Designing with Autonomous Tools: Video Games, Procedural Generation, and Creativity

Stefan Seidel
University of Liechtenstein, stefan.seidel@uni.li

Nicholas Berente
University of Notre Dame, nberente@nd.edu

John Gibbs
University of Georgia, jkundert@uga.edu

Designing with Autonomous Tools: Video Games, Procedural Generation, and Creativity

Completed Research Paper

Stefan Seidel
University of Liechtenstein
Vaduz, Liechtenstein
stefan.seidel@uni.li

Nicholas Berente
University of Notre Dame
Indiana, USA
nberente@nd.edu

John Gibbs
University of Georgia
Georgia, USA
jkundert@uga.edu

Abstract

Designers are increasingly using autonomous tools to perform complex activity more quickly and in different ways, often with little or no human intervention. These autonomous tools are changing the way many designers approach their work, and the creative outcomes of that work. On the one hand, autonomously generated designs might not provide the same creative user experience as manually designed content, because autonomous tools are fundamentally deterministic in nature. On the other hand, the potentially creative benefits of using autonomous tools relate to the potential variety—that may be perceived as unpredictability by the designers—of their output. We report on a study of three cases of video game development that use procedural generation of content and we theorize about how organizations manage for creative user experiences when using autonomous tools in videogame production. We find that they address the inherent tension between determinism and variety by using a set of modular design practices that help guide autonomous tools in ways that allow for open-endedness and creativity. We refer to these practices as "architectural structuring" and "injecting variety" and we propose that the "level of modularity" as well as the "level of manual design" are critical for creative outcomes. We generate propositions associated with the two practices.

Keywords: autonomous design tools, design, video game production, creativity

Introduction

"I have seen quite a few planets now in the starting galaxy Euclid and they really get boring after a while because the procedural generation is quite the same all the time."

- TheLastTraveller, July 21, 2018, on a No Man's Sky discussion board[1]

Autonomous algorithms are finding their way into many organizational applications, and knowledge-intensive processes such as design work are no exception. *Autonomous design tools* are a class of software tools that generate and modify design artifacts with little or no user intervention (Seidel, Berente, Lindberg, Nickerson, & Lyytinen, 2019), and these tools are typically used in conjunction with human designers (Smelik, Tutenel, de Kraker, & Bidarra, 2010). Autonomous tools are used in a host of applications, from the design of semi-conductor chips (Brown & Linden, 2011) to large scale buildings and architecture (Müller, Wonka, Haegler, Ulmer, & Van Gool, 2006). Even books can be generated by autonomous algorithms.[2] One of the most widespread and mature uses of autonomous design tools is in the production of video games (Grey, 2017; Hendrikx, Meijer, Van Der Velden, & Iosup, 2013; Seidel et al., 2018).

A key promise of autonomous design tools is that they can generate large amounts of content with a certain level of apparently random variation that leads to design outcomes that humans would not necessarily predict. Given this unpredictability, one might expect such tools to foster creativity. Creative outcomes are those that are not only purposeful but also novel (Amabile, 1988; Woodman, Sawyer, & Griffin, 1993). However, there is a fundamental tension with regards to this potential to generate creative outcomes: while these tools may indeed create unpredictable outcomes from the perspective of the designer, they are still deterministic, algorithmic tools (i.e., their results are determined by a previously existing situation such as chosen parameters and other causes). As a result, there is a risk that these tools will generate repetitive, perhaps "boring" (Backus, 2017) and non-creative content. Alternatively, the tools might create a chaos of variety that is wholly unusable for the end product. It is vital for organizations that use autonomous design tools to understand the impact of such tools on creativity. Our research question is:

How do designers navigate the tension between determinism and unpredictability when using autonomous tools for creative purposes?

To begin answering this question, we conducted an exploratory study of video game development that uses autonomous design tools. Video game development provides an exemplar application of leading-edge design with autonomous tools. Moreover, we contend that creativity—novelty and purposefulness—is a key goal in the development of video games as video game developers seek to make—and keep—gamers interested. Through widespread use of technologies such as procedural generation (Shaker, Togelius, & Nelson, 2016; Short & Adams, 2017), the video game industry acts as a sort of laboratory for other industries wishing to understand practices associated with the use of autonomous design tools. Further, anecdotal evidence in the video game industry illustrates the tension between unpredictability and determinism in autonomous game design. For example, in the case of a video game called *No Man's Sky,* a virtually unlimited number of worlds are procedurally generated during game play. Early iterations of the game, however, reportedly suffered from too much uniformity of planets and life forms, due to deterministic nature of algorithmic content generation, resulting in what players said was "boring" gameplay. The designers needed to adjust their practices to better foster creativity in the procedural generation of content to avoid boredom, while at the same time not creating such wild variations of landscape, flora, and fauna that the game became unplayable. The goal of this research is to understand how such designers work with autonomous tools to manage this creativity.

In a first step, we develop a conceptual framework for analyzing the role and use of autonomous tools in design processes. In a second step, we use this model to analyze three quite different cases from video game production. Through our analysis, we identify that designers (1) define architectural constraints within which autonomous tools operate to guide the tools' quasi-random unpredictability in accordance with a particular vision, and (2) deliberately inject variety using both manual and procedural techniques. We thus identify two key practices when using autonomous design tools—***architectural structuring*** and ***injecting variety***—and propose that the ***level of modularity*** as well as the ***level of manual design*** are critical for creative outcomes. We generate propositions associated with the two practices.

[1] https://steamcommunity.com/app/275850/discussions/0/1762482479184633822/, accessed May 1, 2018.
[2] https://aktuelles.uni-frankfurt.de/englisch/first-machine-generated-book-published/, accessed May 1, 2018.

Our paper contributes to the ongoing stream of work investigating how autonomous tools are contributing to major changes in the nature of creative work.

Video Game Development and Procedural Generation

Video games are complex software systems whose success much depends on the quality and quantity of content with which users (gamers) interact (Hendrikx et al., 2013). Video games are highly interactive, dynamic, and content-intensive (Liapis, Yannakakis, & Togelius, 2014). Many contemporary video games provide their users with ever larger virtual environments that cannot be developed with reasonable resources, in a reasonable amount of time, through manual processes alone. Such games require the use of autonomous tools that generate much of the game content (Hendrikx et al., 2013; Seidel et al., 2019; Seidel et al., 2018). Well-known examples include *Star Citizen* (Cloud Imperium Games)*, No Man's Sky* (Hello Games)*, Red Dead Redemption 2* (Rockstar Games)*,* and *Ghost Recon Wildlands* (Ubisoft). In addition, game design platforms like the popular Unity suite of game creation tools have begun integrating advanced AI and procedural terrain and level generation into their platforms. Developments in this area mean that even non-technical developers creating very low budget games can utilize advanced computer-based tools. Approaches that can be subsumed under the notion of autonomous design tools in video game production include *procedural content generation* (Hendrikx et al., 2013; Togelius, Kastbjerg, Schedl, & Yannakakis, 2011; Yannakakis & Togelius, 2015), *procedural modeling* (Müller et al., 2006; Watson et al., 2008), and *computational creativity* (Colton & Wiggins, 2012; Liapis et al., 2014).

