CONTROLLING RANDOMNESS:
USING PROCEDURAL GENERATION
TO INFLUENCE PLAYER UNCERTAINTY IN VIDEO GAMES

by

TRAVIS L. FORT

A thesis submitted in partial fulfillment of the requirements
for the Honors in the Major Program in Digital Media
in The School of Visual Arts and Design
and in The Burnett Honors College
at the University of Central Florida
Orlando, Florida

Spring Term 2015

Thesis Chair: Dr. Rudy McDaniel

ABSTRACT

As video games increase in complexity and length, the use of automatic, or procedural, content generation has become a popular way to reduce the stress on game designers. However, the usage of procedural generation has certain consequences; in many instances, what the computer generates is uncertain to the designer. The intent of this thesis is to demonstrate how procedural generation can be used to intentionally affect the embedded randomness of a game system, enabling game designers to influence the level of uncertainty a player experiences in a nuanced way. This control affords game designers direct control over complex problems like dynamic difficulty adjustment, pacing, or accessibility. Game design will be examined from the perspective of uncertainty and how procedural generation can be used to intentionally add or reduce uncertainty. Various procedural generation techniques will be discussed alongside the types of uncertainty, using case studies to demonstrate the principles in action.

DEDICATIONS

For my parents, you are my strongest supporters.

Thank you for being there when I need you and for your unconditional love.

ACKNOWLEDGMENTS

I owe the deepest gratitude to all of those who made this thesis possible and helped me along the way. Thank you to my thesis chair, Dr. Rudy McDaniel, for bearing with me as I explored my research topic, being a great mentor, and opening the door to many opportunities. Without you none of this would have been possible. Thank you to my committee members, Dr. Peter Smith and Dr. Sumanta Pattanaik, for inspiring me through your classes as well as with your advice. Thank you to the members of the Games Research Group for providing interesting discussions and literature, and for being a sounding board when I needed it. To the members of the Game Development Club, thank you for your endless source of encouragement. Finally, to all of my family, friends, and mentors, thank you for making the last four years at the University of Central Florida an invaluable and memorable experience.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

As game consoles and computer technology continue to evolve, the cost of manually creating content for games is becoming prohibitively expensive. Game companies already employ hundreds of employees, with production budgets for some AAA games being in the hundreds of millions. The creation of content can cost up to 30-40% of a game's production budget (Hendrikx et al., 2013). Thus, there is a large demand for methods to reduce the time and money it takes to create game content. One of the most popular ways to do this is through the implementation of procedural content generation.

Procedural content generation (PCG) is the programmatic generation of variable content (Merrick et al., 2013). PCG can reduce the burden on content designers by automating part or all of the design process. Contrarily, devising a complex procedural algorithm can be time-consuming and frustrating to configure precisely to meet a game's needs. There are many different genres of PCG, including content generated in real-time versus content generated offline. Real-time PCG includes the dynamic creation of content in game, such as infinite terrain that generates as the player moves. Offline PCG defines content that is created before the game is released and is stored like any other content. This could include landscapes or textures. While PCG techniques come in many forms, most suffer from a common issue: the lack of reliability and creativity. In particular, Merrick et al. assert that it is challenging to ensure that content generated using PCG is novel, useful, and of high quality (2013). Much of the research in PCG is focused around addressing this issue.

While considered an issue, unreliability is also a paradoxical benefit of PCG. Unexpected great content, as well as poor content, have potential to be created with PCG. Thus, it takes

careful design to create a successful PCG technique. Specifically, the algorithm needs to exist in a state between a fixed or periodic system, where the results are completely predictable, and boring, and a chaotic system, where the results are completely unpredictable, and often undesirable (Salen & Zimmerman, 2004). Salen & Zimmerman define this space as a complex system, or a system that exists in the gray area between predictability and unpredictability (2004).

This lack of certainty is something that not only needs to be present in a PCG algorithm, but is essential to the design of a game itself. In fact, Salen & Zimmerman claim that uncertainty is a central feature of every game, as it's a necessary ingredient in giving a game a feeling of purpose (2004). For example, two players who know the logic to Tic-Tac-Toe derive no satisfaction from playing, since the game will always end in a draw. Giving players meaningful choices and consequence through action means that they must forge their path through uncertain ground. PCG techniques in game design embrace uncertainty in a unique way. Some techniques generate caves in a random way, like the level generator in *Spelunky* (2008). Others can create interesting enemy behaviors or narratives for the player to experience by reacting directly to the player's input in real-time.

Procedural techniques in game design are becoming increasingly popular. The genre "Roguelike," whose namesake is taken from the classic game *Rogue* (1980), features games that have, among other things, procedural level generation. Roguelikes have blossomed from a niche field into the mainstream, with popular hits such as *Minecraft* (2009), *Dwarf Fortress* (2006), and *The Binding of Isaac* (2011). In *Minecraft* (2009), each new game generates a new world for the players to explore. In *Dwarf Fortress* (2006), the world and background is simulated to such

a depth that even the history of the generated civilizations is determined. And finally, in *The Binding of Isaac* (2011), each room surprises the player with random bosses and loot. Game developers have recognized that implementing broad, high-impact procedural design can influence the feel of the game further than just the layout of the levels. For example, the terrain generation in *Minecraft* (2009) does more than generate pleasing levels: the near-infinite variability suggests a mysterious, expansive world in which the player can explore and survive. While there are clear benefits of the properties of certain PCG techniques, like randomness giving the player more variety, the full scope of how PCG, uncertainty, and game design work together is not as obvious. PCG techniques, through the combination of randomness and a ruleset, create various degrees and types of uncertainty that have profound effects on the design of a game.

