are included (green) or excluded (red) by applying the criteria from the search protocol. Publications not connected to the graph are omitted.

The languages of games and play we have identified originate from the fields of software engineering, artificial intelligence, humanities, social sciences, education,

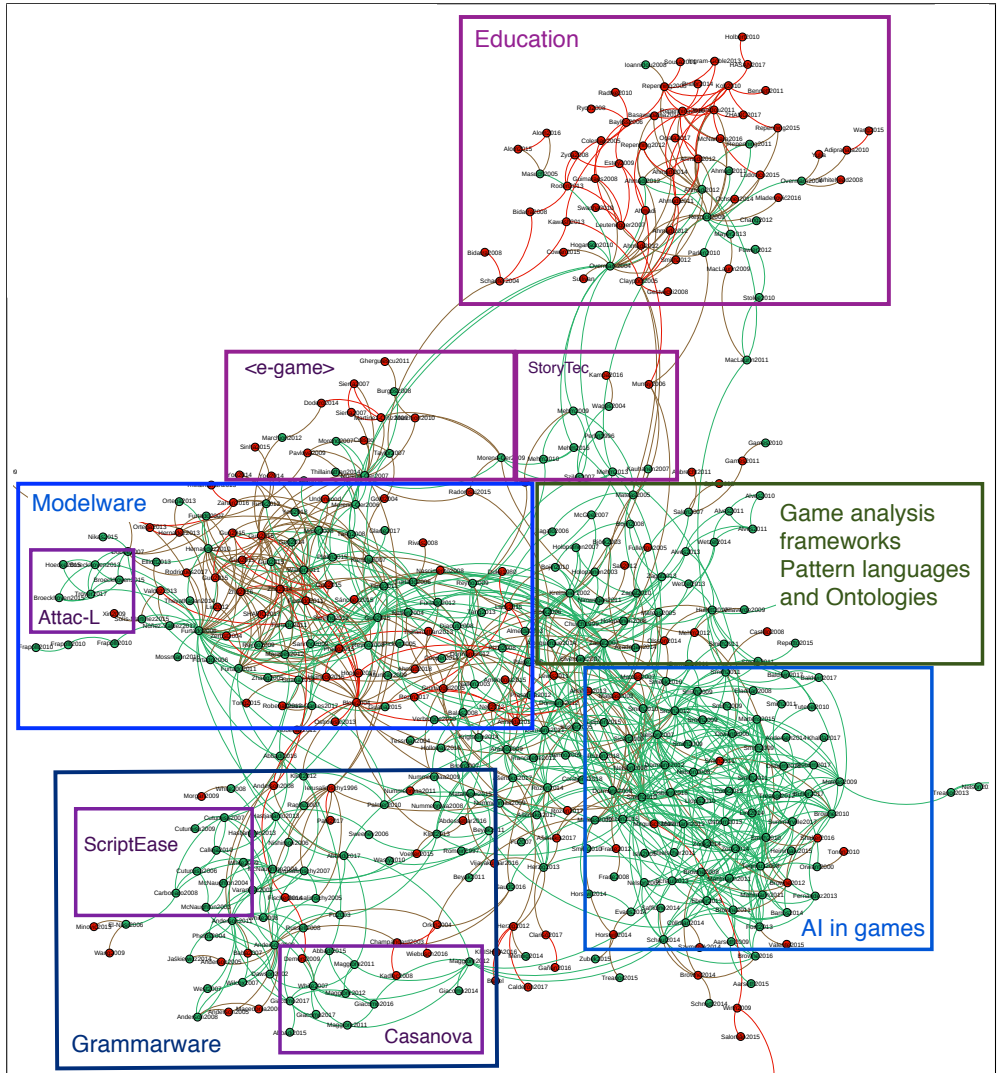


Figure 2.3: Systematic map consisting of publications and citations between them

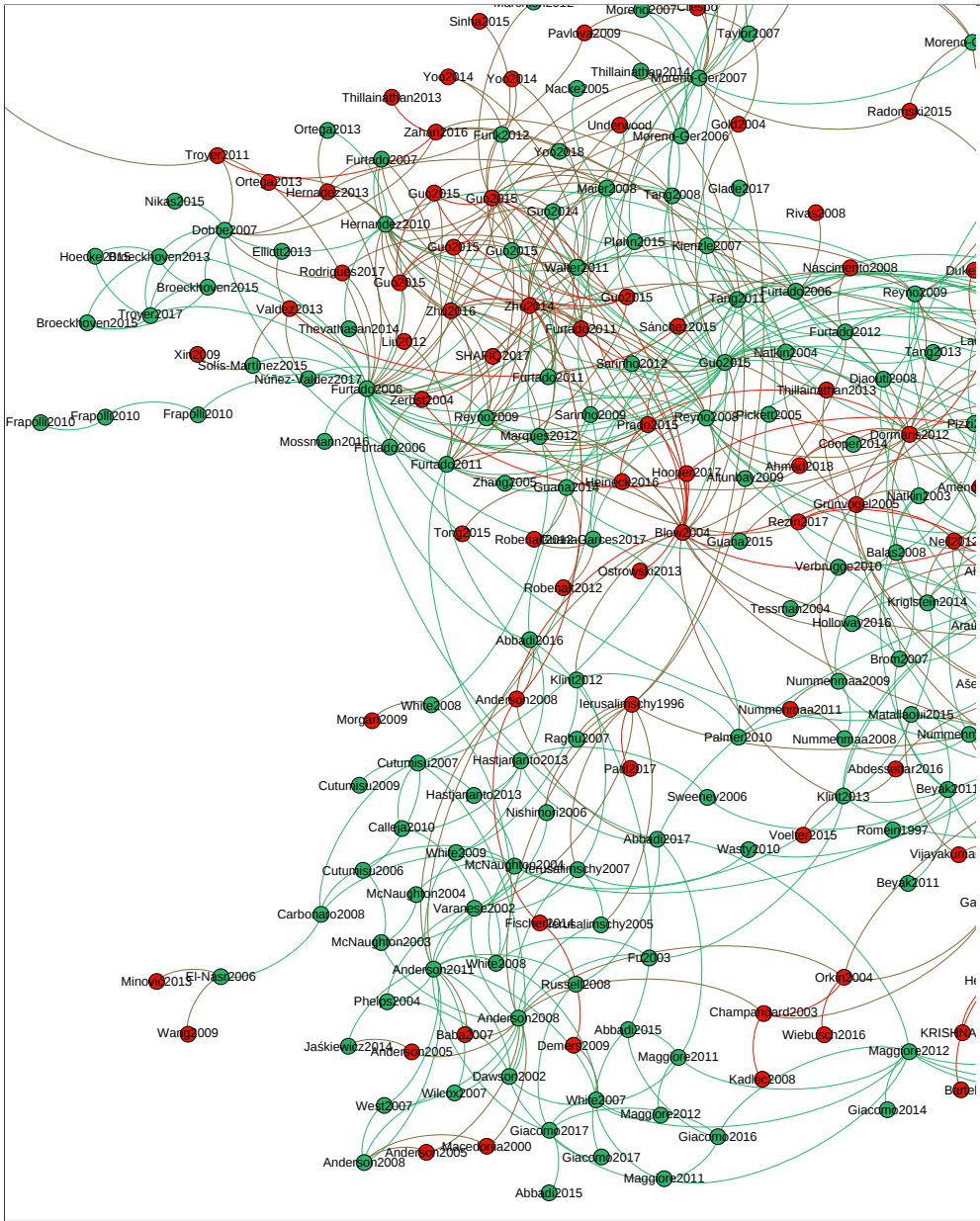


Figure 2.4: Mapping of citations shown as a graph (left part) – Software Engineering, Modelware (mainly the top half) and Grammarware (mainly the bottom half)

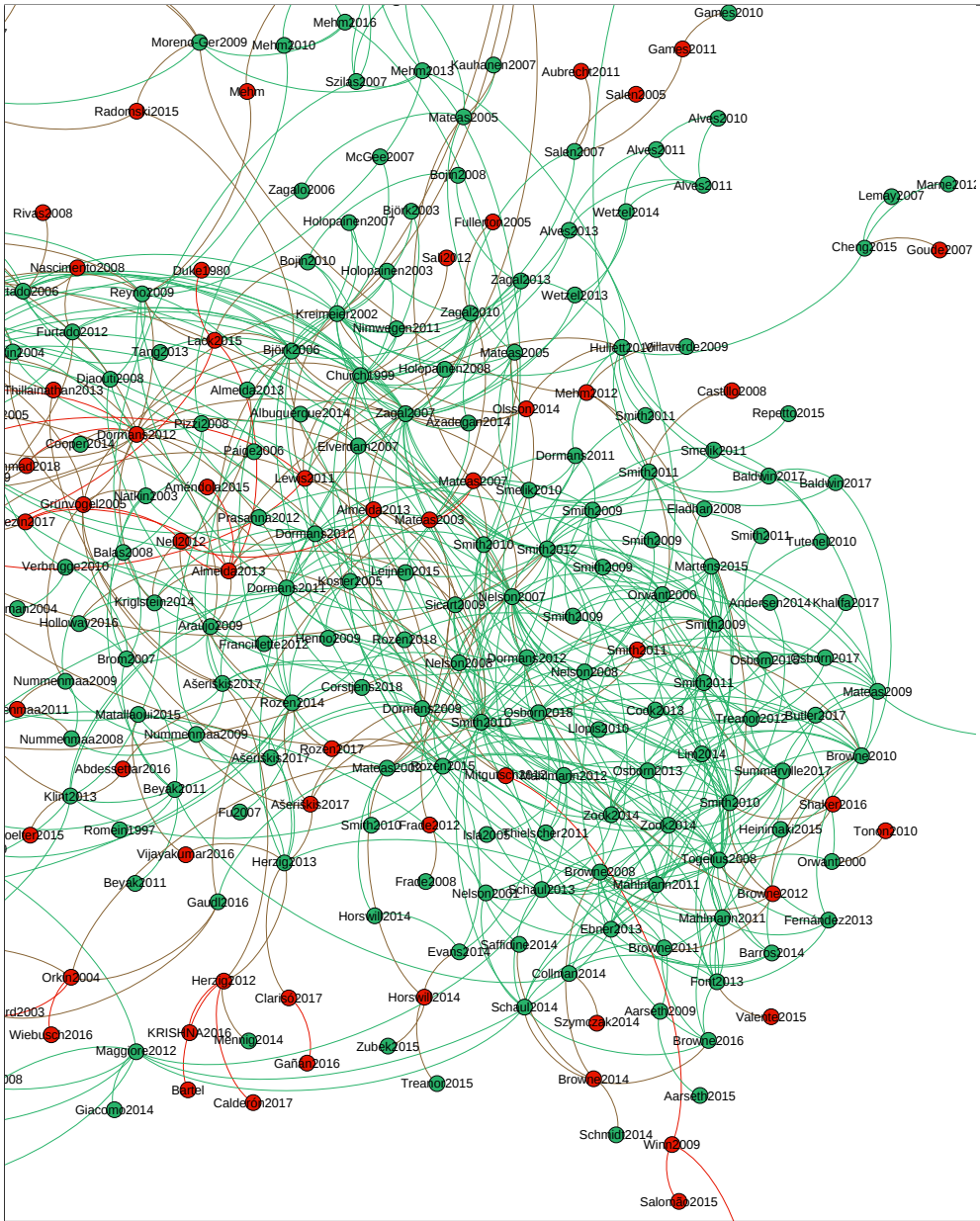


Figure 2.5: Mapping of citations shown as a graph (right part) – Analysis frameworks, pattern languages, ontologies (mainly the top half) and AI in Games (mainly the bottom half)

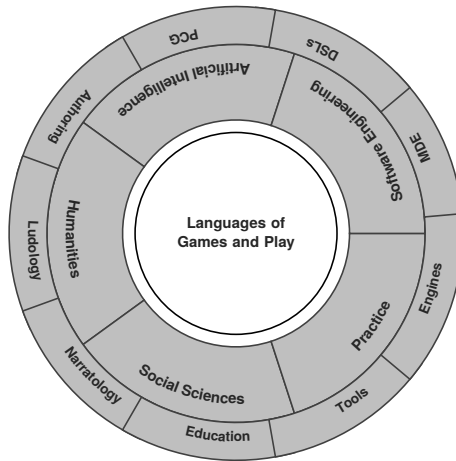


Figure 2.6: Language-centric approaches crosscut areas, disciplines and topics

and game studies, with some cross-disciplinary overlap and diffuse areas. Figure 2.6 illustrates the diversity of publication areas and topics we have selected. The reader is invited to spin the outer wheel of research topics around the publication areas. We describe research areas one by one. We briefly introduce each area, give an overview of venues and link related research perspectives, which are detailed in Section 2.6. For conciseness, we only describe venues when the number of identified publications is at least two, as specified by the search protocol.

2.5.1 *Software Engineering and Programming Languages*

Software Engineering (SE) researchers study the game domain by developing and applying structured methods, languages, techniques and tools for engineering better game software. Lämmel covers several subjects of Software Language Engineering in his textbook on *Software Languages: Syntax, Semantics, and Metaprogramming* [Läm18]. *Compilers: Principles Techniques and Tools* (a.k.a. the “*dragon book*”) by Aho et al., first published in 1986, is still regarded as a classic foundational textbook [ASU86].

We identify contributions from Programming Language (PL) research in particular, as shown in Table 2.8. Figure 2.4 zooms in on related publications. The ACM Special Interest Group on Programming Languages (SIGPLAN) “*explores programming language concepts and tools, focusing on design, implementation, practice, and theory*”.

The main source of publications is the International Conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH), a

Table 2.8: Publication venues in the field of Software Engineering and Programming Languages

Venue	Acronym	Years	Ct.
International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (conference umbrella)	SPLASH	1986–	
Conference on Object-Oriented Programming Systems, Languages and Applications	OOPSLA	1986–	1
Symposium on New Ideas in Programming and Reflections on Software	Onward!	2002–	2
International Conference on Generative Programming and Component Engineering	GPCE	2002–	1
International Conference on Software Language Engineering	SLE	2008–	3
Workshop on Domain-Specific Modeling	DSM	2001–	4
International Conference on Software Engineering	ICSE	1975–	2
Workshop on Games and Software Engineering	GAS	2011– 2016	2
International Conference on Automated Software Engineering	ASE	1990–	2
Symposium on Principles of Programming Languages	POPL	1973–	2
Science of Computer Programming		2000–	2
ACM Sigplan Notices		1966–	2
Communications of the ACM	CACM	1958–	3
ACM Queue	Queue	2003–	2

large conference ‘umbrella’ of colocated events, which includes: 1) Workshop on Domain-Specific Modeling (DSM) [FS06a; FSR11; HO10; MBB⁺12]; 2) International Conference on Software Language Engineering (SLE) [dGAC⁺17b; KvR13; NK12b]; 3) Conference Object-Oriented Programming Systems, Languages and Applications³ (OOPSLA) [SBI⁺08a]; 4) Symposium on New Ideas in Programming and Reflections on Software (Onward!) [MK13; Pal10]; and 5) International Conference on Generative Programming and Component Engineering (GPCE) [AGM⁺06]. SLE research is traditionally split between modelware and grammarware, which respectively revolve around meta-models and grammars [PKP13].

In addition, the search revealed two publications at the International Conference on Software Engineering (ICSE) [BC04; COS⁺06] and two more at the colocated workshop on Games and Software Engineering (GAS), which was organized five times [GSN15; HJL13].

Other conferences include the International Conference on Automated Software Engineering (ASE) [MCS⁺04a; MCS⁺04b], and the International Conference on Model

³A track of SPLASH since 2010

Table 2.9: Publication venues in the field of Artificial Intelligence and Games

Venue	Acronym	Years	Ct.
AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment	AIIDE	2005–	14
Workshop on Experimental AI in Games	EXAG	2014–	2
IEEE Conference on Computational Intelligence and Games	CIG	2005– 2018	8
IEEE Conference on Games	CoG	2019–	–
International Conference on Computational Creativity	ICCC	2010–	2
EvoStar – The Leading European Event on Bio-Inspired Computation (conference umbrella)	EvoStar	1998–	
International Conference on the Applications of Evolutionary Computation – Games track (EvoGames)	Evo-Applications	2010–	3
IEEE Transactions on Computational Intelligence and AI in Games	TCIAIG	2009– 2017	12
IEEE Transactions on Games	T-G	2017–	–

Driven Engineering Languages and Systems (MoDELS) [KDV07]. In addition, we find two invited talks at the Symposium on Principles of Programming languages (POPL) [Mac11a; Sweo6] intended to inspire PL research.

Several journals stand out. The monthly SIGPLAN Notices includes special issues from associated conferences, including SPLASH, SLE, Onward!, GPCE and POPL [Mac11b; SBI⁺08b]. Communications of the ACM is a journal that covers a wider computer science space [Fla12; RMM⁺09; WKG⁺09] and articles from ACM Queue are included in its practitioners section [Fla11; WSG⁺08]. In addition, we find two publications in special issues of Elsevier’s Science of Computer Programming [COM⁺07; MSM⁺07].

We highlight the following related perspectives:

- Automated Game design, a multi-disciplinary area that includes SE, in Section 2.7.3,
- Applied (or serious) game design, in particular DSLs for expressing subject matter, in Section 2.6.3,
- Script- and Programming Languages for game development, in Section 2.6.12,
- Modeling Languages and model-driven engineering for game development, in Section 2.6.13,
- Metaprogramming, primarily illustrative examples explaining the power of generic language technology, in Section 2.6.14.

2.5.2 Artificial Intelligence and Games

The Artificial Intelligence (AI) community has studied how games can benefit from intelligent, usually algorithmic approaches, yielding efficient algorithms, techniques and tools. In their textbook on AI and Games, Yannakakis and Togelius describe the theory, use and application of algorithms and techniques [YT18].

When languages are created, it is often in the context of intelligent systems (or expert systems), or content generators. Classical AI favored logic programming in Prolog for knowledge engineering, the construction of intelligent systems, which explains why some modern solutions are also based on this paradigm. Notable approaches include logic (Prolog, Answer Set Programming) and Machine Learning. Figure 2.5 zooms in on related publications, mainly clustered together in the bottom half of the graph. Conferences, symposia and workshops include the following.

The Association for the Advancement of Artificial Intelligence (AAAI) “*aims to promote research in, and responsible use of, artificial intelligence*”. This includes the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE) [BBA⁺08; Mar15; MS05b; NMO8a; OZ10; OSM⁺15; OGM13; PCW⁺08; SNM09a; SNM09b; SWM10b; TSB⁺10; VKM⁺08]

and colocated workshops, such as Experimental AI in Games (EXAG) [KT17], which are annually organized in North America.

IEEE Conference on Computational Intelligence and Games (CIG) covers “*advances in and applications of machine learning, search, reasoning, optimization and other CI/AI methods related to various types of games.*” We find contributions related to methods for general game playing such as game description languages and generation of level, strategies and game rules. [BDF⁺17a; BT14; LH14; MTY11a; Sch13; SM10; SNM10; TSo8]. The IEEE Conference on Games (CoG), evolved from CIG and widened the scope to cover among other topics, game technology, game design, and game education.

The prime journal is IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG) [BM10; DB11; ES14; PLW⁺10; Saf14; Sch14a; Sch14c; SM11a; SLH⁺11; SWM⁺11; SMH⁺19; vdLLB14]. The journal recently widened its technological scope and was renamed to Transactions on Games (T-G).

We highlight the following related perspectives:

- Automated Game design, a multi-disciplinary research topic with many contributions from AI and games, in Section 2.7.3,
- Game mechanics, frameworks and systems providing analysis, generation and explanations of a game’s rules, in Section 2.6.4,
- Virtual worlds and game levels, spaces whose structure and composition can be described by languages, tools and generative techniques, in Section 2.6.5,
- Behavior languages, the design of algorithms, tools and engines for non-player character behavior, in Section 2.6.6,

Table 2.10: Articles and presentations from Game Development Practice

Venue	Acronym	Years	Ct.
Game Developers Conference (UBM Technology Group)	GDC	1988–	4
Game Developer Magazine (UBM Technology Group)		1994– 2013	3
Gamasutra (UBM Technology Group)		1997–	3

- Narratives and storytelling, in particular technical language-centric approaches, in Section 2.6.7,
- Game analytics and metrics, in Section 2.6.8.
- General gameplaying and Game Description Languages, formalisms for a diverse test-bed for AI and general game playing, in Section 2.6.11.

2.5.3 *Game Development Practice*

The Game Developers Conference (GDC) owned by the UBM Technology Group is a large annual event and not a publishing venue. However, it hosts presentations by practitioners, some of whom share presentation slides identified by this study [Chao7; HB03; Iso5b; Kos05a]. The GDCVault⁴ contains audio and video recordings of these presentations and all volumes of Game Developer Magazine⁵, which ran until 2013 [Chu99a; Llo10; Wes07]. Gamasutra⁶ is a web site that continues to publish articles, and hosts selected articles from Game Developer Magazine [Chu99b; Kre02; Wil07]. In particular, post-mortems in which developers share experiences about challenges, solutions, decisions (the good and the bad) during a game’s life cycle offer glimpses of game development practice [Gro03]. Given the enormous size of the game industry, the practical accounts we identified are few. These languages likely represent the tip of the iceberg. We reflect on this issue in Section 2.9.

2.5.4 *Social Sciences and Humanities and Storytelling*

Game scholars have extensively studied, analyzed and critiqued games. They provide insight into how games work, what constitutes play, and how games impact culture and society. In game studies, ludologists have proposed vocabularies, ontologies and pattern languages aimed at understanding and critiquing games and play in a cultural context.

Aside from studies, game education has yielded textbooks on game design and development practice. Schell introduces the art of game design, offering theory,

⁴<https://gdcvault.com> (visited August 20th 2019)

⁵<https://gdcvault.com/gdmag> (visited August 20th 2019)

⁶<http://www.gamasutra.com> (visited August 20th 2019)

Table 2.11: Multi-disciplinary publication venues on Game Studies, Education and Storytelling

Venue	Acronym	Years	Ct.
Conference of the International Simulation and Gaming Association	ISAGA	1970–	3
Simulation & Gaming	S&G	1970–	2
Game Studies		2001–	2
European Conference on Games Based Learning	ECGBL	2007–	2
Computers & Education (Elsevier)		2000–	2
International Conference on Interactive Digital Storytelling	ICIDS	2008–	2
Conference on Technologies for Interactive Digital Storytelling and Entertainment	TIDSE	2003, 4, 6	2
International Conference on Virtual Storytelling	ICVS	2001, 3, 5, 7	2

approaches and conceptual lenses that help designers think and practice [Sch14b]. Fullerton describes a play centric approach to creating games highlighting the disciplines prototyping and play testing [FSHo8]. Salen and Zimmerman describe game design fundamentals with focus on core concepts, rules, play and culture [SZ03].

The International Conference on Interactive Digital Storytelling [CKM15; vBVD15] is the result of merging between its predecessors Technologies for Interactive Digital Storytelling and Entertainment [WGC04; ZGT⁺06], and Conference on Virtual Storytelling [BŠHo7; Szio7]. We also mention the workshops on Games and Natural Language Processing [Hor14] and Intelligent Narrative Technologies [MFB⁺14].

The International Simulation and Gaming Association (ISAGA) has organized an annual conference since 1970 [HBK07; HBo8; MBS⁺07]. ISAGA can trace its origins back to the now famous book *Homo Ludens* [Hui38], and aside from game studies also has an education focus. Related is the journal *Games & Simulation* [ZTS12]. In addition, we identify articles in the *Journal of Game studies* [Grü05; Sico8].

We highlight the following related perspectives:

- Ontological approaches and typologies, in Section 2.6.1,
- Pattern languages and design patterns, in Section 2.6.2,
- Narratives and storytelling, in Section 2.6.7,
- Educative languages, in Section 2.6.9,
- Gamification, in Section 2.6.10.

Table 2.12: Multi-disciplinary publication venues on Games and Entertainment Computing

Venue	Acronym	Years	Ct.
International Conference on the Foundations of Digital Games	FDG	2006–	12
Workshop on Procedural Content Generation in Games	PCG	2010–	9
Workshop on Design Patterns in Games	DPG	2012– 2015	5
International Conference on Entertainment Computing	ICEC	2002–	5
Workshop on Game Development and Model-Driven Software Development	GD & MDS	2011, 2012	3
Digital Games Research Association International Conference	DiGRA	2003–	7
International Conference on Intelligent Games and Simulation	GAME-ON	2000–	6
International North American Conference on Intelligent Games and Simulation	GAME-ON-NA	2005– 2011. . .	3
International Conference on Computer Games	CGAMES	2004– 2015	4
Brazilian Symposium on Computer Games and Digital Entertainment	SBGames	2002–	4
International Conference on Computers and Games	CG	1998–	2
International Conference on Advances in Computer Entertainment Technology	ACE	2004– 2018	5
International Academic Conference on the Future of Game Design and Technology	Future Play	2002– 2010	3
ACM Computers in Entertainment	CIE	2003– 2018	2

2.5.5 Multi-disciplinary Games Research

Researchers and practitioners from different fields meet at multi-disciplinary conferences on games and entertainment computing. They exchange points of view, and apply and mix diverse expertise on game design, AI and language technology, which results in synergy and inter-disciplinary advances. Although challenges, objectives approaches may differ, multi-disciplinary venues are the main source of publications included in this study, and perhaps the best source of nuanced approaches. Table 2.12 shows the publications venues we identified. We briefly describe each one.

The International Conference on the Foundations of Digital Games (FDG) is a multi-disciplinary conference that alternates between Europe and the USA. We include contributions resulting from narrow [BTP17; Hol14; vRoz15a; vRD14] and

wide [FMM⁺13b; TZE⁺15; ZBL13] queries, and we added several others by hand [AC15; HW10; SWM10a; TBM⁺12]. Two colocated workshops are of special note. The first, Design Patterns in Games (DPG) includes pattern languages and gameplay design patterns [AR13; AH14; DN12; Wet13; Wet14]. DPG was last organized in 2015, but might be revived. The second, Procedural Content Generation in Games (PCG) is concerned with generative methods for games, often using AI techniques [ALY15; BDF⁺17b; DN12; Dor10; Dor11b; LBB15; STdK⁺10; vRH18; ZR14b].

The International Conference on Entertainment Computing (ICEC) [KBW14; MSO⁺12c; MMS⁺06; SSP04; VZ10] is organized annually. The continent of the venue varies. The colocated Workshop on Game Development and Model-Driven Software Development (GD&MDSD) was organized twice [FR12; KRvR12; MSO⁺12a].

The Digital Games Research Association (DiGRA) organizes its annual DiGRA International Conference, which is a mix of game studies, humanities and technology [AR09; BLH03; CD09; Dor11a; Lem07; MS05a; MC09a; ZMF⁺05].

The International Conference on Intelligent Games and Simulation (GAMEON-ON) focuses on structured methods for programming of games that benefit industry and academia. GAME-ON is organized annually in Europe [BD10; EHvdW⁺09; MC08; NV03; SMM07; WM00], but also occurs on different continents, e.g., North America (GAME-ON-NA) [DKV06; Dor09; PVM05]. Another conference branched off from GAME-ON in 2004, as explained by Spronck [Spro4]. It was first known as Computer Games: Conference on Artificial Intelligence, Design and Education (CGAIDE) [NVG04] and later became the International Conference on Computer Games (CGames), which was last organized in 2015 [FGH⁺12; SMO⁺12; vNvOM⁺11].

The International Conference on Computers and Games (CG) is a venue that is not organized every year [Bro16; Ver03]. The Brazilian Symposium on Computer Games and Digital Entertainment (SBGames) is a national event with international visibility, and also a source of several DSLs [ARC⁺14; AVG⁺13; FSo6a; SA09].

Some venues we identified are discontinued. For instance, the Conference on Advances in Computer Entertainment Technology (ACE) was a technology oriented multi-disciplinary conference [AR11a; Bru08; EM08; NKH09; WM11]. In 2018, most members of its steering committee resigned when they could no longer guarantee an impartial review process, and the conference closed down after a community boycott that condemned the conduct of the event's owner. The International Academic Conference on the Future of Game Design and Technology (Future Play) was organized from 2002 until 2010 [Ando8b; Boj10; MV08]. The journal, ACM Computers in Entertainment (CIE) ceased in 2018 [MC09b; ES06],

In the next section, we discuss a series of research perspectives. Each of these can be considered a multi-disciplinary point of view.

2.6 RESEARCH PERSPECTIVES

We present a series of fourteen complementary research perspectives on languages of games and play. Each perspective sheds light on what informs the design and construction of good games. Together they form an overview that provides answers to our research questions RQ2 and RQ3. We acknowledge that different decompositions would have been possible, and ours is merely one of many ways to aggregate the results of this study. Our structure represents a best-effort that adheres to the phrasings and research frames of the authors whose works we summarize.

We derive the perspectives from the search results as follows. We group summaries of similar languages. We distill common research frames, highlight challenges and describe theoretical foundations, systematic techniques and practical solutions. For conciseness, we only illustrate these views with a selection of representative language summaries.

2.6.1 *Ontologies and Typologies*

In search of a common vocabulary for game studies, scholars have defined ontologies that relate written symbols, abstract concepts and real-world objects. In general, an ontology defines a set of named concepts, their properties and the relations between them, usually with the goal of categorizing objects of interest in a specific subject area or problem domain for understanding its meaning. In game studies in particular, ontologies and typologies are used to describe, categorize, analyze, understand and critique games and play.

Aarseth explains that ontologies are essential for game studies to reach a consensus on the meaning of words, concepts and relations between them [Aar11]. He scrutinizes different ontological models and describes challenges in their construction. For instance, choosing the level of description, i.e. how fine-grained the ontological elements (or meta-categories) should be, is difficult because there is no natural halting point in adding nuances, details or patterns for highlighting differences. In addition, choosing generality or specificity limits applications since general-purpose ontologies may not be as useful as ontologies constructed for a specific purpose.

There are many languages for constructing ontologies. For instance the w3c Web Ontology Language (OWL) is an XML based notation with good tool support for describing semantic relationships⁷. In some cases languages of games and play use ontologies for describing or guiding parts of the solution. Table 2.13 lists ontological languages we describe. Identified, but not summarized are a Business Process Modeling Ontology for game-based learning [RKB13], and a Gamification Ontology [Men14].

⁷<https://www.w3.org/OWL> (Visited June 1st 2019)

Table 2.13: Ontological Languages

Nr.	Language	Domain	Notation
1	Game Typology	Game studies	tables and visual diagrams
2	Game Ontology Project	Game studies	tables / pattern-language
3	Pervasive Games Ontology	Pervasive games	class diagram
4	Ontology of Journalism	Journalism	Web Ontology Language

Language

1

Game Typology*generic / framework / research*

Elverdam and Aarseth discuss a multi-dimensional typology of games [EA07], an extension of prior work [ASS03]. Instead of using game genres for classifying games, which may be arbitrary, contradictory, or overlapping, they propose using an open-ended ontological model. It has a meta-categories Player Composition, Player Relation, Struggle, Game State, and Time (Internal and External) and Space (Virtual and Physical). Virtual Space has the dimensions Perspective, Positioning, and Environment Dynamics. Figure 2.7 depicts its Perspective dimension in a visual notation. Read from left to right (and from abstract to concrete), dimensions (shown as rounded rectangles) have alternatives (marked by arrows) and sub-dimensions (horizontal dashed lines), which are also categories. On the far right are game instances, distinguished by the typology.

publication	query	publication type	research category	note
[ASS03]	language	conference paper	proposal of solution	
[EA07]	16w	journal article	proposal of solution	

Language

2

Game Ontology Project*generic / framework / practice*

Zagal et al. present the Game Ontology Project (GOP), a framework offering a unified game design vocabulary to describe, analyse and study existing games and facilitate the design of new ones [ZMF⁺05]. The ontology abstracts away representational details of games such as setting, genre and player knowledge, and its aim is to characterize the game design space. The top-level elements are Interface, Rules, Entity Manipulation and Goals. GOP is available online in Wiki form¹.

¹<https://www.gameontology.com/> (last visited November 17th 2018)

publication	query	publication type	research category	note
[ZMF ⁺ 05]	15w	conference paper	proposal of solution	
[ZMF ⁺ 07]	15w	book chapter	proposal of solution	reprint
[Zag10]	601w	book	validation research	Ludoliteracy

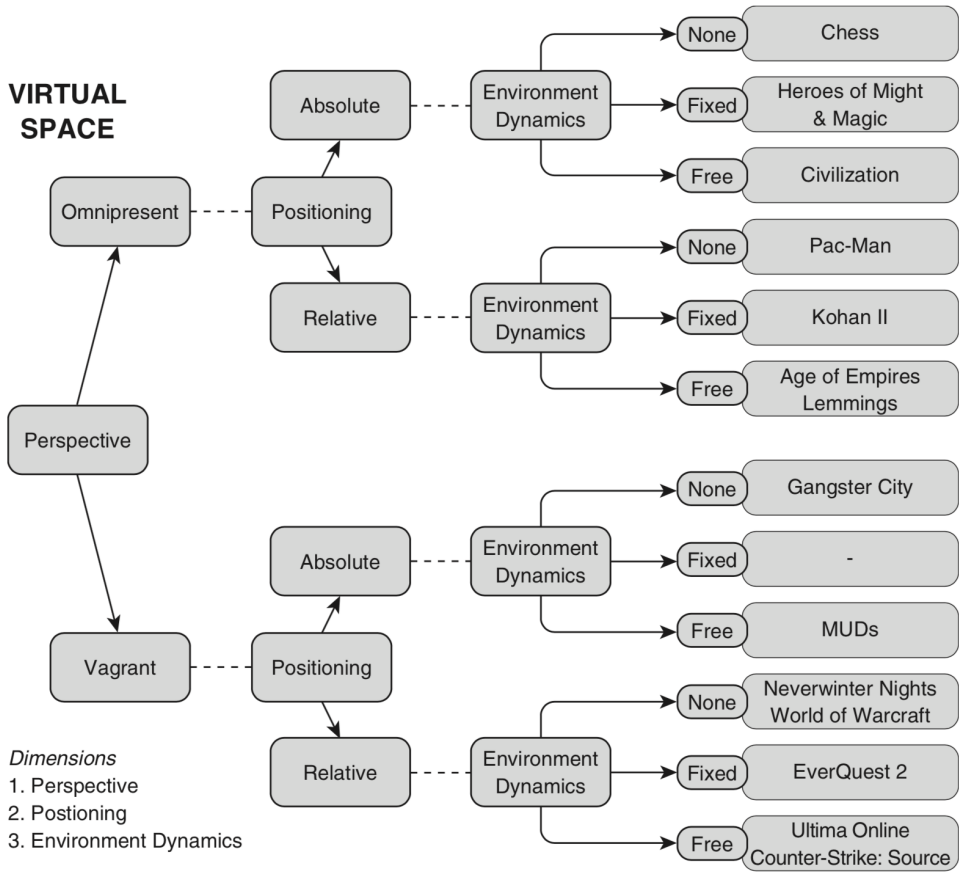


Figure 2.7: Game Typology – The Metacategory Space Containing the Dimensions Perspective, Positioning, and Environment Dynamics (appears in Elverdam and Aarseth [EA07])

Guo and Trætteberg propose a Pervasive Games Ontology (PerGO) for structuring and accelerating the process of analyzing the game domain, specifically aimed at pervasive games, i.e. games extending into the real world [GTW⁺14]. Guo et al. evaluate a model-driven development methodology for a location-based game called RealCoins [GTW⁺15b], and propose a domain-specific modeling workflow [GTW⁺15a]. Pløhn et al. extend PerGO and perform a case study on a game called Nuclear Mayhem [PKG15].

Table 2.14: Pattern-languages for analyzing games and forming gameplay hypotheses

Nr.	Language	Informs the design of
5	Formal Abstract Design Tools	gameplay goals / pattern languages
6	Game Design Patterns	gameplay goals / pattern languages
7	Gameplay Design Pattern	gameplay goals
8	Mechanics Dynamic Aesthetics	digital game systems
11	Collaboration Patterns	collaborative gameplay goals
15	Flow Experience Patterns	flow experience goals
16	Patterns Language for Serious Games Design	applied gameplay goals

publication	query	publication type	research category	note
[GTW ⁺ 14]	9n	workshop paper	proposal of solution	
[GTW ⁺ 15b]	30n	journal paper	validation research	
[GTW ⁺ 15a]	50n	conference paper	validation research	
[Guo15]	97n	PhD Thesis	evaluation research	
[PKG15]	107n	journal paper	validation research	

Language

4

Ontology of Journalism

genre-specific / tool / practice

Dowd explores how the design of persuasive learning systems for journalists can be guided by an ontology for journalism [Dow13]. The ontology describes vocabulary, concepts and emotions in the journalism domain, including social media and crowdsourcing. Robojourno is a synthetic player that helps journalists reflect on their core values by responding to emotional inputs. Defined as a Finite State Machine, it leverages links between roles, actions, before- and after-intentions and Hoare logic, e.g., “{Curiosity} publish story {satisfied}” and “{Competitive} scoop please {smug}”.

publication	query	publication type	research category	note
[Dow13]	166w	conference paper	proposal of solution	

2.6.2 Pattern Languages and Design Patterns

A pattern language describes best practices with empirically proven good results as a reusable solution to commonly recurring problems in a particular area of interest. The approach originates from Alexander et al. who describe a pattern language for towns, buildings and construction [AIS⁺77]. Often presented in table form or a template, sequential sections highlight different facets of the problem, proposed

Table 2.15: Pattern-languages offering authorial affordances over designing games and play

Nr.	Language	Game Artifact	Affects
9	Pattern Language for Sound Design	sound	sound-supported experiences
10	Verbs	abstract player actions	gameplay affordances
12	Pattern Cards for Mixed Reality Games	mixed reality game rules	mixed-reality experience
13	Design Patterns for FPS Games	structure of FPS levels	progression, experiences
14	3D Level Patterns	structure of 3D levels	progression, experiences
17	Operational Logics	depends on concrete representations	inner game workings, external player communication
26	Machinations	game-economic mechanics, feed-back loops	gameplay affordances, strategies and trade-offs

solution, examples, and related contextual information. In Software Engineering, Object Oriented (OO) design patterns are a well-known means to create, explain and understand software designs, design decisions and implementations [GH]⁺94]. In contrast, in game studies and humanities, game design patterns are a means to analyze and explain player experiences, also referred to as gameplay design patterns. The key difference between these kinds of patterns is that the former are prescriptive for structuring software and the latter are analytic regarding gameplay effects. We categorize pattern languages in two categories. The first, shown in Table 2.14, lists languages for analyzing, categorizing, understanding and critiquing gameplay. The second, shown in Table 2.15, lists languages for predicting the effect of changes to a game’s design, and making informed design decisions, e.g., level design patterns and game mechanics patterns. Programming patterns that directly affect a game’s mechanics, narratives, levels, and behaviors are discussed in Section 2.6.4, 2.6.5, 2.6.6 and 2.6.7.

Language

5

Formal Abstract Design Tools

generic / framework / practice

Church proposes Formal Abstract Design Tools (FADT), a kind of pattern language for game design, with broad categories intention, perceivable consequence and story. This influential publication has influenced later works, because it offered a new frame for research and practice for building on past discoveries, sharing concepts behind successes, and applying lessons learnt from one domain (or genre) to another.

publication	query	publication type	research category	note
[Chu99b]	-w	magazine article	proposal of solution	
[Chu99a]	-w	magazine article	proposal of solution	reprint

Language

6

Game Design Patterns

generic / framework / practice

Kreimeier proposes a pattern language for game design aimed at promoting reuse that describes problem, solution, consequence, examples and references. Example patterns include Privileged Move, for restricting actions and Weenie, for reorienting players. This work inspired later approaches.

publication	query	publication type	research category	note
[Kre02]	67w	article	proposal for solution	

Language

7

Gameplay Design Patterns

generic / framework / practice

Björk et al. propose a framework for game design patterns, which was later renamed *gameplay design patterns*, to support the design, analysis, and comparison of games. The pattern language describes components of games and interaction patterns that express how players or a computer use these components to affect aspects of gameplay. Language elements include name, description, usage, consequences, relations, relations and history. Holopainen et al. describe teaching gameplay design patterns using a tool called CAGE, which visualizes design goals as a graph of related design facets [HBK07]. Holopainen and Björk further expand the gameplay design patterns collection through exploring how games support motivation [HB08]. Zagal et al. discuss dark design patterns that cause negative player experiences and whose intent is questionable and perhaps even unethical, and how to identify them [ZBL13]. A pattern catalogue is available online in Wiki form, along with a list of related publications¹.

¹<http://www.gameplaydesignpatterns.org> (Last visited November 19th 2018)

publication	query	publication type	research category	note
[HB03]	150w	lecture notes	lecture	
[BLH03]	313w	conference paper	proposal of solution	
[BH06]	927w	book chapter	proposal of solution	reprint
[HBK07]	605w	conference paper	proposal of solution	
[HB08]	839w	conference paper	validation research	
[ZBL13]	830w	conference paper	proposal of solution	

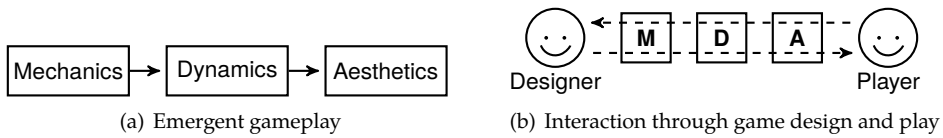


Figure 2.8: MDA perspectives (adapted from Hunnicke et al. [HLZ04])

Language 8 **Mechanics Dynamics Aesthetics** generic / framework / research

Hunnicke et al. present the Mechanics Dynamics and Aesthetic (MDA) framework for understanding games, bridging the gap between design, development, game criticism, and technical game research [HLZ04]. The framework is schematically shown in Figure 2.8. Players interact with games via mechanisms (a.k.a. mechanics, or rules) created by game designers. During a game’s execution, playful acts result in dynamic interaction sequences. Ideally, these are also aesthetically pleasing experiences called *gameplay*, e.g., fellowship, challenge, fantasy, narrative, discovery or self-expression.

publication	query	publication type	research category	note
[HLZ04]	language	workshop paper	philosophical paper	

Language 9 **Pattern Language for Sound Design** generic / tool / practice

Alves and Roque propose a sound pattern language for empowering game developers in sound design. They present a collection of illustrative patterns in an accessible format based on best practices [AR10]. In addition, they propose and evaluate a deck of cards for sound design [AR11a; AR11b] and report experiences [AR13]. Examples of sound patterns include Achievement, Failure, Anticipation, Directionality and Hurry Up! The patterns are available in Wiki form¹.

¹<http://www.soundingames.com> (Last visited March 21st 2019)

publication	query	publication type	research category	note
[AR10]	65w	conference paper	proposal of solution	
[AR11a]	575w	conference paper	evaluation research	
[AR11b]	820w	conference paper	evaluation research	
[AR13]	-w	workshop paper	experience report	

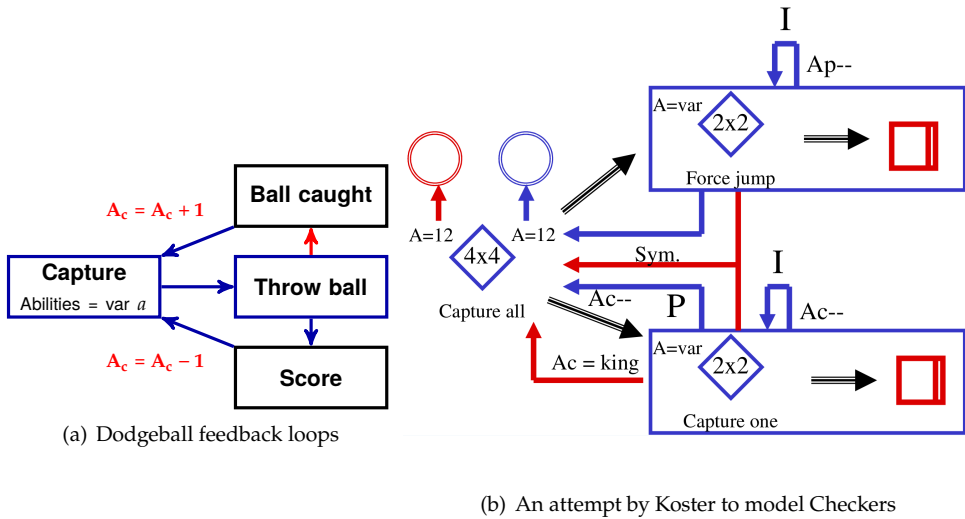


Figure 2.9: A Grammar of Gameplay (diagrams appear in Koster [Koso5a])

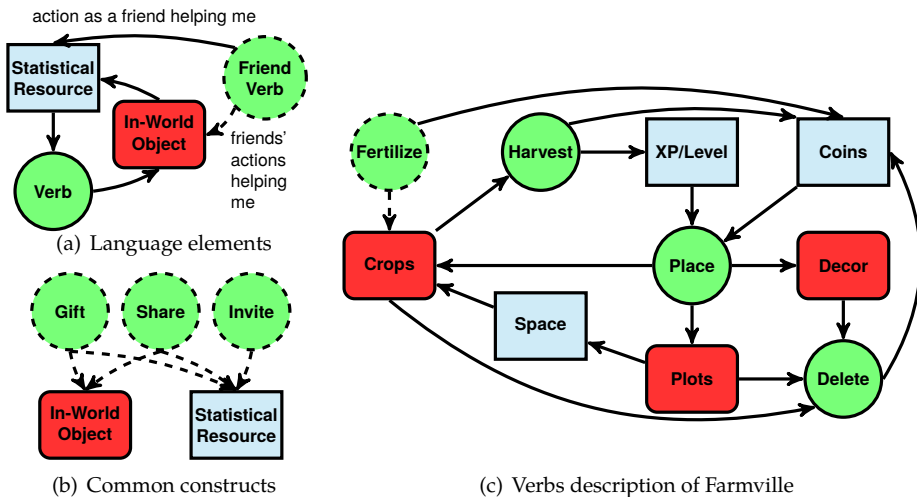


Figure 2.10: The Verbs language fits on a napkin (diagrams adapted from Koster [Kos16])

Koster proposes “A Grammar of Gameplay” [Kos05a], an informal notation for gameplay design that describes atomic player activities and affordances by using *verbs* (or *ludemes*). The notation is inspired by “A Theory of Fun” [Kos05b]. It consists of atoms shown as rectangles enclosing a verb, which can be labeled with additional information, e.g., board game size (e.g., 8x8), conditions (e.g., $var < 256$) and time passing (vertical bar on the right). The arrows denote a sub-relation between general and specific atoms, and also express success and failure outcomes (marked in blue and red respectively) that help in analyzing feedback loops, e.g., in Dodgeball as shown in Figure 2.9.

Bojin studies game design epistemologies from a language philosophy perspective that applies Wittgenstein’s language games [Boj08]. The author examines ludemes with respect to limits of formal language for expressing experiences [Boj10]. Koster challenges academics to explore these limits, and shows several excerpts from an updated and simplified Verbs language depicted in Figure 2.10 [Kos16]. The language consists of verbs (enclosed by a green circle), friend verbs (dashed line) statistical resources (light blue rectangle) and in-world objects (red rounded rectangle).

publication	query	publication type	research category	note
[Kos05a]	–w	presentation slides	proposal of solution	Grammar of G.
[Boj08]	17w	journal article	philosophical paper	language games
[Boj10]	377w	conference paper	philosophical paper	Grammar of G.
[Kos16]	–w	presentation slides	discussion piece	Verbs

Azadegan and Harteveld examine how Collaboration Engineering (CE) can help study the design of collaborative games, and how such games support collaboration through their game mechanics [AH14]. They analyse two games using facilitation techniques from CE called *Thinklets*. Thinklets are modular sets of rules intended for creating predictable patterns of collaboration that describe how people interact and work together towards a common goal. Thinklets specify preferred actions that, given constraints and capabilities, are appropriate for specific roles.

publication	query	publication type	research category	note
[AH14]	944w	workshop paper	proposal of solution	

Wetzel introduces a set of playing cards as a pattern language for designing Mixed Reality games, and presents initial findings on their application [Wet14]. These cards are available for purchase¹.

¹<https://www.pervasiveplayground.com/mixed-reality-game-cards/> (visited May 10th 2019)

publication	query	publication type	research category	note
[Wet13]	871w	workshop paper	proposal of solution	
[Wet14]	133w	workshop paper	proposal of solution	

Hullett and Whitehead present level design patterns for creating varied and interesting First Person Shooter (FPS) games [HW10]. The pattern language associates building blocks of level design to resulting gameplay, and consists of the sections description, affordances, consequences and examples. Categories and patterns include Positional Advantage (Sniper location, Gallery and Choke Point), Large-scale Combat (Arena and Stronghold), Alternate Gameplay (Turret and Vehicle Selection) and Alternate Routes (Split Level, Hidden Area and Flanking Route).

publication	query	publication type	research category	note
[HW10]	-w	conference paper	proposal of solution	

Winters and Zhu propose guiding the spacial navigation of players in 3D adventure games with structural composition patterns [WZ14]. They analyze the games *Uncharted 3*, *Dear Esther*, and *Journey*, and derive five patterns that direct a player's attention: Contrasting Shape, Framed Structure, Directional Line, Shifting Elevation and Structural Exaggeration. They perform user tests on simulated 3D environments that encode the patterns, interview the players, and show that especially Shifting Elevation and Directional Line patterns influence player movement.

publication	query	publication type	research category	note
[WZ13]	201w	poster abstract	proposal of solution	
[WZ14]	-w	conference paper	validation research	

Lemay proposes a pattern language for analyzing and understanding how a video game's elements can help trigger and maintain the most positive and intense player experiences [Lem07]. Flow patterns describe a name, problem statement, context, solution, forces affecting the problem, and examples. They relate to the facets sensation, emotion, cognition, behavior and social interaction.

publication	query	publication type	research category	note
[Lem07]	39w	conference paper	proposal of solution	

Marne proposes a pattern library for serious games design aimed at improved understanding and cooperation between project team members, organizations and stakeholders¹. Reified Knowledge is an example pattern described in detail.

¹http://seriousgames.lip6.fr/site/spip.php?page=design_patterns&lang=en (visited April 26th 2019)

publication	query	publication type	research category	note
[MWH ⁺ 12]	46w	conference paper	proposal of solution	

Mateas and Wardrip-Fruin propose a framework for game analysis called Operational Logics (OLs). They define the term as follows: "An operational logic defines an authoring (representational) strategy, supported by abstract processes or lower-level logics, for specifying the behaviors a system must exhibit in order to be understood as representing a specified domain to a specified audience." [MN09]. Thus, OLs describe both how a game functions internally and how its simulation is communicated to players. Osborn et al. expand on this work by refining and operationalizing several OLs [OWM17; Osb18]. Example logics are: Collision-, Resource-, Persistence- and Character-State Logics. Logics are shown as a pattern language with the fields: Communicative role, Abstract process, Abstract operations, Presentation, Required concepts and Provided concepts. Osborn et al. propose a new field of research called Automated Game Design Learning (AGDL) for learning game designs expressed as OLs through simulating play [OSM17].

Table 2.16: Languages for domain-experts to help design applied games scenarios

Nr.	Language	Subject matter expert	Objectives
18	StoryTec	educator, non-programmer	educate through stories that integrate learning goals
19	GameDNA	psychologist	assess cognitive processes
20	ATTAC-L	pedagogical expert	educate, prevent cyber bullying
21	EngAGe DSL	educator	improve feedback to learners
22	VR-MED	medical expert	teach family medicine

publication	query	publication type	research category	note
[MN09]	-w	conference paper	philosophical paper	
[OWM17]	-w	conference paper	proposal of solution	pattern language
[OSM17]	-w	conference paper	philosophical paper	AGDL
[Osb18]	-w	PhD thesis	validation research	

2.6.3 Applied Game Design

Applied (or serious) games have a primary purpose other than entertainment and usually require designs that incorporate subject matter knowledge. Diverse experts from education, psychology and even medical doctors can help improve game designs, e.g., for learning⁸, overcoming traumas and speeding-up recovery. The challenge is integrating domain knowledge in a game’s design to achieve specific gameplay goals such that players (for instance patients or students) learn, reflect or modify behaviors. Naturally, there are ethical and privacy implications and restrictions of studying player choices, especially if the game also serves as a diagnostic tool. Dörner et al. provide an overview of foundations, concepts and practice of serious games for prospective developers and users [DGE⁺16]. Here, we identify languages, mainly DSLs, intended for helping domain-experts design and vary scenarios of applied games, e.g., for learning and assessment, as shown in Table 2.16.

Language

18

StoryTec

generic / tool / educative

Mehm et al. present the StoryTec system, an authoring tool for non-programmers for creating Digital Education Games (DEGs) [MGR⁺09]. Story descriptions consist of

⁸Please note that we excluded the term ‘*game based learning*’ from the wide query due to the large amount of false positives.

scenes. These are visually modeled using story units that have dramatic functions, e.g., derived from the Heroes Journey template. The arrows between the units define possible paths a story can take. Step-by-step authors add details to units, such as selecting virtual characters, props that participate, and actions that modify the story state. In later work, Mehm et al. present Bat Cave, a prototyping tool for evaluation and testing DEGs [MWG⁺10]. Here, authors can define so-called Narrative Game-Based Learning Objects (NGBLOBs) that express a scene's learning context, gaming context and storytelling function. The narrative engine executes story descriptions in an XML format. This engine is extended to handle NGBLOBs and adapt DEGs to a model of learner knowledge. In a book chapter on serious games, Mehm gives an overview of authoring processes and tools, which includes StoryTec [MDM16].

publication	query	publication type	research category	note
[MGR ⁺ 09]	471w	workshop paper	proposal of solution	StoryTec
[MWG ⁺ 10]	128n	conference paper	validation research	Bat Cave
[Meh13]	238n	PhD thesis	evaluation research	both
[MDM16]	152n	textbook chapter	introduction and overview	overview

Language

19

GameDNA

generic / tool / practice

Van Nimwegen et al. describe Game Discourse Notation and Analysis (GameDNA), a graphical modeling language intended to develop serious games for assessment more effectively [vNvOM⁺11]. GameDNA is designed to improve visualization methods for the assessment of a player's cognitive processes and mental states during gameplay. Its notation is composed of two levels. The first describes narrative story elements that form the story plot. The second describes the discourse between the player and the system, and a players' corresponding mental actions. Modeling elements include player perceptual actions (see), mental actions (decide), physical actions (perform) and system actions (react/feedback) that are connected via triggers and loops.

publication	query	publication type	research category	note
[vNvOM ⁺ 11]	-w	conference paper	proposal of solution	

Language

20

ATTAC-L

genre-specific / tool / educative

Van Broeckhoven and de Troyer propose ATTAC-L: a visual modeling language for describing educational virtual scenarios that help prevent cyber bullying [vBdT13]. In addition, they apply controlled natural language to improve collaboration [vBVD15]. ATTAC-L helps pedagogical experts compose scenarios from *story bricks*, nouns and

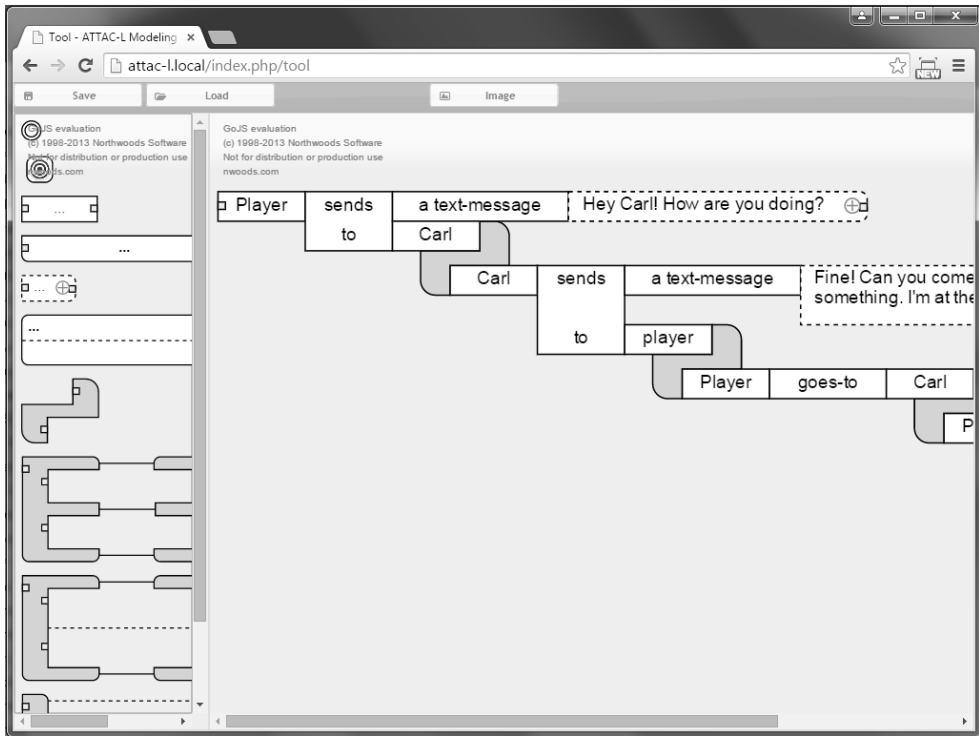


Figure 2.11: Example game narrative in ATTAC-L (appears in Broeckhoven et al. [vBVD15])

verbs that can be combined in sequence, as choices or as concurrent events. Figure 2.11 shows a screen shot of its web-based editor. Friendly Attac is a related research project¹.

¹<http://www.friendlyattac.be> – does not share software (visited April 10th 2019)

publication	query	publication type	research category	note
[vBdT13]	15n	conference paper	proposal of solution	
[vBVD15]	206n	conference paper	proposal of solution	
[vBVdT15]	693w	conference paper	validation research	
[vHSD ⁺ 16]	253w	conference paper	proposal of solution	
[dTvBV17]	123n	book chapter	validation research	

Chaudy et al. aim to improve the effectiveness of game-based learning. They propose an Engine for Assessment in Games (EngAGe) and a DSL that helps teachers take ownership of the feedback provided in serious games [CCH14]. They describe the DSL as a Feature Model (see Language 95) and implement a prototype in Xtext. Features include serious game kind, player data, learning goals, feedback messages, a feedback model, and actions related to evidence and reactions. Chaudy's website gives an overview of related research¹

¹<http://www.yaellechaudy.com> (visited July 15th 2019).

publication	query	publication type	research category	note
[CCH14]	199n	conference paper	proposal of solution	
[Bor15]	163n	PhD thesis	evaluation research	

Mossmann et al. present a prototype of VR-MED, a visual DSL that expresses game scenarios for teaching family medicine. Developers and health-care professionals can use VR-MED for creating simple games based on textual medical cases.

publication	query	publication type	research category	note
[MRF ⁺ 16]	88n	conference paper	validation research	

2.6.4 Game Mechanics

Although most agree that *game mechanics* are rules that affect gameplay, there are many different explanations, theories on how this works, e.g., [FSH04; Juu11; SZ03; Sch14b; Sico8]. In the previous sections we have described frameworks (Section 2.6.1) and design patterns (Section 2.6.2) for understanding, analyzing, and creating game mechanics. This section can be regarded as a *proceduralist view* that applies formalizations and generative techniques to program game mechanics, and analyze effects.

Table 2.17 shows languages, generators and tools for game mechanics. These works explore the limits of formalism, and study to what extent models of mechanics (and players) can be leveraged for predicting and improving a game's quality. Each language attempts to relate mechanics to aesthetics in different ways. Combinatorial rule spaces expressed with logical notations use constraints for exploring the design space and homing in on desired qualities, for instance using Answer Set-Programming

Table 2.17: Game Mechanics Languages

Nr.	Language	Mechanics	Analysis	Generation
23	Petri Nets	game-economic, story	yes	?
24	Game Space Definitions	'stock' logics	yes	
25	BIPED and LUDOCORE	combinatorial	yes	yes
26	Machinations	game-economic	yes	no
27	Micro-Machinations	game-economic	yes	yes
28	Game-o-Matic	combinatorial	yes	yes
29	Mechanic Miner	avatar-centric	yes	yes
30	Gamelan and Modular Critics	combinatorial	yes	?
31	PDDL Mechanics	combinatorial, avatar-centric	yes	yes
32	Sygnus and Gemini	combinatorial	yes	yes

for exploring the design space. Examples include BIPED and LUDOCORE (Language 25) or Gamelan and Modular Critics (Language 30).

Meaningful relationships between real-world subjects, e.g., derived from WordNet and ConceptNet, can be used to instantiate the structure of mechanics, e.g. Game Space Definitions (Language 24). Using Rhetorical arguments intended for adding meaning structure the mechanics of news games or micro-games, e.g., to convince players with political or cultural statements. Examples are Game-o-Matic (Language 28), and Sygnus and Gemini (Language 32). Sicart critiques *procedural rhetorics*, and presents opposing arguments [Sic11]. Nelson clarifies a more general position on proceduralism [Nel12], and Treanor and Mateas present an account of proceduralist meaning [TM13].

Avatar-centric mechanics encoded in ASP, rewrite rules or Java describe the physics of characters in 2D levels, such as moving, jumping and bouncing, e.g., Mechanic Miner (Language 29), Planning Domain Description Language Mechanics (Language 31) and PuzzleScript (Language 91).

Game-economic mechanics described in graph notations express how in-game resources flow and which choices players have. They foreground feed-back loops that represent investments and trade-offs. Examples are the well-known Petri Nets (Language 23), the design framework Machinations (Language 26) or its cousin the programming language Micro-Machinations (Language 27). Notably not identified, and not described here, is the ANGELINA system [CCG17].

Petri Nets are a visual notation originally intended for describing chemical processes, which has been extensively studied and applied to numerous other fields, including game design. Petri Nets are directed graphs with two node types: transitions (shown as bars) and places (shown as circles) that respectively model events and variable amounts of resources. The arrows between the nodes describe pre- and post-conditions of events. The operational semantics and mathematical properties of different classes of Petri Nets are well known, and there is ample tool support. Murata provides a historical introduction, comprehensive overview, and tutorial [Mur89].

Verbrugge proposes representing Narrative Flow Graphs (NFGs) (Language 50) as Petri Nets [Ver03]. Natkin and Vega propose describing and analyzing the non-deterministic structure of the game narration with Petri Nets [NV03]. Natkin et al. propose a methodology for spatiotemporal game design with directed hypergraphs that relate missions modeled as Petri Nets to spaces for validating mission properties such as reachability [NVG04].

Brom and Abonyi propose authoring non-linear story plots featuring intelligent virtual humans with Petri Nets [BA06]. Brom et al. describe a Petri Nets dialect that supports token ageing and its application in the design of Europe 2045, an on-line multiplayer strategy game for teaching high-school in economics, politics, and media studies [BŠH07]. Balas et al. extend the approach by combining timed colored Petri Nets and non-deterministic FSMs for developing Karo, a social simulation intended for teaching. Araújo and Roque describe an approach for modeling game systems and flow with Petri Nets for analyzing and simulating behaviors [AR09]. Ortega et al. propose Petri Nets for modeling multi-touch games systems [OHB⁺13].

publication	query	publication type	research category	note
[Ver03]	-w	conference paper	proposal of solution	story plot
[NV03]	language	conference paper	proposal of solution	game design
[NVG04]	118w	conference paper	proposal of solution	game design
[BA06]	language	workshop paper	proposal of solution	story plot
[BŠH07]	language	conference paper	validation research	story plot
[BBA ⁺ 08]	gd	conference paper	validation research	story plot
[AR09]	207w	conference paper	proposal of solution	game design
[OHB ⁺ 13]	382w	conference paper	proposal of solution	multi-touch

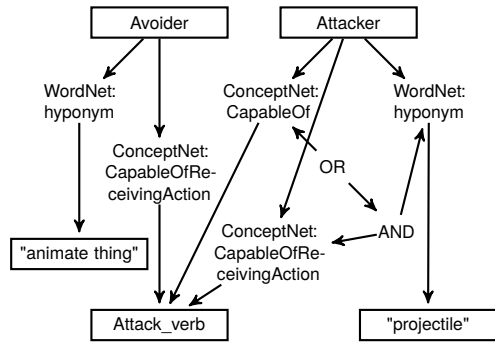
Nelson and Mateas describe an approach for automated game design that describes game mechanics of micro-games such as news games. They use WordNet and ConceptNet for relating design goals to *nouns and verbs* that instantiate predefined (or

```

noun Avoider
noun Attacker
verb Attack_verb: shoot, attack, damage, chase,
injure, hit
constraint:
  (ConceptNet CapableOfReceivingAction ?Avoider ?
  Attack_verb)
constraint:
  (WordNet hyponym ?Avoider "animate_thing")
constraint:
  (or
  (and (WordNet hyponym ?Attacker "projectile")
  (ConceptNet CapableOfReceivingAction ?
  Attacker ?Attack_verb))
  (ConceptNet CapableOf ?Attacker ?Attack_verb))

```

(a) Definition of an attacker-avoidance game space



(b) Graphical view

Figure 2.12: Example of a game space (Adapted from Nelson and Mateas [NM08a])

stock) mechanics. They demonstrate the approach by generating WarioWare-style games [NM07]. Extending the approach, they propose an interactive game design assistant that helps novice designers create games. Authoring and understanding happens in a mixed-initiative fashion, by alternating user-directed decisions and computer-generated suggestions [NM08a]. Figure 2.12 shows an example definition of an attacked-avoider game space.

publication	query	publication type	research category	note
[NM07]	135w	conference paper	position paper	
[NM08a]	328w	conference paper	proposal of solution	

Smith et al. propose a light-weight game sketching approach with computational support for a class of physical prototypes that use board-game-like elements to represent complex videogames [SNM09a]. The BIPED system supports manual and automated play testing of prototypes that are formalized using the *game sketching language*, a subset of Prolog for describing the game state, player actions and state update rules. A sketch can be played as an interactive visual representation that maps specifications to board game primitives. Connected spaces and tokens permit user actions such as clicking and dragging. Designers can also analyze its properties by specifying scenarios and conditions, obtaining feedback as logical game traces from an analysis engine that leverages Answer Set Programming (ASP). The running example is DrillBot 6000. In later work, they present LUDOCORE, a logical game engine that formalizes the event calculus and drives BIPED [SNM10]. Smith and Mateas add a notation for pattern-based

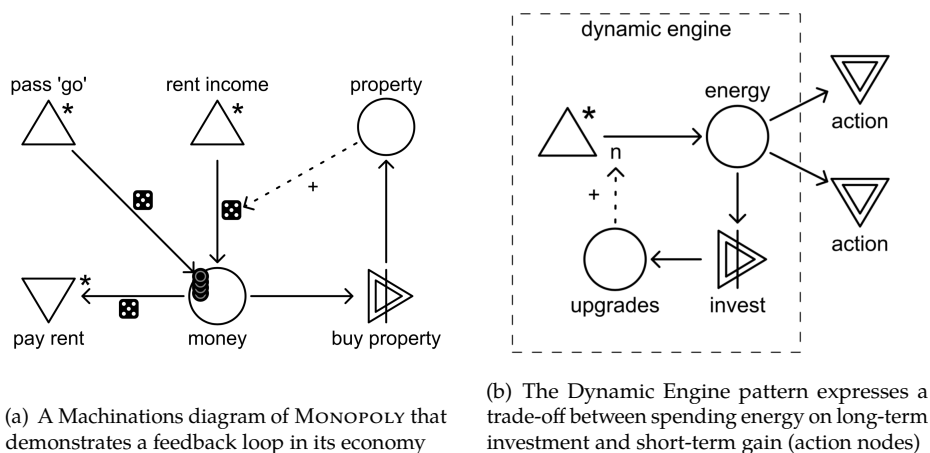


Figure 2.13: Machinations diagram and pattern (adapted from Dormans [Dor12a])

gameplay analysis [SM11b]. They describe a design space approach that leverages ASP for PCG [SM11a].

publication	query	publication type	research category	note
[NM08b]	-w	conference paper	proposal of solution	event calculus
[SNM09a]	537w	conference paper	proposal of solution	BIPED
[SNM09b]	880w	ext. abstract (demo)	proposal of solution	BIPED
[Smi09]	72n	PhD thesis proposal	proposal of solution	BIPED
[SNM10]	-w	conference paper	validation research	LUDOCORE
[SM10]	-w	conference paper	proposal of solution	ASP for PCG
[SM11b]	52w	conference paper	proposal of solution	LUDOCORE
[SM11a]	-w	journal article	validation research	ASP for PCG
[Smi12]	494w	PhD thesis	validation research	all of the above

Dormans proposes the Machinations framework as a common game design vocabulary for visualizing elemental feedback loops associated to emergent gameplay [Dor09]. Models (or diagrams) are directed graphs resembling Petri Nets (Language 23). When set in motion through play, activated nodes act by pushing or pulling economic resources along its edges. Figure 2.6.4 demonstrates a feedback loop in MONOPOLY, where buying property raises a player’s income, which can again be invested. Figure 2.6.4 shows an

example pattern from the pattern catalogue, described as a pattern language. Designers can use models for simulating and balancing games before they are built [Dor11c], and for describing emergent physics [Dor12b]. The original Machinations tool was Flash-based and now discontinued, but still available for download together with a set of examples¹. A commercial web-based tool that uses JavaScript is under development².

¹<https://machinations.io/FAQ.html> – Technical ‘old version’ (visited April 23rd 2019)

²<https://machinations.io> (visited April 23rd 2019)

publication	query	publication type	research category	note
[Dor09]	–w	conference paper	proposal of solution	
[Dor11b]	–w	workshop paper	proposal of solution	
[Dor11c]	–w	workshop paper	validation research	
[Dor12a]	97w	PhD thesis	validation research	
[Dor12b]	–w	workshop paper	proposal of solution	
[AD12]	–w	textbook	validation research	

Micro-Machinations (MM) is a textual and visual programming language, a continuation of Machinations (Language 26) that addresses several technical shortcomings of its evolutionary predecessor. Klint and van Rozen present MM, a DSL for game-economies (or game-economic mechanics) that speeds up the game development process by improving game designer productivity and design quality. MM formalizes an extended subset of Machinations features, notably adding modularity and a textual storage format. For accurately predicting a game’s behavior, they provide MM Analysis in Rascal (MM AiR), a visual framework for analyzing the reachability and invariant properties¹. In addition, van Rozen and Dormans propose a *live programming* approach for rapidly prototyping, adapting and fine-tuning game mechanics, which includes an embeddable MM library written in C++². Finally, van Rozen presents a pattern-based Mechanics Design Assistant (MeDeA) featuring pattern-based editing using an extensible and programmable *pattern palette*. MeDeA can recognize and explain patterns, and generate model extensions³. Figure 2.14 depicts an example of a game’s mechanics and shows a screenshot of MeDeA. MM AiR uses the SPIN model checker⁴. MM AiR and MeDeA leverage meta-programming features of Rascal⁵ e.g., pattern matching and visualization. A new version of MM for live programming that is based on C#, edit scripts and novel model migration techniques is ongoing work⁶.

¹<https://github.com/vrozen/MM-AiR> (visited May 14th 2019)

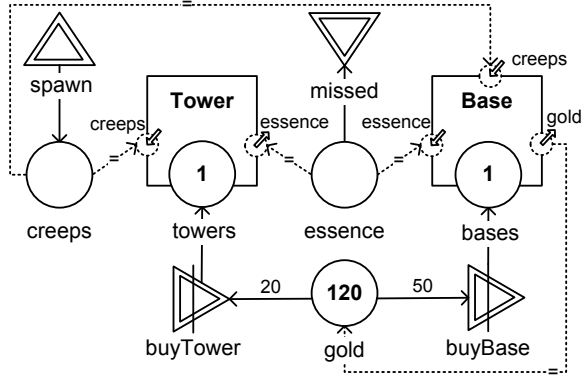
²<https://github.com/vrozen/MM-Lib> (visited May 14th 2019)

³<https://github.com/vrozen/MeDeA> (visited May 14th 2019)

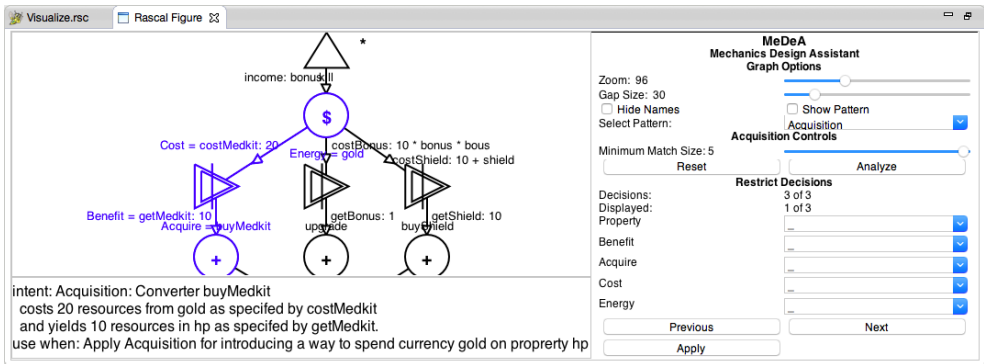
⁴<http://spinroot.com> (visited May 14th 2019)



(a) AdapTower is a tower defense game whose embedded mechanics can be modified at run time. Towers prevent creeps from passing and bases catch essence for buying more. (adapted from [vRD14])



(b) AdapTower's visual Micro-Machinations definition contains Tower and Base sub-modules (adapted from [vRD14])



(c) A Mechanics Design Assistant for pattern-based analysis and generation (adapted from [vRoz15a])

Figure 2.14: Micro-Machinations is an embeddable and modular script language for live programming of game mechanics whose qualities can be predicted using design patterns

⁵<https://www.rascal-mpl.org> (visited May 14th 2019)

⁶<http://livegamedesign.github.io> (visited May 14th 2019)

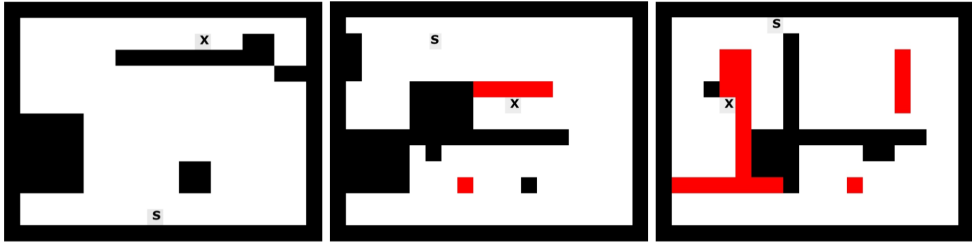


Figure 2.15: Mechanic Miner: Levels for ‘gravity inversion’ (appears in Cook et al. [CCR⁺13])

publication	query	publication type	research category	note
Chapter 3 [KvR13]	508w	conference paper	validation research	MM AiR
Chapter 4 [vRD14]	17n	conference paper	validation research	MM Lib
Chapter 5 [vRoz15a]	76n	conference paper	validation research	MeDeA

Language

28

Game-o-Matic

generic / tool / practice

Treanor et al. propose generating arcade-style videogames that represent ideas with so-called *micro-rhetorics*. Micro-rhetorics are parameterized structures with a unique id, a verb and entity roles (parameters). For instance, “A avoids B” consists of a subject A, a predicate B and a verb avoids. For each verb, Game-o-Matic randomly selects a representative micro-rhetoric from its library that form partial game descriptions. These are completed with recipes that modify the rules and completes the game’s mechanics, adding win and lose conditions and concrete structures for player interaction. Proceduralist Readings [TSB⁺11] is a related framework (see Language 32).

publication	query	publication type	research category	note
[TSB ⁺ 12]	language	conference paper	proposal of solution	
[TBM ⁺ 12]	language	workshop paper	proposal of solution	

Language

29

Mechanic Miner

generic / tool / practice

Cook et al. introduce Mechanic Miner, “an evolutionary system for discovering simple game mechanics for puzzle platform games” [CCR⁺13]. Mechanic Miner inspects and modifies the mechanics using Java reflection. Levels are generated as tile maps with accessible (white), solid (black) and deadly (red) cells. The objective is to use the mechanics

and toggle effect on and off to navigate the player (s) to the destination (x). The playability of the resulting game is evaluated using a solver that attempts sequences of button presses until it finds a combination that completes a level, which must include the mechanic. Figure 2.15 shows an example level generated for mechanic named ‘gravity inversion’, which modifies how the game engine handles gravity: `INVERTSIGN player.acceleration.y;`. Other examples include , ‘teleportation’: `HALVE player.y;` and ‘bouncing’: `ADD 1 player.elasticity;` In a large user study on a selection of generated puzzle mechanics called A Puzzling Present¹, they evaluate enjoyability and difficulty, which entailed play testing followed by questionnaires [CCG13].

¹<http://www.gamesbyangelina.org/downloads/app.html> (visited August 7th 2019)

publication	query	publication type	research category	note
[CCR ⁺ 13]	-w	conference paper	proposal of solution	
[CCG13]	284w	conference paper	evaluation research	

Language

30

Gamelan and Modular Critics

generic / tool / practice

Osborn et al. propose a framework for automated game design with computational support for play testing. The game definition language (Gamelan) models turn-taking games, such as board- and card games, and game design critics quantify gameplay qualities for automated analysis. Gamelan games consist of rules and procedures. Rules are side-effect-free relations that can only succeed or fail. In contrast, logical functions are expressions defined over a game’s current state and can succeed with different parameter bindings. Examples of critics are: no rules should go unused, players get equal turns (unfair play), repeating similar actions should be a losing strategy (dull), and rankings of players should shift frequently over the course of the game (unsuspenseful). Core Gamelan is implemented in XSB Prolog. As a demonstration they detect known design problems in a card game called Dominion.

publication	query	publication type	research category	note
[OGM13]	400w	conference paper	proposal of solution	

Language

31

Planning Domain Definition Language

generic / tool / practice

Zook and Riedl present an approach for generating game mechanics, with special focus on turn-based domains with deterministic actions and avatar-centric mechanics. Game mechanics are expressed in a subset of the Planning Domain Definition Language (PDDL) with game-specific extensions. The first, *time-indexing* enables preconditions

```

<Jump,
  {<Relative, 1, Equal(Ypos(e), Ypos(Block)+1)>,
    <Relative, 1, Equal(Xpos(e), Xpos(Block))>},
  {<Relative, 1, Update(Xpos(e), 1)>,
    <Relative, 1, Update(Ypos(e), 1)>}>

```

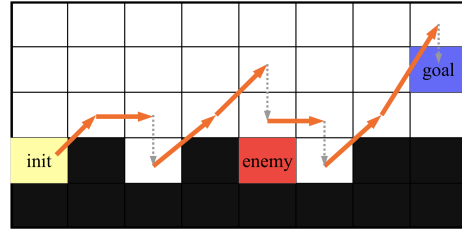
(a) 'Jump' mechanic where e is an entity that has x and y coordinates $Xpos(e)$ and $Ypos(e)$

```

<DoubleJump,
  {<Relative, 1, Equal(Ypos(e), Ypos(Block)+1)>,
    <Relative, 1, Equal(Xpos(e), Xpos(Block))>,
    <Absolute, -1, Equal(Performed(Jump), e)>},
  {<Relative, 1, Update(Xpos(e), 1)>,
    <Relative, 1, Update(Ypos(e), 2)>}>

```

(b) 'Double Jump' requires first performing 'Jump'



(c) Platformer level with a superimposed playtrace that uses a generated mechanics (shown as arrows) and gravity (shown as dotted arrows)

Figure 2.16: Generated PDDL Mechanics of a Platformer (adapted from Zook and Riedl [ZR14a])

to refer to earlier times for expressing delayed effects. The second, *coordinate frames of reference* distinguishes global world-state from avatar-relative terms for expressing avatar perception. Figure 2.16 shows an example of jumping mechanics for a platform game level. They leverage ASP to generate mechanics that conform to a set of design constraints. A planner, also implemented in ASP, analyzes the playability of those mechanics. Instead of generating mechanics from scratch, they iteratively adapt and refine mechanics, favoring fewer mechanics. The approach is demonstrated by generating combat mechanics of a role-playing game, and avatar-mechanics of a platform game, and a hybrid of the two.

publication	query	publication type	research category	note
[ZR14b]	-w	workshop paper	proposal of solution	
[ZR14a]	11.4w	conference paper	proposal of solution	

Summerville et al. propose Gemini, a system for analysis and generation of a game's mechanics [SMH⁺19]. They formalize Proceduralist Readings [TSB⁺11], a framework for interpreting and understanding the meaning of games by providing rhetorical affordances. Sygnus is a language based on ASP predicates that not only describes mechanics, processes and interactions, but also aesthetic and cultural expressions. As a result, 'meaning' can be directly derived from the source code. The Gemini system statically analyzes Sygnus programs, producing readings as reasoning chains.