Procedural content generation for video games involves the use of software tools to generate game content, to identify interesting instances from those that were generated, and to select useful instances (Togelius et al., 2011). Applications include content drafting, on-the-fly generation of content, and the generation of responsive narratives (Grey, 2017). There is now a large body of knowledge on algorithmic techniques for procedural generation of video game content. Hendrikx et al. (2011), in their review of procedural content generation in games, identify five broad categories of methods: (1) pseudo-random number generation; (2) generative grammars; (3) image filtering; (4) spatial algorithms; and (5) modeling and simulation of complex systems. These are all tactics for automatically generating textures, features, landscapes, and a wide variety of in-game design elements.

The use of procedural content generation allows for producing games at bigger scale with fewer content designers (Seidel et al., 2018; Smelik, Tutenel, Bidarra, & Benes, 2014; Togelius et al., 2011) and is now an important strategy for fostering resource efficiency in video game production. Star Citizen, for instance, is an *open world game,* providing a game space that can be freely explored by players, where this open world includes significant procedurally generated pseudo-random content, such as surfaces of planets and moons, space stations, assets such as asteroids, and even missions/quests. Procedurally generated content is an integral part of the game, as this content is vital for the game to accomplish its vision and goals (Grey, 2017).

Game development projects can be decomposed broadly into the stages of pre-production (ideation, story development, look development, and storyboarding), production (content generation, or asset building, rigging, motion capture, lighting, texturing, special effects, etc.), and testing. While all these stages may involve autonomous design tools, autonomous content generation in particular interfaces with human designers in two ways. First, designers must decide on the tools they use and the specific parameter settings for these tools; second, human designers can (typically) select and sometimes alter the computer-generated outcome (Seidel, Berente, Martinez, Lindberg, Lyytinen, Nickerson 2018).

The involvement of autonomous tools can take place at different levels. On the one hand, games can incorporate autonomous tools to create the game artifact itself. For example, some games rely on procedural generation for their core content—some of these games even design the playing space procedurally at runtime. Other games rely on autonomous capabilities for drafting content, where initial versions of the game space are created through an autonomous tool, but then these drafts are changed by the designer. Some games use autonomous tools in a modal fashion, where they simply add non-critical elements (like backgrounds) to the game. Finally, there is a segmented approach, where only certain game elements are designed using procedural generation (Grey, 2017). What all of these approaches have in common is that their use is always in relation to designer decisions, ranging from designers selecting and adjusting tools, to changing the content generated by these tools, to deliberately using autonomous tools to change their manual design. The outcome of procedural generation is altered by game designers (Hendrikx et al. 2011) in an interactive workflow (Smelik et al., 2010) that typically aims to create a compelling user experience (Seidel et al 2018).

Method

Understanding the organizational practices around autonomous design tools is a new topic area in the context of the changing nature of work. We thus used a qualitative method to analyze game development cases in an exploratory fashion. In this section, we describe (a) the data sources we sampled, (b) the coding scheme we used, and (c) the coding procedures.

Data Collection

We sampled three cases based on their intense use of autonomous tools. We focused on firsthand accounts of game development that are publicly available, as well as reports on game development processes, as our data sources. Video game development is increasingly a public act, since many game developers publicly reflect on their experiences, sometimes even blogging throughout their development. Furthermore, it is in the strategic interest of video game companies to encourage their developers to communicate with gamers throughout the game development process to garner feedback and manage expectations throughout the development process ("generate a buzz," in industry speak). Video games are big business and development organizations need to avoid major missteps that can cost millions of dollars for a game franchise. The games we analyzed—Star Citizen, Sunless Sea, and No Man's Sky—heavily leveraged procedural generation, but in different ways, both in terms of the game type and the size of the project. We thus sampled intense cases for both similarity (use of procedural approaches to generate content) and differences (game type, project type). We looked for and included in our analysis data that we expected to provide (a) information about the general scope and goals of the game and (b) more detailed accounts of how the games were developed—i.e., the designers' practices. We also had access to the games themselves.

Table 1 provides an overview of the data sources we used for our analysis.

Table 1. Data Sources

Case	Data Source #	Data Source	Type
Star Citizen	SC1	***Chris Roberts on Star Citizen's Procedural Planets, Alpha 3.0, & CitizenCon*** https://www.gamersnexus.net/gg/2613-chris-roberts-on-star-citizen-procedural-planets-alpha3-citizencon **Author:** Steve Burke **Published:** September 24, 2016 **Last accessed**: April 13, 2019	Interview
Star Citizen	SC2	**Star Citizen: Around the Verse - Crafting Procedural Moons** https://www.youtube.com/watch?v=DbEKn6gN4Qk **Published:** July 6, 2017 **Last accessed:** April 18, 2019	Video with multiple developers
Star Citizen	SC3	**Building Believable Worlds in Star Citizen** https://www.redbull.com/mea-en/making-star-citizens-planets-believable **Author:** Mike Stubbsy **Published:** April 3, 2017 **Last accessed**: April 14, 2019	Article reporting on interview
Sunless Sea	SUS1	***Sunless Skies Pre-Production: Talkin' Bout Proc Generation*** https://www.failbettergames.com/sunless-skies-pre-production-talkin-bout-proc-generation/ **Author:** Failbetter Games **Published:** November 24, 2016 **Last accessed**: April 18, 2019	Developer blog
Sunless Sea	SUS2	***Aesthetics in Procedural Generation***, *b*ook chapter reporting on the experiences from developing a game, in: Procedural Generation in Game Design, CRC Press **Author:** Liam Welton, Failbetter games,	Book chapter written by developer