Thus, PCG-based design is an important topic, and the relationship between PCG and uncertainty in games is a topic that has seen little discussion. The purpose of this thesis is to explore and analyze common PCG techniques relative to principles of uncertainty in games. First, there will be a technical overview of PCG techniques and uncertainty in the literature review. Following that the techniques will be compared and analyzed in respect to their impact on uncertainty. Next, there will be several case studies with games that include PCG techniques. Then, PCG will be discussed holistically in respect to game design and conclusions will be made regarding how their presence can influence game design decisions.

Literature Review

In this section, different PCG techniques will be discussed in relation to how they attempt to solve the various challenges that inherently exist in the field. The purpose of this overview is

to provide a cursory introduction to each technique, so that simple references to how the mechanics of an algorithm interact with design can be made. While some techniques may not be often utilized in game design (instead, in game content like audio or textures), each one has the potential to be, and the implementation will be discussed in the analysis section.

Many of these algorithms have particular or niche use in games. A new algorithm is often developed to solve a specific problem. Thus, it's more appropriate to consider the techniques categorically. For the sake of this paper, PCG algorithms will be split into three categories: PRNG techniques, axiomatic techniques, and AI techniques. PRNG techniques include random generation as well as noise techniques. Axiomatic techniques involve the spectrum of generative grammar algorithms, such as cellular automata, Lindenmeyer Systems, or Shape Grammars. AI techniques include advanced algorithms such as genetic algorithms, self-learning algorithms like NEAT, and other self-adaptive algorithms.

First, two perspectives on games as uncertain systems are introduced. Then, pseudo-random-number-generation (PRNG) techniques will be discussed and how they use randomness to create structure will be shown. Next, generative grammars, including axiomatic techniques and cellular automata, will be surveyed, showing how different forms of content can be abstracted into rules and starting values. Following, artificial-intelligence-based techniques are discussed, especially the ones used in the website *Picbreeder*. *Picbreeder* provides a first-hand example of using PCG techniques in the design process, and the benefits and detriments are considered in relation to the goals of this paper.

Uncertainty

There are several ways to define what an uncertain system is. To Salen and Zimmerman, a certain outcome is completed predetermined (2004). An uncertain outcome is completely unknown to the player. In between, a risk is an outcome with a known probability. For example, rolling a five on a six-sided dice has a 1/6 probability of happening. Therefore, an uncertain system is one where actions involve some degree of risk. Salen and Zimmerman (2004) have identified several interesting properties of uncertain systems that are relevant to PCG. First, it is possible for a game to possess a "feeling of randomness" without actual random mechanisms present in the game system. Second, even games of pure chance can provide meaningful gameplay as long as players are given meaningful opportunities to take action within the game system. Ultimately, uncertain systems are about obscuring the outcomes that the player can predict while playing a game. Thus, the player must rely on taking risks in order to succeed (Salen & Zimmerman, 2004).

Costikyan (2013) offers additional nuance to the proposed system of uncertainty. While Salen and Zimmerman focus primarily on outcomes, Costikyan notes that the outcome of a game does not need to be uncertain for the game to possess uncertainty (2013). Many games in the arcade era, like *Space Invaders* (1978), have certain outcomes. In *Space Invaders* (1978), the player always loses. However, the process of earning a high score in each game session. The uncertainty lies in a mechanic within the game and not in the outcome itself. Additionally, the game itself does not need to have a quantifiable outcome, as is suggested in Salen and Zimmerman's definition of a game system (2004). Some games never end, like *Dungeons & Dragons* (1974) or *World of Warcraft* (2004), and they still keep player's attention and interest

through systems of uncertainty. In *Dungeons & Dragons*, the players perpetually keep the narrative going as their adventurers progress through the game. The uncertainty is in the path the players follow while playing, not the outcome (Costikyan, 2013).

PRNG Techniques

As a creative process, game design relies on novelty and uncertainty to provide engaging experiences to players (Zimmerman, Costikyan). This novelty needs to not only be unexpected but also useful, valuable, and appropriate. (Merrick et al., 2013; Liapis et al., 2014). One of the best ways a computer simulates variation and novelty is through random number generation. However, images generated randomly pixel-by-pixel have no meaningful structure. Many procedural techniques address this issue by finding the balance between iterative generation and random generation. Hendrikx, Meijer, van der Velden, and Iosup (2013) define this category of procedural content generation as PRNG techniques.

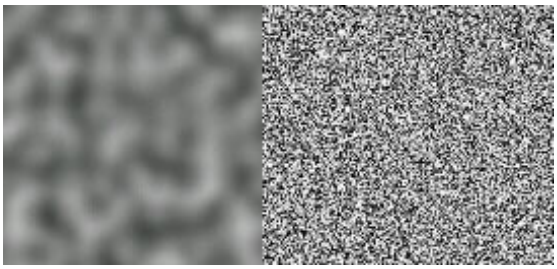

Figure 1. Perlin Noise (left) and White Noise (right). Perlin Noise smoothly interpolates between pixels to create meaningful structure. Static noise randomly generates pixels, and has no meaningful structure.