Table 2.18: Tools and Languages for Mixed-Initiative Design of Levels and Virtual Worlds

Nr.	Language or Tool	Content
33	Semantic Scene Description Language	classes of concepts and relationships
34	SketchaWorld	3D worlds
35	Tanagra	platform game levels
36	Ludoscope	grammar-based transformation pipelines of 2D levels
37	The Sentient Sketchbook	tile map sketches of 2D levels
38	Evolutionary Dungeon Designer	2D level maps and patterns

publication	query	publication type	research category	note
[TSB ⁺ 11]	-w	conference paper	proposal of solution	
[SMH ⁺ 19]	173n	journal article	proposal of solution	Gemini

2.6.5 Virtual Worlds and Levels

Virtual worlds are spaces in games or simulations that through various integrated audiovisual content and interactive mechanisms support exploration, communication or play. Game worlds and levels can be populated by and player avatars, virtual characters, stories, missions, quests, etc. Procedural Level Generation is a form of PCG that focuses on generating game levels, spaces that integrate missions, quests and (lock and key) puzzles, e.g., for dungeon crawlers and platformers. Van der Linden et al. survey Procedural Dungeon Generation [vdLLB14]. We identify relatively few language-centric approaches using our protocol, since most authors in this perspective call their solutions ‘tools’.

Table 2.18 shows tools and languages for designing levels and virtual worlds. Each of these tools support creating and improving (this is usually called authoring) content a mixed-initiative style. In *mixed-initiative* approaches, intelligent services (the tool) and the designer collaborate and take turns to achieve the designer’s goals [Hor99]. Typically, designers receive visual computer-generated suggestions, and based on gradually improving insight, make decisions that refine the content iteration by iteration, in a conversational style. However, due to a lack of direct manipulation of generated content, it can be challenging to assure all possible results represent meaningful and high quality content. In this context, *semantic scenes* are structures for describing meaningful and consistent content that can guide generators. As in other perspectives, authors also apply patterns, constraints and metrics for generating and analyzing level qualities. Metrics are discussed separately in Section 2.6.8.

Tutenel et al. propose a Semantic Scene Description Language for guiding how generators produce consistent and meaningful content [TSB⁺10]. Using its visual notation, designers can express abstract *scene classes*, descriptions of scenes consisting of objects (or components), the relationship between them, and time- and context-specific variations, e.g., dining area, office, street, dungeon, forest, etc. Concrete scenes are situated instances whose constraints are solved and generated as an integral part of a larger whole. Kessing et al. propose Entika, a framework for designers that offers an editor for authoring semantic game worlds and an engine for semantic layout solving [KTB12].

publication	query	publication type	research category	note
[TSB ⁺ 10]	gd	conference paper	proposal of solution	
[KTB12]	-w	workshop paper	validation research	Entika

Smelik et al. aim to simplify modeling virtual worlds by combining semantics-based modeling and PCG techniques in a declarative modeling approach [STdK⁺11]. SketchaWorld is a tool for designers for rapidly sketching 3D worlds, which integrates two novel techniques: interactive procedural sketching and virtual world consistency maintenance.

publication	query	publication type	research category	note
[STdK ⁺ 10]	language	workshop paper	proposal of solution	
[STdK ⁺ 11]	-w	journal article	validation research	

Smith et al. describe Tanagra, a *mixed-initiative* level design tool for 2D platformers. In response to changes to the pacing of the level generates levels with corresponding “beat patterns” (sequences of obstacles) and verified playability, using constraint programming and reactive planning, Tanagra integrates reactive planning language ABL (Language 39) and numerical constraint solving.

publication	query	publication type	research category	note
[SWM10a]	-w	conference paper	proposal of solution	
[SWM10b]	783w	extended abstract	tool demonstration	
[SLH ⁺ 11]	-w	journal article	proposal of solution	

Lindenmayer systems (or L-systems) are generative grammars that were originally intended for describing plant growth patterns [Lin68] and are now also used for PCG. Dormans investigates strategies for generating levels for action adventure games, and proposes mission and spaces as two separate structures. He analyzes a Zelda game level, and generates its missions and spaces using transformative grammars [Dor10]. Ludoscope is a tool for designing procedurally generated game levels based on these principles. In Ludoscope, level transformation pipelines step-by-step transform level content, gradually adding detail, dungeons, enemies, encounters, missions, etc. These pipelines consist of grammar rules that work on content represented as tile maps, graphs, Voronoi Diagrams and Strings. Karavolos et al. explore applying Ludoscope in the design of two distinct pipelines that generate dungeons and platform levels [KBB15]. Van Rozen and Heijn propose two techniques for analyzing the quality of level generation grammars called MAD and SAnR (see Language 58).

publication	query	publication type	research category	note
[Dor10]	-w	workshop paper	proposal of solution	grammars
[DB11]	-w	journal article	proposal of solution	grammars
[Dor11b]	-w	workshop paper	proposal of solution	grammars
[DL13]	language	workshop paper	validation research	Ludoscope
[KBB15]	language	conference paper	validation research	Ludoscope
[vRH18]	-n	workshop paper	proposal of solution	Ludoscope Lite

Liapis et al. introduce The Sentient Sketchbook, a tool intended to support level designers in rapidly creating abstract game levels, which represents levels as low-resolution tile map sketches. The tool supports a mixed-initiative design and refinement process, allowing designers to choose level suggestions generated using genetic algorithms. The running example discusses a strategy game where players control a base and require resources to build units. Its tile maps consist of tiles that are passable, impassable, player bases or resources. Designers can analyze the maps for playability and visualize gameplay properties using built-in metrics that calculate properties such as navigable space, resource safety, safe areas and unused space. The tool is available on its website as a Java application or an online version, along with a list of related publications¹.

¹<http://www.sentientsketchbook.com> (visited August 12th 2019)

publication	query	publication type	research category	note
[LYT13]	language	conference paper	evaluation research	

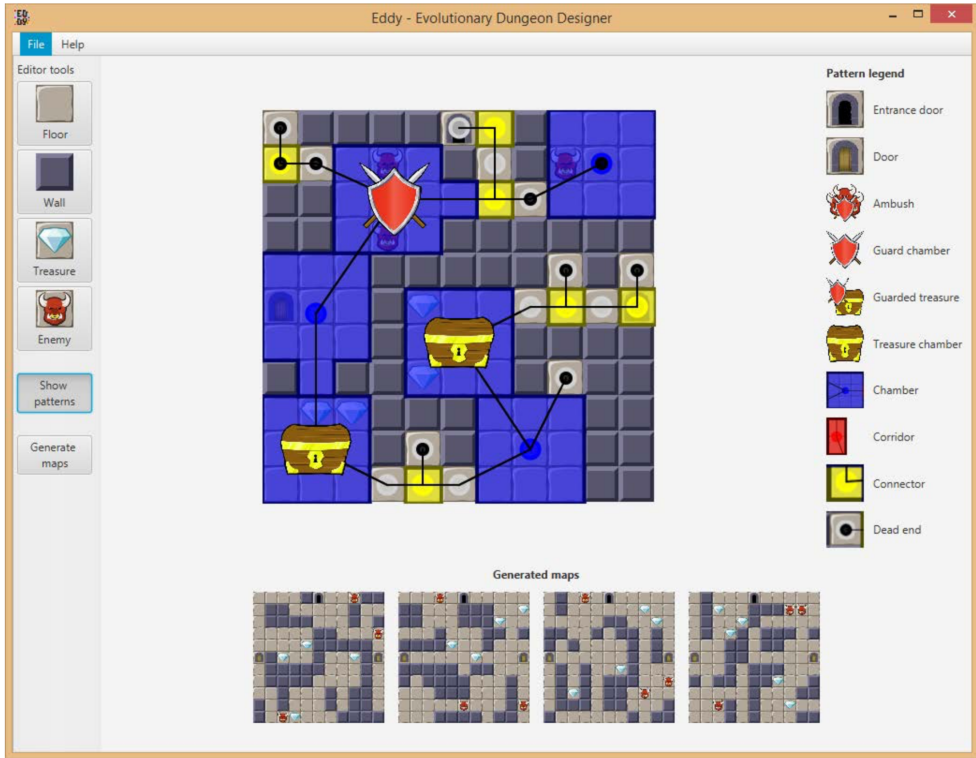


Figure 2.17: Room editing view with pattern overlay (appears in Baldwin et al. [BDF⁺17b])

Baldwin et al. present the latest iteration of the Evolutionary Dungeon Designer (EDD), a mixed-initiative level design tool that assists level designers in collaboratively creating game content [BDF⁺17a]. EDD uses evolutionary search algorithms and patterns for generating dungeon levels with desirable properties. EDD detects patterns in levels and displays on instances superimposed on the tile map, as shown in Figure 2.17. Spatial micro-patterns consist of paths and multiple tiles: corridor (red tiles), connector (yellow tile) and chamber (blue tiles). Inventorial micro-patterns placed on one tile are door, treasure and enemy. Meso-patterns are composed from spatial combinations of micro-patterns. Examples, each shown using descriptive icons, are: treasure chamber, guard chamber, ambush, dead end, and guarded treasure. Meso-patterns are detected using breadth first search of a pattern graph, starting at a room's entrance. The level quality is estimated with several fitness functions that guide the search.

Table 2.19: Languages for domain-experts to help design behaviors

Nr.	Language	Description
39	ABL	ABL is a reactive planning language for authoring believable agents with rich personality
40	SEAL	Simple Entity Annotation Language (SEAL) is A C-like script language for describing NPC behaviors
41	BEcool	BEcool is a visual graph language with sensors and actuators for describing expressive virtual agents
42	Behavior Trees	Behavior Trees is a visual language for authoring AI behaviors
43	BTNs	Behavior Transition Networks is a visual notation of hierarchical state machines for describing behaviors
44	POSH#	framework for creating behavior-based AI for robust and intuitive agent development
94	Statecharts	Modeling formalism for describing behaviors
103	RAIL	Reactive AI Language (RAIL) is a metamodel-based DSL for modeling behaviors in adventure games

publication	query	publication type	research category	note
[BDF ⁺ 17b]	-w	workshop paper	validation research	
[BDF ⁺ 17a]	206w	conference paper	validation research	

2.6.6 Behaviors

Defining behaviors of Non Player Characters (NPCs) has been an active topic of technically oriented research. A Behavior Definition Language (BDL) is a programming language that offers a notation that controls powerful AI features for describing believable virtual entities that inhabit game worlds [Ando8a]. Many BDLs are implemented as reusable software libraries that complement game engines. According to Anderson, many BDLs are DSLs that also maintain the flexibility of programming languages [Ando8a]. Challenges include developing appropriate notations and features, authoring for dramatic realism, improving scalability of parallel behaviors, and raising the fault tolerance. Table 2.19 shows a limited selection of formalisms including Reactive Planning Languages, Behavior Trees, (Hierarchical) Finite State Machines, UML and Rhapsody statecharts. Of course, many languages describe behaviors in one way or another. Our selection represents a wider set of languages.

```

joint sequential behavior OfferDrink(){
    team Trip, Grace;
    // wait for Trip's first line
    with (success_test
        { OfferDrinkMemory
          (CompletedGoalWME
            name == iInitialDrinkOffer
            status == SUCCEEDED
          }) wait;
        subgoal
          iLookAtPlayerAndWait(0.5);
        with (synchronize) subgoal
          jSuggestMartini();
        // react to Grace's line about fancy
        shakers
        with (synchronize) subgoal
          jFancyCocktailShakers();
    }
}

```



(a) Trip's 'offer drink' behavior (b) Grace's 'offer drink' behavior (c) Trip and Grace in Façade

Figure 2.18: ABL scripts adapted from Mateas and Stern [MS02] (a, b) and [MS05b] (c)

Mateas and Stern describe A Behavior Language (ABL), pronounced “able”, a reactive planning language for authoring believable agents expressing rich personality built on Hap [MS02]. ABL extends Hap with atomic behaviors, reflection, private memories and goal spawning. ABL was notably used in the interactive drama Façade¹ [MS05a; MS05b] and Tanagra, which is described as Language 35. Figure 2.18 shows example scripts that demonstrate synchronization with individual ‘i’ and joint ‘j’ subgoals. Simpkins et al. extend ABL (as A²BL) with reinforcement learning [SBI⁺08a]. Its Java sources are released under the GNU GPL².

¹<https://www.playablstudios.com/facade/> (visited May 9th 2019)

²<https://www.cc.gatech.edu/~simpkins/research/afabl/abl.html> (visited May 9th 2019)

publication	query	publication type	research category	note
[MS02]	gd	journal article	proposal of solution	ABL
[MS05a]	63w	conference paper	philosophical paper	ABL
[MS05b]	773w	conference paper	proposal of solution	ABL
[SBI ⁺ 08a]	–w	conference paper	proposal of solution	A ² BL
[SBI ⁺ 08b]	–w	journal article	proposal of solution	A ² BL, reprint

```

entity defender {
  scalar manGun;
  scalar turret;
  scalar gunAvailable = 0;

  event unused {
    manGun = getGlobal("use");
    if(manGun!=NULL) {
      turret=getEntity(manGun);
      gunAvailable = 1;
      trigger lock @ turret;
    }
  };

  state fsm {
    patrolling(), NULL;
    defending(), patrolling();
    fsm(), NULL;
  };

  event enemy_detected {
    setstate fsm::defending;
  };

  fsm::patrolling() {
    while(1) {
      /* execute 'patrolling'
      behaviour */
      ...
    }
  }

  fsm::defending {
    /* if gun-turret available */
    if(gunAvailable){
      /* man gun turret */
      manGun();
      trigger unlock @ turret;
      gunAvailable = 0;
      ...
    } else {
      /* execute default defence
      behaviour */
      ...
    }
  }

  fsm::fsm() {
    setstate patrolling;
  }

  defender() {
    setstate fsm;
  }
};

entity turret {
  event lock {
    /* stop advertising */
    setSilent();
  };

  event unlock {
    /* advertise */
    setBroadcast();
  };
  ...

  global void use() {
    /* action to fire the gun */
    action fire();
    ...
    fire();
    ...
  }

  turret(){}
};

```

(a) Defender entity

(b) Turret entity

Figure 2.19: SEAL script of a defender manning a turret (adapted from Anderson [Ando8a])

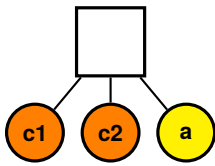
Language 40 **Simple Entity Annotation Language** *generic / engine / practice*

Anderson proposes a Simple Entity Annotation Language (SEAL), a behavior language that resembles C for scripting believable NPC behaviors with domain-specific features and datatypes, e.g., state machines. Figure 2.19 shows an example script with two entities, states, events, and associated behavior functions.

publication	query	publication type	research category	note
[Ando8b]	20n	conference paper	proposal of solution	
[Ando8a]	247n	PhD thesis	proposal of solution	

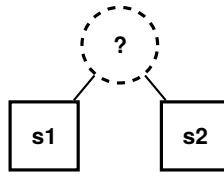
Language 41 **BEcool** *generic / engine / practice*

Szilas presents BEcool, a behavior engine for authoring high performance expressive virtual agents. Behaviors are directed graphs with nodes for animations and arrows for transitions, arrows' labels for environment's sensing (events) and dashed arrows for



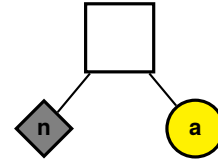
A sequence is a node, shown as a rectangle, whose success depends on each of its child nodes succeeding in sequence, where any failure ends its execution in failure. Here, conditions c1 and c2 must succeed for action a to happen.

(a) Sequence node



A selector is a node shown as a dashed circle whose success depends on trying to execute its child nodes one-by-one until one succeeds, ending the selection in success. Here, sequences s1 and s2 are alternative plans.

(b) Selector node



A lookup node, shown as a diamond, serves as a modular construct for looking up a sub-tree by name. Here, if action a happens depends on the success of the sub-tree represented by lookup n.

(c) Lookup node

Figure 2.20: Behavior Trees (visual variant adapted from Champandard [Chao7; Cha12])

event-based animation triggering. BEcool supports sequencing, branching, parallelism and inter-characters behaviors [Szi07].

publication	query	publication type	research category	note
[Szi07]	gd	conference paper	proposal of solution	

Behavior Trees (BTs) is a visual notation for authoring AI behaviors that is said to be understandable, easy to use, and scale well in parallel [Chao7; Islo5a; Islo5b]. Behaviors are modeled by hierarchies of nodes that represent plans whose details are specified in a top-down fashion. More general plans appear near the top, whereas the leaves at the bottom represent conditions and “atomic” actions exposing lower-level logic such as moves. Figure 2.20 shows three node types: sequence, selector and lookup, which can be composed with conditions and actions in authoring complex modular behaviors.

Lim et al. apply evolutionary techniques in developing competitive AI-bots that can play video games, in particular for the real-time strategy game DEFCON. Martens et al. present a formal operational semantics, a type system and an implementation [MBO18]. Variants of BTs are commonly used in practice, e.g., in Halo 2 and Spore [Chao7; Islo5b], and several implementations and engine plugins are available, e.g., BTs are a built-in feature of Unreal Engine 4¹.

¹<https://docs.unrealengine.com/en-us/Engine/AI/BehaviorTrees> (visited May 9th 2019)

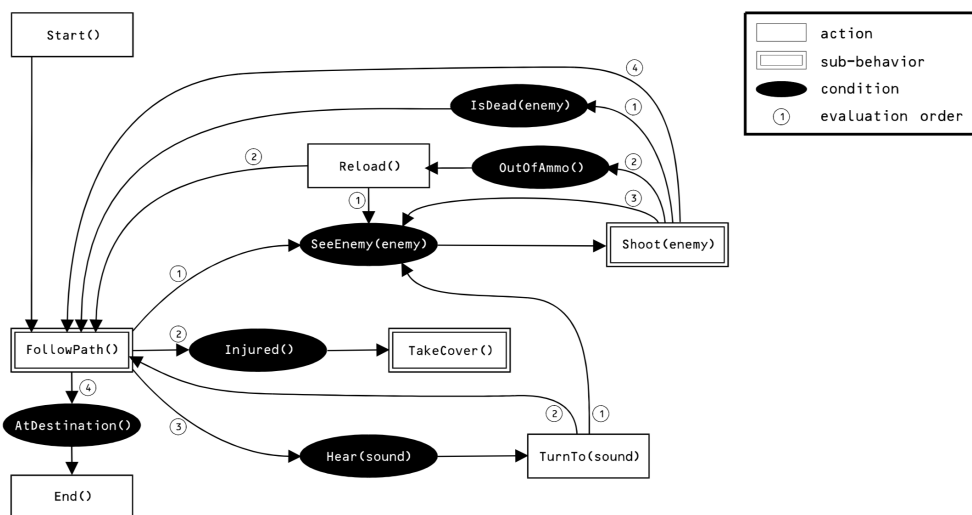


Figure 2.21: Behavior Transition Network of combat patrol behavior (appears in Fu et al. [FHJ03])

publication	query	publication type	research category	note
[Islo5b]	-	presentation (audio)	practice	
[Islo5a]	language	article	experience report	
[Chao7]	-	presentation (video)	tutorial / practice	
[LBC10]	language	conference paper	validation research	
[Cha12]	-	presentation (video)	tutorial / practice	
[MBO18]	-w	unpublished	validation research	

Fu et al. describe a visual framework for designers, developers and subject-matter experts that simplifies authoring behavior as Behavior Transition Networks (BTNs), an extension of finite state machines [FHJ03]. In addition to current states and transitions, BTNs support hierarchical decomposition, variables, communication to other BTNs and code invocation. Figure 2.21 shows an example aimed at specifying realistic behavior of NPCs in a first person shooter. SimBionic is a visual editor and a run-time engine for embedding behaviors [FHL07] that is available under the 3-clause BSD licence¹.

¹<http://www.simbionic.com/> (visited March 28th 2019)

publication	query	publication type	research category	note
[FH02]	-w	journal article	proposal of solution	BrainFrame
[FHJ03]	gd	conference paper	proposal of solution	BTNs
[FHL07]	gd	conference paper	proposal of solution	Simbionic

Language

44

POSH#

generic / engine / practice

Gaudl describes POSH# a C# framework for creating behavior-based AI for robust and intuitive agent development¹ ABODEstar is an IDE for behavior oriented design.

¹<https://github.com/suegy/posh-sharp> (Visited November 24th 2018)

publication	query	publication type	research category	note
[GDB13]	-w	conference paper		POSH
[GGG ⁺ 14]	-w	conference paper		POSH, ABL
[Gau16]	274n	PhD thesis		POSH

Language

45

DSL for AI in real-time

generic / tool / practice

Hastjarjanto et al. introduce a DSL for modeling the decision making process of the AI in real-time video games, an embedded DSL in Haskell.

publication	query	publication type	research category	note
[HJL13]	138n	workshop paper	proposal of solution	
[Has13]	155n	Master's thesis	proposal of solution	

2.6.7 Narratives and Storytelling

Storytelling, a field on its own, is concerned with writing, telling and sharing stories by means of narratives to convey ideas and experiences to an intended audience. Similar to games, stories in a cultural context, can be used for entertainment, education, cultural values, etc. The perspective we describe here is a technical interpretation that envisions programming languages for creating narratives and programming interactive stories using generative techniques. Here, we refer to the creation process as *authoring*, since the users of these languages are first and foremost authors.

Various researchers have focused on Interactive Fiction (IF), interactive drama, and the story components of games, e.g., quests, missions and stories of adventure games or educational games. Branching narratives can be expressed as graphs, where

Table 2.20: Languages for authoring, analyzing and generating narratives, stories and dramas

Nr.	Language	Expresses
39	ABL	Reactive planning language used for interactive drama
46	<e-Game>	Storyboards for educational adventure games, formally analyzed
47	ScriptEase	Patterns of behaviors, quests and stories, interactive user interface
18	StoryTec	Educational stories and related learning goals, part of a tool set
48	Ceptre	Story worlds and experimental game mechanics, offers logical proofs
49	SAGA	Stories whose compilers target different platforms
50	(P)NFG	Structured computer narratives and IF, playability and correctness
51	Wander	Number games, represents an early historical account
52	Storyboards	Generating storyboards of game levels, leveraging a planner
53	Versu	Interactive text-based drama, including social conventions
54	Tracery	Stories and art, example of a ‘casual creator’
107	Ficticious	Demonstrates the use of DSLs for Interactive Fiction

choices represent alternative sequences of events or paths. Moreover, emergent stories with generative and dramatic components requires integrating social values and knowledge of virtual personas. Challenges include checking the correctness of the paths by analyzing constraints and providing insight with appropriate visualizations and debugging facilities, e.g. into the causality of emergent scenarios. Storyboards are a sequences of images (or illustrations) and text (e.g. dialogues) used for analyzing stories of various kinds of media, including games. Kybartas and Bidarra survey story generation techniques [KB17]. We identify several languages intended for authoring, generating and analyzing narratives, summarized in Table 2.20.

Language

46

<e-Game>

genre-specific / engine / educative

Moreno-Ger et al. introduce <e-Game>, a textual DSL for describing storyboards of adventure games [MMS⁺06; MSM⁺07], which is extended and applied for game based learning [BMS⁺08; MBS⁺07]. <e-Game> and <e-Adventure> have an operational semantics, which enables formal analysis and supports model checking [MFS⁺09]. They describe the structure of storyboards using XML Schemas, which are abbreviated as Document Type Definitions (DTDs). Figure 2.22 shows DTDs of conditions (a), effects (c) and resources (e) and <e-Game> examples (b, d and f). The language also describes scenes, objects, characters, conversations and actions in a similar way. Marchiori et al. provide a Writing Environment for Educational Video games (WEEV), a visual language and tool for educational adventure game authoring that builds on prior work. The <e-Adventure> project web site¹ refers to SourceForge² for the distribution, which includes several games and Java sources, released under LGPL.

¹<http://e-adventure.e-ucm.es> (visited January 7th 2019)

```

<!ELEMENT condition (%basic-condition;|either)+>
<!ELEMENT active EMPTY>
<!ATTLIST active flag NMTOKEN #REQUIRED>
<!ELEMENT inactive EMPTY>
<!ATTLIST inactive flag NMTOKEN #REQUIRED>
<!ELEMENT either (%basic-condition;)+>
<!ENTITY %basic-condition "(active|inactive)">

```

(a) Condition Description

```

<condition>
  <active flag="FirstTaskInitiated"/>
  <either>
    <inactive flag="UsedSandSack1Container"/>
    <inactive flag="UsedSandSack2Container"/>
  </either>
</condition>

```

(b) Condition Example

```

<!ELEMENT effects ((activate|consume-object|speak-
  player|speak-char)*,trigger-cutscene?)>
<!ELEMENT activate EMPTY>
<!ATTLIST activate flag NMTOKEN #REQUIRED>
<!ELEMENT consume-object EMPTY>
<!ELEMENT speak-player (#PCDATA)>
<!ELEMENT speak-char (#PCDATA)>
<!ELEMENT trigger-cutscene EMPTY>
<!ATTLIST trigger-cutscene idTarget IDREF #REQUIRED>

```

(c) Effects Description

```

<effects>
  <speak-player>Aaaahhhh!!!</speak-player>
  <activate flag="PlayerDamaged"/>
  <trigger-cutscene idTarget="Ambulance"/>
</effects>

```

(d) Effects Example

```

<!ELEMENT resources (condition?, asset+)>
<!ATTLIST resources id ID #IMPLIED>
<!ELEMENT asset EMPTY>
<!ATTLIST asset type CDATA #REQUIRED uri CDATA #
  REQUIRED>

```

(e) Resources Description

```

<resources>
  <asset type="image/jpeg" uri="images/background1.jpg"/>
  <asset type="audio/mpeg" uri="sounds/working1.mp3"/>
</resources>

```

(f) Resources Example

Figure 2.22: <e-Game> Examples (adapted from Moreno-Ger et al. [MSM⁺07])

²<https://sourceforge.net/projects/e-adventure/> (visited January 7th 2019)

publication	query	publication type	research category	note
[MMS ⁺ 06]	11n	conference paper	proposal of solution	<e-Game>
[MSM ⁺ 07]	125w	journal article	proposal of solution	<e-Game>
[MBS ⁺ 07]	220n	conference paper	proposal of solution	<e-Adventure>
[BMS ⁺ 08]	227n	journal article	proposal of solution	<e-Adventure>
[MFS ⁺ 09]	122n	journal article	validation research	<e-Adventure>
[MTdB ⁺ 12]	126n	journal article	proposal of solution	WEEV

McNaughton et al. propose ScriptEase, a tool for game designers intended to reduce the effort in defining complex AI behaviours for Role Playing Games [MRS⁺03]. ScriptEase is an IDE for pattern-based script design that offers drop-down lists, checkboxes, etc.

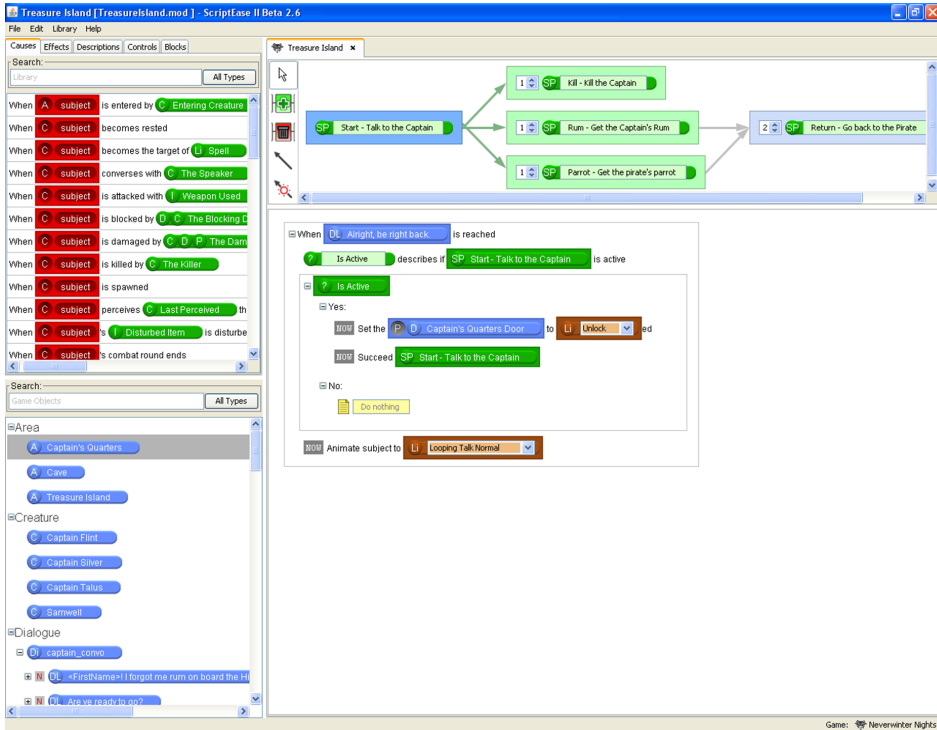


Figure 2.23: ScriptEase II – Treasure Island story (appears in Schenk et al. [SLC⁺13])

for pattern instantiation, adaptation and use. For instance, the Guard pattern specifies a guard, a guarded object and a list of situations, which consists of a name, conditions and actions. Other patterns include Patrol and Encounter.

Cutumisu et al. propose four metrics to evaluate the effectiveness of pattern catalogues [COS⁺06], and Carbonaro et al. evaluate ScriptEase in teaching [CCD⁺08]. Schenk et al. present ScriptEase II, which adds support for game-specific generators and a drag-and-drop interface that simplifies the story component [SLC⁺13]. Figure 2.23 shows a screen shot of the prototype. Its Java sources are available on GitHub¹. The main example and generator target is Neverwinter Nights, a game developed at Bioware.

¹<https://github.com/UA-ScriptEase/scriptease> (visited March 21st 2019)

```
do/flirt/conflict
: eros Flirter Flirtee * eros Other Flirtee
-o {eros Flirter Flirtee * eros Flirtee Flirter anger Other Flirter * anger Other Flirtee}.
```

Figure 2.24: Narrative action that expresses the effect of flirting in Linear Logic (adapted from Martens et al. [MFB⁺14])

publication	query	publication type	research category	note
[MRS ⁺ 03]	-w	conference paper	proposal of solution	ScriptEase
[MCS ⁺ 04b]	-w	conference paper	proposal of solution	ScriptEase
[MCS ⁺ 04a]	language	tool demo	proposal of solution	ScriptEase
[COS ⁺ 06]	-w	conference paper	validation research	ScriptEase
[COM ⁺ 07]	145n	journal article	evaluation research	ScriptEase
[CCD ⁺ 08]	-w	journal article	evaluation research	ScriptEase
[Cuto9]	-w	PhD thesis	evaluation research	ScriptEase
[SLC ⁺ 13]	-w	conference paper	proposal of solution	ScriptEase II

Martens et al. investigate the use of Linear Logic (LL) programming for expressing story worlds, settings in which the effect of narrative actions may create emergent behaviors [MFB⁺14]. Narrative actions modify story states consisting of logical predicates. Figure 2.24 shows a rule that expresses how flirting between a Flirter and a Flirtee may result in an angry lover (Other). Describing interactive stories using LL enables an interpretation of stories as logical proofs. To demonstrate this, the authors describe an example (dramatic) story world in the language Celf, and discuss how the approach enables generation, analysis, and interactive interpretation of stories.

Martens et al. present Ceptre, a language for rapid prototyping of experimental game mechanics that builds on prior work [Mar15]. Game designers and researchers can use Ceptre to create, analyze and debug ‘core systems’ and relate logical proofs to gameplay. Ceptre adds interactivity and modules called *stages* for structuring independent components. Stages run until no more actions are available (a quiescence state) allowing a transfer of control to another stage. They present two case studies. The first is an updated interactive drama. The second specifies actions and effects of a dungeon-crawler-like game. Ceptre and a tutorial are available on Github¹.

¹<https://github.com/chrisamaphone/ceptre-tutorial> (visited August 14th 2019)

publication	query	publication type	research category	note
[MFB ⁺ 14]	gd	workshop paper	proposal of solution	linear logic
[Mar15]	147w	conference paper	proposal of solution	Ceptre

Language

49

SAGA

genre-specific / engine / practice

Beyak and Carette describe SAGA, a DSL for story management meant to augment the productivity of artistic teams who create multi-platform narrative-driven RPGs [Bey11; BC11]. SAGA (Story as an Acyclic Graph Assembly) describes story states and transitions as graphs. A meta-language called AbstractCode is simplifies translating SAGA programs to different target platforms, e.g., C++, C# and Java. Figure 2.25 shows an example story graph and its associated story description. The Haskell implementation of SAGA is available online¹.

¹<http://www.cas.mcmaster.ca/~carette/SAGA/> (visited August 14th 2019)

publication	query	publication type	research category	note
[BC11]	51n	conference paper	proposal of solution	
[Bey11]	66n	Master's thesis	proposal of solution	

Language

50

(P)NFG

genre-specific / tool / practice

Picket et al. address a lack of tool support for expressing *computer game narratives*, and in particular resolving the inherent logical consistency and playability issues [PVM05]. They present a textual language and an environment for programming and analyzing structured narratives and assuring good narrative properties. This language, Programmable NFG, is based on Narrative Flow Graphs (NFGs). Since NFGs are a class of 1-safe Petri Nets (Language 23), the authors can leverage widely available techniques for formal analysis. (P)NFG's interactive narrative interpreter (and runtime) analyzes NFGs using the symbolic model checker NuSMV¹. (P)NFG has specific statements for object, state, room and more general ones such as *if*, *for* and *thread*. Figure 2.26 shows examples. As a demonstration, they model several IF games and analyze narrative properties.

¹<http://nusmv.fbk.eu> (visited August 17th 2019)

STORY Sealed Fate

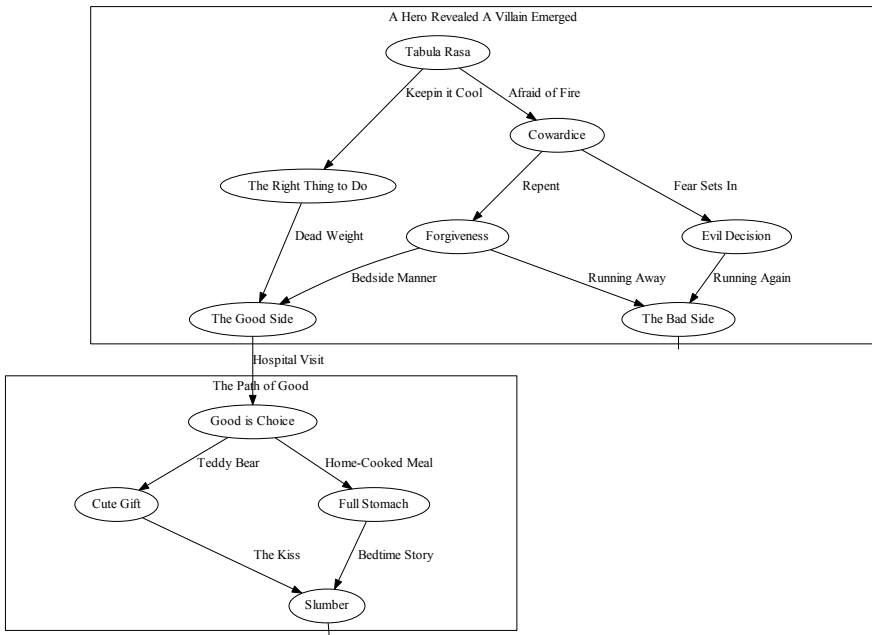
INITIAL Tabula Rasa

```
SECTION A Hero Revealed A Villain Emerged {  
  Tabula Rasa GOES The Right Thing to Do WHEN Keepin' it Cool,  
  Tabula Rasa GOES Cowardice WHEN Afraid of Fire,  
  The Right Thing to Do GOES The Good Side WHEN Dead Weight,  
  Cowardice GOES Forgiveness WHEN Repent,  
  Cowardice GOES Evil Decision WHEN Fear Sets In,  
  Forgiveness GOES The Good Side WHEN Bedside Manner,  
  Forgiveness GOES The Bad Side WHEN Running Away,  
  Evil Decision GOES The Bad Side WHEN Running Again  
}
```

```
SECTION The Path of Good {  
  Good is Choice GOES Cute Gift WHEN Teddy Bear,  
  Good is Choice GOES Full Stomach WHEN Home-Cooked Meal,  
  Cute Gift Goes Slumber WHEN The Kiss,  
  Full Stomach GOES Slumber WHEN Bedtime Story  
}
```

...

(a) Story Description



(b) (Partial) Story Graph

Figure 2.25: SAGA excerpt of "Sealed Fate" (adapted from Beyak and Carette [BC11])

```

object cloak { }

room closet {
  state {lit, locket}
}

room you {
  counter {lives 0 3 }
}

room lighthousefront {
  (you,go,north) {
    "You_are_now_on_the_mountain_pass.";
    move you from lighthousefront to mountainpass;
  }
  (you,go,east) {
    "You_are_now_behind_the_lighthouse.";
    move you from lighthousefront to
    lighthouseback;
    ...
  }
}

```

(a) Declaring an object, room with two states and a lives counter

(b) Room-specific Actions

Figure 2.26: (P)NFG code snippets (adapted from Pickett and Verbrugge [PVM05])

```

"Behold..._THE_PHONE_BOOTH_GAME!"

words (objects)
phone      0 1
telephone  1
"rotary_phone" 2

pre action
"look_phone" o?phone m=\
"The_phone_is_a_robust_contraption_with_a_rotary_dial."

#1 Telephone Booth
You are in a telephone booth.
exit 2

#2 Outside Telephone Booth
You are not in a telephone booth.
enter 1

```

(a) .misc file, containing location-independent code

(b) .world file, containing location-dependent code

Figure 2.27: Text game featuring a phone booth in Wander (adapted from Aycock [Ayc16])

publication	query	publication type	research category	note
[Ver03]	-w	conference paper	proposal of solution	NFG
[PVM05]	-w	conference paper	proposal of solution	(P)NFG
[VZ10]	-w	conference paper	validation research	(P)NFG

Language

51

Wander

genre-specific / tool / practice

In “Retrogame Archeology”, Aycock shares an excerpt of Wander, an early example of a DSL for textual adventures and ‘number games’ from 1974 that ran on mainframe [Ayc16]. Figure 2.27 shows a game featuring a phone booth.

publication	query	publication type	research category	note
[Ayc16]	186n	book chapter	historical account	

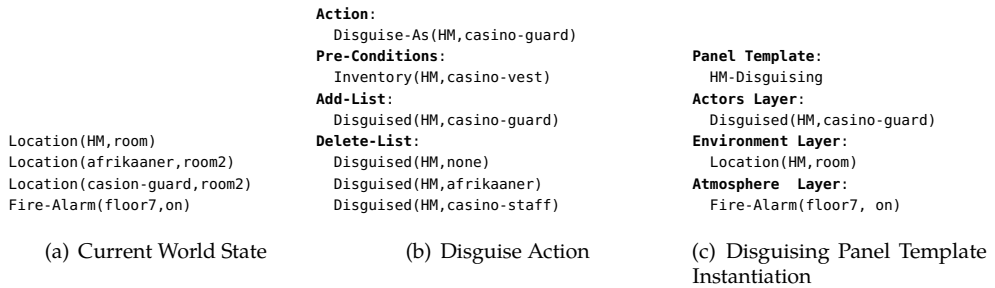


Figure 2.28: Example template instantiation from an action and a current state (adapted from Pizzi et al. [PLW⁺10])

Pizzi et al. propose an authoring tool to allow game designers to formalize, visualize, modify and validate game level solutions in the form of automatically generated storyboards [PLW⁺10; PCW⁺08]. First, AI programmers represent game worlds as a set of propositions, characterizing its *planning domain*, based on the input provided by game designers. Game states are conjunctions of propositions and transitions between states, or *planning operators*. These are represented using a Stanford Research Institute Problem Solver (STRIPS)-like formalism. Operators are categorized according to different styles which can be used in the solution generation, which is the second step. The tooling supports two modes. In the off-line mode a level designer select the style and the heuristics planner generates a complete storyboard, if one exists, depending on the constraints. In the on-line mode the level designer can simulate the level (plan) step-by-step and explore alternatives. The results are visualized in a solution tree. They validate the approach on a design of a game called Hitman. Figure 2.28 shows (a) an example world state; (b) a disguise action; and (c) a template instantiation, which is used to generate an image in the storyboard image sequence.

publication	query	publication type	research category	note
[PCW ⁺ 08]	language	conference paper	proposal of solution	
[PLW ⁺ 10]	language	journal article	validation research	

Versu is a text-based interactive drama and storytelling system that simulates autonomous agents. Praxis is a DSL for describing social practices as reactive plans that


```

process.greet.X(agent).Y(agent)
  action "Greet"
  preconditions
    // They must be co-located
    X.in!L and Y.in!L
  postconditions
    text "[X] says 'Hi' to [Y obj]"
end

```

Figure 2.29: The social convention of greeting in Praxis (adapted from Evans and Short [ES14])

provide affordances to the agents who participate in them. Prompter is an environment for authoring Praxis that speeds up the content creation process. Figure 2.29 shows an example of two agents greeting each other. Applying exclusion logic, the '!' operator expresses that variables can only have one value. In the example, agents X and Y must both be at location L. This is a more concise notation than in STRIPS (see Language 52) or PDDL (see Language 31).

publication	query	publication type	research category	note
[ES14]	-n	journal article	experience report	

Compton et al. present Tracery, a language and tool for authoring stories and art using generative grammars. Its users have created a wide variety of creative generators, e.g., visual patterns, poetry, Twitter bots and games [CKM15]. Specifications are JavaScript Object Notation (JSON) objects consisting of rewrite rules that express how strings of characters can be produced. Features include recursion and storing results. Figure 2.30 shows an example of a “Hero and Pet” story. Tracery is implemented in JavaScript. Other versions support platforms such as Python and Ruby. Users (also non-programmers) can build generators using an online visual interactive authoring environment¹

¹tracery.io (visited March 29th 2019)

publication	query	publication type	research category	note
[CKM15]	-w	conference paper	validation research	

2.6.8 Analytics and Metrics

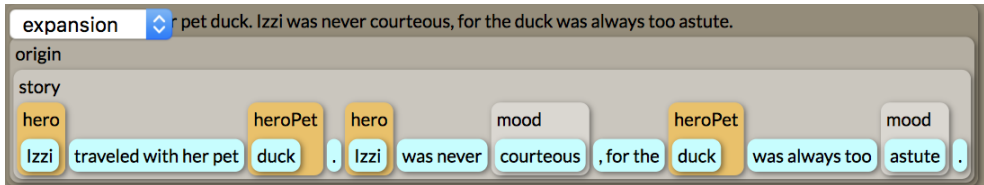
Data science combines techniques, approaches and tools from statistics, data mining, data analysis and machine learning. Data scientists leverage the available data to

```

{
  "name": ["Yuuma", "Darcy", "Mia", "Chiaki", "Izzi", "Azra", "Lina"],
  "animal": ["eagle", "owl", "lizard", "zebra", "duck", "kitten"],
  "mood": ["impassioned", "wistful", "astute", "courteous"],
  "story": ["#hero# traveled with her pet #heroPet#.
            #hero# was never #mood#, for the #heroPet# was always too #mood#."],
  "origin": ["#[hero:#name#][heroPet:#animal#]story#"]
}

```

(a) Tracery grammar of “*Hero and Pet*” in JavaScript Object Notation



(b) Example story output string and production

Figure 2.30: Tracery example of “*Hero and Pet*” stories (adapted from a tutorial on tracery.io)

extract knowledge, gain insights and predict trends. The game industry increasingly relies on game analytics for developing high quality games. One technique is using *metrics*, algorithms quantifying system properties, as measures of quality. Designers can use these metrics to test gameplay hypotheses and assess the gameplay quality by studying how metrics evolve over time. For instance, by relating player models or ‘personas’ to how game mechanics are used (Language 55). PlaySpecs can be used to analyze sequences of player actions (Language 57). Launchpad uses metrics to assess platform level quality (Language 56). In contrast MAD and SANR work directly on the engine source code of grammar-based level generators, relating gameplay- and software quality (Language 58). Fenton and Bieman describe a rigorous approach for software metrics based on measurement theory [FB14]. Research challenges include evaluating the quality of content generators [STN16], identifying suitable metrics for different types of content, and relating metrics to player models and experience.

Canossa and Drachen propose adopting the ‘personas’ framework for improving player experiences. They add *gameplay metrics*, patterns of player behaviors, to the model for analyzing how different player categories use game mechanics, e.g., by measuring the amount of jumping, solving and shooting. Designers can use gameplay metrics to compare player behaviors to their gameplay hypotheses, and use the values to estimate

player engagement. The use of data in games has evolved rapidly. In their book, *Game Analytics*, El-Nasr et al. provide an overview for a wide audience [EDC16].

publication	query	publication type	research category	note
[CD09]	142w	conference paper	proposal of solution	gameplay metrics
[EDC16]	–	textbook	multiple categories	game analytics

Language

56

Launchpad

genre-specific / engine / practice

Smith et al. describe Launchpad, a level generator for 2D platform games that can generate a wide variety of game levels. These levels are varied by adjusting rhythm parameters and level geometry [SWM⁺11]. The expressive range of a level generator is analyzed using two metrics that measure quantities of level structure associated with its qualities. The first, *linearity*, measures the aesthetic ‘profile’ of generated levels by fitting a straight line to the level (under an optimal angle), and calculating to what extent the geometry fits that line. The second, *leniency*, measures how forgiving a level is to player mistakes by aggregating a score of level elements. Gaps, enemies and falls: -1. Springs and stompers: -0.5. Moving platforms: +0.5 and jumps with no associated gaps +1.0.

publication	query	publication type	research category	note
[STW ⁺ 09]	language	conference paper	proposal of solution	rhythms
[SWM ⁺ 11]	language	journal article	validation research	Launchpad

Language

57

Playspecs

generic / tool / practice

Osborn et al. introduce PlaySpecs, regular expressions for specifying and analyzing desirable properties of game play traces, sequences of player actions. PlaySpecs are validated with the PuzzleScript engine, which is itself described as Language 91, and Prom Week, a social simulation puzzle game. The TypeScript sources of PlaySpecs are released under the MIT license¹.

¹<https://github.com/JoeOsborn/playspecs-js>

publication	query	publication type	research category	note
[OSM ⁺ 15]	930w	conference paper	validation research	
[Osb18]	–w	PhD thesis	validation research	

college students		Squeak (Lang. 62) StarlogoTNG (Lang. 64)	
high school students			
middle school children	Scratch (Lang. 63) Alice (Lang. 59)		AgentSheets and AgentCubes (Lang. 65) Gamestar Mechanic (Lang. 60)
young children	Kodu (Lang. 61)		
	computational thinking	programming	creating and designing games

Figure 2.31: Relating languages to education levels and learning goals

Language

58

MAD and SANR

generic / engine / practice

Van Rozen and Heijn study Ludoscope (Language 36) and address quality issues of grammar-based level generation. They propose two techniques for improving grammars to generate better game levels. The first, Metric of Added Detail (MAD), leverages the intuition that grammar rules gradually add detail, and uses a detail hierarchy that indicates for calculating the score of rule applications. The second, Specification Analysis Reporting (SANR) proposes a language for specifying level properties, and analyzes level generation histories, showing how properties evolve over time. LudoScope Lite (LL), a prototype that implements the techniques and demonstrates their feasibility¹.

¹<https://github.com/visknut/LudoscopeLite> (visited April 25th 2019)

publication	query	publication type	research category	note
[vRH18]	-n	workshop paper	proposal of solution	LL

2.6.9 Education

Here we describe educative languages that are end-user solutions aimed at improving learning experiences, e.g., for helping students or children learn programming, computational thinking and game design in a playful and explorative manner. Figure 2.31 shows examples, and roughly relates languages to educational activities, goals and (minimum) education levels. Educational languages usually come with an ample amount of study material. In addition, these languages may include learning programmes and web sites that cater for active online communities.

```

obj.move(forward, 1)
obj.move(forward, 1, duration=3)
obj.move(forward, 1, speed=4)
obj.move(forward, speed=2)
# change of coordinate system
obj.move(forward, 1, AsSeenBy=camera)
# different interpolation function
obj.move(forward, 1, style=abruptly)

ArmsOut = DoTogether(
    Bunny.Body.LeftArm.Turn(Left, 1/8),
    Bunny.Body.RightArm.Turn(Right, 1/8))
ArmsIn = DoTogether(
    Bunny.Body.LeftArm.Turn(Right, 1/8),
    Bunny.Body.RightArm.Turn(Left, 1/8))
BangTheDrumSIowly = DoInOrder(
    ArmsOut,
    ArmsIn,
    Bunny.PlaySound('bang'))
BangTheDrumSIowly.Loop()

```

(a) Object Movements

(b) Bunny Drum Script

Figure 2.32: Alice script example (adapted from [CAB⁺00])

Languages for learning usually pay extra attention to usability. Of special note are the “block-based languages” that enable users to fit syntactic constructs together like puzzle pieces. These do not permit syntax mistakes that can be especially frustrating to novices, and instead ensure every adjustment is meaningful and educational.

In addition, several languages use a Logo-style positional movement where one can imagine moving around. Logo is an educational programming language that is well-known for its ‘turtle’, which can be steered using commands for drawing vector graphics.

Alice is intended for authoring interactive 3D animations, and teaching programming constructs to undergraduates with no prior programming knowledge. Figure 2.32 shows a textual Alice example that uses Logo-style coordination and movement. Whereas the earlier versions were textual and based on Python, later version of Alice support mediated transfer from block based notation to script. Alice is available online for free¹.

¹<http://www.alice.org> (visited March 27th 2019)

publication	query	publication type	research category	note
[Con97]	gd	PhD thesis	proposal of solution	
[CAB ⁺ 00]	g	conference paper	experience report	
[VJP09]	108w	conference paper	proposal of solution	Cheshire



(a) Visual Kodu Rule

(b) Textual Kodu Rules

Figure 2.33: Visual and Textual Kodu Examples (adapted from MacLaurin [Mac11b])

Language

60

Gamestar Mechanic

genre-specific / tool / educative

Salen presents the overview of the pedagogy and development process of Gamestar Mechanic, an RPG style online game for teaching the fundamentals of game design to children aged 7 to 14. Games presents the results of a study into teaching middle school children to think like game designers by repairing broken games and developing games from scratch given specifications [Gam10]. The game is available commercially¹.

¹<http://gamestarmechanic.com/> (visited April 10th 2019)

publication	query	publication type	research category	note
[Sal07]	2w	journal article	proposal of solution	
[Gam10]	19w	journal article	validation research	

Language

61

Kodu

application-specific / tool / educative

Kodu is a graphical programming language for helping young children learn through independent playful exploration. Kodu expresses rules that control robots in a realtime 3D gaming environment. Figure 2.33(a) shows a visual rule, which states that when a monster sees a red apple, it moves towards it. Figure 2.33(b) shows two textual rules. Kodu GameLab and educational materials are available from Microsoft Research¹.

¹<https://www.kodugamelab.com> (visited March 27th 2019)

publication	query	publication type	research category	note
[Sto10]	gd	manual	practice	
[Mac11a]	466w	invited talk	proposal of solution	
[Mac11b]	466w	journal article	proposal of solution	reprint

Squeak is a dialect of Smalltalk, an object-oriented, class-based, and reflective programming language¹. Its *Morphic framework* facilitates visual and interactive programming and debugging of applications for domains such as education, gaming and research. Masuch and Rueger report experiences on using Squeak for teaching game design [MR05]. They investigate requirements for a collaborative learning environment that uses OpenCroquet, an audio-visual 3D environment that has built-in features supporting collaboration. Because the OpenCroquet project web site no longer exists, we share a blog with several related frameworks².

¹<http://squeak.org> (visited January 9th 2019)

²<http://planetcroquet.squeak.org> (visited March 27th 2019)

publication	query	publication type	research category	note
[MR05]	142w	conference paper	experience report	

Scratch is an visual environment for creating, designing and remixing interactive stories, games, animations, and simulations, which is intended for children between 6 and 12 years old. Scratch is a block based language implemented on Squeak (Language 62) whose its syntactic constructs fit together as puzzle pieces, for learning creative thinking and understanding logic and programming concepts [RMM⁺09]. A web-based editor of the current version and a large collection of projects contributed by its users are available online¹.

¹<https://scratch.mit.edu/> (visited March 27th 2019)

publication	query	publication type	research category	note
[RMM ⁺ 09]	gd	journal article	experience report	
[CCK12]	56w	short paper	evaluation research	

StarLogo The Next Generation (TNG) is a language, tool and 3D simulation environment for novices for creating and understanding complex systems such as games. The language is a block-based extension of Logo, a dialect of Lisp and a successor of StarLogo. Its elements are represented as colored blocks that fit together like puzzle pieces that do not permit syntax mistakes. As such, it lends itself teaching introductory

game development [BK05; WMW⁺06]. A distribution is available for Windows and MAC OS¹. An open source version, OpenStarLogo, is available under the MIT license². A successor called StarLogo Nova can be used online³.

¹<http://web.mit.edu/mitstep/projects/starlogo-tng.html> (visited January 8th 2019)

²<http://web.mit.edu/mitstep/openstarlogo/index.html> (visited January 8th 2019)

³<https://www.slnova.org> (visited January 8th 2019)

publication	query	publication type	research category	note
[BK05]	170w	journal article	proposal of solution	
[WMW ⁺ 06]	12w	extended abstract	experience report	

Repenning proposes Agentsheets, a tool for building domain-specific visual environments [RS95]. Later, AgentSheets becomes a tool for creating agent-based games and simulation, also used for teaching game design. In *conversational programming*, when a programmer edits a game or simulation, an agent executes the program and provides syntactic and semantic feedback [Rep11]. Ioannidou et al. propose AgentCubes, a 3D game-authoring environment for teaching middle school children modeling, animation and programming. Both are commercial products¹.

¹<http://www.agentsheets.com> (visited April 17th 2019)

publication	query	publication type	research category	note
[RS95]	game	journal article	proposal of solution	AgentSheets
[IRW08]	151w	conference paper	proposal of solution	AgentCubes
[Rep11]	488w	conference paper	demo paper	AgentSheets
[Ahm11]	75w	conference paper	proposal of solution	AgentWeb
[AJR12]	34n	conference paper	evaluation research	AgentSheets
[Ahm12]	158n	PhD Thesis	evaluation research	AgentWeb

2.6.10 Gamification

Gamification aims to apply or retrofit standard game designs to a new or existing system to improve user experiences⁹. For instance, score, competition and reward systems have been used in areas like online marketing to stimulate participation with a product or service. We identify languages intended to gamify information

⁹Please note that we excluded the term 'gamification' from the wide query due to the large amount of false positives.

systems in general, e.g., GAML (Language 66), UAREI (Language 68) and GLiSMo (Language 67). While these languages are technically reusable, they lack subject matter concepts that help domain experts solve design problems. Recently the term *playification* has been used to describe gamification that facilitates play more effectively by means of tailor-made game designs.

Language

66

Gamification Language

generic / tool / practice

Herzig et al. aim to enrich information systems with game design elements that increase the engagement and motivation of its users [HJM⁺13]. The Gamification Language (GAML) is a textual and declarative DSL that helps domain-experts define these elements, and helps IT experts more easily incorporate them.

Matallaoui et al. propose a model-driven architecture for designing and generating building blocks of serious games, and apply it in the creation of an achievement system [MHZ15]. An Xtext grammar is available under the MIT license¹.

¹<https://github.com/AmirIKM/GamiAsService/> (visited April 10th 2019)

publication	query	publication type	research category	note
[HJM ⁺ 13]	5n	conference paper	proposal of solution	
[MHZ15]	7n	conference paper	validation research	

Language

67

GLiSMo

genre-specific / tool / educative

Thillainathan et al. aim to enable educators without prior programming skills to create didactically sound serious games [TL14]. They propose Serious Game Logic and Structure Modeling Language (GLiSMo), a visual modeling language that applies model-driven development techniques for generating games from visual models.

publication	query	publication type	research category	note
[TL14]	14n	conference paper	proposal of solution	

Language

68

UAREI

generic / tool / practice

Ašeriškis et al. present User-Action-Rule-Entities-Interface (UAREI), a visual language for representing game mechanics for software gamification. They use UAREI to simulate and evaluate the effects of gamified systems on different types of players. UAREI models are directed graphs consisting of five node types (one for each word in the acronym) with intentionally limited expressiveness. An operational semantics is not defined. Case studies include OilTrader and the Trogon Project Management System.

Table 2.21: Game Description Languages (GDLs) for General Game Playing

Nr.	Language	Game domain	Examples of represented games
69	Multigame	mainly board games	Chess, Checkers
70	(Stanford) GDL	combinatorial games	various combinatorial games
71	'Rule Sets'	pac-man-like games	generated games
72	Ludi GDL	combinatorial games	e.g. Javalath
73	Strategy GDL	strategy games	Rock Paper Scissors, Dune II
74	Card GDL	card games	Texas hold'em, Blacjack, Uno
75	Video GDL	video games	a variety of classic 2D games
76	RECYCLE	card games	Agram, Pairs, War

publication	query	publication type	research category	note
[ABD17]	120n	journal article	solution proposal	
[AD17]	127n	workshop paper	solution proposal	
[Aše17]	241n	PhD thesis	validation research	

2.6.11 General Game Playing

A key research challenge in AI is developing generally applicable intelligent techniques capable of solving a wide range of complex problems. General game playing, for instance, refers to algorithms that can play many games well.

Game Description Languages (GDLs) are notations with expressive power over restricted game domains intended for evaluating the performance of AI techniques called *general game players* against a wide variety of manually created or generated games. GDLs are meant to cover a varied and representative game space.

The systems that execute GDL programs are used for validating if these techniques are indeed more widely applicable, e.g., by letting them compete. General game players can be search-based [TYS⁺11], e.g., Monte-Carlo Tree Search (MCTS) or leverage Machine Learning, e.g., genetic algorithms or neural networks. In a guest editorial of a special issue on "general games", Browne et al. summarize the state-of-the-art, challenges and directions for future research [BTS14]. We show representative GDLs in Table 2.21 whose domains reflect a shifting research interest of the community over the years. Notably not identified by this study is Zillions of Games¹⁰.

General game playing has the advantage of developing and applying the state-of-the-art AI to digital games, offering advanced tools, simulations and analyses. GDLs and are not necessarily suited for fine-tuning rules and improving gameplay

¹⁰<http://www.zillions-of-games.com> (visited April 3rd 2019)

```

knight_move =
  find own knight,
  pickup, orthogonal, step,
  either rotate 45 or rotate -45, step,
  not points at own piece, putdown.

dimensions (3,3)
symmetry all directions
pieces { mark 'X' 'O' }
main =
  irreversible, # each move is a conversion
  try new_mark else draw.
new_mark =
  find empty field,
  replace by mark,
  try [ test three_in_a_row, win ].
three_in_a_row =
  find own piece, # start from any position
  any direction,
  repeat 2 times [ step, points at own piece ].

```

(a) A Knight's move in Chess

(b) Tic Tac Toe

Figure 2.34: Multigame Examples (adapted from Romein et al. [RBG97] (a) and [RBG00] (b))

as in automated game design Section 2.7.3. Therefore, not all GDLs are a-priori well-suited for developing games, especially not those intended as research platforms. For instance, the language features of VGDL are course grained and the Stanford GDL is low-level and verbose. GDLs for restricted game domains, such as board- and card games, represent a concise and expressive middle ground.

Language

69

Multigame

genre-specific / tool / research

Romein et al. present Multigame, a procedural DSL for expressing the rules of board games intended to research automatic parallelism and parallel game tree search in particular [RBG97]. Multigame helps programmers focus on choosing parameters that influence behavior instead of resolving issues in communication, synchronization, work- and data distribution and deadlocks. The manual describes Chess and Checkers [RBG00]. Figure 2.34 shows two simpler examples.

publication	query	publication type	research category	note
[RBG97]	gd	conference paper	validation research	
[RBG00]	gd	manual	validation research	
[Rom01]	gd	PhD thesis	validation research	

Language

70

Game Description Language

generic / engine / research

General game playing studies how generic AI algorithms and techniques can help computer systems play more than one game successfully. The Game Description Language (GDL) provides a formal description of a game's rules that systems can use

```

role(candidate).
role(random).
init(closed(1)).
init(closed(2)).
init(closed(3)).
init(step(1)).

legal(random,hide_car(?d))
  <= true(step(1)), true(closed(?d)).
legal(random,open_door(?d))
  <= true(step(2)),
  true(closed(?d)),
  not true(car(?d)),
  not true(chosen(?d)).
legal(random,noop) <= true(step(3)).

legal(candidate,choose(?d))
  <= true(step(1)), true(closed(?d)).
legal(candidate,noop) <= true(step(2)).
legal(candidate,noop) <= true(step(3)).
legal(candidate,switch) <= true(step(2)).
sees(candidate,?d)
  <= does(random,open_door(?d)).

next(car(?d))
  <= does(random,hide_car(?d)).
next(car(?d)) <= true(car(?d)).
next(closed(?d))
  <= true(closed(?d)),
  not does(random,open_door(?d)).
next(chosen(?d))
  <= does(candidate,choose(?d)).
next(chosen(?d))
  <= true(chosen(?d)),
  not does(candidate,switch).
next(chosen(?d))
  <= does(candidate,switch),
  true(closed(?d)),
  not true(chosen(?d)).
next(step(2)) <= true(step(1)).
next(step(3)) <= true(step(2)).
next(step(4)) <= true(step(3)).
terminal <= true(step(4)).
goal(candidate, 100)
  <= true(chosen(?d)), true(car(?d)).
goal(candidate, 0)
  <= true(chosen(?d)), not true(car(?d)).

```

There are two players, a candidate and a host. Initially, three doors are closed. The host may first hide a car behind a door, and open a closed door at step 2 if it does not conceal a car and is not chosen. The candidate may first choose a closed door, optionally switch doors at step 2, and sees it when the host opens a door. When a car is hidden behind a door it remains there. Doors remain closed when not opened. When the candidate chooses a door, it remains chosen unless they switch. Steps are sequential and the candidate wins only when choosing correctly.

Figure 2.35: GDL description of the Monty Hall game (adapted from Thielscher [Thi11b])

as a testbed for intelligent agents and algorithms. Thielscher translates GDL into action language semantics [Thi11b] and introduces GDL-II, an extension of GDL for incomplete information games [Thi10]. Figure 2.35 shows a simple example that explicitly defines turn-taking, next states and sequence. Stanford hosts a web site, which refers to the annual general game playing competition and also includes additional examples¹.

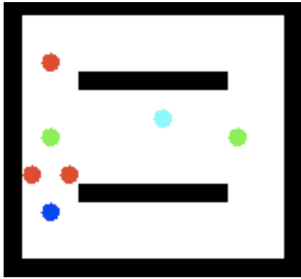
¹<http://gpp.stanford.edu> (visited March 26th 2019)

```

tmax= 28
scoremax= 6
0 Red things, random short
4 Green things, clockwise
9 Blue things, still

//When Red and Green things collide, Red survives and
  Green dies, and the score is -1-1 = -2.
collision: Red, Green → none, death, -1, -1
collision: Red, Blue → death, death, 1, 1
collision: Red, Agent → death, death, -1, 0
collision: Green, Blue → none, death, -1, -1
collision: Green, Agent → teleport, none, -1, 1
collision: Blue, Agent → death, none, 1, 1

```



(a) Race against green: score 6 within 28 time steps (b) A game’s start state: things and the agent (cyan) are randomly placed on the fixed grid

Figure 2.36: Rule set of a Pac-man-like game (adapted from Togelius and Schmidhuber [TS08])

publication	query	publication type	research category	note
[LHH ⁺ 08]	g&	technical report	report	GDL
[Thi10]	g&	conference paper	validation research	GDL-II
[Thi11a]	g&	conference paper	validation research	GDL
[Thi11b]	507w	book chapter	validation research	GDL
[Saf14]	g&	journal article	validation research	GDL
[Sad17]	217n	Master’s thesis	validation research	GDL-II

Language

71

Rules of Pac-man-like Games application-specific / tool / research

Togelius and Schmidhuber propose automatic game design as a means to generalize AI techniques. They demonstrate playable rule sets can be generated and evolved for the restricted domain of Pac-man-like games [TS08]. Games consist of a fixed grid of cells populated by an agent (cyan) and things (red, blue and green) with random start positions. Variable movement logics allow for the agent and the things to move and collide, i.e. end up on the same cell. The rule space consists of parameters for limiting the amount of time steps $t_{max} \in \{0..100\}$ and scoring $score_{max} \in \{1..50\}$, the movement logic, and effects of collisions on things (none, death or teleportation to random cell) and scoring (limited to $-1, 0, +1$). Figure 2.36 shows an evolved rule set of a game where the objective is to compete with green things to catch blue things.

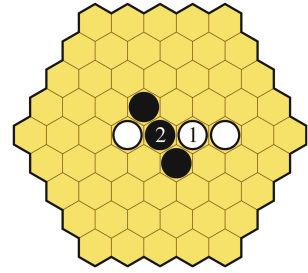
publication	query	publication type	research category	note
[TS08]	83w	conference paper	proposal of solution	

```
(game Tic-Tac-Toe
  (players White Black)
  (board
    (tiling square i-nbors)
    (size 3 3)
  )
  (end
    (All win (in-a-row 3))
  )
)
```

(a) Tic Tac Toe

```
(game Yavalath
  (players White Black)
  (board (tiling hex)
    (shape hex) (size 5))
  (end
    (All win (in-a-row 4))
    (All lose
      (and (in-a-row 3)
        (not (in-a-row 4))))
  )
)
```

(b) Yavalath



(c) Yavalath: white forces a win (adapted from Browne [Bro11])

Figure 2.37: Ludi GDL – Source code examples adapted from Browne and Maire [BM10]

Browne and Maire examine how to synthesize and evaluate high quality combinatorial games using evolutionary game design, an approach that combines evolutionary search with quality measurements in self-play simulations. They describe Ludi, a game system that synthesizes board games and evaluates their qualities [BM10]. Ludi’s Game Description Language includes game facets (called *ludemes*) for player (name), board (shapes and size), pieces with definitions of how they can move and end conditions. Figure 2.37 shows two examples. Yavalath is a novel commercially published game generated by Ludi where making four-in-a-row is winning, but making three-in-a-row before is losing. Browne proposes generating context-free grammars by analyzing the class hierarchies of game systems, in particular Ludii, to obtain so-called *class grammars* for varying constructor parameters and evolving games [Bro16]. Ludii is being developed in the context of the Digital Ludeme Project¹, which studies how historical games developed by means of modern AI techniques.

¹<http://ludeme.eu> (visited March 25th 2019)

publication	query	publication type	research category	note
[Bro08]	-w	PhD thesis	evaluation research	Ludi
[BM10]	99w	journal article	evaluation research	Ludi
[Bro11]	803w	book	evaluation research	Ludi
[Bro16]	80n	conference paper	proposal of solution	Ludii

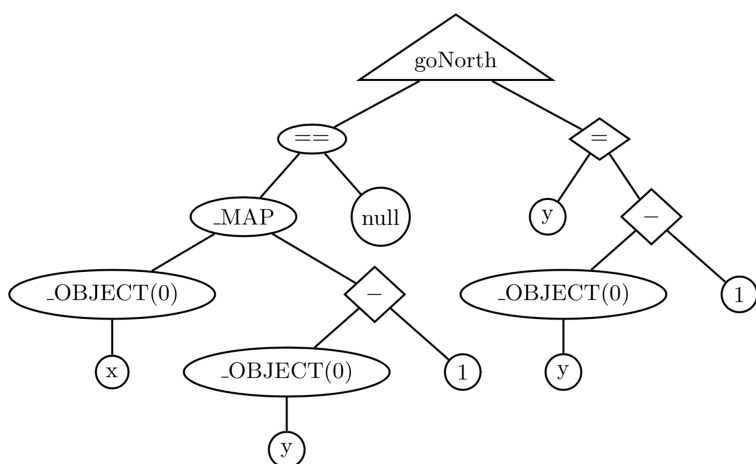


Figure 2.38: Strategy Game Description Language – “Go North” action (adapted from Mahlmann et al. [MTY11a])

Mahlmann et al. propose the Strategy Game Description Language (SGDL). Combined with evolutionary algorithms and appropriate fitness functions, SGDL serves as a means to describe and generate complete new strategy games, and variations of old ones [MTY11a; MTY11b]. SGDL visually models behaviors as trees of conditions and consequences. These are expressions and statements whose nodes are actions (triangles) comparators and functions (ovals), operators (diamonds) and constants (circles). Figure 2.38 shows a simple example of a ‘Go North’ action. In its left hand condition, the *_Map* function takes attributes *x* and *y - 1* as input, and its right hand consequence $y = y - 1$ happens if the output equals *null*. Other examples include complex variations of Rock Paper Scissors and Dune II [Mah13].

publication	query	publication type	research category	note
[MTY11a]	646w	conference paper	validation research	
[MTY11b]	-w	conference paper	validation research	
[Mah13]	289n	PhD thesis	validation research	

THE ANT AND THE GRASSHOPPER**Stages and rules**

Stage 0

COMPUTER COMMAND <Unconditional> GIVE Player: 0 Amount : 89 tokens
COMPUTER COMMAND <Unconditional> DEAL Table: 0 Amount: 1 cards
COMPUTER COMMAND <Unconditional> GIVE Player: 2 Amount : 39 tokens
COMPUTER COMMAND <Unconditional> GIVE Player: <all> Amount: 87 tokens

Stage 1

if SHOW >= T0 then PLAY IT
COMPUTER COMMAND <Unconditional> DEAL Player: <all> Amount: 6 cards
COMPUTER COMMAND <Unconditional> GIVE Player: <all> Amount: 58 tokens
PLAY ONLY ONCE if SHOW SAME RANK T1 then PLAY IT

Stage 2

if SHOW < T1 then PLAY IT
PLAY ONLY ONCE if SHOW >= T0 then PLAY IT
PLAY ONLY ONCE if SHOW > T0 then PLAY IT

Stage 3

COMPUTER COMMAND <Unconditional> GIVE Player: 0 Amount : 77 tokens

Stage 4

COMPUTER COMMAND <Unconditional> GIVE Player: 0 Amount : 44 tokens
MANDATORY if PLAY 994, > T0 then BET

Stage 5

PLAY ONLY ONCE if DRAW then BET
if SHOW >= T0 then PLAY IT
COMPUTER COMMAND <Unconditional> GIVE Player: <all> Amount: 63 tokens
PLAY ONLY ONCE if DRAW then BET

Ranking

Card(s)	Value
Four of a kind	190
6 + 8 + Jack	212

Winning conditions

5 points for each token.
3 points for finishing the game.

Figure 2.39: Card Game Description Language – “The Ant and the Grasshopper” (adapted from Font et al. [FMM⁺13a])

Font et al. present initial findings on generating and analyzing both novel and existing card games [FMM⁺13a]. They present a Card Game Description Language for expressing a wide variety of card games by formalizing the rules. They evolve playable card games using grammar-guided genetic programming. They assess playability and balance by measuring the performance of several agents. In addition, they filter games with too many stages and rules [FMM⁺13b]. Figure 2.39 shows an example. Other examples include poker variant Texas hold ‘em, Blackjack and UNO.

publication	query	publication type	research category	note
[FMM ⁺ 13a]	47w	conference paper	proposal of solution	
[FMM ⁺ 13b]	590w	conference paper	validation research	

```

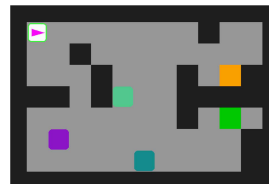
BasicGame
LevelMapping
G > goal
+ > key
A > nokey
l > monster
SpriteSet
goal > Immovable color=GREEN
key > Immovable color=ORANGE
sword > Flicker limit=5 singleton=True
movable >
avatar > ShootAvatar stype=sword
nokey >
withkey > color=ORANGE
monster > RandomNPC cooldown=4
InteractionSet
movable wall > stepBack
nokey goal > stepBack
goal withkey > killSprite
monster sword > killSprite scoreChange=1
avatar monster > killSprite
key avatar > killSprite scoreChange=5
nokey key > transformTo stype=withkey
TerminationSet
SpriteCounter stype=goal win=True
SpriteCounter stype=avatar win=False

```

```

wwwwwwwwwwww
wA      w w
w w     w
w w w  +ww
www w1  wwwwww
w      w G w
w l     ww
w      l  ww
wwwwwwwwwwww

```



(b) Side-by-side level description (left) and rendering (right) of a Legend of Zelda-like game level, where the hero Link (A) confined by dungeon walls (w) must find a key (+) and the exit goal (G) while killing or avoiding monsters (1).

```

def killIfFromAbove(s, p, game):
    """Kills the sprite, if the other one is higher
    and moving down."""
    if (s.lastrect.top > p.lastrect.top and p.rect.
        top > p.lastrect.top)
        killSprite(s, p, game)

```

(c) Extension for Super Mario that restricts the *killSprite* procedure to downward movement

(a) VGDL description for a Legend of Zelda-like game

Figure 2.40: Video Game Description Language (adapted from in Schaul [Sch14a])

Ebner et al. propose a Video Game Description Language (VGDL) as a means for general video game playing that expresses a wide range of classic 2D game types in a high-level, concise and human readable manner [ELL⁺13], e.g., approximations of Pong, Boulder-Dash, Tank Wars, Super Mario, Lunar Lander and Pac-Man.

Schaul proposes PyVGDL, a Python implementation of VGDL and a game simulation environment for conducting research [Sch13; Sch14a]. Figure 2.40 shows an example description (a). This description maps each game object to an ASCII character (LevelMapping) used in level descriptions (b). Next, it specifies their behaviors (SpriteSet) by using predefined functions (c). Finally, it defines the effects of possible collisions (InteractionSet) and win conditions (TerminationSet). PyVGDL is used in The General Video Game AI Competition¹ for benchmarking algorithms for planning, level generation and learning. PyVGDL is available under the 3-clause BSD license².

¹<http://www.gvgai.net> (visited April 5th 2019) – also maintains a list of related publications

²<https://github.com/schaul/py-vgdl> (visited April 5th 2019)

publication	query	publication type	research category	note
[ELL ⁺ 13]	31w	book chapter	proposal of solution	VGDL
[Sch13]	-w	conference paper	proposal of solution	PyVGDL
[BT14]	56n	extended abstract	proposal of solution	VGDL
[Sch14a]	236w	journal article	proposal of solution	PyVGDL

Bell and Goadrich describe RECYCLE, a card game description language and its implementation CARDSTOCK, which can automatically playtest card games with algorithms that represent intelligent players. As a demonstration, they playtest variants of the games Agram, Pairs and War.

publication	query	publication type	research category	note
[BG16]	-w	journal article	proposal of solution	

2.6.12 Script and Programming

Here we describe a programming language perspective on game development. For creating a programming language, one usually constructs a *grammar* using the Extended Backus-Naur form (EBNF) or a similar notation, for parsing the textual source code of programs [ASU86]. Here, source code (or textual model) refers to end-user notations called *concrete syntax*. The result of parsing is a parse tree that is often represented using suitable intermediate representations referred to as *abstract syntax*, which usually omits white-space and comment. In the resulting tree, references between definitions and uses must be resolved. This processes of *reference resolution* yields a graph that forms an input to analyzers, code generators and interpreters that further transform or run programs.

Game programming usually follows a bottom-up approach that composes game systems from reusable parts. Specialized software libraries called *game engines* offer developers reusable Application Programming Interfaces (APIs) for solving challenge in 3D modeling, physics, directional audio or networking. Many commercial game engines have been developed as reusable platforms for game development, e.g., Unity 3D, Unreal, CryoEngine, etc.¹¹.

¹¹These are not identified by this study.

Table 2.22: Generic script and programming languages applied to game development

Nr.	Language	Application domain
62	Squeak	dialect of Smalltalk applied in teaching game design
77	Python	programming language, applied for scripting in games
78	Lua	programming language, scripting in games
79	vision on game programming	reflections on features of game programming languages and Haskell
80	DisCo	language and system for creating, executing and analyzing formal specifications
81	Design by contract	generic approach that uses pre- and post-conditions for checking function calls

Table 2.23: Domain-specific languages for game development

Nr.	Language	Application domain
82	GameMaker	2D game development with C-like scripting
83	Extensible Graphical Game Generator	game programming
84	Mogemoge	2D games
85	Scalable Games Language	scripting for games
86	Network Scripting Language	scripting and networking
87	4Blocks DSL	DSL for Tetris games (Haskell-based)
88	Casanova	game programming (integrated game engine)
89	Sound scene DSL	sound scene DSL (Haskell-based)
90	MUDDLE (historical account)	multi-user dungeon games (MUDs)
91	PuzzleScript	puzzle games

One approach separates game engines from game-specific source code by using a generic interpreters as-is, e.g. Python (Language 77) or Lua (Language 78). Table 2.22 also shows other examples.

Another approach leverages general purpose languages by adding domain-specific language extensions, e.g., as an *internal DSL* that reuses the syntax and semantics of the host language. For instance, Scalable Games Language (Language 85) extends SQL and 4Blocks (Language 87) and Sound Scene DSLs (Language 89) are Haskell-based.

In contrast, purpose-built languages, also known as *external DSLs*, have separate parsers, compilers and/or interpreters. PuzzleScript (Language 91) and Micro-Machinations (Language 27) are examples of external DSLs. Table 2.23 shows examples of DSLs for game development. For conciseness, we do not list all textual DSLs that we already describe in other sections. Notably not identified is DarkBasic [HS07].

Python is an interpreted general-purpose programming language originally developed by van Rossum. Its language features include modules, exceptions, dynamic typing, data types and classes. Examples of languages built on top of Python include versions of Alice (Language 59) and PyVGD (Language 75). The Python Package Index¹ hosts many reusable modules for various purposes, including game development. The current version (v3) of its portable and embeddable C implementation is released under the Python Software Foundation License².

¹<https://pypi.org> (visited July 14th 2019)

²<https://www.python.org> (visited July 14th 2019)

publication	query	publication type	research category	note
[Daw02]	-w	article	experience report	
[Var03]	10w	book	practice	
[Jon05]	227w	short paper / tutorial	practice	
[Phi14]	377n	book	practice	

Lua is an interpreted general-purpose programming language developed by Ierusalim-schy et al. [IDC05; IdFC07]. Originally intended for the petrochemical industry, Lua is now also used for scripting in Games. Its APIs enable embedding in C and its functional and dynamic features support constructing embedded DSLs.

Wasty et al. describe ContextLua, a context-oriented programming extension to Lua that is suitable for implementing dynamic behavioral variations in computer games [WSA⁺10]. Layers modify the behaviour of function calls as shown in Figure 2.41. The proceed method calls the next appropriate method in the current layer composition. The with and without statements are used to activate and deactivate layers respectively.