Sunless Sea	SUS3	***Of London And The Sunless Sea: Failbetter Interview Pt 2,*** Interview with Alexis Kennedy and Paul Arendt of Failbetter Games https://www.rockpapershotgun.com/2013/09/19/of-london-and-the-sunless-sea-failbetter-interview-pt-2/ **Author:** Adam Smith **Published:** September 19, 2013 **Last accessed:** May 01, 2019	Interview
No Man's Sky	NMS1	**No Man's Sky Developer Sean Murray: 'It was as bad as it can get'** https://www.theguardian.com/games/2018/jul/20/no-mans-sky-next-hello-games-sean-murray-harassment-interview **Author:** Keza MacDonald, **Published:** July 20, 2018 **Last accessed:** April 24, 2019	Article reporting on interview
No Man's Sky	NMS2	***Maths, No Man's Sky, and the Problem with Procedural Generation*** https://www.thumbsticks.com/no-mans-sky-problem-procedural-generation **Author:** Tom Baines **Published:** August 31, 2016 **Last accessed:** April 23, 2019	Game analysis article
No Man's Sky	NMS3	***The Galactic Potential of No Man's Sky*** https://www.redbull.com/us-en/no-mans-sky-interview-and-preview **Author:** Benjamin Kratsch **Published:** March 23, 2017 **Last accessed:** April 24, 2019	Article reporting on interview
No Man's Sky	NMS4	***No Man's Sky—Procedural Content*** http://3dgamedevblog.com/?m=201610 **Author:** GregkWaste **Published:** October 17, 2016 **Last accessed:** April 23, 2019	Developer blog
No Man's Sky	NMS5	***World Without End: Creating a Full-scale Digital Cosmos*** https://www.newyorker.com/magazine/2015/05/18/world-without-end-raffi-khatchadourian **Author:** Raffi Khatchadourian **Published:** May 11, 2015 **Last accessed:** April 24, 2019	Article reporting on interview

Analytical Framework

We use a simple analytical framework that sensitizes our analysis and allows for exploration and emergence. Our analysis is thus based on inductive and abductive reasoning. This coding scheme comprises of four broad categories that were derived from our research interest in how the use of autonomous tools changes design work **(process: design practices)** as designers seek to navigate the tension between determinism and unpredictability **(challenges)** when using autonomous tools for creative purposes **(goals, outcomes)**.

Goals. First, we are interested in the general goals that are pursued when using autonomous tools. As explained earlier, these may range from the procedural generation of core content to the modal use of autonomous design tools, where they simply add non-critical elements to the game. Generally, autonomous tools bear the potential of generating large amounts of content that could not be generated through manual design given typical resource constraints (Hendrikx et al. 2011).

Challenges. Second, we are interested in the challenges that are encountered at the outset, and throughout the game creation project, as they relate to the use of autonomous tools. For instance, early in our analysis it became noticeable that game studios might turn to autonomous design tools because they want to develop large games with (comparably) small team sizes and low budgets. Challenges are thus a key reason for game designers to use autonomous tools, and challenges (limited resources) and goals (developing much content) are hence closely related.

Process: design practices. Third, we are interested in the specific design practices that are applied in order to use autonomous tools throughout the different stages, i.e., in pre-production, production, and testing of video games, particularly to navigate the tension between the determinism and simultaneous unpredictability of autonomous design tools. It has been asserted that using autonomous tools requires different workflows (Smelik et al., 2010) and design practices in general terms (Seidel et al., 2018).

Outcomes. Finally, we aim to understand the impact of autonomous design tools on the outcome—the game itself. Here, we are particularly interested in the extent to which autonomous design tools in the observed cases led to creative (i.e., novel and purposeful) results, considering the tension between determinism and unpredictability associated with autonomous design tools. In the case of video game content, this creativity is reflected by the variety of content that is generated through manual design and design through autonomous tools. Unpredictability can be associated with novelty, which is a key element of creativity, next to purposefulness (Amabile, 1996; Woodman et al., 1993).

Coding Procedures

Our analysis and construction of our explanation proceeded in two key steps that were guided by the four broad categories—goals, challenges, practices, and outcomes—which served as a sensitizing device (Klein & Myers, 1999). In a first step, by reading an initial dataset, we identified key themes—these revolved around the understanding that (1) game development using autonomous tools always included some form of architectural vision within which these tools were used and (2) that manual design in many cases still plays an important role. Based on this initial analysis of three games, we were able to construct three different stories of how these games used autonomous tools. Around these themes, first concepts emerged, most notably *architectural vision* and *injecting creativity* as key practices. Each of the three authors did this for one case, but we then discussed the emergent themes to seek consensus about what we found in the data. This initial process helped us understand the phenomenon of interest in much more depth. In all three cases we found evidence for the existence of an architectural vision and also for the specific role of human designers and their creativity.

In a second step, in order to theoretically sample (Glaser & Strauss, 1967), we collected additional data for each of the cases and moved into a formal coding process. Note that our initial analysis had already produced first concepts, so our theoretical sampling was indeed guided by emergent concepts and not only by mere themes, which is key to theoretical sampling (Charmaz, 2006). We thus included additional data sources that we expected would help us learn more about the architectural vision and injection of creativity. In the spirit of open coding, we started to develop a list of open codes—now sensitized not only by the broad categories of goals, challenges, process, and outcomes, but also the two categories of architectural structuring and injecting variety. Still, we aimed to remain open in this stage and treated these emergent categories as sensitizing, in order to find what else was important in the data (Urquhart & Fernández, 2013). We then, in the spirit of axial coding (Strauss & Corbin, 1998), started to group the emergent concepts and identify relationships among them—that is, we coded around the 'axis' of these concepts. An example result from this stage is that we were able to learn about how "content variety" is grounded in the "level of modularity," which in turn depends on key architectural choices that are made by the design team.

Finally, in the spirit of selective coding (Strauss & Corbin, 1998), we integrated the emergent concepts into propositions that explain how designers interact with autonomous tools to foster creative outcomes. This process of open, axial, and selective coding was iterative, and coding and data collection overlapped. The result of this procedure is the identification of a set of key practices organizations apply to manage the tension between determinism and unpredictability of autonomous tools when working to accomplish creativity in video game production, and propositions related to these practices and their outcome.

Case Analysis

The Sunless Sea Story

In 2015, Failbetter Games released Sunless Sea, a virtual-world role-playing game involving sea exploration. They developed the game with limited resources, funded by a Kickstarter campaign. Developers leveraged procedural generation techniques to generate large portions of the game's world (see Table 2 for a summary of the results from our case analysis).

Table 2. Sunless Sea—Key Findings Related to the Use of Autonomous Design Tools

Category	**Key Findings**
Goals	● Developing a game that allowed users to explore ● Developing a game based on relatively little funding (Kickstarter campaign)
Challenges	● Uniformity ● Errors through manually crafted content
Design Practices	● Architectural design ○ Definition of design rules in terms of modular operators ○ Procedural generation based on these design rules ● Manual design to inject variety was error-prone
Creative Outcome	● Variation that kept the game challenging for players

In Sunless Sea, the goal for the game play involved traveling from familiar land to explore the unknown sea in the world under conditions of darkness. The developers thus aimed to create a user experience based on the feeling of exploration:

> "We wanted the world to provide both a challenge and a sense of threat, and didn't want these to be lost as the player became familiar with the setting." **(Source SUS2)**

To accomplish this, the developers used procedural generation to regenerate large portions of the map each time the game was started in a way that was always new.