Some of the most used PRNG techniques are noise functions. Noise is considered the "random number generator of computer graphics" and is defined as an approximation to white noise band-limited to a single octave (Lagae et al., 2010). One of the most popular noise

functions is Perlin noise, which involves using pseudo-random gradients at vertices on a cubic lattice then doing a splined interpolation (Lagae et al., 2010). Noise functions are usually drawn to a texture and then interpreted by the program implementing them. Common implementations include heightmaps for terrain modeling and textures for things like sand or straw.

Generative Grammars

To avoid relying on randomness, many algorithms require the designer to specify an axiom or starting value. Using the axiom, a specific procedure is applied to generate content. Among these techniques is the category of generative grammars (Hendrikx et al., 2013). Generative grammars define a rule-set then recursively generate content based on what kind of grammar it is. So if the axiom is defined as A, and the rules defined were 1) A→AB and 2) B→A, the iterations would be: 1) A 2) AB 3) ABA 4) ABAAB … and so on (van der Linden et al., 2014). The designer would then generate content based off of the axiom and how many iterations were defined. Other types of grammars, like shape grammars or graph grammars, build upon this concept but utilize other content instead of letters and words. Graph grammars use an algebraic approach to generation, while shape grammars use common shapes like triangles, squares, and circles (Gips, 1999). Generative grammars are often used for sound, vegetation, trees, and buildings (Hendrikx et al., 2013).

Additionally, there have been novel attempts to extend the idea of generative grammars to other game content. To do this, a vocabulary must be defined, which can manifest as shapes, letters, or any other form of modular visual representation. Then, rules must be specified that define the relationship between these components. Finally, an initial configuration is defined to start the generation process (Merrick et al., 2013). For example, gameplay can be generated by

defining atomic gameplay components (representing letters in a generative grammar), game loop operators (the rules of the grammar), and an axiom combination of atomic gameplay components and game loop operators (Francillette et al., 2012).

Another axiom-based generation technique is cellular automata. Like generative grammars, there exists an axiom and a rule-set. However, cellular automata's rules are applied iteratively rather than recursively. For cellular automata, a finite grid of any dimension is first specified. Next, the designer provides the initial state of the grid, which consists of the state of the cell at each point in the grid. Then, the designer provides a rule to apply to each cell that is dependent on their state (van der Linden et al., 2014). To create content, the designer then provides a time *t* that represents the number of iterations. Cellular automata have been used to generate sound and NPC (non-player character) behaviors as well as large caves.

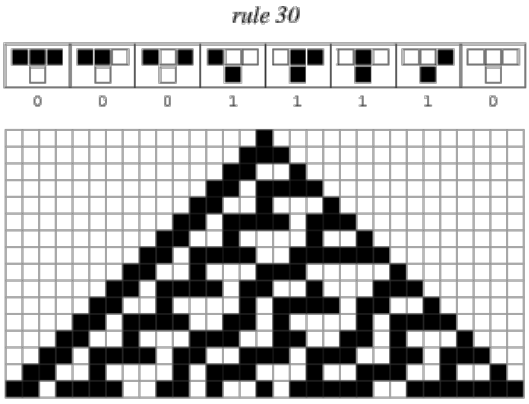

Figure 2. An example of the evolution of a simple cellular automata. At the top are the rules, where 0 defines an empty cell, 1 defines a full cell, and the patterns are the immediate neighbors needed to activate a particular rule. Below, the evolution of a single black cell after 15 steps is shown.

Artificial Intelligence Techniques

Other algorithms use evolutionary techniques or artificial intelligence to generate unique and interesting content. In Secretan et al.'s *Picbreeder*, users are introduced to a wide variety of pictures evolved using Neuro-Evolution of Augmenting Topologies with Compositional Pattern Producing Networks (NEAT-CPPN). This technique evolves the images from random starting points by using the user's input as new points of origin. The evolution happens by cross-breeding two images and taking elements of each. The data structures that define these images become increasingly complex as time progresses. Using this technique, complex, comprehensible, and novel pictures have evolved from something completely nonsensical. Secretan et al. (2011) note that in Picbreeder, users would lose interest within 10 to 20 generations, as they see no significant change from the original iteration. In Picbreeder, this issue is addressed by providing a feature called branching, which allows users to save their current state and have other users pick up where they left off.

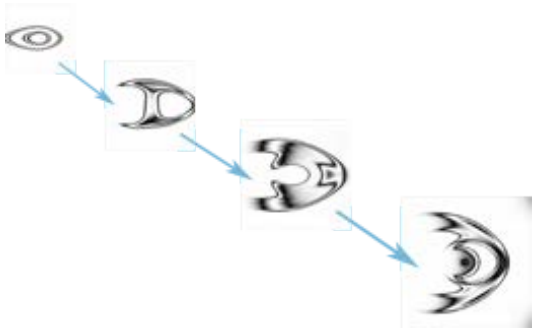

Figure 3. The evolution of an image in Picbreeder. Users evolve pictures from abstract shapes (top-left) to something more defined (bottom left) by choosing between families of similar images generated by NEAT-CPPN. Any of these images could "branch" and evolve along a different path.

When the designer is left out of the iteration cycle, as is the case with PCG, they are unable to connect the stepping stones they need to get to their intended result. Stepping stones are the iterations needed to get to the final objective (Lehman & Stanley, 2011), which in the designer's case is creating an intended experience. Say the designer was constructing a house – the first stepping stone may be the door, followed by the foundation and walls, then finally windows and a chimney. However, houses can be constructed in many different ways, and no two designers would create the exact same house. In this way stepping stones are surprisingly complex and inherently local to problem at hand (Lehman & Stanley, 2011). Picbreeder shows that humans are quite good at finding and identifying stepping stones. There have been many unique and identifiable images that have come from users selecting from meaningless pictures.