The sources of Lua¹ and ContextLua² are available online under the MIT license.

¹<https://www.lua.org> (visited March 21st 2019)

²<https://www.hpi.uni-potsdam.de/hirschfeld/trac/Cop/wiki/ContextLua> (visited May 1st 2019)

publication	query	publication type	research category	note
[IDC05]	551w	journal paper	proposal of solution	Lua
[IdFC07]	834w	conference paper	experience report	Lua
[WSA ⁺ 10]	g&	workshop paper	proposal of solution	ContextLua
[KRvR12]	54n	workshop paper	proposal of solution	Lua

```
function Monster:getSensingDistance()
  return 10
end
```

```
function Monster:Night_getSensingDistance()
  return proceed() - 5
end
```

(a) Monster behaviour variation

```
with(Night, function()
  print(monster:getSensingDistance())
end)
```

```
with({Night, Sneak}, function()
  print(monster:getSensingDistance())
end)
```

(b) Night Layer Activation

Figure 2.41: ContextLua code snippets (adapted from Wasty et al. [WSA⁺10])

Language

79

Vision on Game Programming

generic / framework / practice

In an invited talk on “The Next Mainstream Game Programming Language”, Sweeney (Epic Games) shares a perspective on language constructs for game development with focus on performance, modularity, reliability and concurrency [Sweo6]. He argues for productivity, modular libraries, debugging facilities, and reflects briefly on perceived strengths (unions, maybe) and weaknesses of Haskell.

publication	query	publication type	research category	note
[Sweo6]	3w	abstract and slide deck	opinion paper	

Language

80

DisCo

generic / tool / practice

Nummenmaa et al. propose simulating gameplay on a logical event level in the early states of the game development process [NKH09]. As a design tool, simulating long-term dynamics of abstract and simplified game prototypes can reveal problems early on. They use DisCo¹, a software package for creating and executing formal specifications, which has been extended to for the analysis and simulation of games. The DisCo language has an action-oriented execution model based on temporal logic. A simulation model of a game called Tower Bloxx demonstrates the approach.

¹<http://disco.cs.tut.fi> (visited August 15th 2019)

publication	query	publication type	research category	note
[Numo8]	298w	Master’s thesis	proposal of solution	
[NKH09]	26w	conference paper	vision paper	simulate prototypes
[NBM09]	662w	workshop paper	validation research	analyze changes

```

move is choose
pieces are Rock and Paper and Scissors
board starts [[Rock, Paper, Scissors]]
turns synchronize
Beat means player(Rock) && opponent(Scissors)
      or player(Scissors) && opponent(Paper)
      or player(Paper) && opponent(Rock)
goal is Beat # success!
score increments
3x1 grid

```

(a) Rock Paper Scissors

```

turn is player place piece
3x3 grid
pieces are X and O
turns alternate
players are X and O
goal is &Three_in_a_row
Three_in_a_row means
      (x-1,y) && (x,y) && (x+1,y)
      or (x,y-1) && (x,y) && (x,y+1)
      or (x-1,y-1) && (x,y) && (x+1,y+1)
      or (x-1,y+1) && (x,y) && (x+1,y-1)
board starts empty

```

(b) Tic Tac Toe

Figure 2.42: EGG example specifications (adapted from Orwant [Orwooa])

Language

81

Design by Contract

generic / tool / practice

Paige et al. present qualitative and empirical results showing that light-weight formal methods are effective for developing a networked, multiplayer game. Their results, obtained in a pilot study on applying the *Design-by-Contract* approach, show that contracts (pre- and post-conditions) indeed help in diagnosing defects.

publication	query	publication type	research category	note
[PABo6]	186w	journal article	evaluation research	

Language

82

GameMaker

generic / engine / practice

GameMaker is a commercial graphical game creation tool with a drag and drop interface by YOYO Games¹, which is described by Overmars [Oveo4a; Oveo4b]. The Game Maker Language (GML) is a C-like language for scripting.

¹<https://www.yoyogames.com/gamemaker> (visited November 19th 2018)

publication	query	publication type	research category	note
[Oveo4a]	14w	journal article	experience report	
[Oveo4b]	358w	journal article	experience report	

Orwant describes the Extensible Graphical Game Generator (EGGG), a system for game programming aimed at productivity and reuse. EGGG offers a textual formalism, and leverages an ontology that codifies similarities between traditional games such as board- and card games. Examples include, Rock Paper Scissors, Tic Tac Toe, Poker, Crossword, Deducto, Tetris and Chess [Orwooa]. Figure 2.42 shows the two simplest examples.

publication	query	publication type	research category	note
[Orwoob]	-w	journal article	proposal of solution	
[Orwooa]	-w	PhD thesis	proposal of solution	

Nishimori and Kuno address the lack support in game script languages for interactions among multiple concurrent activities in a state-dependent manner. They propose a novel event handling framework called *join token* as a supplementary mechanism to conventional object orientation, in which the states of game characters can be expressed as tokens and interactions as handlers. The language Mogemoge implements join tokens, and is used for creating two simple 2D games, Balloon (defending against bombs) and Descender (climbing down a wall). Its Java sources are available online¹. Copyright is retained by Nishimori.

¹<http://www.nisnis.jp/mogemoge/> (visited January 10th 2019)

publication	query	publication type	research category	note
[NK06]	76w	conference paper	proposal of solution	
[NK12b]	-w	conference paper	proposal of solution	
[NK12a]	-w	journal article	proposal of solution	

White et al. propose the Scalable Game Language (SGL), a declarative language that extends SQL for improving the quality of games, notably scalability. They describe two patterns: the *state-effect-pattern*, which is similar to the well-known game loop, and the *restricted iteration pattern*, which prevents out-of-bounds exceptions.

publication	query	publication type	research category	note
[WDK ⁺ 07]	-w	conference paper	proposal of solution	
[WSG ⁺ 08]	-w	conference paper	proposal of solution	
[WKG ⁺ 08]	544w	journal article	philosophical paper	
[WKG ⁺ 09]	457w	journal article	philosophical paper	reprint

Language

86

Network Scripting Language

generic / tool / practice

Russell et al. present a novel DSL called Network Scripting Language (NSL) for programming bandwidth-efficient online games. Developers can use NSL to create the game logic of deterministic, concurrent and distributed games. The system automatically maintains consistency between the clients and the sever that run the scripts. NSL has a Java-like syntax. In NSL, objects are lightweight processes that execute a game loop. Scripts contain specialized statements for sending and receiving messages and handling synchronization. PointWorld is a simulation that demonstrates the approach.

publication	query	publication type	research category	note
[RDS08]	33n	workshop paper	proposal of solution	

Language

87

Haskell – 4Blocks DSL

application-specific / engine / practice

Calleja and Pace propose scripting game-specific AI with embedded DSLs in Haskell. They demonstrate their approach with the 4Blocks DSL for Tetris.

publication	query	publication type	research category	note
[CP09]	gd	workshop paper	proposal of solution	
[CP10]	52n	workshop paper	proposal of solution	

Language

88

Casanova

generic / engine / practice

Maggiore et al. describe Casanova [MBO11; MSO⁺12a; MSO⁺12b; MSO⁺12c]., a language-extension to F# for engineering games aimed at consistency and performance. Rules inside entity type declarations determine how entities change during a tick of the game loop. Additionally, imperative processes are supported through coroutines integrated with the rules. Game scripts consist of the main script and pairs of event detection- and event response scripts. Abbadi et al. [AdGC⁺15; Abb17; ADC⁺15] and di Giacomo et al. [dGia14; dGAC⁺17a; dGAC⁺16; dGAC⁺17b] continue work on Casanova, in the context

of optimized compilation, meta-programming and high performance encapsulation. Casanova 2 is available for Unity or stand-alone under the MIT license on GitHub¹. The distribution includes an asteroid game and several tutorials.

¹<https://github.com/vs-team/casanova-mk2> (visited November 19th 2018)

publication	query	publication type	research category	note
[MBO11]	47n	workshop paper	proposal of solution	Casanova
[MSO ⁺ 12a]	34w	conference paper	validation research	Casanova
[MSO ⁺ 12b]	95w	conference paper	proposal of solution	Casanova
[MSO ⁺ 12c]	308w	conference paper	validation research	Casanova
[dGia14]	177n	Master's thesis	validation research	Casanova
[AdGC ⁺ 15]	60n	conference paper	proposal of solution	Casanova II
[ADC ⁺ 15]	26n	conference paper	proposal of solution	Casanova II
[dGAC ⁺ 16]	53n	conference paper	proposal of solution	Metacasanova
[Abb17]	195n	PhD thesis	validation research	Casanova II
[dGAC ⁺ 17b]	101n	conference paper	validation research	Metacasanova
[dGAC ⁺ 17a]	86n	journal article	validation research	Casanova II

Language

89

Haskell – Sound Specification DSL

generic / tool / practice

Bäärnhielm et al. describe a sound specification DSL intended for designing immersive and interactive experiences for a Nordic technology-supported Live Action Role Playing (LARP) game. They demonstrate features of the Haskell-based DSL, by expressing *sound scenes* of a Nordic LARP called The Monitor Celestra. In this game, which takes place on a space ship, participants receive roles such as crew, passengers and refugees. Within a framework of plots, storylines and clues, supported by sound, they act out a story where choices determine the outcome.

publication	query	publication type	research category	note
[BSV14]	196n	journal article	experience report	

Language

90

MUDDLE

application-specific / tool / practice

Bartle gives a historical account of the creation of MUDDLE, a language for the first Mutli-User Dungeon (MUD) game, which gave the genre its name, also known as Massive Multiuser Online (MMO) games.

```

title Simple Block Pushing Game
author Stephen Lavelle
homepage www.puzzlescript.net
=====
OBJECTS
=====
Background
lightgreen green .000.
11111 .111.
01111 22222
11101 .333.
11111 .3.3.
10111

Target
darkblue
.....
.000.
.0.0.
.000.
.....

Wall
brown darkbrown
00010
11111
01000
11111
00010

=====
LEGEND
=====
. = Background
# = Wall
P = Player
* = Crate

@ = Crate and
Target
0 = Target

=====
SOUNDS
=====
Crate MOVE
36772507

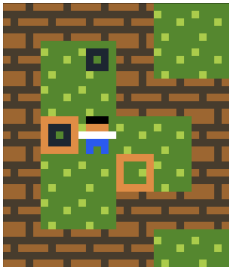
=====
COLLISIONLAYERS
=====
Background
Target
Player, Wall,
Crate

=====
RULES
=====
[> Player |
Crate] ->
[> Player
|> Crate]

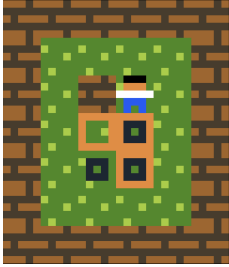
=====
WINCONDITIONS
=====
All Target on
Crate

=====
LEVELS
=====
#####
#.0#..
#.#.#
#@P.#
#.#.#
#.#.#
#####
#####
#...#
#.#P.#
#.*@.#
#.0@.#
#...#
#####

```



(b) First level



(c) Second level

(a) Source code with dynamic dynamic syntax highlighting of sprites

Figure 2.43: PuzzleScript tutorial: “Simple Block Pushing Game” (from puzzlescript.net)

publication	query	publication type	research category	note
[Bar16]	303n	book chapter	historical account	

PuzzleScript is an online textual puzzle game design language and interpreter¹ created by Stephen Lavelle using JavaScript and HTML5/CSS. PuzzleScript game levels are tile maps populated by objects (named sprites of 5x5 pixels) that can move and collide, and whose game logic is defined as a set of rewrite rules. Figure 2.43 shows an example where the objective is to push crates into place. When the player collides with a crate, both directionally move if possible. The source are released under the MIT license². Lim and Harell present an approach for automated evaluation and generation of PuzzleScript videogames and propose two heuristics [LH14]. The first, level state heuristics, determines how close the state of given level is to completion during gameplay. The second, ruleset heuristics, evaluates rules defining a videogame’s mechanics and assesses them for playability. Osborn et al. apply Playspecs (Language 57).

Table 2.24: Generic modeling languages applied to game design and development

Nr.	Language	Application domain
92	UML – Metamodeling	generic formalism for meta-modeling, e.g., applied to 2D platform games
93	UML – Class and State Diagrams	generic formalism for object-oriented analysis and design applied in model-driven game development
94	Statecharts	variants of a formalism that describes behaviors as state machines, applied to Game AI and dialogue in games
95	Feature Models	visual variability modeling formalism applied to managing game (and game engine) variability

¹<https://www.puzzlescript.net> (visited November 24th 2018)

²<https://github.com/increpare/PuzzleScript> (visited November 24th 2018)

publication	query	publication type	research category	note
[LH14]	110W	conference paper	validation research	

2.6.13 Model-Driven Engineering

Model-driven game development revolves around abstract *models* that describe game content or work processes, often using visual diagrammatic representations. Modeling languages are based on the principles, techniques and tools from an area called Model-Driven Engineering (MDE). These models are step-by-step translated, transformed and combined into a resulting model or source code that integrates with the game software. We identify applications of generic modeling languages in Table 2.24, and Domain-Specific Modeling Languages in Table 2.25.

Metamodeling represents the abstract syntax of models as a graph of Unified Modeling Language (UML) classes and references between them. Because metamodeling is often based on the Eclipse Modeling Framework (EMF) and Ecore (the EMF model engine), it enjoys the advantage of generic reusable tools for model transformation, analysis and productivity, e.g., for adding a textual concrete syntax (Xtext, EMFText), or graphical ones (e.g., using GMF, Graphiti) [PKP13].

Table 2.25: Domain-specific modeling languages for game development

Nr.	Language	Application domain
96	SharpLudus	RPG games, mobile touch-based games, 2D arcade games
97	Eberos GML2D	2D Games
98	FLEXIBLERULES	digital board game creation and customization
99	PhyDSL	2D mobile physics-based games
100	Pong Designer	Pong-like games
101	Board Game DSL	board games
102	RougeGame Language	visibility Rogue-like dungeon maps
103	Reactive AI Language	behaviors in adventure games

Language

92

UML – Metamodeling

generic / engine / practice

Montero Reyno and Carsí Cubel address the increasing complexity of game development by applying model-driven engineering and UML to the development of 2D platform games [MCo8]. They aim to enhance productivity in terms of quality, time and cost [MCo9a]. A prototype tool uses Platform Independent Models (PIMs) for defining the structure and behaviour of the game, and Platform Specific Model (PSM) for mapping game actions to hardware control devices for player interaction [MCo8] and UML metamodels for social context, structure diagram and rule set [MCo9a]. The tool generates C++ prototype games for a middleware called HAAF Game Engine. These are then iteratively play tested and manually completed and fine-tuned.

publication	query	publication type	research category	note
[MCo8]	58w	conference paper	proposal of solution	
[MCo9a]	27w	conference paper	proposal of solution	
[MCo9b]	72w	journal article	proposal of solution	

Language

93

UML – Class and State Diagrams

generic / tool / practice

Tang and Hanneghan investigate how to define a Domain-Specific Modeling Language for serious game design. They perform an analysis and propose a modeling framework that uses UML class diagrams and state diagrams for modeling user interactions and in-game components. They extend state diagrams with UI modeling elements [THo8]. In later work, they examine the state of the art in model-driven game development from a game-based learning perspective [TH11]. We compare this related work in Section 2.8. Tang et al. propose a Game Technology Model for modeling serious games [THC13].

publication	query	publication type	research category	note
[THo8]	1w	conference paper	proposal of solution	
[TH11]	13n	journal article	survey	
[THC13]	-w	journal article	proposal of solution	

Language

94

Statecharts

generic / tool / practice

Statecharts are visual diagrams for modeling behavior. Several several variants of the notation exist [CD07]. We identify two used in model-driven game development. Kienzle et al. propose visual modeling game AI of NPCs in a Rhapsody Statechart variant to ease the difficulty of programming consistent, modular and reusable game AI. They demonstrate the approach in an AI competition of EA Games called Tank Wars. Brusk and Lager propose applying State Chart XML (SCXML) to the design and implementation of games, in particular games featuring natural language dialogue [BL07]. Brusk investigates how statecharts can be used for describing social interaction and dialogue behavior for believable characters in game worlds [Bru08]. Various tools and libraries for Statecharts have since become available. The latest recommendation for v1.0 of SCXML as w3c standard dates from September 1st 2015¹.

¹<https://www.w3.org/TR/scxml/> (visited March 27th 2019)

publication	query	publication type	research category	note
[DKV06]	gd1	conference paper	proposal of solution	RHAPSODY Sc.
[KDV07]	-w	conference paper	proposal of solution	RHAPSODY Sc.
[BL07]	180w	conference paper	proposal of solution	SCXML
[Bru08]	569w	conference paper	proposal of solution	SCXML

Language

95

Feature Models

generic / tool / practice

Feature Models (FMs) are a visual notation for describing the variability of product features, e.g., in software product lines for the automotive or aerospace industries. Sarinho et al. propose an approach that entails using FMs for representing and manipulating the variability of game features, and an environment that integrates and adapts features of available game engines, e.g., for configuring game logic, rules and goals.

publication	query	publication type	research category	note
[SA09]	6n	conference paper	proposal of solution	
[SAA12]	language	workshop paper	proposal of solution	

Furtado et al. study how game development can be improved using visual domain-specific modeling languages, software product lines, software factories, generators and semantic validators aimed at software reuse and productivity [FSR⁺11; FSo6b]. SharpLudus is a software factory intended to empower game designers in creating 2D adventure video games [FSo6b], but over the years targets also included RPG games, mobile touch-based games and 2D arcade games [FSR⁺11]. For instance, ArcadEx is a factory for 2D arcade games for the PC based on Microsoft XNA and the FlatRedBall engine [FSR⁺11]. DSLs are provided for describing games, mapping input of Xbox 360 buttons into XNA Keyboard keys, and modeling variability using feature models. Game descriptions are visual models of introduction screens and rooms with transitions between them (arrows), sound, entities, input handling, triggers, events and actions of NPCs. The project web site¹ contains videos and demos of Ultimate Berzerk, Stellar Quest and Tank Brigade, and links a to a distribution².

¹<http://cin.ufpe.br/~sharpludus/> (visited march 27th 2019)

²<https://archive.codeplex.com/?p=sharpludus> (visited March 27th 2019)

publication	query	publication type	research category	note
[FSo6b]	1n	workshop paper	proposal of solution	SharpLudus
[Furo6]	3n	Master's thesis	validation research	SharpLudus
[FSo6a]	87n	conference paper	tutorial	MS DSL tools
[FSRo7]	25n	journal article	validation research	SharpLudus
[FSR11]	16n	workshop paper	experience report	SharpLudus
[FSR ⁺ 11]	25w	journal article	proposal of solution	ArcadEx
[Fur12]	93n	PhD thesis	validation research	all the above

Hernandez and Ortega wish to learn how the game industry can profit from model-driven development approaches [HO10]. Eberos Game Modeling Language 2D (GML2D) is a graphical DSL that aims for expressiveness, simplicity, platform independence and library independence. Figure 2.44 shows the UI and a Pong model.

publication	query	publication type	research category	note
[HO10]	2n	workshop paper	proposal of solution	

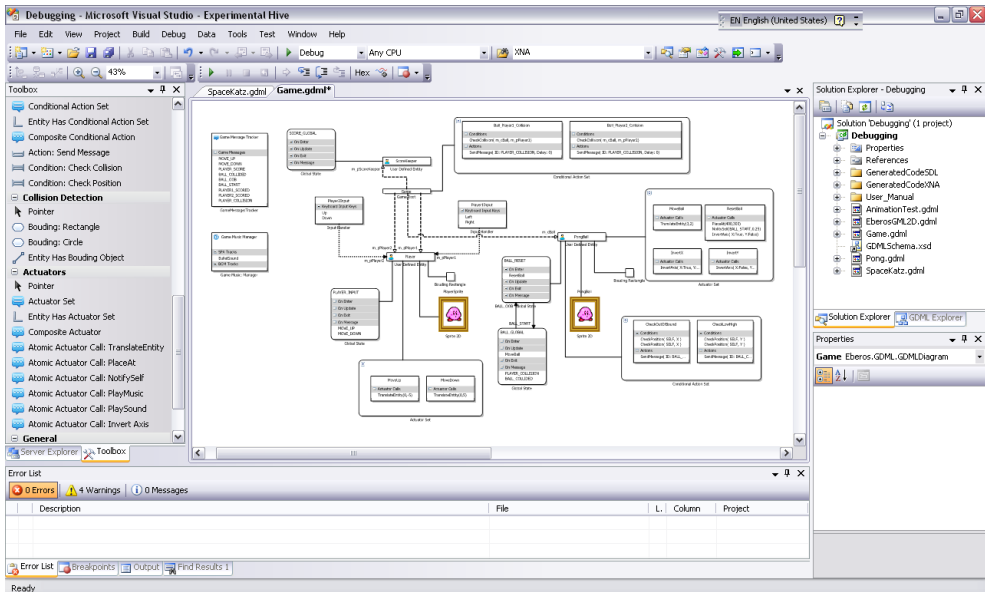


Figure 2.44: Eberos GML2D showing a model of Pong (appears in Hernandez and Ortega [HO10])

Frapolli et al. present FLEXIBLE RULES, a framework for implementing all aspects of digital board games aimed at customization, adaptability and end-user programming [FBM⁺10a; FMH10; FBM⁺10b]. Its toolkit offers a *logic editor* for visually defining a directed graph of game entities (nodes), properties and relationships (edges). The *code editor* for a Lisp-like DSL enables programming games as the behavior specification of those entities. Aside from statements for control flow and messaging for communicating between entities, it includes rules that are defined as *laus* and *side effects*, similar to point-cuts and advice in aspect oriented programming. FLEXIBLE RULES is available under GPL v3¹. Examples include Tic Tac Toe, Go and Snakes and Ladders.

¹<http://flexiblerules.fulviofrapolli.net> (visited April 10th 2019)

publication	query	publication type	research category	note
[FBM ⁺ 10a]	21n	conference paper	validation research	
[FBM ⁺ 10b]	57n	journal article	proposal of solution	
[FMH10]	78n	conference paper	validation research	

Guana and Stroulia propose PhyDSL, a textual DSL for rapidly prototyping mobile 2D physics-based games, and a model-driven environment that generates code for Android devices. PhyDSL has features for defining actors, environment and layout, activities and scoring rules. PhyDSL-2 is implemented in Xtext and available on GitHub¹.

¹<https://guana.github.io/phydsl> (visited September 1st 2019)

publication	query	publication type	research category	note
[GS14]	24n	conference paper	proposal of solution	PhyDSL
[GSN15]	29n	workshop paper	experience report	PhyDSL-2
[Gua17]	209n	PhD thesis	validation research	PhyDSL-2

Mayer and Kuncak aim to empower end-users and to simplify modifying running programs [MK13]. They explore *game programming by demonstration* and present Pong Designer, an environment for developing 2D physics games through direct manipulation of object behaviors. Internally, a game's rules are expressed in an embedded DSL implemented in Scala. These rules are updated whenever a user performs a new demonstration. Sources are available on GitHub under the Apache 2.0 license¹. Examples include Pong, Brick Breaker, Pacman and Tilting maze.

¹<https://github.com/epfl-lara/pongdesigner> (visited July 12th 2019)

publication	query	publication type	research category	note
[MK13]	285n	conference paper	proposal of solution	
[May17]	390n	PhD thesis	validation research	

Altunbay et al. describe a model-driven software development approach aimed at addressing increased complexity in video games, which is illustrated by a DSL for the board game domain based on UML meta-modeling [AÇM09].

publication	query	publication type	research category	note
[AÇM09]	383w	workshop paper	proposal of solution	

Féher and Lengyel illustrate the strength of model transformations based on graph rewriting-based in a case study on Rouge-like games, a genre of 2D dungeon crawlers. In particular, they study which cells are visible from a specific location on a 2D level map. They define the RougeGame language, a DSL defined as a meta-model expressing maps and visibility parameters. A transformation pipeline calculates the cell visibility based on rewrite rules.

publication	query	publication type	research category	note
[FL12]	189n	conference paper	proposal of solution	

Zhu describes the Reactive AI Language (RAIL), a DSL for modeling behaviors in adventure games, and a model-driven game development approach that uses meta-modeling and EMF. The approach is validated in a case study called Orc's gold, a 2D action adventure game.

publication	query	publication type	research category	note
[Zhu14]	165n	PhD Thesis	validation research	

2.6.14 Metaprogramming

The metaprogramming perspective considers game development as an application domain for generic language technology. Metaprogramming refers to techniques, tools and approaches for creating metaprograms that read and transform the source code of other programs, e.g., compilers, interpreters and integrated development environments. Applying these techniques to game development promises to raise productivity, improve quality and reduce maintenance costs.

Constructing languages and tools by means of metaprogramming requires appropriate meta-tooling. *Language work benches* are tools that provide high-level mechanisms for the implementation of software languages. Erdweg et al. describe the state of the art in language workbenches [EVV⁺13]. Examples of metaprogramming languages and language work benches include Epsilon, Gemoc Studio, Meta-Edit+, MPS, Racket, Rascal, Spoofox and Xtext. Several authors illustrate metaprogramming techniques and apply approaches to example languages. Table 2.26 shows examples of language of games and play created by means of language work benches.

Table 2.26: Applications of metaprogramming languages and language work benches

Nr.	Language	Metaprogramming language or work bench	URL
27	Micro-Machinations	Rascal	https://www.rascal-mpl.org
66	GAML	Xtext	https://www.eclipse.org/Xtext
96	SharpLudus	Microsoft DSL tools	https://visualstudio.microsoft.com
99	PhyDSL	Xtext	https://www.eclipse.org/Xtext
104	Whimsy	C++	(various versions exist)
105	Level editors	DiaMeta	http://www2.cs.unibw.de/tools/DiaGen
107	Ficticious	Ginger	(not found)
106	Text adventures	Racket	https://racket-lang.org
108	Dialog Script	Xtext	https://www.eclipse.org/Xtext

The strength of this perspective is the application of state-of-the-art in language engineering and its weakness is that, with some exceptions, many illustrations remain toy examples that are never extensively validated.

Language

104

Whimsy

application-specific / engine / educative

West discusses potential uses for DSLs in games and demonstrates Whimsy, a DSL for creating whimsical flowery shapes inspired by the works of Rodney Alan Greenblatt [Wes07]. Figure 2.45 shows how SuperEgg, Inner and Petal primitives can be used for generating an image similar to a painting. Whimsy is an external DSL implemented in C++ that requires the Windows SDK and DirectX 9. Its sources are available under the MIT license from GDCVault¹.

¹https://twvideo01.ubm-us.net/o1/vault/GD_Mag_Archives/aug07.zip (visited May 9th 2019)

publication	query	publication type	research category	note
[Wes07]	gd	magazine article	philosophical paper	practice

Language

105

Level Editors in DiaMeta

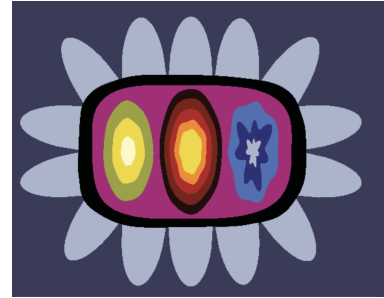
genre-specific / engine / educative

Maier and Volk report teaching experiences on applying DiaMeta, an EMF-based language workbench for creating visual domain-specific languages, e.g., for level editors for classic games such as PacMan and the platform game Pingus [MV08]. Insights include that meta-modeling has a steep learning curve and that the proposed approach speeds-up game prototyping.

```

superegg 0.15,0.10,3.5 at .3,.7 size 1.2 black distort .01
petals 14 0.05 size 1.8 petalblue
inner .88,.01 tvpurple
superegg .1,.2,2 at .20,.7 size .4 distort .01 tvlime
inner .65,.01 tvyellow
inner .45,.01 tvlightyellow
superegg .1,.2,2 at .3,.7 size .5 distort .01 tvblack
inner .85 tvbrown
inner .80 tvred
inner .75 distort .03 tvorange
inner .70 tvyellow
superegg .1,.2,2 at .4,.7 size .4 distort .05 tvblue
inner .6 distort .2 tvdarkblue
inner .4 petalblue

```



(a) Source code

(b) Generated image

Figure 2.45: Whimsy example replicating the style of a painting (adapted from West [Weso7])

publication	query	publication type	research category	note
[MV08]	4w	conference paper	experience report	

Language

106

Text Adventures in Racket

genre-specific / tool / educative

Flatt demonstrates in a tutorial-like manner how to create languages in Racket. He describes an illustrative text-adventure DSL for interactive fiction [Fla11; Fla12]. The Racket metaprogramming language is distributed under the GNU LFPL¹.

¹<https://racket-lang.org> (visited May 9th 2019)

publication	query	publication type	research category	note
[Fla11]	181n	journal article	philosophical paper	
[Fla12]	181n	journal article	philosophical paper	reprint

Language

107

Ficticious

genre-specific / tool / practice

Palmer reports experiences on developing a set of micro-languages (DSLs) called Ficticious for describing narrative worlds in Interactive Fiction [Pal10], including rich text markup, virtual world design and character interaction. The approach demonstrates how Ginger, a language with support for literate programming through so-called G-expressions, can be used to separate concerns in DSLs. Figure 2.46 shows code snippets for describing (a) people places and things; (b) page layout; (c) rich text and grammar; and (d) dialogue.

```

object Hook extends FixedItem
  set name "hook"
  set aliases ("hook" "peg")
  set adjective ("small" "brass")
  set location 'CloakRoom
  action examine
    :story It's just a small brass hook,
    if (isIn cloak self)
      :story with a cloak hanging on it.
    else
      :story screwed to the wall.

```

(a) People, Places and Things

```

panel GamePageLakeShore extends: GamePage
  property Image panel1
  property Image panel2
  init
    setText 10 445 580 250
    setImage panel1 "imgs/charonGone.png"
    setImage panel2 "imgs/charonWaits.png"
  draw
    if (eq? 'Boatman.location 'LakeShore)
      drawImage panel2 7 7
    else:
      drawImage panel1 7 7

```

(b) Page Layout

```

:story
  The sign reads "No_Loitering" but
  ironically a cowboy whittles a small
  piece of wood *right beside* the sign.

```

(c) Rich Text and Grammar

```

conversation on OldMine
  oldMine "Ask_about_the_old_mine."
  :dialog
    Sam: I keep seeing an old donkey
    at the mine.
  donkey "Ask_about_the_donkey." => oldMine
  :dialog
    Sam: The donkey comes and goes.

```

(d) Dialogue

Figure 2.46: Fictitious microlanguages code snippets (adapted from Palmer [Pal10])

publication	query	publication type	research category	note
[Pal10]	62n	conference paper	experience report	

Language

108

Dialog Script in Xtext

genre-specific / tool / educative

In a textbook chapter on engineering DSLs for games, Walter proposes DSLs for bridging the gap between game design and implementation [Wal14]. He describes a textual language called Dialog Script, as an introductory example for creating interactive branching narratives [Wal14], which closely resembles an earlier version [WM11]. Dialog Script is implemented in Xtext and its prototype is available on GitHub¹ under version 2.0 of the Apache license.

¹<https://github.com/RobertWalter83/DialogScriptDSL> (visited May 9th 2019)

publication	query	publication type	research category	note
[WM11]	4n	conference paper	proposal of solution	
[Wal14]	94n	textbook chapter	proposal of solution	

2.7 CHALLENGES AND OPPORTUNITIES

Here we discuss research trends, synthesize insights and describe challenges and opportunities for future research and development. First, we discuss the results in general in Section 2.7.1. Next, we compare and analyze success factors in Section 2.7.2. Finally, we synthesize one additional perspective on languages of games and play in Section 2.7.3. Our Automated Game Design perspectives is a specific language-centric discussion on challenges and opportunities for research and development. This section answers research question RQ4.

2.7.1 *General Analysis*

Our area of interest ‘languages of games and play’ is a well-studied research topic with a growing number of publications. Figure 2.2 shows that most papers we included were published after the turn of the millennium. Around this time, roughly between 1998 and 2006, most of the interdisciplinary game publishing venues we identified also came into existence. Since 2005, there is a gradual increase in the term ‘domain-specific language’. Two factors explain the declining number of publications in this study after 2015. First, the query date limits the search results. Second, GS orders the results of the wide query to show older results first. Therefore, it is likely we missed newer results that could have been included.

Our analysis of the citation graph, shown in Figure 2.3, reveals a low cohesion between publications. We observe clusters that represent distinct technological spaces, tight-knit communities, fragmented sub-topics and diffuse areas. Therefore, authors may not find or recognize related work in a wider research context. As a result, relevant literature has gone uncited, efforts and successes have often been one-off, lessons learnt have gone overlooked, and several studies and areas have remained isolated. We view this mapping study as an opportunity to relate relevant primary sources to help frame research problems, reuse available approaches, and benefit from documented experiences. Our map serves to navigate between related perspectives and technological spaces. As time progresses, the map can be extended and reshaped for charting new research trajectories that continue to explore the limits of formalism.

2.7.2 *Success Factors*

Our analysis reveals a “grave yard” of dead language prototypes. Few languages ever grow to maturity. Many languages remain solution proposals that are now no longer maintained, available or in use. This lack of reuse is unfortunate, since creating one language often requires years of research, design and development. Naturally, this leads to the question why so many prototypes were abandoned. Here we discuss observations and insights about shared success factors. Table 2.27 shows examples of

Table 2.27: Examples of multi-year, multi-disciplinary work on Languages of Games and Play
(AI: Artificial Intelligence, SE: Software Engineering, Edu: Education, G: Games)

Nr.	Language	Ct.	Years	Areas	Perspectives
65	AgentSheets	6	1995–2012	SE+Edu	visual DSLs, education
39	ABL	5	2002–2008	AI+G+SE	programming, behaviors, interactive drama
7	Gameplay Design Patterns	6	2003–2013	G+Edu	pattern language
47	ScriptEase	8	2003–2013	SE+G	pattern language, visual DSL
96	SharpLudus	6	2006–2012	SE+G	model-driven engineering, visual DSL
46	<e-game>	6	2006–2012	SE+Edu	storyboards, education, visual DSL
25	BIPED and LUDOCORE	9	2008–2012	AI+G	automated game design, mechanics
26	Machinations	6	2009–2012	G	pattern language, automated game design
27	Micro-Machinations	3	2013–2015	SE+G	automated game design, mechanics, programming, visual DSL
88	Casanova	11	2011–2017	SE+G	game programming

Table 2.28: Highlighting the differences between applied and forgotten languages

	Alive languages	Dead languages
Language experts	multiple	one
Publication count	multiple	one or two
Publication areas	multiple	one
Validation	applied and validated in practice	not applied, toy examples
Availability	sources or wiki pages are released and maintained up to a point,	no source code is available
Examples	tutorials, workshops and study materials are available	not available

languages that stand out as multi-faceted research with a relatively high publication count. We explain success factors summarized in Table 2.28.

Languages of games and play represent a considerable effort and a long-term investment. Therefore, success requires a multi-year research trajectory, perhaps spanning multiple research grants and PhD projects. Publishing in different research areas helps answer different related questions and sheds light on challenges and solutions from different perspectives. Involving multiple researchers, language developers and practitioners creates co-ownership and continuity.

Traditionally, academia and the game industry have not always seen eye to eye. However, collaborating in applied research projects is essential for validating research in practice. To that end, some research departments include labs and game studios, and host in-house designers. Of course, working with innovative indie game developers or AAA studios on industrial case studies costs time and effort. The main benefit is that case studies can show case approaches and lead to better and more applicable solutions. Stakeholders can formulate common goals, agree to make research results open source, and protect intellectual property of businesses with Non-Disclosure Agreements (NDAs). As a selling point, students participating in these projects might become employees who bring expertise and help to create new and innovative game products.

Naturally, making solutions available is necessary in order to apply them. Open source software might include reusable script engines, content generators, visual tools or programming environments. In addition, for learning to apply solutions effectively, users may require Wiki pages, blogs, example materials, tutorials and workshops.

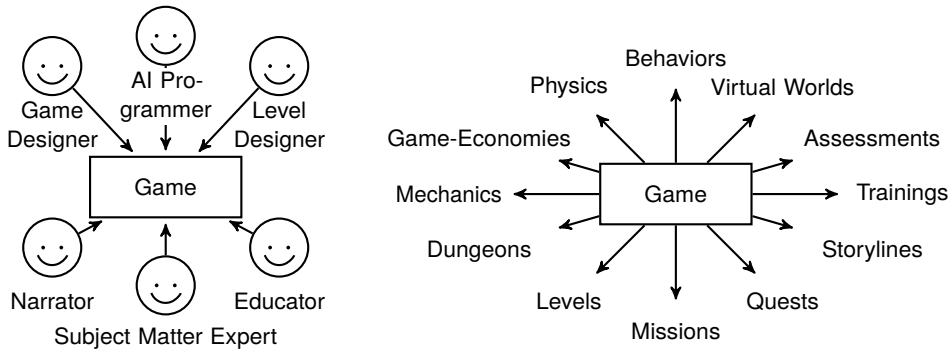
Some languages, in particular educational online languages, have thriving user communities that create significant impact. Their users apply solutions in practice, which increases the research visibility and grows a network. Naturally, these benefits come at great cost, e.g., time spent on tools, demos, maintenance and legacy support. However, when users become stakeholders invested in validation, they also assist in building data sets. Ultimately, empirical evidence is necessary for scientific research.

2.7.3 *Automated Game Design*

Here, we synthesize one final perspective on languages of games and play by relating research perspectives from the previous section to challenges and opportunities for future research and development.

Automated Game Design (AGD) aims to speed-up and improve iterative game design processes by automating design processes. Here we discuss how languages, structured notations, patterns and tools can help game design experts raise their productivity and improve the quality of games and play. We distill research challenges and relate this perspective to other perspectives on languages of games and play.

Various languages, techniques and tools have been proposed for creating, generating, analyzing and improving a game's parts. These design tools offer interactive user interfaces that support prototyping, sketching designs, automating play tests and exploring design spaces, usually in a mixed-initiative, conversational and collaborative manner. Moreover, these tools are often intended to support game design as an explorative, playful or enjoyable activity. AGD usually shifts design and implementation efforts away from manual, repetitive, time consuming and error-prone tasks. That way, designers can more efficiently create, improve and maintain growing amounts of *game content*. We define content as follows:



(a) Examples of experts that might contribute to a game's design

(b) Examples of interaction-bound content types whose evolution represents gameplay dimensions that span game design spaces

Figure 2.47: Game design experts contribute content to evolve games in different dimensions

“Game content refers to every asset of a digital game that can be separately (or together) viewed, understood, modeled, generated, recombined and improved to affect audio-visual and interactive player experiences.”

Visual content includes textures, models, sprites, imagery, objects that form composite structures such as trees, landscapes, cities and nature [STN16]. Audio content includes music, voices and effects. Procedural Content Generation (PCG) refers to generative techniques that produce and transform game content [HMV⁺13; KB17; STN16; vdLLB14]. Game engines and software libraries, reusable components for the construction of digital games, are not content [STN16].

The focus of this study is on ‘*interaction-bound content*’, content that expresses how players interact and communicate, which especially affects player experiences. Figure 2.47(a) illustrates the diverse experts that might contribute to a game's design. Figure 2.47(b) illustrates types of content (or facets) a game might have. Modifying these content facets evolves a game along multiple related axes or dimensions.

The problem is that these dimensions overlap, often in non-trivial ways, which results in a web of criss-crossing interconnected content dependencies that may differ from game to game. This makes it exceptionally difficult to align a particular interpretation of a dimension with a reusable form of content representation that separates a game design concern. As a result, improving a game's qualities along one dimension is often difficult without negatively affecting another. Therefore, design experts require *orchestrated* content generators [LYN⁺19] for composing high quality games from different kinds of inter-connected content, intricately interwoven to support meaningful experiences, in a reusable manner. Next, we relate key challenges of AGD to perspectives on languages of games and play.

Frameworks and pattern catalogues for studying games describe *what* games are. These perspectives inform automating game design by providing context, theory and structured frameworks for common vocabularies and notations.

- Game designers lack a common *vocabulary*, which hampers specification, communication and agreement among developers and designers [KvR13]. However, automating game design requires formalizing game concepts, e.g., by performing a *domain analysis* that identifies concepts, names, meanings and relationships. Useful frameworks and resources are *ontologies and typologies* that help to describe, understand and characterize games, and distinguish what makes games unique. Section 2.6.1 highlighted this perspective.
- Game designers require design tools, reusable *patterns*, and techniques that help them analyze and predict how modifications to a game's design will affect the gameplay. Section 2.6.2 highlighted pattern languages and game design patterns that describe best practices, recurring structures of content and gameplay, and represent steps towards standardization and reuse.

Other perspectives are content-centric. Related publications typically envision design experts who contribute design artifacts by means of languages and tools. We describe the following perspectives:

- Games may require integrating *subject matter knowledge*. The challenge is providing languages and tools that enable domain experts such as educators and psychologists to describe scenarios and participate in game design processes. Section 2.6.3 highlighted a perspective on applied (or serious) game design.
- Many games integrate *game mechanics*, rules that offer playful affordances and bring about interesting player experiences. Game designers require formalisms to express game mechanics, e.g., game economies or avatar physics. Additionally, they require tools for analyzing behaviors, generating rules, balancing strategies, introducing trade-offs, managing feedback-loops and automating play testing. Section 2.6.4 highlighted a perspective on game mechanics.
- Many games include generated *spaces* such as virtual worlds and game levels. Designers require tools to assist in populating these spaces with various kinds of content, e.g., to create varied and interesting game levels, dungeons, missions and quests. Section 2.6.5 gave a perspective on spaces.
- Game designs may integrate *behaviors* of in-game entities such as non-player characters that require dramatic realism or challenge. Designers and AI programmers require formalisms for expressing these behaviors. Section 2.6.6 illustrated a perspective on behavior languages with different strengths and applications.
- Games designs may integrate *stories* that allow players to progress through stages of a plot. Designers and narrators require languages and tools to expressing

these stories. Section 2.6.7 illustrated a perspective on techniques that express narratives and story plot using textual notations and graphs.

Technical views on *how* to automate game design with available techniques and approaches originate mainly from the fields of AI and games, software engineering, and education. We relate the following challenges to technical perspectives on languages of games and play:

- Raising the quality of game content requires automated iterative analyses, especially when dealing with generated content. Section 2.6.8 highlighted a perspective on metrics, which can be used to relate content and player actions to gameplay.
- Usability, understandability and ease of use are essential qualities for game design tools. Designers require appropriate visual notations and timely feedback to comprehend concepts, master language features and learn to apply user interfaces effectively. We described a perspective on educational end-user environments in Section 2.6.9.
- During a digital game's life span, designers may need to make significant changes to its design. Designers require tools that enable modifying a game's design during its entire evolution. Section 2.6.10 gave a perspective on gamification, which studies techniques for redesigning games and retrofitting game designs.
- Powerful, cutting-edge *AI techniques* are constantly being researched, developed and improved. The challenge is leveraging these for automate game design, e.g., for analysis, generation and testing. Section 2.6.11 described a perspective on a field called *general game playing*, which uses games as a test-bed for AI.
- Developing high quality games in a time-to-market manner requires *programming and maintaining* growing amounts of source code, and rapidly evolving that code towards new gameplay goals. Section 2.6.12 highlighted a perspective on DSLs, script languages and programming environments, which have been created to speed up development and improve the quality and maintainability.
- Developing *visual languages* for game design from scratch is a difficult and time-consuming endeavour. Model-driven engineering offers several reusable formalisms, techniques and approaches. We highlighted a perspective on how design can be automated by means of visual models and tools in Section 2.6.13.
- Developing and maintaining DSLs, generators and tools costs a considerable amount of time and effort. Language developers require appropriate *meta-tooling* and metaprogramming techniques to alleviate this effort, that enable rapid prototyping. Several authors advocate the use of generic language technology by demonstrating its power in illustrative examples. We highlighted this perspective in Section 2.6.14.

We describe the following additional open research challenges.

- A game’s quality is limited by the number of game design iterations. Producing high quality games more quickly requires the duration of game reducing iterations. An open challenge is leveraging *live programming* techniques for providing *live* (immediate and continuous) *feedback* on changes to a program. This may be the key to forming more accurate mental models and better predicting behavioral effects and gameplay outcomes.
- The composition of content alone does not explain how a game’s simulation is communicated to its players. To fully understand how a game works, designers might require content creation strategies that relate content representations to a set of communication strategies or Operational Logics (Language 17).

2.8 RELATED WORK

The research perspectives we have described in Section 2.6 relate work on languages of games and play. We have already mentioned several surveys on more specific topics that align with those perspectives. Here we briefly discuss more general related surveys, literature reviews and mapping studies. In addition, we give an overview of related PhD dissertations.

2.8.1 *Related Surveys*

Ampatzoglou et al. perform a systematic review on the more general topic of software engineering research for computer games [AS10]. They identify topics, research approaches and empirical research methods. Unlike this study, sources are limited to well-known publishers (IEEE, ACM, Elsevier and Springer), which excludes many games conferences. They search for the terms ‘games’ and ‘software’, and analyze publications on software engineering using a concise protocol. The result is a limited overview that gives one-sided perspective of a general field. This study instead presents a multi-faceted map of a more specific topic.

Tang and Hanneghan examine the state-of-the-art in model-driven game development from a game-based learning perspective [TH11]. Their overview, which describes model transformations and several game design languages, overlaps with the model-driven engineering perspective presented in Section 2.6.13. For instance, Rich Pictures (shown as ovals and arrows) represent the abstract progression of a story. Flow boards, similar to storyboards and flow charts document a game’s structure. In addition, they identify Statecharts (Language 94), Petri Nets (Language 23), UML Class Diagrams and Metamodeling (Language 92), Alice (Language 59) and Scratch (Language 63). Finally, they compare several game engines.

Almeida and da Silva survey game design methods and tools [AdS13b], and synthesize requirements from 32 selected publications [AdS13a]. They present a map of design frameworks and visual modeling formalisms [AdS13b], which overlaps

with our perspectives on Ontologies (Section 2.6.1), Pattern Languages (Section 2.6.2) and Game Mechanics (Section 2.6.4). In particular, they discuss FADTs (Language 5), MDA (Language 8), Gameplay Design Patterns (Language 7), UML (Language 92), Petri Nets (Language 23), Machinations (Language 26) and LUDOCORE (Language 25). They describe requirements for software tools that integrate with industry approaches and help aid game design. In particular, they propose constructing pattern catalogues (databases) that let designers: 1) relate design concepts, games and genres from MDA perspectives using pattern languages; 2) analyze concepts in relation to market, critics and player data; and 3) build, manage, moderate, maintain and evolve the concept collection together as a collaborative effort. Additionally, they propose tailor-made visual design languages that let designers: 1) model and assemble games from smaller concepts; and 2) reuse existing formalisms and proven technology.

Several vision papers align with this study. Walter and Masuch discuss how to integrate DSLs into the game development process [WM11]. Mehm et al. take an authoring tools perspective when discussing research trends [MRG⁺12].

2.8.2 *Related Dissertations*

Several dissertations also describe one or more languages of games and play, usually as part of a literature study chapter. These chapters have a more narrow focus than this study, and few are systematic studies. Table 2.29 shows a selection of PhD dissertations. Authors of dissertations approach the topic from various angles. For instance, Maggiore includes libraries and systems when discussing languages [Mag12]. Neil discusses several existing game design tools, mainly academic prototypes, and evaluates their application in supporting practical game design activities [Nei15]. We refer to our citation data for additional PhD, Master and Bachelor dissertations.

2.9 THREATS TO VALIDITY

Systematic mapping studies are intended to create an *unbiased* and *complete* overview of a subject. With that in mind, we have applied methodology guidelines [KC07], and designed a reproducible and an unambiguous protocol. However, the results of this study are a compromise. By definition the word ‘game’ is a concept whose ‘essence’ cannot be captured in words [Wit53]. Therefore this study can never be fully complete, unambiguous, and unbiased. Here we address threats to validity.

Scoping the area of interest

We formulated two queries to obtain evidence for our hypotheses. Our narrow query, aimed at our second hypothesis, is biased towards software engineering where the term ‘domain-specific language’ is common. To obtain a more nuanced and complete

Table 2.29: Several PhD Dissertations related to Languages of Games and Play

author	thesis	query	research angle	language
Abbadi	[Abb17]	195n	DSL for general game development	88: Casanova
Ahmadi	[Ahm12]	158n	Game based learning	65: AgentSheets
Anderson	[Ando8a]	247n	Behaviors and Game AI	40: SEAL
Ašeriškis	[Aše17]	241n	Gamification	68: UAREI
Borghini	[Bor15]	263n	Assessment systems in game based learning	21: EngAGe DSL
Browne	[Broo8]	–		72: Ludi
Dormans	[Dor12a]	97w	Game mechanics and level generation	26: Machinations 36: Ludoscope
Furtado	[Fur12]	93n	Domain-Specific Modeling Languages	96: SharpLudus
Guo	[Guo15]	97n	Modeling Pervasive Games	3: PerGO
Gaudl	[Gau16]	274n	Real-Time Game AI	44: POSH#
Guana	[Gua17]	209n	Modeling Games	99: PhyDSL
Holloway	[Hol16]	272n	Modeling storylines	
Mahlmann	[Mah13]	289n		73: SGDL
Mayer	[May17]	390n		
Martens	[Mar15]	–	Programming narratives in Linear Logic	48: Ceptre
Neil	[Nei15]	–	Evaluates game design tools	
Mehm	[Meh13]	238n	authoring tools for the educational domain	18: StoryTec
Osborn	[Osb18]	–	Operationalizing Operational Logics	17: Operational Logics 57: Playspecs 30: Gamelan
Adam Smith	[Smi12]	494w	Mechanizing exploratory game design	25: BIPED and LUDOCORE
Zhu	[Zhu14]	165n		103: RAIL
Zook	[Zoo16]	–w	Automated iterative game design	31: PDDL

overview that also includes other research fields, we formulated a wide query aimed at our first hypothesis, with focus on languages in general. However, more than 16K GS results was more than is feasible for us to analyze. We compromised and chose to limit our analysis to its top 1K results. In addition, we filtered terms that are often, but not always, off-topic. As a result, despite our best efforts, we may have overlooked relevant publications.

Breaking protocol

We cited publications not conforming to either of our queries to clarify the origins, descriptions and applications of a language. In addition, we included several papers that conform to the wide query, but did not appear in the top 1K results. We have clearly marked these in the language summaries, and added the bibliographical data in a separate library, as can be seen in Figure 2.2. While this makes our overview more complete, it also reintroduces the selection bias we wished to avoid in the first place.

Pilot error

We have summarized and related publications from a wide array research fields, areas and topics. Unfortunately, despite our best efforts, we have inevitably overlooked or mischaracterized contributions. This study is intended as an inclusive, constructive and 'living' document that we hope to discuss, improve and extend over time.

Synthesizing perspectives

We synthesized fifteen perspectives on languages of games and play from over one hundred language summaries. Our decomposition of the topic of interest is a best-effort interpretation that relies on our personal experience. We acknowledge that it is possible to formulate other research perspectives that extend the collection. Different authors, who have distinct research needs and goals, might want to shed light on a problem from a different angle. They could choose finer granularity to zoom in on an area, and reuse different subsets of languages that cross-cut topics in different ways.

Missing in action: game development practice

We have mapped the state-of-the-art in languages of games and play for a wide audience, which in our view, should include practitioners. Unfortunately, because we identified relatively few practical sources, we have not fully delivered on this promise. We acknowledge this is a limitation of our research method, which does not include non-written sources such as games, development kits, engines and tools. Of course, GS primarily contains academic sources, and the game development industry is not in the business of publishing papers. In addition, fierce competition and time-to-market pressure have led to a degree of industrial secrecy. By not sharing information, many businesses are simply protecting their intellectual property and competitive edge.

Ongoing work

The advantages of mapping studies come at the cost of a high effort. We have spent a significant amount of time analyzing a large number of publications while authors continued to publish. However, we have neither categorized every publication we

included, nor have we analyzed every language we identified in detail. As a result, our analysis remains an ongoing process of updates and extensions.

Using Google Scholar

Our choice for GS is motivated by its high recall. Using GS we obtained publications from independent venues we did not know existed. However, Google owns the information records on GS, maps the interests of its users, and does not provide bulk access¹². This complicates systematic studies in general, which require an off-line analysis in order to ‘stand on the shoulders of giants’. As a result, it is not straightforward to reproduce this study.

Applying bibliometrics

We obtained citation data from GS and constructed the citation graph using a combination of Python scripts and Gephi. Given the right tools, we could have extracted the citation data directly from the PDFs. In addition, we used Gephi’s built-in layout algorithms to obtain a suitable image. However, the same data can produce different graph layouts as well. Mapping studies are complicated by a lack of tools for bibliographic analysis and bibliometrics. Standardizing and automating mapping studies can help save precious time and improve the quality of literature reviews and surveys in general.

2.10 CONCLUSION

We have presented an overview of the state-of-the-art in languages of games and play that relates research areas, goals and applications. We identified and summarized over one hundred languages, and synthesized fifteen research perspectives (or angles) on the topic, each illustrated by selected language summaries.

The results show that there is evidence to support both of our research hypotheses. First, languages, structured notations, patterns and tools can offer designers and developers theoretical foundations that offer experts systematic techniques and practical solutions they need to raise their productivity and improve the quality of games and play. Second, we obtained evidence that DSLs can help in automated game design, and described illustrative examples that suggest how to achieve this.

We also mapped related approaches and described perspectives other than our own departure point with distinct approaches and motivations. Instead of clarifying a single perspective, our map leads in many related research directions, each representing possible departure points for related studies. Ultimately, our map

¹²<https://scholar.google.com/intl/en/scholar/help.html> (visited August 23rd 2019)

provides a good starting point for anyone who wishes to study and learn more about languages of games and play.

2.10.1 *Future Work*

We foresee the following future work.

- We plan to create a Wiki on languages of games and play. That we can grow and maintain related work as a 'living document', like *Gameplay Design Patterns* and (Language 7) and *Game Ontology Project* (Language 2).
- We see opportunities for additional mapping studies and literature reviews that intersect with this study and zoom in on specific related areas, such as automated game design, mixed-initiative approaches, procedural content generation and live programming.

Acknowledgements

We thank Paul Klint, Tijs van der Storm and Daria Polak for proof reading and providing valuable feedback that helped improve this paper.

In addition, we thank the many colleagues who over the years have pointed out so many languages, venues and publications.

Last but not least, we thank the authors whose contributions we had the privilege to read, summarize and relate. We would like to apologize to any peer who feels their work is unfairly treated or incorrectly portrayed.

Abstract

In the multi-billion dollar game industry, time to market limits the time developers have for improving games. Game designers and software engineers usually live on opposite sides of the fence, and both lose time when adjustments best understood by designers are implemented by engineers.

Designers lack a common vocabulary for expressing gameplay, which hampers specification, communication and agreement. We aim to speed up the game development process by improving designer productivity and design quality. The language *Machinations* has introduced a graphical notation for expressing the rules of game economies that is close to a designer’s vocabulary.

We present the language *Micro-Machinations* (MM) that details and formalizes the meaning of a significant subset of *Machination*’s language features and adds several new features most notably modularization. Next we describe *MM Analysis in RASCAL* (MM AiR), a framework for analysis and simulation of MM models using the RASCAL meta-programming language and the SPIN model checker. Our approach shows that it is feasible to rapidly simulate game economies in early development stages and to separate concerns. Today’s meta-programming technology is a crucial enabler to achieve this.

3.1 INTRODUCTION

There is anecdotal evidence that versions of games like *Diablo III*¹ and *Dungeon Hunter 4*² contained bugs in their game economy that allowed players to illicitly obtain game resources that could be purchased for real money. Such errors seriously threaten the business model of game manufacturers. In the multi-billion dollar game industry, time to market limits the time designers and developers have for creating, implementing and improving games. In game development speed is everything. This applies not only to designers who have to quickly assess player experience and to developers that are under enormous pressure to deliver software on time, but also

This chapter was previously published as P. Klint and R. van Rozen. “Micro-Machinations: a DSL for Game Economies”. In: *Software Language Engineering – Proceedings of the 6th International Conference on Software Language engineering, SLE 2013, Indianapolis, IN, USA, October 26–28, 2013*. Ed. by M. Erwig, R. F. Paige, and E. Van Wyk. Vol. 8225. LNCS. Springer, 2013, pp. 36–55. ISBN: 978-3-319-02654-1. DOI: 10.1007/978-3-319-02654-1_3

¹<http://us.battle.net/d3/en/forum/topic/8796520380> (visited May 14th 2019)

²<https://web.archive.org/web/20130813083409/http://www.data-apk.com/2013/04/dungeon-hunter-4-v1-0-1.html> – misses the comments section (visited May 14th 2019)

to the performance of the software itself. Common software engineering wisdom does not always apply when pushing technology to the limits regarding performance and scalability. Domain-Specific Languages (DSLs) have been successfully applied in domains ranging from planning and financial engineering to digital forensics resulting in substantial improvements in quality and productivity, but their benefits for the game domain are not yet well-understood.

There are various explanations for this. The game domain is diffuse, encompassing disparate genres, varying objectives and concerns, that often require specific solutions and approaches. Because the supporting technologies are constantly changing, domain analysis tracks a moving target, and opportunities for domain modeling and software reuse are limited [Blo04]. Existing academic language-oriented approaches, although usually well-scoped, are often poorly adaptable, one-off, top-down projects that lack practical engineering relevance. Systematic bottom-up development and reuse have yielded libraries called game engines but such (commercial) engines are no silver bullet either, since they only provide general purpose solutions to technical problems and need significant extension and customization to obtain the functionality for a completely new game. Engines represent a substantial investment, and also create a long-term dependency on the vendor for APIs and support.

Our objective is to demonstrate that game development can benefit from DSLs despite the challenges posed by the game domain and the perceived shortcomings of existing DSL attempts. We envision light-weight, reusable, inter-operable, extensible DSLs and libraries for well-scoped game concerns such as story-lines, character behavior, in-game entities, and locations. We focus in this paper on the challenge of speeding up the game development process by improving designer productivity and design quality. Our main contributions:

- Micro-Machinations (MM), a DSL for expressing game economies.
- Micro-Machinations Analysis in RASCAL (MM AiR), an interactive simulation, visualization and validation workbench for MM.
- The insight that combining state-of-the-art tools for meta-programming (RASCAL³ [KvdSV11]) and model checking (PROMELA/SPIN⁴ [Holo3]) enable rapid prototyping of and experimentation in the game domain with frameworks like MM AiR.

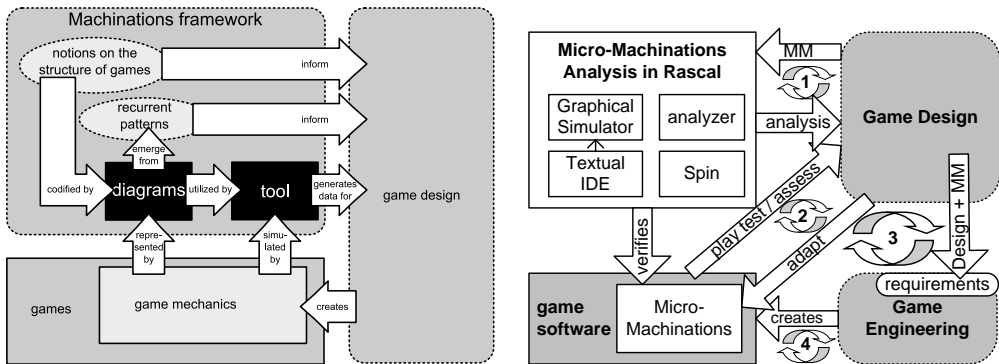
3.2 MICRO-MACHINATIONS

3.2.1 *Background*

Our main source of inspiration is the language Machinations [AD12] that has been based on an extensive analysis of game design practices in industry and provides

³<http://www.rascal-mpl.org/> (visited May 14th 2019)

⁴<http://spinroot.com/spin/whatispin.html> (visited May 14th 2019)



(a) Machinations Conceptual Framework

(b) Micro-Machinations Architecture

Figure 3.1: Side-by-side comparison of Machinations (a) and Micro-Machinations (b)

a graphical notation for designers to express the rules of game economies. A *game economy* is an abstract game system governed by rules (e.g., how many coins do I need to buy a crystal) that offers players a playful interactive means to spend and exchange atomic game resources (e.g., crystals, energy). Resources are characterized by *amount* and *unit kind*.

Its focus is on the simulation of game designs. Various game design patterns have been identified in this context [AD12; Dor11b] as well. Machinations takes an approach that closely resembles Petri Nets that have been used in the game domain by others. For instance, Brom and Abonyi [BA06] use Petri nets for narratives, and Araújo and Roque [AR09] propose general game design using Petri Nets. Other approaches to formalisms for game development related to design include hierarchical state machines [FHJ03], behavior trees [Chao7], and rule-based systems [MCS⁺04b].

Machinations is a visual game design language intended to create, document, simulate and test the internal economy of a game. The core *game mechanics* are the rules that are at the heart of a game. Machinations diagrams allow designers to write, store and communicate game mechanics in a uniform way. Perhaps the hardest part of game design is adjusting the game balance and make a game challenging and fun.

Figure 3.1(a) shows the Machinations framework as presented in [AD12]. Machinations can be seen as a design aid, that augments paper prototyping, which is used by designers to understand game rules and their effect on play. The Machinations tool⁵ can be used to generate automatic random runs, that represent possible game developments, as feedback on the design process. Machinations is already in use by several game designers in the field.

⁵<https://web.archive.org/web/20180620162709/JorisDormans.nl/machinations> (visited May 14th 2019)

Micro-Machinations (MM) is an evolutionary continuation of Machinations aiming at software prototyping and validation. MM is a formalized extended subset of Machinations, that brings a level of precision (and reduction of non-determinism) to the elements of the design notation that enables not only simulation but also formal analysis of game designs. MM also adds new features, most notably modularization. MM is intended as embedded scripting language for game engines that enables interaction between the economic rules and the so-called *in-game entities* that are characterized by one or more atomic resources. An advantage of early paper prototyping is that loosely defined rules can be changed quickly and analyzed informally.

Later, during software prototyping the rules have to be described precisely and making non-trivial changes usually takes longer. To start software prototyping as early as possible, a quick change in a model should immediately change the software that implements it. Therefore we study the precise meaning of the language elements and how they affect the game state. By leveraging meta-programming, language work-benches and model checking we can provide additional forms of analysis and prototyping. This enables us to answer questions about models designers might have, that affect both the design and the software that implements it. Figure 3.1(b) shows schematically how MM relates to game development.

Our objectives are to introduce short and separate design iterations (1 and 2) to free time for separate software engineering iterations (4) and alleviate relying on the usually longer interdependent development iterations (3).

3.2.2 *Micro-Machinations Condensed*

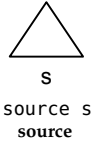
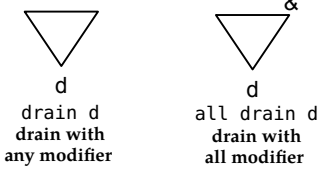
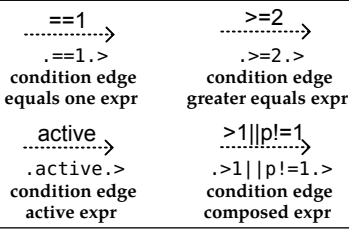
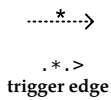
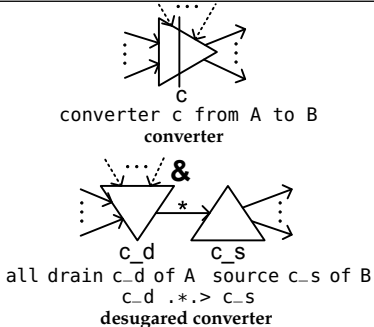
MM models are graphs that consist of two kinds of elements, *nodes* and *edges*. Both may be annotated with extra textual or visual information. These elements describe the rules of internal game economies, and define how resources are step-by-step propagated and redistributed through the graph. Here is a cheat sheet for the most important language elements⁶.

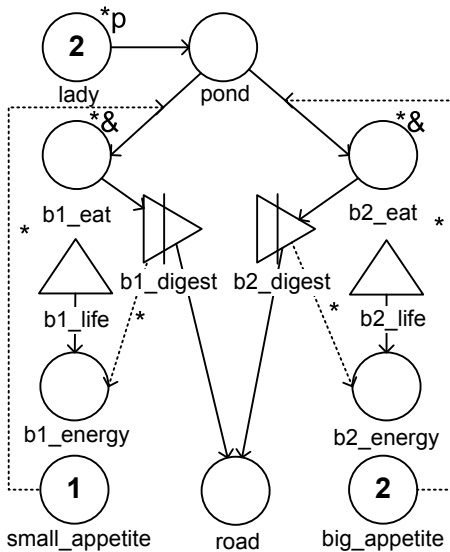
⁶For conciseness we only give an informal description here, closely adhering to [AD12].

Table 3.1: Cheat sheet for basic Micro-Machinations language elements (part 1)

		<p>A <i>pool</i> is a named node, that abstracts from an in-game entity, and can contain <i>resources</i>, such as coins, crystals, health, etc. Visually, a pool is a circle with an integer in it representing the current amount of resources, and the initial amount at which a pool starts when first modeled.</p>
pool p Empty pool	pool p at 1 Pool & resource	
		<p>Pools may specify a maximum capacity for resources, which can never be exceeded, that is visually a prefix max followed by an integer.</p>
pool p at 1 max 2 Limited capacity	pool p at 2 max 2 Full pool	
		<p>A <i>resource connection</i> is an edge with an associated expression that defines the rate at which resources can flow between source and target nodes. During each transition or <i>step</i>, nodes can <i>act</i> once by redistributing resources along the resource connections of the model. The <i>inputs</i> of a node are resource connections whose arrowhead points to that node, and its <i>outputs</i> are those pointing away.</p>
---> resource connection flow rate of one	- /2 -> resource connection half flow rate	
		<p>The <i>activation modifier</i> determines if a node can act. By default, nodes are <i>passive</i> (no symbol) and do not act unless activated by another node. <i>Interactive</i> (double line) nodes signify user actions that during a step can activate a node to act in the next state. <i>Automatic</i> (*) nodes act automatically, once every step. <i>Start</i> (s) nodes are active in the initial state, but become passive afterwards.</p>
all-> resource connection unlimited flow rate	4*p+1-> resource connection flow expression	
		<p>By default, nodes are <i>passive</i> (no symbol) and do not act unless activated by another node. <i>Interactive</i> (double line) nodes signify user actions that during a step can activate a node to act in the next state. <i>Automatic</i> (*) nodes act automatically, once every step. <i>Start</i> (s) nodes are active in the initial state, but become passive afterwards.</p>
pool p passive pool	auto pool p automatic pool	
		<p><i>Interactive</i> (double line) nodes signify user actions that during a step can activate a node to act in the next state. <i>Automatic</i> (*) nodes act automatically, once every step. <i>Start</i> (s) nodes are active in the initial state, but become passive afterwards.</p>
user pool p interactive pool	start pool p start pool	
		<p>Nodes act either by pulling (default, no symbol) resources along their inputs or pushing (p) resources along their outputs. Nodes that have the <i>any modifier</i> (default, no symbol), interpret the flow rate expressions of their resource connections as upper bounds, and move as many resources as possible. Additionally, these nodes may process their resource connections independently and in any order. Nodes that instead have the <i>all modifier</i> (&) interpret them as strict requirements, and the associated flows all happen or none do.</p>
pool p pool with pull act and any modifier	all pool p pool with pull act and all modifier	
push pool p pool with push act and any modifier	push all pool p pool with push act and all modifier	

Table 3.2: Cheat sheet for basic Micro-Machinations language elements (part 2)

	<p>A <i>source</i> node, appearing as a triangle pointing up, is the only element that can generate resources. A source can be thought of as a pool with an infinite amount of resources, and therefore always pushes all resources or all resources are pulled from it. The any modifier does not apply, and resources may never flow into a source. Also, infinite amounts may not flow from sources.</p>
	<p>A <i>drain</i> node, appearing as a triangle pointing down, is the only element that can delete resources. Drains can be thought of as pools with an infinite negative amount of resources, and have capacity to pull whatever resources are available, or whatever resources are pushed into them. No resources can ever flow from a drain.</p>
	<p>A node can only be active if all of its <i>conditions</i> are true. A <i>condition</i> is an edge appearing as a dashed arrow with an associated Boolean expression. Its source node is a pool that forms an implicit argument in the expression, and the condition applies to the target node.</p>
	<p>A <i>trigger</i> is an edge that appears as a dashed arrow with a multiply sign. The origin node of a trigger activates the target node when for each resource connection the source works on, there is a flow in the transition that is greater or equal to that of the associated flow rate expression. Additionally, automatic pulling nodes without inputs and automatic pushing nodes without outputs always activate targets of their triggers.</p>
	<p>Converters are nodes, appearing as a triangle pointing right with a vertical line through the middle, that consume one kind of resources and produce another. Converters are not core elements because they can be rewritten as a combination of a drain, a trigger and a source. Unlike basic node types, converters therefore take two steps to complete. Converters can only pull, and the any modifier does not apply. If specified, the unit kinds on the inputs and outputs must match the converter's unit kinds.</p>



(a) Visual Model

```

1 unit Bread      : "bread_crumbs"
2 unit Droppings : "bird_residue"
3 unit Energy     : "bird_energy"
4 pool BIG_APPETITE of Bread at 2
5 pool SMALL_APPETITE of Bread at 1
6 auto push all pool lady of Bread at 2
7 pool pond of Bread
8 pool road of Droppings
9 lady --> pond
10 auto all pool b1_eat of Bread
11 pond -SMALL_APPETITE-> b1_eat
12 auto converter b1_digest
13 from Bread to Droppings
14 b1_eat --> b1_digest
15 b1_digest --> road
16 pool b1_energy of Energy
17 source b1_life of Energy
18 b1_digest .*> b1_energy
19 b1_life --> b1_energy
20 auto all pool b2_eat of Bread
21 pond -BIG_APPETITE-> b2_eat
22 auto converter b2_digest
23 from Bread to Droppings
24 b2_eat --> b2_digest
25 b2_digest --> road
26 pool b2_energy of Energy
27 source b2_life of Energy
28 b2_digest .*> b2_energy
29 b2_life --> b2_energy

```

(b) Textual model that demonstrates code duplication

Figure 3.2: Modeling two birds that both eat from the same pond

3.2.3 Introductory Example

Figure 3.2(a) shows an example how a designer might model a lady feeding birds in the original Machinations language. Figure 3.2(b) shows the textual equivalent as introduced in MM. The lady automatically throws bread crumbs in a pond (*p) one at a time, and two birds with different appetites compete for them. The first has a small appetite and the latter a big a appetite. Both birds automatically try to eat the whole amount (*&) their appetite compels them to. The edges from *small_appetite* and *big_appetite* are not triggers but *edge modifiers*, and we have replaced them by flow rate expressions in MM (lines 11 & 21). Birds digest food automatically which gives them energy and produces *droppings* on the road.

3.2.4 Game designer's questions

Given a model such as the example from Section 3.2.3, a designer might have the following questions.

- **Inspect:** Given a game state, what are the values of the pools, which nodes are active and what do they do?
- **Select:** Given a game state, what are the possible transitions? Are there alternatives? What are these alternatives and what are their successor states?
- **Reach:** Given this model, does a node ever act? Does a flow ever happen? Does a trigger ever happen? Where in the model can resources be scarce? Is an undesired state reachable, e.g., can the player ever have items from the store without paying crystals? Is a desired state always reachable, e.g., can the game be won or can the level be finished?
- **Balance:** Are the rules well balanced?

3.2.5 *Technical challenges*

Before answering these questions (in Section 3.2.6), we discuss engineering challenges and how to tackle them leveraging meta-programming, language workbenches and model checking.

- **Parse:** To analyze any of these questions we need a representation that can easily be parsed. Therefore, MM introduces a textual representation of the game model, that serves as an intermediate format, that is compact and easy to read, parse, serialize and store.
- **Reuse:** Having a closer look at the example in Figure 3.2(a), we see mirroring in the game graph that corresponds to code duplication in Figure 3.2(b). We need modular constructs for reuse, encapsulation, scaling views, partial analysis and testing, and embedding MM in games (by way of connecting nodes and edges with in-game entities).
- **Inspect:** We need an environment that enables users to inspect states by visualizing serialized models.
- **Select:** Detailed insight in the game behavior can be obtained by interactively choosing successors and seeing transition alternatives. This is similar to debugging when stepping through code, and requires the calculation of alternatives. This can, for instance, reveal a lack of resources or capacity.
- **Analyze context constraints:** Some structural elements of models, related to contextual constraints can introduce errors that we want to catch statically. Examples are: (i) Sources cannot have inputs; (ii) Drains cannot have outputs; (iii) Edges are dead code if no active node can use them by pushing or pulling; and (iv) Edges are doubly used when both origin and target are pushing and pulling, which can lead to confusing results. Modeling errors can also be detected. Optionally, resource types of nodes can be defined making resource connections easily checkable. Additionally, missing references can be reported.
- **Analyze reachability:** Analyzing reachability is hard because it requires calculation of all possible paths through the game graph. Normally, we cannot

calculate all possible executions of programs due to the sheer number of possibilities, and use abstractions to allow forms of analysis. Because a MM model is itself an abstraction of the actual game, and types and instances —MM’s modularization mechanism is described in more detail in Section 3.2.7—enable partial analysis, we can exhaustively verify models in an experimental context using model checking techniques. The challenge is to translate MM diagrams to models that a model checker can analyze, and making that analysis *scalable*. Non-deterministic choices lead to a combinatorial explosion of execution path and this results in a state explosion in the model checker. When searching for undesired situations, an exhaustive search may not be necessary, since the moment an invalid state is found, the execution stack trace represents a result.

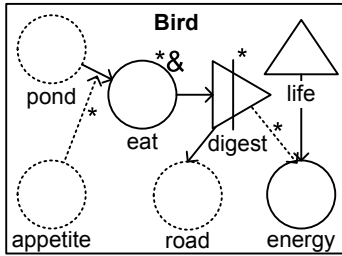
- **Balance:** Providing useful analysis to support balancing games is very hard, since this requires analyzing multiple types of play, each dynamic with different *unpredictable* player choices and non-deterministic events. Experimental set-ups in which instance interfaces are subjected to modeled input may provide designers with useful feedback, but building such set-ups is hard and is the expertise of game designers.
- **Prototype and adjust:** Prototyping game software and making adjustments requires code. In addition to the MM format we require a light-weight embeddable interpreter that enables using script for prototyping and adjusting game software. A simple API for integrating MM in existing architectures should at least provide a means for calculating successor states (step), observing pools value changes, activating interactive nodes and reading and storing information. We require that this API relates the run-time state of models to the state and the behavior of game elements that affect how the game behaves when played. This is not further explored in the current paper.

3.2.6 Answers to game designer’s questions

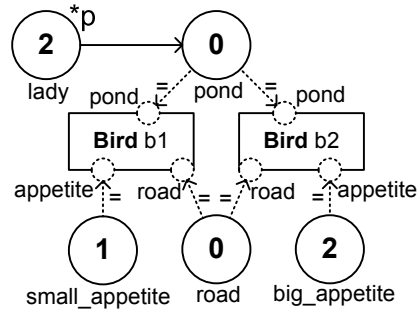
We will now answer the questions raised in Section 3.2.4 and illustrate them using the bird feeding example.

Figure 3.3 shows a rewrite of the example using new language elements to be detailed in Section 3.2.7. Figure 3.3(a) shows the definition of *Bird*, which references external nodes *pond*, *road* and *appetite*. These external nodes act as formal parameters of the Bird specification and are bound twice in Figure 3.3(b). Figure 3.4(a) and Figure 3.4(b) show the textual equivalent of this model.

Next, we introduce assertions and pose that birds shall never starve by adding an assertion at the bottom lines of Figure 3.4(a). Then, we run the analysis to check for reachability and find that (i) bird *b2* starves because *b1.eat* always happens before *big.appetite* is available, and (ii) the acts of bird *b2.eat* and *b2.energy* are unreachable for all execution paths. Finally, we can explore the model and understand it better



(a) A bird's life



(b) A lady feeding two birds

Figure 3.3: Graphically modeling birds that eat, digest and live

```

Bird (ref appetite ,
      ref pond,
      ref road)
{
  //birds eat exactly all
  they want
  auto all pool eat of Bread
  pond -appetite-> eat
  auto converter digest
  from Bread to Droppings

  //digest Bread
  eat --> digest

  //produce Dropping
  digest --> road

  pool energy of Energy
  source life of Energy
  digest .*> energy
  life --> energy
  assert fed:
  energy > 0 || road < 2
  "birds_always_get_fed"
}

unit Bread : "bread_crumbs"
unit Droppings : "bird_residue"
unit Energy : "bird_energy"

pool BIG_APPETITE of Bread at 2
pool SMALL_APPETITE of Bread at 1

//a lady throws crumbs in a pond
auto push all pool lady
of Bread at 2
pool pond of Bread
pool road of Droppings
lady --> pond

lady -1-> pond
step
pond -1-> b1_eat
lady -1-> pond
step
pond -1-> b1_eat
b1_eat -1-> b1_digest_drain
step
b1_eat -1-> b1_digest_drain
b1_life -1-> b1_energy
b1_digest_source -1-> road
step
b1_life -1-> b1_energy
b1_digest_source -1-> road
step
violate b2_fed

//one bird has a big appetite
Bird b1
BIG_APPETITE .=> b1.appetite
pond .=> b1.pond
road .=> b1.road

//the other has a small appetite
Bird b2
SMALL_APPETITE .=> b2.appetite
pond .=> b2.pond
road .=> b2.road

```

(a) A bird's life

(b) A lady feeding two birds

(c) Bird b2 starves

Figure 3.4: Textual model and analysis that shows birds with a big appetite starve

by inspecting states, observing lack of alternative transitions, and automatically simulating the trace that lead to the assertion violation visually, shown textually in Figure 3.4(c).

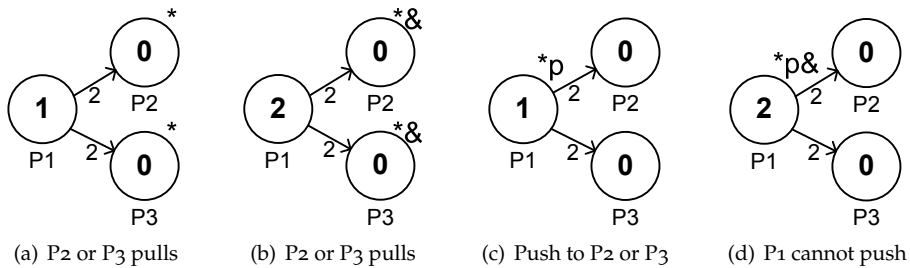


Figure 3.5: Non-determinism due to shortage of resources

3.2.7 Language Extensions

We have designed MM and have introduced new language features as necessary to attain our goals. MM has modular constructs for reuse, encapsulation, scaling views, partial analysis and testing, and relating MM to in-game entities. MM has reduced non-determinism and increased control over competition for resources and capacity by introducing priorities. Time is modeled and understood, in a way that is embeddable in games. Finally, invariants are introduced for defining simple properties for analysis.

Types definitions and instances The following table introduces⁷ our modularization features *type definitions* and *instances*.