A key challenge involved balancing the freshness of each iteration of gameplay with coherence between gaming sessions. The developers wanted to avoid totally arbitrary gameplay (chaotic variety) by including consistent, familiar elements across game sessions. They accomplished this through a square grid-and-tile design for the world. A "L" shaped grid of stable, manually created elements were consistently part of the world, and framed the world on two sides (the open side of the "L" was open-ended, but did include an island grid). Procedurally generated grid tiles were merged according to categories (i.e., features, difficulty, themes) and the characteristics generated influenced the generation of the adjacent tiles to maintain continuity in the experience.

A major challenge in development involved navigating the grid size. Although the developers began with a relatively large 18x18 grid design, they found that the team was more comfortable designing in a smaller, 3x3 design space and could better maintain continuity in this smaller space. They eventually settled on a 6x6 grid with larger tiles as a compromise solution (Figure 1, left). This decision limited the differential components of the game, while at the same time enabling developers to make the components that they did more engaging. These changes reduced the logic needed to procedurally inform adjacent tiles, but increased requirements for content within a tile.

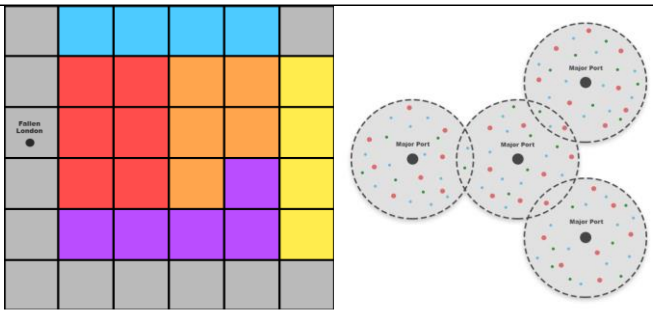

Figure 1. Approaches to Game Layout for Procedural Generation: Tile & Grid (left, Sunless Sea—gray grids unvarying) **versus Adjacent Regions Approach** (right, Sunless Skies—major ports are unvarying)
(source: failbettergames.com)

The team thus managed to balance the freshness of new experiences with stability of gameplay:

> "In the end, the tile-based approach we took to procedural generation in Sunless Sea was successful. It enabled enough variation to keep the challenge fresh for players, while allowing us to retain sufficient aesthetic control to craft the experience we wanted." **(Source: SUS2)**

However, developers were concerned that there was too much uniformity and not enough novelty in the game experience. They addressed this in two ways. First, because of the structured nature of the design rules intended to keep consistency in game play, the developers found themselves manually attempting to inject variety into the game play. This, however, was prone to error as well as inefficient. The developers addressed this shortcoming by incorporating a procedural logic to inform the game how to inject variety into the procedurally generated content. They incorporated this new system into an underwater extension to the game in 2016 (called *Zubmariner).*

Second, they realized that the grid arrangement limited novelty in the procedurally generated content. Essentially, the trip back from uncharted areas to known areas was boring, as it repeated the outgoing voyage's challenges. They needed to find a way to inject variety into the gameplay in both directions, yet still maintain some stability. In the sequel to the game, "Sunless Skies," developers opted for putting stable "ports" in overlapping regions. In this configuration, the overlapping regions can be explored in a greater variety of ways, offering more flexibility than did the earlier grid. Ports offer the stability and consistency required for gameplay, but now players can approach the ports in a variety of ways, enabling a richer experience (Figure 1, right).

Procedural generation enabled Failbetter Games to design a game in a novel way, using relatively small Kickstarter funds. The game was a success commercially. The first thing that needed to be addressed was the granularity of the design rules—this granularity had implications on the locus of creativity (adjacent versus within tiles) that impacted gameplay. One implication involved limited creativity in the game, since procedurally generated content lacked some novelty, partially because developers were focusing on consistent gameplay in their design rules. As developers indicated:

> "while every discovery should feel exciting and fresh, it should also feel natural to the player in hindsight. Discoveries needed to build on the expectations players would have developed from exploring areas adjacent to them. Our greatest fear when designing our procedural generation process was that we would expose the players to tonally jarring contrasts that would throw them out of the world…" **(Source SUS2)**

The No Man's Sky Story

In 2016, Hello Games released No Man's Sky, a mass-scale universe exploration game. The game consists of a vast universe of planets (18 quintillion plus, according to the developers) that can be explored and exploited for resources. Gamers fly space ships between planets, and are free to explore the ones they land on at their leisure. Though there are potentially aggressive creatures, and gamers must exploit planetary resources to garner the energy needed to get to other planets, the game is not a classic "shoot-em-up" space adventure. Rather, many users refer to it as a "chill game" (**Source NMS1**). Since the primary intent of No Man's Sky is exploration, and there are millions of planets in this virtual world, procedural generation was a requisite for creating the game.

The game "began life as some lines of code on [Sean] Murray's computer" (**Source NMS1**). Murray, originator of the game, in fact worked alone for a year on base code for the game before showing a tech demo and getting funding from Sony to complete the game for Playstation and PC. For the five years of development, the average team size was only six people, less than a tenth the size of a normal project team for this sized game (**Source NMS1**), so the bulk of generative work had to be performed by procedural tools. Table 3 gives an overview of the key findings from our analysis.

Table 3. No Man's Sky—Key Findings Related to the Use of Autonomous Design Tools

Category	Key Findings
Goals	● Scope: developing 18 quintillion unique worlds and biomes ● Speed: develop the game rapidly ● Runtime generation of varied but reproducible elements ● Exploration with large number of interesting planets
Challenges	● Very low budget ● Very small team (1 to 6 people)

	● How to generate many planets with interesting things on most of them ● How to find variety in the procedural code
Design Practices	● Artist-driven concept (the "look") ● Highly procedural-centric ○ Build the code, then let it run ○ Create drones to explore planets and return pictures ○ Tweak code to improve variety, excitement, and stability ● For animals, create base creature with all parts, then procedurally choose body parts and size of parts ● For plants and biomes, use biology simulation: L-Systems ● Music procedurally samples from a vast music library at run-time, based on the game state ● Co-evolution of game design and tools
Creative outcome	● Nearly infinite universe of planets (18,000,000,000,000,000,000+) ● Each planet is unique ● Flora and fauna can take on vast number of shapes and sizes ● Music for the game is procedurally generated based on current game state

With a small development team and such ambitious universe creation goals, procedural generation of everything—solar systems, planets, terrain, plants, animals, and even music—was in the design from the very beginning. Murray notes that the planet terrain generator is only 1,400 lines of code, and that the code was tweaked to ensure that the planets it generated were not only visually interesting but traversable by a gamer. Early builds of the game produced many stunning planets that were non-navigable, while some tweaks caused all the planets to be dull and unvaried (**Source NMS5**). Utilizing a procedural engine to generate the content, then having humans examine the content for compliance with their vision, then tweaking the procedural code to better match their vision, was the primary mode of asset development for the game.