In practice, many procedural algorithms are carefully constructed from simpler techniques like PRNG in order to generate "acceptable" results without risk of "catastrophic failure" (Togelius et al., 2011). Togelius, Yannakakis, Stanley, and Brown demonstrate that most algorithms fall under either the "generate-and-test" category or the "constructive" category. A generate-and-test algorithm is split into two parts: generation and testing. Content is creating during the generation phase, and then this content is tested to meet the designer's standards. Using the constructive method, the content is generated one time only, but the algorithm for generation usually ensures the content is fail proof. In both of these categories, content is generated all at once, and thus the designer has no input while the content is being generated. Thus the designer must be incredibly careful when utilizing PCG techniques in a game. Ultimately the designer is satisfied with the probable or guaranteed results of an algorithm, and it is at this point where the consequences of having an uncertain system affect gameplay.

ANALYSIS

In this section, the algorithms mentioned in the literature review will be analyzed in relation to their effect on a game's uncertainty. Specifically, I will be analyzing PCG as it applies to Greg Costikyan's (2013) ten categories of uncertainty. The topic of uncertainty is a broad one, and different perspectives can lend to highly varying metric. For example, some authors approach uncertainty from a logical angle. Game theorists derive complex formulae to explain a player's decision-making process in games of strategy. These kinds of experiments draw deeply from economics and mathematics to come to their conclusions. While the findings are applicable to game design, they do not consider the full scope of a player's involvement in a game. For example, what happens when there is no discernable outcome, or the outcome is irrelevant to the stated objectives of the game, as in a game of make-believe? What if the game does not include more than one human player, like a single-player video game or solitaire?

To account for these questions, game scholars tend to draw more from sociology and cultural history when analyzing gameplay elements. For example, to explain the relationship between competition and chance, Callois (1958) draws upon the history of shamans in ancient times, progressing to ancient Greece, and finally to modern France and the lottery. Using mythological stories, accounts of rituals and games from the time, and an over-arching narrative, he proposes a case for games of chance being a psychological equalizer in a world of inequity. Avedon and Sutton-Smith (1971) survey children's games to qualify their propositions for defining games. Drawing too from other scholars of the time, they propose a dimensional structure to games, ranging from skill requirements to bodily contact. In a similar way, Costikyan (2013) surveys videos games to help frame his categorizations of uncertainty in

games. As with most generalizations, Costikyan's (2013) categories do not completely envelop the entirety of uncertainty in games. However, they do serve a strong point of reference, and have a solid foundation in seminal works like *Rules of Play* (2004) and *Man, Play, and Games* (1958). Thus, to analyze PCG techniques in respect to uncertainty, Costikyan's (2013) categories will be considered. The many ways in which PCG can adapt to each category demonstrates the power and utility of introducing procedural content.

Performative Uncertainty

Costikyan (2013) considers performative uncertainty to be the "uncertainty of physical performance." This category deals with hand-eye coordination, challenge, and reflexes. Games that have a large amount of performative uncertainty are games that require continuous attention and excellent timing for success. These games manifest primarily in the genres of first-person-shooters, real-time-strategies, or platformers. The uncertainty is derived from both the game and the player. If the game is too easy, even a novice player will not feel the sense of uncertainty while playing. Conversely, a game too challenging is unconquerable to the player, thus leading to a feeling of certain defeat. Players who are experienced with a game eventually develop skill. High skill leads to previous challenges becoming simple and thus losing uncertainty.

A solution to this progression is what Costikyan (2013) calls "dynamically adjusted difficulty." Difficulty adjustment is a common topic in procedural generation. In its simplest form, difficulty could be adjusted based on the player's achievement of game objectives. If the player is doing well, increase the difficulty. If the player is struggling, reduce the challenge. Scholars have found that a nuanced approach is necessary. For example, what if player A is struggling with the games requirement for quick reactions, but player B is having difficulty with

the spatial navigation (Shaker et al., 2015)? Adaptive AI algorithms can analyze a player's experience and dynamically adjust the next level according to a predetermined formula. For example, Shaker, Togelius, and Yannakakis (2015) modified the game *Infinite Mario*, a public domain clone of *Super Mario Bros.*, to dynamically record and quantify the experience of the players. Then, personalized levels will be generated to best fit the player. The goal of this algorithm is to maintain a state of constant uncertainty, so that even if the player's skill is unequal in different areas, there will always be a degree of uncertainty present. Ultimately, by modifying the difficulty of the game, PCG can influence the performative uncertainty of a game in a direct and impactful way.

Solver's Uncertainty

Puzzles are the basis of solver's uncertainty. It's the uncertain nature of trying to figure out a solution, and of not immediately knowing the answer. Once the answer is discovered, the challenge drastically reduces. Costikyan (2013) considers that many digital games provide "no compelling reason to want to play a second time." However, Solver's uncertainty extends beyond simple puzzles. Games often create complex puzzles through their own rules of interaction. Consider a game where a player must jump onto a ledge to continue. First, the player must reason how far from the ledge to jump, and how fast of a speed is required to make the jump. If additional challenges are added, like a spike pit, or an enemy that must be avoided, it further complicates the problem. Even in games not primarily concerned with puzzles, solver's uncertainty is present as an embedded element (Costikyan, 2013). Solving any challenge in a game can be considered a puzzle to some degree.