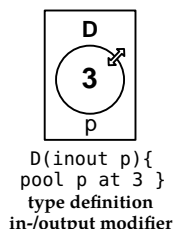
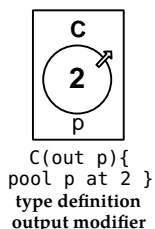
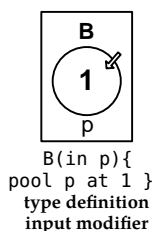
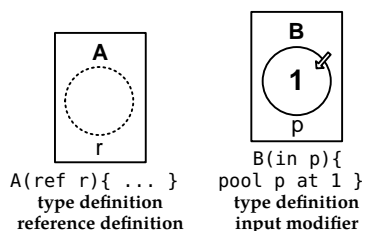
Nodes have priorities The sources of non-determinism that we have identified are nodes *competing* for resources and the *any* modifier. Alternative transitions exist due to lack of resources or capacity, as illustrated by Figure 3.5 and Figure 3.6.

We have already mentioned that each activated node can act once during a step. Since the order in which nodes act is not defined, models under-specify behavior and this can result in undesirable non-determinism. To allow a degree of control, we specify that active nodes with the following actions and modifiers are scheduled in the following order: pull all, pull any, push all, push any. Groups of nodes from different categories do not compete for resources or capacity, which helps in analyzing models and in understanding them. Section 3.4 makes use of this feature.

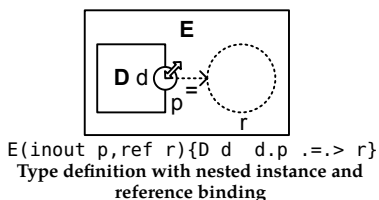
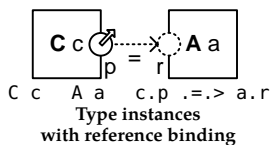
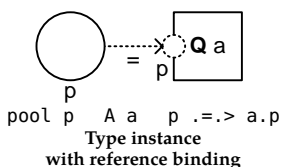
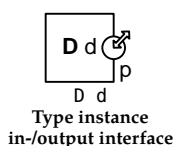
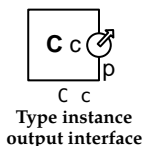
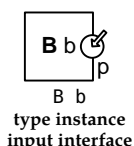
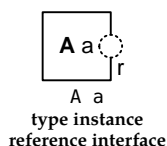
Steps take time MM does not support different *time modes* as Machinations does. In MM each node may act at most once during a step, which conforms to the Machinations notion of *synchronous time*. We do not support *asynchronous time*, in which user activated nodes may act multiple times during a step without affecting other nodes. Machinations supports a *turn-based mode*, in which players can each

⁷Once again, for conciseness, only informally.

Table 3.3: Cheat sheet for modular Micro-Machinations language elements



A *type definition* is a named diagram that functions as parameterized module for encapsulating elements. Type definitions define internal elements and how they can be used externally. A *reference*, represented by a circle with a dashed line, is an alias that refers to a node that is defined externally. Internal nodes annotated with an *interface modifier* *input*, *output* or *input/output* become interfaces on the instances of the type. The input modifier denotes that an interface accepts inputs, output implies it accepts outputs and input/output accepts both. Interface modifiers appear as an arrow in the top right corner of a node, where an input modifier point into the node, an output modifier points out of the node, and an in-/output modifier does both.



An *instance* is a named object that has individual instance data, whose interfaces are defined by its type and can be bound to other models, acting as formal parameters.

An *interface* makes internal elements of an instance available to the outside, and can be used by connecting resource connections. Visually, an interface is a small circle at the border of an instance with its name under it. Input interfaces have an arrow pointing into the circle, outputs have an arrow pointing outward, and in-/outputs have a bidirectional arrow. The direction of the arrow implies the validity of the direction of the edges that connect to it. Only reference interfaces appear with a dashed line.

References must be bound to definitions using edges called *bindings*, represented by dashed arrows annotated with an equal sign, that originate from a defining node and target a reference.

Additionally, instances can be nested inside type definitions and build a name space, e.g., a nested pool *p* inside an instance *a* of type definition *A* is referred to as *a.p*.

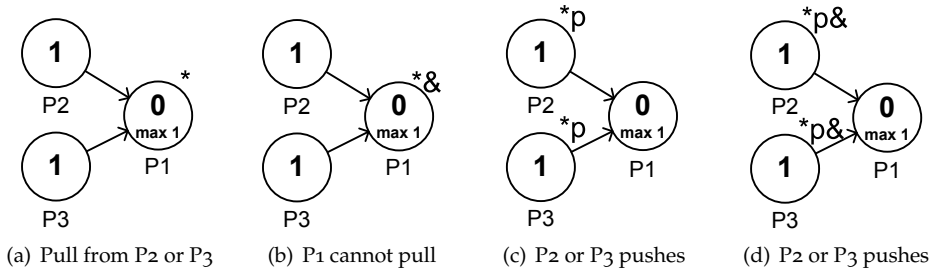


Figure 3.6: Non-determinism due to shortage of capacity

spend a fixed number of action points on activating interactive nodes each step. We note that *turns* are game assets that can be modeled, using pools, conditions and triggers, enabling turn-based analysis. MM does not specify how long a step takes, it only assumes that steps happen and its environment determines what the step intervals are.

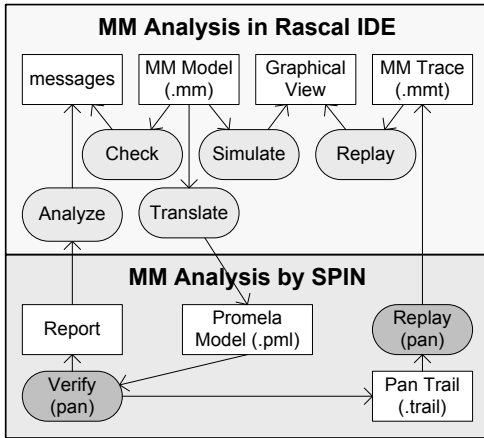
Invariants Defining property specifications to verify a model against can be hard, requiring knowledge of linear temporal logic. Defining invariants, Boolean expressions that must be true for each state, is easier to understand. MM adds *assertions* which consist of a name, a boolean expression that must invariantly be true, and a message to explain what happened when the assertion is violated, i.e. becomes false for some state. Figure 3.4(a) contains an example of an assertion (lines 14–15).

3.3 MM AIR FRAMEWORK

Figure 3.7(a) shows the main functions of the MM Analysis in RASCAL (MM AiR) framework and Figure 3.7(b) relates them to the challenges they address. The framework is implemented as a RASCAL meta-program of approximately 4.5 KLOC. We will now describe the main functions of the framework.

3.3.1 Check contextual constraints

Starting with a grammar for MM’s textual syntax, using RASCAL we generate a basic MM Eclipse IDE that supports editing and parsing textual MM models with syntax highlighting. This IDE is extended with functionality to give feedback when models are incorrect or do not pass contextual analysis. This is implemented in a series of model transformations, leveraging RASCAL’s support for pattern matching, tree visiting and comprehensions. This includes labeling the model elements, for storing information in states and for resource redistributions in transitions. We check models against the contextual constraints described in Section 3.2.5.



(a) MM AiR IDE functions

§	functionality	challenges
3.3.1	<i>check</i> contextual constraints (parse, desugar, perform static analysis)	define syntax, semantics, reuse, constraints
3.3.2	<i>simulate</i> MM model (interpret and evaluate successor states, interactive graphical visualizations)	make models debuggable, improve scalability and performance
3.3.3	<i>translate</i> MM to PROMELA	relate formalisms, ensure interoperability, improve scalability
3.3.4	<i>verify</i> MM in SPIN	ensure interoperability, improve scalability
3.3.5	<i>analyze</i> reachability	ensure interoperability
3.3.6	<i>replay</i> behaviors and verification results	source level debugging, ensure interoperability, readability

(b) Sections, functions and challenges

Figure 3.7: MM AiR Overview

3.3.2 Simulate models

Simulate provides a graphical view of a MM model and enables users to inspect states, choose transitions and successors, and navigate through the model by stepping forward and backward. We generate figures and interactive controls for simulating flattened states and transitions. This is easily done by applying RASCAL's extensive visualization library, which renders figures and provides callbacks we use to call an interpreter. The *interpreter* calculates successor states by evaluating expressions, checking conditions and generating transitions.

3.3.3 Translate to PROMELA

The biggest challenge in analyzing MM is providing a scalable reachability analysis. We achieve this by translating MM to PROMELA, the input language of the SPIN model checker. A naive approach is to model each node as a process, enabling every possible scheduling permutation to happen. However, not every scheduling results in a unique resource distribution, which hampers performance and scalability. Therefore we take steps to reduce the number of calculations without excluding possible behaviors. We take the following measures to reduce the state space explosion.

- **Reduce non-determinism.** We model only necessary non-determinism. We have identified two sources that are currently in MM: nodes *competing* for resources or capacity and the *any modifier*. For competing nodes every permutation

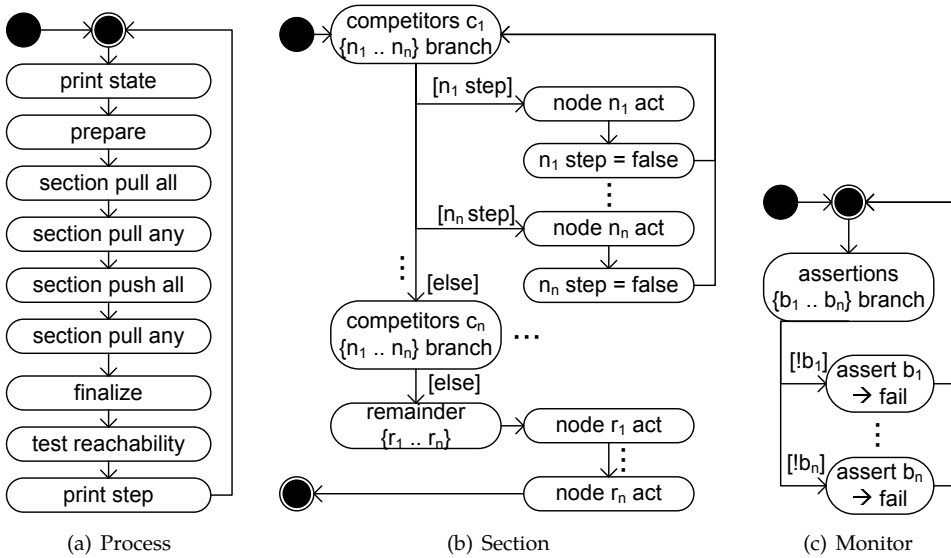


Figure 3.8: Skeleton for generated PROMELA code: process, section and monitor

potentially results in a unique transition that must be computed, but nodes that do not compete can be sequentially processed.

- **Avoid intermediate states.** PROMELA has a `d_step` statement that can be used to avoid intermediate states, by grouping statements in single transitions.
- **Store efficiently and analyze partially.** Pools can specify a maximum that we use to specify which type to use in PROMELA (bit, byte or int), minimizing the state vector. For partial analysis we can limit pool capacities.

Translating an MM model to PROMELA works as follows. We bind references to definitions and transform the model to core MM. We generate one proctype per model, schematically shown in Figure 3.8, and a monitor proctype that tests assertions for each state. Figure 3.8(a) depicts their general structure. At the beginning of a step the state is printed, and step guards are enabled if a node is active. This is followed by *sections* for each priority level as determined by node type. In each section, groups of nodes may be competing for resources or capacity.

For each group of competitors c_i consisting of nodes n_1, \dots, n_n , we introduce a non-deterministic choice using guards that are disabled after a competing node acts as shown in Figure 3.8(b). The remaining independent nodes r_1, \dots, r_n are just sequentially processed, since they never affect each other during a step. Figure 3.8(c) shows that each path in the monitor process remains blocked until an invariant becomes false, and a violation is found.

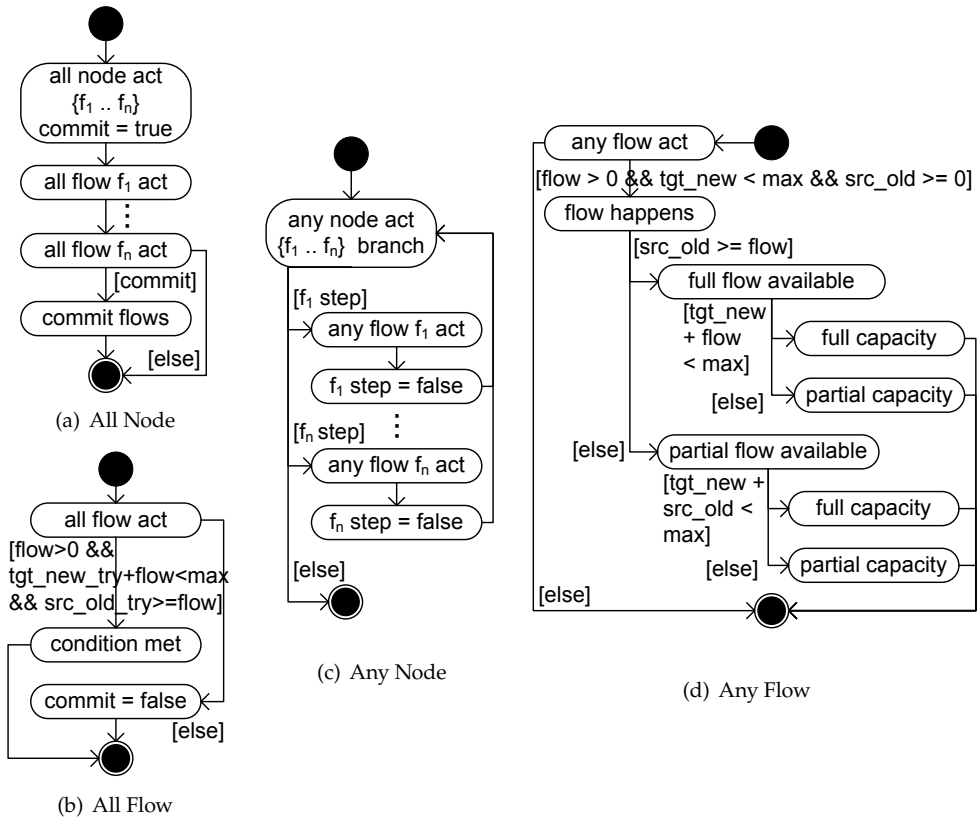


Figure 3.9: Skeleton of generated PROMELA code for nodes

The behavior of nodes with the *all* modifier is deterministic, as shown in Figure 3.9(a) and Figure 3.9(b). All flows f_1, \dots, f_n are executed sequentially and per flow conditions are checked. The effect of all flows is only committed if the conditions for all flows have been satisfied.

The behavior of nodes with the *any* modifier is shown in Figure 3.9(c) and Figure 3.9(d) models the non-determinism by introducing a non-deterministic choice between the flows f_1, \dots, f_n .

Individual nodes act by checking shortages of resources on the old state from which subtractions are made and check shortage of capacity on the new state, to which additions are also made. Finally, when each node has acted the state is finalized by copying the new state to the current state. Temporary values and guards are reset, and active nodes are calculated by applying activation modifiers, triggers and conditions. Next reachability is tested, the step is printed and we start at the beginning to determine the next step.

3.3.4 *Verify invariant properties*

MM models are verified against their assertions by translating them to PROMELA and then running a shell script. The script invokes SPIN and compiles it to a highly optimized model-specific PROMELA analyzer (PAN). It then runs this verifier to perform the state space exploration, and captures the verification report PAN outputs, which may contain unreachable states and associated PROMELA source lines. If the verifier finds an assertion violation, it also produces a *trail*, a series of numbers that represent choices in the execution of the state machine representing the PROMELA model. The challenge is interoperability, relating the verification report and the trail back to MM and showing understandable feedback to the user. We show how this is solved in Section 3.3.5 and Section 3.3.6.

3.3.5 *Analyze reachability*

We tackle the interoperability challenge of relating a SPIN reachability analysis to MM as follows. During the generation of PROMELA we add *reachability tests*, in which states and source lines become reachable if an element acts. We collect the source lines using a tiny language called MM Reach, which specifies the test case by defining whether a node receives full or partial flow via a resource connection or if it activates a trigger. We extract unreachable PROMELA source lines from the PAN verification report and map them back to MM elements to report the following messages, which are relative to a partial or exhaustive search.

- **Starvation.** Nodes that never push or pull full or partial flow via a resource connection starve, and represent dead code.
- **Drought.** A resource connection through which resources do not flow runs dry, and is unused dead code.
- **Inactivity.** A trigger that never activates its target node is idle.
- **Abundance.** A node with the any modifier that always receives full flow along all of its resource connections indicates a lack of shortage.

3.3.6 *Replay behaviors*

We tackle the interoperability challenge of relating PAN trails for PROMELA models we obtained in Section 3.3.4 to MM model resource redistributions by introducing an intermediate language called MM Trace (MMT). A sequence of MMT statements forms a program that contains the transitions that an MM model performs, which MM AiR graphically replays in a guided simulation.

Replaying a trail on PAN simulates the steps of a PROMELA model while calling printf statements that generate an MMT program, ending in an assertion violation. The program is obtained by embedding the following MMT statements prefixed with MM: for filtering in the PROMELA model.

- **Flow.** Node causes flow to occur: source-amount->target
- **Trigger.** Trigger activates a target node in the next state: trigger node
- **Violation.** A state violates an assertion: violate name
- **Step.** Terminate a transition: step

3.4 CASE STUDY: SIMWAR

SimWar is a simple hypothetical Real-Time Strategy (RTS) game introduced by Wright [Wri03] that illustrates the game design challenge of *balancing* a game. This entails ensuring different player choices and strategies represent engaging and challenging experiences. Common strategies for RTS games are *turtling*, a low-risk, long-term strategy that favors defense, and *rushing*, a high-risk short-term strategy that favors attack. Adams and Dormans [AD12] study the game using the Machinations tool⁸. By simulating many random runs, they show the game is indeed poorly balanced and that turtling is the dominant strategy.

Our MM adaptation of SimWar, shown in Figure 3.10, is based on [AD12], but it models the rules for players in a definition called *Base*, avoiding duplication. It also replaces probabilities on resource connections with amounts. Two players compete by spending *resources*, choosing to buy *defense* (cost 1), *attack* (cost 2) or *factories* (cost 5). This is modeled by three converters in line 15–17 of Figure 3.10(b) that pull their respective costs from *resources* when activated.

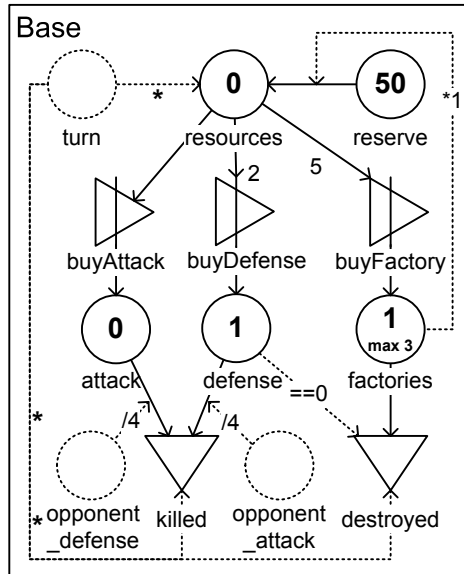
Factories produce income every turn, and represent an investment enabling more purchases. We model this by *turn* triggering *resources* (line 5), which pulls from *reserve* (line 9) the current amount of factories (line 18). A player must destroy their opponent’s factories to win. Two references, *opponent_defense* and *opponent_attack* determine the (rounded down) casualty rate of one in four (line 26, 27) for *attack* and *defense* respectively. Opponents fight until one player has no defense, and her factories are destroyed (line 28).

3.4.1 Experimental setup

In an experiment with SimWar and two strategies shown in Figure 3.11 we apply the MM AiR framework, analyzing (i) the reachability of modeling elements, and (ii) the existence of a strategy that beats a turtling strategy.

The *Turtle* strategy, defined in Figure 3.11(a) and Figure 3.11(b), simply counts turns, and based on this triggers references for buying. The *Random* strategy defined in Figure 3.11(c) and Figure 3.11(d) also counts, but adds a non-deterministic element which uses priorities. Drains *skip*, *getDefence*, *getFactory*, *getAttack* compete for the resource in *choice* before it pulls a resource from *tick*, enabling the next choice. In

⁸<http://www.jorisdormans.nl/machinations/>



(a) SimWar Base

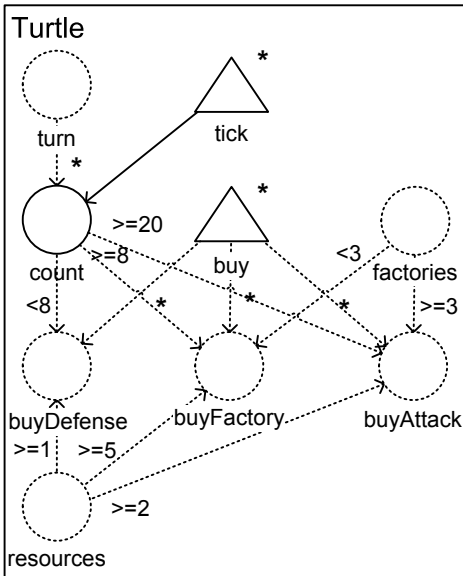
```

1 Base(in BuyAttack, in BuyFactory, in BuyDefense, //choices
2     ref opponent_attack, ref opponent_defense, ref turn,
3     out attack, out defense, out factories, out resources){
4     turn .*> resources //turn triggers resources to pull
5     turn .*> killed //turn triggers killed
6     turn .*> destroyed //turn triggers destroyed
7     pool reserve of Gold at 50 //Gold reserve (starts at 50)
8     pool resources of Gold //Gold resources (for purchases)
9     pool factories of Factory at 1 max 3 //factories for income
10    pool defense of Defense at 1 //defending units
11    pool attack of Attack //attacking units
12    drain killed of Defense, Attack //units can be killed
13    drain destroyed of Factory //factories can be destroyed
14    converter buyDefense from Gold to Defense //buy defense
15    converter buyAttack from Gold to Attack //buy attack
16    converter buyFactory from Gold to Factory //buy factory
17    reserve -factories-> resources //produce income
18    resources -5-> buyFactory //buyFactory consumes 5 Gold
19    buyFactory -> factories //buyFactory produces 1 Factory
20    resources -1-> buyDefense //buyDefense consumes 2 Gold
21    buyDefense -> defense //buyDefense produces 1 Defense
22    resources -2-> buyAttack //buyAttack consumes 1 Gold
23    buyAttack -> attack //buyAttack produces 1 Attack
24    factories -all-> destroyed //factories destruction
25    defense -opponent_attack/4-> killed //defense casualty rate
26    attack -opponent_defense/4-> killed //attack casualty rate
27    defense .defense == 0.> destroyed //undefended condition
28 }

```

(b) SimWar Base

Figure 3.10: The rules of SimWar



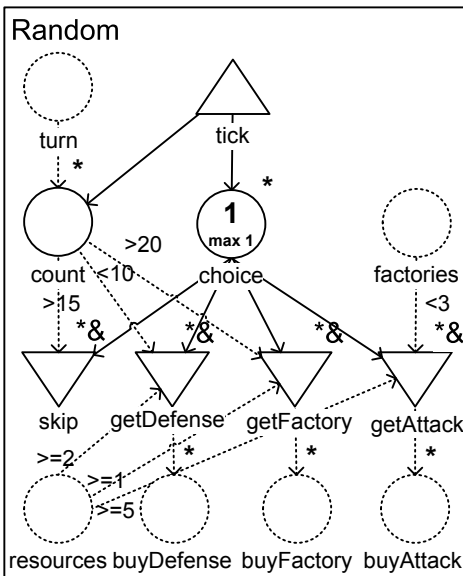
(a) Turtle Strategy

```

1 Turtle(ref buyAttack, ref buyDefense,
2       ref buyFactory, ref factories,
3       ref resources, ref turn){
4   source tick
5   turn .>. count
6   tick --> count
7   pool count
8   auto source buy
9   buy .>. buyAttack
10  buy .>. buyFactory
11  buy .>. buyDefense
12  count .>=20.> buyAttack
13  factories .>=3.> buyAttack
14  resources .>=2.> buyAttack
15  count .<8.> buyDefense
16  resources .>=1.> buyDefense
17  count .>=8.> buyFactory
18  factories .<3.> buyFactory
19  resources .>=5.> buyFactory
20 }

```

(b) SimWar Turtle



(c) Random Strategy

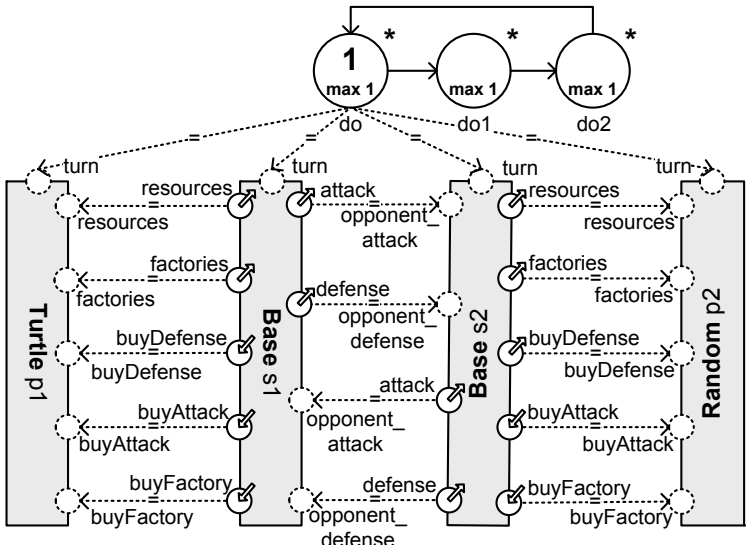
```

1 Random(ref buyAttack, ref buyDefense,
2       ref buyFactory, ref factories,
3       ref resources, ref turn){
4   source tick
5   turn .>. count
6   tick --> count
7   pool count
8   tick --> state
9   auto pool state max 1
10  auto all drain skip
11  auto all drain getFactory
12  auto all drain getAttack
13  auto all drain getDefense
14  getAttack .>. buyAttack
15  getFactory .>. buyFactory
16  getDefense .>. buyDefense
17  state --> skip
18  state --> getAttack
19  state --> getDefense
20  state --> getFactory
21  count .>15.> skip
22  resources .>= 2.> getAttack
23  count .>= 20.> getAttack
24  resources .>= 1.> getDefense
25  count .<10.> getDefense
26  resources .>= 5.> getFactory
27  factories .<3.> getFactory
28 }

```

(d) SimWar Random

Figure 3.11: SimWar Test Strategies



(a) A Turtle instance battling a Random instance

```

1 unit Gold : "gold"
2 unit Factory : "factories"
3 unit Defense : "defense"
4 unit Attack : "attack"
5 Turtle p1 Base s1 //player p1 is turtling
6 Random p2 Base s2 //player p2 is random
7 auto all pool doit at 1 max 1
8 auto all pool do1 max 1
9 auto all pool do2 max 1
10 doit ==> do1 do1 ==> do2 do2 ==> doit
11 doit.==1.>do1 do1.==1.>do2 do2.==1.>doit
12 doit .=> s1.turn
13 s2.defense .=> s1.opponent_defense
14 s2.attack .=> s1.opponent_attack
15 doit .=> p1.turn
16 s1.resources .=> p1.resources
17 s1.buyAttack .=> p1.buyAttack
18 s1.buyFactory .=> p1.buyFactory
19 s1.buyDefense .=> p1.buyDefense
20 s1.factories .=> p1.factories
21 doit .=> s2.turn
22 s1.defense .=> s2.opponent_defense
23 s1.attack .=> s2.opponent_attack
24 doit .=> p2.turn
25 s2.resources .=> p2.resources
26 s2.buyAttack .=> p2.buyAttack
27 s2.buyFactory .=> p2.buyFactory
28 s2.buyDefense .=> p2.buyDefense
29 s2.factories .=> p2.factories
30 assert turtleLives:
31 s1.factories != 0 "turtle_dies"

```

(b) SimWar Battle

Figure 3.12: SimWar experimental test setup

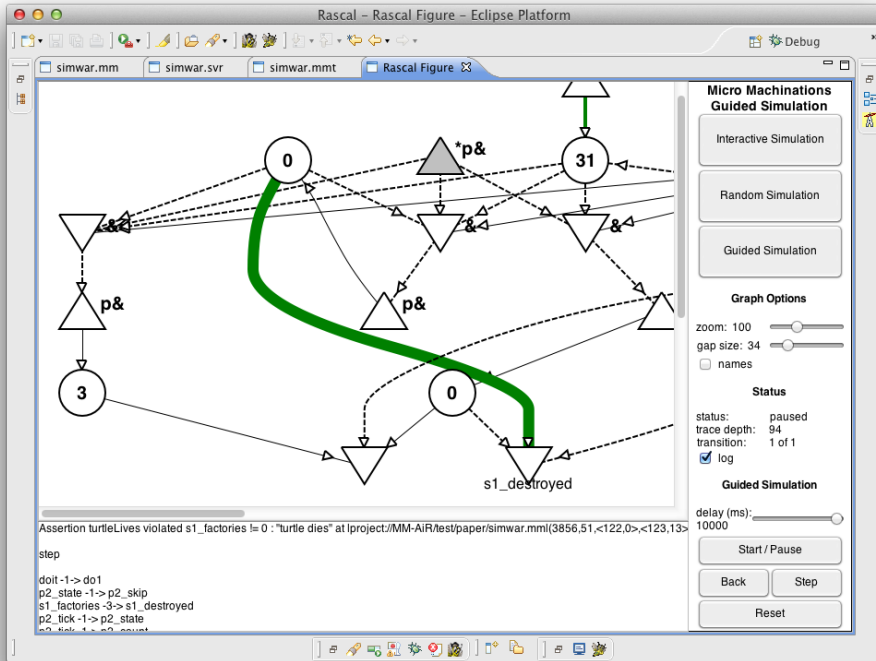


Figure 3.13: MM AiR playing back a counter-example showing *Turtle* defeated

our test set-up shown in Figure 3.12, we bind instances of *Random* and *Turtle* to a *Base* instance in lines 16–20 and 25–29 of Figure 3.12(b). We bind *base* instances as opponents in lines 13–14, 22–23 and bind *turn* to *doit*, our means for activity. Finally, we assert in lines 30–31 that the factories of *Turtle* are never destroyed. A violation of this assertion represents a behavior of *Random* that beats *Turtle*.

3.4.2 Experimental results

We apply MM AiR by translating the models to PROMELA and running SPIN. PAN reports using 2500MB of memory, mostly for storing 10.5M states of 220 bytes, generating 188K states/second, taking 56 seconds on an Intel Core i5-2557M CPU. It reports 11.9M transitions, of which 9.5M are atomic steps, and an assertion violation ($s1_factories != 0$) at depth 8810.

The shortest trail yields an MMT file of 95 steps. Figure 3.13 shows its graphical play-back. We find 22 strategies that beat our Turtle behavior, but these strategies all fall into the turtling category, confirming the strategy is dominant.

During its limited state space exploration, PAN collects unreached PROMELA source lines. Using these, our analysis reports the following:

```
Drought: No flow via s1_factories -s1_factories-> s1_destroyed at line 25 column 2
Drought: No flow via s2_factories -s2_factories-> s2_destroyed at line 25 column 2
Starvation: Node s2_destroyed does not pull at line 14 column 2
Starvation: Node s1_destroyed does not pull at line 14 column 2
Starvation: Node p1_buy does not push at line 39 column 2
Inactivity: Node doit does not trigger s2_destroyed at line 7 column 2
```

Initially puzzled by the first drought and the second starvation message, we concluded that the assertion in the monitor process is violated before the reachability check happens. Indeed node *p1_buy* never pushes, since it has no resource connections, it serves only to trigger choices.

The final message of inactivity tells us that *s2_destroyed* is never triggered by *doit*, the binding of *turn*. This experiment shows MM AiR provides feedback for analyzing and refining MM models intended to be embedded in game software.

3.5 CONCLUSION

Machinations was a great first step in turning industrial experience in game design into a design language for game economies. In this paper we have taken the original Machinations language as starting point and have analyzed and scrutinized it. It turned out that the definitions of various of the original language elements were incomplete or ambiguous and therefore not yet suitable for a formal analysis of game designs. During this exercise, we have learned quite a few lessons:

- Formal validation of rules for game economies is feasible.
- Unsurprisingly, modularity is a key feature also for a game design language. Modularity not only promotes design reuse, but also enables modular validation that can significantly reduce the state space.
- In our refinement and redefinition of various language features, we have observed that non-determinism had to be eliminated where possible in order to reduce the state space.
- While a graphical notation is good for adoption among game designers, a textual notation is better for tool builders.
- PROMELA is a flexible language that offers many features to represent the model to be validated. Different representation choices lead to vastly different performance of the model checker and it is non-trivial to choose the right representation for the problem at hand.

- The RASCAL language workbench turned out to be very suitable for the design and implementation of MM AiR. In addition to compiler-like operations like parsing and type checking MM AiR also offers editing, interactive error reporting and visualization. It also supports generation of PROMELA code that is shipped to the SPIN model checker and the resulting execution traces produced by SPIN can be imported and replayed in MM AiR.

MM AiR in its current form is an academic prototype, but it is also a first step towards creating embeddable libraries of reusable, validated, elements of game designs. Next steps include the use of probabilistic model checkers, mining of recurring patterns in game designs and finally designing and implementing embeddable APIs for MM. These will form the starting point for further empirical validation. We see as the major contributions of the current paper both the specific design and implementation of MM and MM AiR and the insight that the combination of state-of-the-art technologies for meta-programming and model checking provide the right tools to bring game design to the next level of productivity and quality.

Acknowledgements

We thank Joris Dormans for answering our many questions about Machinations, Tijs van der Storm for providing advice and feedback, and the anonymous reviewers for giving valuable suggestions.

Abstract

In early game development phases game designers adjust game rules in a rapid, iterative and flexible way. In later phases, when software prototypes are available, play testing provides more detailed feedback about player experience. More often than not, the realized and the intended gameplay emerging from game software differ. Unfortunately, adjusting it is hard because designers lack a means for efficiently defining, fine-tuning and balancing game mechanics.

The language *Machinations* provides a graphical notation for expressing the rules of game economies that fits with a designer's understanding and vocabulary, but is limited to design itself. *Micro-Machinations* (MM) formalizes the meaning of core language elements of *Machinations* enabling reasoning about alternative behaviors and assessing quality, making it also suitable for software.

We propose an approach for designing, embedding and adapting game mechanics iteratively in game software, and demonstrate how the game mechanics and the gameplay of a tower defense game can be easily changed and promptly play tested. The approach shows that MM enables the adaptability needed to reduce design iteration times, consequently increasing opportunities for quality improvements and reuse.

4.1 INTRODUCTION

In computer game development, developers face problems that arise from increased complexity. Challenges include increased development speed, changing technologies, and growing teams of experts with varying vocabularies. Teams may consist of artists, software engineers, domain experts and game designers who work together to create games. Practice is still catching up with the relatively new profession of the computer game designer. Game design, despite the large number of games produced today lacks a common vocabulary, methods for designing games and sharing knowledge and artifacts. The need for common design vocabularies [Chu99a; ZMF⁺05], game design patterns [BLHo3], specialized game grammars [Dor09; Koso5a], and computer assisted design tools [Nei12; NMo8a] has been expressed for some time, but so far no tool, method, or framework has surfaced as an industry standard. As result, game

This chapter was previously published as R. van Rozen and J. Dormans. "Adapting Game Mechanics with Micro-Machinations". In: *Proceedings of the 9th International Conference on the Foundations of Digital Games, FDG 2014, Liberty of the Seas, Caribbean, April 3-7, 2014*. Ed. by M. Mateas, T. Barnes, and I. Bogost. Society for the Advancement of the Science of Digital Games, 2014. ISBN: 978-0-9913982-2-5

design relies strongly on iterative prototyping, play-testing, and reprogramming parts to improve games.

Good gameplay is an emergent property of the game system defined by the game mechanics [Dor12a]. Therefore, gameplay can only be evaluated after the system has been built and set into motion through play. More often than not, the realized gameplay differs from the intended gameplay. Because a designer's understanding about how game rules affect the player is constantly changing, they have to make adaptations quickly and often. However, as software development progresses, making changes becomes harder and more time consuming. This seriously compromises the ability of the designer to design, play test, gain feedback and improve, which results in longer design iterations and missed opportunities for improving the quality. From a software engineering point-of-view gameplay adaptations represent a problematic stream of badly defined, poorly understood requirements that result in wasted effort, ineffective attempts at reuse, and repetitive and error-prone coding cycles, instead of focus on maintenance, libraries and tools.

We aim to accelerate the game development process by boosting game designer productivity and improving quality feedback. The language *Machinations* [AD12] provides a graphical notation for expressing the rules of game economies that is gaining popularity with designers. *Micro-Machinations* (MM) [KvR13] formalizes the meaning of core language features of *Machinations* and adds modularity, making it also suitable for software development and formal analysis. We propose a game design approach for adapting mechanics using MM, that aims for brief design iterations with informed and well-documented design decisions. The approach entails modeling game mechanics as embeddable software artifacts with MM. Additionally, it provides a means for adapting game mechanics at run-time using a library. We demonstrate that changes to the game economy of a tower defense game can be easily designed and embedded in software. Our approach shows that it is feasible to significantly reduce design iteration times, by improving flexibility and adaptability, thereby increasing opportunities for quality improvements and reuse.

4.2 BACKGROUND

4.2.1 *Related Work*

Language-oriented approaches for game development include Petri Nets [AR09; BA06], state machines [FHL07], and rule based and constraint based systems [MCS⁺04b; NMo8a]. Additionally, configuration files have been used for loading constants, and interpreters that provide run-time adaptability using scripts have been embedded in games, e.g. Lua [WSA⁺10] and Python [Daw02]. Commercial game engines also include script languages [UDNb] and graphical languages [UDNa]. Script languages are general purpose solutions that can be used for solving problems in many domains.

In contrast, a Domain-Specific Language (DSL) focuses on providing increased expressiveness over solutions for a smaller set of problems. We adopt the DSL definition of van Deursen et al. [vDKVoo] which states that:

“A Domain-Specific Language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain”.

DSLs have advantages over script languages such as improved productivity and quality, division of labor. However, DSL approaches also come at a cost. Time and effort goes into analyzing a domain, choosing appropriate notations and meanings, learning the language and maintaining it. In comparison, using an existing script language is technically easier to achieve, e.g. through its meta-programming capabilities Lua has been used for creating internal DSLs [KB13].

4.2.2 *Machinations Evolution*

We give a brief history of the evolution of the Machinations language, its tools and frameworks, from its inception for game design to its formalization and use in game software.

Game Design The original Machinations framework, intended solely for *game design*, helps designers to design, understand, and balance complex game systems [AD12; Dor12a]. Its starting point is the notion of internal economy for games [ARo6] that describes game dynamics in terms of distribution and flow of game resources. Game resources include tangible resources such as money, property, and food, but the concept also applies to abstract notions such as hit points, experience points, and strategic advantage. The framework uses a diagrammatic language to visualize a game’s internal economy. Machinations diagrams foreground the structural characteristics of the game economy that are critical in the emergence of gameplay. In particular dynamic gameplay can be attributed to feedback loops in the internal economy [ARo6; SZo3].

The framework augments paper prototyping, and can be used to assess game balance, emergent properties, and potential dominant strategies. Diagrams are abstract, dynamic, and playable representations of a game. Moreover, they are game design artifacts only, and not suitable as software requirements, since they lack a programmable semantics. Machinations is used in education and practice.

Formal Analysis Micro-Machinations (MM) is a DSL that we describe in Section 4.2.3 intended for both *game design* and *software development*. It is a formalized extended subset of Machinations which adds a precise meaning to the design notation, enabling

formal reasoning about alternative model behaviors and assessing their quality. MM also introduces new features, notably modularization, and has both a visual and a textual notation. Micro-Machinations Analysis in RASCAL (MM AiR) is a framework for analyzing MM that uses the RASCAL meta-programming language¹ and the SPIN model checker [KvR13]. It offers an IDE that reads textual MM and displays visual MM for simulating models interactively or randomly and analyzing behaviors partially or exhaustively. MM AiR is described in Chapter 3.

Software Development Thus far, MM have only been analyzed. Here, we introduce a library for building games with MM. The Micro-Machinations Library (MM Lib) is a light-weight software library written in C++ for embedding MM in games and tools². MM Lib tackles technical challenges related to interoperability, traceability and debugging. In particular, it enables embedding and adapting models in game software and replaying behavior traces.

The library interprets textual MM as changes that are reflected at run-time in the game economy state, enabling adapting model elements between evaluation steps. Developers can integrate MM Lib using its simple embedding APIs, most notably for evaluating model changes, activating nodes, stepping to a next state and informing its context about changes to type definitions and instances.

4.2.3 *Micro-Machinations*

MM models are directed graphs consisting of *nodes* and *edges*, which can be annotated with extra information. They describe the rules of internal game economies and define how resources are redistributed step by step between nodes. We provide a description of modeling elements needed for the case study of Section 4.4, which is based on prior work, described in Chapter 3. Chapter 3 gives a more detailed explanation that also includes textual models. Here, we use only the visual variant for conciseness. First, we explain basic language elements in Table 4.1 and Table 4.2. Next, we explain modular language elements. Table 4.3 introduces two new features for embedding instances of MM diagrams into games: *instance pools* and the *self pool*.

4.3 ADAPTING GAME MECHANICS

This section introduces a game design approach for designing, embedding and adapting game mechanics and gameplay of working software by embedding MM. Its goals are shortening design iterations, gaining immediate feedback and maximizing opportunities for quality improvements. We aim at flexible integration in processes

¹<http://www.rascal-mpl.org> (visited September 1st 2019)

²<https://github.com/vrozen/MM-Lib> (visited September 1st 2019)

Table 4.1: Cheat sheet for basic Micro-Machinations language elements (part 1)



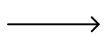







 Empty pool	 Pool and resource	<p>A <i>pool</i> is a named node, that abstracts from an in-game entity, and can contain <i>resources</i>, such as coins, crystals, health, etc. Visually, a pool is a circle with an integer in it representing the current amount of resources, and the initial amount at which a pool starts when first modeled.</p>	
 Resource connection with flow rate of one	 Resource connection with flow expression	<p>A <i>resource connection</i> is an edge with an associated expression that defines the rate at which resources can flow between source and target nodes. During each transition or <i>step</i>, nodes can <i>act</i> once by redistributing resources along the resource connections of the model. The <i>inputs</i> of a node are resource connections whose arrowhead points to that node, and its <i>outputs</i> are those pointing away.</p>	
 Automatic pool	 Passive pool	 Interactive pool	<p>The <i>activation modifier</i> determines if a node can act. By default, nodes are <i>passive</i> (no symbol) and do not act unless activated by another node. <i>Interactive</i> (double line) nodes signify user actions that during a step can activate a node to act in the next state. <i>Automatic</i> (*) nodes act automatically, once every step.</p>
 Pool with push act and any modifier	 Pool with pull act and all modifier	<p>Nodes act either by pulling (default, no symbol) resources along their inputs or pushing (p) resources along their outputs. Nodes that have the <i>any modifier</i> (default, no symbol), interpret the flow rate expressions of their resource connections as upper bounds, and move as many resources as possible. Additionally, these nodes may process their resource connections independently and in any order. Nodes that instead have the <i>all modifier</i> (&) interpret them as strict requirements, and the associated flows all happen or none do.</p>	
 Pool with push act and all modifier			

Table 4.2: Cheat sheet for basic Micro-Machinations language elements (part 2)


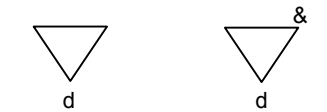

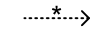
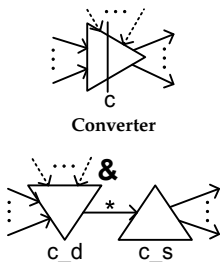
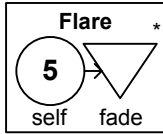
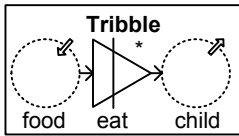
 <p style="text-align: center;">s Source</p>	<p>A <i>source</i> node, appearing as a triangle pointing up, is the only element that can generate resources. A source can be thought of as a pool with an infinite amount of resources, and therefore always pushes all resources or all resources are pulled from it. The any modifier does not apply, and resources may never flow into a source. Also, infinite amounts may not flow from sources during a step.</p>
 <p style="text-align: center;">d Drain with any modifier</p> <p style="text-align: center;">d & Drain with all modifier</p>	<p>A <i>drain</i> node, appearing as a triangle pointing down, is the only element that can delete resources. Drains can be thought of as pools with an infinite negative amount of resources, and have capacity to pull whatever resources are available, or whatever resources are pushed into them. No resources can ever flow from a drain.</p>
 <p style="text-align: center;">==1 Condition edge with equals one expr</p> <p style="text-align: center;">>=2 Condition edge with greater equals expr</p>	<p>A node can only be active if all of its <i>conditions</i> are true. A <i>condition</i> is an edge appearing as a dashed arrow with an associated Boolean expression. Its source node is a pool that forms an implicit argument in the expression, and the condition applies to the target node.</p>
 <p style="text-align: center;">* Trigger edge</p>	<p>A <i>trigger</i> is an edge that appears as a dashed arrow with a multiply sign. The origin node of a trigger activates the target node when for each resource connection the source works on, there is a flow in the transition that is greater or equal to that of the associated flow rate expression. Additionally, automatic pulling nodes without inputs and automatic pushing nodes without outputs always activate their trigger targets.</p>
 <p style="text-align: center;">c Converter</p> <p style="text-align: center;">& c_d * c_s Desugared converter</p>	<p><i>Converters</i> are nodes, appearing as a triangle pointing right with a vertical line through the middle, that consume one kind of resources and produce another.</p> <p>Converters are not core elements because they can be rewritten as a combination of a drain, a trigger and a source. Unlike basic node types, converters therefore take two steps to complete. Converters always implicitly have pull and all modifiers.</p>

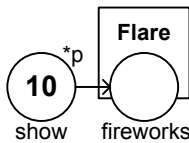
Table 4.3: Cheat sheet for modular Micro-Machinations language elements



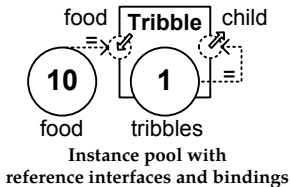
Definition containing a self pool



Definition containing references with in and out modifiers



Instance pool without interfaces



Instance pool with reference interfaces and bindings

A *type definition* is a named diagram that functions as parameterized module for encapsulating elements. Type definitions define internal elements and how they can be used externally. A *reference*, represented by a circle with a dashed line, is an alias that refers to a node that defines it. Internal nodes annotated with an *interface modifier* *input*, *output* or *input/output* become interfaces on the instances of the type. The input modifier denotes that an interface accepts inputs, output implies it accepts outputs and input/output accepts both. Interface modifiers appear as an arrow in the top right corner of a node, where an input modifier point into the node, an output modifier points out of the node, and an in-/output modifier does both.

An *instance pool* is a pool whose resource type is a definition. It represents a set of instances, objects with individual instance data, whose shared interfaces are defined by that type, and can be bound to other models, acting as formal parameters. Additionally, the size of the set is the amount of resources in the pool. Visually, an instance pool appears as a circle and a rectangle. Instances are local to a pool and cannot flow out through resource edges. Resources flowing in create new instances, and those flowing out delete them. An *interface* makes internal elements of instances available to the outside, and can be used by connecting resource connections. Visually, an interface is a small circle at the border of an instance with its name under it. Input interfaces have an arrow pointing into the circle, outputs have an arrow pointing outward, and in-/outputs have a bidirectional arrow. The direction of the arrow implies the validity of the direction of the resource edges that connect to it. Only reference interfaces appear with a dashed line. References must be bound to definitions using edges called *bindings*, represented by dashed arrows annotated with an equal sign, that originate from a defining node and target a reference. When a type definition contains a pool named *self*, instances of this type end when the pool is empty.

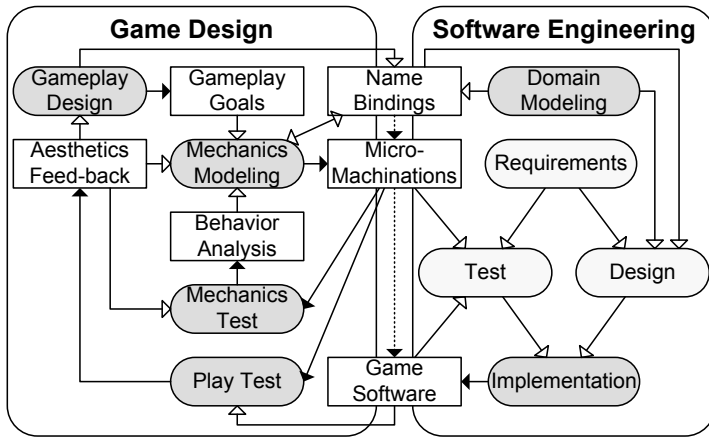


Figure 4.1: Game design approach for adapting mechanics

Table 4.4: Disciplines, artifacts and expected iteration activity when using the Micro-Machinations approach

occupation	discipline	main artifact	expected iteration activity level			
			concept	elaboration	construction	delivery
game design	gameplay design	gameplay goals	high	high	medium	low
	mechanics modeling	mechanics model	medium	high	high	high
	play test	aesthetics feed-back	medium	high	high	high
	mechanics test	behavior analysis	low	low	low	low
software engineering	domain modeling	name bindings	low	high	low	low
	implementation	game software	low	medium	high	low

that employ iterative development, agile practices and Scrum. The approach separates concerns, dividing the work between game designers and software engineers. Because a game designer’s creative genius flourishes with expressive freedom, flexibly switching between activities, we describe only *what* artifacts game designers and software engineers create, but not *when*. Working on disciplines and artifacts is *optional* at each given time.

Figure 4.1 shows disciplines (rounded rectangles) and artifacts (rectangles). Closed arrow heads denote an artifact is required for activities related to a discipline, whereas open arrow heads signify optional artifacts that provide added value. Table 4.4 describes expected iteration activity during phases of the project life cycle. These

phases are 1) *concept*: when a game is conceived of and its feasibility is discussed, 2) *elaboration*: when plans are made concrete using models and diagrams documenting intended functionality, 3) *construction*: when game software is built, tested, balanced and fine-tuned, and 4) *delivery* when a product is prepared for its release. We explain the design disciplines one by one.

- **Gameplay Design:** *How should the game affect the player experience during play?* Designers describe the intended effect of rules on players [SZ03] in what we call *gameplay goals*. Gameplay design includes taking *design decisions* concerning patterns, intended behavior and feedback loops. Activities may include paper prototyping and using other conceptual game design tools for assisting in the creative process, e.g. the Machinations tool [Dorog].
- **Mechanics Modeling:** *What are the rules composing the mechanics?* The approach centers around the discipline of *mechanics modeling*, which involves designing an artifact using MM that we call the *mechanics model*. It describes the rules of the game, that when set in motion by run-time and player interaction aims to achieve the gameplay goals. The mechanics model is an embeddable *software artifact* that interacts with other parts of the software using its *name bindings*. It includes descriptions that relate design decisions and diagram elements to expected behavior and player interaction. Designers can add, change, balance and fine-tune mechanics at any development stage.
- **Play Test:** *How do the rules affect player experience?* During play testing designers validate if the mechanics model achieves the gameplay goals, gathering *aesthetics feedback* in order to make improvements. Traditionally play testing happens on paper prototypes in early stages. Mechanics models enable play testing without game software, but with the guarantee the models behave the same when embedded in games. Later, when game software is available play testing becomes less abstract. This approach enables play testing effectively throughout the development process.
- **Mechanics Test:** *How do the rules behave, and how can that be improved?* Mechanics models are not just embeddable in games. Tools such as MM AiR described in Section 4.2.2 can also interpret MM, enabling designers to analyze the behavior separately. Applications include interactively stepping through states and transitions, analyzing alternatives, programmable test setups prerecorded or programmed player actions for random simulation runs or exhaustive analysis, and reproducing states and transitions using MM traces, as in see Chapter 3.

For integrating the design and software engineering processes we relate shared artifacts to the software engineering disciplines of domain modeling and implementation.

- **Domain Modeling:** *What are the name bindings for embedding the mechanics in game software?* During domain analysis software engineers analyze and visualize important concepts and relationships, and agree with designers on concept



Figure 4.2: Screenshot of the running AdapTower prototype

names and activation points in a contract that we call *name bindings*. Changes to name bindings may cost time because they affect the software design, and integration points must be coded.

- **Implementation:** *How are the game mechanics integrated in software?* Software engineers build *game software* glue libraries, program integration points specified in the name bindings, and integrate content and mechanics models with the rest of the game system. The MM Lib, introduced in Section 4.2.2, enables embedding the mechanics model. As soon as the first prototype runs the model, play testing can commence.

4.4 CASE STUDY: ADAPTOWER

4.4.1 Experimental setup

We demonstrate adaptability of the mechanics and gameplay of *AdapTower*, a prototype tower defense game built using the approach discussed in Section 4.3. The game, shown in Figure 4.2, is implemented in C# and embeds the MM Lib (which is C++) using a wrapper that marshals data to .NET.

Figure 4.3(a) shows a partition of AdapTower in software layers. Players trigger mechanics in the User Interface (UI) layer. The Physics layer interprets user actions and tracks in-game entities, their location, speed and trajectories. It ensures the UI

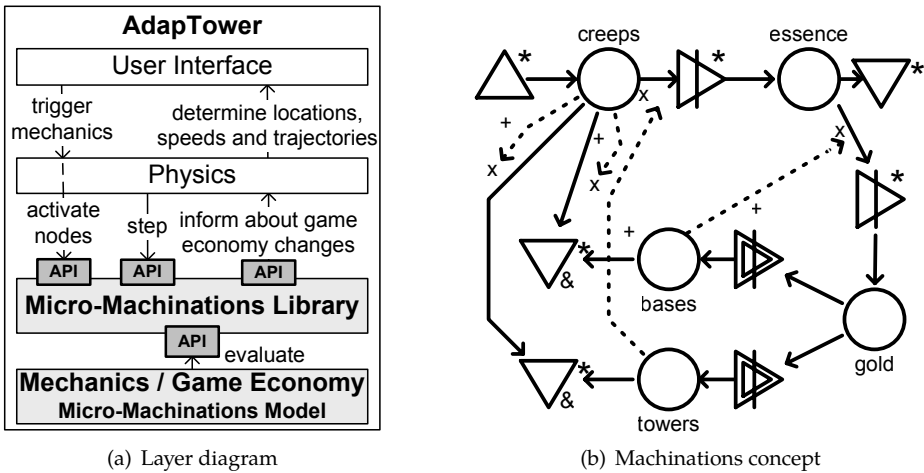


Figure 4.3: AdapTower diagrams

displays them accordingly. MM Lib interprets Physics calls to activate nodes, e.g. for collisions and time passing, and evaluates textual MM defining adaptations. MM Lib computes a next game economy state when the Physics calls the *step* API, and informs it about changes to definitions and instances with callbacks and messages.

4.4.2 Adapting AdapTower

Here we demonstrate adaptations to the game economy of AdapTower in a series of six design iterations. We provide visual MM definitions with additions and changes³.

Design Iteration 0: Concept phase

Gameplay Design. In AdapTower, the *creeps* spawn at random locations on the top of the screen and march downwards. Defensive *towers* shoot creeps and convert killed creeps into *essence*, a resource that falls down. *Bases* collect essence and convert it into *gold*, which can be used to build more towers and bases. Both are destroyed when they come into contact with the advancing creeps. To reach the objective of building a fixed number of bases the player needs to construct defensive configurations that minimize the risk of losing bases, but maximize the collection of essence. AdapTower’s internal economy consists of two interconnected positive feedback loops. First, towers convert creeps into essence and bases convert essence into gold, which players use to buy more towers and bases. Second, the more creeps there are, the more likely it is they

³The textual MM of AdapTower can be found at <https://github.com/vrozen/MM-Lib/tree/master/mm/tests/towers>

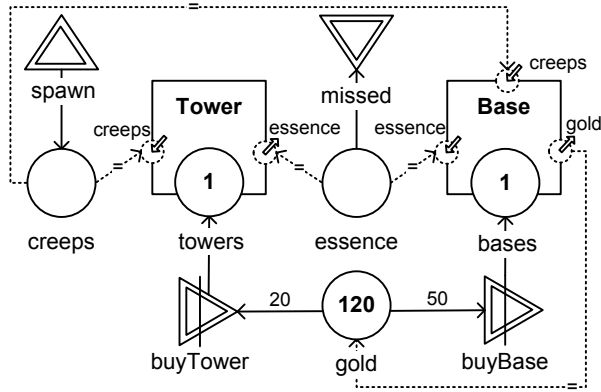


Figure 4.4: Definition of AdapTower in visual Micro-Machinations

Table 4.5: Global name bindings

name	meaning	embedding
spawn	a creep enters the world	activate node
creeps	amount of creeps in the world	notify value
essence	amount of essence in the world	notify value
missed	essence leaves the world	activate node
towers	amount of towers in the world	notify new/del
bases	amount of bases in the world	notify new/del
gold	amount of gold the player has	notify value
buyTower	player buys a tower	activate node
buyBase	player buys a base	activate node

collide and destroy more towers, meaning more creeps will survive. Figure 4.3(b) shows a concept sketch modeled with the Machinations tool [AD12; Dor09].

Design Iteration 1: Creeps, towers and bases

Mechanics Modeling. The first mechanics model version of the game economy of AdapTower consists of three MM models. The integrated game is modeled in Figure 4.4. We model *creeps*, *essence* and *gold* by pools, which are bound to *Tower* and *Base* instances on their shared interfaces using binding edges. Creeps enter the world by externally activating the interactive source *spawn* which pushes one resource along its edge to the pool *creeps*. The drain *missed* models *essence* disappearing from the world without being caught. The converters *buyTower* and *buyBase* consume 20 and 50 gold to respectively produce a tower and a base instance.

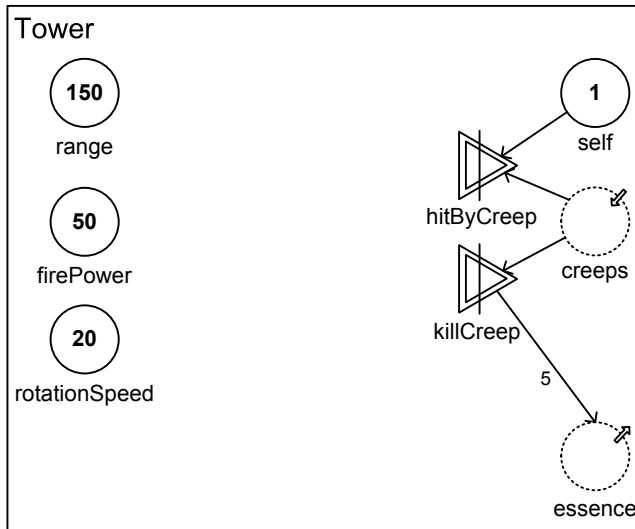


Figure 4.5: First version of the Tower definition

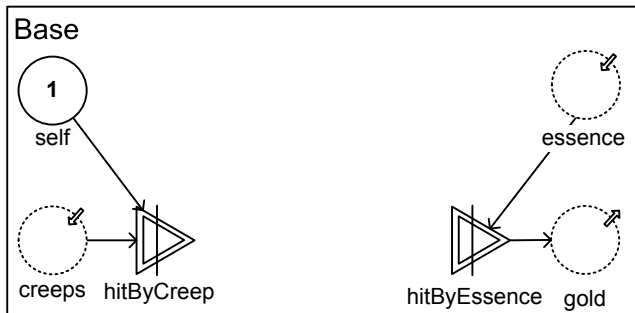


Figure 4.6: First version of the Base definition

Table 4.6: Name bindings of Tower and Base type definitions

name	meaning	embedding
range	tower range in game yards	notify value
firePower	tower fire power in hit points	notify value
rotationSpeed	tower rotation speed degree/s	notify value
hitByCreep	physics: a creep hits a tower	activate node
killCreep	physics: a tower kills a creep	activate node
hitByCreep	physics: a creep hits a base	activate node
hitByEssence	physics: essence hits a base	activate node

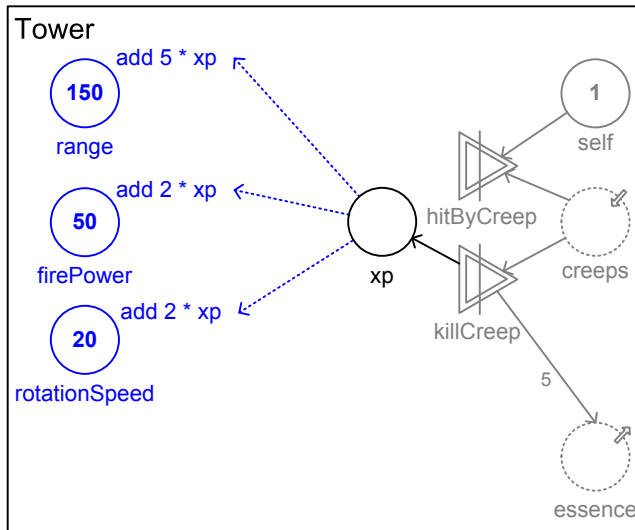


Figure 4.7: Second version of the Tower definition

Figure 4.5 models the first *Tower* type definition. Each tower has a *range* of 150, a *firePower* of 50 and a *rotationSpeed* of 20. When activated, converter *killCreep* pulls one resource from *creeps*, and produces five resources in *essence*. Figure 4.6 models the first *Base* type definition. When activated, converter *hitByEssence* pulls one resource from *essence* and generates one resource in *gold*. *Tower* and *Base* both contain a converter *hitByCreep*, which pulls one resource from external node *creeps* and one from *self*, collapsing the instance.

Domain Modeling. Table 4.5 and Table 4.6 show name bindings for embedding versions of the mechanics. Some interactive nodes are activated externally using their names, such as *spawn*, *missed*, *killCreep*, *hitByCreep*, and *hitByEssence*. Others, such as *buyTower* and *buyBase* are UI elements activated by the player. For of passive pools, such as *creeps*, *essence*, *towers*, *bases*, *gold*, *range*, *firePower* and *rotationSpeed* the game registers observers for using current values.

Play Test. The gameplay of the initial version works, but is not very exciting. Once players set up their bases and defenses, they simply need to wait and collect enough essence for quickly building the number of bases required to win.

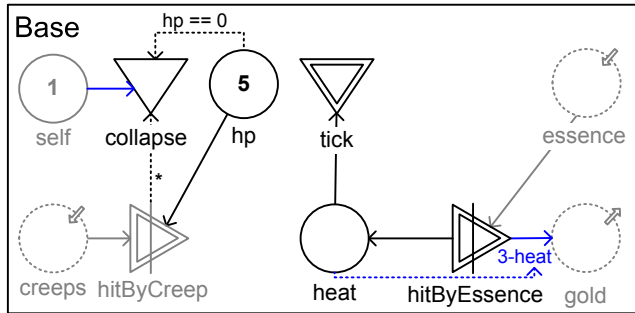


Figure 4.8: Second version of the Base definition

Design Iteration 2: Towers gain experience

Mechanics Modeling. We adapt the Tower definition in Figure 4.7 by adding an experience pool xp and by changing the pools $range$, $firePower$ and $rotationSpeed$, adding a bonus based on xp .

Gameplay Design. The rationale behind this change is that adding another feedback loop improves positioning effectively towers and speeds up the end game.

Design Iteration 3: Bases have health

Mechanics Modeling. We adapt the Base definition in Figure 4.8 by adding a pool hp that denotes hit points, starting at five resources. A resource is drained every time a creep hits the base, but bases only *collapse* when hp is empty. Additionally, bases gain heat in a pool called $heat$ every time it is hit by essence. Heat inhibits the conversion from essence to gold. This change is represented by the modified flow from converter $hitByEssence$ to reference $gold$. Heat diminishes over time, since every time drain $tick$ activates, it pulls one resource from pool $heat$.

Gameplay Design. Hit points make bases more stable, whilst the heat mechanism introduces a negative feedback loop that diminishes the effectiveness of bases collecting a lot of essence. These changes aim to force players to spread bases for collection resources, and reduce the effectiveness of funnelling all essence and creeps into a single place.

Design Iteration 4: Towers lose experience

Gameplay Design. We introduce two feedback loops to make towers more dynamic. The goal is that towers accumulate experience points much faster, but also lose them over time. At the same time the experience points inhibit the number of essence

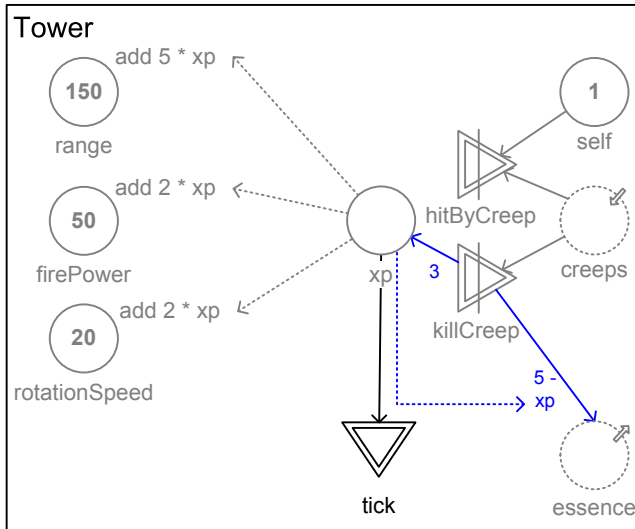


Figure 4.9: Third version of the Tower definition

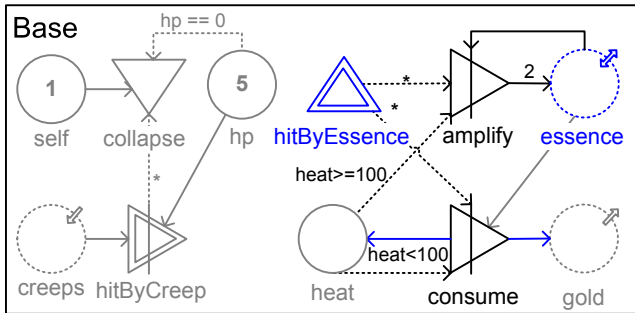


Figure 4.10: Third version of the Base definition

produced by each kill. The effect is that towers can go into a sort of “*killing spree*”, but these towers essence production is reduced at the same time.

Mechanics Modeling. We realize these effects by increasing the flow from converter *killCreep* to *xp* and adding a drain *tick* that pulls resources from pool *xp* in Figure 4.9. Additionally, the amount of generated essence is reduced by changing the flow expression to $5 - xp$.

Domain Modeling. The drain *tick* specifies the name of a node the game must activate once each second.

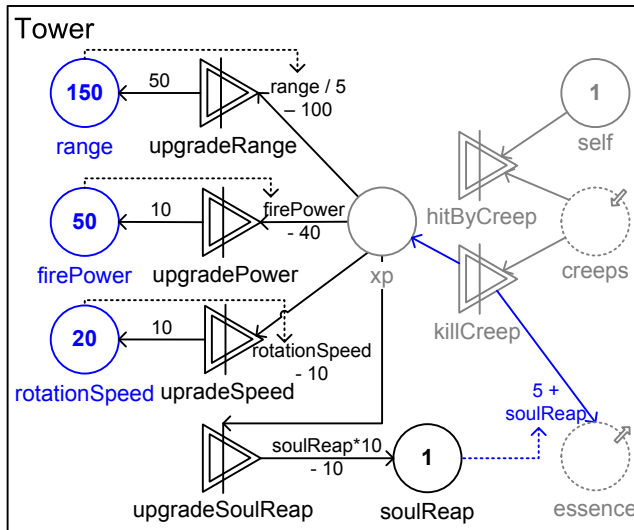


Figure 4.11: Fourth version of the Tower definition

Design Iteration 5: Bases amplify essence

Play Test. One problem we notice is that the player is not encouraged to place bases at the top of the screen.

Gameplay Design. To make positioning bases more interesting we introduce an ageing mechanism. A base ages for every essence it converts into gold. Once a base has produced 100 gold it evolves into a new mode, duplicating essence instead. Because essence falls down, it is more effective to place newer bases below older bases, and this creates a high-risk reward strategy that encourages players to build their first bases in relatively exposed positions.

Mechanics Modeling. Figure 4.10 shows the third Base definition. We delete drain tick, reversing the decision of draining heat over time. We change *hitByEssence* into a drain and reference *essence* gains an output modifier. We add converters *amplify* and *consume* and triggers to activate them from *hitByEssence*. Consume returns one gold and one heat for each essence when heat is less than 100, but otherwise amplify instead returns two essence resources.

Design Iteration 6: Towers are upgradeable

Play Test. We see the problem that all towers act alike.

Gameplay Design. We try to overcome this by allowing players to choose different possible upgrades for their towers, and specialize individual towers in different ways.

Table 4.7: Additional name bindings for the Tower definition

name	meaning	embedding
upgradeRange	upgrade range	activate node
upgradePower	upgrade power	activate node
upgradeSpeed	upgrade rotation speed	activate node
upgradeSoulReap	upgrade essence efficiency	activate node

Mechanics Modeling. The final tower model, shown in Figure 4.11, adds a new pool called *soulReap*, and increases the amount of essence for killing a creep to $5+soulReap$. Additionally, users can upgrade *range*, *firePower* and *rotationSpeed* and *soulReap* by respectively activating converters *upgradeRange*, *upgradePower*, *upgradeSpeed* and *upgradeSoulReap* that require more *xp* each upgrade.

Domain Modeling. Previous game adaptations only involved mechanics modeling, but choosing different tower upgrades is a feature not previously agreed upon. The MM Lib lets games observe type changes, enabling us to meta-program the handy default of adding new interactive nodes as clickable names near UI elements. In this case the game adds different upgrades near towers, and infers the name bindings of Table 4.7. This way, designers can continue mechanics modeling and play testing.

4.4.3 Experimental results

We showed that after the AdapTower prototype was built, its game economy could be modified using MM, with a flexibility reminiscent of the concept phase. Because we expect continuously high activity levels of mechanics modeling and play testing, as shown in Table 4.4, the approach is especially useful in later project stages. It allowed us to iterate and explore the design quickly, providing us with time to experiment and improve the game. Each iteration took us about 15 minutes to design gameplay and to model additions and changes of the mechanics in textual MM. We play tested briefly, and confirmed that we achieved the emergence we were looking for. Additionally, we spent only a fraction of the implementation time on the name bindings that integrate the model.

4.5 CONCLUSION

We proposed a game design approach for adapting game mechanics with MM. We applied the approach and implemented AdapTower, a prototype tower defense game. We demonstrated that its mechanics and emergent gameplay could be modified after its construction. We showed that MM is a suitable notation for making adaptations.

We argued that structured and clear artifacts and better, faster feedback about gameplay changes improve designer productivity. Our approach showed that it

is feasible to significantly reduce design iteration times and accelerate the game development process by improving the adaptability, thereby increasing opportunities for quality improvements and reuse. We see our contributions as an important step towards practical game design methods for producing game software. However, our experience concerns a prototype and the technology still needs to mature before industry can adopt this approach.

4.5.1 *Future Work*

Future work entails the following:

- *Game design tools* can embed MM Lib and show editable visual MM, and send textual MM changes to adapt games like we proposed, e.g., graphical debuggers for tracing and replaying behavior, and integrated analysis tools like MM AiR. We envision a tool-of-tools approach, using language work-benches and meta-programming, for tailoring tools towards games.
- *Validation* requires that we apply tools and libraries in industrial case studies. Such experiences provide valuable information on the usage of language and tool features, and for making improvements.
- Comparable, reusable, analyzable game designs enable *pattern mining*. To do this, we first require a corpus of MM models to find empirical evidence of patterns, and an assessment of their gameplay qualities. Then, we can extract and compare them, e.g., for providing tools for mechanics comparison and prototype generation. Using extracted domain knowledge can also augment evolutionary techniques for mining patterns [CCR⁺13].
- MM do not necessarily describe the full run-time behavior of a game system, which limits its analysis. Productivity and quality can be further improved by integrating MM with other DSLs, e.g., for locations and speed, and story-lines. Additionally, more accurate behavior predictions enable preventing bugs.
- Combining player satisfaction information with adaptable mechanics enables tailoring mechanics to the skill and enjoyment of individual players.

Acknowledgements

We thank Paul Klint for proof reading this paper, and the anonymous reviewers for their insightful comments.

Abstract

Video game designers iteratively improve player experience by play testing game software and adjusting its design. Deciding how to improve gameplay is difficult and time-consuming because designers lack an effective means for exploring decision alternatives and modifying a game’s mechanics.

We aim to improve designer productivity and game quality by providing tools that speed-up the game design process. In particular, we wish to learn how patterns encoding common game design knowledge can help to improve design tools. Micro-Machinations (MM) is a language and software library that enables game designers to modify a game’s mechanics at run-time.

We propose a pattern-based approach for leveraging high-level design knowledge and facilitating the game design process with a *game design assistant*. We present the Mechanics Pattern Language (MPL) for encoding common MM structures and design intent, and a Mechanics Design Assistant (MeDeA) for analyzing, explaining and understanding existing mechanics, and generating, filtering, exploring and applying design alternatives for modifying mechanics. We implement MPL and MeDeA in RASCAL, and evaluate them by modifying the mechanics of a prototype of Johnny Jetstream, a 2D shooter developed at IC3D Media.

5.1 INTRODUCTION

Video game designers work to improve player experience by iteratively play testing game software and adjusting the game’s design. Improving gameplay is difficult and time-consuming because designers lack an effective means for quickly exploring design alternatives and modifying a game’s mechanics. Specifically, designers cannot efficiently 1) author, modify and fine-tune mechanics—rules of games that influence gameplay—without help of programmers; 2) receive immediate feedback for comparing design intent to change result; and 3) explore decision alternatives efficiently. Since making well-informed design decisions is costly due to long iteration times, play testing is limited to fewer software versions than necessary for achieving the best game quality.

This chapter was previously published as R. van Rozen. “A Pattern-Based Game Mechanics Design Assistant”. In: *Proceedings of the 10th International Conference on the Foundations of Digital Games, FDG 2015, Pacific Grove, CA, USA, June 22–25, 2015*. Ed. by J. P. Zagal, E. MacCallum-Stewart, and J. Togelius. Society for the Advancement of the Science of Digital Games, 2015

We aim to improve designer productivity and game quality by providing tools that speed-up the game design process. In particular, we wish to learn how patterns encoding common game design knowledge can help to improve such tools.

Micro-Machinations (MM) is a visual language and software library that facilitates brief design iterations by enabling game designers to modify a game’s mechanics while it is running, and to play test simultaneously [KvR13; vRD14]. MM programs can be represented as visual diagrams that are directed graphs describing game-economic mechanics using various kinds of nodes and edges. A diagram works inside a game, steering its mechanics, and when set in motion through run-time and player interaction, the nodes redistribute resources along the edges between the nodes. Understanding the dynamics of diagrams is hard because designers combine elements in non-trivial ways, expressing design intent by providing choices, challenges, trade-offs and strategies for interesting gameplay. Adams and Dormans have suggested patterns for MM’s evolutionary predecessor Machinations, as a mental framework for understanding, explaining and designing game mechanics, and they provide a mechanics design rationale with example diagrams [AD12].

We propose a pattern-based approach that assists designers in the discipline of modeling mechanics for modifying game software. We define parameterized micro-mechanism patterns (*patterns* for short) that capture a wide range of diagrams with shared structures and design intent. We provide a Mechanics Pattern Language (MPL) for programming patterns and a Game Mechanics Design Assistant (MeDeA) that recognizes patterns in existing diagrams, explains intent of design choices in understandable text and enables interactively and visually exploring design choices that can be step-by-step filtered and applied yielding new diagrams and software versions. We contribute the following:

- A Mechanics Pattern Language (MPL) for programming patterns that capture a wide range of diagrams with shared structures and design intent.
- A Mechanics Design Assistant (MeDeA) implemented in the RASCAL meta-programming language for analyzing, explaining and understanding existing mechanics and generating, filtering, exploring and applying design alternatives for modifying mechanics.

5.2 RELATED WORK

We relate our work to languages, game design patterns and design tools providing analysis or procedural generation techniques. Figure 5.1 schematically shows how a designer modifies mechanics during play testing for improving the gameplay of a working game system, which includes both players and software. Before we elaborate on our approach for improving designer productivity we first sketch the research context.

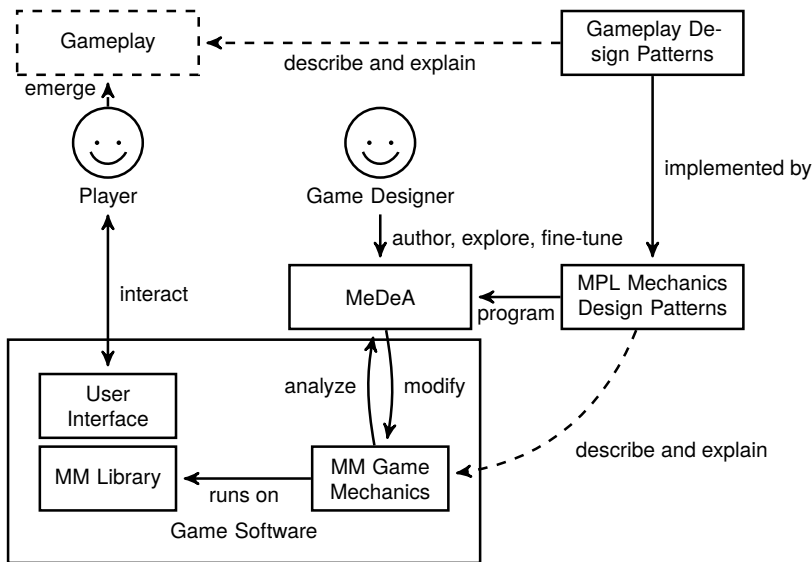


Figure 5.1: Relating design patterns to a game system

Languages

Game designers have expressed the need for a common game design vocabulary [Chug9a]. Languages and tools are necessary for describing, communicating and improving game designs using both mental frameworks and authoring tools for constructing parts of game software. We relate our perspective to the Mechanics Dynamics and Aesthetics (MDA¹) framework of Hunicke et al. who describe an approach to game design and game research [HLZ04]. As shown in Figure 5.1, we view mechanics as part of the game software, dynamics as its run-time and player interaction, and aesthetics as good player experience (gameplay) that emerges.

MeDeA is an authoring tool that uses Micro-Machinations (MM), a language for game-economic mechanics described by Klint, van Rozen and Dormans [KvR13; vRD14]. MM is an evolutionary successor of Machinations, a mental framework for understanding the effect of game mechanics on gameplay introduced by Adams and Dormans [AD12]. MM and MPL are so-called Domain-Specific Languages (DSLs) [vDKV00], declarative languages that offer designers expressive power focussed specifically on game-economic mechanics [KvR13]. Domains other than game development have benefited from DSLs and seen substantial gains in productivity and software quality at the cost of maintenance and education of DSL users [vDKV00].

¹Not to be confused with the Model-Driven Architecture (MDA), a software design approach for developing software systems by the Object Management Group (OMG) and a proprietary form of Model Driven Engineering (MDE).