One fascinating aspect of No Man's Sky is that even Sean Murray, originator of the game and creator of the code, does not know what lies behind every corner of his game.

> "It's rare that a game can surprise its creator, but such is the nature of the No Man's Sky universe, which is procedurally generated and uncovered as players traverse it" (**Source NMS3**).

The team, in fact, had to invent virtual drones that can go visit sample planets and take 'selfies' while there—pictures which the team will then review to ensure that planets near each other, for example, have a variety of landscapes and creatures so there is a "little bit of variety" in each solar system (**Source NMS 3**). When creating a build of the game, the engine drops resources in a spot in space (these must be exploited in order for the gamer to move on), then figures out the aspects of the planet, from terrain to flora and fauna, based on the resources at that spot (**Source NMS3**). A primary theme with the developers is clearly the tension concerning variety versus accessibility: too much variety and the game becomes impossible to play; too little variety and the game—vast as it is—is boring.

Concomitant with the vast scope of the game is the small installation size. The game installation is around 6GB, most of which is music files, and this small amount of data creates the large number of planets, and all that is on them (**Source NMS2**). A solipsistic artifact of the game's algorithmic nature is that nothing exists until one arrives at that point in space. The game uses a smallish set of assets, created by humans, to create hundreds or thousands of variations of everything (**Source NMS4**).

> "Because all the necessary visual information in the game is described by formulas, nothing needs to be rendered graphically until a player encounters it.... [T]he game continuously identifies a player's location, and then renders only what is visible. Turn away from a mountain, an antelope, a star system, and it will vanish just as quickly as it appeared" (**Source NMS5**).

This economy is available to the game due to the deterministic nature of a game build: each point in space is created procedurally, but with the same seed number and the same code, it will regenerate the exact same thing at each point every time. Thus variety and nearly infinite possibility for exploration exists, but this is contained within formulae that will dependably reproduce the same elements each time.

The life-forms for No Man's Sky are also produced procedurally, but in a somewhat different manner. Life is created using a version of the famous bio-mathematical L-Systems, originally developed by Aristid Lindenmayer in the late 1960s, and used by biologists and others to mimic everything from how life is created to how complex ecosystems behave. The development team first created a number of life-form elements (e.g., heads, backs, legs, tails, etc. for an animal) for a given life-form type. Figure 2 shows an example of this "total mess" of an elemental creature (**Source NMS4**). The procedural engine then algorithmically selects body parts that can go together, selects a size, and selects a texture for the creature. Thus animals and vegetative matter are created procedurally from a relatively small set of hand-created parts. Again, during game play, each of these creatures is created, assembled, and rendered just-in-time as the player approaches it—a technical feat that is fairly astounding.

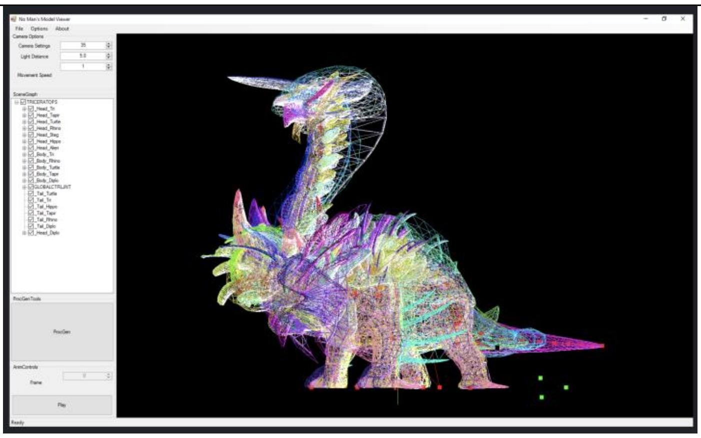

Figure 2. Multiple possible bodies, displayed at once, for a creature in No Man's Sky
(source: http://3dgamedevblog.com/?m=201610)

The Star Citizen Story

Star Citizen, in development by Cloud Imperium Games at the time of writing this paper, is a game where players can move freely in a virtual open world of star systems, including planets, moons, and space stations. Players can use different vehicles such as various space ships for their explorations. This open world includes significant procedurally generated pseudo-random content, such as the surfaces of planets and moons, space stations, assets such as asteroids, and even missions/quests. Procedurally generated content is an integral part of the game—this content is vital for the game to accomplish its vision and goals (Grey, 2017). Star Citizen got its initial funding through a Kickstarter campaign that started in 2012 and collected more than two million USD. While stretching the initial goals multiple times, the game's crowdfunding continued, including the sale of virtual space ships and even concept sales of digital artifacts. In April 2019 the total funding exceeded 220 million USD.[3] Table 4 provides an overview of key findings from our analysis.

Table 4. Star Citizen—Key Findings Related to the Use of Autonomous Design Tools

Category	Key Findings
Goals	● Scope: developing extremely large amounts of content and vast game worlds ● Speed: developing contents quickly ● Build time content generation ● Exploration, fresh experience, excitement
Challenges	● Kickstarter funded project; uncertainty ● Stretch goals ● Generating large amounts of content, but providing variety
Design practices	● Artist-driven design ● Confluence of procedural and manual design ○ Manual design of key structure at different levels (planets, biomes, etc.)

[3] https://robertsspaceindustries.com/funding-goals, accessed 2019-04-18

	○ Tools fill in key areas, e.g., using brushes ○ Artists craft specific art; to do so, they carve out areas ● Co-evolution of game design and tools
Creative outcome	● Extremely large game spaces ● Content variety

The game's initiator, Chris Roberts, describes the goals of the project as developing a universe, which requires extremely large amounts of content. The key goals of using procedural generation at Star Citizen involve allowing designers and artists to generate game worlds at scale, and rapidly generating the content at build time—but always in conjunction with manual design:

> "We are going to use it [procedural generation] as a tool for universe building. I know a lot of people think Star Citizen is purely hand-crafted... but the reality is that all of these games have a mix of hand-crafted and procedural stuff in them." **(Source SC1)**

This process involves using procedural generation in sync with authored content to provide a large content scope and an exciting, fresh experience through the procedural generation of stories (missions/ quests).