The main factor in solver's uncertainty is whether or not the player has completed the certain challenge. The bigger the focus on solver's uncertainty, the more this becomes an issue. For example, in a point-and-click adventure game, once the player knows the correct sequence to complete the game, there will be no further challenge and the uncertainty will be lost. However, in a game where there is a combination of puzzle and action elements, like a platforming game, the player can still find enjoyment in previous levels if the skill requirement is tuned appropriately.

Thus, a good solution to create more uncertainty is to simply create more challenges. With games that feature narrative-based challenges, like point-and-click adventure games, this is nearly impossible. However, games that are more mechanical by nature find new levels easily generated. This is the basis of the roguelike genre: endless level generation so that there is always a new puzzle for the player to discover. In fact, procedural level generation is common across almost all genres, including first-person-shooters, RPGs, and more. The types of algorithms used to create levels is just as varied, including basic random generation, axiomatic generation, and self-adaptive techniques. For example, the endless hills, mountains, and caves of *Minecraft* are spawned using a seed value and a constructive algorithm. Cellular automata can used to create cave structures or dungeons in 2D games. AI techniques can use AI agents (sometimes called "miners") to carve out a level from an initial block, with the algorithm determining how the agents carve out the space (Shaker et al, 2015). Regardless of the method, procedural level generation creates uncertainty in a game through the introduction of novel content. While these levels may not match the design artistry of a game designer, the availability of nearly infinite content is the basis for many successful games.

Player Unpredictability

When designing games, predicting the player's behavior is nearly impossible. There are so many factors that go into how a player plays a particular game: how old they are, what gender they are, how many games they have played, what kind of games they have played, what they had for breakfast, etc. However, the player is not random, as player psychology is not random. The quintessential example of player unpredictability is *Rock/Paper/Scissors*, as Costikyan (2013) explains: "The reason *Rock/Paper/Scissors* is not a purely arbitrary game, and the reason that an excellent player will win more often than chance would predict, is that human psychology is *not* random, and some behaviors are – not necessarily predictable, but likely to occur more often than chance would dictate." Thus, for a player, any time another player is encountered in a game, a degree of uncertainty that is associated with player unpredictability arises. Consequently games that do not feature multiplayer do not have player unpredictability.

Games mitigate or emphasize player unpredictability by how they let players interact. *Rock/Paper/Scissors* is almost purely based on the unpredictability and pure player interaction. On the other hand, Costikyan suggests *Monopoly* (1903) as a game that minimizes player unpredictability. In *Monopoly* (1903), the players can not directly interact with each other, but rather indirectly through the purchasing of property. While it may be unpredictable to determine whether or not another player is going to purchase a property, it has no bearing on any decisions a player might make. The player's sole interaction is with the dice and the game systems.

One way to control the level of player unpredictability in a game is to procedurally dictate how players interact. Certain features that can increase or decrease the influence of other players within a game, like friendly-fire in a FPS or the ability to disable an enemy player's

controls, can be dynamically added and removed from the game. For example, in the game *Dark Souls* (2011), the game is a single-player experience, with the exception of invasions. In an invasion, a player enters the world of another player and attempts to defeat them. While the system in the game is not procedural, one could imagine an algorithm to generate invasions, based off of player experience or some other self-adaptive system.

Randomness

Dice rolling games have been a part of human culture since the dawn of civilization (Caillois, 1958). When randomness is integrated into video games, it involves different variations of a simulated dice roll. According to Costikyan (2013), players are not fond of the idea of randomness, especially players of strategy games, since they feel it invalidates any accomplishment they make. Nonetheless, randomness is an integral part of many video games. In FPS games, the way the shots of a shotgun scatter may be random. In RPGs, there is often a chance to "critically hit" which drastically increases your damage for one attack. Through uncertainty, randomness creates a sense of drama, when "the player otherwise commits himself to a course of action the outcome of which is luck dependent."

In the case of video games, randomness is not truly random. Any random number a computer generates is actually pseudo-random. As Salen and Zimmerman (2004) note, "computers can never computer purely random numbers, because the numbers they provide are always the result of algorithms." What is more important is the feeling of randomness. As with computer-generated random numbers, a sense of randomness can be created with an algorithmic, sequential sequence. Salen and Zimmerman continue: "When four, five, or six players play [*Chinese Checkers*], it can feel quite random. As the game unfolds and players move their pieces,

the center of the board becomes crowded with a seemingly random arrangement of pieces… This feeling of randomness is only an illusion, however, as there is no formal chance mechanism in the game" (2004).

In a similar way, this is how PCG techniques function. They use a predetermined algorithm to create a feeling of randomness and variety. This is particularly true with PRNG techniques and noise functions, but the concept also extends to axiomatic algorithms and AI techniques. For example, when an axiom grid has been determined, cellular automata are completely deterministic. However, if someone were to look at certain random patterns in *The Game of Life* (1970), a famous visualization of cellular automata, this person would likely conclude that there is no structure. With *Picbreeder*, even though each new panel of images is generated based on the user's decisions, each modification is abstract and small enough to be incomprehensible. Thus the user might conclude that each new panel is random. For any algorithm to induce a feeling of randomness in a player, it needs to be perceived as random. If a random number generator displayed the numbers 1, 2, 3, 4, 5 in sequence, it would not appear random. Similarly, the player encounters any sort of content in a logically progressing way, it appears less random. Ultimately, how the particular algorithm is designed and implemented is what affects the degree of perceived randomness in a game.