Table 5.1: Game Description languages and tools for authoring, analyzing and modifying different game facets

scope/concern	language	tool/system	goals/functionality
“2D games” (research platform)	PyVGDL [Sch13]	PyVGDL lib.	Design games and analyze dynamics and learning algorithms
board games	GDL / Ludemes [BM10]	Ludi system	Playing, measuring, and synthesizing board games
mechanics of “board-games”	Prolog subset [SNM09a]	BIPED	Prototype complete board-like games and analyze their dynamics
mechanics (avatar-centric)	PDDL subset [ZR14a]	Generator sys.	Generate playable mechanics using a constraint solver+planner
mechanics (game-economic)	Machinations [AD12]	Flash Tool	Conceptually analyze mechanics designs (not software artefacts)
	MM, MPL [this chapter]	MeDeA	Statically analyze & modify embedded mechanics using patterns
discrete domain games	Gamelan [OGM13]	“Tool set”	Analyze Gamelan game dynamics against Computational critics
stories of Zelda- like 2D RPGs	Plot points seq. [HZD ⁺ 11]	GAME FORGE	Generate & render playable game world & story configurations
stories in interactive drama	Praxis [ES14]	Prompter	Author & analyze stories in agent-based interactive drama

Unlike ontologies [ZMF⁺05] or mental frameworks that focus on analyzing and understanding games, DSLs also serve to modify software systems.

Patterns

Kreimeier made a general case for game design patterns [Kre02]. Design patterns have been used to describe recurring patterns of gameplay [BLH03] and game mechanics [AD12]. Björk et al. propose game design patterns for analyzing and describing gameplay that enable understanding and explaining games and how they relate, and together form a game design body of knowledge [BLH03]. We view these patterns as *gameplay design patterns*, which can be used to understand what gameplay goals a completed game has, or should have while play testing during a game’s construction.

In contrast, *game mechanics patterns* can be used to describe how a game’s mechanics work before it is built, and to modify them afterwards, offering a way to achieve gameplay goals. MPL can be used to model mechanics design patterns and MM offers a way to implement a game’s mechanics aimed at improving gameplay as shown in Figure 5.1. We view both pattern kinds as complementary, and discovering relationships between them as an exciting direction of future research. Unlike gameplay design patterns, MeDeA facilitates understanding mechanics by recognizing

and visualizing patterns that run in software, and offering high-level explanations about their dynamics.

Tools

Game Description Languages (GDLs) are DSLs for the game domain. GDLs, tools and libraries have been described that offer game designers affordances for authoring, analyzing, understanding and modifying different game facets, e.g., measuring a game’s qualities or by procedurally and/or iteratively generating content in a mixed-initiative way [SLH⁺11] or for automatic improvements. Comparing GDLs is hard because their goals, tools and notations differ, and their expressive power over specific or general abstractions and behaviors for achieving novel gameplay goals vary greatly. Therefore their effectiveness and usage scenarios range from game- or genre-specific, to more general and abstract. Table 5.1 shows a small list of representative GDLs. Disciplines include modeling *mechanics* of game-economies [KvR13; vRD14], avatar acts [ZR14a] and board games [BM10; SNM09a] but also *levels* [SLH⁺11], *missions* and *stories* of e.g., Zelda-like 2D games [HZD⁺11] or interactive drama [ES14].

Declarative textual notations have been proposed for specific classes of games. Nelson an Mateas’ game design assistant facilitates iteratively prototyping micro-games with stock mechanics by authoring nouns, verbs and constraints [NM08a]. Smith et al. propose prototyping abstract board-like games in a lightweight logic-based sketching language that models game state and mutation events on the BIPED system, which integrates the logical LUDOCORE engine and produces gameplay traces using answer-set-programming [SNM09a]. Browne and Maire demonstrate the feasibility of evolutionary techniques for closed-system combinatorial board games by measuring specific playability qualities in Ludi [BM10]. In contrast, Osborn et al. argue for a more general procedural language. They propose Gamelan for games over discrete domains (e.g., board, card games), and modular computational critics that can measure rule coverage, turn-taking fairness and existence of uninteresting strategies [OGM13]. Evans and Short describe the Versu storytelling system, which is specific for interactive drama but also more generally reusable. Their Praxis language describes autonomous agents and social practices, and Prompter is a tool for debugging emergent stories [ES14].

Micro-Machinations

MM is a visual and textual DSL aimed at reuse that raises the abstraction level above that of game- or genre-specific, tightly integrated languages. MM separates the concern of game economies and supports both closed systems and integration into larger systems composed of parts. In prior work, MM was analyzed against invariants using model-checking [KvR13], and used to rapidly modify the mechanics



Figure 5.2: Screenshot of a Johnny Jetstream prototype that embeds the Micro-Machinations library

of a tower defense game iteratively at run-time using the embeddable reusable MM library² [vRD14]. MeDeA extends MM’s tool set and the spectrum of mixed-initiative interactive game design tools with a novel and general pattern-based approach for deciding how to improve a game’s mechanics, offering alternatives designers might otherwise overlook.

5.3 MECHANICS DESIGN ASSISTANT

We present a Game Mechanics Design Assistant (MeDeA) and evaluate it using game mechanics of Johnny Jetstream (JJ), a 2D side-scrolling shooter developed at Dutch game business IC3D Media that embeds MM. Figure 5.2 shows a screenshot of JJ. MM programs can be represented as visual diagrams which are directed graphs that consist of two kinds of elements, *nodes* and *edges*. Both may be annotated with extra textual or visual information. These elements describe the rules of internal game economies, and define how *resources* are step-by-step propagated and redistributed through the graph. A diagram works inside a game, controlling its internal economy, and when set in motion through run-time and player interaction, the nodes redistribute resources along the edges through the graph.

Figure 5.3 shows a sample MM implementation part of JJ that is hard to read for novices. This is our running example. We introduce MM and the Mechanics

²<https://github.com/vrozen/MM-Lib> (visited September 1st 2019)

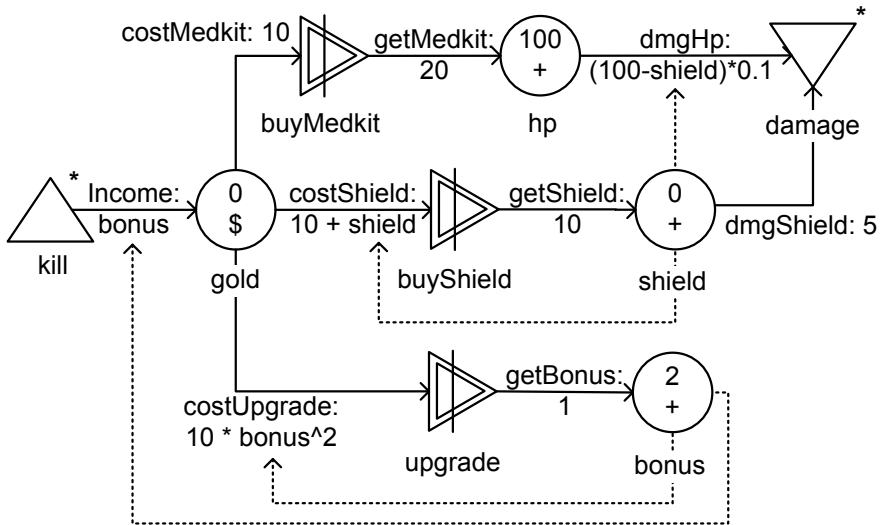
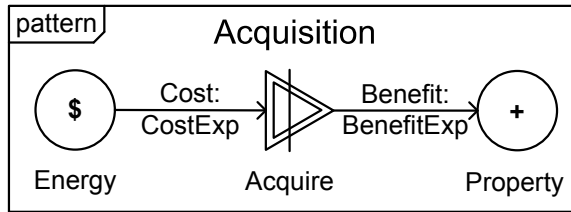


Figure 5.3: Micro-Machinations diagram of example mechanics of Johnny Jetstream

Pattern Language (MPL) by analyzing it against patterns with MeDeA. The tool explains the design by generating explanations from patterns and templates explained in Section 5.3.1. Next, we describe the mixed-initiative decision making process of MeDeA in Section 5.3.2 and show how MeDeA also assists in authoring the example using the same patterns in Section 5.3.3. We sketch MeDeA’s architecture in Section 5.3.4.

5.3.1 Pattern-Based Analysis and Explanation

We analyze and explain example mechanics of JJ using a collection of patterns we call our *pattern palette*. Although usually such a collection is called a catalogue, we will show that the term palette more closely resembles its uses. Our example palette is written in MPL, which enables palette composition and maintenance. It is based on a subset of patterns proposed by Adams and Dormans, to whom we refer for a high level discussion that omits programmed parameterized patterns as discussed here [AD12]. We introduce MM and MPL by example, briefly stating pattern intent. For conciseness we omit general explanations on motivation and applicability, focusing on how the patterns work. We do not claim these patterns are complete for explaining and authoring all interesting MM diagrams, and we expect that designers will have varying opinions on what makes a good palette. We now begin MeDeA’s automated analysis.



Intent: Activating converter (Acquire) costs (CostExp) resources from pool (Energy) as specified by resource connection (Cost) and yields (BenefitExp) resources in pool (Property) as specified by resource connection (Benefit).

Figure 5.4: Palette: an Acquisition pattern

Acquisition

Designers can apply the Acquisition pattern for offering players a way to acquire property by spending currency. The visual representation of the Acquisition pattern is shown in Figure 5.4. Visual MPL can be distinguished from MM by a rectangle with a crooked edge pattern label on the top left, its name appearing in the top center. Patterns consist of named elements called *participants*. Acquisition has five participants, three nodes and two edges. Each participant name represents the *role* the participant plays in its context. We explain them one by one.

- **Energy** is a node of type *pool*, which abstracts from an in-game entity modeled by an MM diagram and can contain resources. Pools appear as a circle that contains an optional amount of starting resources, and zero or more *categories* that specify design intent. In our example palette, the category symbol (\$) marks that a node abstracts from currency such as gold, crystals or lumber, and is intended for spending.
- **Property** is another pool. Its category symbol (+) marks that it abstracts from a diagram node whose resources players desire to have such as health (hp).
- **Cost** is a *resource connection*, an edge with a *flow rate expression* that specifies the amount of resources that can flow, in this case how much the acquisition costs.
- **Benefit** is another resource connection that specifies how much Property the acquisition yields.
- **Acquire** is a *converter*, appearing as a triangle pointing to the right with a vertical line through the middle, that converts one type of resource into another. Acquire is the only node that *acts* in this pattern by *pulling* resources along its *input* (Cost), and *pushing* resources along its *output* (Benefit). Using the game's user interface, players can activate nodes that have an *interactive activation modifier*, visually marked with a double line, but converters only work when all resources on its inputs are available.

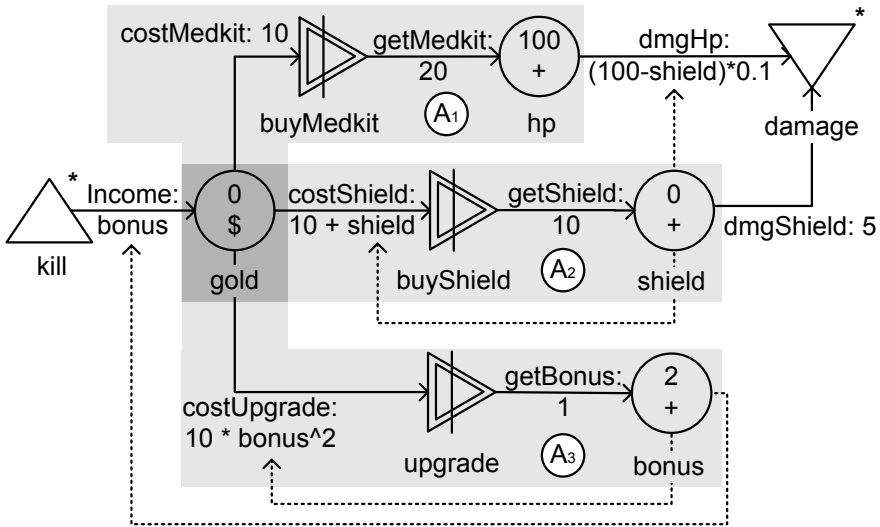


Figure 5.5: MM diagram showing three Acquisition cases

Table 5.2: Acquisition pattern cases in JJ

role	A ₁	A ₂	A ₃
Energy	gold	gold	gold
Acquire	buyMedkit	buyShield	upgrade
Property	hp	shield	bonus
Cost	costMedkit	costShield	costUpgrade
CostExp	10	10 + shield	10 * bonus ²
Benefit	getMedit	getShield	getBonus
BenefitExp	20	10	1

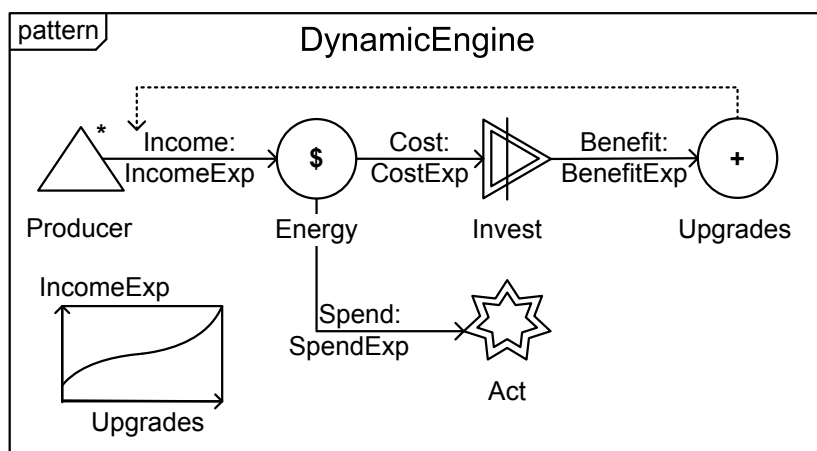
MeDeA analyzes our example diagram of Figure 5.3 against the Acquisition pattern, and recognizes three cases. Figure 5.5 shows pattern instances A₁, A₂ and A₃ (in light gray) that overlap in role Energy played by diagram pool gold (dark gray). Table 5.2 shows how roles are assigned over diagram elements and flow rates. MeDeA uses the pattern template to explain each pattern case to designers as shown in Table 5.3.

Dynamic Engine

Players require resources for activating one or more game-economic actions, in this case *gold*. The Dynamic Engine pattern, shown in Figure 5.6, introduces income and a

Table 5.3: MeDeA explains three Acquisition cases in JJ

case	explanation
A_1	Activating converter <i>buyMedkit</i> costs 10 resources from pool <i>gold</i> as specified by resource connection <i>costMedit</i> and yields 20 resources in pool <i>hp</i> as specified by resource connection <i>getMedkit</i> .
A_2	Activating converter <i>buyShield</i> costs $10+shield$ resources from pool <i>gold</i> as specified by resource connection <i>costShield</i> and yields 10 resources in <i>poolshield</i> as specified by resource connection <i>getShield</i> .
A_3	Activating converter <i>upgrade</i> costs $10*bonus^2$ resources from pool <i>gold</i> as specified by resource connection <i>costUpgrade</i> and yields 1 resources in pool <i>hp</i> as specified by resource connection <i>getBonus</i> .



Intent: Source (Producer) produces an adjustable flow (Income) of (IncomeExp) resources. Players can invest using converter (Invest) to improve the flow.

Apply when: Apply Dynamic Engine for introducing a trade-off between spending currency (Energy) on long-term investment (Invest) and short-term gains (Act).

Figure 5.6: Palette: a Dynamic Engine pattern

tradeoff between long-term investments and short-term gains [AD12]. We explain its participants.

- **Producer** is a *source*. A *source* node, appearing as a triangle pointing up, is the only element that can generate resources. A source can be thought of as a pool with an infinite amount of resources. It can push any amount of resources, and therefore provides the flow rates specified by its outputs to the

Table 5.4: Dynamic Engine pattern cases in JJ

role	D_1	D_2
Producer	kill	kill
Energy	gold	gold
Invest	upgrade	upgrade
Act	buyMedkit	buyShield
Upgrades	bonus	bonus
Income	income	income
IncomeExp	bonus	bonus
Cost	costUpgrade	costUpgrade
CostExp	$10 + \text{bonus}^2$	$10 + \text{bonus}^2$
Benefit	getBonus	getBonus
BenefitExp	1	1
Spend	costMedkit	costShield
SpendExp	10	10 + shield

respective targets. The automatic activation modifier (*) of *Producer* specifies it automatically provides *IncomeExp* resources via resource connection *Income*.

- **Energy** is a pool specifying currency (\$) gained.
- **Invest** is a converter converting Energy into Upgrades.
- **Act** is an *abstract node*, visually represented as a star, that represents any kind of node that players can activate for spending Energy. In this pattern it also represents an alternative user action to Invest.
- **Upgrades** is a pool whose resource amount influences *Income* positively and is marked as property (+).
- **Income** is a resource edge where its flow rate *IncomeExp* grows monotonically with Upgrades, as defined by the constraint on the left bottom of the diagram. This has to be the case for investment to be beneficial. A dashed edge signifying Upgrades is used in *Income* makes the feedback loop explicit.
- **Cost** is a resource edge specifying the cost of Invest from Energy.
- **Benefit** is resource edge specifying the benefit of Invest to Upgrades.
- **Spend** is a resource edge specifying the cost of Act.

MeDeA again analyzes our example diagram, now using the Dynamic Engine pattern and finds it twice as shown in Figure 5.7, and the pattern roles are distributed as shown in Table 5.4. The difference between instances D_1 and D_2 is *buyMedkit* is replaced by *buyShield*. MeDeA's explanation of pattern case D_1 is shown in Table 5.5. We omit its explanation of D_2 . We remark that in JJ *kill* happens automatically, but only when a player shoots an enemy, not every step.

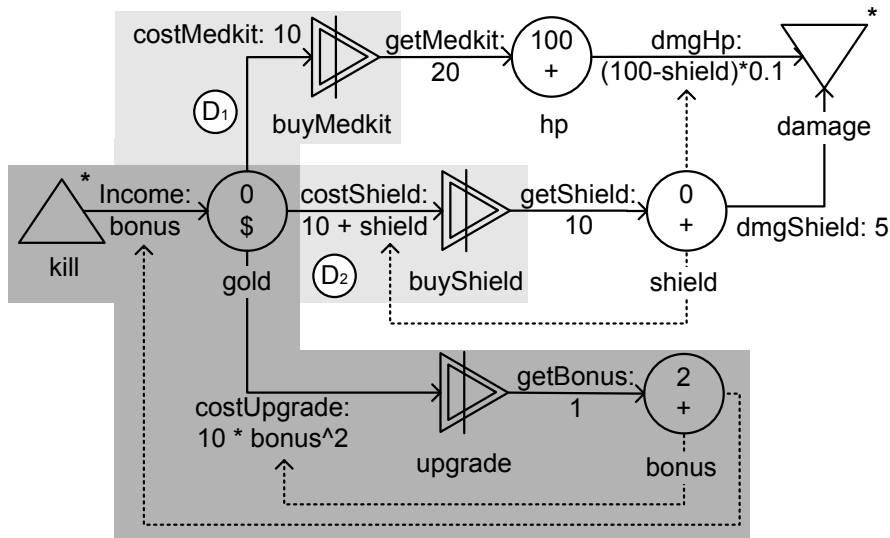


Figure 5.7: MM diagram showing two Dynamic Engine cases

Table 5.5: MeDeA explains a Dynamic Engine case in JJ

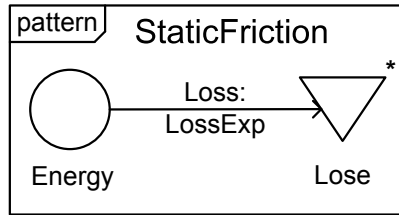
case	explanation
D_1	Source <i>kill</i> produces an adjustable flow <i>income</i> of <i>bonus</i> resources. Players can invest using converter <i>upgrade</i> to improve the flow. Apply Dynamic Engine for introducing a trade-off between spending currency <i>gold</i> on long-term investment <i>upgrade</i> and short-term gains <i>buyMedkit</i> .

Static Friction

So far our palette contains only patterns for gaining and converting resources. We need just one more pattern to explain all elements in the diagram. The Static Friction pattern is intended to counter positive effects a player tries to achieve, posing a challenge [AD12]. Its visual representation and pattern cases and MeDeA's explanations are shown in Figure 5.8, Table 5.6 and Table 5.7. This concludes our concise explanation of MeDeA's pattern-based analysis.

5.3.2 Mixed-Initiative Design Decision Making

MeDeA facilitates the process of understanding and making game design decisions as we have seen in the previous section. It supports a process that consists of a sequence



Intent: Drain (Lose) causes a loss by pulling flow rate (LossExp) via resource connection (Loss) from pool (Energy).

Figure 5.8: Palette: a Static Friction pattern

Table 5.6: Static Friction pattern cases in JJ

role	F_1	F_2
Energy	shield	hp
Loss	dmgShield	dmgHp
LossExp	5	$(100-\text{shield}) * 0.1$
Lose	damage	damage

Table 5.7: MeDeA explains Static Friction cases in JJ

case	explanation
F_1	Drain <i>damage</i> causes a loss by pulling flow rate $(100-\text{shield}) * 0.1$ via resource connection <i>dmgHp</i> from pool <i>hp</i> .
F_2	Drain <i>damage</i> causes a loss by pulling flow rate 5 via resource connection <i>dmgShield</i> from pool <i>shield</i> .

of simple steps as shown in Figure 5.9. Rectangles represent data structures and rounded rectangles represent processes either primarily controlled by the user (light gray) or MeDeA (dark gray). By default MeDeA recognizes a full pattern in a diagram, associating a diagram element to each role in the pattern. However, MeDeA can also recognize partial matches, where not all roles in the pattern are represented, yielding a possibly large set of *extension points* to the diagram with respect to the pattern. Each extension point is associated with a set of pattern elements that did not match we call the *extension*. Given a designer's decision to apply the pattern, we reason that each extension is a possible *design decision*. The problem is that exploring a large set of decisions alternatives is not feasible. Our solution is to step-by-step filter the design alternatives by letting the designer associate roles to diagram elements for cherry

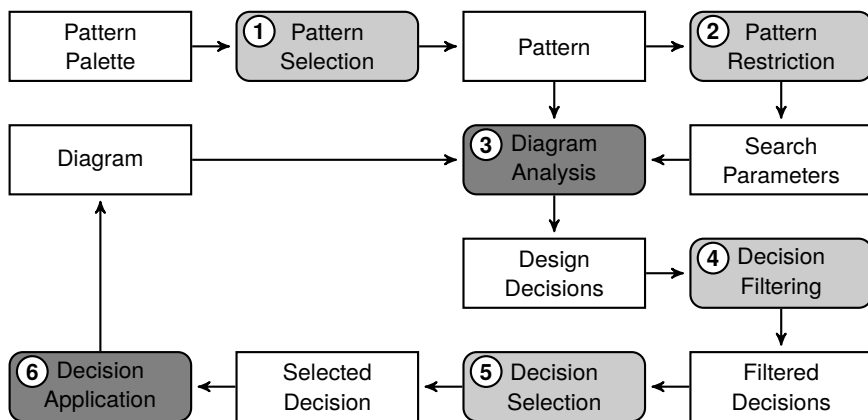


Figure 5.9: MeDeA iterative decision making process

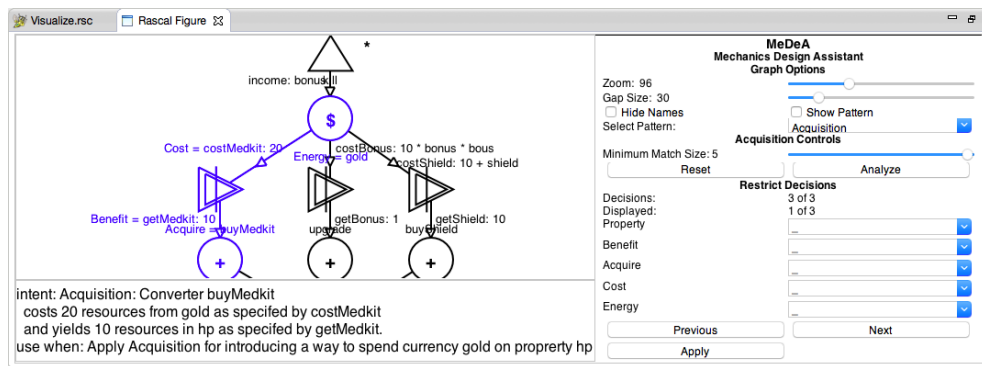


Figure 5.10: MeDeA: A Pattern-Based Game Mechanics Design Assistant

picking from a restricted set of alternatives. We explain the decision-making process and detail the steps designers take.

1. **Pattern Selection.** Select a pattern from the palette displayed by MeDeA for modifying a diagram.
2. **Pattern Restriction.** Choose the minimum amount of pattern elements (minimum match size) for which roles must be mapped to diagram elements, affecting the size of the extension point. Optionally, restrict the search for design decisions by step-by-step assigning roles to names of elements in the diagram. This yields role constraints that target the search.
3. **Diagram Analysis.** Initiate the analysis of the diagram against the pattern. MeDeA generates a potentially large set of design decisions, every way the

pattern applies to the diagram, given the minimum match size and predefined role constraints.

4. **Decision Filtering.** Filter the generated design decisions by further assigning roles to names of elements in the diagram. This yields additional role constraints that exclude generated decisions by filtering them out.
5. **Decision Selection.** Visually inspect the remaining design decisions, and select one that expresses design intent. MeDeA shows a visual rendering of diagram resulting from the design decision, providing colors for distinguishing between existing (black), found (blue), added/not found (green). For each visual representation MeDeA also shows the design intent specified by the pattern in which roles and amounts are replaced by their concrete names and values in the diagram.
6. **Decision Application.** Apply the selected design decision after providing MeDeA with names for all added elements (green) and flow amounts for all added edges. Additionally, found elements (blue) are optionally adjusted and replaced. MeDeA then replaces the current diagram by the newly created one. This concludes the iteration, continue at step 1.

5.3.3 Assisted Game Mechanics Authoring

We now demonstrate how MeDeA assists in authoring³ the mechanics of our running example shown in Figure 5.3 by following the decision process of Figure 5.9. We start with an empty diagram that we view as an empty canvas. First we select the Acquisition pattern from our palette. The pattern cannot match, because the canvas is empty, but we restrict the pattern to a minimum match size of zero, allowing us to add it. We provide MeDeA with the role names and edge flow rates specified by A_1 shown in Table 5.2 and apply the change. We again select Acquisition, restricting the pattern to a minimum match size of one, which yields seven decisions. We restrict the role *Energy* to *gold*, leaving four decisions. We select the decision where the other roles are added, providing values for them from A_2 as before and repeat these steps for A_3 . Our diagram now has the elements highlighted with gray in Figure 5.5.

Next we select the Dynamic Engine pattern from our palette and restrict the minimum match size to five. MeDeA offers twenty-seven design decisions. We associate the role *Upgrades* to *bonus*, *Energy* to *gold* and *Invest* to *upgrade*, leaving just three decisions. We pick one of two where *buyMedkit* or *buyShield* play role *Act* and apply it. The unassigned roles are *Producer* and *Income*, which we name *kill* and *income* respectively. We give *income* a flow rate of *bonus*, thereby satisfying the pattern constraint. Our diagram now contains the elements highlighted in gray in Figure 5.7.

Next we select the Static Friction pattern from our palette, restricting *Energy* to *hp* yielding one design decision. We apply it, providing vales for pattern roles *Loss*

³https://vrozen.github.io/fdg2015/MeDeA_authoring.mp4 (visited September 1st 2019)

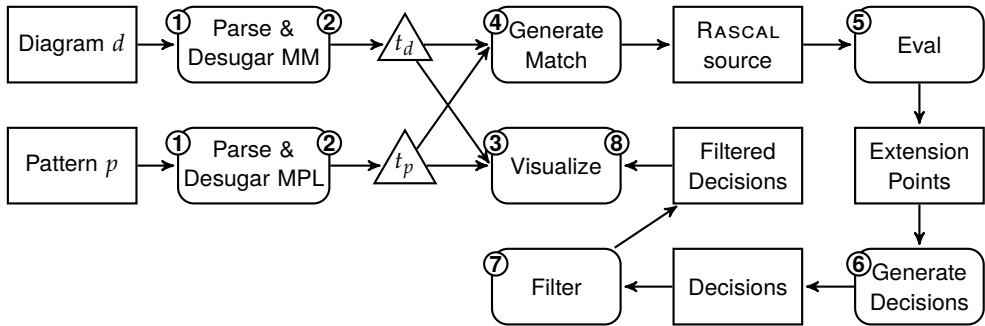


Figure 5.11: MeDeA: Model Transformation Steps

and *Lose* as specified by F_1 shown in Table 5.6. Finally, we select *Static Friction* one more time for also applying *Static Friction* to *shield* using the values specified by F_2 for pattern role *Loss*. With these simple steps our diagram is now complete.

5.3.4 Tool Architecture

MeDeA is implemented in RASCAL, a functional metaprogramming language and language workbench for source code analysis and manipulation [KvdSV09]. MeDeA’s implementation⁴ counts just 2.7K lines of code excluding comments and white-space. Each of MeDeA’s analysis and transformation steps is controlled from its UI shown in Figure 5.10, which is programmed using the RASCAL Figure library as interactive visualization offering designers UI elements for mixed-initiative decision making as explained in Section 5.3.2.

Figure 5.11 schematically shows the steps of how MeDeA processes diagrams and patterns. MM and MPL each have their own *grammar* for parsing (1) textual programs that are represented as visual diagrams and patterns in this paper. MM and MPL parse trees are *imploded* against an Algebraic Data Type (ADT) such that we get Abstract Syntax Trees (ASTs). These ASTs are *desugared* (2), a transformation in which syntactic constructs added for user convenience are transposed to a more fundamental ADT that we visualize (3). In our case the desugared ADT is the same for MM and MPL, which is necessary for our approach because we rely on a RASCAL feature called *set matching* for our results.

We generate RASCAL programs (4) in which parameterized ADTs for patterns are matched against the ADTs of diagrams. Some ADT fields are parameters and some are constants, depending on the pattern, the diagram and optional user-supplied constraints for limiting the search space. When we evaluate (5) these programs with

⁴<https://github.com/vrozen/MeDeA> (visited September 1st 2019)

RASCAL it uses *backtracking* to bind the parameters to constants in every possible way, yielding a possibly large set of extension points to the diagram with respect to the pattern. From these extension points we generate design decisions (6) which are visualized as partial diagrams without values for the added diagram elements. Filtering (7) happens over this set using constraints posed by assigning diagram element names to roles. We again use RASCAL's matching to filter out unwanted solutions in which the roles (the pattern parameters) are bound to different constants. When applying a decision the current diagram is replaced with a new one (8) in which added elements have their variables bound to user supplied constants such as element names and flow rate expressions.

5.4 DISCUSSION

MeDeA simplifies authoring, understanding and modifying a game's game-economic mechanics, and for its users is an improvement over editing textual MM programs. However, the tool is currently limited to text or pattern-based authoring because RASCAL does not yet support visual edits on figures. Moreover, MeDeA is intended for game designers that can work as *gameplay engineers*, because mechanics modeling is an inherently complex technical discipline. MeDeA performs exhaustive calculations, which means that larger diagrams matched against patterns with fewer user-defined constraints result in larger search spaces, more extension points and longer calculation times. However, diagrams are composed of modular constructs called components [KvR13] (abstracted away in this paper), which ensures that diagrams are relatively small. MeDeA supports a programmable extensible pattern palette for maintaining patterns, which mitigates the lack of MPL patterns mined from software.

5.5 CONCLUSION

We have presented a Mechanics Pattern Language (MPL) for programming patterns that capture a wide range of game-economic mechanics with shared structures and design intent, and a Mechanics Design Assistant (MeDeA) for analyzing, explaining and understanding existing mechanics that also supports exploring and applying decision alternatives for modifying mechanics. Its simple interface enables generating decisions from patterns, filtering and selecting using simple point-and-click controls, and all mechanics modifications result from applying generated design decisions. MeDeA is implemented in the RASCAL, a meta-programming language and language workbench. Of course, MeDeA is an academic prototype, and the case study on modifying the mechanics of Johnny Jetstream is a relatively small informal evaluation. Because the approach is general, embeddable, reusable and maintainable we believe it is a step towards industrially applicable game design tools. A more systematic evaluation is part of future work.

5.5.1 Future Work

Our pattern-based mixed-initiative model-driven game design approach can be extended by automating additional game disciplines and facets representing separated concerns.

- Our approach for generating alternative design decisions for modifying mechanics can also be used for fully automatic game design. It complements evolutionary approaches [TSo8], which can also provide an alternative for filtering, and can drive a game generator such as the Game-O-Matic [TBM⁺12], Ludi [BM10] or the Angelina system [CC11]. Because MM is a reusable embeddable formalism that expresses extensible game economies in general, it is less dependent on specific generators and constrained input. Recipes can consist of names of patterns, resources and actions, and designers can analyze, modify and fine-tune the generated results.
- Modifying games *live*—while they run—may help tackle adaptivity challenges [LB11]. Modeling player behaviors and player experience respectively enable automating mechanics testing and play testing.
- The gap between high-level game design patterns and programming game mechanics may be bridged by further automating pattern-based transformations. Case studies on games embedding MM can help evaluate how MPL patterns relate to game design patterns, and how predictive they are for emergence. MPL can support pattern mining, relating palettes to existing game design knowledge for forming genre-specific pattern palettes of game-economic game design lore.
- A major challenge remains engineering languages and tools for different experts participating in game development processes. We argue for software language engineering of game languages. This entails performing domain analyses and meta-programming game design tools, e.g., writers require view points on storylines whereas game designers need view points on game mechanics, gameplay, levels and missions. Each of these view points could exist as a separated concern, modified via *live* user interfaces designed to model, analyze and generate content for composing high quality games. Ultimately, games could be pieced together from sets of expressive, reusable, extensible, interoperable and embeddable formalisms.

Acknowledgements.

I thank Paul Klint for discussions and proof reading, and the anonymous reviewers for their insightful comments that helped restructure this paper.

Abstract

Live programming is a style of development characterized by incremental change and immediate feedback. Instead of long edit-compile cycles, developers modify a running program by changing its source code, receiving immediate feedback as it instantly adapts in response.

In this paper we propose an approach to bridge the gap between running programs and textual Domain-Specific Languages (DSLs). The first step of our approach consists of applying a novel model differencing algorithm, `TMDIFF`, to the textual DSL code. By leveraging ordinary text differencing and origin tracking, `TMDIFF` produces deltas defined in terms of the metamodel of a language.

In the second step of our approach the model deltas are applied at run time to update a running system, without having to restart it. Since the model deltas are derived from the static source code of the program, they are unaware of any run-time state maintained during model execution. We therefore propose a generic, dynamic patch architecture, `RMPATCH`, which can be customized to cater for domain-specific state migration. We illustrate `RMPATCH` in a case study of a live programming environment for a simple DSL implemented in `RASCAL` for simultaneously modeling and executing state machines.

6.1 INTRODUCTION

The “gulf of evaluation” represents the cognitive gap between an action performed by a user and the feedback provided to her about the effect of that action [LF95]. Live programming aims to bridge the gulf of evaluation by shortening the feedback loop between editing a program’s textual source code and observing its behavior. In a live programming environment the running program is updated instantly after

The content of this chapter was first published at the `ICMT 2015` conference, and later extended as a `SOSYM` journal publication. This chapter is based on the latter, which extends the original work with a generic run-time patch architecture (`RMPATCH`), as well as the live state machine case study. It was published as R. van Rozen and T. van der Storm. “Toward Live Domain-Specific Languages: From Text Differencing to Adapting Models at Run Time”. In: *Software & Systems Modeling* 18.1 (Feb. 2019). Special Section Paper on `STAF2015`. Received June 27th 2016. Revised May 26th 2017. Accepted June 20th 2017. First Online August 14th 2017, pp. 195–212. ISSN: 1619-1374. DOI: 10.1007/s10270-017-0608-7. For the evaluation of `TMDIFF` itself we refer to the original paper, which was published as R. van Rozen and T. van der Storm. “Origin Tracking + Text Differencing = Textual Model Differencing”. In: *Theory and Practice of Model Transformations – Proceedings of the 8th International Conference on Model Transformation, ICMT 2015, L’Aquila, Italy, July 20–21, 2015*. Ed. by D. Kolovos and M. Wimmer. Vol. 9152. LNCS. Springer, 2015, pp. 18–33. ISBN: 978-3-319-21155-8. DOI: 10.1007/978-3-319-21155-8_2.

every change to the code [Tan13]. As a result, developers immediately see the behavioral effects of their actions, and learn predicting how the program adapts to targeted improvements to the code. In this paper we are concerned with providing generic, reusable frameworks for developing “live DSLs”, languages whose users enjoy the immediate feedback of live execution. We consider such techniques to be first steps towards providing automated support for live languages in language workbenches [EVV⁺13].

In particular, we propose two reusable components, `TMDIFF` and `RMPATCH` to ease the development of textual live DSLs, based on a foundation of metamodeling and model interpretation. `TMDIFF` is used to obtain model-based deltas from textual source code of a DSL. These deltas are then applied at run time by `RMPATCH` to migrate the execution of the DSL program [vdSto13]. This enables the users of a DSL to modify the source and immediately see the effect.

The first component of our approach is the `TMDIFF` algorithm [vRvdS15]. `TMDIFF` employs textual differencing and origin tracking to derive model-based deltas from changes to textual source code. A textual difference is translated to a difference on the abstract syntax of the DSL, as specified by a metamodel. As a result, standard model differencing algorithms (e.g., [AP03]) can be applied in the context of textual languages.

The second component, `RMPATCH`, is used to dynamically adapt model execution to changes in the source code. This is achieved by “patching” the execution using the deltas produced by `TMDIFF`. We call differences applied to running programs executable deltas. To apply executable deltas we require that a language is implemented as a model interpreter [MFJ05]. In particular, we require that every class defined in a language’s metamodel has an implementation counterpart in some programming language (we use Java). The `RMPATCH` architecture supports applying an executable delta on the instances of those classes while the model is interpreted. To support run-time state, we allow the run-time classes to extend the classes of the metamodel with additional attributes and relations. Since the deltas produced by `TMDIFF` are unaware of those attributes and relations, the `RMPATCH` engine is designed to be open for extension to cater for migrating such domain-specific run-time state. `RMPATCH` has been applied in the development of a prototype live programming environment for a simple state machine DSL. A state machine definition can be changed while it is running, and the runtime execution will adapt instantly.

The key contribution of this paper is the combination of textual model differencing and run-time model patching for adapting models at run time with “live” textual DSLs, and to this end:

- We reiterate how textual differencing can be used to match model elements based on origin tracking information and provide a detailed description of `TMDIFF`, including a prototype implementation (Section 6.3).

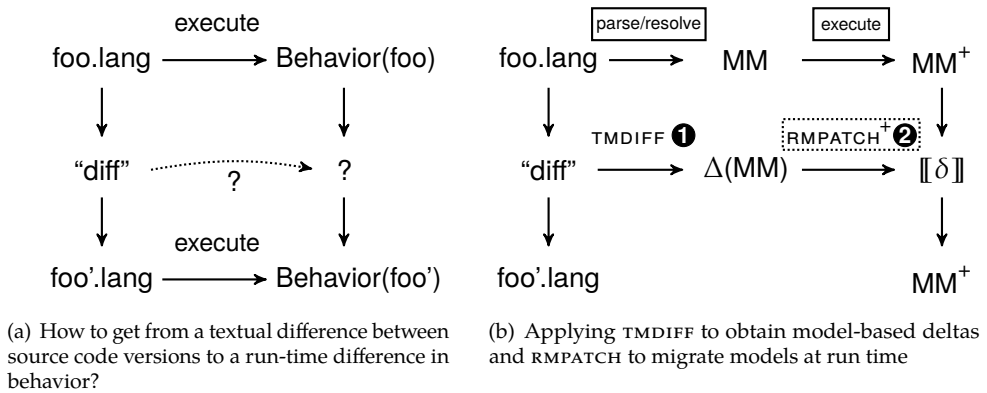


Figure 6.1: Bridging the gap between textual models and running programs

- We present a generic architecture for run-time patching of interpreted models (Section 6.4).
- We illustrate the framework using a live DSL environment for a simple state machine language (Section 6.5).

6.2 FROM TEXT DIFFERENCING TO LIVE MODELS AT RUN TIME

We motivate our work by taking the perspective of developers who use textual DSLs to iteratively modify and improve programs. Figure 6.1(a) gives an overview of the challenge of bridging the gap between a developer’s textual model edits and the associated program behavior that the developer needs to quickly observe, understand and improve.

A developer writes a program (`foo`) in some language (`lang`), which can be executed to obtain its behavior. The developer then evolves the program to a new version (`foo'`) by updating its source, yielding a textual difference. In a traditional setting, the effect of the change can only be observed by re-executing the program. However, this involves compiling and executing the program from scratch. This can be a time consuming distraction, losing all dynamic context observed while running `foo`. In particular, all run-time state accumulated during the execution of program version `foo` is lost when its next version `foo'` is executed (again). We aim to make this experience more fluid and live by obtaining a “run-time diff” from the textual “diff” between successive program versions (`foo` and `foo'`), and then migrating its execution (from `Behavior(foo)` to `Behavior(foo')`) at run time.

Figure 6.1(b) shows an overview of our solution to this problem. The `foo` program is mapped to an instance of a metamodel (`MM`), through parsing and name resolution.

Parsing constructs an initial containment hierarchy of the program in the form of an Abstract Syntax Tree (AST). Name resolution, on the other hand, creates cross references in the model based on the (domain-specific) referencing and scoping rules of the language, yielding an Abstract Syntax Graph (ASG). The model is then executed by an interpreter, which creates a run-time model corresponding to `foo`. This run-time model is an instance of an enhanced metamodel (MM^+), representing run-time state as additional attributes and relations. We require that MM^+ is an extension of MM .

Whenever the developer evolves the program's source, the textual difference between `foo` and `foo'` is now mapped to a model-based delta over the metamodel MM using `TMDIFF`. Such a delta consists of an edit script which changes the model of `foo` to a model representing `foo'`. That delta is then applied as an executable delta to the executing run-time model of `foo` by `RMPATCH`. Because the executing model has additional run-time state that could become invalid, `RMPATCH` needs to be augmented with language-specific migrations. The generic part of `RMPATCH` will only migrate the parts defined by MM ; the domain-specific customization defines what to do with the extensions defined by MM^+ . At specific points during execution, the interpreter will swap out the old version of the model, and start executing the new one, without having to restart, and without losing state.

Note that the parts in boxes are the components that are language-specific. This includes parsing and name resolution, which often need to be defined anyway, and a model-based interpreter. `TMDIFF` is completely language parametric, and thus can be reused for multiple live DSLs. `RMPATCH` is partially generic: it is generically defined for deltas produced by `TMDIFF`, but needs to be extended for dealing with the run-time state extensions defined by MM^+ .

The rest of the paper is structured as follows. Next in Section 6.3 we describe how `TMDIFF` works. In Section 6.4, we show how the deltas produced by `TMDIFF` are applied at run time using the generic patch architecture of `RMPATCH`. The customization of this architecture to support run-time state migration is described as part of our case study based on state machines in Section 6.5. We show how this enables a live programming environment for state machines using a prototype interpreter. We conclude the paper with a discussion of related work and an outline for further research.

6.3 TMDIFF: TEXTUAL MODEL DIFF

6.3.1 *Overview*

`TMDIFF` is a novel differencing algorithm that leverages ordinary text differencing and origin tracking to derive model-based deltas from textual source code. Traditional model differencing algorithms (e.g., [AP03]) determine which elements are added, removed or changed between revisions of a model. A crucial aspect of such algo-

rithms is that model elements need to be identified across versions. This allows the algorithm to determine which elements are still the same in both versions. In textual modeling [GBUo8], models are represented as textual source code, similar to DSLs and programming languages.

The actual model structure represented by an Abstract Syntax Graph (ASG) is not first-class, but is derived from the text by a text-to-model mapping, which, apart from parsing the text into an Abstract Syntax Tree (AST) specifying a containment hierarchy also provides for reference resolution. After every change to the text, the corresponding structure needs to be derived again. As a result, the identities assigned to the model elements during text-to-model mapping are not preserved across versions, and model differencing cannot be applied directly.

Existing approaches to textual model differencing are based on mapping textual syntax to a standard model representation (e.g., languages built with Xtext are mapped to EMF [EB10]) and then using standard model comparison tools (e.g., EMFCompare [BPo8; Ecl12]). As a result, model elements in both versions are matched using name-based identities stored in the model elements themselves. One approach is to interpret such names as globally unique identifiers: match model elements of the same class and identity, irrespective of their location in the containment hierarchy of the model. Other approaches are to match elements in collections at the same position in the containment hierarchy, to use similarity-based heuristics or to construct a purpose-built algorithm.

Unfortunately, each of these approaches has its limitations. In the case of global names, the language cannot have scoping rules: it is impossible to have different model elements of the same class with the same name. On the other hand, matching names relative to the containment hierarchy entails that scoping rules must obey the containment hierarchy, which limits flexibility in terms of scoping. While similarity-based matching techniques can deal with scopes, these may also require fine-tuning the heuristic to obtain more accurate results for specific languages and uses.

TMDIFF is a language-parametric technique for model differencing of textual languages with complex scoping rules, but at the same time is agnostic of the model containment hierarchy. As a result, different elements with the same name, but in different scopes can still be identified. TMDIFF is based on two key techniques:

- **Origin tracking.** In order to map model element identities back to the source, we assume that the text-to-model mapping applies origin tracking [IvdSE14; vDKT93]. Origin tracking induces an *origin relation* which relates source locations of definitions to (opaque) model identities. Each semantic model element can be traced back to its defining name in the textual source, and each defining name can be traced forward to its corresponding model element.
- **Text Differencing.** TMDIFF identifies model elements by textually aligning definition names between two versions of a model using traditional text differencing

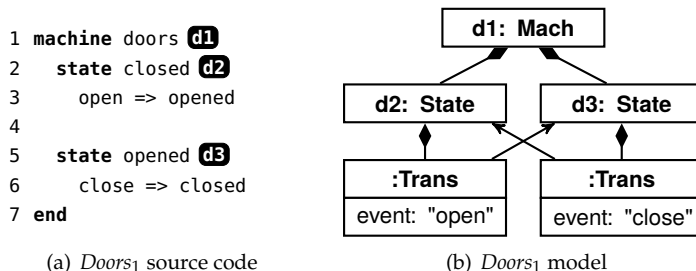


Figure 6.2: *Doors₁*: a simple textual representation of a state machine and its model

techniques (e.g., [MM85]). When two names in the textual representations of two models are aligned, they are assumed to represent the same model element in both models. In combination with the origin relation this allows TMDIFF to identify the corresponding model elements as well.

The resulting identification of model elements can be passed to standard model differencing algorithms, such as the one by Alanen and Porres [AP03].

TMDIFF enjoys the important benefit that it is fully language parametric. TMDIFF works irrespective of the specific binding semantics and scoping rules of a textual modeling language. In other words, how the textual representation is mapped to model structure is irrelevant. The only requirement is that semantic model elements are introduced using symbolic names, and that the text-to-model mapping performs origin tracking.

Here we introduce textual model differencing using a simple motivating example that is used as a running example throughout the paper. Figure 6.2 shows a state machine model for controlling doors. It is both represented as text (left) and as object diagram (right). A state machine has a name and contains a number of state declarations. Each state declaration contains zero or more transitions. A transition fires on an event, and then transfers control to a new state.

The symbolic names that *define* entities are annotated with unique labels d_n . These labels capture *source locations* of names. That is, a name occurrence is identified with its line and column number and/or character offset¹. Since identifiers can never overlap, labels are guaranteed to be unique, and the actual name corresponding to label can be easily retrieved from the source text itself. For instance, the machine itself is labeled d_1 , and both states `closed` and `opened` are labeled d_2 and d_3 respectively.

The labels are typically the result of *name analysis* (or reference resolution), which distinguishes definition occurrences of names from use occurrences of names according to the specific scoping rules of the language. For the purpose of this paper

¹For the sake of presentation, we use the abstract labels d_i for the rest of the paper, but keep in mind that they represent source locations

```

1 machine doors d4
2   state closed d5
3   open => opened
4   lock => locked
5
6   state opened d6
7   close => closed
8
9   state locked d7
10  unlock => closed
11
12 end

1 machine doors d8
2   state closed d9
3   open => opened
4   lock => locking.locked
5
6   state opened d10
7   close => closed
8
9   locking d11 {
10    state locked d12
11    unlock => closed
12  }
13 end

```

(a) *Doors₂* (b) *Doors₃*

Figure 6.3: Two new versions of the simple state machine model *Doors₁*

it is immaterial how this name analysis is implemented, or what kind of scoping rules are applied. The important aspect is to know which name occurrences represent definitions of elements in the model.

By propagating the source locations (d_i) to the fully resolved model, symbolic names can be linked to model elements and vice versa. On the right of Figure 6.2, we have used the labels themselves as object identities in the object model. Note that the anonymous Transition objects lack such labels. In this case, the objects do not have an identity, and the difference algorithm will perform structural differencing (e.g., [Yan91]), instead of semantic, model-based differencing [AP03].

Figure 6.3 shows two additional versions of the state machine of Figure 6.2. First the machine is extended with a locked state in *Doors₂* (Fig. 6.3a). Second, *Doors₃* (Fig. 6.3b), shows a grouping feature of the language: the locked state is part of the locking group. The grouping construct acts as a scope: it allows different states with the same name to coexist in the same state machine model.

Looking at the labels in Figure 6.2 and 6.3, however, one may observe that the labels used in each version are disjoint. For instance, even though the defining name occurrences of the machine *doors* and state *closed* occur at the exact same location in *Doors₂* and *Doors₃*, this is an accidental result of how the source code is formatted. Case in point is the name *locked*, which now has moved down because of the addition of the group construct.

The source locations, therefore, cannot be used as (stable) identities during model differencing. The approach taken by TMDIFF involves determining added and removed definitions by aligning the textual occurrences of defining names (i.e. labels d_i). Based

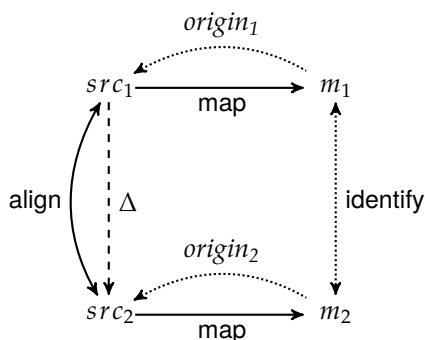


Figure 6.4: Identifying model elements in m_1 and m_2 through origin tracking and alignment of textual names.

```

--- a/doors1.sl          --- a/doors2.sl
+++ b/doors2.sl          +++ b/doors3.sl
@@ -3,0 +4              @@ -4 +4
+ lock => locked        - lock => locked
@@ -6,0 +8,3           + lock => locking.locked
+                       @@ -8,0 +9
+ state locked          + locking {
+ unlock => closed      @@ -10,0 +12
+                       + }

```

(a) Textual diff between $Doors_1$ and $Doors_2$ (b) Textual diff between $Doors_2$ and $Doors_3$ ²

Figure 6.5: Textual diff between $Doors_1$ and $Doors_2$, and $Doors_2$ and $Doors_3$ ²

on the origin tracking between the textual source and the actual model we identify which model elements have persisted after changing the source text.

This high-level approach is visualized in Fig. 6.4. src_1 and src_2 represent the source code of two revisions of a model. Each of these textual representations is mapped to a proper model, m_1 and m_2 respectively. Mapping text to a model induces origin relations, $origin_1$ and $origin_2$, mapping model elements back to the source locations of their defining names in src_1 and src_2 respectively. By then aligning these names between src_1 and src_2 , the elements themselves can be identified via the respective origin relations.

²The diffs are computed by the diff tool included with the git version control system. We used the following invocation: `git diff --no-index --patience --ignore-space-change --ignore-blank-lines --ignore-space-at-eol -U0 <old> <new>`.

<pre> create State d7 d7 = State("locked",[Trans("unlock", d2)]) d2.out[1] = Trans("lock", d7) d1.states[2] = d7 </pre>	<pre> create Group d11 d11 = Group("locking",[d7]) remove d4.states[2] d4.states[2] = d11 </pre>
(a) <code>tmdiff Doors₁ Doors₂</code>	(b) <code>tmdiff Doors₂ Doors₃</code>

Figure 6.6: TMDIFF differences between $Doors_i$ and $Doors_{i+1}$ ($i \in \{1, 2\}$)

TMDIFF aligns textual names by interpreting the output of a textual diff algorithm on the model source code. The diffs between $Doors_1$ and $Doors_2$, and $Doors_2$ and $Doors_3$ are shown in Fig. 6.5. As we can see, the diffs show for each line whether it was added (“+”) or removed (“-”). By looking at the line number of the definition labels d_i it becomes possible to determine whether the associated model element was added or removed.

For instance, the new `locked` state was introduced in $Doors_2$. This can be observed from the fact that the diff on the left of Fig. 6.5 shows that the name “locked” is on a line marked as added. Since the names `doors`, `closed` and `opened` occur on unchanged lines, TMDIFF will identify the corresponding model elements (the machine, and the 2 states) in $Doors_1$ and $Doors_2$. Similarly, the diff between $Doors_2$ and $Doors_3$ shows that only the group `locking` was introduced. All other entities have remained the same, even the `locked` state, which has moved into the group `locking`.

With the identification of model elements in place, TMDIFF applies a variant of the standard model differencing introduced in [AP03]. Hence, TMDIFF deltas are imperative edit scripts that consist of edit operations on the model. Edit operations include creating and removing of nodes, assigning values to fields, and inserting or removing elements from collection-valued properties. Figure 6.6 shows the TMDIFF edit scripts computed between $Doors_1$ and $Doors_2$ (a), and $Doors_2$ and $Doors_3$ (b). The edit scripts use the definition labels d_n as node identities.

The edit script shown in Fig. 6.6(a) captures the difference between source version $Doors_1$ and target version $Doors_2$. It begins with the creation of a new state d_7 . On the following line d_7 is initialized with its name (`locked`) and a fresh collection of transitions. The transitions are *contained* by the state, so they are created anonymously (without identity). Note that the created transition contains a (cross-)reference to state d_2 . The next step is to add a new transition to the `out` field of state d_2 (which is preserved from $Doors_1$). The target state of this transition is the new state d_7 . Finally, state d_7 is inserted at index 2 of the collection of states of the machine d_1 in $Doors_1$.

The edit script introducing the grouping construct `locking` between $Doors_2$ and $Doors_3$ is shown in Fig. 6.6(b). The first step is the creation of a new group d_{11} . It is initialized with the name “`locking`”. The set of nested states is initialized to contain


```

1 list[Operation] tmDiff(str src1, str src2, obj m1, obj m2) {
2   <A, D, M> = match(src1, src2, m1, m2)
3   Δ = [ new Create(da, da.class) | da ← A ]
4   M' = M + { <da, da> | da ← A }
5   Δ += [ new SetTree(da, build(da, M')) | da ← A ]
6   for (<d1, d2> ← M)
7     Δ += diffNodes(d1, d1, d2, [], M')
8   Δ += [ new Delete(dd) | dd ← D ]
9   return Δ
10 }

```

Figure 6.7: TMDIFF

state d_7 which already existed in $Doors_2$. Finally, the state with index 2 is removed from the machine d_4 in $Doors_3$, and then replaced by the new group d_{11} .

In this section we have introduced the basic approach of TMDIFF using the state machine example. The next section presents TMDIFF in more detail.

6.3.2 TMDiff in More Detail

Top-level Algorithm

Figure 6.7 shows the TMDIFF algorithm in high-level pseudo code. Input to the algorithm are the source texts of the models (src_1, src_2), and the models themselves (m_1, m_2). The first step is to determine corresponding elements in m_1 and m_2 using the matching technique introduced above. We further describe the match function later in this section.

Based on the matching returned by `match` (line 2), TMDIFF first generates global `Create` operations for nodes that are in the A set (line 3). After these operations are created, the matching M is “completed” into M' , by mapping every added object to itself (line 4). This ensures that reverse lookups in M' for elements in m_2 will always be defined. Each entity just created is initialized by generating `SetTree` operations which reconstruct the containment hierarchy for each element d_a using the `build` function (line 5). The function `diffNodes` then computes the difference between each pair of nodes originally identified in M (lines 6–7). The edit operations will be anchored at object d_1 (first argument). As a result, `diffNodes` produces edits on “old” entities, if possible. Finally, the nodes that have been deleted from m_1 result in global `Delete` actions (line 8).

```

1 Matching match(str src1, str src2, obj m1, obj m2) {
2   P1 = project(m1)
3   P2 = project(m2)
4   <Ladd, Ldel> = split(diff(src1, src2))
5
6   i = 0, j = 0; A = {}, D = {}; I = {}
7   while (i < |P1| ∨ j < |P2|) {
8     if (i < |P1| ∧ P1[i].line ∈ Ldel)
9       D += {P1[i].object}; i += 1; continue
10    if (j < |P2| ∧ P2[j].line ∈ Ladd)
11      A += {P2[j].object}; j += 1; continue
12    if (P1[i].object.class = P2[j].object.class)
13      I += {<P1[i].object, P2[j].object>}
14    else
15      D += {P1[i].object}; A += {P2[j].object}
16    i += 1; j += 1
17  }
18  return <A, D, I>;
19 }

```

Figure 6.8: Matching model elements based on source text diffs.

Matching

The match function uses the output computed by standard `diff` tools. In particular, we employ a `diff` variant called *Patience Diff*³ which is known to often provide better results than the standard, LCS-based algorithm [Mye86].

The matching algorithm is shown in Fig. 6.8. The function `match` takes the textual source of both models (src_1 , src_2) and the actual models as input (m_1 , m_2). It first projects out the origin and class information for each model (lines 1–2). The resulting projections P_1 and P_2 are sequences of tuples $\langle x, c, l, d \rangle$, where x is the symbolic name of the entity, c its class (e.g. State, Machine, etc.), l the textual line it occurs on and d the object itself.

As an example, the projections for $Doors_1$ and $Doors_2$ are as follows:

³See: <http://bramcohen.livejournal.com/73318.html>

$$\begin{aligned}
P_1 &= \begin{bmatrix} \langle \text{doors}, & \text{Machine}, & 1, & d_1 \rangle, \\ \langle \text{closed}, & \text{State}, & 2, & d_2 \rangle, \\ \langle \text{opened}, & \text{State}, & 5, & d_3 \rangle \end{bmatrix} \\
P_2 &= \begin{bmatrix} \langle \text{doors}, & \text{Machine}, & 1, & d_4 \rangle, \\ \langle \text{closed}, & \text{State}, & 2, & d_5 \rangle, \\ \langle \text{opened}, & \text{State}, & 6, & d_6 \rangle, \\ \langle \text{locked}, & \text{State}, & 9, & d_7 \rangle \end{bmatrix}
\end{aligned}$$

The algorithm then partitions the textual diff in two sets L_{add} and L_{del} of added lines (relative to src_2) and deleted lines (relative to src_1) (line 4). The main **while**-loop then iterates over the projections P_1 and P_2 in parallel, distributing definition labels over the A , D and I sets that will make up the matching (lines 6–17). If a name occurs unchanged in both src_1 and src_2 , an additional type check prevents that entities in different categories are matched (lines 12–15).

The result of matching is a triple $M = \langle A, D, I \rangle$, where $A \subseteq L_{m_2}$ contains new elements in m_2 , $D \subseteq L_{m_1}$ contains elements removed from m_1 , and $I \subseteq L_{m_1} \times L_{m_2}$ represents identified entities (line 18), where L_{m_1} and L_{m_2} are labels of elements in m_1 and m_2 respectively. For instance the matchings between $Doors_1$, $Doors_2$, and between $Doors_2$ and $Doors_3$ are:

$$\begin{aligned}
M_{1,2} &= \langle \{d_7\}, \{\}, \{\langle d_1, d_4 \rangle, \langle d_2, d_5 \rangle, \langle d_3, d_6 \rangle\} \rangle \\
M_{2,3} &= \langle \{d_{11}\}, \{\}, \{\langle d_4, d_8 \rangle, \langle d_5, d_9 \rangle, \langle d_6, d_{10} \rangle, \langle d_7, d_{12} \rangle\} \rangle
\end{aligned}$$

Next we explain how the matching result is used for differencing nodes.

Differencing

The heavy lifting of TMDIFF is realized by the `diffNodes` function. It is shown in Fig. 6.9. It receives an existing entity as the current context (ctx), the two elements to be compared (m_1 and m_2), a Path p which is a list recursively built up out of names and indexes and the matching relation to provide reference equality between elements in m_1 and m_2 . `diffNodes` assumes that both m_1 and m_2 are of the same class (line 3). The algorithm then loops over all fields that need to be differenced (lines 4–16). Fields can be of four kinds: primitive (lines 5–6), containment (lines 7–11), reference (lines 12–13) or list (lines 14–15). For each case the appropriate edit operations are generated, and in most cases the semantics is straightforward and standard. For instance, if the field is list-valued, we delegate differencing to an auxiliary function `diffLists` (not shown) which performs Longest Common Subsequence (LCS) differencing using reference

```

1 list[Operation] diffNodes(obj ctx, obj m1, obj m2, Path p, Matching M) {
2   assert m1.class = m2.class;
3   Δ = []
4   for (f ← m1.class.fields) {
5     if (f.isPrimitive && m1[f] ≠ m2[f])
6       Δ += [new SetPrim(ctx, p + [f], m2[f])]
7     else if (f.isContainment)
8       if (m1[f].class = m2[f].class)
9         Δ += diffNodes(ctx, m1[f], m2[f], p + [f], M)
10      else
11        Δ += [new SetTree(ctx, p + [f], build(m2[f], M))]
12      else if (f.isReference &&  $M^{-1}[m2[f]] \neq m1[f]$ )
13        Δ += [new SetRef(ctx, p + [f],  $M^{-1}[m2[f]]$ )]
14      else if (f.isList)
15        Δ += diffLists(ctx, m1[f], m2[f], p + [f], M)
16    }
17  return Δ
18 }

```

Figure 6.9: Differencing nodes

equality. The interesting bit happens when differencing reference fields. References are compared via the matching M , highlighted in Figure 6.9.

In order to know whether two references are “equal”, `diffNodes` performs a reverse lookup in M on the reference in m_2 (line 12). If the result of that lookup is different from the reference in m_1 the field needs to be updated. Recall that M was augmented to M' (cf. Fig. 6.7) to contain entries for all newly created model elements. As a result, the reverse lookup (line 13) is always well-defined. Either we find an already existing element of m_1 , or we find a element created as part of m_2 , highlighted in Fig. 6.9.

6.3.3 Implementation in RASCAL

We have implemented `TMDIFF` in `RASCAL`, a functional programming language for metaprogramming and language workbench for developing textual DSLs [KvdSV09]. The code for the algorithm, the application to the example state machine language, and the case study can be found on GitHub⁴.

Since `RASCAL` is a textual language workbench [EVV⁺13] all models are represented as text, and then parsed into an abstract syntax tree (AST). Except for primitive values

⁴<https://github.com/cwi-swat/textual-model-diff>

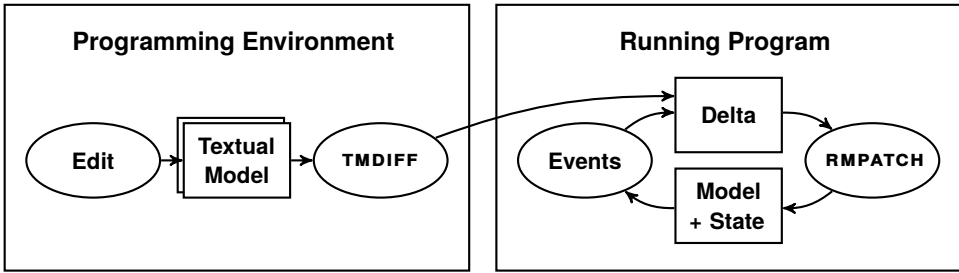


Figure 6.10: Approach: using TMDIFF and RMPATCH for live programming

(string, boolean, integer etc.), all nodes in the AST are automatically annotated with source locations to provide basic origin tracking.

Source locations are a built-in data type in RASCAL (**loc**), and are used to relate sub-trees of a parse tree or AST back to their corresponding textual source fragment. A source location consists of a resource URI, an offset, a length, and begin/end and line/column information. For instance, the name of the closed state in Fig. 6.3 is labeled:

```
|project://textual-model-diff/input/doors1.sl|(22,6,<2,8>,<2,14>)
```

Because RASCAL is a functional programming language, all data is immutable and first-class references to objects are unavailable. Therefore, we represent the containment hierarchy of a model as an AST, and represent cross-references by explicit relations **rel**[**loc** from, **loc** to], once again using source locations to represent object identities.

In prior work [vRvdS15], we have evaluated TMDIFF on the version history of *file format specifications* written in Derric, a real-life DSL that is used in digital forensics analysis [vdBvdS11]. We found that TMDIFF reliably computes small deltas between consecutive versions of the Derric specifications of JPEG, GIF, and PNG.

6.4 RMPATCH: GENERIC RUN-TIME MODEL PATCHING

6.4.1 Overview

The previous section described the TMDIFF algorithm to obtain model-based deltas from textual source files. Here we introduce RMPATCH, a generic architecture to apply these deltas to run-time models that drive the execution of the models of a language. During interpretation of such a model, users edit the textual model using a live programming environment that embeds TMDIFF for generating deltas for successive model versions, as shown in Figure 6.10 on the left. These edit scripts are

applied by `RMPATCH` to migrate the model as part of the running program to reflect the new version of the source code, as shown in Figure 6.10 on the right. Together `TMDIFF` and `RMPATCH` provide a foundation for the design and implementation of live programming environments, where textual models can be edited while they are executing.

In order to provide a unified approach for recording and replaying model differences, we record a run-time history of events such as user interactions and changes to the source code as edit operations on the run-time model. This history can be used for implementing “undo”, persisting application state (cf. event sourcing), and back-in-time debugging. When the developer edits a textual model and saves a modified version, the programming environment applies `TMDIFF` to the current and the previous version of the textual model. It then passes the resulting delta to `RMPATCH`, which pauses the interpreter, applies the delta to the run-time model, possibly migrating run-time state, and continues the interpreter. Similarly, we also represent the effects of other events as deltas, e.g., resulting from a user pressing a button or a sensor firing. In Figure 6.10 the oval “events” represents these cases.

6.4.2 *Models at Run Time*

Live programming environments enable adapting models at run time as text. Specifically, a model is an instance of a static metamodel of a language represented by an ASG, which is obtained from text through parsing and name resolution. `RMPATCH` requires that a model interpreter is implemented in an object-oriented language, like Java. In particular, it requires reflection for interpreting executable deltas that create objects and assign values to fields. The interpreter executes a model as a *run-time model*, an instance of a *run-time metamodel*, which extends the static metamodel of the language by adding additional attributes and relations to model run-time state, and methods that implement behavior.

For instance, a state machine can be executed by interpreting incoming events and updating a current state attribute. In between such transitions, the run-time model may need to be migrated however, because, in a live programming environment, the source code of the state machine may have changed in the meantime. At dedicated points in the execution, the interpreter must check for pending deltas (as produced by `TMDIFF`), and if there are any, apply them to the run-time model, before continuing execution.

6.4.3 *Applying Deltas at Run Time*

The deltas produced by `TMDIFF` are converted to run-time edit operations that can be evaluated against an instance of the run-time metamodel. Every change computed by `TMDIFF` can be mapped to a change at run time, because the model of the source is

subsumed by the run-time model. Applying a run-time delta contributes a sequence of atomic edits to the run-time history of the running program. The edit operations produced by TMDIFF, however, are unaware of any additional state maintained in the run-time models. For avoiding information loss and invalid run-time states, RMPATCH can be extended with custom state migrations. Migration effects are represented as model edits too, making them part of the run-time history.

Recall that TMDIFF produces edit scripts as shown in Figure 6.6:

```

create State d7                                // create
d7 = State("locked",[Trans("unlock", d2)]) // setTree
d2.out[1] = Trans("lock", d7)                 // insertTree
d1.states[2] = d7                             // insertRef

```

Such a script is represented as a list of edits, such as `create`, `setTree`, `insertTree` and `insertRef`. In addition to these four, TMDIFF generates `delete`, `setPrim`, `remove`, `insertRef` and `setRef` operations. `create` and `delete` are global operations, creating or deleting objects from the model, respectively. The other, relative operations traverse a path through the features of their *owner object*, the object operated on, (e.g., d_7 , d_2 , or d_1), and modify the traversed field accordingly. For instance, the last operation in the edit script above, inserts state d_7 in the machine's (d_1) list of states at index 2.

The edit operations `setTree` and `insertTree` take trees as arguments. Java makes no distinction between a tree argument's containment references and cross references, and encodes both as object references. We therefore flatten tree operations to a sequence of `create`, `setPrim`, `setRef` and `insertRef` operations. As a result RMPATCH only implements these operations, and `delete` and `remove`.

Owner objects are represented using opaque identities used internally by TMDIFF. RMPATCH maintains an `objectSpace` table that maps these identities to Java objects. The `create` and `delete` operations respectively add and remove objects in this table. Since the identities are not stable across versions of a model, RMPATCH uses the TMDIFF matching (see Section 6.3.2) information to adjust the object space to reflect the situation after the edit operations have been applied.

Applying the edit operations to the run-time model is implemented using the Visitor pattern [GHJ⁺94]. A base visitor defines `visit` methods for each type of edit operation, and modifies the current model according to the semantics of the operation. When an edit has been applied, it is added to the global history object to support undo and replay.

The application of edit operations to a run-time model is unaware of invariants concerning the run-time state extensions of that model. Naively applying a TMDIFF delta to the run-time model of a DSL program, might bring its execution in an inconsistent state. For instance, in the case of state machines, what happens if the current state is removed? What happens if the last remaining state is removed? These questions cannot be answered in a generic, language independent way. We therefore

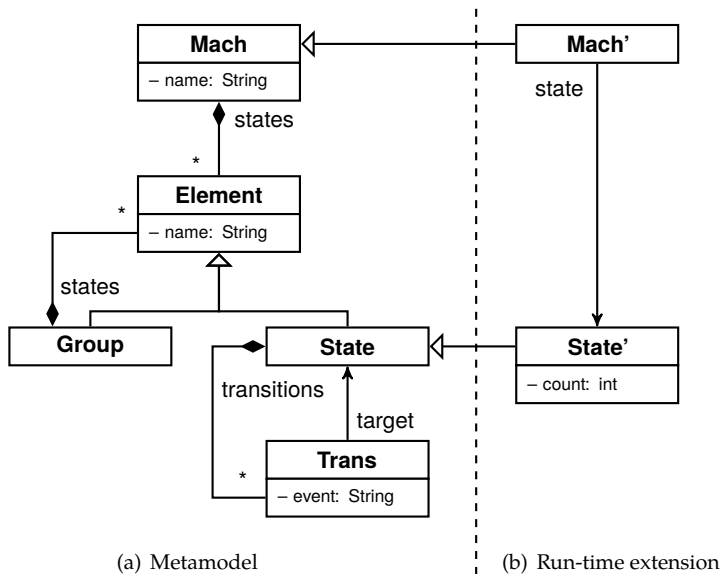


Figure 6.11: Static and run-time metamodel of SML

allow the base visitor to be extended with custom state migration logic to address such questions. If such additional migration steps are realized as edit operations as well, they can also be added to the global application history, to ensure that undo and replay maintain consistency.

The next section describes how these technique have been applied in the development of a live programming environment for the state machine language of Section 6.3.

6.5 CASE STUDY: LIVE STATE MACHINE LANGUAGE

6.5.1 Overview

Here we present a case study based on the simple State Machine Language (SML) used as the running example in Section 6.3. We have used both `TMDIFF` and `RMPATCH` to obtain a live programming environment for SML, called LiveSML. The static and run-time metamodels of SML are shown in Figure 6.11.

The run-time model (Figure 6.11(b)) can be seen as an extension of the static metamodel (Figure 6.11(a)); it includes all the attributes and relations of the static model. However, to represent run-time state, there are additional attributes and relations that do not exist in the static metamodel. For instance, run-time machines

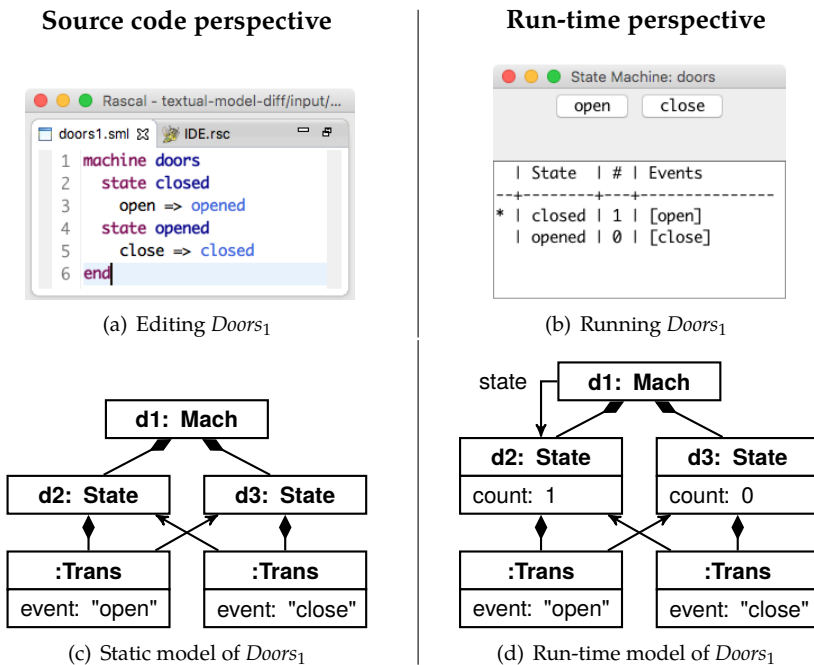


Figure 6.12: LiveSML: the left shows the source code perspective with the IDE at the top and the static model at the bottom. The right shows the run-time perspective with the state machine GUI at the top, and the (extended) run-time model at the bottom.

(Mach objects) have a state field, representing the current state. Furthermore, the State objects are extended with a count field, indicating how many times this state has been visited.

LiveSML consists of two application components, shown in the top row of Figure 6.12. On the left, Figure 6.12(a) shows the programming environment of LiveSML, which consists of an Eclipse-based IDE for editing state machines, implemented in RASCAL. The editor shows the *Doors*₁ state machine.

On the right, Figure 6.12(b) shows the execution of *Doors*₁ as an interactive GUI. The user can click buttons corresponding to events defined in the state machine. The main window shows a textual rendering of the state state machine in tabular form. An asterisk indicates which state is the current one, and the column marked with the pound symbol indicates how many times a state has been visited. The bottom row shows the actual *Doors*₁ state machine models. Figure 6.12(c) shows the static state machine model that represents the textual source code of *Doors*₁ shown in the editor. Figure 6.12(d) shows the same state machine, represented as a dynamic model that is executing at run time, which is shown in the GUI.

```

1  class MigrateSML extends ApplyDelta {
2    private Mach machine; //run-time model to migrate
3
4    @Override
5    public void visit(Create create) {
6      super.visit(create);
7      Object x = create.getCreated(this);
8      if (x instanceof Mach) { //new machine
9        this.machine = (Mach) x;
10     }
11     else if (x instanceof State) { //new state
12       Edit e = new SetPrim(reverseLookup(x), new Path(new Field("count"), 0);
13       e.accept(this);
14     }
15   }
16
17   @Override
18   public void visit(Insert insert) {
19     super.visit(insert);
20     Object owner = insert.getOwner(this);
21     if (machine != null && machine.state == null && owner == machine) {
22       // Added a group or state to a machine without a current state.
23       goToInitialState();
24     }
25   }
26
27   @Override
28   public void visit(Delete delete) {
29     super.visit(delete);
30     Object x = delete.getDeleted(this);
31     if (machine != null && x == machine.state) { // Deleted the current state.
32       goToInitialState();
33     }
34   }
35
36   private void goToInitialState(){
37     State s = machine.findInitial();
38     Edit e1 = new Set(reverseLookup(machine),
39     new Path(new Field("state"), s);
40     e1.accept(this); //Set the current state.
41     if (s != null){
42       Edit e2 = new Set(reverseLookup(s), new Path(new Field("count"), s.count+1);
43       e2.accept(this); //Increment current state count.
44     }
45   }
46 }

```

Figure 6.13: MigrateSML extends ApplyDelta for SML state migration

When a developer edits a textual model and saves a modified version, the programming environment applies `TMDIFF` to the current and the previous version of the textual model. It then passes the resulting delta to the executing program that embeds `RMPATCH`. Similarly, when the user triggers an event, the program calculates its own delta for updating its model elements. As a result, run-time model transformations result either from textual model edits or user-level application events.

6.5.2 *Migrating Domain-Specific Run-time State*

Since the deltas produced by `TMDIFF` only take the static metamodel of the source into account, the generic `RMPATCH` system needs to be extended to support dealing with the *state* and *count* attributes. Note that in most cases, `RMPATCH` will simply leave these attributes intact, but in special cases, the outcome would lead to an inconsistent state of the execution.

We define domain-specific state migration logic by extending the `ApplyDelta` visitor provided by `RMPATCH`, as shown in Figure 6.13. The class `ApplyDelta` defines a visit method for each kind of edit supported by `RMPATCH`. For `LiveSML`, we address the following cases:

- **Creation of a new machine.** Initially there is no machine because we start with an empty object space. We store a reference to the machine when it is first created (lines 9 and 10).
- **Creation of a new state.** The *count* attribute is initialized to 0 (lines 12–15).
- **Insertion of an element in an uninitialized machine.** When a state or group is inserted into a machine that has no current state (lines 24–29), it is initialized to the *initial state* (lines 43–54). The initial state is the first state in the textual model.
- **Deletion of the current state.** When a machine’s current state is deleted (lines 36–37), it is reinitialized to the initial state (lines 43–54).

Each domain-specific migration is represented using edit operations. For each required side effect, new edit objects are created. For instance, initializing the *count* field of a new state to 0, is enacted by a `SetPrim` edit, anchored at the new state, with a path to field “count”. Applying these operations through the extended visitor (`MigrateSML`) adds them to the application history of `LiveSML`.

6.5.3 *Evolving and Using State Machines with LiveSML*

The key point of `LiveSML` is that state machines can be edited and used at the same time. In a sense, the source and run-time models coevolve in lockstep: changes to the code are interleaved with user events, – both transform the run-time model using deltas. To illustrate this coevolution, we present a prototype live editing scenario with `LiveSML`.