The team started with simple tools that co-evolved as the game design evolved. Initially, the procedural generation of planets was a mere idea that would be implemented only after the initial project launch, being available with the Alpha 3.0 version. Version 1 (internally called V1) of the procedural tools was an early, rudimentary version. Version 2 (V2) allowed for content generation at planetary scale. As the project developed, it became clear that the team needed advanced tools for procedural generation to quickly generate entire planets. Procedurally generated planets were thus one of the so-called stretch goals. Chris Roberts described how the procedural generation of planets or space stations takes place at build time, the tools providing constitutive elements of the game **(Source SC 3)**. Designers start with a sphere and a height map for the terrain of the planet. Then they add a distribution map to describe where biomes (forests, mountains, deserts, etc.) are located. Designers manually paint some features and let procedural tools generate the rest (Figure 3 shows a procedurally generated city).

There are two important observations to be made. First, certain architectural decisions are made (and architectural rules are applied), and the procedural tools are used within the constraints of these architectural decisions. Second, there is a significant amount of manual activity, i.e., "hand crafting" involved. Overall, the approach is described as "artist-driven" by Chris Roberts:

> "[Procedural planets V2] is a lot more artist authored and driven... We use these techniques to allow artists to build the world out at scale, but they're determining where the continents are, where the forests are, where the mountains are, where the desert plains are." **(Source SC1)**

Planets in the game have surfaces of millions of square kilometers (as opposed to, for instance four or so square kilometers for maps in conventional games). Users can freely navigate and explore this game space in a way that would not be possible without the use of procedural tools (see Figure 3).

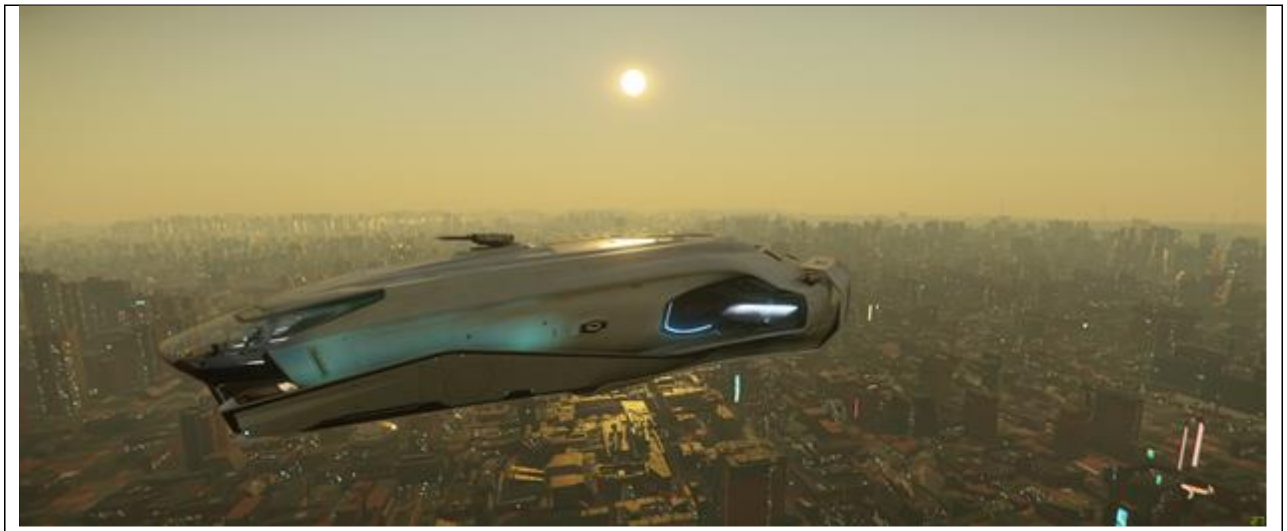

Figure 3. Procedurally Generated City in Star Citizen
(source: screen shot from Star Citizen Version 3.3.7-LIVE-1007767)

Discussion

The analysis of the three cases highlights how designing with autonomous tools leads to a rethinking of formerly held notions about the process of game design. While the three cases differ in their specific goals, the nature of the games developed, and the way they used procedural generation, there are certain commonalities: (a) In all three cases, organizations confronted new types of unpredictability related to the use of autonomous tools; (b) in the three examples, this process was associated with certain risks (Grey, 2017) and involved learning how to best use the tools; (c) in the each case, a relatively small team aimed to design an unusually large game space for the size of the design team.

From the perspective of the design practices in particular, we observe two key commonalities: in all cases the developing teams (a) structured a particular architecture to shape the bounds of the procedural tools; and (b) used manual designs in conjunction with procedural algorithms to produce a controlled amount of variety for a rich user experience. Still, the specific strategies that were followed in the three games are different, and we can thus identify a set of more detailed design practices. We discuss these two sets of key design practices when using autonomous tools in what follows, and then suggest key propositions related to these two sets of design principles.

Architectural Structuring

Our analysis suggests that the design practices applied in the three cases relate to capitalizing on the potential of autonomous design to generate large amounts of content with certain levels of unpredictability. However, to benefit from this unpredictability, the organizations used certain strategies to guide unpredictability in order to align it with other design activities, such as those carried out by human designers, as well as to implement a specific game idea or vision (Table 5). Broad architectural choices create a sense of gameplay stability and allow game designers to convey their broad vision to the consumer. Still, these choices might also lead to limited novelty in the procedurally generated content. The level at which the game architecture is defined by human designers is highly variable.

Structuring for coherence. In all three cases, designers made architectural choices that would ensure coherence—coherence among the various planets (Star Citizen, No Man's Sky) and coherence among each restart of the game (No Man's Sky, Sunless Sea) to create consistent user experience. To this end, certain design rules were defined, such as the arrangement of tiles in Sunless Sea.

Structuring for navigating granularity. In the case of Star Citizen, designers navigated different levels of granularity, manually defining certain elements at high resolution, then allowing procedural tools to deal with all other elements at lower resolution, using modular building blocks. This approach enabled them to navigate different levels of procedural generation and thus have creative control over widely different levels of granularity of the project.

Structuring for novelty. In Sunless Sea, the arrangement of grids allowed implementing a certain level of gameplay stability. At the same time, this move hindered the creation of novel experiences through the use of procedural generation; the results created a repetitive experience for the user. However, an alternative architectural vision mitigated this effect, highlighting the importance of viewing game architecture and procedural generation in context. Sunless Sea involved determining the bounds of procedurally generated content in different ways, and the strategies they implemented had much different impacts on creative outcomes.

Structuring for procedural completion. Overall, each of the three games managed procedural generation by creating limits, or bounds on the elements to be created on autonomous tools. Procedural generation takes place within the constraints of the game's architecture and architectural decisions clearly matter to creativity. This involves a balance between the granularity of components, and the degree to which components are specific.

Essentially, the organizations follow different approaches to modularity in their architectures (Baldwin & Clark, 2003) to manage creativity in the projects, while at the same time leveraging autonomous tools.