Analytic Complexity

Chess is a game that presents players with so many possible options, so many strategies, and so many paths, that it is impossible reliably predict beyond a few moves. It is a game that is so complex that supercomputers still have not mastered it. Analytic complexity is all providing too much information for a player to keep track of. Tic-tac-toe is an example of a game that, for

most adults, has no analytic complexity. It is easy to keep track of the current state of the board and predict future moves of other players. In the realm of video games, grand strategy games like *Dwarf Fortress* (2006) or *Civilization* (1991) provide worlds simulated to meticulous detail. For players, the uncertainty is in deciding where to place their attention or focus in making decisions.

Crafting a world as deep as one found in *Dwarf Fortress* (2006), where weather patterns, mineral deposits, family lineages, and much more are accounted for, is not something feasible to do by hand. *Dwarf Fortress* (2006) is an excellent example of using procedural generation to enhance the design of the game. Unencumbered from having to write the stories of hundreds of thousands of dwarves, the creators of the game were free to experiment with the dynamics and gameplay that emerged from such a detailed simulated world. While the nature of the generation is not fully publicly available, worlds can be generated using a variety of techniques. This includes some of the previous discussion: using cellular automata for level generation, for example. However, this extends to other forms of content, such as sounds, text, and character design. Specialized PRNG algorithms that contain configurable seeds, like in *Dwarf Fortress* (2006), are popular among game developers, and scholars have devised other methods to generate this kind of content. For example, generative grammars and Lindenmeyer systems are often used to generate vegetation in games.

Hidden Information

Often a player's main source of uncertainty comes from a lack of information. Many games, such as *Poker*, deliberately hide information from the player. In video games, the implications of hidden information are especially apparent. The computer provides the role of "games master" and controls how information is disseminated. To contrast, in most board games

and sports, every player reads the rules and knows the full scope of possibilities before beginning. Costikyan (2013) calls this the difference between “known unknowns and unknown unknowns.” An unknown unknown is some part of the game that the player is completely unaware until they first encounter it. A known unknown is a part of the game that the player knows exists but does not know where it exists in the game state yet. An example of an unknown unknown is a secret boss encounter in *The Binding of Isaac* (2011). An example of a known unknown is your opponents hand in a game of poker. The player knows it could be one of the 52 cards in a deck, but does not know which one. In other words, the player knows the range of possibility.

Hidden information is a procedural algorithm’s secret weapon. Even if the player knows that a level is being generated procedurally, most of the time there is no possibility of the player knowing the full range of possibilities when encountering generated content. With evolutionary algorithms, novelty is reached right before the player’s eyes as the content is steered towards a certain direction. In *Picbreeder*, no user knows what will be generated, or what even could be generated. It is the payoff of the hidden information being revealed which is satisfying to the player. Clever game designers hide special events in only a small amount of generated levels in their game. For example, in *Desert Golfing* (2014), the player is exposed to hundreds of levels featuring the exact same environment: sand, sky, and a hole. However, once every couple hundred levels, something novel appears: a cloud, a rock, a cactus, and others. The result transforms “an otherwise flat design into one that is quite compelling” (Costikyan, 2013).

Narrative Anticipation

Uncertainty and narrative anticipation go hand in hand. If the player can anticipate what's going to happen next, there's no uncertainty. While it's important for story in games to have this quality, it extends to the gameplay as well. For example, in *Chess* if a player has lost most of his pieces and left with just a king, while the player's opponent has multiple pieces remaining, the game has a foregone conclusion. In *Monopoly* (1903), if a player is on the verge of bankruptcy and does not own any properties, it's just a matter of time before the game is over. Most games want the outcome to be uncertain until the very last moment. Costikyan (2013) details games that implement "negative reinforcement loops," in which strength is dynamically move around throughout the game in order to give weaker players a chance to affect the outcome.

Within the domain of story, the concept of procedural narrative is one that is often discussed. *Façade*, a game by Michael Mateas, is an interactive story that attempts "procedural authorship" (2003). It uses player input (typing responses to character prompts) to determine story "beats," the small sub-plots in the overarching narrative. Within each beat the characters react to the player's input and transition to the next beat. Each beat is only a few seconds long, facilitating intermixing of beat and allowing more divergent paths. New beats are chosen based off of the player's input or lack of input. In *Façade*, the player can not only play through a different experience in two consecutive play sessions, but is entertained with a system that directly reacts to his choices in the game (Mateas & Stern, 2003).

Outside of story, unpredictability can be asserted in games through a variety of procedural methods. Random events can occur that can drastically affect the current game state, such as a major character falling in battle or a new member being added to the party. Players can

be given harder or different challenges based on their current progress in the game. For example, the game *Left 4 Dead* (2008) includes an AI system called the "AI Director" (Booth, 2009). One of the key systems of the AI Director is to have adaptive dramatic pacing. In other words, creating peaks and valleys of intensity to develop drama on a moment-to-moment basis in game. To do this, the AI Director would monitor the player state (called "Survivor Intensity") and dynamically add or remove zombies in the game to match a dramatic intensity scale (Booth, 2009). The AI Director is an example of a practical, local PCG algorithm built for one specific game. While recreating algorithms is time consuming, it can pay great dividends by promoting replayability, increasing output of the development team, and creates great, curated game experiences.