Table 6.1: Interleaved coevolution of models $Doors_n$ and run-time states s_n over time

Model	State	Event	Edit operation	Origin
\emptyset	s_0	Save $Doors_1$	δ_1 create lang.sml.runtime.State d2 δ_2 d2.count = 0 δ_3 create lang.sml.runtime.State d3 δ_4 d3.count = 0 δ_5 create lang.sml.runtime.Mach d1 δ_6 d2 = State(name("closed"),[Trans("open",d3)]) δ_7 d3 = State(name("opened"),[Trans("close",d2)]) δ_8 d1 = Mach(name("doors"),[d2,d3]) δ_9 d1.state = d2 δ_{10} d2.count = 1	TMDIFF \emptyset $Doors_1$ side effect side effect side effect side effect
$Doors_1$	s_1	Click <i>open</i>	δ_{11} d1.state = d3 δ_{12} d3.count = 1	user action
$Doors_1$	s_2	Click <i>close</i>	δ_{13} d1.state = d2 δ_{14} d2.count = 2	user action
$Doors_1$	s_3	Save $Doors_2$	δ_{15} create lang.sml.runtime.State d7 δ_{16} d7.count = 0 δ_{17} d7 = State(name("locked"),[Trans("unlock",d2)]) δ_{18} insert d2.transitions[1] = Trans("lock",d7) δ_{19} insert d1.states[2] = d7 δ_{20} rekey d1 \rightarrow d4 δ_{21} rekey d2 \rightarrow d5 δ_{22} rekey d3 \rightarrow d6	TMDIFF $Doors_1$ $Doors_2$ side effect
$Doors_2$	s_4	Click <i>lock</i>	δ_{23} d4.state = d7 δ_{24} d7.count = 1	user action
$Doors_2$	s_5	Save $Doors_3$	δ_{25} create lang.sml.runtime.Group d11 δ_{26} d11 = Group("locking",[d6]) δ_{27} remove d4.states[2] δ_{28} insert d4.states[2] = d0 δ_{29} rekey d4 \rightarrow d8 δ_{30} rekey d5 \rightarrow d9 δ_{31} rekey d6 \rightarrow d10 δ_{32} rekey d7 \rightarrow d12	TMDIFF $Doors_2$ $Doors_3$
$Doors_3$	s_6	Save $Doors_1$	δ_{33} remove d8.states[2] δ_{34} remove d9.transitions[1] δ_{35} delete d11 δ_{36} delete d12 δ_{37} d13.state = d9 δ_{38} d9.count = 3 δ_{39} rekey d8 \rightarrow d13 δ_{40} rekey d9 \rightarrow d14 δ_{41} rekey d10 \rightarrow d15	TMDIFF $Doors_3$ $Doors_1$ side effect side effect

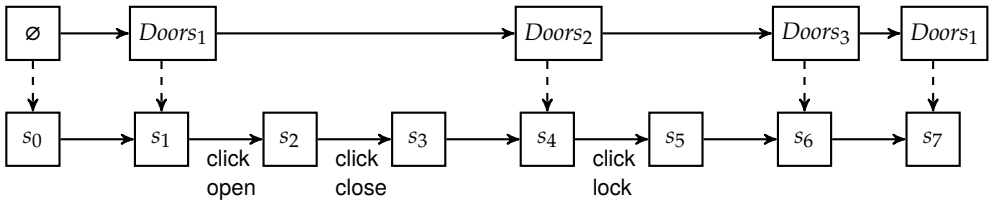


Figure 6.14: Interleaved coevolution of models $Doors_n$ and application run-time states s_n over time

Figure 6.14 shows its general time line. The top row shows five successive versions of the state machine definition, starting in the version where there is no state machine at all (\emptyset). The bottom row shows successive states of the executing state machine. Some state changes are triggered by source changes (e.g., from s_0 to s_1), while others result from user interactions (e.g., s_2 to s_3).

The details of the application state transitions are listed in Table 6.1. The first two columns indicate the start source model and run-time model state. The third column (“Event”) captures what happened (“saving” or “clicking an event button”). Each event causes a sequence of edits δ_i to be applied to the run-time model. Edits

Table 6.2: Sequence of screen shots of LiveSML’s programming environment (top) and running application (bottom) while in application state s_i ($i \in 0, \dots, 7$) of the interactive session with LiveSML.

s_0	s_1	s_2																		
	<table border="1"> <thead> <tr> <th>State</th> <th>#</th> <th>Events</th> </tr> </thead> <tbody> <tr> <td>* closed</td> <td> 1</td> <td> [open]</td> </tr> <tr> <td> opened</td> <td> 0</td> <td> [close]</td> </tr> </tbody> </table>	State	#	Events	* closed	1	[open]	opened	0	[close]	<table border="1"> <thead> <tr> <th>State</th> <th>#</th> <th>Events</th> </tr> </thead> <tbody> <tr> <td>* opened</td> <td> 1</td> <td> [close]</td> </tr> <tr> <td> closed</td> <td> 1</td> <td> [open]</td> </tr> </tbody> </table>	State	#	Events	* opened	1	[close]	closed	1	[open]
State	#	Events																		
* closed	1	[open]																		
opened	0	[close]																		
State	#	Events																		
* opened	1	[close]																		
closed	1	[open]																		

S₃

S₄

S₅

The screenshot shows the Rascal IDE with a state machine code editor and three GUI windows. The code editor shows the following code:

```

1 machine doors
2   state closed
3   open => opened
4   lock => locked
5   state opened
6   close => closed
7   state locked
8   unlock => closed
9 end

```

The three GUI windows are:

- S₃:** Shows buttons for 'open' and 'close'. The state table below is:

State	#	Events
* closed	2	[open]
opened	1	[close]
- S₄:** Shows buttons for 'open', 'lock', 'close', and 'unlock'. The state table below is:

State	#	Events
* closed	2	[open, lock]
opened	1	[close]
locked	0	[unlock]
- S₅:** Shows buttons for 'open', 'lock', 'close', and 'unlock'. The state table below is:

State	#	Events
closed	2	[open, lock]
opened	1	[close]
* locked	1	[unlock]

S₆

S₇

The screenshot shows the Rascal IDE with a state machine code editor and a GUI window. The code editor shows the following code:

```

1 machine doors
2   state closed
3   open => opened
4   lock => locking, locked
5   state opened
6   close => closed
7   locking
8   {
9     state locked
10    unlock => closed
11  }
12 end

```

The GUI window for S₆ and S₇ shows buttons for 'open' and 'close'. The state table below is:

State	#	Events
* closed	3	[open]
opened	1	[close]

correspond directly to the operations generated by TMDIFF. One additional operation (rekey) is used to realign the internal object identities of the run-time model with the opaque identities used by TMDIFF; this operation is needed because the TMDIFF identities are not stable across revisions. The last column shows the origin of the edit operations: an edit can originate from a TMDIFF delta, a migration side-effect (as described in Section 6.5.2), or a user action. The sequence of δ_i ($i \in 1..41$) represents the full history of run-time model transformations.

Finally, Table 6.2 shows, yet again, the sequence of source models and program states of the LiveSML session, – this time showing both the editor and the runtime GUI. From left to right, the upper row shows states s_0 to s_2 , the middle s_3 to s_5 and the bottom row s_6 and s_7 . An empty cell indicates that nothing has changed in the editor with respect to the previous state.

We now briefly describe how each run-time model state s_n in the sequence results from textual model edits and user actions.

- s_0 . The application starts and the initial model is \emptyset . Both the editor and GUI are empty.
- s_1 . *Doors*₁ is entered into the editor, and saved. In response, the environment computes the difference TMDIFF \emptyset *Doors*₁. As a result, the GUI shows the execution of *Doors*₁. Both state count attributes are initialized to zero (δ_2 and δ_4). The machine's initial state is *closed* (marked by *) and its count is set to one (δ_9 and δ_{10}).
- s_2 . The user clicks button *open*, which triggers the transition and produces δ_{11} and δ_{12} .
- s_3 . The user clicks button *close*, which triggers the transition and produces δ_{13} and δ_{14} .
- s_4 . The model is modified such that it becomes *Doors*₂. In response, the environment computes the difference between *Doors*₁ and *Doors*₂. The *count* attribute of the *locked* state is initialized to zero (δ_{16}). The UI now also displays buttons for the *lock* and *unlock* events.
- s_5 . The user clicks button *lock*, which triggers the transition and produces operations δ_{23} and δ_{24} .
- s_6 . The model is modified such that it becomes *Doors*₃. In response, the environment computes the difference between *Doors*₂ and *Doors*₃. This time, there are no migration side effects because the change has no semantic effect: grouping is just a scoping mechanism.
- s_7 . Finally, the model is modified such that it becomes *Doors*₁ again. As a result of applying the differences, the current state *locked* is removed and therefore the current state is reinitialized to the first state *closed* (δ_{37}). Accordingly, its *count* is set to three (δ_{38}). Note that the buttons *lock* and *unlock* have been removed from the UI since no such events exist anymore.

The sequence of states of this LiveSML session shows the fine-grained interleaving of edit operations originating from different sources. The execution of the state machine adapts to both user events and changes in the source code. As such, LiveSML provides a very fluid developer experience. Long edit-compile cycles are completely eliminated.

6.6 DISCUSSION AND RELATED WORK

This paper presents an approach for live programming environments for textual DSLs that builds on two reusable components: `TMDIFF` and `RMPATCH`. We reflect on limitations, challenges and future work, and discuss related work.

6.6.1 *Towards Live Domain-Specific Languages*

Live DSLs aim for a low representation gap between domain, notation and run time. Users can adapt run-time models directly from the textual source. We assume that the run-time metamodel extends the static language metamodel, such as is the case in LiveSML. This design choice facilitates applying changes of the source code to the running program. The assumption does not hold in general, however. For instance imperative languages have more complex mappings between code and execution. Such languages therefore offer less direct affordances over a program's execution, breaking the continuous link between the mental model of the programmer, the code and the running program.

Edit scripts are commonly used to encode model differences between versions of models representing the abstract syntax of a language. Edit scripts precisely encode *what* changed and in which order, but not *why* these effects happen. Typically, language semantics refers to a formal definition that does include the precise causal relationships from which these run-time changes result, which also enables formal proofs. In our approach the behavioral evolution of executing models is influenced by the way model differences are computed. When entities are not detected as “the same” between versions the corresponding run-time objects will be removed or added, even if this was not the behavior intended by the user of the modeling language. This problem is not unique to our application of `TMDIFF`, since any differencing algorithm will have to use heuristics to match model elements. We hypothesize, however, that in the context of live programming where immediacy of feedback is paramount, changes tend to be small and local, reducing the risk of unintuitive matchings.

One question is whether replacing `TMDIFF` by an alternative algorithm would provide a better programmer experience. For instance, `SiDiff` [KKP⁺12; TBW⁺07], `DSMDiff` [LGJ07] or `EMFCCompare` [Ecl12] may result in a more accurate matchings for specific circumstances. `SiDiff` in particular would be a candidate since it is independent from any kind of scoping rules used to create references between model

elements. SiDiff can be configured to make the algorithm perform better based on certain language features. Unfortunately, adjusting the weights used in comparing language features, often requires substantial empirical testing [KDP⁺09].

The question is if similarity-based heuristics would offer more predictable differences, and as a result more predictable run-time adaptation. Our hypothesis is that TMDIFF has the benefit that its mechanism for identifying model elements stays close to the textual source representation of a model, which is precisely the material the modeler is manipulating. Comparing alternative differencing approaches in terms of predictability and run time performance is part of future work.

Our experience in using TMDIFF and RMPATCH shows that migrating run-time state is complex. Even for a relatively simple language like LiveSML, the extensions of RMPATCH to migrate state must account for many possible transformation scenarios. Since edit operations are applied in sequence, one must make careful assumptions about the existence or absence of objects and references. The key question is then if the correct interleaving of migration edits with the original edits produced by TMDIFF could be automatically derived. In future work we plan to address this challenge by separately modeling and maintaining migration scenarios that abstract from underlying edits, and use dependency analysis to derived possible orderings of run-time model modifications.

Assessing if RMPATCH scales to larger systems requires additional case studies on real-world live DSLs, in particular those whose source and run-time metamodels differ more substantially than in the case of LiveSML. To investigate this question further, we plan to apply RMPATCH to Micro-Machinations, a visual language and execution engine that enables game designers to adapt a game’s mechanics while it is running [vRD14]. Its live programming environment is called Mechanics Design Assistant (MeDeA) [vRoz15a].

The run-time metamodel of Micro-Machinations adds a new level of dynamic instantiation: at run time there are “instance” level models which are not directly represented by textual source code, but which depend on source-defined entity definitions. Such languages require a pipeline of coupled transformations between source and runtime. The question is how modification effects propagate in a well-defined way. This problem is not unlike migrating objects after a change in class (e.g., in Smalltalk), or database migration upon schema change. In fact, these kinds of migrations are instances of the general class of *coupled transformations* [Lämo4] where a transformation of one model induces a “coupled” transformation on another (possibly over a different metamodel). Further research is needed to formalize run-time patching presented here using this framework. This could help to precisely delineate the scope and limitations of RMPATCH-like run-time adaptation.

Reversible transformations support features for programming environments such as undoing edits, rollback, restoring system states, replaying and debugging. RMPATCH

operations can be augmented with extra information to make every edit operation – and thus complete edit scripts – reversible. The question is to what extent such features can be supported by generic, reusable components. Although it is clear how to “unapply” edit operations on the run-time model, performing this same operation on the textual source code requires more advanced machinery, such as origin tracking, source code formatting and reversing source-to-source transformations.

At this time, `TMDIFF` and `RMPATCH` offer no special support for model merging, which, for instance, would be interesting for hypothetical exploration of dynamic what-if scenarios. Further research is needed to investigate how different deltas produced by `TMDIFF` can be combined for this purpose and how to resolve merge conflicts at run time.

6.6.2 *Limitations of TMDiff*

Unlike `RMPATCH`, the `TMDIFF` algorithm can be used independently. In this section we identify a number of limitations of `TMDIFF` as a separate component and discuss directions for further research.

The matching of entities uses textual deltas computed by `diff` as a guiding heuristic. In rare cases this affects the quality of the matching. For instance, `diff` works at the granularity of a line of code. As a result, any change on a line defining a semantic entity will incur the entity to be marked as added. The addition of a single comment may trigger this incorrect behavior. Furthermore, if a single line of code defined multiple entities, a single addition or removal will trigger the addition of all other entities. Nevertheless, we expect entities to be defined on a single line most of the time.

If not, the matching process can be made immune to such issues by first pretty-printing a textual model (without comments) before performing the textual comparison. The pretty-printer can then ensure that every definition is on its own line. Note, that simply projecting out all definition names and performing longest common subsequence (LCS) on the result sequences abstracts from a lot of textual context that is typically used by `diff`-like tools. In fact, this was our first approach to matching. The resulting matchings, however, contained significantly more false positives.

Another factor influencing the precision of the matchings is the dependence on the textual order of occurrence of names. As a result, when entities are moved without any further change, `TMDIFF` will not detect it as such. We have experimented with a simple move detection algorithm to mitigate this problem, however, this turned out to be too computationally expensive. Fortunately, edit distance problems with moves are well-researched, see, e.g., [Tic84]. A related problem is that `TMDIFF` will always see renames as an addition and removal of an entity. In general, edit scripts consisting of long sequences of atomic operations are hard to understand. However, user-level composite operations such as renaming and more complex refactorings

can be detected in existing sequences of atomic operations, e.g., using the approach proposed by Langer et al. [LWB⁺13], or the rule-based semantic lifting approach proposed by Kehrer et al. [KKT11].

6.6.3 Related Work

The key contribution of this paper intersects two areas of related work: model differencing and dynamic adaptation of models at run time. Below we discuss important related work in both these areas.

Model Differencing

Much work has been done in the research area of model comparison that relates to TMDIFF. We refer to a survey of model comparison approaches and applications by Stephan and Cordy for an overview [SC13]. In the area of model comparison, *calculation* refers to identifying similarities and differences between models, *representation* refers to the encoding form of the similarities and differences, and *visualization* refers to presenting changes to the user [KDP⁺09; SC13]. Here we focus on the calculation aspect.

Calculation involves matching entities between model versions. Strategies for matching model elements include matching by 1) *static identity*, relying on persistent global unique entity identifiers; 2) *structural similarity*, comparing entity features; 3) *signature*, using user defined comparison functions; 4) *language specific algorithms* that use domain specific knowledge [SC13]. With respect to this list, our approach represents a new point in the design space: matching by textual alignment of names.

The differencing algorithm underlying TMDIFF is directly based on Alanen and Porres' seminal work [AP03]. The identification map between model elements is explicitly mentioned, but the main algorithm assumes that model element identities are stable. Additionally, TMDIFF supports elements without identity. In that case, TMDIFF performs a structural diff on the containment hierarchy (see, e.g., [Yan91]).

TMDIFF's differencing strategy resembles the model merging technique used Ensō [vdSCL14]. The Ensō "merge" operator also traverses a spanning tree of two models in parallel and matches up object with the same identity. In that case, however, the objects are identified using primary keys, relative to a container (e.g., a set or list). This means that matching only happens between model elements at the same syntactic level of the spanning tree of an Ensō model. As a result, it cannot deal with "scope travel" as in Fig. 6.3c, where the locked state moved from the global state to the locking scope. On the other hand, the matching is more precise, since it is not dependent on the heuristics of textual alignment.

Epsilon is a family of languages and tools for model transformation, model migration, refactoring and comparison [KPP08]. It integrates HUTN [RPK⁺08], the

OMG’s Human Usable Text Notation, to serialize models as text. As result, which elements define semantic identities is known for each textual serialization. In other words, unlike in our setting, HUTN provides a fixed concrete syntax with fixed scoping rules. TMDIFF allows languages to have custom syntax, and custom binding semantics.

Lin et al. describe DSMDiff, a signature-based differencing approach which is intended specifically for Domain-Specific Modeling Languages [LGJ07]. DSMDiff uses a signature-based matching over node and edge model elements, augmented by structural matching when the signature-based matching produces multiple matching candidates.

Maoz et al. propose *semantic differencing*, an approach that defines *diff operators* for comparing two models where the resulting differences are presented as a set of semantic *diff witnesses*, instances of the first model that are not instances of the second [MRR11]. These instances are concrete examples explaining how the models differ. Maoz and Ringert relate syntactic changes to semantic witnesses by defining necessary and sufficient sets of change operations [MR15].

Langer et al. present a general approach for semantic differencing that can be customized for specific modeling languages. This approach is based on the behavioral semantics of a modeling language [LMK14]. Two versions of a model are executed to capture execution traces that represent its semantic interpretation. Comparing these traces then provide a “semantic” interpretation of the difference between the two versions. In contrast, our approach starts at the opposite end: instead of using execution traces to explain syntactic differences, we use syntactic differences to drive the execution in the first place.

Cicchetti et al. propose a representation of model differences which is model-based, transformative, compositional and metamodel independent [CDP10]. Differences are represented as models that can be applied as patches to arbitrary models. Although no special extension points are offered for supporting run-time state migrations, the model-based differences themselves could be used to represent them.

Dynamic Adaptation

“Models at run time” is a well-researched topic, as, for instance, witnessed by the long running workshop on Models@run.time [GBF15]. Executable modeling can be considered a subdomain of models at run time, where a software system’s execution is defined by a model interpreter. Executable modeling was pioneered in the context of the Kermeta system [CCP12; MFJ05]. Kermeta is also the basis for recent work on omniscient debugging features for xDSMLs [BCC⁺15]. Omniscient debuggers allow the execution of a program or model to be reversed and replayed. This work can be positioned on an orthogonal axis of “liveness”, where the focus is on providing better feedback through time travel. We consider our delta-based approach to be a

fruitful ground for further exploration of such features. In the LiveSML case study we already have implemented a reversible history of application state. However, a particular challenge will be to apply reversed edits back to the source code of a DSL program.

Models at run time in general are often motivated from the angle of dynamic adaptation. For instance, Morin et al. [MBJ⁺09] describe an architecture to support adaptation at run time through aspect weaving. However, this work focuses on adapting behavior and dynamically selecting alternative variants of behavior, rather than changing the run-time models themselves.

The specific requirements for run-time metamodeling are explored by Lehmann et al. [LBT⁺11]. The authors present a process to identify the core run-time concepts occurring in run-time models. In particular, they propose to identify possible model adaptations at run time, to explicitly address potential run-time consistency issues. In our case we allow any kind of modification, but leave the door open to implement arbitrary run-time state migration policies.

RMPATCH requires the run-time metamodel to be an “extension” of the static metamodel. This relation is similar to the concept of “subsumption” in description logics [MB95]. Although we have not yet explored this link in more detail, it would allow formal checking of whether a run-time metamodel is suitable for live patching. Another assumption underlying RMPATCH is that it should be possible to pause the model interpreter at a stable point in the execution in order to apply the runtime modifications. This is related to the concept of quiescence explored in the area of dynamic software updating [VEB⁺07].

6.7 CONCLUSION

Live programming promises to improve developer experience through immediate and continuous feedback. These benefits have not yet been explored from the perspective of executable domain-specific modeling languages. In this paper we have described a framework for developing “live textual languages”, based on a metamodeling foundation. Our framework consists of two components.

First, we presented TMDIFF, a novel model differencing algorithm, based on textual differencing and origin tracking. Origin tracking traces the identity of an element back to the symbolic name that defines it in the textual source of a model. Using textual differencing these names can be aligned between versions of a model. Combining the origin relation and the alignment of names is sufficient to identify the model elements themselves. It then becomes possible to apply standard model differencing algorithms. TMDIFF is a fully language parametric approach to textual model differencing. A prototype of TMDIFF has been implemented in the RASCAL metaprogramming language [KvdSV09].

The second component, *RMPATCH*, represents an architecture for dynamically adapting run-time models which encode the execution of the model. *RMPATCH* receives model deltas from *TMDIFF*, and evolves the execution accordingly. To avoid information loss and invalid run-time states, *RMPATCH* can be extended to define custom, language-specific migration policies. *RMPATCH* is used in the development of a live state machine DSL, which allows simultaneous editing *and* using of state machine definitions.

To the best of our knowledge, this paper is the first work connecting the worlds of model differencing and dynamic adaptation of models at run time. Nevertheless, some important directions for further research remain. The most important directions are formalizing the relation between static metamodel and (extended) run-time metamodel of a DSL, investigating how dependencies between edit operations can be inferred and used to (re)order their application, and determining how to separately model and maintain run-time state migration scenarios at a higher level of abstraction. Ultimately, we expect that delta-based run-time adaptation provides a fertile foundation for developing live programming support for executable DSLs.

Acknowledgements

We thank the reviewers for their insightful comments that helped improve this paper.

MEASURING QUALITY OF GRAMMARS FOR PROCEDURAL LEVEL GENERATION

Abstract

Grammar-based procedural level generation raises the productivity of level designers for games such as dungeon crawl and platform games. However, the improved productivity comes at the cost of level quality assurance. Authoring, improving and maintaining grammars is difficult because it is hard to predict how each grammar rule impacts the overall level quality, and tool support is lacking. We propose a novel metric called Metric of Added Detail (MAD) that indicates if a rule adds or removes detail with respect to its phase in the transformation pipeline, and Specification Analysis Reporting (SAnR) for expressing level properties and analyzing how qualities evolve in level generation histories. We demonstrate MAD and SAnR using a prototype of a level generator called Ludoscope Lite. Our preliminary results show that problematic rules tend to break SAnR properties and that MAD intuitively raises flags. MAD and SAnR augment existing approaches, and can ultimately help designers make better levels and level generators.

7.1 INTRODUCTION

Grammar-based level generation is a form of Procedural Content Generation (PCG) that raises the productivity of game level designers. Instead of hand-crafting levels, designers create a level transformation pipeline that generates levels for them by authoring modules, grammars and rewrite rules. The grammar rules work on data structures such as strings, tile maps and graphs, which can be used for generating names, level layouts and missions. These artifacts are step-by-step transformed and combined until a final detailed and fully populated level is generated, with missions, power-ups, challenges, enemies, hidden treasures, secret pathways, encounters, etc. Ideally, each generated level has the intended qualities. Unfortunately, improving the productivity of level designers comes at the cost of quality assurance.

In practice, many small problems arise, such as levers in walls, blocked pathways, missing encounters and lava adjacent to water. A lack of direct manipulation compromises the ability of designers to isolate and improve level qualities, e.g., when

This chapter was previously published as R. van Rozen and Q. Heijn. "Measuring Quality of Grammars for Procedural Level Generation". In: *Proceedings of the 13th International Conference on Foundations of Digital Games, FDG 2018, as part of the 9th Workshop on Procedural Content Generation, PCG 2018, Malmö, Sweden, August 7–10, 2018*. Ed. by S. Dahlskog, S. Deterding, J. Font, M. Khandaker, C. M. Olsson, S. Risi, and C. Salge. ACM, 2018, pp. 1–8. ISBN: 978-1-4503-6571-0. DOI: 10.1145/3235765.3235821

authoring bridges, forests or paths. As a result, some generated levels may lack intended goals, challenges and missions.

The qualities of generated levels depend on the composition of grammar rules and how they are combined in sequence. Therefore, potential bugs often remain unknown until they are observed during playtesting. Additionally, the combinatorial explosion resulting from recursive rule expansions complicates forming mental models required for reasoning about intended qualities, and how they are represented in the grammar or intermediate data. Moreover, it is hard to predict how individual rules affect the overall level quality.

Grammars are brittle, i.e. code that is liable to break easily. Designers require special measures to ensure that qualities once introduced, remain intact, preventing successive rewrites from breaking levels. Fixing one level with a rule that prevents an occurrence may introduce new problems in others. In general, there is a lack of tools and techniques for authoring, debugging, testing and improving rules that introduce and preserve design intent. As a result, the full potential of these techniques has not yet been realized.

We aim to improve the quality of grammar-based procedural level generation in general, and focus on grammars that work on tile maps in particular. We motivate our research by studying and improving Ludoscope, a state-of-the-art development environment for generating very diverse game levels. Since its inception, Ludoscope was developed by Ludomotion for indie game development, and successfully applied to a rogue-like dungeon crawler called Unexplored. We address the need of developers for better tools. This paper proposes and contributes two enabling techniques:

1. Metric of Added Detail (MAD), a novel metric that indicates if a grammar rule adds or removes detail to a tile map. We hypothesize that grammars gradually add detail. MAD leverages a detail hierarchy, a binary relation on alphabet symbols indicating which symbol is more detailed, which can easily be derived from transformation pipelines.
2. Specification Analysis Reporting (SAnR), a technique that offers a level property language for expressing level qualities. SAnR analyzes and reports how these properties evolve over time in level generation histories.

We demonstrate the feasibility of MAD and SAnR by implementing Ludoscope Lite (LL), a light-weight version of Ludoscope intended to study level quality. LL is implemented using RASCAL, a meta-programming language and language workbench. Our preliminary evaluation shows that SAnR can express and analyze simple level properties, and that MAD raises flags for rules that remove detail. MAD and SAnR augment existing approaches by supporting gradually adding detail and analyzing level generation histories, which ultimately helps designers make better levels and level generators.

7.2 RELATED WORK

Evaluating content generators and their output is a key open research problem [STN16; SMS⁺17; YT18]. Generators can be analyzed in terms of generated content, e.g., Summerville et al. evaluate metrics for difficulty, visual aesthetics and enjoyment of platform games [SMS⁺17]. We take an authoring perspective on level grammars. Our approach stands apart by also taking into account how generated levels are generated. This enables level designers to relate qualities of generated levels back to the source code of the generator (grammar rules) and make targeted improvements.

Level grammars are under-specified, since they also generate many levels that are bad with respect to design constraints. The challenge is authoring a set of rules that efficiently generates varied and well-structured results capturing design intent while limiting the recursion. Smith and Mateas propose explicitly describing design spaces as an answer set programs, and show generators can be sculpted for a variety of content domains [SM11a]. Van der Linden et al. focus on improving authoring and controlling level generators by expressing gameplay-related design constraints. They use graph grammars to encode these constraints, and generate action graphs that associate player actions and content for generating complete layouts of game levels [vdLLB13]. We refer to a survey of van der Linden et al. for a wider discussion on techniques for procedural dungeon generation [vdLLB14].

We relate our work to other content generators that use grammars. Tracery is a grammar-based tool for authoring stories and art as structured strings that has been used for generating names, descriptions, stories in poetry, Twitter bots and games¹ [CKM15]. PuzzleScript is a language and authoring environment which uses rewrite rules to express puzzle mechanics². Ludoscope is a visual environment for authoring level transformation pipelines as grammars that builds upon the mission and spaces framework [Dor10; DB11]. Pipelines consist of modules that contain grammars, alphabets and recipes that transform level artifacts such as strings, tile maps, graphs and Voronoi diagrams. In particular, recipes are crucial to control the generation and focus the application of rules for obtaining aesthetically pleasing levels. Recipes parameterize modules with instructions, that determine the ordering of rules and limit how often rules work. Member values annotate tiles with extra information. Both help reducing the generation space, but neither are well-suited to check qualities off-line and independently. Ludoscope is neither extensively documented, nor currently available as open source software. Karavolos et al. report experiences on applying Ludoscope to a platformer and a dungeon crawl game, which require very different transformation pipelines [KBB15]. Our approach closely follows the pipeline structure of Ludoscope, but it improves upon its capabilities for analyzing grammar and level quality.

¹<http://tracery.io>

²<https://www.puzzlescript.net>

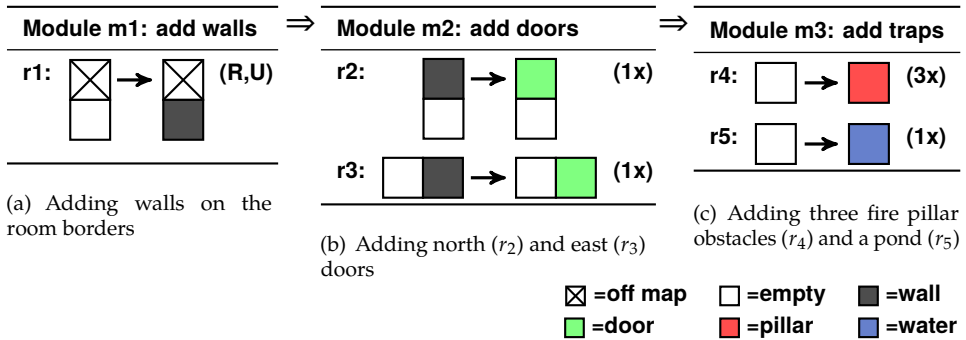


Figure 7.1: Level transformation pipeline consisting of three modules

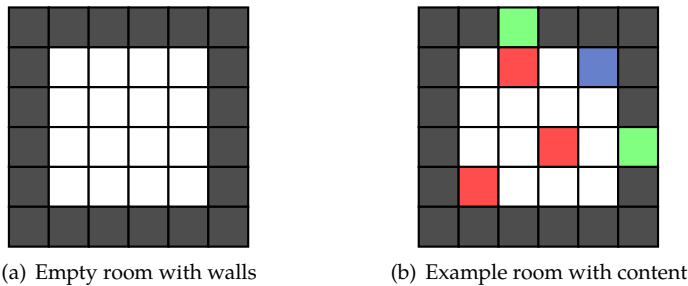


Figure 7.2: Tile maps that are input and output of the pipeline

7.3 GRAMMARS FOR LEVEL GENERATION

Here, we introduce quality issues in grammar-based level design using a simple example that generates a room for a dungeon crawler, which illustrates some of the challenges that arise during authoring grammars. It isolates problems that have larger more complex forms in practice, e.g., in *Unexplored*. We relate questions designers might have in Section 7.3.2 to technical challenges in Section 7.3.3.

7.3.1 Introductory Example

In dungeon crawlers, tile maps often represent rooms connected by pathways. Our level generation pipeline, shown in Figure 7.1, generates rooms with two doors connecting to a larger dungeon. It consists of three modules of grammar rules that represent sequential level transformation phases. The grammar rules rewrite pieces of the tile map matched by their pattern on their left hand side to the pattern on their right hand side. The pipeline takes an empty tile map as input, e.g., of 6x6 tiles. Each

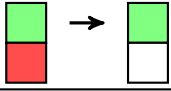

module m4a: remove obstacles	MAD score	heat map
r6:  (R,U)	-1 (+0-1)	
(a) Module removing pillars	(b) MAD score and heat map	

Figure 7.3: Module for removing pillars that block doors

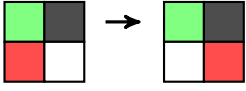



module m4b: move obstacles	MAD score	heat map
r7:  (R,U)	0 (+1-1)	
r8:  (R,U)	0 (+1-1)	
(a) Moving pillars left (r_7) or right (r_8)	(b) MAD score and heat map	

Figure 7.4: Module for moving pillars that block doors

phase randomly selects and applies rules, gradually adding detail. Many levels can result, and as we will see, not all of these are what a designer might deem desirable.

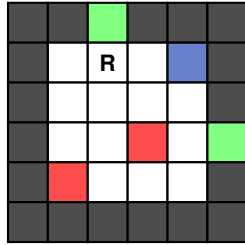
First, module m_1 adds walls on the borders of the tile map (Figure 7.1(a)). It contains one rule called r_1 , whose left hand pattern matches on an empty tile on the north edge of the map. Grammar rule r_1 replaces an empty tile on the north edge of the map with a wall. Rules can have modifier symbols to its right. The (U) symbol to the right indicates that rule r_1 is applied as many times as possible. The (R) symbol indicates that rule r_1 is also applied to the east, south and west borders of the map. The result of module m_1 is always a tile map with walls on its borders, e.g., Figure 7.2(a) is the output at 6x6.

Next, module m_2 adds doors in the north and east walls that connect the room to other parts of the dungeon (Figure 7.1(b)). The rules r_2 and r_3 respectively add a door in the north and east walls. These rules are applied exactly once (1x).

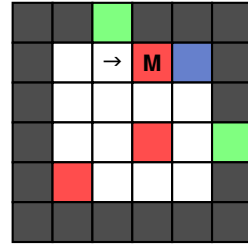
Finally, module m_3 introduces challenge (Figure 7.1(c)). Rule r_4 places three fire pillars, traps that set players on fire if they remain close too long. In addition, rule r_5 adds a pond of water the player can use to extinguish the flames.

7.3.2 Level designer questions

The pipeline of Figure 7.1 can also generate problematic levels. For instance, in Figure 7.2(b), a fire pillar in front of the north door prevents players from passing.

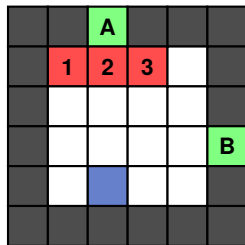


(a) Module m4a removed a pillar at R

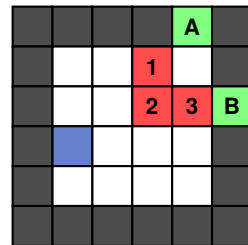


(b) Module m4b moved pillar M

Figure 7.5: Repairing the example level of Figure 7.2(b) in two ways



(a) No space to move pillar 2 away from door A



(b) Moving pillar 3 can block door A
Water remains unreachable

Figure 7.6: Levels that cannot be repaired by Module m4b

One way to fix this is to *patch* the level by removing obstacles, as shown in Figure 7.3(a) results in Figure 7.5(a). However, fewer pillars than intended may reduce the difficulty. Another way is moving obstacles away from doors, as shown in Figure 7.4(a), which results in Figure 7.5(b). Unfortunately, other problematic output still exists, e.g., Figure 7.6. Authoring level grammars is hard, even for this tiny example. Questions about quality a designer might have are:

1. **Efficiency.** Do the grammar rules efficiently generate levels, or is time wasted on overwritten dead content?
2. **Effectiveness.** Do the grammar rules effectively generate levels that contain all the intended objects, composite structures, problems and solutions, or are some parts missing?
3. **Root-cause analysis.** Given a level with a problem, by which rules were the affected tiles generated?
4. **Bug-fixing.** Does changing a rule improve levels, or does it also introduce new problems?
5. **Bug-free.** How can unwanted situations be prevented and removed from the level generation space?

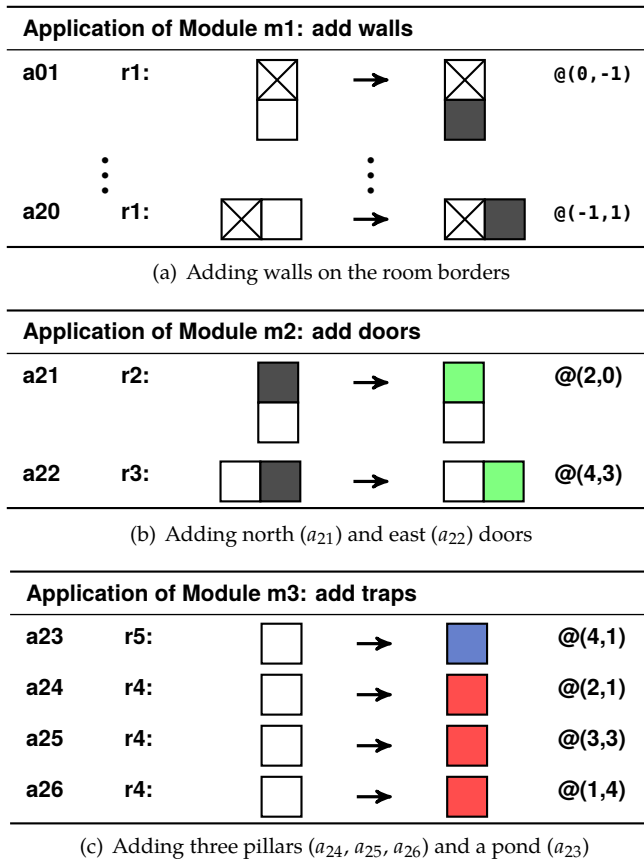


Figure 7.7: Level generation history showing how rules generated the example level of Figure 7.2(b)

Other relevant questions not further discussed here are, e.g.,

- **Playability.** Are the challenges of all generated levels solvable, or are there ways in which players can get stuck?
- **Challenge.** Are the levels challenging to play?

7.3.3 Challenges

Here, we identify technical challenges that need to be addressed for answering questions of level designers described in Section 7.3.2.

1. **Static analysis and metrics.** Profiling the applications of rules helps to assess efficiency measuring (relative) times and amounts. However, static analysis

may also help predict rule efficiency. Upper bounds on rule applications enable reasoning about worst-case scenarios. Left hand patterns that can never match indicate dead code. In addition, metrics can help assess to which extent rules contribute to generating an intended result, to find bad rules.

2. **Analyzing the level generation space.** Viewed as a state-space exploration problem, rules might rewrite levels to prior states. For a given level, the shorter its trace of rewrites, the more efficient its generation.
3. **Expressing and analyzing level qualities.** Grammar rules lack ways to specify properties at specific points in the pipeline, e.g., if objects are (not) adjacent, contained, intact or missing. Designers need an additional formalism for effectively specifying properties that intuitively capture design intent. To see how qualities evolve, levels can be checked against these properties after each transformation.
4. **History analysis.** Generators produce tile maps by applying grammar rules in sequence, e.g. Figure 7.7. However, these generation histories are usually not stored. For identifying rules that impact tiles, or groups of tiles, designers require an analysis of the level transformation history.
5. **Impact analysis.** Assessing the impact of rules on many generated results requires isolating rule effects. The position in the pipeline scopes the locality of impact, and a dependency analysis can exclude side-effects, but an exhaustive impact analysis requires generating examples.
6. **Test Automation.** Testing the impact of changes on all possible levels is not feasible. As a result, levels may exist that contain bugs. The challenge is devising a test harness that generates representative levels for finding bugs.
7. **Debugging.** Identifying and fixing bugs requires appropriate views and tools for setting break points and making modifications, e.g., selecting one or more adjacent tiles to filter and analyze selected properties.

7.4 GRAMMAR ANALYSIS AND DEBUGGING

We approach the challenges of Section 7.3.3 from a software evolution perspective. We propose two solutions, Metric of Added Detail (MAD) and Specification Analysis Reporting (SAnR). Figure 7.8 schematically shows how designer activities and algorithmic processes (respectively shown as pink and blue rounded rectangles) produce (outgoing arrows) and consume (incoming arrows) artifacts (rectangles). The field of software evolution studies how software evolves over time [Meno8]. As software ages, it conforms less and less to the changing expectations of its users. In addition, for developers it also becomes harder over time to adjust software and maintain its quality. Research includes methods and techniques for analyzing source code and for making changes to improve the software quality. Since game requirements are mainly non-functional and evolve rapidly, these techniques are also vital for game quality.

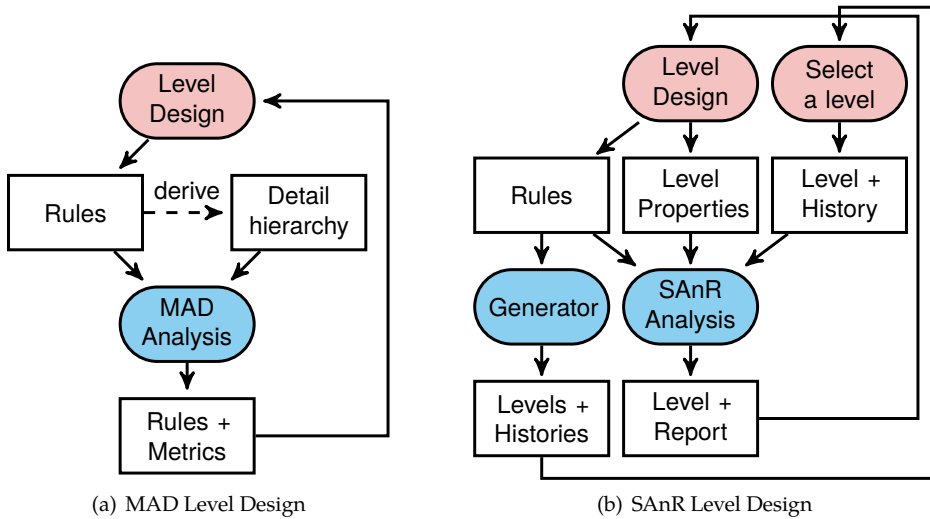


Figure 7.8: Producing MAD and SANR level design artifacts

7.4.1 Metric of Added Detail

Metrics have been proposed to analyze how changes to source code impact software quality. Volume (or size) can be measured by counting Lines Of Code (LOC), and branch points in the control flow of methods can be measured using Cyclometric Complexity (CC). At any moment, metrics are just abstract values, but when studied over time they can provide insight into phenomena and quality, in particular when developers have questions regarding the effect of maintenance and new requirements that require programming. Heitlager et al. describe a software maintainability model [HKV07], which requires that measures are 1) technology independent; 2) simply defined; 3) easy to understand and explain; and 4) enablers of root cause-analysis, relating source code properties to system qualities.

Here we introduce the Metric of Added Detail (MAD), a simple metric for grammars operating on tile maps, which is easy to explain and understand. MAD does not directly predict level quality, but instead measures the effect on detail of individual rules by leveraging the assumption that details are gradually added (Figure 7.8(a)).

We define MAD in Figure 7.9, using the concise functional notation of RASCAL. MAD requires a *detail hierarchy*, represented as a binary relation on grammar symbols (line 2). Rules are represented as lists of tuples of source and target symbols that abstract from tile map dimensions (line 3). The result of the metric adds a *score* element to each tuple that records if detail is added (score +1), removed (score -1) or persisted (score 0) (line 4). The function *getRuleScore* specifies the rule metric as a list

```






1 module util::mad::Metric
2 alias Detail = rel[str greaterSymbol, str lesserSymbol];
3 alias Rule = lrel[str lhs, str rhs];
4 alias RuleScore = lrel[str lhs, str rhs, int score];
5
6 RuleScore getRuleScore(Rule r, Detail d)
7   = [<lhs, rhs, getTileScore(lhs, rhs, d)> | <lhs, rhs> ← r];
8
9 //rewriting a tile
10 int getTileScore(str lhs, str rhs, Detail d){
11   if(<lhs, rhs> in d) return -1; //removes detail
12   else if(<rhs, lhs> in d) return 1; //adds detail
13   else return 0; //retains detail
14 }

```

Figure 7.9: Metric of Added Detail as a RASCAL program

comprehension (lines 6–7). Given a rule and a detail hierarchy, it calculates for each symbol on the left hand side if the right hand side adds or removes detail using the function *getTileScore* (lines 10–14). Displayed as a heat map, the result is aggregated as a sum of tile detail scores.

7.4.2 Deriving Detail Hierarchies

MAD is tool independent and rule parametric, but it requires a detail hierarchy, which needs to be derived. Modules imply a natural hierarchy for tools that use level transformation pipelines, each phase introducing symbols that are more detailed than the last. Using this approach, we derive the following detail hierarchy for the example of Section 7.3.1 Figure 7.1 as follows: {water, pillar} > door > wall > empty, or visually {  ,  } >  >  > .

Competing non-deterministic rules do not sequentially add detail, e.g., r_4 or r_5 adds water or a pillar first. Therefore, deriving a symbol hierarchy for exposing data generated and overwritten within a module is less straightforward. We see the following alternatives:

1. Allow an explicit user-defined detail hierarchy, or derive it from an explicit rule ordering such as a Ludoscope recipe.
2. Assume detail is sequential to the rules in the module.
3. Add the inverse to the relation for symbols with the same rank in the hierarchy, e.g., ‘pillar > water’ and ‘water > pillar’. However, this is not very intuitive.

```

1 start syntax LevelSpec
2   = spec: Property*;
3 syntax Property
4   = property: Condition TileSet;
5 syntax Condition
6   = none: "no"           //tile set is empty
7   | count: INT size "x"; //tile set is of specific size
8 syntax TileSet           //defines a set of tiles (now visible)
9   = tileSet: ID tileName FilterNow FilterWhere;
10 syntax FilterNow        //filters the tile set (now visible)
11  = nowAny:               //empty alternative, no filter
12  | nowAdjacent: "adjacent to" ID tileName;
13 syntax FilterWhere      //filters a tile set (historically)
14  = everAny:              //empty alternative, no filter
15  | everRule: "in" ID ruleName; //topographical location

```

Figure 7.10: Syntax of the Level Property Language in RASCAL

7.4.3 Analyzing Rules with MAD

Using the detail hierarchy derived in Section 7.4.2 we calculate MAD scores for rules of modules *m4a* and *m4b* intended to fix broken levels, shown in Figure 7.3(a) and Figure 7.4(a). Rule r_6 , which removes fire pillars, has a negative effect on detail, as shown in Figure 7.3(b). The effect of rules r_7 and r_8 that instead move them, shown in Figure 7.4(b), is neutral. MAD helps designers assess if rules contribute to generating intended results, and augments intuitions with facts. Rules that remove details may be fixes, but may also cause dead content or regressions in the level generation space that waste time.

7.4.4 Expressing and Analyzing Level Properties

Here we address the challenges of expressing and analyzing level qualities from a Software Language Engineering perspective [Läm18]. We propose Specification Analysis Reporting (SAnR), a technique for analyzing level grammars against level properties. In the mixed-initiative design process shown in Figure 7.8(b), designers author a grammar (rules and modules) and SAnR level properties, a generator generates levels, and the designers selects one level to analyze, for which SAnR generates a report.

SAnR provides a property notation. This is a so-called Domain-Specific Language (DSL), a language that offers appropriate notations and abstractions with expressive

	$a_{01} \dots a_{21}$	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}	a_{27} (alt.)	a_{27} (alt.)	
2x door in walls									
1x water			✓						
3x pillar						✓	✗		
no pillar adjacent to door	✓				✗			✓	
no pillar adjacent to water	✓						✓	✗	
(a) Level Property Language specification encoding level qualities	(b) Level generation report showing how the level of Figure 7.2(b) evolved over time in Figure 7.7						(c) Certain result of module m4a	(d) Possible result of module m4b	

Figure 7.11: Level properties and a level generation report showing two alternatives

power and affordances over a particular problem domain [vDKVoo], in this case specifying properties of tile maps as correct outcomes of tile map transformations.

We show its syntax in Figure 7.10, and give an informal description of its language semantics. Instead of writing new grammar rules, a SAnR level specification is a set of declarative properties, which refer to names used in the grammar (line 1). Given a level history as a sequence of rule-based model transformations, e.g., Figure 7.7, properties can be evaluated at each point in time, yielding either *true* or *false*. Properties work on tile locations, places on tile maps specified by x and y coordinates denoted as $@(x,y)$, the top left tile being $@(0,0)$. A property is a condition on a set of tile locations visible on a tile map (line 3), which must either be empty (line 6) or of a specific size (line 7). The set is built by collecting tile locations using names from the grammar alphabet, e.g., “door” retrieves a set containing each location of a door. On the example of Figure 7.2(b) this yields $\{ @(2,0), @(5,3) \}$, which means “2x door” is true and “1x door” is false. Locations can be filtered in two optional ways.

1. **Adjacency.** The **adjacent to** keyword (lines 10–12), filters locations that do not share at least one side with tiles of another kind, e.g., “door adjacent to pillar”, denotes a set of locations of door tiles next to at least one pillar.
2. **Topography.** The **in** keyword (lines 13–15), filters out locations that were never affected by a rule rewrite. In other words, we use rule names to collect sets of tile locations from the level generation history as “topographical regions”. The resulting set is the intersection between the left and right hand operands. For example “door in walls” gives the set of *door* locations in the region affected by rule *walls*.

7.4.5 Analyzing Level Generation Histories

The SAnR analysis uses properties for generating level generation reports that show when properties were valid, and when they became invalid. For example, given

the level generation history of Figure 7.7, and the properties of Figure 7.11(a), SAnR evaluates the properties after each transformation step, yielding the report of Figure 7.11(b). From the report we read that at step a_{24} transformation r_4 : $\square \rightarrow \blacksquare @ (2,1)$ places a pillar in front of the north door, which invalidates the property “no pillar adjacent to door”.

7.4.6 Analyzing Rule Impact

SAnR can also be used to analyze the impact of new rules on existing levels with respect to level properties. For instance, we can spot problems at alternative steps a_{27} in the report of Figure 7.11 caused by modules $m4a$ and $m4b$ intended as fixes, shown in Figure 7.3(a) and Figure 7.4(a). On the one hand, Figure 7.11(c) shows that when module $m4a$ removes the pillar with transformation r_6 : $\begin{matrix} \blacksquare \\ \blacksquare \end{matrix} \rightarrow \begin{matrix} \blacksquare \\ \square \end{matrix} @ (2,0)$ this breaks the property “3x pillar”. On the other hand, Figure 7.11(d) shows that when module $m4b$ moves the pillar to the east with transformation r_7 : $\begin{matrix} \blacksquare & \blacksquare \\ \square & \square \end{matrix} \rightarrow \begin{matrix} \blacksquare & \blacksquare \\ \square & \blacksquare \end{matrix} @ (2,0)$ this breaks the property “no pillar adjacent to water”.

7.5 PRELIMINARY EVALUATION

Here, we report on a preliminary evaluation of the use of MAD and SAnR in the implementation of a prototype level generator called Ludoscope Lite.

7.5.1 Implementation of Ludoscope Lite

Ludoscope Lite (LL) is a light weight version of Ludoscope intended for rapid prototyping, research and experimentation with analysis and generation techniques for making better grammar-based game levels and generators. Its focus is initially on designing and validating approaches for tile maps, which are later implemented and applied in Ludoscope. We use language work bench [EVV⁺13] and meta-programming language RASCAL³ [KvdSV09] to implement MAD and SAnR as separate reusable modules and integrate both in LL⁴.

Table 7.1 gives an overview of the components of LL and their size in Lines of Code (LOC) relative to Ludoscope. Of course, the user-friendly IDE of Ludoscope has many features LL lacks, explaining the size difference. LL integrates a grammar-based parser that reads the storage format of Ludoscope. The ultimate goal is compatibility, sharing syntax and semantics for generating and analyzing rules. We apply test-driven development, encoding expected behaviors for most of its features in a combination of unit and integration tests for regression testing. The histories and reports shown

³<https://www.rascal-mpl.org>

⁴<https://github.com/visknut/LudoscopeLite>

Table 7.1: Source code size of Ludoscope and Ludoscope Lite

Component	Ludoscope (KLOC)	LL (KLOC)
IDE (features differ)	10.5	0.3
Parser + execution	10	1.7 + 0.4
Test + test data	?	1.5 + 0.7
Metric of Added Detail	not yet	0.1
Level Property Language	not yet	0.3
Extension wrappers	-	0.4
Total	20.5	5.5

Table 7.2: SAnR data on the example the pipeline of Figure 7.1 and its two extensions, modules *m4a* and *m4b*

Data	Example	+ <i>m4a</i>	+ <i>m4b</i>
Unique histories	9846	9858	9844
Unique tile maps	9171	9014	8775
Broken tile maps	6254	6132	4613
Bugs found	2	2	4

Table 7.3: SAnR level generation reports for 10K random executions. The rules r_n refer to Figure 7.1, Figure 7.3(a) and Figure 7.4(a)

Property	Example	+ <i>m4a</i>	+ <i>m4b</i>
2x door in walls	-	-	-
1x water	-	-	-
3x pillar	-	r6 (3226x)	r7 (111x) r8 (112x)
no pillar adjacent to door	r5 (3164x)	-	r5 (438x)
no water adjacent to pillar	r5 (5686x)	r5 (5209x)	r5 (5482x)

in this paper are generated by LL, which currently still generates them as strings. A more user friendly visualization is work in progress.

7.5.2 Test Automation

We use LL to evaluate SAnR on the running example⁵ of Section 7.3.1. We wish to learn if LL and SAnR can help automate tests, and run 10K random executions (or

⁵There is one difference, LL implements Ludoscope recipes for limiting the amount of times that rules are applied. As a side-effect, this limits sequences and reduces the level generation space.

simulations) on the pipeline Figure 7.1, and its extensions, shown in Figure 7.3(a) and Figure 7.4(a), which makes 30K executions total. For each execution, we record the model transformation history and use SAnR and the properties of Figure 7.11(a) to obtain a report.

Table 7.2 displays an overview of the results, which were obtained in about 10 minutes of run time. The unique number of histories is lower than 10K because some executions yielded the same transformations. In addition, different transformation sequences can produce the same tile map, which explains why there are fewer unique tile maps. We consider a tile map *broken* when not all SAnR property are satisfied. In addition, Table 7.3 shows which rules break properties (in how many histories) for each pipeline version, which helps designers compare and analyze causes.

We gain the following insights. The test automation approach is feasible, and issues can be found in seconds. In addition, by relating the number of unique outputs to the number of broken outputs we can get an idea how serious issues are. Naturally, 10K random executions says nothing about test coverage, but it improves upon random manual testing. We confirm that module *m4a* is a bad fix. We note that although extension *m4b* increases the number bugs, it also generates fewer broken tile maps. Clearly, the pipeline still requires fixes. Of course, the example is small and not representative of the size and complexity of transformation pipelines of games such as Unexplored. However, our test automation setup is reusable, and enables testing other grammars with larger pipelines too.

7.6 DISCUSSION

MAD and SAnR provide a means for answering designer questions of Section 7.3.2. Here we discuss the benefits and limitations of the approaches and threats to validity.

7.6.1 MAD Level Design

MAD gives a partial answer to the question if rules generate levels efficiently. The metric helps designers identify rules that remove detail, and possibly waste time on generating cause dead content. It supports the single responsibility principle, exposing modules add many details at once. However, MAD does not address the challenge of analyzing the state space. At best, it can help identify rules that may lead to longer level generation traces. In addition, we do not know if MAD can be used for data structures other than tile maps, e.g., for grammars that work on graphs. Finally, MAD is not yet empirically validated.

7.6.2 *SAnR Level Design*

SAnR properties enable analyzing how effectively rules generate intended levels, e.g. for simple tile adjacency, counting, missing tiles, and topographical inclusion. Properties depend only on the names of rules and tiles, which separates concerns but complicates refactoring grammar rules. SAnR analyzes levels by checking properties against generation histories, and assumes these are correctly generated. Therefore, SAnR reports are only as good as the grammar engine, which may also contain bugs. Of course, our approach is not the first that checks simple invariant conditions. However, to the best of our knowledge, checking properties that use level generation histories and grammar rule names to collect topographical regions of tile locations is new.

SAnR can help designers analyze quality and remove unwanted situations from the level generation space by identifying transformations and rules that break properties. However, those rules may not be the root cause of the problem, which can originate earlier in the pipeline. In addition, it is hard for developers to analyze the history, since it is not clear where the branch points in the generation process are, and how alternatives would have played out. Finally, the expressive range of properties is currently still rather limited, and a formal semantics relating properties and histories is not yet defined.

7.7 CONCLUSION

This paper proposes two novel techniques that aim to improve the quality of grammar-based procedural level generation for grammars that work on tile maps. The first, is the Metric of Added Detail (MAD), a novel metric that indicates if a grammar rule adds or removes detail to a tile map. The second, is Specification Analysis Reporting (SAnR), a technique that offers level property language for expressing level qualities. SAnR analyzes and reports how these properties evolve over time in level generation histories. We demonstrated the feasibility of MAD and SAnR with LudoScope Lite, a light-weight version of Ludoscope intended to study level quality.

Our preliminary evaluation shows that SAnR can express and analyze simple level properties, and that MAD is intuitive and raises flags for rules that remove detail. In addition, SAnR can be used in test automation. MAD and SAnR augment existing approaches by supporting gradually adding detail and analyzing level generation histories, which ultimately helps designers make better levels and level generators. Of course, LL is an academic research prototype that is not yet extensively validated in practice.

7.7.1 *Future Work*

Future work includes the following.

- **Validation.** A case study on Boulder Dash is current work. We also plan to study Unexplored to identify which additional SAnR property features are needed to express design intent more fully, e.g., better filters, validity ranges, and for shapes, paths and relative positions. We hope to identify bugs that would otherwise be hard or impossible to find.
- **Analyses.** Additional analyses on rule dependencies, and partial orderings may be identified of different rule orders generating the same levels, e.g., for increasing test coverage and level generation variety. For assessing the variety of generated content, existing metrics can be reused. For instance, Smith and Whitehead assess the expressive range of a generator by comparing metrics for linearity and leniency of platform levels [SW10].
- **Generation.** Here we use SAnR for analyzing level generation histories after they are generated. However, by integrating SAnR into a level generator we could also prune the search space and filter out potential unwanted levels before they are ever produced. A feasibility study can assess the impact on efficiency and scalability of this approach.
- **Formal semantics.** Reproducible dynamic analyses require a formal semantics for the execution of generative grammars, separate from tools and games that interpret them.
- **Parsing.** We observe that ambiguous grammars for parsing and level grammars generating the same tile map with different rule orderings are related. Given a bugged tile map, how many different rule orderings can reproduce it? When changing the rules, can the new rules produce the tile map with a different generation history?
- **Debugging.** Debugging level grammars requires an interactive debugger, in particular for back in time debugging, exploring what-if scenarios and saving and replaying generated levels while testing new rules. Additional visualizations are needed to see how the generation space unfolds.

Acknowledgements

We thank Paul Klint, Anders Bouwer, Rafael Bidarra and the anonymous reviewers for their insightful comments that helped improve this paper. We thank Joris Dormans for collaborating with us and answering our many questions about the design of Ludoscope.

BIBLIOGRAPHY

- [Aar11] E. Aarseth. "Define Real, Moron! Some Remarks on Game Ontologies". In: *DIGAREC Keynote-Lectures 2009/10*. Ed. by S. Günzel, M. Liebe, and D. Mersch. DIGAREC 6. Potsdam UP, 2011, pp. 50–69 (cit. on p. 42).
- [AC15] E. Aarseth and G. Calleja. "The Word Game: The Ontology of an Undefinable Object". In: *Proceedings of the 10th International Conference on the Foundations of Digital Games, FDG 2015, Pacific Grove, CA, USA, June 22–25, 2015*. Ed. by J. P. Zagal, E. MacCallum-Stewart, and J. Togelius. Society for the Advancement of the Science of Digital Games, 2015 (cit. on p. 41).
- [ASSo3] E. Aarseth, S. M. Smedstad, and L. Sunnanå. "A Multi-Dimensional Typology of Games". In: *Proceedings of the 2003 DiGRA International Conference: Level Up, DiGRA 2003, Utrecht, The Netherlands, November 4–6, 2003*. Ed. by M. Copier and J. Raessens. Utrecht University, 2003 (cit. on pp. 28, 43).
- [AdGC⁺15] M. Abbadi, F. di Giacomo, A. Cortesi, P. Spronck, C. Giulia, and G. Maggiore. "High Performance Encapsulation in Casanova 2". In: *Proceedings of the 7th Computer Science and Electronic Engineering Conference, CEEC, Colchester, UK, September 24–25, 2015*. IEEE, 2015, pp. 201–206. DOI: 10.1109/CEEC.2015.7332725 (cit. on pp. 107, 108).
- [Abb17] M. Abbadi. "Casanova 2: A Domain-Specific Language for General Game Development". PhD thesis. Tilburg University, Sept. 2017 (cit. on pp. 107, 108, 128).
- [ADC⁺15] M. Abbadi, F. Di Giacomo, A. Cortesi, P. Spronck, G. Costantini, and G. Maggiore. "Casanova: A Simple, High-Performance Language for Game Development". In: *Serious Games – Proceedings of the 1st Joint International Conference on Serious Games, JCSG 2015, Huddersfield, UK, June 3–4, 2015*. Ed. by S. Göbel, M. Ma, J. Baalsrud Hauge, M. F. Oliveira, J. Wiemeyer, and V. Wendel. Springer, 2015, pp. 123–134. ISBN: 978-3-319-19126-3. DOI: 10.1007/978-3-319-19126-3_11 (cit. on pp. 107, 108).
- [ALY15] R. Abela, A. Liapis, and G. N. Yannakakis. "A Constructive Approach for the Generation of Underwater Environments". In: *Proceedings of the 6th International Workshop on Procedural Content Generation in Games, PCG 2015, Pacific Grove, CA, USA, June 22–25, 2015*. Society for the Advancement of the Science of Digital Games, 2015 (cit. on p. 41).
- [AD12] E. Adams and J. Dormans. *Game Mechanics: Advanced Game Design*. 1st ed. Thousand Oaks, CA, USA: New Riders Publishing, 2012. ISBN: 9780321820273 (cit. on pp. 6, 7, 61, 136–138, 152, 162, 163, 172, 184–186, 189, 192, 194).
- [ARo6] E. Adams and A. Rollings. *Fundamentals of Game Design*. 1st ed. Prentice Hall, 2006. ISBN: 9780131687479 (cit. on p. 163).
- [AJR12] N. Ahmadi, M. Jazayeri, and A. Repenning. "Engineering an Open-Web Educational Game Design Environment". In: *Proceedings of the 19th Asia-Pacific Software Engineering Conference, APSEC 2012, Hong Kong, China, December 4–7, 2012*. IEEE, 2012, pp. 867–876. ISBN: 978-1-4673-4930-7. DOI: 10.1109/APSEC.2012.88 (cit. on p. 91).
- [Ahm11] N. Ahmadi. "Beyond Upload and Download: Enabling Game Design 2.0". In: *End-User Development – Proceedings of the 3rd International Symposium, IS-EUD 2011, Torre Canne, Italy, June 7–10, 2011*. Ed. by M. F. Costabile, Y. Dittrich, G. Fischer, and A. Piccinno. Vol. 6654. LNCS. Springer, 2011, pp. 371–374. ISBN: 978-3-642-21530-8. DOI: 10.1007/978-3-642-21530-8 (cit. on p. 91).

- [Ahm12] N. Ahmadi. "Broadening Educational Game Design using the World Wide Web". PhD thesis. Università della Svizzera Italiana – Faculty of Informatics, 2012 (cit. on pp. 91, 128).
- [ASU86] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986. ISBN: 0-201-10088-6 (cit. on pp. 34, 101).
- [AP03] M. Alanen and I. Porres. "Difference and Union of Models". In: «UML» 2003 – *The Unified Modeling Language. Modeling Languages and Applications – Proceedings of the 6th International Conference, San Francisco, CA, USA, October 20–24, 2003*. Ed. by P. Stevens, J. Whittle, and G. Booch. Vol. 2863. LNCS. Springer, 2003, pp. 2–17. ISBN: 978-3-540-45221-8. DOI: 10.1007/978-3-540-45221-8_2 (cit. on pp. 204, 206, 208, 209, 211, 230).
- [ARC⁺14] M. T. C. F. Albuquerque, G. L. Ramalho, V. Corruble, A. L. M. Santos, and F. Freitas. "Helping Developers to Look Deeper inside Game Sessions". In: *Proceedings of the 13th Brazilian Symposium on Computer Games and Digital Entertainment, SBGAMES 2014, Porto Alegre, RS, Brazil, November 12–14, 2014*. IEEE, 2014, pp. 31–40. ISBN: 978-1-4799-8065-9. DOI: 10.1109/SBGAMES.2014.28 (cit. on p. 41).
- [AIS⁺77] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language – Towns, Buildings, Construction*. Oxford University Press, 1977. ISBN: 978-0-19-501919-3 (cit. on p. 45).
- [AdS13a] M. S. Almeida and F. S. da Silva. "Requirements for Game Design Tools: a Systematic Survey". In: *Proceedings of the 12th Brazilian Symposium on Games and Digital Entertainment, São Paulo, Brazil, October 16–18, 2013, SBGames 2013*. 2013 (cit. on p. 126).
- [AVG⁺13] M. S. O. Almeida, L. G. Valentin, R. A. Gonçalves, and F. S. C. da Silva. "Towards a Game Design Patterns Suggestion Tool: The Documentation of a Computerized Textual Analysis Experiment". In: *Proceedings of the 12th Brazilian Symposium on Computer Games and Digital Entertainment, SBGames 2013, São Paulo, Brazil, October 16–18, 2013*. SBGames.org, 2013 (cit. on p. 41).
- [AdS13b] M. S. O. Almeida and F. S. C. da Silva. "A Systematic Review of Game Design Methods and Tools". In: *Entertainment Computing – Proceedings of the 12th International Conference, ICEC 2013, São Paulo, Brazil, October 16–18, 2013*. Ed. by J. C. Anacleto, E. W. G. Clua, F. S. C. da Silva, S. Fels, and H. S. Yang. Vol. 8215. LNCS. Springer, 2013, pp. 17–29. ISBN: 978-3-642-41106-9. DOI: 10.1007/978-3-642-41106-9_3 (cit. on p. 126).
- [AÇM09] D. Altunbay, M. E. Çetinkaya, and M. G. Metin. "Model-driven Approach for Board Game Development". In: *Proceedings of the 1st Turkish Symposium of Model-Driven Software Development, TMODELS 2009, Ankara, Turkey, May 20, 2009*. Bilkent University, 2009 (cit. on p. 115).
- [AR10] V. Alves and L. Roque. "A Pattern Language for Sound Design in Games". In: *Proceedings of the 5th Audio Mostly Conference: A Conference on Interaction with Sound, AM 2010, Piteå, Sweden, September 15–17, 2010*. ACM, 2010, pp. 1–8. ISBN: 978-1-4503-0046-9. DOI: 10.1145/1859799.1859811 (cit. on p. 48).
- [AR11a] V. Alves and L. Roque. "A Deck for Sound Design in Games: Enhancements based on a Design Exercise". In: *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology, ACE 2011, Lisbon, Portugal, November 8–11, 2011*. ACM, 2011, pp. 1–8. ISBN: 978-1-4503-0827-4. DOI: 10.1145/2071423.2071465 (cit. on pp. 41, 48).

- [AR11b] V. Alves and L. Roque. "An Inspection on a Deck for Sound Design in Games". In: *Proceedings of the 6th Audio Mostly Conference: A Conference on Interaction with Sound, AM 2011, Coimbra, Portugal, September 7–9, 2011*. ACM, 2011, pp. 15–22. ISBN: 978-1-4503-1081-9. DOI: 10.1145/2095667.2095670 (cit. on p. 48).
- [AR13] V. Alves and L. Roque. "Design Patterns in Games; The Case for Sound Design". In: *Workshop Proceedings of the 8th International Conference on the Foundations of Digital Games, as part of the 2nd Workshop on Design Patterns in Games, DPG 2013, Chania, Crete, Greece, May, 14–17, 2013*. Society for the Advancement of the Science of Digital Games, 2013 (cit. on pp. 41, 48).
- [AGM⁺06] V. Alves, R. Gheyi, T. Massoni, U. Kulesza, P. Borba, and C. Lucena. "Refactoring Product Lines". In: *Proceedings of the 5th International Conference on Generative Programming and Component Engineering, GPCE 2006, Portland, Oregon, USA, October 22–26, 2006*. ACM, 2006, pp. 201–210. ISBN: 1-59593-237-2. DOI: 10.1145/1173706.1173737 (cit. on p. 35).
- [AS10] A. Ampatzoglou and I. Stamelos. "Software Engineering Research for Computer Games: A Systematic Review". In: *Information & Software Technology* 52.9 (2010), pp. 888–901. DOI: 10.1016/j.infsof.2010.05.004 (cit. on p. 126).
- [Ando8a] E. F. Anderson. "On the Definition of Non-Player Character Behaviour for Real-Time Simulated Virtual Environments". PhD thesis. Bournemouth University, Apr. 2008 (cit. on pp. 70, 72, 128).
- [Ando8b] E. F. Anderson. "Scripted Smarts in an Intelligent Virtual Environment: Behaviour Definition Using a Simple Entity Annotation Language". In: *Proceedings of the 2008 Conference on Future Play: Research, Play, Share, Future Play 2008, Toronto, Ontario, Canada, November 3–5, 2008*. ACM, 2008, pp. 185–188. ISBN: 978-1-60558-218-4. DOI: 10.1145/1496984.1497016 (cit. on pp. 41, 72).
- [AR09] M. Araújo and L. Roque. "Modeling Games with Petri Nets". In: *Proceedings of the 3rd annual DiGRA conference Breaking New Ground: Innovation in Games, Play, Practice and Theory, DiGRA 2009, London, UK, September 1–4, 2009*. Ed. by T. Krzywinska, H. W. Kennedy, and B. Atkins. Digital Games Research Association, 2009 (cit. on pp. 41, 58, 137, 162).
- [Aše17] D. Ašeriškis. "Modeling and Evaluation of Software System Gamification Elements". PhD thesis. Kaunas University of Technology, 2017. ISBN: 978-609-02-1375-9 (cit. on pp. 93, 128).
- [ABD17] D. Ašeriškis, T. Blažauskas, and R. Damaševičius. "UAREI: A Model for Formal Description and Visual Representation /Software Gamification". In: *DYNA* 84.200 (Mar. 2017), pp. 326–334. ISSN: 0012-7353. DOI: 10.15446/dyna.v84n200.54017 (cit. on p. 93).
- [AD17] D. Ašeriškis and R. Damaševičius. "Player Type Simulation in Gamified Applications". In: *Proceedings of the IVUS International Conference on Information Technology, Kaunas, Lithuania, April 28, 2017*. Ed. by R. Damaševičius, T. Krilavičius, and A. Lopata. Vol. 1856. CEUR-WS, 2017, pp. 1–7 (cit. on p. 93).
- [Ayc16] J. Aycok. "Endgame". In: *Retrogame Archeology: Exploring Old Computer Games*. Springer, 2016, pp. 205–213. ISBN: 978-3-319-30004-7. DOI: 10.1007/978-3-319-30004-7 (cit. on p. 82).

- [AH14] A. Azadegan and C. Hartevelde. "Work for or Against Players: On the Use of Collaboration Engineering for Collaborative Games". In: *Proceedings of Workshops Colocated with the 9th International Conference on the Foundations of Digital Games – as part of the 3rd Workshop on Design Patterns in Games, DPG 2014, Liberty of the Seas, Caribbean, April 3–7, 2014*. Society for the Advancement of the Science of Digital Games, 2014. ISBN: 978-0-9913982-3-2 (cit. on pp. 41, 50).
- [BSV14] H. Bäärnhielm, D. Sundström, and M. Vejdemo-Johansson. "A Haskell Sound Specification DSL: Ludic Support and Deep Immersion in Nordic Technology-Supported LARP". In: *The Monad Reader 23* (2014). Ed. by E. Z. Yang (cit. on p. 108).
- [BD10] S. Bakkes and J. Dormans. "Involving Player Experience in Dynamically Generated Missions and Game Spaces". In: *Proceedings of the 11th International Conference on Intelligent Games and Simulation, GAME-ON 2010, Leicester, United Kingdom, November 17–19, 2010*. Ed. by A. Ayesh. 2010, pp. 72–79. ISBN: 978-90-77381-58-8 (cit. on p. 41).
- [BBA⁺08] D. Balas, C. Brom, A. Abonyi, and J. Gemrot. "Hierarchical Petri Nets for Story Plots Featuring Virtual Humans". In: *Proceedings of the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2008, Stanford, California, USA, October 22–24, 2008*. Ed. by M. Mateas and C. Darken. AAAI, 2008, pp. 2–9. ISBN: 978-1-57735-392-8 (cit. on pp. 37, 58).
- [BDF⁺17a] A. Baldwin, S. Dahlskog, J. M. Font, and J. Holmberg. "Mixed-Initiative Procedural Generation of Dungeons using Game Design Patterns". In: *2017 IEEE Conference on Computational Intelligence and Games, CIG 2017, New York, NY, USA, August 22–25, 2017*. 2017, pp. 25–32. ISBN: 978-1-5386-3233-8. DOI: 10.1109/CIG.2017.8080411 (cit. on pp. 37, 69, 70).
- [BDF⁺17b] A. Baldwin, S. Dahlskog, J. M. Font, and J. Holmberg. "Towards Pattern-Based Mixed-initiative Dungeon Generation". In: *Proceedings of the 12th International Conference on the Foundations of Digital Games, FDG 2017 as part of the 8th International Workshop on Procedural Content Generation, PCG 2017, Hyannis, Massachusetts, USA, August 14–17, 2017*. ACM, 2017, pp. 1–10. ISBN: 978-1-4503-5319-9. DOI: 10.1145/3102071.3110572 (cit. on pp. 41, 69, 70).
- [BC04] E. Baniassad and S. Clarke. "Theme: An Approach for Aspect-Oriented Analysis and Design". In: *Proceedings of the 26th International Conference on Software Engineering, ICSE 2004, Edinburgh, UK, May 23–28, 2004*. IEEE, 2004, pp. 158–167. ISBN: 0-7695-2163-0. DOI: 10.1109/ICSE.2004.1317438 (cit. on p. 35).
- [BT14] G. A. B. Barros and J. Togelius. "Exploring a Large Space of Small Games". In: *Proceedings of the 2014 IEEE Conference on Computational Intelligence and Games, CIG 2014, Dortmund, Germany, August 26–29, 2014*. 2014, pp. 1–2. ISBN: 978-1-4799-3547-5. DOI: 10.1109/CIG.2014.6932922 (cit. on pp. 37, 101).
- [Bar16] R. A. Bartle. *MMOs from the Inside Out: The History, Design, Fun, and Art of Massively-Multiplayer Online Role-Playing Games*. Apress, 2016. ISBN: 978-1-4842-1723-8. DOI: 10.1007/978-1-4842-1724-5 (cit. on p. 109).
- [BK05] A. Begel and E. Klopfer. "Starlogo TNG: An Introduction to Game Development". In: *Journal of E-Learning 53* (2005) (cit. on p. 91).
- [BG16] C. Bell and M. Goadrich. "Automated Playtesting with RECYCLEd CARDSTOCK". In: *Game & Puzzle Design 2.1* (2016), pp. 71–83. ISSN: 2376-5097 (cit. on p. 101).
- [Bey11] L. Beyak. "SAGA: A Story Scripting Tool for Video Game Development". MA thesis. McMaster University – Department of Computing and Software, 2011 (cit. on p. 80).

- [BC11] L. Beyak and J. Carette. "SAGA: A DSL for Story Management". In: *Proceedings IFIP Working Conference on Domain-Specific Languages, DSL 2011, Bordeaux, France, September 6–8, 2011*. Ed. by O. Danvy and C. Shan. Vol. 66. EPTCS. arXiv, 2011, pp. 48–67. DOI: 10.4204/EPTCS.66.3 (cit. on pp. 80, 81).
- [BH06] S. Björk and J. Holopainen. "Games and Design Patterns". In: *The Game Design Reader: A Rules of Play Anthology*. Ed. by K. Salen and E. Zimmerman. MIT Press, 2006, pp. 410–437. ISBN: 0-262-19536-4 (cit. on p. 47).
- [BLH03] S. Björk, S. Lundgren, and J. Holopainen. "Game Design Patterns". In: *Proceedings of the 2003 DiGRA International Conference: Level Up, DIGRA 2003, Utrecht, The Netherlands, November 4–6, 2003*. Digital Games Research Association, 2003, pp. 180–193 (cit. on pp. 41, 47, 161, 186).
- [Bloo4] J. Blow. "Game Development: Harder Than You Think". In: *ACM Queue* 1.10 (Feb. 2004), pp. 28–37. ISSN: 1542-7730. DOI: 10.1145/971564.971590 (cit. on p. 136).
- [Bojo8] N. Bojin. "Language Games/Game Languages: Examining Game Design Epistemologies Through a 'Wittgensteinian' Lens". In: *Journal for Computer Game Culture* 2.1 (2008), pp. 55–71. ISSN: 1866-6124 (cit. on p. 50).
- [Boj10] N. Bojin. "Ludemes and the Linguistic Turn". In: *Proceedings of the International Academic Conference on the Future of Game Design and Technology, Future Play 2010, Vancouver, British Columbia, Canada, May 6–7, 2010*. ACM, 2010, pp. 25–32. ISBN: 978-1-4503-0235-7. DOI: 10.1145/1920778.1920783 (cit. on pp. 41, 50).
- [Bor15] Y. C. Borghini. "An Assessment and Learning Analytics Engine for Games-Based Learning". PhD thesis. University of the West of Scotland, Dec. 2015 (cit. on pp. 56, 128).
- [BCC⁺15] E. Bousse, J. Corley, B. Combemale, J. G. Gray, and B. Baudry. "Supporting Efficient and Advanced Omniscient Debugging for xDSMLs". In: *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering, SLE 2015, Pittsburgh, PA, USA, October 26–27, 2015*. Ed. by R. F. Paige, D. di Ruscio, and M. Völter. ACM, 2015, pp. 137–148. ISBN: 978-1-4503-3686-4. DOI: 10.1145/2814251.2814262 (cit. on p. 231).
- [BA06] C. Brom and A. Abonyi. "Petri Nets for Game Plot". In: *Proceedings of Artificial Intelligence and the Simulation of Behaviour as part of the Workshop on Narrative AI and Games*. AISB, 2006 (cit. on pp. 58, 137, 162).
- [BSH07] C. Brom, V. Šisler, and T. Holan. "Story Manager in 'Europe 2045' Uses Petri Nets". In: *Virtual Storytelling. Using Virtual Reality Technologies for Storytelling – Proceedings of the 4th International Conference, ICVS 2007, Saint-Malo, France, December 5–7, 2007*. Ed. by M. Cavazza and S. Donikian. Vol. 4871. LNCS. Springer, 2007, pp. 38–50. ISBN: 978-3-540-77039-8. DOI: 10.1007/978-3-540-77039-8_4 (cit. on pp. 39, 58).
- [BM10] C. Browne and F. Maire. "Evolutionary Game Design". In: *IEEE Transactions on Computational Intelligence and AI in Games* 2.1 (Mar. 2010), pp. 1–16. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2010.2041928 (cit. on pp. 37, 97, 186, 187, 200).
- [BTS14] C. Browne, J. Togelius, and N. Sturtevant. "Guest Editorial: General Games". In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.4 (Dec. 2014), pp. 317–319. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2014.2369009 (cit. on p. 93).
- [Bro08] C. Browne. "Automatic Generation and Evaluation of Recombination Games". PhD thesis. Queensland University of Technology, Feb. 2008 (cit. on pp. 97, 128).
- [Bro11] C. Browne. *Evolutionary Game Design*. Ed. by S. Zdonik, P. Ning, S. Shekhar, J. Katz, X. Wu, L. C. Jain, D. Padua, X. Shen, and B. Furht. SpringerBriefs in Computer Science. Springer, 2011. ISBN: 978-1-4471-2178-7. DOI: 10.1007/978-1-4471-2179-4 (cit. on p. 97).