Table 5. Architecting Design Practices

Design Practice	**Creative Outcome**	**Examples**
Structuring for coherence • Defining rules that create a sense of consistency	Combing content variety with consistent user experience	In Sunless Sea, similarities between the worlds generated at game start allows users to become more experienced in playing the game. In No Man's Sky, spatial coordinates and time dictate what exists at each point, and at which point animals and vegetation will be in their life-cycle.
Structuring for procedural completion: • Manually design basis • Allow tools to complete design	Scale magnitude of work Greater efficiency of creative talent	In Star Citizen, broad areas (up to the level of planets) are defined and then filled procedurally. In No Man's Sky, animals are built from a set of proto-elements that are procedurally selected and put together.
Structuring for navigating granularity: • Manually design different elements of the game environment at different resolutions, different levels of abstraction • Tools fill in the gaps with low resolution content, and modular assembly of content	Scale magnitude of work and greater efficiency of creative talent More flexibility in designing general or specific game elements	In Star Citizen, the map architecture involves different levels of granularity, and procedural tools are used at these different levels of granularity.
Structuring for novelty: • Arranging modular operators and interfaces in such a way that procedural tools maintain clear limits, but arranged in a way where these limits effect the design less	More novelty in procedurally generated content	In Sunless Skies, the map architecture was adjusted based on the experiences from Sunless Sea is in such way that it fosters content variety.

Our analysis suggests that these strategies are not mutually exclusive but can be combined. In Star Citizen, for instance, designers (a) decide on what should be generated procedurally (structuring for procedural completion), (b) at what level (e.g., planets, villages—structuring for navigating granularity), and (c) to what extent this is then changed manually in order to inject variety—the design practice to which we turn next.

Injecting Variety

Not only do autonomous tools operate within the boundaries of a specified architectural vision, designers also interact with the generated outcomes to create rich experiences. Manual intervention helps to meet architectural requirements (Smelik et al., 2010) and to introduce new sources of variety (Seidel et al., 2018) into gameplay. We identified two key approaches.

Procedural completion. In No Man's Sky, for example, a number of assets that were created manually (including textures etc.), are procedurally combined at runtime to generate worlds. The final product—in fact, the entire game—is generated procedurally.

Manual completion. In Star Citizen, for example, artists created game elements such as ruins or mine shafts by carving out specific areas on planets, and then placing their specific art within that area. The final product was thus generated manually, based on procedurally generated content. Table 6 shows how (human) variety can be injected into procedurally generated content.

Table 6. Injecting Variety Design Practices

Design Practice	**Creative Outcome**	**Examples**
Procedural completion • Designers initiate designs that are instantiated procedurally	Ability to generate large design artifacts where manual generation provides the basis	New game space for every restart in Sunless Sea; large portions of the game space are generated each time the game is started. Generation of planets, including landscape, flora and fauna, at runtime in No Man's Sky.
Manual completion • Designers create and place game elements manually based on procedurally generated assets	Ability to generate large design artifacts where procedural generation provides the basic building blocks	Immersive experience in specific areas of the game space in Star Citizen; designers created game elements such as ruins or mine shafts by carving out specific areas on planets, and then placing their specific art within that area.

Propositions: Level of Modularity and Level of Manual Design

Clearly each of our cases attests to the need for a combination of manual design activity and procedural generation (Schneider, Boldte, & Westermann, 2006; Smelik et al., 2010), but our key questions involved how the two combine to generate creativity. The strategies related to architecture and injecting variety essentially address two key issues: (1) key architectural choices determine the level of modularity; (2) decisions about how variety is injected determine the level of manual design. Together, the level of modularity and the level of manual design impact content variety, which is a proxy of creativity that is perceived by the users. From these two strategies, we can infer two key insights.

The first insight, around the architecture, and thus the architecting design practices, involves the **level of modularity** in the architecture. Clearly each of the cases determined the appropriate level of granularity for the in-game modules based on the creative needs for gameplay. Star Citizen used a layering strategy to quickly generate planets at scale, where each planet was a generated unit, and where each planet is composed of biomes, which then contain mountains etc. Sunless Sea used a grid for generation, and the Sunless Sea example vividly shows that this level is critical to the right amount of creativity as they moved from an 18x18 grid to a 3x3 to finally settle on 6x6 to balance designer attention with variety in the gameplay. Finally, the designers of No Man's Sky had to balance the scale at which procedural generation could run: too little granularity and the worlds are too uniform; too much and they are unplayably chaotic. The granularity of the modules, resulting from the application of the key architecting design practices, is critical to maintaining creativity and at the same time economizing designer resources, leading to our first proposition.

> ***Proposition 1:*** *More (less) granularity in the modular architecture for procedural generation will result in greater (less) content variety.*

Our analysis highlights how the use of autonomous tools in video game design is embedded in, and thus constrained by, a game's overall architecture, which reflects the designers' broader architectural vision. Procedural generation—which is inherently deterministic but should still produce outcomes that are unpredictable from the designer's perspective—is guided by this architecture, and the overall outcome depends on how procedural generation is guided, or constrained, by that architecture. Moreover, since procedural generation is inherently deterministic, the interaction of manual design steps and procedural generation are essential as game developers aim to create variety.

The second key insight relates to the way creativity is injected into the design artifact. Here, the key is the **level of manual design.** Our data suggests that the level of manual design is high when a procedurally generated design artifact is manually completed or changed. The opposite is that the level of manual design is lower when pre-built assets that are manually designed are then procedurally combined. While in the first case determinism is followed by human ingenuity, in the second case, human ingenuity is followed by deterministic recombination.

In all three cases, designers manually created critical components of the games—those components where gamers were likely to spend more time and scrutinize more; those components that were critical to game play. In Star Citizen, for example, we can see a combination of the two modes. First, designers make architectural choices to generate large amounts of content (procedural completion—and manual initiation), but this content then provides the basis for manual interventions (manual completion—and procedural initiation) as designers go on to identify and alter important areas in the procedurally generated game space. In No Man's Sky, for example, where the game space is generated at runtime, designers focused attention on handcrafting key static elements, such as the playable characters as well as elemental assets that are then re-combined to create vegetation and fauna at runtime—a case of manually initiated, but procedurally completed content generation. That is, on the one hand designers can design key components and let the procedural generation finish the design, resulting in more efficient leveraging of designer resources, but potentially with less interesting components. On the other hand, designers can manually complete what algorithms produce. This observation leads to our second proposition:

> ***Proposition 2:*** *Manually completed components of design are likely to show more content variety than procedurally completed components of design.*

These propositions point out how decisions associated with architecture and the way designers choose to inject variety have implications for both the variety and associated apparent creativity of the resulting artifact, but do so at the cost of the scope of the design that designers can address.