Development Anticipation

With the rise of the internet and the ubiquity of game industry practices such as "patches" or online updates, new content can be delivered seamlessly to a player even after a game is released. Downloadable content is becoming such a common practice that companies are developing post-release content during the actual game's development lifetime. This content is sometimes added for free, whether as part of its business model or a feature that did not make it to release. Just as often is paid downloadable content made available, and comes in the form of additional characters or levels, or some sort of content that extends the gameplay.

Along with this comes the need and expectation for more content. Offline procedural techniques, like using noise to generate textures for a 3D model, are being used frequently in the industry to speed up development time (Hendrikx et al., 2013). Sometimes procedural algorithms are more accurate than designers could be, as they are based in real science or simulation

research. This is the case with terrain generation or foliage generation. The fact that games change form as they are played is something that game players have to adapt to, and is certainly shaping the game industry today.

Schedule Uncertainty

Schedule uncertainty points to how long a player is allowed to play a game for. A young child might only be able to play outside until supper. In the arcade, you only had a certain amount of coins to spend. Many mobile games restrict play time, unless you pay for additional lives or game time. Costikyan (2013) calls this kind of uncertainty "crude and fairly unaesthetic," yet "proven financial success."

Intentionally limited the amount of time a player can spend playing a game is a rarely discussed topic in games scholarship, but it does raise an interesting point. Costikyan (2013) notes that at the end of a limited session, "players typically have more things they wish to accomplish than their available resources permit." Perhaps a system similar to the AI Director in *Left 4 Dead* (2008) could be appropriate; it would analyze the player's current state and limit content based on various criteria. This could be useful in order to guide player action and retain players longer on platforms where users are more fickle such as mobile.

Uncertainty of Perception

Without the ability to perceive what is happening on the screen, the player would not be able to interact with a video game to any degree of success. Manipulating the difficulty of perceiving what's going on in is a common game mechanism. For example, there is a common street trick where a performer asks someone to identify which cup (of three) holds a marble. The performer then puts the marble in one of the cups, spins and moves the cups in a way that the

guesser loses track, then allows them to guess. Additionally, Costikyan (2013) mentions rhythm games, like *Guitar Hero* (2005), or hidden object games, where a player has to visually identify items on screen. These games involve overloading the player's perceptual stimuli there by making them uncertain of what they just perceived.

The inherent perceived randomness of many PCG techniques benefits this type of design well. Humans are superb at identifying patterns, so if there is any hint of a pattern to be found, the player would no longer need to rely as heavily on their senses. As a counter point, in rhythm games, the player is presented with the same sequence each time. However, songs can be hundreds or thousands of notes long, and the player would have to play many hours to memorize an entire piece. For these games, a generative grammar could be used to generate levels in a logical way so that the gameplay of songs had an internal structure, as well as decreasing development time if a large catalog of songs was needed. Games like *AudioSurf* (2008) are even able to analyze user-supplied songs and generate gameplay automatically.

Summary

As this section has shown, the utilization of PCG to impact uncertainty of games are varied. There are, however, some trends to be discovered. Frequently, the design of level generation algorithms are what have the biggest impact on uncertainty in various forms. Perhaps this is due to the broad and malleable nature of a level. A level can represent a tiny puzzle or an entire world. A level can include some gameplay or the entirety of it. Procedural techniques can generate an infinite number of levels to create solver's uncertainty or randomness. Entire worlds can be generated in order to promote analytic complexity. In fact, level generation can impact the

entire spectrum of uncertainty, because levels are the fundamental parts of the game that the player interacts with.

Another common use of PCG with respect to uncertainty is to dynamically alter the game. This is done by devising metrics that measure a certain quality of a player's experience. How many times have they died? How long did it take them to complete an objective? Which route did they take? Compiled together, these form a statistic that a procedural algorithm uses to alter the game in some way. This can be done through increasing or decreasing the difficulty of the game, as found in the AI Director for *Left 4 Dead* (2008), or by choosing a different narrative path to follow, like in *Façade*. This practice allows designers to deal with a wider variety of players, and ultimately empowers designers with the ability to make their game accessible to more people.

Finally, a newer use of PCG is to deal with meta-uncertainty. This is when uncertainty bridges the gap between reality and the gameplay experience. For example, how might a game be designed differently if the designers knew about a potential update down the road? Perhaps content might be generated in real time to match a new area that was just patched into the game. This could be a street sign that generated text that matched the name of the city, or a texture of a building dynamically changing color scheme for a special holiday event. As a whole, PCG is a useful and practical tool in many situations to modify the amount of uncertainty in a game, and already has found myriad uses in the games industry. In the next section, a few of those games will be discussed.

CASE STUDIES

In this section two video games, *Left 4 Dead* (2008) and *Spelunky* (2008), will be examined. These two games do not represent all of the PCG techniques in action, nor all the types of uncertainty. What they do represent are two examples of intentional and thoughtful use of PCG to enhance the design of a critically successful game. They used PCG in different ways to influence uncertainty, therefore providing a good reference for seeing how PCG is applied in practice. In each examination, an overview of how they implement PCG techniques will first be discussed. Following that there will be a discussion on how the use of PCG impacts the game and its development. Finally, uncertainty will be considered in relation to the PCG techniques used and the design of the game.