- [Bro16] C. Browne. "A Class Grammar for General Games". In: *Computers and Games – Proceedings of the 9th International Conference on Computers and Games, CG 2016, Leiden, The Netherlands, June 29–July 1, 2016*. Ed. by A. Plaat, W. Kusters, and J. van den Herik. Vol. 10068. LNCS. Springer, 2016, pp. 167–182. ISBN: 978-3-319-50935-8. DOI: 10.1007/978-3-319-50935-8_16 (cit. on pp. 41, 97).
- [BPo8] C. Brun and A. Pierantonio. "Model Differences in the Eclipse Modeling Framework". In: *UPGRADE, The European Journal for the Informatics Professional* 9.2 (Apr. 2008), pp. 29–34 (cit. on p. 207).
- [BLo7] J. Brusik and T. Lager. "Developing Natural Language Enabled Games in (Extended) SCXML". In: *Proceedings of the International Symposium on Intelligence Techniques in Computer Games and Simulations, GAME-ON-ASIA 2007, Shiga, Japan, March 1–3, 2007*. EUROSIS, 2007 (cit. on p. 112).
- [Bruo8] J. Brusik. "Dialogue Management for Social Game Characters Using Statecharts". In: *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology, ACE 2008, Yokohama, Japan, December 3–5, 2008*. ACM, 2008, pp. 219–222. ISBN: 978-1-60558-393-8. DOI: 10.1145/1501750.1501801 (cit. on pp. 41, 112).
- [BMS⁺o8] D. Burgos, P. Moreno-Ger, J. L. Sierra, B. Fernández-Manjón, M. Specht, and R. Koper. "Building Adaptive Game-based Learning Resources: The Integration of IMS Learning Design and <e-Adventure>". In: *Simulation & Gaming* 39.3 (July 2008), pp. 414–431. DOI: 10.1177/1046878108319595 (cit. on pp. 76, 77).
- [BTP17] E. Butler, E. Torlak, and Z. Popović. "Synthesizing Interpretable Strategies for Solving Puzzle Games". In: *Proceedings of the 12th International Conference on the Foundations of Digital Games, FDG 2017, Hyannis, Massachusetts, August 14–17, 2017*. Ed. by S. Deterding, A. Canossa, C. Hartevelde, J. Zhu, and M. Sicart. ACM, 2017, pp. 1–10. ISBN: 978-1-4503-5319-9. DOI: 10.1145/3102071.3102084 (cit. on p. 40).
- [CPo9] A. Calleja and G. J. Pace. "A Domain-Specific Embedded Language Approach for the Scripting of Game Artificial Intelligence". In: *Proceedings of the 2nd National Workshop in Information and Communication Technology, WICT 2009, Valletta, Malta, November 17, 2009*. University of Malta, 2009, pp. 1–7 (cit. on p. 107).
- [CP10] A. Calleja and G. J. Pace. "Scripting Game AI: An Alternative Approach using Embedded Languages". In: *Proceedings of the 3rd National Workshop in Information and Communication Technology, WICT 2010, Valletta, Malta, November 16, 2010*. University of Malta, 2010 (cit. on p. 107).
- [CDo9] A. Canossa and A. Drachen. "Patterns of Play: Play-Personas in User-Centred Game Development". In: *Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory, DiGRA 2009, West London, UK, September 1–4, 2009*. Brunel University, 2009 (cit. on pp. 41, 86).
- [CCD⁺o8] M. Carbonaro, M. Cutumisu, H. Duff, S. Gillis, C. Onuczko, J. Siegel, J. Schaeffer, A. Schumacher, D. Szafron, and K. Waugh. "Interactive Story Authoring: A Viable form of Creative Expression for the Classroom". In: *Computers & Education* 51.2 (Sept. 2008), pp. 687–707. ISSN: 0360-1315. DOI: 10.1016/j.compedu.2007.07.007 (cit. on pp. 78, 79).
- [Chao7] A. J. Champandard. *Behavior Trees for Next-Gen Game AI*. AIGameDev.com. Lecture delivered at the Game Developers Conference, GDC 2007. Dec. 2007. URL: <http://aigamedev.com/open/article/behavior-trees-part1/> (visited on Oct. 22, 2018) (cit. on pp. 38, 73, 74, 137).
- [Cha12] A. J. Champandard. *Understanding the Second-Generation of Behavior Trees – AltDev-Conf*. AIGameDev.com. Presentation. Feb. 2012. URL: <http://aigamedev.com/insider/tutorial/second-generation-bt/> (visited on Oct. 22, 2018) (cit. on pp. 73, 74).

- [CCK12] C. Chang, T. Chuang, and W. Kuo. "Relationships between Engagement and Learning Style for using VPL on Game Design". In: *Proceedings of the 4th IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning, DIGITEL 2012, Takamatsu, Japan, March 27–30, 2012*. IEEE, 2012, pp. 153–155. ISBN: 978-1-4673-0885-4. DOI: 10.1109/DIGITEL.2012.43 (cit. on p. 90).
- [CCH14] Y. Chaudy, T. M. Connolly, and T. Hainey. "EngAGE: A Link between Educational Games Developers and Educators". In: *Proceedings of the 6th International Conference on Games and Virtual Worlds for Serious Applications, VS-GAMES 2014, Valletta, Malta, September 9–12, 2014*. IEEE, 2014, pp. 1–7. ISBN: 978-1-4799-4056-1. DOI: 10.1109/VS-Games.2014.7012156 (cit. on p. 56).
- [Chu99a] D. Church. "Formal Abstract Design Tools". In: *Game Developer* (Aug. 1999), pp. 44–50. ISSN: 1073-922X (cit. on pp. 38, 47, 161, 185).
- [Chu99b] D. Church. "Formal Abstract Design Tools". In: *Gamasutra* (July 1999), pp. 44–50. URL: http://www.gamasutra.com/view/feature/3357/formal_abstract_design_tools.php (visited on Nov. 15, 2018) (cit. on pp. 38, 47).
- [CDP10] A. Cicchetti, D. Di Ruscio, and A. Pierantonio. "Model Patches in Model-Driven Engineering". In: *Models in Software Engineering – Workshops and Symposia at MODELS 2009 Reports and Revised Selected Papers, Denver, CO, USA, October 4–9, 2009*. Ed. by S. Ghosh. Vol. 6002. LNCS. Springer, 2010, pp. 190–204. ISBN: 978-3-642-12261-3. DOI: 10.1007/978-3-642-12261-3_19 (cit. on p. 231).
- [CCP12] B. Combemale, X. Crégut, and M. Pantel. "A Design Pattern to Build Executable DSMLs and Associated V&V Tools". In: *Proceedings of the 19th Asia-Pacific Software Engineering Conference, APSEC 2012, Hong Kong, China, December 4–7, 2012*. Vol. 1. IEEE, 2012, pp. 282–287. ISBN: 978-1-4673-4930-7. DOI: 10.1109/APSEC.2012.79 (cit. on p. 231).
- [CKM15] K. Compton, B. A. Kybartas, and M. Mateas. "Tracery: An Author-Focused Generative Text Tool". In: *Interactive Storytelling – Proceedings of the 8th International Conference on Interactive Digital Storytelling, ICIDS 2015, Copenhagen, Denmark, November 30–December 4, 2015*. Ed. by H. Schoenau-Fog, L. E. Bruni, S. Louchart, and S. Baceviciute. Vol. 9445. LNCS. Springer, 2015, pp. 154–161. ISBN: 978-3-319-27035-7. DOI: 10.1007/978-3-319-27036-4_14 (cit. on pp. 39, 84, 239).
- [Con97] M. J. Conway. "Alice: Easy-to-Learn 3D Scripting for Novices". PhD thesis. University of Virginia, Dec. 1997 (cit. on p. 88).
- [CAB⁺00] M. Conway, S. Audia, T. Burnette, D. Cosgrove, and K. Christiansen. "Alice: Lessons Learned from Building a 3D System for Novices". In: *Proceedings of the CHI 2000 Conference on Human factors in computing systems, The Hague, The Netherlands, April 1–6, 2000*. Ed. by T. Turner and G. Szwillus. ACM, 2000, pp. 486–493. DOI: 10.1145/332040.332481 (cit. on p. 88).
- [CC11] M. Cook and S. Colton. "Multi-faceted Evolution of Simple Arcade Games". In: *IEEE Conference on Computational Intelligence and Games, CIG 2011, Seoul, South Korea, August 31–September 3, 2011*. IEEE, Aug. 2011, pp. 289–296. ISBN: 978-1-4577-0010-1. DOI: 10.1109/CIG.2011.6032019 (cit. on p. 200).
- [CCG13] M. Cook, S. Colton, and J. Gow. "Nobody's A Critic: On The Evaluation Of Creative Code Generators – A Case Study In Video Game Design". In: *Proceedings of the Fourth International Conference on Computational Creativity, ICC3 2013, Sidney, Australia, June 12–14, 2013*. Ed. by M. L. Maher, T. Veale, R. Saunders, and O. Bown. computationalcreativity.net, 2013, pp. 123–130 (cit. on p. 64).

- [CCG17] M. Cook, S. Colton, and J. Gow. "The ANGELINA Videogame Design System - Part I". In: *IEEE Transactions on Computational Intelligence and AI in Games* 9.2 (June 2017), pp. 192–203. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2016.2520256 (cit. on p. 57).
- [CCR⁺13] M. Cook, S. Colton, A. Raad, and J. Gow. "Mechanic Miner: Reflection-Driven Game Mechanic Discovery and Level Design". In: *Applications of Evolutionary Computation – Proceedings of the 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3–5, 2013*. Ed. by A. I. Esparcia-Alcázar. Vol. 7835. LNCS. Springer, 2013, pp. 284–293. ISBN: 978-3-642-37191-2. DOI: 10.1007/978-3-642-37192-9_29 (cit. on pp. 63, 64, 179).
- [Coy05] R. Coyne. "Wicked problems revisited". In: *Design studies* 26.1 (2005), pp. 5–17 (cit. on p. 20).
- [CD07] M. L. Crane and J. Dingel. "UML vs. Classical vs. Rhapsody Statecharts: Not all Models are Created Equal". In: *Software and System Modeling* 6.4 (2007), pp. 415–435. DOI: 10.1007/s10270-006-0042-8 (cit. on p. 112).
- [COS⁺06] M. Cutumisu, C. Onuczko, D. Szafron, J. Schaeffer, M. McNaughton, T. Roy, J. Siegel, and M. Carbonaro. "Evaluating Pattern Catalogs: The Computer Games Experience". In: *Proceedings of the 28th International Conference on Software Engineering, ICSE 2006, Shanghai, China, May 20–28, 2006*. ACM, 2006, pp. 132–141. ISBN: 1-59593-375-1. DOI: 10.1145/1134285.1134305 (cit. on pp. 35, 78, 79).
- [Cut09] M. Cutumisu. "Using Behaviour Patterns to Generate Scripts for Computer Role-Playing Games". PhD thesis. University of Alberta, 2009. ISBN: 978-0-494-52438-1. DOI: 10.7939/R30W2K (cit. on p. 79).
- [COM⁺07] M. Cutumisu, C. Onuczko, M. McNaughton, T. Roy, J. Schaeffer, A. Schumacher, J. Siegel, D. Szafron, K. Waugh, M. Carbonaro, H. Duff, and S. Gillis. "ScriptEase: A Generative/Adaptive Programming Paradigm for Game Scripting". In: *Science of Computer Programming* 67.1 (June 2007). Special Issue on Aspects of Game Programming, pp. 32–58. ISSN: 0167-6423. DOI: 10.1016/j.scico.2007.01.005 (cit. on pp. 36, 79).
- [Daw02] B. Dawson. "GDC 2002: Game Scripting in Python". In: *Gamasutra* (Aug. 2002). URL: <https://www.gamasutra.com/view/feature/131372/> (visited on Oct. 10, 2018) (cit. on pp. 103, 162).
- [dTvBV17] O. de Troyer, F. van Broeckhoven, and J. Vlieghe. "Creating Story-Based Serious Games Using a Controlled Natural Language Domain Specific Modeling Language". In: *Serious Games and Edutainment Applications: Volume II*. Ed. by M. Ma and A. Oikonomou. Springer, 2017, pp. 567–603. ISBN: 978-3-319-51645-5. DOI: 10.1007/978-3-319-51645-5_25 (cit. on p. 55).
- [DKV06] A. Denault, J. Kienzle, and H. Vangheluwe. "Model-Based Design of Game AI". In: *Proceedings of the 2nd International North American Conference on Intelligent Games and Simulation, GAME-ON-NA 2006, Monterey, USA, September 19–20, 2006*. Ed. by P. McDowell. EUROSIS, 2006, pp. 67–71. ISBN: 90-77381-29-5 (cit. on pp. 41, 112).
- [dGia14] F. di Giacomo. "Design of an Optimized Compiler for Casanova Language". MA thesis. Università Ca' Foscari Venezia – Corso di Laurea magistrale in Informatica, 2014 (cit. on pp. 107, 108).
- [dGAC⁺17a] F. di Giacomo, M. Abbadi, A. Cortesi, P. Spronck, G. Costantini, and G. Maggiore. "High Performance Encapsulation and Networking in Casanova 2". In: *Entertainment Computing* 20 (May 2017), pp. 25–41. ISSN: 1875-9521. DOI: 10.1016/j.entcom.2017.03.001 (cit. on pp. 107, 108).

- [dGAC⁺16] F. di Giacomo, M. Abbadi, A. Cortesi, P. Spronck, and G. Maggiore. "Building Game Scripting DSLs with the Metacasanova Metacompiler". In: *Intelligent Technologies for Interactive Entertainment – Proceedings of the 8th International Conference, Revised Selected Papers, INTETAIN 2016, Utrecht, The Netherlands, June 28–30, 2016*. Ed. by R. Poppe, J.-J. Meyer, R. Veltkamp, and M. Dastani. Vol. 178. LNICTS. Springer, 2016, pp. 231–242. ISBN: 978-3-319-49616-0. DOI: 10.1007/978-3-319-49616-0_22 (cit. on pp. 107, 108).
- [dGAC⁺17b] F. di Giacomo, M. Abbadi, A. Cortesi, P. Spronck, and G. Maggiore. "Metacasanova: An Optimized Meta-Compiler for Domain-Specific Languages". In: *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2017, Vancouver, BC, Canada, October 23–24, 2017*. ACM, 2017, pp. 232–243. ISBN: 978-1-4503-5525-4. DOI: 10.1145/3136014.3136015 (cit. on pp. 35, 107, 108).
- [DN12] C. Dormann and M. Neuvians. "Humor Patterns: Teasing, Fun and Mirth". In: *Proceedings of the 1st Workshop on Design Patterns in Games, DPG 2012, Raleigh, USA, May 29, 2012*. ACM, 2012, pp. 1–4. ISBN: 978-1-4503-1854-9. DOI: 10.1145/2427116.2427118 (cit. on p. 41).
- [Dor09] J. Dormans. "Machinations: Elemental Feedback Patterns for Game Design". In: *Proceedings of the 5th International North American Conference on Intelligent Games and Simulation, GAME-ON-NA 2009, Atlanta, USA, August 26–28, 2009*. Ed. by J. Saur and M. Loper. EUROESIS, 2009, pp. 33–40. ISBN: 978-90-77381-49-6 (cit. on pp. 6, 41, 60, 61, 161, 169, 172).
- [Dor10] J. Dormans. "Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games". In: *Proceedings of the 1st Workshop on Procedural Content Generation in Games, PCG 2010, Monterey, California, USA, June 18, 2010*. ACM, 2010, pp. 1–8. ISBN: 978-1-4503-0023-0. DOI: 10.1145/1814256.1814257 (cit. on pp. 41, 68, 239).
- [Dor11a] J. Dormans. "Integrating Emergence and Progression". In: *Proceedings of the 2011 DiGRA International Conference: Think, Design, Play, DiGRA 2011, Hilversum, The Netherlands, September 14–17, 2011*. Ed. by M. Copier, A. Waern, and H.W. Kennedy. Digital Games Research Association, 2011 (cit. on p. 41).
- [Dor11b] J. Dormans. "Level Design as Model Transformation: A Strategy for Automated Content Generation". In: *Proceedings of the 2nd Workshop on Procedural Content Generation in Games, PCG 2011, Bordeaux, France, June 28, 2011*. ACM, 2011, pp. 1–8. ISBN: 978-1-4503-0872-4. DOI: 10.1145/2000919.2000921 (cit. on pp. 41, 61, 68, 137).
- [Dor11c] J. Dormans. "Simulating Mechanics to Study Emergence in Games". In: *Workshops at the 7th Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2011, as part of the workshop on Artificial Intelligence in the Game Design Process, Stanford University, October 10–14, 2011*. Vol. WS-11-19. AAAI Workshops. AAAI, 2011 (cit. on p. 61).
- [Dor12a] J. Dormans. "Engineering Emergence: Applied Theory for Game Design". PhD thesis. University of Amsterdam, 2012. ISBN: 9789461907523 (cit. on pp. 60, 61, 128, 162, 163).
- [Dor12b] J. Dormans. "Generating Emergent Physics for Action-Adventure Games". In: *Proceedings of the 3rd Workshop on Procedural Content Generation in Games, PCG 2012, Raleigh, NC, USA, May 29–June 01, 2012*. ACM, 2012, pp. 1–7. ISBN: 978-1-4503-1447-3. DOI: 10.1145/2538528.2538535 (cit. on p. 61).
- [DB11] J. Dormans and S. Bakkes. "Generating Missions and Spaces for Adaptable Play Experiences". In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (Sept. 2011), pp. 216–228. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2011.2149523 (cit. on pp. 9, 37, 68, 239).

- [DL13] J. Dormans and S. Leijnen. "Combinatorial and Exploratory Creativity in Procedural Content Generation". In: *Workshop Proceedings of the 8th International Conference on the Foundations of Digital Games, as part of the 4th Workshop on Procedural Content Generation in Games, PCG 2013, Chania, Crete, Greece, May, 14–17, 2013*. Society for the Advancement of the Science of Digital Games, 2013 (cit. on p. 68).
- [DGE⁺16] R. Dörner, S. Göbel, W. Effelsberg, and J. Wiemeyer, eds. *Serious Games*. Springer, 2016. ISBN: 978-3-319-40611-4. DOI: 10.1007/978-3-319-40612-1 (cit. on p. 53).
- [Dow13] C. Dowd. "The Scrabble of Language towards Persuasion: Changing Behaviors in Journalism". In: *Persuasive Technology – Proceedings of the 8th International Conference, PERSUASIVE 2013, Sydney, NSW, Australia, April 3–5, 2013*. Ed. by S. Berkovsky and J. Freyne. Vol. 7822. LNCS. Springer, 2013, pp. 39–50. ISBN: 978-3-642-37157-8 (cit. on p. 45).
- [ELL⁺13] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius. "Towards a Video Game Description Language". In: *Artificial and Computational Intelligence in Games*. Ed. by S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius. Vol. 6. Dagstuhl Follow-Ups. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013, pp. 85–100. ISBN: 978-3-939897-62-0. DOI: 10.4230/DFU.Vol6.12191.85 (cit. on pp. 100, 101).
- [Ecl12] Eclipse Foundation. *EMF Compare Project*. Eclipse Public License v1.0. 2012. URL: <https://www.eclipse.org/emf/compare/> (visited on Nov. 9, 2018) (cit. on pp. 207, 227).
- [EO12] M. P. Eladhari and E. M. I. Ollila. "Design for Research Results: Experimental Prototyping and Play Testing". In: *Simulation & Gaming* 43.3 (2012), pp. 391–412. DOI: 10.1177/1046878111434255 (cit. on p. 20).
- [EM08] M. P. Eladhari and M. Mateas. "Semi-Autonomous Avatars in World of Minds: A Case Study of AI-based Game Design". In: *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology, ACE 2008, Yokohama, Japan, December 3–5, 2008*. ACM, 2008, pp. 201–208. ISBN: 978-1-60558-393-8. DOI: 10.1145/1501750.1501798 (cit. on p. 41).
- [EHvdW⁺09] A. Eliëns, H. C. Huurdeman, M. R. van de Watering, and W. Bhikharie. "XIMPEL Interactive Video - Between Narrative(s) and Game Play". In: *Proceedings of the 10th International Conference on Intelligent Games and Simulation, GAME-ON 2009, Dusseldorf, Germany, November 26–28, 2009*. Ed. by L. Breitlauch. EUROSIS, 2009, pp. 132–136. ISBN: 978-90-77381-53-3 (cit. on p. 41).
- [EA07] C. Elverdam and E. Aarseth. "Game Classification and Game Design: Construction Through Critical Analysis". In: *Games and Culture* 2.1 (Jan. 2007). DOI: 10.1177/1555412006286892 (cit. on pp. 43, 44).
- [EVV⁺13] S. Erdweg, T. Van Der Storm, M. Völter, M. Boersma, R. Bosman, W. R. Cook, A. Gerritsen, A. Hulshout, S. Kelly, A. Loh, et al. "The State of the Art in Language Workbenches – Conclusions from the Language Workbench Challenge". In: *Software Language Engineering – Proceedings of the 6th International Conference, SLE 2013, Indianapolis, IN, USA, October 26–28, 2013*. Ed. by M. Erwig, R. F. Paige, and E. Van Wyk. Vol. 8225. LNCS. Springer, 2013, pp. 197–217. ISBN: 978-3-319-02653-4. DOI: 10.1007/978-3-319-02654-1_11 (cit. on pp. 21, 116, 204, 215, 249).
- [ES14] R. Evans and E. Short. "Versu—A Simulationist Storytelling System". In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.2 (June 2014), pp. 113–130. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2013.2287297 (cit. on pp. 37, 84, 186, 187).

- [EB10] M. Eysholdt and H. Behrens. "Xtext: Implement Your Language Faster Than the Quick and Dirty Way – Tutorial Summary". In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion, SPLASH 2010, Reno/Tahoe, Nevada, USA, October 17–21, 2010*. ACM, 2010, pp. 307–309. ISBN: 978-1-4503-0240-1. DOI: 10.1145/1869542.1869625 (cit. on p. 207).
- [FL12] P. Féher and L. Lengyel. "The Power of Graph Transformation – Implementing a Shadow Casting Algorithm". In: *Proceedings of the 10th Jubilee International Symposium on Intelligent Systems and Informatics, SISY 2012, Subotica, Serbia, October 25, 2012*. IEEE, 2012, pp. 121–127. DOI: 10.1109/SISY.2012.6339500 (cit. on p. 116).
- [FB14] N. Fenton and J. Bieman. *Software Metrics: A Rigorous and Practical Approach*. 3rd ed. CRC press, 2014. ISBN: 9780429106224. DOI: 10.1201/b17461 (cit. on p. 85).
- [Fla11] M. Flatt. "Creating Languages in Racket". In: *Queue* 9.11 (Nov. 2011), pp. 20–34. ISSN: 1542-7730. DOI: 10.1145/2063166.2068896 (cit. on p. 36, 118).
- [Fla12] M. Flatt. "Creating Languages in Racket". In: *Communications of the ACM* 55.1 (Jan. 2012), pp. 48–56. ISSN: 0001-0782. DOI: 10.1145/2063176.2063195 (cit. on p. 36, 118).
- [FMM⁺13a] J. M. Font, T. Mahlmann, D. Manrique, and J. Togelius. "A Card Game Description Language". In: *Applications of Evolutionary Computation – Proceedings of the 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3–5, 2013*. Ed. by A. I. Esparcia-Alcázar. Vol. 7835. LNCS. Springer, 2013, pp. 254–263. ISBN: 978-3-642-37192-9 (cit. on p. 99).
- [FMM⁺13b] J. M. Font, T. Mahlmann, D. Manrique, and J. Togelius. "Towards the Automatic Generation of Card Games through Grammar-Guided Genetic Programming". In: *Proceedings of the 8th International Conference on the Foundations of Digital Games, FDG 2013, Chania, Crete, Greece, May 14–17, 2013*. Society for the Advancement of the Science of Digital Games, 2013, pp. 360–363. ISBN: 978-0-9913982-0-1 (cit. on p. 41, 99).
- [FGH⁺12] Y. Francillette, A. Gouaïch, N. Hocine, and J. Pons. "A Gameplay Loops Formal Language". In: *Proceedings of the 17th International Conference on Computer Games, CGAMES 2012, Louisville, KY, USA, August 30–July 1, 2012*. 2012, pp. 94–101. ISBN: 978-1-4673-1121-2. DOI: 10.1109/CGames.2012.6314558 (cit. on p. 41).
- [FBM⁺10a] F. Frapolli, A. Brocco, A. Malatras, and B. Hirsbrunner. "FLEXIBLE RULES: A Player Oriented Board Game Development Framework". In: *Proceedings of the 3rd International Conference on Advances in Computer-Human Interactions, ACHI 2010, Sint Maarten, Netherlands, Antilles, February 10–16, 2010*. IEEE, 2010, pp. 113–118. ISBN: 978-1-4244-5694-9. DOI: 10.1109/ACHI.2010.14 (cit. on p. 114).
- [FMH10] F. Frapolli, A. Malatras, and B. Hirsbrunner. "Exploiting Traditional Gameplay Characteristics to Enhance Digital Board Games". In: *Proceedings of the 2nd International IEEE Consumer Electronics Society's Games Innovations Conference, GiC 2010, Hong Kong, China, December 21–23, 2010*. IEEE, 2010, pp. 1–8. ISBN: 978-1-4244-7178-2. DOI: 10.1109/ICEGIC.2010.5716897 (cit. on p. 114).
- [FBM⁺10b] F. Frapolli, A. Brocco, A. Malatras, and B. Hirsbrunner. "Decoupling Aspects in Board Game Modeling". In: *International Journal of Gaming and Computer-Mediated Simulations* 2.2 (Apr. 2010), pp. 18–35. ISSN: 1942-3888. DOI: 10.4018/jgcms.2010040102 (cit. on p. 114).
- [FH02] D. Fu and R. T. Houlette. "Putting AI in Entertainment: An AI Authoring Tool for Simulation and Games". In: *IEEE Intelligent Systems* 17.4 (July 2002), pp. 81–84. ISSN: 1941-1294. DOI: 10.1109/MIS.2002.1024756 (cit. on p. 75).

- [FHJ03] D. Fu, R. Houlette, and R. Jensen. "A Visual Environment for Rapid Behavior Definition". In: *Proceedings of the 12th Conference on Behavior Representation in Modeling and Simulation, BRIMS 2003, Scottsdale, Arizona, USA, May 12–15, 2003*. SISO, 2003. ISBN: 978-1-61567-168-7 (cit. on pp. 74, 75, 137).
- [FHL07] D. Fu, R. Houlette, and J. Ludwig. "An AI Modeling Tool for Designers and Developers". In: *IEEE Aerospace Conference Proceedings, Big Sky, MT, USA, March 3–10, 2007*. IEEE, 2007, pp. 1–9. DOI: 10.1109/AERO.2007.352769 (cit. on pp. 74, 75, 162).
- [FSH04] T. Fullerton, C. Swain, and S. Hoffman. *Game Design Workshop: Designing, Prototyping, and Playtesting Games*. CMP Books, 2004. ISBN: 1578202221 (cit. on p. 56).
- [FSH08] T. Fullerton, C. Swain, and S. Hoffman. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. 2nd ed. Morgan Kaufmann, 2008. ISBN: 1578202221 (cit. on pp. 20, 39).
- [FR12] M. Funk and M. Rauterberg. "PULP Scription: A DSL for Mobile HTML5 Game Applications". In: *Entertainment Computing – Proceedings of the 11th International Conference on Entertainment Computing, ICEC 2012, as part of the 2nd Workshop on Game Development and Model-Driven Software Development, GD&MDS 2012, Bremen, Germany, September 26–29, 2012*. Ed. by M. Herrlich, R. Malaka, and M. Masuch. Vol. 7522. LNCS. Springer, 2012, pp. 504–510. ISBN: 978-3-642-33542-6. DOI: 10.1007/978-3-642-33542-6_65 (cit. on p. 41).
- [FSR⁺11] A. W. B. Furtado, A. L. M. Santos, G. L. Ramalho, and E. S. de Almeida. "Improving Digital Game Development with Software Product Lines". In: *IEEE Software* 28.5 (Sept. 2011), pp. 30–37. ISSN: 0740-7459. DOI: 10.1109/MS.2011.101 (cit. on p. 113).
- [FS06a] A. W. B. Furtado and A. L. M. Santos. "Tutorial: Applying Domain-Specific Modeling to Game Development with the Microsoft DSL Tools". In: *Proceedings of the 5th Brazilian Symposium on Computer Games and Digital Entertainment, SBGames 2006, Recife, Brazil, November 8–10, 2006*. Ed. by B. Feijó, A. Neves, E. Clua, L. Freire, G. Ramalho, and M. Walter. UFPE, 2006. ISBN: 85-7669-098-5 (cit. on pp. 35, 41, 113).
- [FS06b] A. W. B. Furtado and A. L. M. Santos. "Using Domain-Specific Modeling Towards Computer Games Development Industrialization". In: *Proceedings of the 6th Workshop on Domain-Specific Modeling, DSM 2006, Portland, Oregon, USA, October 22, 2006*. Computer Science and Information System Reports, Technical Reports, TR-37. University of Jyväskylä, Finland, 2006, pp. 1–14. ISBN: 951-39-2631-1 (cit. on p. 113).
- [FSR11] A. W. B. Furtado, A. L. M. Santos, and G. L. Ramalho. "SharpLudus Revisited: From Ad Hoc and Monolithic Digital Game DSLs to Effectively Customized DSM Approaches". In: *SPLASH '11 Workshops: Proceedings of the Compilation of the Co-located Workshops on DSM'11, TMC'11, AGERE! 2011, AOOPEs'11, NEAT'11, & VMIL'11 – as part of the 11th workshop on Domain-Specific Modeling, DSM 2011, Portland, Oregon, USA, October 23–24, 2011*. ACM, 2011, pp. 57–62. ISBN: 978-1-4503-1183-0. DOI: 10.1145/2095050.2095061 (cit. on pp. 35, 113).
- [Furo6] A. W. B. Furtado. "SharpLudus: Improving Game Development Experience Through Software Factories and Domain-Specific Languages". MA thesis. Universidade Federal de Pernambuco (UFPE) – Mestrado em Ciência da Computação – Centro de Informática (CIN), 2006 (cit. on p. 113).
- [Fur12] A. W. B. Furtado. "Domain-Specific Game Development". PhD thesis. Universidade Federal de Pernambuco, 2012 (cit. on pp. 113, 128).
- [FSR07] A. W. B. Furtado, A. L. M. Santos, and G. L. Ramalho. "A Computer Games Software Factory and Edutainment Platform for Microsoft.NET". In: *IET Software* 1.6 (Dec. 2007), pp. 280–293. ISSN: 1751-8806. DOI: 10.1049/iet-sen:20070023 (cit. on p. 113).

- [Gam10] I. A. Games. "Gamestar Mechanic: Learning a Designer Mindset through Communicational Competence with the Language of Games". In: *Learning, Media and Technology* 35.1 (Mar. 2010), pp. 31–52. ISSN: 1743-9884. DOI: 10.1080/17439880903567774 (cit. on p. 89).
- [GH]⁺94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. ISBN: 9780201633610 (cit. on pp. 46, 218).
- [Gau16] S. Gaudl. "Building Robust Real-Time Game AI: Simplifying & Automating Integral Process Steps in Multi-Platform Design". PhD thesis. University of Bath, May 2016 (cit. on pp. 75, 128).
- [GDB13] S. E. Gaudl, S. Davies, and J. J. Bryson. "Behaviour Oriented Design for Real-Time-Strategy Games". In: *Proceedings of the 8th International Conference on the Foundations of Digital Games, FDG 2013, Chania, Crete, Greece, May 14–17, 2013*. Ed. by G. N. Yannakakis, E. Aarseth, K. Jørgensen, and J. C. Lester. Society for the Advancement of the Science of Digital Games, 2013, pp. 198–205 (cit. on p. 75).
- [GBU08] T. Goldschmidt, S. Becker, and A. Uhl. "Classification of Concrete Textual Syntax Mapping Approaches". In: *Model Driven Architecture – Foundations and Applications – Proceedings of the 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9–13, 2008*. Ed. by I. Schieferdecker and A. Hartman. Vol. 5095. LNCS. Springer, 2008, pp. 169–184. ISBN: 978-3-540-69095-5. DOI: 10.1007/978-3-540-69100-6_12 (cit. on p. 207).
- [GBF15] S. Götz, N. Bencomo, and R. France. "Devising the Future of the Models@Run.Time Workshop". In: *SIGSOFT Software Engineering Notes* 40.1 (Feb. 2015), pp. 26–29. ISSN: 0163-5948. DOI: 10.1145/2693208.2693249 (cit. on p. 231).
- [Gro03] A. Grossman, ed. *Postmortems from GameDeveloper*. CMP Books, 2003. ISBN: 1-57820-214-0 (cit. on p. 38).
- [GGG⁺14] A. Grow, S. E. Gaudl, P. Gomes, M. Mateas, and N. Wardrip-Fruin. "A Methodology for Requirements Analysis of AI Architecture Authoring Tools". In: *Proceedings of the 9th International Conference on the Foundations of Digital Games, FDG 2014, Liberty of the Seas, Caribbean, April 3–7, 2014*. Ed. by M. Mateas, T. Barnes, and I. Bogost. Society for the Advancement of the Science of Digital Games, 2014 (cit. on p. 75).
- [Grü05] S. Grünvogel. "Formal Models and Game Design". In: *Game Studies* 5.1 (Oct. 2005) (cit. on p. 39).
- [Gua17] V. Guana. "End-to-end Fine-grained Traceability Analysis in Model Transformations and Transformation Chains". PhD thesis. Department of Computing Science – University of Alberta, 2017 (cit. on pp. 115, 128).
- [GS14] V. Guana and E. Stroulia. "PhyDSL: A Code-generation Environment for 2D Physics-based Games". In: *IEEE Games, Entertainment, and Media Conference, GEM 2014, Toronto ON Canada, October 22–24, 2014*. IEEE, 2014, pp. 1–6 (cit. on p. 115).
- [GSN15] V. Guana, E. Stroulia, and V. Nguyen. "Building a Game Engine: A Tale of Modern Model-Driven Engineering". In: *Proceedings of the 4th International Workshop on Games and Software Engineering, GAS 2015, Florence, Italy, May 18, 2015*. IEEE, 2015, pp. 15–21. ISBN: 978-1-4673-7046-2. DOI: 10.1109/GAS.2015.11 (cit. on pp. 35, 115).
- [Guo15] H. Guo. "Concepts and Modelling Techniques for Pervasive and Social Games". PhD thesis. Norwegian University of Science et al., June 2015. ISBN: 978-82-326-0944-4 (cit. on pp. 45, 128).

- [GTW⁺15a] H. Guo, H. Trætteberg, A. I. Wang, and S. Gao. "A Workflow for Model Driven Game Development". In: *Proceedings of the IEEE 19th International Enterprise Distributed Object Computing Conference, EDOC 2015, Adelaide, SA, Australia, September 21–25, 2015*. IEEE, 2015, pp. 94–103. ISBN: 978-1-4673-9203-7. DOI: 10.1109/EDOC.2015.23 (cit. on pp. 44, 45).
- [GTW⁺14] H. Guo, H. Trætteberg, A. I. Wang, and S. Gao. "PerGO: An Ontology Towards Model Driven Pervasive Game Development". In: *On the Move to Meaningful Internet Systems: OTM 2014 Workshops, as part of Ontologies, DataBases, and Applications of Semantics, ODBASE 2014 Posters, Amantea, Italy, October 27–31, 2014*. Ed. by R. Meersman, H. Panetto, A. Mishra, R. Valencia-García, A. L. Soares, I. Ciuciu, F. Ferri, G. Weichhart, T. Moser, M. Bezzi, and H. Chan. Vol. 8842. LNCS. Springer, 2014, pp. 651–654. ISBN: 978-3-662-45550-0. DOI: 10.1007/978-3-662-45550-0_67 (cit. on pp. 44, 45).
- [GTW⁺15b] H. Guo, H. Trætteberg, A. I. Wang, S. Gao, and M. L. Jaccheri. "RealCoins: A Case Study of Enhanced Model Driven Development for Pervasive Games". In: *International Journal of Multimedia and Ubiquitous Engineering* 10.5 (2015), pp. 395–410. ISSN: 1975-0080. DOI: 10.14257/ijmue.2015.10.5.37 (cit. on pp. 44, 45).
- [HS07] J. S. Harbour and J. R. Smith. *DarkBASIC Pro Game Programming*. 2nd ed. Thomson Course Technology, 2007. ISBN: 1-59863-287-6 (cit. on p. 102).
- [HZD⁺11] K. Hartsook, A. Zook, S. Das, and M. O. Riedl. "Toward Supporting Stories with Procedurally Generated Game Worlds". In: *IEEE Conference on Computational Intelligence and Games, CIG 2011, Seoul, South Korea, August 31–September 3, 2011*. IEEE, 2011, pp. 297–304. ISBN: 978-1-4577-0011-8. DOI: 10.1109/CIG.2011.6032020 (cit. on pp. 186, 187).
- [Has13] T. Hastjarjanto. "Strategies for Real-Time Video Games". MA thesis. Utrecht University, Mar. 2013 (cit. on p. 75).
- [HJL13] T. Hastjarjanto, J. Jeuring, and S. Leather. "A DSL for Describing the Artificial Intelligence in Real-time Video Games". In: *Proceedings of the 3rd International Workshop on Games and Software Engineering: Engineering Computer Games to Enable Positive, Progressive Change, GAS 2013, San Francisco, CA, USA, May 18–26, 2013*. IEEE, 2013, pp. 8–14. ISBN: 978-1-4673-6263-4. DOI: 10.1109/GAS.2013.6632583 (cit. on pp. 35, 75).
- [Hei18] Q. Heijn. "Improving the Quality of Grammars for Procedural Level Generation: A Software Evolution Perspective". MA thesis. University of Amsterdam, Aug. 2018 (cit. on p. 11).
- [HKV07] I. Heitlager, T. Kuipers, and J. Visser. "A Practical Model for Measuring Maintainability". In: *Proceedings of the 6th International Conference on the Quality of Information and Communications Technology, QUATIC 2007, Lisbon, Portugal, September 12–14, 2007*. Ed. by R. J. Machado, F. B. e Abreu, and P. R. da Cunha. IEEE, 2007, pp. 30–39. ISBN: 0-7695-2948-8. DOI: 10.1109/QUATIC.2007.8 (cit. on p. 245).
- [HMV⁺13] M. Hendriks, S. Meijer, J. Van Der Velden, and A. Iosup. "Procedural Content Generation for Games: A Survey". In: *ACM Transactions on Multimedia Computing, Communications and Applications* 9.1 (Feb. 2013), pp. 1–22. ISSN: 1551-6857. DOI: 10.1145/2422956.2422957 (cit. on p. 123).
- [HO10] F. E. Hernandez and F. R. Ortega. "Eberos GML2D: A Graphical Domain-Specific Language for Modeling 2D Video Games". In: *Proceedings of the 10th Workshop on Domain-Specific Modeling, DSM 2010, Reno/Tahoe, Nevada, USA, October 17–18, 2010*. Ed. by M. Rossi, J.-P. Tolvanen, J. Sprinkle, and S. Kelly. Aalto-Print, 2010, pp. 1–6. ISBN: 978-952-60-1043-4. DOI: 10.1145/2060329.2060342 (cit. on pp. 35, 113, 114).

- [HJM⁺13] P. Herzig, K. Jugel, C. Momm, M. Ameling, and A. Schill. "GaML: A Modeling Language for Gamification". In: *Proceedings of the IEEE/ACM 6th International Conference on Utility and Cloud Computing, Dresden, Germany, December 9–12, 2013*. IEEE, 2013, pp. 494–499. ISBN: 978-0-7695-5152-4. DOI: 10.1109/UCC.2013.96 (cit. on p. 92).
- [HRS⁺04] A. R. Hevner, S. Ram, T. M. Salvatore, and J. Park. "Design Science in Information Systems Research". In: *MIS Quarterly* 28.1 (Mar. 2004). Ed. by A. S. Lee, pp. 75–105 (cit. on pp. 6, 8, 9).
- [Hol16] L. T. Holloway. "Modeling and Formal Verification of Gaming Storylines". PhD thesis. The University of Texas at Austin, May 2016 (cit. on p. 128).
- [Hol14] I. Holmes. "A Web-Based Editor for Multiplayer Choice Games". In: *Proceedings of the 9th International Conference on the Foundations of Digital Games, FDG 2014, Liberty of the Seas, Caribbean, April 3–7, 2014*. Ed. by M. Mateas, T. Barnes, and I. Bogost. Society for the Advancement of the Science of Digital Games, 2014, pp. 1–4. ISBN: 978-0-9913982-2-5 (cit. on p. 40).
- [HBK07] J. Holopainen, S. Björk, and J. Kuittinen. "Teaching Gameplay Design Patterns". In: *Organizing and Learning through Gaming and Simulation – Proceedings of the 38th Conference of the International Simulation And Gaming Association, ISAGA 2007, Nijmegen, The Netherlands, July 9–13, 2007*. Ed. by I. Mayer and H. Mastik. Eburon, 2007. ISBN: 9789059722316 (cit. on pp. 39, 47).
- [HB03] J. Holopainen and S. Björk. "Game Design Patterns – Lecture Notes". In: *Game Developers Conference, GDC 2003*. 2003. URL: http://www.gents.it/FILES/ebooks/Game_Design_Patterns.pdf (visited on Oct. 24, 2018) (cit. on pp. 38, 47).
- [HB08] J. Holopainen and S. Björk. "Gameplay Design Patterns for Motivation". In: *Games: Virtual Worlds and Reality – Proceedings of the 39th Conference of the International Simulation And Gaming Association, ISAGA 2008, Kaunas, Lithuania, July 7–11, 2008*. Ed. by E. Bagdonas, I. Patasiene, and D. Jovarauskiene. Kaunas University of Technology, 2008. ISBN: 978-9955-25-528-4 (cit. on pp. 39, 47).
- [Holo3] G. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. 1st ed. Addison-Wesley, 2003. ISBN: 0-321-22862-6. URL: http://spinroot.com/spin/Doc/Book_extras/ (visited on Nov. 8, 2018) (cit. on p. 136).
- [Hor14] I. D. Horswill. "Architectural Issues for Compositional Dialog in Games". In: *Papers from the 2014 AIIDE Workshop – Proceedings of the Workshop on Games and Natural Language Processing, GAMNLP 2014, Raleigh, NC, USA, October 3–7, 2014*. AAAI Technical Report WS-14-17. AAAI, 2014. ISBN: 978-1-57735-686-8 (cit. on p. 39).
- [Hor99] E. Horvitz. "Principles of Mixed-Initiative User Interfaces". In: *Proceeding of the CHI '99 Conference on Human Factors in Computing Systems: The CHI is the Limit, Pittsburgh, PA, USA, May 15–20, 1999*. Ed. by M. G. Williams and M. W. Altom. ACM, 1999, pp. 159–166. DOI: 10.1145/302979.303030 (cit. on p. 66).
- [Hui38] J. Huizinga. *Homo Ludens: Proeve Ener Bepaling van het Spelelement der Cultuur*. Wolters-Noordhoff, 1938 (cit. on p. 39).
- [HW10] K. Hullett and J. Whitehead. "Design Patterns in FPS Levels". In: *Proceedings of the Fifth International Conference on the Foundations of Digital Games, FDG 2010, Monterey, California, USA, June 19–21, 2010*. ACM, 2010, pp. 78–85. ISBN: 978-1-60558-937-4. DOI: 10.1145/1822348.1822359 (cit. on pp. 41, 51).
- [HLZ04] R. Hunicke, M. Leblanc, and R. Zubek. "MDA: A Formal Approach to Game Design and Game Research". In: *Proceedings of the AAAI workshop on Challenges in Game Artificial Intelligence*. AAAI, 2004, pp. 1–5 (cit. on pp. 20, 48, 185).

- [IDC05] R. Ierusalimsky, L. H. De Figueiredo, and W. Celes Filho. "The Implementation of Lua 5.0". In: *Journal of Universal Computer Science* 11.7 (2005), pp. 1159–1176. DOI: 10.3217/jucs-011-07 (cit. on p. 103).
- [IdFC07] R. Ierusalimsky, L. H. de Figueiredo, and W. Celes. "The Evolution of Lua". In: *Proceedings of the 3rd ACM SIGPLAN Conference on History of Programming Languages, HOPL III, San Diego, California, June 9–10, 2007*. ACM, 2007, pp. 1–26. ISBN: 978-1-59593-766-7. DOI: 10.1145/1238844.1238846 (cit. on p. 103).
- [IvdSE14] P. Inostroza, T. van der Storm, and S. Erdweg. "Tracing Program Transformations with String Origins". In: *Theory and Practice of Model Transformations – Proceedings of the 7th International Conference, ICMT 2014, York, UK, July 21–22, 2014*. Ed. by D. Di Ruscio and D. Varró. Vol. 8568. LNCS. Springer, 2014, pp. 154–169. ISBN: 978-3-319-08788-7. DOI: 10.1007/978-3-319-08789-4_12 (cit. on p. 207).
- [IRWo8] A. Ioannidou, A. Repenning, and D. Webb. "Using Scalable Game Design to Promote 3D Fluency: Assessing the AgentCubes incremental 3D End-user Development Framework". In: *Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2008, Herrsching am Ammersee, Germany, September 15–19, 2008*. Ed. by P. Bottoni, M. B. Rosson, and M. Minas. IEEE, 2008, pp. 47–54. ISBN: 978-1-4244-2528-0. DOI: 10.1109/VLHCC.2008.4639057 (cit. on p. 91).
- [Isl05a] D. Isla. "GDC 2005 Proceeding: Handling Complexity in the Halo 2 AI". In: *Gamasutra* (Mar. 2005). URL: <https://www.gamasutra.com/view/feature/130663> (cit. on pp. 73, 74).
- [Isl05b] D. Isla. "Managing Complexity in the Halo 2 AI System". In: *Proceedings of the Game Developers Conference, GDC 2005*. Audio recording. Gdcvault.com, 2005. URL: <https://gdcvault.com/play/1020270/Managing-Complexity-in-the-Halo> (visited on Oct. 25, 2018) (cit. on pp. 38, 73, 74).
- [Jon05] R. Jones. "Rapid Game Development in Python". In: *OpenSource Developers' Conference*. 2005, pp. 84–90 (cit. on p. 103).
- [Juu11] J. Juul. *Half-real: Video Games between Real Rules and Fictional Worlds*. MIT press, 2011 (cit. on pp. 19, 56).
- [KBB15] D. Karavolos, A. Bouwer, and R. Bidarra. "Mixed-Initiative Design of Game Levels: Integrating Mission and Space into Level Generation". In: *Proceedings of the 10th International Conference on the Foundations of Digital Games, FDG 2015, Pacific Grove, CA, USA, June 22–25, 2015*. Ed. by J. P. Zagal, E. MacCallum-Stewart, and J. Togelius. Society for the Advancement of the Science of Digital Games, 2015 (cit. on pp. 68, 239).
- [KKP⁺12] T. Kehler, U. Kelter, P. Pietsch, and M. Schmidt. "Adaptability of Model Comparison Tools". In: *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012, Essen, Germany, September 3–7, 2012*. ACM, 2012, pp. 306–309. ISBN: 978-1-4503-1204-2. DOI: 10.1145/2351676.2351731 (cit. on p. 227).
- [KKT11] T. Kehler, U. Kelter, and G. Taentzer. "A Rule-Based Approach to the Semantic Lifting of Model Differences in the Context of Model Versioning". In: *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, Lawrence, KS, USA, November 6–10, 2011*. IEEE, 2011, pp. 163–172. ISBN: 978-1-4577-1638-6. DOI: 10.1109/ASE.2011.6100050 (cit. on p. 230).
- [KTB12] J. Kessing, T. Tutenel, and R. Bidarra. "Designing Semantic Game Worlds". In: *Proceedings of the 3rd workshop on Procedural Content Generation in Games, PCG 2012, Raleigh, NC, USA, May 29–June 1, 2012*. ACM, 2012, pp. 1–9. DOI: 10.1145/2538528.2538530 (cit. on p. 67).

- [KT17] A. Khalifa and J. Togelius. “Marahel: A Language for Constructive Level Generation”. In: *Proceedings of the 4th Workshop on Experimental AI in Games, EXAG 2017, co-located with the 13th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2017, Snowbird, Utah, USA, October 5–9, 2017*. AAAI, 2017 (cit. on p. 37).
- [KDV07] J. Kienzle, A. Denault, and H. Vangheluwe. “Model-Based Design of Computer-Controlled Game Character Behavior”. In: *Model Driven Engineering Languages and Systems – Proceedings of the 10th International Conference, MoDELS 2007, Nashville, USA, September 30–October 5, 2007*. Ed. by G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil. Vol. 4735. LNCS. Springer, 2007, pp. 650–665. ISBN: 978-3-540-75209-7. DOI: 10.1007/978-3-540-75209-7_44 (cit. on pp. 36, 112).
- [KC07] B. Kitchenham and S. Charters. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Tech. rep. EBSE-2007-01. Keele University and Durham University Joint Report, 2007 (cit. on pp. 5, 18, 23, 127).
- [KRvR12] P. Klint, L. Roosendaal, and R. van Rozen. “Game Developers Need Lua AiR: Static Analysis of Lua Using Interface Models”. In: *Entertainment Computing – Proceedings of the 11th International Conference on Entertainment Computing, ICEC 2012, as part of the 2nd Workshop on Game Development and Model-Driven Software Development, GD&MDS 2012, Bremen, Germany, September 26–29, 2012*. Ed. by M. Herrlich, R. Malaka, and M. Masuch. Vol. 7522. LNCS. Springer, 2012, pp. 530–535. ISBN: 978-3-642-33542-6. DOI: 10.1007/978-3-642-33542-6_69 (cit. on pp. 12, 41, 103).
- [KvdSV09] P. Klint, T. van der Storm, and J. J. Vinju. “Rascal: A Domain Specific Language for Source Code Analysis and Manipulation”. In: *Ninth IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2009, Edmonton, Alberta, Canada, September 20–21, 2009*. IEEE, 2009, pp. 168–177. ISBN: 978-0-7695-3793-1. DOI: 10.1109/SCAM.2009.28 (cit. on pp. 198, 215, 232, 249).
- [KvdSV11] P. Klint, T. van der Storm, and J. J. Vinju. “EASY Meta-programming with Rascal”. In: *Generative and Transformational Techniques in Software Engineering III – International Summer School, GTTSE 2009, Braga, Portugal, July 6–11, 2009. Revised Papers*. Ed. by J. M. Fernandes, R. Lämmel, J. Visser, and J. Saraiva. Vol. 6491. LNCS. Springer, 2011, pp. 222–289. DOI: 10.1007/978-3-642-18023-1_6 (cit. on p. 136).
- [KvR13] P. Klint and R. van Rozen. “Micro-Machinations: a DSL for Game Economies”. In: *Software Language Engineering – Proceedings of the 6th International Conference on Software Language Engineering, SLE 2013, Indianapolis, IN, USA, October 26–28, 2013*. Ed. by M. Erwig, R. F. Paige, and E. Van Wyk. Vol. 8225. LNCS. Springer, 2013, pp. 36–55. ISBN: 978-3-319-02654-1. DOI: 10.1007/978-3-319-02654-1_3 (cit. on pp. 10, 20, 35, 63, 124, 135, 162, 164, 184, 185, 187, 199).
- [KB13] M. Klotzbuecher and H. Bruyninckx. “A Lightweight, Composable Metamodelling Language for Specification and Validation of Internal Domain Specific Languages”. In: *Model-Based Methodologies for Pervasive and Embedded Software*. Ed. by R. J. Machado, R. S. P. Maciel, J. Rubin, and G. Botterweck. Vol. 7706. LNCS. Springer, 2013, pp. 58–68. ISBN: 978-3-642-38208-6. DOI: 10.1007/978-3-642-38209-3_4 (cit. on p. 163).
- [KDP⁺09] D. S. Kolovos, D. Di Ruscio, A. Pierantonio, and R. F. Paige. “Different Models for Model Matching: An Analysis of Approaches to Support Model Differencing”. In: *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models, CVSM 2009, Vancouver, BC, USA, May 17, 2009*. IEEE, 2009, pp. 1–6. ISBN: 978-1-4244-3714-6. DOI: 10.1109/CVSM.2009.5071714 (cit. on pp. 228, 230).

- [KPP08] D. S. Kolovos, R. F. Paige, and F. A. C. Polack. "The Epsilon Transformation Language". In: *Theory and Practice of Model Transformations: Proceedings of the 1st International Conference on Model Transformations, ICMT 2008, Zürich, Switzerland, July 1–2, 2008*. Ed. by A. Vallecillo, J. Gray, and A. Pierantonio. Vol. 5063. LNCS. Springer, 2008, pp. 46–60. ISBN: 978-3-540-69927-9. DOI: 10.1007/978-3-540-69927-9_4 (cit. on p. 230).
- [Koso5a] R. Koster. "A Grammar of Gameplay". In: *Game Developers Conference, GDC 2005*. Presentation slides. 2005. URL: <https://www.raphkoster.com/gaming/atof/grammarofgameplay.pdf> (visited on Oct. 24, 2018) (cit. on pp. 38, 49, 50, 161).
- [Koso5b] R. Koster. *Theory of Fun for Game Design*. 1st ed. Paraglyph Press, 2005. ISBN: 1932111972 (cit. on p. 50).
- [Kos16] R. Koster. "The Limits of Formalism". In: *Presentation delivered at the BIRS Workshop on Computational Modeling in Games*. Raph Koster's Website, 2016. URL: <https://www.raphkoster.com/games/presentations/the-limits-of-formalism/> (cit. on pp. 49, 50).
- [Kreo2] B. Kreimeier. "The Case for Game Design Patterns". In: *Gamasutra* (Mar. 2002). URL: <https://www.gamasutra.com/view/feature/132649/> (visited on Oct. 15, 2018) (cit. on pp. 38, 47, 186).
- [KBW14] S. Kriglstein, R. Brown, and G. Wallner. "Workflow Patterns as a Means to Model Task Succession in Games: A Preliminary Case Study". In: *Entertainment Computing – Proceedings of the 13th International Conference on Entertainment Computing, ICEC 2014, Sydney, Australia, October 1–3, 2014*. Ed. by Y. Pisan, N.M. Sgouros, and T. Marsh. Vol. 8770. LNCS. Springer, 2014, pp. 36–41. ISBN: 978-3-662-45212-7 (cit. on p. 41).
- [KB17] B. Kybartas and R. Bidarra. "A Survey on Story Generation Techniques for Authoring Computational Narratives". In: *IEEE Transactions on Computational Intelligence and AI in Games* 9.3 (Sept. 2017), pp. 239–253. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2016.2546063 (cit. on pp. 76, 123).
- [Lämo4] R. Lämmel. "Coupled Software Transformations". In: *Proceedings of the 1st International Workshop on Software Evolution Transformations, SET 2004, Delft, the Netherlands, November 9, 2004*. Ed. by Y. Zou and J.R. Cordy. Queen's University, 2004, pp. 31–35. URL: <http://post.queensu.ca/~zouy/files/set-2004.pdf> (visited on Nov. 9, 2018) (cit. on p. 228).
- [Läm18] R. Lämmel. *Software Languages: Syntax, Semantics, and Metaprogramming*. Springer, 2018. ISBN: 978-3-319-90798-7. DOI: 10.1007/978-3-319-90800-7. URL: <http://www.softlang.org/book> (cit. on pp. 34, 247).
- [LMK14] P. Langer, T. Mayerhofer, and G. Kappel. "Semantic Model Differencing Utilizing Behavioral Semantics Specifications". In: *Model-Driven Engineering Languages and Systems: Proceedings of the 17th International Conference, MODELS 2014, Valencia, Spain, September 28 – October 3, 2014*. Ed. by J. Dingel, W. Schulte, I. Ramos, S. Abrahão, and E. Insfran. Vol. 8767. LNCS. Springer, 2014, pp. 116–132. ISBN: 978-3-319-11653-2. DOI: 10.1007/978-3-319-11653-2_8 (cit. on p. 231).
- [LWB⁺13] P. Langer, M. Wimmer, P. Brosch, M. Herrmannsdörfer, M. Seidl, K. Wieland, and G. Kappel. "A Posteriori Operation Detection in Evolving Software Models". In: *Journal of Systems and Software* 86.2 (2013), pp. 551–566. ISSN: 0164-1212. DOI: 10.1016/j.jss.2012.09.037 (cit. on p. 230).
- [LBT⁺11] G. Lehmann, M. Blumendorf, F. Trollmann, and S. Albayrak. "Meta-Modeling Runtime Models". In: *Models in Software Engineering – Workshops and Symposia at MODELS 2010 Reports and Revised Selected Papers, as part of the Workshop on Models@run.time 2010, Oslo, Norway, October 2–8, 2010*. Vol. 6627. LNCS. Springer, 2011, pp. 209–223. ISBN: 978-3-642-21210-9. DOI: 10.1007/978-3-642-21210-9_21 (cit. on p. 232).

- [LBB15] S. Leijnen, P. Brinkkemper, and A. Bouwer. "Generating Game Mechanics in a Model Economy: a MoneyMaker Deluxe Case Study". In: *Proceedings of the 6th Workshop on Procedural Content Generation, PCG 2015, Pacific Grove, CA, USA, June 22–25, 2015*. Society for the Advancement of the Science of Digital Games, 2015 (cit. on p. 41).
- [Lem07] P. Lemay. "Developing a Pattern Language for Flow Experiences in Video Games". In: *Proceedings of the 2007 DiGRA International Conference: Situated Play, DiGRA 2007, Tokyo, Japan, September 24–28, 2007*. The University of Tokyo, 2007 (cit. on pp. 41, 52).
- [LYN⁺19] A. Liapis, G. N. Yannakakis, M. J. Nelson, M. Preuss, and R. Bidarra. "Orchestrating Game Generation". In: *IEEE Transactions on Games* 11.1 (2019), pp. 48–68. DOI: 10.1109/TG.2018.2870876 (cit. on p. 123).
- [LYT13] A. Liapis, G. N. Yannakakis, and J. Togelius. "Sentient Sketchbook: Computer-Aided Game Level Authoring". In: *Proceedings of the 8th International Conference on the Foundations of Digital Games, FDG 2013, Chania, Crete, Greece, May 14–17, 2013*. Ed. by G. N. Yannakakis, E. Aarseth, K. Jørgensen, and J. C. Lester. Society for the Advancement of the Science of Digital Games, 2013, pp. 213–220 (cit. on p. 68).
- [LF95] H. Lieberman and C. Fry. "Bridging the Gulf between Code and Behavior in Programming". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 1995, Denver, Colorado, USA, May 7–11, 1995*. ACM Press/Addison-Wesley Publishing Co., 1995, pp. 480–486. ISBN: 0-201-84705-1. DOI: 10.1145/223904.223969 (cit. on p. 203).
- [LH14] C. U. Lim and D. F. Harrell. "An Approach to General Videogame Evaluation and Automatic Generation using a Description Language". In: *Proceedings of the 2014 IEEE Conference on Computational Intelligence and Games, CIG 2014, Dortmund, Germany, August 26–29, 2014*. IEEE, 2014, pp. 1–8. ISBN: 978-1-4799-3547-5. DOI: 10.1109/CIG.2014.6932896 (cit. on pp. 37, 109, 110).
- [LBC10] C. Lim, R. Baumgarten, and S. Colton. "Evolving Behaviour Trees for the Commercial Game DEFCON". In: *Applications of Evolutionary Computation, EvoApplications 2010: EvoCOMPLEX, EvoGAMES, EvoASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Proceedings, Part I – as part of EvoGAMES, Istanbul, Turkey, April 7–9, 2010*. Ed. by C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. Esparcia-Alcázar, C. K. Goh, J. J. Merelo Guervós, F. Neri, M. Preuss, J. Togelius, and G. N. Yannakakis. Vol. 6024. LNCS. Springer, 2010, pp. 100–110. DOI: 10.1007/978-3-642-12239-2_11 (cit. on p. 74).
- [LGJ07] Y. Lin, J. Gray, and F. Jouault. "DSMDiff: A Differentiation Tool for Domain-Specific Models". In: *European Journal of Information Systems* 16.4 (Aug. 2007), pp. 349–361. ISSN: 1476-9344. DOI: 10.1057/palgrave.ejis.3000685 (cit. on pp. 227, 231).
- [Lin68] A. Lindenmayer. "Mathematical Models for Cellular Interactions in Development". In: *Journal of Theoretical Biology* 18.3 (1968), pp. 280–299. ISSN: 0022-5193. DOI: 10.1016/0022-5193(68)90079-9 (cit. on p. 68).
- [Llo10] N. Llopis. "Data Oriented Design". In: *Game Developer* 17.8 (Sept. 2010), pp. 31–33. ISSN: 1073-922X (cit. on p. 38).
- [LB11] R. Lopes and R. Bidarra. "Adaptivity Challenges in Games and Simulations: a Survey". In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.2 (May 2011), pp. 85–99. ISSN: 1943-0698. DOI: 10.1109/TCIAIG.2011.2152841 (cit. on p. 200).
- [LHH⁺08] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth. "General Game Playing: Game Description Language Specification". In: (Mar. 2008). Technical Report LG-2006-01 (cit. on p. 96).

- [Mac11a] M. B. MacLaurin. "The Design of Kodu: A Tiny Visual Programming Language for Children on the Xbox 360". In: *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, Texas, USA, January 26–28, 2011*. Invited talk – Video recording accessible via ACM. ACM, 2011, pp. 241–246. ISBN: 978-1-4503-0490-0. DOI: 10.1145/1926385.1926413. URL: <http://delivery.acm.org/10.1145/1930000/1926413/22-MPEG-4.mp4> (visited on Oct. 15, 2018) (cit. on pp. 36, 89).
- [Mac11b] M. B. MacLaurin. "The Design of Kodu: A Tiny Visual Programming Language for Children on the Xbox 360". In: *SIGPLAN Notices* 46.1 (Jan. 2011), pp. 241–246. ISSN: 0362-1340. DOI: 10.1145/1925844.1926413 (cit. on pp. 36, 89).
- [MBO11] G. Maggiore, M. Bugliesi, and R. Orsini. "Monadic Scripting in F# for Computer Games". In: *Proceedings of the 5th International Workshop on Harnessing Theories for Tool Support in Software, TTSS 2011, Oslo Norway, September 13, 2011*. UIO, 2011 (cit. on pp. 107, 108).
- [MSO⁺12a] G. Maggiore, A. Spanò, R. Orsini, M. Bugliesi, M. Abbadi, and E. Steffinlongo. "A Formal Specification for Casanova, a Language for Computer Games". In: *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2012, Copenhagen, Denmark, June 25–26, 2012*. ACM, 2012, pp. 287–292. ISBN: 978-1-4503-1168-7. DOI: 10.1145/2305484.2305533 (cit. on pp. 41, 107, 108).
- [MSO⁺12b] G. Maggiore, A. Spanò, R. Orsini, G. Costantini, M. Bugliesi, and M. Abbadi. "Designing Casanova: A Language for Games". In: *Advances in Computer Games – Proceedings of the 13th International Conference, Revised Selected Papers, ACG 2011, Tilburg, The Netherlands, November 20–22, 2011*. Ed. by H. J. van den Herik and A. Plaat. Vol. 7168. LNCS. Springer, 2012, pp. 320–332. ISBN: 978-3-642-31866-5. DOI: 10.1007/978-3-642-31866-5_27 (cit. on pp. 107, 108).
- [MSO⁺12c] G. Maggiore, P. Spronck, R. Orsini, M. Bugliesi, E. Steffinlongo, and M. Abbadi. "Writing Real-Time .Net Games in Casanova". In: *Entertainment Computing – Proceedings of the 11th International Conference on Entertainment Computing, ICEC 2012, Bremen, Germany, September 26–29, 2012*. Ed. by M. Herrlich, R. Malaka, and M. Masuch. Vol. 7522. LNCS. Springer, 2012, pp. 341–348. ISBN: 978-3-642-33542-6. DOI: 10.1007/978-3-642-33542-6_30 (cit. on pp. 41, 107, 108).
- [Mag12] G. Maggiore. "Casanova: A Language for Game Development". PhD thesis. Università Ca' Foscari di Venezia, Dec. 2012 (cit. on p. 127).
- [MTY11a] T. Mahlmann, J. Togelius, and G. N. Yannakakis. "Modelling and Evaluation of Complex Scenarios with the Strategy Game Description Language". In: *Proceedings of the 2011 IEEE Conference on Computational Intelligence and Games, CIG 2011, Seoul, South Korea, August 31–September 3, 2011*. 2011, pp. 174–181. ISBN: 978-1-4577-0011-8. DOI: 10.1109/CIG.2011.6032004 (cit. on pp. 37, 98).
- [Mah13] T. Mahlmann. "Modelling and Generating Strategy Games Mechanics". PhD thesis. IT University of Copenhagen, Mar. 2013 (cit. on pp. 98, 128).
- [MTY11b] T. Mahlmann, J. Togelius, and G. N. Yannakakis. "Towards Procedural Strategy Game Generation: Evolving Complementary Unit Types". In: *Applications of Evolutionary Computation – Proceedings of EvoApplications 2011: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Torino, Italy, April 27–29, 2011*. Ed. by C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. I. Esparcia-Alcázar, J. J. Merelo, F. Neri, M. Preuss, H. Richter, J. Togelius, and G. N. Yannakakis. Vol. 6624. LNCS. Springer, 2011, pp. 93–102. ISBN: 978-3-642-20525-5 (cit. on p. 98).

- [MV08] S. Maier and D. Volk. "Facilitating Language-Oriented Game Development by the Help of Language Workbenches". In: *Proceedings of the 2008 Conference on Future Play: Research, Play, Share, Future Play 2008, Toronto, Ontario, Canada, November 3-5, 2008*. ACM, 2008, pp. 224-227. ISBN: 978-1-60558-218-4. DOI: 10.1145/1496984.1497029 (cit. on pp. 41, 117, 118).
- [MR15] S. Maoz and J. O. Ringert. "A Framework for Relating Syntactic and Semantic Model Differences". In: *Proceedings of the ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems, MODELS 2015, Ottawa, ON, Canada, September 30 - October 2, 2015*. IEEE, 2015, pp. 24-33. ISBN: 978-1-4673-6908-4. DOI: 10.1109/MODELS.2015.7338232 (cit. on p. 231).
- [MRR11] S. Maoz, J. O. Ringert, and B. Rumpe. "A Manifesto for Semantic Model Differencing". In: *Models in Software Engineering - Workshops and Symposia at MODELS 2010, Oslo, Norway, October 2-8, 2010, Reports and Revised Selected Papers*. Ed. by J. Dingel and A. Solberg. Vol. 6627. LNCS. Springer, 2011, pp. 194-203. ISBN: 978-3-642-21210-9. DOI: 10.1007/978-3-642-21210-9_19 (cit. on p. 231).
- [MTdB⁺12] E. J. Marchiori, J. Torrente, Á. del Blanco, P. Moreno-Ger, P. Sancho, and B. Fernández-Manjón. "A Narrative Metaphor to Facilitate Educational Game Authoring". In: *Computers & Education* 58.1 (2012), pp. 590-599. ISSN: 0360-1315. DOI: 10.1016/j.compedu.2011.09.017 (cit. on p. 77).
- [MWH⁺12] B. Marne, J. Wisdom, B. Huynh-Kim-Bang, and J.-M. Labat. "A Design Pattern Library for Mutual Understanding and Cooperation in Serious Game Design". In: *Intelligent Tutoring Systems - Proceedings of the 11th International Conference, ITS 2012, Chania, Crete, Greece, June 14-18, 2012*. Ed. by S. A. Cerri, W. J. Clancey, G. Papadourakis, and K. Panourgia. Vol. 7315. LNCS. Springer, 2012, pp. 135-140. ISBN: 978-3-642-30950-2 (cit. on p. 52).
- [MBB⁺12] E. Marques, V. Balegas, B. F. Barroca, A. Barisic, and V. Amaral. "The RPG DSL: A Case Study of Language Engineering Using MDD for Generating RPG Games for Mobile Phones". In: *Proceedings of the 12th Workshop on Domain-Specific Modeling, DSM 2012, Tucson, Arizona, USA, October 22 2012*. ACM, 2012, pp. 13-18. ISBN: 978-1-4503-1634-7. DOI: 10.1145/2420918.2420923 (cit. on p. 35).
- [Mar15] C. Martens. "Ceptre: A Language for Modeling Generative Interactive Systems". In: *Proceedings of the 11th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2015, University of California, Santa Cruz, USA, November 14-18, 2015*. AAAI, 2015 (cit. on pp. 37, 79, 80, 128).
- [MBO18] C. Martens, E. Butler, and J. C. Osborn. "A Resourceful Reframing of Behavior Trees". In: *CoRR abs/1803.09099* (2018). URL: <http://arxiv.org/abs/1803.09099> (cit. on pp. 73, 74).
- [MFB⁺14] C. Martens, J. F. Ferreira, A.-G. Bossler, and M. Cavazza. "Generative Story Worlds as Linear Logic Programs". In: *Proceedings of the 7th Intelligent Narrative Technologies Workshop, Wisconsin, USA, June 17-18, 2014*. AAAI, 2014 (cit. on pp. 39, 79, 80).
- [MR05] M. Masuch and M. Rueger. "Challenges in Collaborative Game Design: Developing Learning Environments for Creating Games". In: *Proceedings of the 3rd International Conference on Creating, Connecting and Collaborating through Computing, C5 2005, Kyoto, Japan, Japan, January 28-29, 2005*. IEEE, 2005, pp. 67-74. DOI: 10.1109/C5.2005.7 (cit. on p. 90).

- [MHZ15] A. Matallaoui, P. Herzig, and R. Zarnekow. "Model-Driven Serious Game Development Integration of the Gamification Modeling Language GaML with Unity". In: *Proceedings of the 48th Annual Hawaii International Conference on System Sciences, HICSSS 2015, Kauai, HI, USA, January 5–8, 2015*. IEEE, 2015, pp. 643–651. DOI: 10.1109/HICSS.2015.84 (cit. on p. 92).
- [MSo2] M. Mateas and A. Stern. "A Behavior Language for Story-Based Believable Agents". In: *IEEE Intelligent Systems* 17.4 (July 2002), pp. 39–47. ISSN: 1541-1672. DOI: 10.1109/MIS.2002.1024751 (cit. on p. 71).
- [MN09] M. Mateas and W.-F. Noah. "Defining Operational Logics". In: *Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory, DiGRA 2009, London, UK, September 1–4, 2009*. Ed. by T. Krzywinska, H. W. Kennedy, and B. Atkins. Brunel University and Digital Games Research Association, 2009 (cit. on pp. 52, 53).
- [MSo5a] M. Mateas and A. Stern. "Build It to Understand It: Ludology Meets Narratology in Game Design Space". In: *Proceedings of the 2005 DiGRA International Conference: Changing Views: Worlds in Play, DiGRA 2005, Vancouver, Canada, June 16–20, 2005*. Digital Games Research Association, 2005 (cit. on pp. 20, 41, 71).
- [MSo5b] M. Mateas and A. Stern. "Structuring Content Within the Façade Interactive Drama Architecture". In: *Proceedings of the 1st AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2005, Marina del Rey, California, USA, June 1–3, 2005*. Ed. by R. M. Young and J. Laird. AAAI, 2005. ISBN: 978-1-57735-235-8 (cit. on pp. 37, 71).
- [May17] M. Mayer. "Interactive Programming by Example". PhD thesis. École Polytechnique Fédérale de Lausanne, Apr. 2017 (cit. on pp. 115, 128).
- [MK13] M. Mayer and V. Kuncak. "Game Programming by Demonstration". In: *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, Onward! 2013, Indianapolis, Indiana, USA, October 29–31, 2013*. ACM, 2013, pp. 75–90. ISBN: 978-1-4503-2472-4. DOI: 10.1145/2509578.2509583 (cit. on pp. 35, 115).
- [MB95] D. L. McGuinness and A. T. Borgida. "Explaining Subsumption in Description Logics". In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI 1995, Montreal, Quebec, Canada, August 20–25, 1995*. Morgan Kaufmann, 1995, pp. 816–821. ISBN: 978-1-558-60363-9 (cit. on p. 232).
- [MCS⁺04a] M. McNaughton, M. Cutumisu, D. Szafron, J. Schaeffer, J. Redford, and D. Parker. "ScriptEase: Generating Scripting Code for Computer Role-Playing Games". In: *Proceedings of the 19th International Conference on Automated Software Engineering, ASE 2004, Linz, Austria, September 20–24, 2004*. IEEE, 2004, pp. 386–387. ISBN: 0-7695-2131-2. DOI: 10.1109/ASE.2004.1342770 (cit. on pp. 35, 79).
- [MCS⁺04b] M. McNaughton, M. Cutumisu, D. Szafron, J. Schaeffer, J. Redford, and D. Parker. "ScriptEase: Generative Design Patterns for Computer Role-Playing Games". In: *Proceedings of the 19th International Conference on Automated Software Engineering, ASE 2004, Linz, Austria, September 20–24, 2004*. IEEE, 2004, pp. 88–99. ISBN: 0-7695-2131-2. DOI: 10.1109/ASE.2004.1342727 (cit. on pp. 35, 79, 137, 162).
- [MRS⁺03] M. McNaughton, J. Redford, J. Schaeffer, and D. Szafron. "Pattern-based AI Scripting using ScriptEase". In: *Proceedings of the 16th Canadian Society for Computational Studies of Intelligence Conference on Advances in Artificial Intelligence, AI 2003, Halifax, Canada, June 11–13, 2003*. Springer, 2003, pp. 35–49. ISBN: 3-540-40300-0 (cit. on pp. 77, 79).

- [Meh13] F. Mehm. "Authoring of Adaptive Single-Player Educational Games". PhD thesis. Technische Universität Darmstadt, Jan. 2013 (cit. on pp. 54, 128).
- [MDM16] F. Mehm, R. Dörner, and M. Masuch. "Authoring Processes and Tools". In: *Serious Games: Foundations, Concepts and Practice*. Ed. by R. Dörner, S. Göbel, W. Effelsberg, and J. Wiemeyer. Springer, 2016, pp. 83–106. ISBN: 978-3-319-40612-1. DOI: 10.1007/978-3-319-40612-1_4 (cit. on p. 54).
- [MGR⁺09] F. Mehm, S. Göbel, S. Radke, and R. Steinmetz. "Authoring Environment for Story-Based Digital Educational Games". In: *Proceedings of the 1st International Open Workshop on Intelligent Personalization and Adaptation in Digital Educational Games*. Ed. by M. D. Kickmeier-Rust. 2009, pp. 113–124 (cit. on pp. 53, 54).
- [MRG⁺12] F. Mehm, C. Reuter, S. Göbel, and R. Steinmetz. "Future Trends in Game Authoring Tools". In: *Entertainment Computing – Proceedings of the 11th International Conference on Entertainment Computing, ICEC 2012, as part of the 2nd Workshop on Game Development and Model-Driven Software Development, GD&MDS 2012, Bremen, Germany, September 26–29, 2012*. Ed. by M. Herrlich, R. Malaka, and M. Masuch. Springer, 2012, pp. 536–541. ISBN: 978-3-642-33542-6. DOI: 10.1007/978-3-642-33542-6_70 (cit. on p. 127).
- [MWG⁺10] F. Mehm, V. Wendel, S. Göbel, and R. Steinmetz. "Bat Cave: A Testing and Evaluation Platform for Digital Educational Games". In: *Proceedings of the 4th European Conference on Games Based Learning, ECGBL 2010, Copenhagen, Denmark, October 21–22, 2010*. Academic Conferences International Limited, 2010, pp. 251–260. ISBN: 978-1-62276-708-3 (cit. on p. 54).
- [Men14] P. Mennig. "Modeling Gamification Rules with Ontologies for Achieving a Generic Platform Architecture". In: *FHWS Science Journal* 2.1 (2014), pp. 60–70 (cit. on p. 42).
- [Meno8] T. Mens. "Introduction and Roadmap: History and Challenges of Software Evolution". In: *Software Evolution*. Springer, 2008, pp. 1–11. ISBN: 978-3-540-76440-3. DOI: 10.1007/978-3-540-76440-3_1 (cit. on pp. 20, 244).
- [MHS05] M. Mernik, J. Heering, and A. M. Sloane. "When and How to Develop Domain-Specific Languages". In: *ACM Computing Surveys* 37.4 (Dec. 2005), pp. 316–344. ISSN: 0360-0300. DOI: 10.1145/1118890.1118892 (cit. on pp. 28, 29).
- [MM85] W. Miller and E. W. Myers. "A File Comparison Program". In: *Software Practice and Experience* 15.11 (Nov. 1985), pp. 1025–1040. DOI: 10.1002/spe.4380151102 (cit. on p. 208).
- [MC09a] E. Montero Reyno and J. Á. Carsí Cubel. "A Platform-Independent Model for Videogame Gameplay Specification". In: *Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory, DiGRA 2009, West London, UK, September 1–4, 2009*. Brunel University, 2009 (cit. on pp. 41, 111).
- [MC08] E. Montero Reyno and J. Á. Carsí Cubel. "Model Driven Game Development: 2D Platform Game Prototyping". In: *Proceedings of the 9th International Conference on Intelligent Games and Simulation, GAME-ON 2008, Valencia, Spain, November 17–19, 2008*. EUROESIS, 2008, pp. 5–7. ISBN: 978-90-77381-45-8 (cit. on pp. 41, 111).
- [MC09b] E. Montero Reyno and J. Á. Carsí Cubel. "Automatic Prototyping in Model-driven Game Development". In: *Computers in Entertainment – Special Issue on Media Arts and Games* 7.2 (June 2009), pp. 1–9. ISSN: 1544-3574. DOI: 10.1145/1541895.1541909 (cit. on pp. 41, 111).

- [MBS⁺07] P. Moreno-Ger, D. Burgos, J. L. Sierra, and B. Fernández-Manjón. "An eLearning Specification Meets a Game: Authoring and Integration with IMS Learning Design and <e-Adventure>". In: *Organizing and Learning through Gaming and Simulation – Proceedings of the 38th Conference of the International Simulation And Gaming Association, ISAGA 2007, Nijmegen, The Netherlands, July 9–13, 2007*. Ed. by I. Mayer and H. Mastik. Eburon, 2007. ISBN: 9789059722316 (cit. on pp. 39, 76, 77).
- [MFS⁺09] P. Moreno-Ger, R. Fuentes-Fernández, J.-L. Sierra-Rodríguez, and B. Fernández-Manjón. "Model-Checking for Adventure Videogames". In: *Information and Software Technology* 51.3 (2009), pp. 564–580. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2008.08.003 (cit. on pp. 76, 77).
- [MMS⁺06] P. Moreno-Ger, I. Martínez-Ortiz, J. L. Sierra, and B. F. Manjón. "Language-Driven Development of Videogames: The <e-Game> Experience". In: *Entertainment Computing – Proceedings of the 5th International Conference on Entertainment Computing, ICEC 2006, Cambridge, UK, September 20–22, 2006*. Ed. by R. Harper, M. Rauterberg, and M. Combetto. Vol. 4161. LNCS. Springer, 2006, pp. 153–164. ISBN: 978-3-540-45261-4. DOI: 10.1007/11872320_19 (cit. on pp. 41, 76, 77).
- [MSM⁺07] P. Moreno-Ger, J. L. Sierra, I. Martínez-Ortiz, and B. Fernández-Manjón. "A Documental Approach to Adventure Game Development". In: *Science of Computer Programming – Special Issue on Aspects of Game Programming* 67.1 (June 2007). Ed. by K. de Leeuw, pp. 3–31. ISSN: 0167-6423. DOI: 10.1016/j.scico.2006.07.003 (cit. on pp. 36, 76, 77).
- [MBJ⁺09] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg. "Models@Runtime to Support Dynamic Adaptation". In: *Computer* 42.10 (Oct. 2009), pp. 44–51. ISSN: 0018-9162. DOI: 10.1109/MC.2009.327 (cit. on p. 232).
- [MRF⁺16] J. B. Mossmann, R. Rieder, C. D. Flores, and M. S. Pinho. "Project and Preliminary Evaluation of VR-MED, a Domain-Specific Language for Serious Games in Family Medicine Teaching". In: *Proceedings of the IEEE 40th Annual Computer Software and Applications Conference, COMPSAC 2016, Atlanta, Georgia, USA, June 10–14 2016*. IEEE, 2016, pp. 663–667. DOI: 10.1109/COMPSAC.2016.171 (cit. on p. 56).
- [MFJ05] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel. "Weaving Executability into Object-Oriented Meta-Languages". In: *Model Driven Engineering Languages and Systems – Proceedings of the 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2–7, 2005*. Vol. 3713. LNCS. Springer, 2005, pp. 264–278. ISBN: 978-3-540-32057-9. DOI: 10.1007/11557432_19 (cit. on pp. 204, 231).
- [Mur89] T. Murata. "Petri nets: Properties, analysis and applications". In: *Proceedings of the IEEE* 77.4 (Apr. 1989), pp. 541–580. ISSN: 0018-9219. DOI: 10.1109/5.24143 (cit. on p. 58).
- [Mye86] E. W. Myers. "An $O(ND)$ Difference Algorithm and its Variations". In: *Algorithmica* 1.1-4 (Nov. 1986), pp. 251–266. ISSN: 1432-0541. DOI: 10.1007/BF01840446 (cit. on p. 213).
- [EDC16] M. S. El-Nasr, A. Drachen, and A. Canossa. *Game Analytics*. Springer, 2016. ISBN: 978-1-4471-4769-5. DOI: 10.1007/978-1-4471-4769-5 (cit. on p. 86).
- [ESo6] M. S. El-Nasr and B. K. Smith. "Learning Through Game Modding". In: *Computers in Entertainment* 4.1 (Jan. 2006). ISSN: 1544-3574. DOI: 10.1145/1111293.1111301 (cit. on p. 41).
- [NV03] S. Natkin and L. Vega. "Petri Net Modelling for the Analysis of the Ordering of Actions in Computer Games". In: *Proceedings of the 4th International Conference on Intelligent Games and Simulation, GAME-ON 2003, London, UK, November 19–21, 2003*. Ed. by Q. H. Mehdi, N. E. Gough, and S. Natkin. EUROSIS, 2003, pp. 82–89. ISBN: 9-0773-8105-8 (cit. on pp. 41, 58).