Conclusion

Digital technologies with autonomous capabilities, fueled by advancements in algorithms, artificial intelligence, and related methods, play a key role in the changing nature of human work (Daugherty & Wilson, 2018; Seidel et al., 2019). It is clear that even traditionally human-centric practices such as design—knowledge-based activities aimed at constructing artifacts and typically involving creativity (Dorst & Cross, 2001)—change as autonomous tools are used (Seidel et al., 2019).

This paper presents how game development processes are structured as organizations seek to capitalize on the affordances provided by autonomous design tools—tools that are based on algorithms and thus deterministic, but at the same time capable of creating content of scale, complexity, and variety that is impossible for humans to achieve within any reasonable time frame, and that *appears* to be unpredictable to designers and users. Two key sets of practices allow game designers to navigate this tension: architectural structuring and injecting variety. These two categories of design practices can be broadly subsumed under what seems to be the current paradigm of game development when using autonomous design tools: artist-driven procedural generation, which involves the combination of manual and procedural activities. Procedural generation is an incredible way to efficiently generate a lot of design content, but in order to ensure creativity, the architectural decisions and the process through which designers inject variety into a game will have implications on both the realized design creativity, and on the scope of the design. In this paper we take a step toward theorizing about these tensions.

Our study has some limitations. First, our data sample only considers cases from the substantive area of video game development. Second, within this substantive area, we investigated three development projects that only cover a small part of the video game development landscape (e.g., our sample did not involve a "Triple-A" game developed by a major studio and two of the games are crowdfunded; also, these games used procedural generation in specific ways). However, in order to allow for applying our ideas in different contexts, we aimed to move towards formal concepts in terms of the key practices we identified and we developed two key propositions. To what extent the suggested concepts are applicable in other contexts where autonomous design tools are used will be subject to future empirical work. Finally, our study has focused on creativity from the perspective of designers and their design practices, and we have argued that content-variety is a proxy for what will be perceived by users—it will thus be worthwhile to also explore how the different outcomes are indeed perceived by users.

References

Amabile, T. M. (1988). A Model of Creativity and Innovation in Organizations. In B. M. Staw & L. L. Cummings (Eds.), *Research in Organizational Behavior* (Vol. 10, pp. 123-167). Greenwich, CT: JAI Press.

Amabile, T. M. (1996). *Creativity in Context*: Westview press.

Backus, K. (2017). Managing output: Boredom versus chaos. In *Procedural Generation in Game Design* (pp. 13-21): AK Peters/CRC Press.

Baldwin, C. Y., & Clark, K. B. (2003). Managing in an Age of Modularity. *Managing in the Modular Age: Architectures, Networks, and Organizations, 149*, 84-93.

Brown, C., & Linden, G. (2011). *Chips and Change: How Crisis Reshapes the Semiconductor Industry*: MIT Press.

Charmaz, K. (2006). *Constructing Grounded Theory: A Practical Guide through Qualitative Analysis*. Thousand Oaks, CA: Sage.

Colton, S., & Wiggins, G. A. (2012). Computational Creativity: The Final Frontier*?* Paper presented at the *ECAI*.

Daugherty, P. R., & Wilson, H. J. (2018). *Human + Machine: Reimagining Work in the Age of AI*: Harvard Business Press.

Dorst, K., & Cross, N. (2001). Creativity in the Design Process: Co-evolution of Problem-Solution. *Design Studies, 22*(5), 425–437.

Glaser, B. G., & Strauss, A. L. (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago, IL: Aldine Publishing Company.

Grey, D. (2017). When and Why to Use Procedural Generation. In *Procedural Generation in Game Design* (pp. 3-12): AK Peters/CRC Press.

Hendrikx, M., Meijer, S., Van Der Velden, J., & Iosup, A. (2013). Procedural Content Generation for Games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 9*(1), 1.

Klein, H. K., & Myers, M. D. (1999). A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly, 23*(1), 67-93.

Liapis, A., Yannakakis, G. N., & Togelius, J. (2014). *Computational Game Creativity.* Paper presented at the ICCC.

Müller, P., Wonka, P., Haegler, S., Ulmer, A., & Van Gool, L. (2006). Procedural Modeling of Buildings. *ACM Transactions On Graphics (Tog), 25*(3), 614-623.

Schneider, J., Boldte, T., & Westermann, R. (2006). Real-time Editing, Synthesis, and Rendering of Infinite Landscapes on GPUs. In: *Vision, Modeling and Visualization.*

Seidel, S., Berente, N., Lindberg, A., Nickerson, J. V., & Lyytinen, K. (2019). Autonomous Tools & Design Work: A Triple-Loop Approach to Human-Machine Learning. *Communications of the ACM, 62*(1), 50-57.

Seidel, S., Berente, N., Martinez, B., Lindberg, A., Lyytinen, K., & Nickerson, J. V. (2018). Succeeding with Autonomous Tools in Systems Design: Reflective Practice & Ubisoft's Ghost Recon Wildlands Project. *IEEE Computer, 51*(10), 16-23.

Shaker, N., Togelius, J., & Nelson, M. J. (2016). *Procedural Content Generation in Games*: Springer.

Short, T. X., & Adams, T. (2017). *Procedural Generation in Game Design*: AK Peters/CRC Press.

Smelik, R. M., Tutenel, T., Bidarra, R., & Benes, B. (2014). A Survey on Procedural Modelling for Virtual Worlds. Paper presented at the *Computer Graphics Forum.*

Smelik, R. M., Tutenel, T., de Kraker, K. J., & Bidarra, R. (2010). Integrating Procedural Generation and Manual Editing of Virtual Worlds. Paper presented at the *2010 Workshop on Procedural Content Generation in Games*.

Strauss, A. L., & Corbin, J. (1998). *Basics of Qualitative Research. Techniques and Procedures for Developing Grounded Theory* (2nd ed.). London, UK: Sage.

Togelius, J., Kastbjerg, E., Schedl, D., & Yannakakis, G. N. (2011). What is Procedural Content Generation?: Mario on the Borderline. Paper presented at the *2nd International Workshop on Procedural Content Generation in Games*.

Urquhart, C., & Fernández, W. (2013). Using Grounded Theory Method in Information Systems: The Researcher as Blank Slate and other Myths. *Journal of Information Technology, 28*(3), 224-236.

Watson, B., Müller, P., Veryovka, O., Fuller, A., Wonka, P., & Sexton, C. (2008). Procedural Urban Modeling in Practice. *IEEE Computer Graphipcs and Applications, 28*(3), 18-26.

Woodman, R. W., Sawyer, J. E., & Griffin, R. W. (1993). Toward a Theory of Organizational Creativity. *Academy of Management Review, 18*(2), 293-321.

Yannakakis, G. N., & Togelius, J. (2015). Experience-driven Procedural Content Generation. Paper presented at the *International Confernce on Affective Computing and Intelligent Interaction (ACII), 2015*.