- [NVGo4] S. Natkin, L. Vega, and S. Grünvogel. "A new Methodology for Spatiotemporal Game Design". In: *Proceedings of the 5th Game-On International Conference on Computer Games: Artificial Intelligence, Design and Education, CGAIDE 2004, Reading, UK, November 8–10, 2004*. Ed. by Q. Mehdi and N. Gough. University of Wolverhampton, 2004, pp. 109–113. ISBN: 0-9549016-0-6 (cit. on pp. 41, 58).
- [Nei12] K. Neil. "Game Design Tools: Time to Evaluate". In: *Proceedings of 2012 International DiGRA Nordic Conference: Local and Global: Games in Culture and Society, Tampere, Finland, June 6–8, 2012*. Digital Games Research Association, 2012 (cit. on p. 161).
- [Nei15] K. Neil. "Game Design Tools: Can They Improve Game Design Practice?" PhD thesis. Flinders University, Dec. 2015 (cit. on pp. 127, 128).
- [Nel12] M. J. Nelson. *Sicart's 'Against Proceduralism' – A reply*. Kmjn.org. May 2012. URL: http://www.kmjn.org/notes/sicart_against_proceduralism.html (visited on Oct. 25, 2018) (cit. on p. 57).
- [NM07] M. J. Nelson and M. Mateas. "Towards Automated Game Design". In: *Artificial Intelligence and Human-Oriented Computing – Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence, AI*IA 2007, Rome, Italy, September 10–13, 2007*. Ed. by R. Basili and M. T. Paziienza. Vol. 4633. LNCS. Springer, 2007, pp. 626–637. ISBN: 978-3-540-74782-6. DOI: 10.1007/978-3-540-74782-6_54 (cit. on p. 59).
- [NM08a] M. J. Nelson and M. Mateas. "An Interactive Game-Design Assistant". In: *Proceedings of the 13th International Conference on Intelligent User Interfaces, IUI 2008, Gran Canaria, Spain, January 13–16, 2008*. ACM, 2008, pp. 90–98. ISBN: 978-1-59593-987-6. DOI: 10.1145/1378773.1378786 (cit. on pp. 37, 59, 161, 162, 187).
- [NM08b] M. Nelson and M. Mateas. "Recombinable Game Mechanics for Automated Design Support". In: *Proceedings of the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2008, Stanford, California, USA, October 22–24, 2008*. Ed. by M. Mateas and C. Darken. AAAI, 2008, pp. 84–89. ISBN: 978-1-57735-392-8 (cit. on p. 60).
- [NK06] T. Nishimori and Y. Kuno. "Mogemoge: A Programming Language Based on Join Tokens". In: *Proceedings of The International Workshop on Information Science Education & Programming Languages, Korean University & University of Tsukuba*. 2006, pp. 22–27 (cit. on p. 106).
- [NK12a] T. Nishimori and Y. Kuno. "Join Token: A Language Mechanism for Programming Interactive Games". In: *Entertainment Computing 3.2* (May 2012), pp. 19–25. ISSN: 1875-9521. DOI: 10.1016/j.entcom.2011.09.001 (cit. on p. 106).
- [NK12b] T. Nishimori and Y. Kuno. "Join Token-Based Event Handling: A Comprehensive Framework for Game Programming". In: *Software Language Engineering – Proceedings of the 4th International Conference, SLE 2011, Braga, Portugal, July 3–4, 2011, Revised Selected Papers*. Ed. by A. Sloane and U. Alsmann. Vol. 6940. LNCS. Springer, 2012, pp. 119–138. ISBN: 978-3-642-28830-2. DOI: 10.1007/978-3-642-28830-2_7 (cit. on pp. 35, 106).
- [Num08] T. Nummenmaa. "Adding Probabilistic Modeling to Executable Formal DisCo Specifications with Applications in Strategy Modeling in Multiplayer Game Design". MA thesis. University of Tampere – Department of Computer Sciences, June 2008 (cit. on p. 104).
- [NBM09] T. Nummenmaa, E. Berki, and T. Mikkonen. "Exploring Games as Formal Models". In: *Proceedings of the 4th South-East European Workshop on Formal Methods, SEEFM 2009, Thessaloniki, Greece, December 4–5, 2009*. IEEE, 2009, pp. 60–65. ISBN: 978-0-7695-3943-0. DOI: 10.1109/SEEFM.2009.15 (cit. on p. 104).

- [NKH09] T. Nummenmaa, J. Kuittinen, and J. Holopainen. "Simulation as a Game Design Tool". In: *Proceedings of the International Conference on Advances in Computer Entertainment Technology, ACE 2009, Athens, Greece, October 29–31, 2009*. ACM, 2009, pp. 232–239. ISBN: 978-1-60558-864-3. DOI: 10.1145/1690388.1690427 (cit. on pp. 41, 104).
- [OZ10] S. Ontañón and J. Zhu. "Story and Text Generation through Computational Analogy in the Riu System". In: *Proceedings of the 6th Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2010, Stanford, CA, USA, October 11–13, 2010*. AAAI, 2010. ISBN: 978-1-57735-479-6 (cit. on p. 37).
- [OHB⁺13] F. R. Ortega, F. Hernandez, A. Barreto, N. D. Rische, M. Adjouadi, and S. Liu. "Exploring Modeling Language for Multi-touch Systems Using Petri Nets". In: *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces, ITS 2013, St. Andrews, Scotland, UK, October 6–9, 2013*. ACM, 2013, pp. 361–364. ISBN: 978-1-4503-2271-3. DOI: 10.1145/2512349.2512400 (cit. on p. 58).
- [Orwooa] J. Orwant. "EGGG: The Extensible Graphical Game Generator". PhD thesis. Massachusetts Institute of Technology, Feb. 2000. URL: <http://hdl.handle.net/1721.1/9164> (cit. on pp. 105, 106).
- [Orwoob] J. Orwant. "EGGG: Automated Programming for Game Generation". In: *IBM Systems Journal* 39.3 (2000), pp. 782–794. ISSN: 0018-8670. DOI: 10.1147/sj.393.0782 (cit. on p. 106).
- [OSM⁺15] J. Osborn, B. Samuel, M. Mateas, and N. Wardrip-Fruin. "Playspecs: Regular Expressions for Game Play Traces". In: *Proceedings of the 11th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2015, University of California, Santa Cruz November 14–18, 2015*. AAAI, 2015 (cit. on pp. 37, 86).
- [OSM17] J. C. Osborn, A. Summerville, and M. Mateas. "Automated Game Design Learning". In: *IEEE Conference on Computational Intelligence and Games, CIG 2017, New York, NY, USA, August 22–25, 2017*. IEEE, 2017, pp. 240–247. DOI: 10.1109/CIG.2017.8080442 (cit. on pp. 52, 53).
- [OWM17] J. C. Osborn, N. Wardrip-Fruin, and M. Mateas. "Refining Operational Logics". In: *Proceedings of the International Conference on the Foundations of Digital Games, FDG 2017, Hyannis, MA, USA, August 14–17, 2017*. Ed. by S. Deterding, A. Canossa, C. Hartevelde, J. Zhu, and M. Sicart. ACM, 2017, pp. 1–10. DOI: 10.1145/3102071.3102107 (cit. on pp. 52, 53).
- [Osb18] J. C. Osborn. "Operationalizing Operational Logics". PhD thesis. UC Santa Cruz, June 2018 (cit. on pp. 52, 53, 86, 128).
- [OGM13] J. C. Osborn, A. Grow, and M. Mateas. "Modular Computational Critics for Games". In: *Proceedings of the 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2013, Boston, USA, October 14–15, 2013*. Ed. by G. Sukthankar and I. Horswill. AAAI, 2013. ISBN: 978-1-57735-607-3 (cit. on pp. 37, 64, 186, 187).
- [Oveo4a] M. Overmars. "Teaching Computer Science Through Game Design". In: *Computer* 37.4 (Apr. 2004), pp. 81–83. ISSN: 0018-9162. DOI: 10.1109/MC.2004.1297314 (cit. on p. 105).
- [Oveo4b] M. Overmars. "Learning Object-Oriented Design by Creating Games". In: *IEEE Potentials* 23.5 (2004), pp. 11–13. ISSN: 1558-1772. DOI: 10.1109/MP.2005.1368910 (cit. on p. 105).
- [PABo6] R. F. Paige, T. S. Attridge, and P. J. Brooke. "Game Development using Design-by-Contract". In: *Journal of Object Technology* 5.7 (Sept. 2006), pp. 57–73. DOI: 10.5381/jot.2006.5.7.a3 (cit. on p. 105).

- [PKP13] R. F. Paige, D. S. Kolovos, and F. A. C. Polack. "Metamodelling for Grammarware Researchers". In: *Software Language Engineering, 5th International Conference, SLE 2012, Dresden, Germany, September 26–28, 2012, Revised Selected Papers*. Ed. by K. Czarnecki and G. Hedin. Vol. 7745. LNCS. Springer, 2013, pp. 64–82. ISBN: 978-3-642-36089-3. DOI: 10.1007/978-3-642-36089-3_5 (cit. on pp. 35, 110).
- [Pal10] J. D. Palmer. "Ficticious: MicroLanguages for Interactive Fiction". In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion, SPLASH 2010, as part of Onward! 2010, Reno/Tahoe, Nevada, USA, October 17–21, 2010*. ACM, 2010, pp. 61–68. ISBN: 978-1-4503-0240-1. DOI: 10.1145/1869542.1869551 (cit. on pp. 35, 118, 119).
- [PFM⁺08] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. "Systematic Mapping Studies in Software Engineering". In: *12th International Conference on Evaluation and Assessment in Software Engineering, EASE 2008, University of Bari, Italy, June 26–27, 2008*. Ed. by G. Visaggio, M. T. Baldassarre, S. G. Linkman, and M. Turner. Workshops in Computing. BCS, 2008 (cit. on p. 26).
- [Phi14] D. Phillips. *Creating Apps in Kivy: Mobile with Python*. O'Reilly, 2014. ISBN: 9781491946671 (cit. on p. 103).
- [PVM05] C. J. F. Pickett, C. Verbrugge, and F. Martineau. "(P)NFG: A Language and Runtime System for Structured Computer Narratives". In: *Proceedings of the 1st International North American Conference on Intelligent Games and Simulation, GAME-ON-NA 2005, Montreal, Canada, August 22–23, 2005*. EUROSIS, 2005, pp. 23–32. ISBN: 90-77381-19-8 (cit. on pp. 41, 80, 82).
- [PLW⁺10] D. Pizzi, J. Lugin, A. Whittaker, and M. Cavazza. "Automatic Generation of Game Level Solutions as Storyboards". In: *IEEE Transactions on Computational Intelligence and AI in Games* 2.3 (Sept. 2010), pp. 149–161. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2010.2070066 (cit. on pp. 37, 83).
- [PCW⁺08] D. Pizzi, M. Cavazza, A. Whittaker, and J.-L. Lugin. "Automatic Generation of Game Level Solutions as Storyboards". In: *Proceedings of the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2008, Stanford, California, USA, October 22–24, 2008*. Ed. by M. Mateas and C. Darken. AAAI, 2008, pp. 96–101. ISBN: 978-1-57735-392-8 (cit. on pp. 37, 83).
- [PKG15] T. Pløhn, J. Krogstie, and H. Guo. "Extending the Pervasive Game Ontology through a Case Study". In: *NOKOBIT* 23.1 (2015) (cit. on pp. 44, 45).
- [RKB13] K. Raies, M. Khemaja, and R. Braham. "Towards Game Based Learning Design Process Based on Semantic Service Oriented Architecture (SSOA)". In: *Proceedings of the 7th European Conference on Games Based Learning, ECGBL 2013, Porto, Portugal, October 3–4, 2013*. Ed. by C. V. de Carvalho and P. Escudeiro. Academic Conferences International Limited, 2013, pp. 698–705. ISBN: 978-1-62993-139-5 (cit. on p. 42).
- [Rep11] A. Repenning. "Making Programming More Conversational". In: *Proceedings of the 2011 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2011, Pittsburgh, Pennsylvania, USA, September 18–22, 2011*. Ed. by G. Costagliola, A. Ko, A. Cypher, J. N. C. Scaffidi, C. Kelleher, and B. Myers. IEEE, 2011, pp. 191–194. DOI: 10.1109/VLHCC.2011.6070398 (cit. on p. 91).
- [RS95] A. Repenning and T. Sumner. "AgentSheets: A Medium for Creating Domain-Oriented Languages". In: *IEEE Computer* 28.3 (Mar. 1995), pp. 17–25. DOI: 10.1109/2.366152 (cit. on p. 91).

- [RMM⁺09] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. "Scratch: Programming for All". In: *Communications of the ACM* 52.11 (Nov. 2009), pp. 60–67. ISSN: 0001-0782. DOI: 10.1145/1592761.1592779 (cit. on pp. 36, 90).
- [RBG97] J. W. Romein, H. E. Bal, and D. Grune. "An Application Domain Specific Language for Describing Board Games". In: *Parallel and Distributed Processing Techniques and Applications*. CSREA, 1997, pp. 305–314 (cit. on p. 94).
- [Rom01] J. W. Romein. "Multigame - An Environment for Distributed Game-Tree Search". PhD thesis. Vrije Universiteit Amsterdam, Jan. 2001 (cit. on p. 94).
- [RBG00] J. W. Romein, H. E. Bal, and D. Grune. *The Multigame Reference Manual*. Tech. rep. Vrije Universiteit Amsterdam, 2000. URL: <https://research.vu.nl/en/publications/the-multigame-reference-manual> (visited on Nov. 13, 2018) (cit. on p. 94).
- [RPK⁺08] L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. C. Polack. "Constructing Models with the Human-Usable Textual Notation". In: *Model Driven Engineering Languages and Systems – Proceedings of the 11th International Conference, MoDELS 2008, Toulouse, France, September 28 – October 3, 2008*. Ed. by K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter. Vol. 5301. LNCS. Springer, 2008, pp. 249–263. ISBN: 978-3-540-87874-2. DOI: 10.1007/978-3-540-87875-9_18 (cit. on p. 230).
- [RDS08] G. Russell, A. F. Donaldson, and P. Sheppard. "Tackling Online Game Development Problems with a Novel Network Scripting Language". In: *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games, NetGames 2008, Worcester, Massachusetts, October 21–22, 2008*. ACM, 2008, pp. 85–90. ISBN: 978-1-60558-132-3. DOI: 10.1145/1517494.1517512 (cit. on p. 107).
- [Sad17] D. V. Sadanand. "Model Checking in General Game Playing: Automated Translation from GDL-II to MCK". MA thesis. Auckland University of Technology – School of Engineering, Computer and Mathematical Sciences, Aug. 2017 (cit. on p. 96).
- [Saf14] A. Saffidine. "The Game Description Language is Turing Complete". In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.4 (Dec. 2014), pp. 320–324. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2014.2354417 (cit. on pp. 37, 96).
- [SMO⁺12] M. G. Salazar, H. A. Mitre, C. L. Olalde, and J. L. G. Sánchez. "Proposal of Game Design Document from Software Engineering Requirements Perspective". In: *Proceedings of the 17th International Conference on Computer Games, CGAMES 2012, Louisville, KY, USA, July 30–August 1, 2012*. IEEE, 2012, pp. 81–85. ISBN: 978-1-4673-1121-2. DOI: 10.1109/CGames.2012.6314556 (cit. on p. 41).
- [Sal07] K. Salen. "Gaming Literacies: A game Design Study in Action". In: *Journal of Educational Multimedia and Hypermedia* 16.3 (July 2007), pp. 301–322. ISSN: 1055-8896. URL: <https://www.learnlib.org/p/24374> (visited on Nov. 8, 2018) (cit. on p. 89).
- [SZ03] K. Salen and E. Zimmerman. *Rules of Play - Game Design Fundamentals*. The MIT Press, 2003. ISBN: 9780262240451 (cit. on pp. 20, 39, 56, 163, 169).
- [SMM07] P. Salomoni, S. Mirri, and L. A. Muratori. "YEAST: The Design of a Cooperative Interactive Story Telling and Gamebooks Environment". In: *Proceedings of the 8th International Conference on Intelligent Games and Simulation, GAME-ON 2007, Bologna, Italy, November 20–22, 2007*. Ed. by M. Roccetti. EUROESIS, 2007, pp. 83–87 (cit. on p. 41).

- [SAA12] V. T. Sarinho, A. L. Apolinário, and E. S. Almeida. "A Feature-Based Environment for Digital Games". In: *Entertainment Computing – Proceedings of the 11th International Conference on Entertainment Computing, ICEC 2012, as part of the 2nd Workshop on Game Development and Model-Driven Software Development, GD&MDS 2012, Bremen, Germany, September 26–29, 2012*. Ed. by M. Herrlich, R. Malaka, and M. Masuch. Vol. 7522. LNCS. Springer, 2012, pp. 518–523. ISBN: 978-3-642-33542-6. DOI: 10.1007/978-3-642-33542-6_67 (cit. on p. 112).
- [SA09] V. T. Sarinho and A. L. Apolinário. "A Generative Programming Approach for Game Development". In: *Proceedings of the 8th Brazilian Symposium on Games and Digital Entertainment, SBGAMES 2009, Rio de Janeiro, Brazil, October 8–10, 2009*. IEEE, 2009, pp. 83–92. DOI: 10.1109/SBGAMES.2009.18 (cit. on pp. 41, 112).
- [Sch13] T. Schaul. "A Video Game Description Language for Model-Based or Interactive Learning". In: *Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games, Canada, August 11–13, 2013*. IEEE, 2013, pp. 1–8. ISBN: 978-1-4673-5311-3. DOI: 10.1109/CIG.2013.6633610 (cit. on pp. 37, 100, 101, 186).
- [Sch14a] T. Schaul. "An Extensible Description Language for Video Games". In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.4 (Dec. 2014), pp. 325–331. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2014.2352795 (cit. on pp. 37, 100, 101).
- [Sch14b] J. Schell. *The Art of Game Design: A Book of Lenses*. AK Peters/CRC Press, 2014 (cit. on pp. 20, 39, 56).
- [SLC⁺13] K. Schenk, A. Lari, M. Church, E. Graves, J. Duncan, R. Miller, N. Desai, R. Zhao, D. Szafron, M. Carbonaro, and J. Schaeffer. "ScriptEase II: Platform Independent Story Creation Using High-Level Patterns". In: *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2013, Boston, Massachusetts, USA, October 14–18, 2013*. Ed. by G. Sukthankar and I. Horswill. AAAI, 2013 (cit. on pp. 78, 79).
- [Sch14c] G. Schmidt. "The Axiom General Purpose Game Playing System". In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.4 (Dec. 2014), pp. 332–342. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2014.2302303 (cit. on p. 37).
- [STN16] N. Shaker, J. Togelius, and M. J. Nelson. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Computational Synthesis and Creative Systems. Springer, 2016. ISBN: 978-3-319-42714-0. DOI: 10.1007/978-3-319-42716-4. URL: <http://pcgbook.com> (cit. on pp. 85, 123, 239).
- [Sico8] M. Sicart. "Defining Game Mechanics". In: *Game Studies* 8.2 (Dec. 2008). ISSN: 1604-7982 (cit. on pp. 39, 56).
- [Sic11] M. Sicart. "Against Procedurality". In: *Game Studies* 11.3 (Dec. 2011). ISSN: 1604-7982 (cit. on p. 57).
- [SBI⁺o8a] C. Simpkins, S. Bhat, C. Isbell Jr., and M. Mateas. "Towards Adaptive Programming: Integrating Reinforcement Learning into a Programming Language". In: *Proceedings of the 23rd ACM SIGPLAN Conference on Object-oriented Programming Systems Languages and Applications, OOPSLA 2008, Nashville, TN, USA, October 19–23, 2008*. ACM, 2008, pp. 603–614. ISBN: 978-1-60558-215-3. DOI: 10.1145/1449764.1449811 (cit. on pp. 35, 71).
- [SBI⁺o8b] C. Simpkins, S. Bhat, C. Isbell Jr., and M. Mateas. "Towards Adaptive Programming: Integrating Reinforcement Learning into a Programming Language". In: *SIGPLAN Notices* 43.10 (Oct. 2008), pp. 603–614. ISSN: 0362-1340. DOI: 10.1145/1449955.1449811 (cit. on pp. 36, 71).

- [STdK⁺11] R. M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra. "A Declarative Approach to Procedural Modeling of Virtual Worlds". In: *Computers & Graphics* 35.2 (2011). Special issue on Virtual Reality in Brazil Visual Computing in Biology and Medicine Semantic 3D media and content Cultural Heritage, pp. 352–363. ISSN: 0097-8493. DOI: 10.1016/j.cag.2010.11.011 (cit. on p. 67).
- [STdK⁺10] R. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra. "Integrating Procedural Generation and Manual Editing of Virtual Worlds". In: *Proceedings of the 1st Workshop on Procedural Content Generation in Games, PCG 2010, Monterey, CA, USA, June 19–21, 2010*. ACM, 2010, pp. 1–8. ISBN: 978-1-4503-0023-0. DOI: 10.1145/1814256.1814258 (cit. on pp. 41, 67).
- [SM10] A. M. Smith and M. Mateas. "Variations Forever: Flexibly Generating Rulesets from a Sculptable Design Space of Mini-Games". In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG 2010, Dublin, Ireland, August 18–21, 2010*. IEEE, 2010, pp. 273–280. ISBN: 978-1-4244-6296-4. DOI: 10.1109/ITW.2010.5593343 (cit. on pp. 37, 60).
- [SM11a] A. M. Smith and M. Mateas. "Answer Set Programming for Procedural Content Generation: A Design Space Approach". In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (Sept. 2011), pp. 187–200. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2011.2158545 (cit. on pp. 37, 60, 239).
- [SNM10] A. M. Smith, M. J. Nelson, and M. Mateas. "LUDOCORE: A Logical Game Engine for Modeling Videogames". In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG 2010, Dublin, Ireland, August 18–21, 2010*. IEEE, 2010, pp. 91–98. ISBN: 978-1-4244-6296-4. DOI: 10.1109/ITW.2010.5593368 (cit. on pp. 37, 59, 60).
- [Smi09] A. M. Smith. "The Intelligent Game Designer: Game Design as a New Domain for Automated Discovery". PhD thesis proposal. UC Santa Cruz, 2009 (cit. on p. 60).
- [Smi12] A. M. Smith. "Mechanizing Exploratory Game Design". PhD thesis. University of California, Santa Cruz, 2012 (cit. on pp. 60, 128).
- [SLH⁺11] A. M. Smith, C. Lewis, K. Hullett, G. Smith, and A. Sullivan. *An Inclusive Taxonomy of Player Modeling*. Tech. rep. UCSC-SOE-11-13. Santa Cruz, California., 2011 (cit. on pp. 37, 67, 187).
- [SM11b] A. M. Smith and M. Mateas. "Towards Knowledge-Oriented Creativity Support in Game Design". In: *Proceedings of the 2nd International Conference on Computational Creativity, ICC 2011, Mexico City, April 27–29, 2011*. Ed. by D. Ventura, P. Gervás, D. F. Harrell, M. L. Maher, A. Pease, and G. Wiggins. Universidad Autónoma Metropolitana, 2011, pp. 129–131. ISBN: 978-607-477-487-0 (cit. on p. 60).
- [SNM09a] A. M. Smith, M. J. Nelson, and M. Mateas. "Computational Support for Play Testing Game Sketches". In: *Proceedings of the 5th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2009, Stanford, California, USA, October 14–16, 2009*. Ed. by C. J. Darken and G. M. Youngblood. Research poster. AAAI, 2009, pp. 167–172. ISBN: 978-1-57735-431-4 (cit. on pp. 37, 59, 60, 186, 187).
- [SNM09b] A. M. Smith, M. J. Nelson, and M. Mateas. "Prototyping Games with BIPED". In: *Proceedings of the 5th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2009, Stanford, California, USA, October 14–16, 2009*. Ed. by C. J. Darken and G. M. Youngblood. Demo. AAAI, 2009, pp. 167–172. ISBN: 978-1-57735-431-4 (cit. on pp. 37, 60).
- [SWM⁺11] G. Smith, J. Whitehead, M. Mateas, M. Treanor, J. March, and M. Cha. "Launchpad: A Rhythm-Based Level Generator for 2-D Platformers". In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.1 (Mar. 2011), pp. 1–16. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2010.2095855 (cit. on pp. 37, 86).

- [STW⁺09] G. Smith, M. Treanor, J. Whitehead, and M. Mateas. "Rhythm-Based Level Generation for 2D Platformers". In: *Proceedings of the 4th International Conference on Foundations of Digital Games, FDG 2009, Orlando, Florida, April 26–30, 2009*. ACM, 2009, pp. 175–182. ISBN: 978-1-60558-437-9. DOI: 10.1145/1536513.1536548 (cit. on p. 86).
- [SW10] G. Smith and J. Whitehead. "Analyzing the Expressive Range of a Level Generator". In: *Proceedings of the 1st Workshop on Procedural Content Generation in Games, PCG 2010, Monterey, California, USA, June 18, 2010*. ACM, 2010, pp. 1–7. ISBN: 978-1-4503-0023-0. DOI: 10.1145/1814256.1814260 (cit. on p. 253).
- [SWM10a] G. Smith, J. Whitehead, and M. Mateas. "Tanagra: A Mixed-Initiative Level Design Tool". In: *Proceedings of the 5th International Conference on the Foundations of Digital Games, FDG 2010, Monterey, CA, USA, June 19–21, 2010*. ACM, 2010, pp. 209–216. ISBN: 978-1-60558-937-4. DOI: 10.1145/1822348.1822376 (cit. on pp. 41, 67).
- [SWM10b] G. Smith, J. Whitehead, and M. Mateas. "Tanagra: An Intelligent Level Design Assistant for 2D Platformers". In: *Proceedings of the 6th Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010, Stanford, California, USA, October 11–13, 2010*. Ed. by G. M. Youngblood and V. Bulitko. Demo. AAAI, 2010. ISBN: 978-1-57735-479-6 (cit. on pp. 37, 67).
- [Spr04] P. Spronck. "CGAIDE AND GAME-ON 2004". In: *ICGA Journal* 27.4 (Dec. 2004), pp. 241–241. ISSN: 2468-2438. DOI: 10.3233/ICG-2004-27410 (cit. on p. 41).
- [SSP04] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma. "Enhancing the Performance of Dynamic Scripting in Computer Games". In: *Entertainment Computing – Proceedings of the 3rd International Conference on Entertainment Computing, ICEC 2004, Eindhoven, The Netherlands, September 1–3, 2004*. Ed. by M. Rauterberg. Vol. 3166. LNCS. Springer, 2004, pp. 296–307. ISBN: 978-3-540-28643-1. DOI: 10.1007/978-3-540-28643-1_38 (cit. on p. 41).
- [SC13] M. Stephan and J. R. Cordy. "A Survey of Model Comparison Approaches and Applications". In: *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2013, Barcelona, Spain, February 19–21, 2013*. Ed. by S. Hammoudi, L. F. Pires, J. Filipe, and R. C. das Neves. SciTePress, 2013, pp. 265–277. ISBN: 978-989-8565-42-6. DOI: 10.5220/0004311102650277 (cit. on p. 230).
- [Sto10] K. T. Stolee. *Kodu Language and Grammar Specification*. Microsoft Research. 2010 (cit. on p. 89).
- [SMS⁺17] A. Summerville, J. R. H. Mariño, S. Snodgrass, S. Ontañón, and L. H. S. Lelis. "Understanding Mario: An Evaluation of Design Metrics for Platformers". In: *Proceedings of the International Conference on the Foundations of Digital Games, FDG 2017, Hyannis, MA, USA, August 14–17, 2017*. Ed. by S. Deterding, A. Canossa, C. Hartevelde, J. Zhu, and M. Sicart. ACM, 2017, pp. 1–10. ISBN: 978-1-4503-5319-9. DOI: 10.1145/3102071.3102080 (cit. on p. 239).
- [SMH⁺19] A. Summerville, C. Martens, S. Harmon, M. Mateas, J. C. Osborn, N. Wardrip-Fruin, and A. Jhala. "From Mechanics to Meaning". In: *IEEE Transactions on Games* 11.1 (Mar. 2019). Online first: October 23rd 2017, pp. 69–78. ISSN: 2475-1510. DOI: 10.1109/TCIAIG.2017.2765599 (cit. on pp. 37, 65, 66).
- [Sweo6] T. Sweeney. "The Next Mainstream Programming Language: a Game Developer's Perspective". In: *Conference record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2006, Charleston, South Carolina, USA, January 11–13, 2006*. Invited talk – Slides available on <https://www.cs.princeton.edu/~dpw/popl/06/Tim-POPL.ppt> (Accessed October 15 2018). ACM, 2006, pp. 269–269. ISBN: 1-59593-027-2. DOI: 10.1145/1111037.1111061 (cit. on pp. 36, 104).

- [Szi07] N. Szilas. "BEcool: Towards an Author Friendly Behaviour Engine". In: *Virtual Storytelling. Using Virtual Reality Technologies for Storytelling – Proceedings of the 4th International Conference, ICVS 2007, Saint-Malo, France, December 5-7, 2007*. Ed. by M. Cavazza and S. Donikian. Vol. 4871. LNCS. Springer, 2007, pp. 102–113. ISBN: 978-3-540-77039-8. DOI: 10.1007/978-3-540-77039-8_9 (cit. on pp. 39, 73).
- [THo8] S. Tang and M. Hanneghan. "Towards a Domain Specific Modelling Language for Serious Game Design". In: *Proceedings of the 6th International Game Design and Technology Workshop and Conference, GDTW 2008, Liverpool, UK, November 12–13, 2008*. Liverpool John Moores University, 2008. ISBN: 9781902560212 (cit. on pp. 111, 112).
- [TH11] S. Tang and M. Hanneghan. "State-of-the-Art Model Driven Game Development: A Survey of Technological Solutions for Game-Based Learning". In: *Journal of Interactive Learning Research* 22.4 (Dec. 2011), pp. 551–605. ISSN: 1093-023X (cit. on pp. 111, 112, 126).
- [THC13] S. Tang, M. Hanneghan, and C. Carter. "A Platform Independent Game Technology Model for Model Driven Serious Games Development". In: *The Electronic Journal of e-Learning* 11.1 (Feb. 2013), pp. 61–79. ISSN: 1479-4403 (cit. on pp. 111, 112).
- [Tan13] S. L. Tanimoto. "A Perspective on the Evolution of Live Programming". In: *Proceedings of the 1st International Workshop on Live Programming, LIVE 2013, San Francisco, CA, USA, May 19, 2013*. IEEE, 2013, pp. 31–34. ISBN: 978-1-4673-6265-8. DOI: 10.1109/LIVE.2013.6617346 (cit. on p. 204).
- [Thi10] M. Thielscher. "A General Game Description Language for Incomplete Information Games". In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11–15, 2010*. Ed. by M. Fox and D. Poole. AAAI, 2010 (cit. on pp. 95, 96).
- [Thi11a] M. Thielscher. "The General Game Playing Description Language is Universal". In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2010, Barcelona, Spain, July 16–22, 2011*. AAAI, 2011, pp. 1107–1112. DOI: 10.5591/978-1-57735-516-8/IJCAI11-189 (cit. on p. 96).
- [Thi11b] M. Thielscher. "Translating General Game Descriptions into an Action Language". In: *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*. Ed. by M. Balduccini and T. C. Son. Vol. 6565. LNCS. Springer, 2011, pp. 300–314. ISBN: 978-3-642-20832-4. DOI: 10.1007/978-3-642-20832-4_19 (cit. on pp. 95, 96).
- [TL14] N. Thillainathan and J. M. Leimeister. "Serious Game Development for Educators - A Serious Game Logic and Structure Modeling Language". In: *EDULEARN14 Proceedings – 6th International Conference on Education and New Learning Technologies, EDULEARN 2014, Barcelona, Spain, July 7–9, 2014*. IATED, 2014, pp. 1196–1206. ISBN: 978-84-617-0557-3 (cit. on p. 92).
- [Tic84] W. F. Tichy. "The String-to-string Correction Problem with Block Moves". In: *ACM Transactions on Computer Systems* 2.4 (Nov. 1984), pp. 309–321. ISSN: 0734-2071. DOI: 10.1145/357401.357404 (cit. on p. 229).
- [TMvdB⁺13] U. Tikhonova, M. Manders, M. van den Brand, S. Andova, and T. Verhoeff. "Applying Model Transformation and Event-B for Specifying an Industrial DSL". In: *Proceedings of the 10th International Workshop on Model Driven Engineering, Verification and Validation MoDeVva 2013, co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2013), Miami, Florida, USA, October 1st, 2013*. Ed. by F. Boulanger, M. Famelis, and D. Ratiu. Vol. 1069. CEUR Workshop Proceedings. CEUR-WS.org, 2013, pp. 41–50 (cit. on p. 21).

- [TS08] J. Togelius and J. Schmidhuber. "An Experiment in Automatic Game Design". In: *Proceedings of the 2008 IEEE Symposium On Computational Intelligence and Games, CIG 2008, Perth, WA, Australia, December 15–18, 2008*. 2008, pp. 111–118. ISBN: 978-1-4244-2973-8. DOI: 10.1109/CIG.2008.5035629 (cit. on pp. 37, 96, 200).
- [TYS⁺11] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. "Search-Based Procedural Content Generation: A Taxonomy and Survey". In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), pp. 172–186. DOI: 10.1109/TCIAIG.2011.2148116 (cit. on p. 93).
- [TBM⁺12] M. Treanor, B. Blackford, M. Mateas, and I. Bogost. "Game-O-Matic: Generating Videogames That Represent Ideas". In: *Proceedings of the 3rd Workshop on Procedural Content Generation in Games, PCG 2012, Raleigh, NC, USA, May 29–June 01, 2012*. ACM, 2012, pp. 1–8. ISBN: 978-1-4503-1447-3. DOI: 10.1145/2538528.2538537 (cit. on pp. 41, 63, 200).
- [TM13] M. Treanor and M. Mateas. "An Account of Proceduralist Meaning". In: *Proceedings of the 2013 DiGRA International Conference: DeFragging Game Studies, DiGRA 2013, Atlanta, GA, USA, August 26–29, 2013*. Ed. by C. Pearce, J. Sharp, and H. W. Kennedy. Digital Games Research Association, 2013 (cit. on p. 57).
- [TSB⁺11] M. Treanor, B. Schweizer, I. Bogost, and M. Mateas. "Proceduralist Readings: How to find meaning in games with graphical logics". In: *Proceedings of the 6th International Conference on the Foundations of Digital Games, FDG 2011, Bordeaux, France, June 28–July 1, 2011*. Ed. by M. Cavazza, K. Isbister, and C. Rich. ACM, 2011, pp. 115–122. DOI: 10.1145/2159365.2159381 (cit. on pp. 63, 65, 66).
- [TSB⁺12] M. Treanor, B. Schweizer, I. Bogost, and M. Mateas. "The Micro-Rhetorics of Game-o-Matic". In: *Proceedings of the 7th International Conference on the Foundations of Digital Games, FDG 2012, Raleigh, North Carolina, USA, May 29–June 01, 2012*. ACM, 2012, pp. 18–25. ISBN: 978-1-4503-1333-9. DOI: 10.1145/2282338.2282347 (cit. on p. 63).
- [TZE⁺15] M. Treanor, A. Zook, M. P. Eladhari, J. Togelius, G. Smith, M. Cook, T. Thompson, B. Magerko, J. Levine, and A. Smith. "AI-Based Game Design Patterns". In: *Proceedings of the 10th International Conference on the Foundations of Digital Games, FDG 2015, Pacific Grove, CA, USA, June 22–25, 2015*. Society for the Advancement of Digital Games, 2015 (cit. on p. 41).
- [TBW⁺07] C. Treude, S. Berlik, S. Wenzel, and U. Kelter. "Difference Computation of Large Models". In: *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ESEC-FSE 2007, Dubrovnik, Croatia, September 3–07, 2007*. ACM, 2007, pp. 295–304. ISBN: 978-1-59593-811-4. DOI: 10.1145/1287624.1287665 (cit. on p. 227).
- [TSB⁺10] T. TuteneL, R. M. Smelik, R. Bidarra, and K. J. de Kraker. "A Semantic Scene Description Language for Procedural Layout Solving Problems". In: *Proceedings of the 6th Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010, Stanford, California, USA, October 11–13, 2010*. Ed. by G. M. Youngblood and V. Bulitko. 2010. ISBN: 978-1-57735-479-6 (cit. on pp. 37, 67).
- [UDNa] UDN Staff. *Unreal Kismet User Guide*. Unreal Engine 3. Epic Games – Unreal Developer Network. URL: <http://udn.epicgames.com/Three/KismetUserGuide.html> (visited on Oct. 24, 2018) (cit. on p. 162).
- [UDNb] UDN Staff. *UnrealScript Language Reference*. Unreal Engine 3. Epic Games – Unreal Developer Network. URL: <http://udn.epicgames.com/Three/UnrealScriptReference.html> (visited on Oct. 24, 2018) (cit. on p. 162).

- [vdBvdS11] J. van den Bos and T. van der Storm. "Bringing Domain-Specific Languages to Digital Forensics". In: *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu, HI, USA, May 21-28, 2011*. Ed. by R. N. Taylor, H. C. Gall, and N. Medvidovic. ACM, 2011, pp. 671–680. DOI: 10.1145/1985793.1985887 (cit. on pp. 21, 216).
- [vdLLB13] R. van der Linden, R. Lopes, and R. Bidarra. "Designing Procedurally Generated Levels". In: *Proceedings of the 2nd workshop on Artificial Intelligence in the Game Design Process. AAAI, 2013* (cit. on p. 239).
- [vdLLB14] R. van der Linden, R. Lopes, and R. Bidarra. "Procedural Generation of Dungeons". In: *IEEE Transactions on Computational Intelligence and AI in Games 6.1* (Mar. 2014), pp. 78–89. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2013.2290371 (cit. on pp. 37, 66, 123, 239).
- [vdSto13] T. van der Storm. "Semantic Deltas for Live DSL Environments". In: *Proceedings of the 1st International Workshop on Live Programming, LIVE 2013, San Francisco, California, USA, May 19 2013*. IEEE, 2013, pp. 35–38. ISBN: 978-1-4673-6265-8. DOI: 10.1109/LIVE.2013.6617347 (cit. on p. 204).
- [vdSCL14] T. van der Storm, W. R. Cook, and A. Loh. "The Design and Implementation of Object Grammars". In: *Science of Computer Programming 96, Part 4.0* (2014). Selected Papers from the 5th International Conference on Software Language Engineering, SLE 2012, pp. 460–487. ISSN: 0167-6423. DOI: 10.1016/j.scico.2014.02.023 (cit. on p. 230).
- [vBVdT15] F. van Broeckhoven, J. Vlieghe, and O. de Troyer. "Mapping between Pedagogical Design Strategies and Serious Game Narratives". In: *Proceedings of the 7th International Conference on Games and Virtual Worlds for Serious Applications, VS-GAMES 2015, Skovde, Sweden, September 16–18, 2015*. 2015, pp. 1–8. ISBN: 978-1-4799-8102-1. DOI: 10.1109/VS-GAMES.2015.7295780 (cit. on p. 55).
- [vBdT13] F. van Broeckhoven and O. de Troyer. "ATTAC-L: A Modeling Language for Educational Virtual Scenarios in the Context of Preventing Cyber Bullying". In: *Proceedings of the IEEE 2nd International Conference on Serious Games and Applications for Health, SeGAH 2013, Algarve, Portugal, May 2–3, 2013*. IEEE, 2013, pp. 1–8. ISBN: 978-1-4673-6165-1. DOI: 10.1109/SeGAH.2013.6665300 (cit. on pp. 54, 55).
- [vBVD15] F. van Broeckhoven, J. Vlieghe, and O. De Troyer. "Using a Controlled Natural Language for Specifying the Narratives of Serious Games". In: *Interactive Storytelling – Proceedings of the 8th International Conference on Interactive Digital Storytelling, ICIDS 2015, Copenhagen, Denmark, November 30–December 4, 2015*. Ed. by H. Schoenau-Fog, L. E. Bruni, S. Louchart, and S. Baceviciute. Vol. 9445. LNCS. Springer, 2015, pp. 142–153. ISBN: 978-3-319-27036-4 (cit. on pp. 39, 54, 55).
- [vDeu97] A. van Deursen. "Domain-Specific Languages versus Object-Oriented Frameworks: A Financial Engineering Case Study". In: *Proceedings Smalltalk and Java in Industry and Academia, STJA'97, Erfurt, September 1997*. Ilmenau Technical University, 1997, pp. 35–39 (cit. on p. 21).
- [vDKT93] A. van Deursen, P. Klint, and F. Tip. "Origin Tracking". In: *Symbolic Computation 15.5/6* (May 1993), pp. 523–545. DOI: 10.1016/S0747-7171(06)80004-0 (cit. on p. 207).
- [vDKV00] A. van Deursen, P. Klint, and J. Visser. "Domain-Specific Languages: An Annotated Bibliography". In: *ACM SIGPLAN NOTICES 35* (2000), pp. 26–36 (cit. on pp. 21, 22, 163, 185, 248).
- [vGOK19] C. van Grinsven, M. Otten, and O. Koops. *Games Monitor The Netherlands 2018 – Full Report*. Ed. by E. Nazarova, E. Muijres, W. Hölzel, A. Loeb, T. Jongens, T. Huisman, K. Kimmel, A. Reimink, and v. S. J. de Wit Richie and. 2019. URL: <https://www.dutchgamegarden.nl> (cit. on p. 17).

- [vHSD⁺16] S. van Hoecke, K. Samyn, G. Deglorie, O. Janssens, P. Lambert, and R. van de Walle. "Enabling Control of 3D Visuals, Scenarios and Non-linear Gameplay in Serious Game Development Through Model-Driven Authoring". In: *Serious Games, Interaction, and Simulation – Proceedings of the 5th International Conference, SGAMES 2015, Novedrate, Italy, September 16–18, 2015*. Ed. by C. Vaz de Carvalho, P. Escudeiro, and A. Coelho. Vol. 161. LNCS. Springer, 2016, pp. 103–110. ISBN: 978-3-319-29060-7. DOI: 10.1007/978-3-319-29060-7.16 (cit. on p. 55).
- [vNvOM⁺11] C. van Nimwegen, H. van Oostendorp, J. Modderman, and M. Bas. "A Test Case for GameDNA: Conceptualizing a Serious Game to Measure Personality Traits". In: *Proceedings of the 16th International Conference on Computer Games, CGAMES 2011, Louisville, KY, USA, July 27–30, 2011*. IEEE, 2011, pp. 217–222. ISBN: 978-1-4577-1452-8. DOI: 10.1109/CGAMES.2011.6000342 (cit. on pp. 41, 54).
- [vRoz15a] R. van Rozen. "A Pattern-Based Game Mechanics Design Assistant". In: *Proceedings of the 10th International Conference on the Foundations of Digital Games, FDG 2015, Pacific Grove, CA, USA, June 22–25, 2015*. Ed. by J. P. Zagal, E. MacCallum-Stewart, and J. Togelius. Society for the Advancement of the Science of Digital Games, 2015 (cit. on pp. 11, 40, 62, 63, 183, 228).
- [vRoz19] R. van Rozen. "Languages of Games and Play: A Systematic Mapping Study". Under submission to ACM Computing Surveys. 2019 (cit. on p. 10).
- [vRD14] R. van Rozen and J. Dormans. "Adapting Game Mechanics with Micro-Machinations". In: *Proceedings of the 9th International Conference on the Foundations of Digital Games, FDG 2014, Liberty of the Seas, Caribbean, April 3–7, 2014*. Ed. by M. Mateas, T. Barnes, and I. Bogost. Society for the Advancement of the Science of Digital Games, 2014. ISBN: 978-0-9913982-2-5 (cit. on pp. 10, 40, 62, 63, 161, 184, 185, 187, 188, 228).
- [vRH18] R. van Rozen and Q. Heijn. "Measuring Quality of Grammars for Procedural Level Generation". In: *Proceedings of the 13th International Conference on Foundations of Digital Games, FDG 2018, as part of the 9th Workshop on Procedural Content Generation, PCG 2018, Malmö, Sweden, August 7–10, 2018*. Ed. by S. Dahlskog, S. Deterding, J. Font, M. Khandaker, C. M. Olsson, S. Risi, and C. Salge. ACM, 2018, pp. 1–8. ISBN: 978-1-4503-6571-0. DOI: 10.1145/3235765.3235821 (cit. on pp. 11, 41, 68, 87, 237).
- [vRvdS15] R. van Rozen and T. van der Storm. "Origin Tracking + Text Differencing = Textual Model Differencing". In: *Theory and Practice of Model Transformations – Proceedings of the 8th International Conference on Model Transformation, ICMT 2015, LAquila, Italy, July 20–21, 2015*. Ed. by D. Kolovos and M. Wimmer. Vol. 9152. LNCS. Springer, 2015, pp. 18–33. ISBN: 978-3-319-21155-8. DOI: 10.1007/978-3-319-21155-8_2 (cit. on pp. 11, 203, 204, 216).
- [vRvdS19] R. van Rozen and T. van der Storm. "Toward Live Domain-Specific Languages: From Text Differencing to Adapting Models at Run Time". In: *Software & Systems Modeling* 18.1 (Feb. 2019). Special Section Paper on STAF2015. Received June 27th 2016. Revised May 26th 2017. Accepted June 20th 2017. First Online August 14th 2017, pp. 195–212. ISSN: 1619-1374. DOI: 10.1007/s10270-017-0608-7 (cit. on pp. 11, 203).
- [VEB⁺07] Y. Vandewoude, P. Ebraert, Y. Berbers, and T. D'Hondt. "Tranquility: A Low Disruptive Alternative to Quiescence for Ensuring Safe Dynamic Updates". In: *IEEE Transactions on Software Engineering* 33.12 (Dec. 2007), pp. 856–868. ISSN: 0098-5589. DOI: 10.1109/TSE.2007.70733 (cit. on p. 232).
- [Varo3] A. Varanese. *Game Scripting Mastery*. Ed. by A. LaMothe. Premier Press, 2003. ISBN: 9781931841573 (cit. on p. 103).

- [Vero3] C. Verbrugge. "A Structure for Modern Computer Narratives". In: *Proceedings of the 3rd international conference on Computers and Games, CG 2002, Edmonton, Canada, July 25-27, 2002, Revised Papers*. Ed. by J. Schaeffer, M. Müller, and Y. Björnsson. Vol. 2883. LNCS. Springer, 2003, pp. 308–325. DOI: 10.1007/978-3-540-40031-8_21 (cit. on pp. 41, 58, 82).
- [VZ10] C. Verbrugge and P. Zhang. "Analyzing Computer Game Narratives". In: *Entertainment Computing – Proceedings of the 9th International Conference on Entertainment Computing, ICEC 2010, Seoul, Korea, September 8–11, 2010*. Ed. by H. S. Yang, R. Malaka, J. Hoshino, and J. H. Han. Vol. 6243. LNCS. Springer, 2010, pp. 224–231. ISBN: 978-3-642-15399-0. DOI: 10.1007/978-3-642-15399-0_21 (cit. on pp. 41, 82).
- [VJP09] K. Villaverde, C. Jeffery, and I. Pivkina. "Cheshire: Towards an Alice Based Game Development Tool". In: *Proceedings of the International Conference on Computer Games, Multimedia & Allied Technology, CGAT 2009, Singapore, May 11–12, 2009*. Research Pub. Services, 2009, pp. 321–328. ISBN: 9789810831905 (cit. on p. 88).
- [VKM⁺08] S. Virmani, Y. Kanetkar, M. Mehta, S. Ontañón, and A. Ram. "An Intelligent IDE for Behavior Authoring in Real-Time Strategy Games". In: *Proceedings of the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2008, Stanford, California, USA, October 22–24, 2008*. Ed. by M. Mateas and C. Darken. AAAI, 2008. ISBN: 978-1-57735-392-8 (cit. on p. 37).
- [WGCo4] R. Wages, B. Grützmacher, and S. Conrad. "Learning from the Movie Industry: Adapting Production Processes for Storytelling in VR". In: *Technologies for Interactive Digital Storytelling and Entertainment – Proceedings of the 2nd International Conference, TIDSE 2004, Darmstadt, Germany, June 24–26, 2004*. Ed. by S. Göbel, U. Spierling, A. Hoffmann, I. Jurgel, O. Schneider, J. Dechau, and A. Feix. Vol. 3105. LNCS. Springer, 2004, pp. 119–125. ISBN: 978-3-540-27797-2. DOI: 10.1007/978-3-540-27797-2_16 (cit. on p. 39).
- [Wal14] R. Walter. "Engineering Domain-Specific Languages for Games". In: *Game Development Tool Essentials*. Ed. by P. Berinstein, R. Arnaud, A. Ardolino, S. Franco, A. Herubel, J. McCutchan, N. Nedelcu, B. Nitschke, D. Olmstead, F. Robinet, C. Ronchi, R. Turkowski, R. Walter, and G. Samour. Apress, 2014, pp. 173–188. ISBN: 978-1-4302-6701-0. DOI: 10.1007/978-1-4302-6701-0_13 (cit. on p. 119).
- [WM11] R. Walter and M. Masuch. "How to Integrate Domain-Specific Languages into the Game Development Process". In: *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology, ACE 2011, Lisbon, Portugal, November 8–11, 2011*. ACM, 2011, pp. 1–8. ISBN: 978-1-4503-0827-4. DOI: 10.1145/2071423.2071475 (cit. on pp. 41, 119, 127).
- [WMW⁺06] K. Wang, C. McCaffrey, D. Wendel, and E. Klopfer. "3D Game Design with Programming Blocks in StarLogo TNG". In: *Proceedings of the 7th International Conference on Learning Sciences, ICLS 2006, Bloomington, Indiana, June 27–July 01, 2006*. International Society of the Learning Sciences, 2006, pp. 1008–1009. ISBN: 0-8058-6174-2 (cit. on p. 91).
- [WSA⁺10] B. H. Wasty, A. Semmo, M. Appeltauer, B. Steinert, and R. Hirschfeld. "ContextLua: Dynamic Behavioral Variations in Computer Games". In: *Proceedings of the 2nd International Workshop on Context-Oriented Programming, COP 2010, Maribor, Slovenia, June 22, 2010*. ACM, 2010, pp. 1–6. ISBN: 978-1-4503-0531-0. DOI: 10.1145/1930021.1930026 (cit. on pp. 103, 104, 162).
- [Wes07] M. West. "Domain-Specific Languages". In: *Game Developer 14.7* (Aug. 2007), pp. 33–36. ISSN: 1073-922X (cit. on pp. 38, 117, 118).

- [Wet13] R. Wetzel. "A Case for Design Patterns Supporting the Development of Mobile Mixed Reality Games". In: *Workshop Proceedings of the 8th International Conference on the Foundations of Digital Games – as part of the 2nd Workshop on Design Patterns in Games, DPG 2013, Chania, Crete, Greece, May, 14–17, 2013*. Society for the Advancement of the Science of Digital Games, 2013 (cit. on pp. 41, 51).
- [Wet14] R. Wetzel. "Introducing Pattern Cards for Mixed Reality Game Design". In: *Proceedings of Workshops Colocated with the 9th International Conference on the Foundations of Digital Games – as part of the 3rd Workshop on Design Patterns in Games, DPG 2014, Liberty of the Seas, Caribbean, April 3–7, 2014*. Society for the Advancement of the Science of Digital Games, 2014. ISBN: 978-0-9913982-3-2 (cit. on pp. 41, 51).
- [WDK⁺07] W. White, A. Demers, C. Koch, J. Gehrke, and R. Rajagopalan. "Scaling Games to Epic Proportions". In: *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, SIGMOD 2007, Beijing, China, June 11–14, 2007*. ACM, 2007, pp. 31–42. ISBN: 978-1-59593-686-8. DOI: 10.1145/1247480.1247486 (cit. on p. 107).
- [WKG⁺08] W. White, C. Koch, J. Gehrke, and A. Demers. "Better Scripts, Better Games". In: *Queue* 6.7 (Nov. 2008), pp. 18–25. ISSN: 1542-7730. DOI: 10.1145/1483101.1483106 (cit. on p. 107).
- [WKG⁺09] W. White, C. Koch, J. Gehrke, and A. Demers. "Better Scripts, Better Games". In: *Communications of the ACM* 52.3 (Mar. 2009), pp. 42–47. ISSN: 0001-0782. DOI: 10.1145/1467247.1467262 (cit. on pp. 36, 107).
- [WSG⁺08] W. White, B. Sowell, J. Gehrke, and A. Demers. "Declarative Processing for Computer Games". In: *Proceedings of the 2008 ACM SIGGRAPH Symposium on Video Games, Sandbox 2008, Los Angeles, California, USA, August 9–10, 2008*. ACM, 2008, pp. 23–30. ISBN: 978-1-60558-173-6. DOI: 10.1145/1401843.1401847 (cit. on pp. 36, 107).
- [WMM⁺06] R. Wieringa, N. A. M. Maiden, N. R. Mead, and C. Rolland. "Requirements Engineering Paper Classification and Evaluation Criteria: A Proposal and a Discussion". In: *Requirements Engineering* 11.1 (Mar. 2006), pp. 102–107. ISSN: 1432-010X. DOI: 10.1007/s00766-005-0021-6 (cit. on p. 26).
- [Wil07] B. Wilcox. "Reflections on Building Three Scripting Languages". In: *Gamasutra* (Apr. 2007). URL: <http://www.gamasutra.com/view/feature/1570/> (visited on Oct. 15, 2018) (cit. on p. 38).
- [WZ13] G. J. Winters and J. Zhu. "Attention Guiding Principles in 3D Adventure Games". In: *SIGGRAPH 2013 Posters*. ACM, 2013, pp. 71–71 (cit. on p. 51).
- [WZ14] G. J. Winters and J. Zhu. "Guiding Players through Structural Composition Patterns in 3D Adventure Games". In: *Proceedings of the 9th International Conference on the Foundations of Digital Games, FDG 2014, Liberty of the Seas, Caribbean, April 3–7, 2014*. Ed. by M. Mateas, T. Barnes, and I. Bogost. Society for the Advancement of the Science of Digital Games, 2014 (cit. on p. 51).
- [Wit53] L. Wittgenstein. *Philosophical Investigations*. Translated by G.E.M. Anscombe. Blackwell, 1953 (cit. on p. 127).
- [WMoo] I. Wright and J. Marshall. "RC++: A Rule-based Language for Game AI". In: *Proceedings of the 1st International Conference on Intelligent Games and Simulation, GAME-ON 2000, London, United Kingdom, November 11–12, 2000*. SCS Europe BVBA, 2000, pp. 42–46 (cit. on p. 41).

- [Wrio3] W. Wright. *Dynamics for Designers – Presentation at the Game Developers Conference, GDC 2003*. GDC Vault. Available online are presentation slides <https://www.slideshare.net/geoffhom/gdc2003-will-wright-presentation>, and a video recording <https://www.gdcvault.com/play/1019938/>. 2003. (Visited on Nov. 22, 2018) (cit. on p. 152).
- [Yan91] W. Yang. “Identifying Syntactic Differences Between Two Programs”. In: *Software Practice and Experience* 21.7 (July 1991), pp. 739–755. DOI: 10.1002/spe.4380210706 (cit. on pp. 209, 230).
- [YT18] G. N. Yannakakis and J. Togelius. *Artificial Intelligence and Games*. Springer, 2018. ISBN: 978-3-319-63518-7. DOI: 10.1007/978-3-319-63519-4. URL: <http://gameaibook.org> (cit. on pp. 37, 239).
- [Zag10] J. P. Zagal. *Ludoliteracy: Defining Understanding and Supporting Games Education*. ETC Press, 2010. ISBN: 978-0-557-27791-9 (cit. on p. 43).
- [ZBL13] J. P. Zagal, S. Björk, and C. Lewis. “Dark Patterns in the Design of Games”. In: *Proceedings of the 8th International Conference on the Foundations of Digital Games, FDG 2013, Chania, Crete, Greece, May 14–17, 2013*. Ed. by G. N. Yannakakis, E. Aarseth, K. Jørgensen, and J. C. Lester. Society for the Advancement of the Science of Digital Games, 2013, pp. 39–46. ISBN: 978-0-9913982-0-1 (cit. on pp. 41, 47).
- [ZMF⁺05] J. P. Zagal, M. Mateas, C. Fernández-vara, B. Hochhalter, and N. Lichti. “Towards an Ontological Language for Game Analysis”. In: *Proceedings of the 2005 DiGRA International Conference: Changing Views: Worlds in Play, DiGRA 2005, Vancouver, Canada, June 16–20, 2005*. Digital Games Research Association, 2005 (cit. on pp. 41, 43, 161, 186).
- [ZMF⁺07] J. P. Zagal, M. Mateas, C. Fernández-vara, B. Hochhalter, and N. Lichti. “Towards an Ontological Language for Game Analysis”. In: *Worlds in Play, International Perspectives on Digital Games Research*. Ed. by S. de Castell and J. Jenson. Peter Lang, 2007, pp. 21–35. ISBN: 978-0-8204-8643-7 (cit. on p. 43).
- [ZTS12] J. P. Zagal, N. Tomuro, and A. Shepitsen. “Natural Language Processing in Game Studies Research: An Overview”. In: *Simulation & Gaming* 43.3 (Oct. 2012), pp. 356–373. ISSN: 1552-826X. DOI: 10.1177/1046878111422560 (cit. on p. 39).
- [ZGT⁺06] N. Zagalo, S. Göbel, A. Torres, R. Malkewitz, and V. Branco. “INSCAPE: Emotion Expression and Experience in an Authoring Environment”. In: *Technologies for Interactive Digital Storytelling and Entertainment – Proceedings of the 3rd International Conference, TIDSE 2006, Darmstadt, Germany, December 4–6, 2006*. Ed. by S. Göbel, R. Malkewitz, and I. Iurgel. Vol. 4326. LNCS. Springer, 2006, pp. 219–230. ISBN: 978-3-540-49935-0. DOI: 10.1007/11944577_23 (cit. on p. 39).
- [Zhu14] M. Zhu. “Model-Driven Game Development Addressing Architectural Diversity and Game Engine-Integration”. PhD thesis. Norwegian University of Science et al., Mar. 2014 (cit. on pp. 116, 128).
- [Zoo16] A. Zook. “Automated Iterative Game Design”. PhD thesis. Georgia Institute of Technology, Dec. 2016 (cit. on p. 128).
- [ZR14a] A. Zook and M. O. Riedl. “Automatic Game Design via Mechanic Generation”. In: *Proceedings of the 28th AAAI Conference on Artificial Intelligence, AAAI 2014, Québec City, Canada, July 27–31, 2014*. AAAI, 2014, pp. 530–537. ISBN: 978-1-57735-661-5 (cit. on pp. 65, 186, 187).
- [ZR14b] A. Zook and M. O. Riedl. “Generating and Adapting Game Mechanics”. In: *Proceedings of the 5th Workshop on Procedural Content Generation, PCG 2014, Liberty of the Seas, Caribbean, April 3–7, 2014*. Society for the Advancement of the Science of Digital Games, 2014 (cit. on pp. 41, 65).

Languages of Games and Play

Automating Game Design & Enabling Live Programming

In game development, the maximum number of game design iterations determines the achievable quality. This thesis explores what informs the design and construction of good games in order to help speed-up game development processes, and create better games more quickly. On the one hand, we study how Domain-Specific Languages (DSLs) can help *automate game design* by offering developers and designers abstractions and notations that raise their productivity, reduce iteration times, and improve the quality of player experiences and a game's source code. On the other hand, we explore how *generic language technology* can be leveraged and developed, in particular for constructing DSLs for *live programming* and automating game design.

The thesis begins with an extensive literature review. We study to what extent *languages, structured notations, patterns and tools*, can offer designers and developers theoretical foundations, systematic techniques and practical solutions they need to raise their productivity and improve the quality of games and play. We propose the term '*languages of games and play*' for language-centric approaches for tackling challenges and solving problems related to game design and development. Despite the growing number of publications on this topic there is currently no overview describing the state-of-the-art that relates research areas, goals and applications. As a result, efforts and successes are often one-off, lessons learned go overlooked, language reuse remains minimal, and opportunities for collaboration and synergy are lost.

Chapter 2 presents a systematic mapping study to map the state of the art in languages of games and play which contributes the following:

1. A systematic map on languages of games and play that provides an overview of research areas and publication venues.
2. A set of fourteen complementary research perspectives on languages of games and play synthesized from summaries of over 100 distinct languages we identified in over 1400 publications.
3. An analysis of general trends and success factors, and one unifying perspective on '*automated game design*', which discusses challenges and opportunities for future research and development.

Our map provides a good starting point for anyone who wishes to learn more about the topic.

The next three chapters focus on *game mechanics*, one of the identified challenge areas. Many games have an *internal economy*, an abstract rule-system that determines player choices and actions that impact gameplay. Using its mechanisms, players face challenges, enact strategies, and manage trade-offs by accumulating, spending and distributing in-game resources (e.g., gems, bricks, or life essence). Unfortunately, game designers lack a common vocabulary for expressing gameplay, which hampers specification, communication and agreement.

The language Machinations has provided a conceptual framework that foregrounds structures and feedback loops associated with patterns of gameplay. We aim to speed up the game development process by improving designer productivity and design quality. We demonstrate how Micro-Machinations (MM), a DSL for game-economic mechanics that evolved from Machinations, can speed-up the game development process by improving designer productivity and game design quality. First, we show in Chapter 3 how meta-programming and model-checking technology can be used to formalize a DSL for game mechanics, and analyze and predict qualities of games. Next, we show in Chapter 4 how the game development process can be accelerated and feedback on game design quality can be improved by adapting and improving the game mechanics of running game software. Finally, we show in Chapter 5 how patterns can be used for constructing an interactive game design assistant (a tool) that statically analyzes and generates game mechanics. We have performed this research in continuous collaboration with industry partners in applied research projects on better languages and tools for automated game design.

Chapter 6 addresses a general software engineering challenge, which is also a key challenge for automated game design. Live programming is a style of development characterized by incremental change and immediate feedback. Instead of long edit-compile cycles, developers modify a running program by changing its source code, receiving immediate feedback as it instantly adapts in response. However, little is known about how the benefits of live programming can be obtained for DSLs. We demonstrate how generic language technology can be developed for constructing DSLs for live programming in a two-part solution. The first, leverages origin tracking and text differencing for *textual model differencing* that produces model-based deltas. The second, demonstrates how to leverage these model-based deltas in the design of DSLs that migrate an application's run-time state by *run-time model patching*. We consider these contributions first steps towards supporting the construction of "live DSLs" in language workbenches.

Finally, Chapter 7 reports preliminary results on improving tool support for automated game level design. Grammar-based procedural level generation raises the productivity of level designers for games such as dungeon crawl and platform games. However, the improved productivity comes at the cost of level quality assurance. Authoring, improving and maintaining grammars is difficult because it is hard to

predict how each grammar rule impacts the overall level quality, and tool support is lacking. We address the lack of tool support by proposing two novel techniques. The first is a novel metric called Metric of Added Detail (MAD) that indicates if a rule adds or removes detail with respect to its phase in the transformation pipeline. The second, Specification Analysis Reporting (SAnR) enables 1) expressing level properties in a simple DSL, and 2) analyzing how qualities evolve in level generation histories. The approach opens a research area for leveraging metaprogramming techniques to address a lack of tool support. The chapter was written as an introductory example, and has the purpose of engaging students as collaborators in future applied research projects to continue this work.

Talen voor Spellen en Spelen

Spelontwerp Automatiseren & Live Programmeren Mogelijk Maken

Bij spelontwerp is het maximale aantal spelontwerpiteraties bepalend voor de kwaliteit die je kunt behalen. Dit proefschrift onderzoekt wat er bepalend is voor het ontwerp en de constructie van goede spellen om te helpen het spelontwikkelproces te versnellen en om sneller betere spellen te maken. Enerzijds bestuderen we hoe Domein-Specifieke Talen (DSLs) kunnen helpen om *spelontwerp te automatiseren* door ontwikkelaars en ontwerpers abstracties en notaties aan te bieden die hun productiviteit verhogen, iteratietijden verkorten, en de kwaliteit van spelervaringen en van de broncode van een spel verbeteren. Anderzijds onderzoeken we hoe *generieke taaltechnologie* kan worden toegepast en ontwikkeld, in het bijzonder voor het bouwen van DSLs voor *live programmeren* om spelontwerp te automatiseren.

Het proefschrift begint met een uitgebreide literatuurstudie. We bestuderen in hoeverre *talen, gestructureerde notaties, patronen en gereedschappen*, ontwerpers en ontwikkelaars theoretische fundamenten, systematische technieken en praktische oplossingen kunnen bieden om hun productiviteit te verhogen en de kwaliteit van spellen en spelen te verbeteren. We introduceren de term *talen voor spellen en spelen* voor taalgerichte benaderingen voor het aanpakken van uitdagingen en het oplossen van problemen met betrekking tot het ontwerp en de ontwikkeling van spellen. Ondanks het groeiende aantal publicaties over dit onderwerp is er op dit moment geen overzicht dat de state-of-the-art in kaart brengt en onderzoeksgebieden, doelen en toepassingen relateert. Hierdoor zijn inspanningen en successen veelal eenmalig, worden wijze lessen over het hoofd gezien, blijft hergebruik van talen minimaal en gaan kansen voor samenwerking en synergie verloren.

Hoofdstuk 2 presenteert een studie die systematisch de state-of-the-art van talen voor spellen en spelen in kaart brengt, en die het volgende bijdraagt:

1. Een systematische kaart van talen voor spellen en spelen die een overzicht biedt van onderzoeksgebieden en publicatieplaatsen.
2. Een lijst van veertien complementaire onderzoeksperspectieven op talen voor spellen en spelen samengesteld uit samenvattingen van meer dan 100 verschillende talen die we geïdentificeerd hebben in meer dan 1400 publicaties.

3. Een analyse van algemene trends en succesfactoren, en een verbindend perspectief op 'geautomatiseerd spelontwerp', waarin uitdagingen en kansen voor toekomstig onderzoek en ontwikkeling worden besproken.

Onze kaart levert een goed beginpunt voor een ieder die meer wil weten over dit onderwerp.

De volgende drie hoofdstukken richten zich op *game mechanics*, een van de geïdentificeerde onderzoeksuitdagingen. Veel spellen hebben een *interne economie*, een abstract regelsysteem dat speleracties en keuzes bepaalt die van invloed zijn op de spelervaringen. Met behulp van deze mechanismen worden spelers geconfronteerd met uitdagingen, voeren ze strategieën uit en maken ze afwegingen door in-game hulpbronnen te verzamelen, te besteden en te verplaatsen (bijv. edelstenen, bouwstenen of levenselixer). Helaas missen spelontwerpers een gemeenschappelijk vocabulaire voor het uitdrukken van spelervaringen, wat de specificatie, de communicatie en de afstemming belemmert.

De taal *Machinations* heeft een conceptueel raamwerk geleverd dat voorziet in structuren en terugkoppelingslussen geassocieerd met patronen van spelervaringen. We stellen tot doel om het spelontwikkelingsproces te versnellen door de productiviteit van ontwerpers en de ontwerp kwaliteit te verbeteren. We demonstreren hoe *Micro-Machinations (MM)*, een DSL voor game-economische mechanics die is voortgekomen uit *Machinations*, het spelontwikkelingsproces kan versnellen door de productiviteit van de ontwerper en de kwaliteit van het spelontwerp te verbeteren.

Eerst laten we in Hoofdstuk 3 zien hoe metaprogrammeren en modelcheckingtechnologie gebruikt kunnen worden om een DSL voor game mechanics te formaliseren, en om de spelkwaliteit te analyseren en te voorspellen. Vervolgens laten we in Hoofdstuk 4 zien hoe het spelontwikkelingsproces versnelt kan worden en hoe terugkoppeling over de kwaliteit van het spelontwerp verbeterd kan worden door aanpassingen en verbeteringen te maken aan de spelmechanica van spelsoftware die in uitvoering is. Ten slotte laten we in Hoofdstuk 5 zien hoe patronen gebruikt kunnen worden voor het construeren van een interactief ontwerpgereedschap dat statische analyses maakt van game mechanics en ze ook genereert. We hebben dit onderzoek uitgevoerd in voortdurende samenwerking met industriële partners in projecten van toegepast onderzoek naar betere talen en gereedschappen voor geautomatiseerd spelontwerp.

Hoofdstuk 6 gaat in op een algemene uitdaging voor software engineering, die ook een belangrijke uitdaging is voor geautomatiseerd spelontwerp. Live programmeren is een stijl van ontwikkelen die wordt gekenmerkt door incrementele verandering en onmiddellijke terugkoppeling. In plaats van lange bewerk-vertaalcycli, passen ontwikkelaars de broncode van een programma aan terwijl het in uitvoering is, waardoor ze onmiddellijk terugkoppeling krijgen wanneer het zich in reactie daarop direct aanpast. Er is echter nog maar weinig bekend over hoe de voordelen van

live programmeren kunnen worden behaald voor DSLs. Wij laten in een tweedelige oplossing zien hoe generieke taaltechnologie ontwikkeld kan worden voor het construeren van DSLs voor live programmeren. Het eerste deel maakt gebruik van oorsprongstracering en tekstverschillen om *textual model differencing* te realiseren dat op modellen gebaseerde delta's produceert. Het tweede deel laat zien hoe deze op modellen gebaseerde delta's kunnen worden gebruikt bij het ontwerp van DSLs die de runtimestatus van een applicatie migreren middels *run-time model patching*. We beschouwen deze bijdragen als eerste stappen ter ondersteuning van de bouw van "live DSLs" in taalontwikkelingsgeedschappen (language workbenches).

Tot slot rapporteert Hoofdstuk 7 voorlopige resultaten over het verbeteren van gereedschapsondersteuning voor geautomatiseerd ontwerp van gamelevels. Op grammatica's gebaseerde procedurele generatie van gamelevels verhoogt de productiviteit van levelontwerpers voor spellen als *dungeon crawl* and *platform-games*. De verbeterde productiviteit gaat echter ten koste van de borging van de kwaliteit van gamelevels. Het schrijven, verbeteren en onderhouden van grammatica's is moeilijk omdat het lastig te voorspellen is hoe elke grammaticaregel de algehele kwaliteit beïnvloedt, en het ontbreekt aan ondersteuning voor gereedschappen. We pakken het gebrek aan gereedschappen aan door twee nieuwe technieken voor te stellen. De eerste is een nieuwe metriek met de naam Metric of Added Detail (MAD) die aangeeft of een regel details toevoegt of verwijdert met betrekking tot zijn fase in de transformatiepijplijn. De tweede, Specification Analysis Reporting (SAnR) maakt het mogelijk 1) level-eigenschappen uit te drukken in een eenvoudige DSL, en 2) te analyseren hoe kwaliteiten evolueren in geschiedenis van levelgeneratie. De aanpak opent een onderzoeksgebied dat metaprogrammeertechnieken gebruikt om het gebrek aan gereedschappen aan te pakken. Het hoofdstuk is geschreven als een inleidend voorbeeld en heeft als doel studenten te betrekken bij toekomstige toegepaste onderzoeksprojecten om dit werk voort te zetten.

Titles in the IPA Dissertation Series since 2017

M.J. Steindorfer. *Efficient Immutable Collections.* Faculty of Science, UvA. 2017-01

W. Ahmad. *Green Computing: Efficient Energy Management of Multiprocessor Streaming Applications via Model Checking.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-02

D. Guck. *Reliable Systems – Fault tree analysis via Markov reward automata.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-03

H.L. Salunkhe. *Modeling and Buffer Analysis of Real-time Streaming Radio Applications Scheduled on Heterogeneous Multiprocessors.* Faculty of Mathematics and Computer Science, TU/e. 2017-04

A. Krasnova. *Smart invaders of private matters: Privacy of communication on the Internet and in the Internet of Things (IoT).* Faculty of Science, Mathematics and Computer Science, RU. 2017-05

A.D. Mehrabi. *Data Structures for Analyzing Geometric Data.* Faculty of Mathematics and Computer Science, TU/e. 2017-06

D. Landman. *Reverse Engineering Source Code: Empirical Studies of Limitations and Opportunities.* Faculty of Science, UvA. 2017-07

W. Lueks. *Security and Privacy via Cryptography – Having your cake and eating it too.* Faculty of Science, Mathematics and Computer Science, RU. 2017-08

A.M. Şutfi. *Modularity and Reuse of Domain-Specific Languages: an exploration*

with MetaMod. Faculty of Mathematics and Computer Science, TU/e. 2017-09

U. Tikhonova. *Engineering the Dynamic Semantics of Domain Specific Languages.* Faculty of Mathematics and Computer Science, TU/e. 2017-10

Q.W. Bouts. *Geographic Graph Construction and Visualization.* Faculty of Mathematics and Computer Science, TU/e. 2017-11

A. Amighi. *Specification and Verification of Synchronisation Classes in Java: A Practical Approach.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-01

S. Darabi. *Verification of Program Parallelization.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-02

J.R. Salamanca Tellez. *Coequations and Eilenberg-type Correspondences.* Faculty of Science, Mathematics and Computer Science, RU. 2018-03

P. Fiterău-Broştean. *Active Model Learning for the Analysis of Network Protocols.* Faculty of Science, Mathematics and Computer Science, RU. 2018-04

D. Zhang. *From Concurrent State Machines to Reliable Multi-threaded Java Code.* Faculty of Mathematics and Computer Science, TU/e. 2018-05

H. Basold. *Mixed Inductive-Coinductive Reasoning Types, Programs and Logic.* Faculty of Science, Mathematics and Computer Science, RU. 2018-06

A. Lele. *Response Modeling: Model Refinements for Timing Analysis of Runtime Scheduling in Real-time Streaming Systems.* Faculty of Mathematics and Computer Science, TU/e. 2018-07

N. Bezirgiannis. *Abstract Behavioral Specification: unifying modeling and programming.* Faculty of Mathematics and Natural Sciences, UL. 2018-08

M.P. Konzack. *Trajectory Analysis: Bridging Algorithms and Visualization.* Faculty of Mathematics and Computer Science, TU/e. 2018-09

E.J.J. Ruijters. *Zen and the art of railway maintenance: Analysis and optimization of maintenance via fault trees and statistical model checking.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-10

F. Yang. *A Theory of Executability: with a Focus on the Expressivity of Process Calculi.* Faculty of Mathematics and Computer Science, TU/e. 2018-11

L. Swartjes. *Model-based design of baggage handling systems.* Faculty of Mechanical Engineering, TU/e. 2018-12

T.A.E. Ophelders. *Continuous Similarity Measures for Curves and Surfaces.* Faculty of Mathematics and Computer Science, TU/e. 2018-13

M. Talebi. *Scalable Performance Analysis of Wireless Sensor Network.* Faculty of Mathematics and Computer Science, TU/e. 2018-14

R. Kumar. *Truth or Dare: Quantitative security analysis using attack trees.* Faculty

of Electrical Engineering, Mathematics & Computer Science, UT. 2018-15

M.M. Beller. *An Empirical Evaluation of Feedback-Driven Software Development.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2018-16

M. Mehr. *Faster Algorithms for Geometric Clustering and Competitive Facility-Location Problems.* Faculty of Mathematics and Computer Science, TU/e. 2018-17

M. Alizadeh. *Auditing of User Behavior: Identification, Analysis and Understanding of Deviations.* Faculty of Mathematics and Computer Science, TU/e. 2018-18

P.A. Inostroza Valdera. *Structuring Languages as Object-Oriented Libraries.* Faculty of Science, UvA. 2018-19

M. Gerhold. *Choice and Chance - Model-Based Testing of Stochastic Behaviour.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-20

A. Serrano Mena. *Type Error Customization for Embedded Domain-Specific Languages.* Faculty of Science, UU. 2018-21

S.M.J. de Putter. *Verification of Concurrent Systems in a Model-Driven Engineering Workflow.* Faculty of Mathematics and Computer Science, TU/e. 2019-01

S.M. Thaler. *Automation for Information Security using Machine Learning.* Faculty of Mathematics and Computer Science, TU/e. 2019-02

Ö. Babur. *Model Analytics and Management.* Faculty of Mathematics and Computer Science, TU/e. 2019-03

A. Afroozeh and A. Izmaylova. *Practical General Top-down Parsers*. Faculty of Science, UvA. 2019-04

S. Kisfaludi-Bak. *ETH-Tight Algorithms for Geometric Network Problems*. Faculty of Mathematics and Computer Science, TU/e. 2019-05

J. Moerman. *Nominal Techniques and Black Box Testing for Automata Learning*. Faculty of Science, Mathematics and Computer Science, RU. 2019-06

V. Bloemen. *Strong Connectivity and Shortest Paths for Checking Models*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-07

T.H.A. Castermans. *Algorithms for Visualization in Digital Humanities*. Faculty of Mathematics and Computer Science, TU/e. 2019-08

W.M. Sonke. *Algorithms for River Network Analysis*. Faculty of Mathematics and Computer Science, TU/e. 2019-09

J.J.G. Meijer. *Efficient Learning and Analysis of System Behavior*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-10

P.R. Griffioen. *A Unit-Aware Matrix Language and its Application in Control and Auditing*. Faculty of Science, UvA. 2019-11

A.A. Sawant. *The impact of API evolution on API consumers and how this can be affected by API producers and language designers*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2019-12

W.H.M. Oortwijn. *Deductive Techniques for Model-Based Concurrency Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-13

M.A. Cano Grijalba. *Session-Based Concurrency: Between Operational and Declarative Views*. Faculty of Science and Engineering, RUG. 2020-01

T.C. Nägele. *CoHLA: Rapid Co-simulation Construction*. Faculty of Science, Mathematics and Computer Science, RU. 2020-02

R.A. van Rozen. *Languages of Games and Play: Automating Game Design & Enabling Live Programming*. Faculty of Science, UvA. 2020-03