Left 4 Dead

Made by the company that developed *Half Life* (1998) and *Portal* (2007), *Left 4 Dead* (2008) is a co-op action horror game that "casts players in an epic struggle for human survival against swarming zombie hordes and terrifying mutant monsters." As a cooperative first-person shooter, the player teams up with computer-controlled allies or player controlled allies as they make their way through various environments. The game has a focus on teamwork and survival, requiring players to band together and help each other out as the number of zombies can be overwhelming at times. Thematically, each campaign is considered a "movie," with the players being the protagonists of the film (Booth, 2009). There is an additional "Versus Mode" where players can play as survivors as well as zombies. If the survivors make it through the level alive, they win. Otherwise, the zombies claim victory.

In *Left 4 Dead* (2008), one of the design goals was to promote "replayability." This means that a player needs to be able to play the same section of the game repeatedly without losing interest.

The designers, when looking back on their previous successes with games like *Counter Strike*, *Team Fortress*, and *Day of Defeat*, saw positive feedback with unpredictable experiences created by interactions between players and comparatively few maps that remained memorable in the player community (Booth, 2009). One of the tools the designers turned to was PCG. In *Left 4 Dead*, procedural population of enemies and loot strongly promotes replayability. The goal with including procedural content was to tailor the game session to feel like a skill challenge rather than a memorization exercise. In games that have static placement of enemies and items, players simply memorize all of the static locations, thereby removing the suspense of not know what will happen next. It is especially harmful for cooperative games, where players expect everyone to have memorized all of the encounters and degenerates the experience into a race (Booth, 2009).

To implement their procedural generation, they designed a system called the AI Director. As mentioned previously, the AI Director manages the dramatic pacing of the game. It does this by dynamically generating encounters for the players based on their current performance. To populate the world, they use a "structured unpredictability," which exists somewhere between being purely random and deterministic (Booth, 2009). The system is a superposition of several "population functions", which are designed by hand and vary by space, time, and entity generated. For example, a function could spawn a mob entity at a randomized interval between 90 and 180 seconds behind the survivor team. Other population functions include "wanderers" who are enemies that wander around in a daze, weapon caches, scavenge items, and bosses

(Booth, 2009). To determine where an entity is placed, the AI Director breaks a level down into "areas." Enemies are not spawned in areas too close or too far away to the survivors. Additionally, the AI Director can detect which areas are in front of or behind the survivors, in order to create encounters like an ambush or a blockade (Booth, 2009). Procedural population of enemies is a practical example of using PRNG techniques in sequence to create a desired effect.

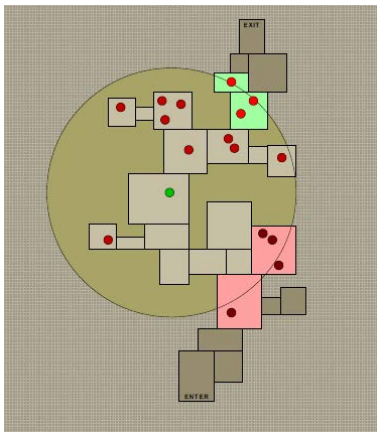

Figure 4. The red dots show potential enemy locations in a Left 4 Dead map. The green dot represents the player.

In addition to the procedural population of enemies, the AI Director also needs to manage game pacing. The inspiration for the pacing of *Left 4 Dead* comes from *Counter-Strike*. In observing *Counter-Strike*, designers found the natural pacing to be "spiky," with periods of quiet tension followed by unpredictable moments of intense combat (Booth, 2009). In fact, it was the unpredictability of these spikes in intensity that create a compelling and replayable experience.

In order to measure the pacing, metrics needed to be established. The AI Director measures the "emotional intensity" of each survivor by representing it as a numeric value. This value increases when a survivor takes damage, becomes incapacitated, is pushed off a ledge, or

killed a zombie. This intensity decays over time when a survivor is not actively fighting zombies (Booth, 2009). With the metrics established, pacing can be dictated. When the AI Director decides a peak in intensity is appropriate, it enters a “Build Up” phase, where the maximum amount of enemies are created. This continues until the combined survivor intensity reaches a threshold value. This peak is sustained for a few seconds, then it transitions into the “Relax” phase, where a minimal amount of zombies are created (Booth, 2009). This system creates natural ebbs and flows that are dictated by the player’s actions, and not a scripted event. The AI Director’s use of adaptive procedural techniques creates a unique feeling to each play session.

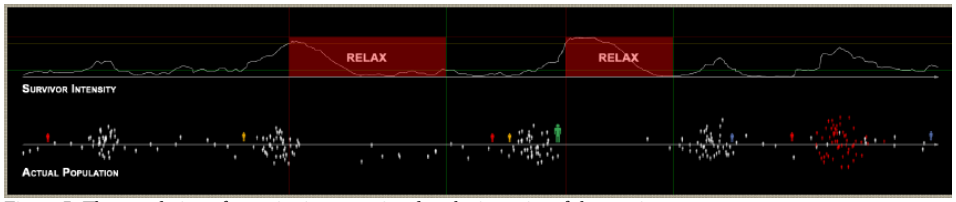

Figure 5. The population of enemies is proportional to the intensity of the survivors.

A term that often gets used in game design discussions is “replayability.” It has already been used several times. Logically, it means the ability for an experience to be enjoyed multiple times. In practice, it means maintaining a constant level of uncertainty for the player. In *Left 4 Dead* (2008), the answer was to create an experience that was catered to the player’s performance. In this way, the player would always be adequately satisfied, and never feel bored or overwhelmed. With the procedural population of enemies, the AI Director utilized Costikyan’s (2013) ideas of randomness, performative uncertainty, and hidden information. The players never exactly know where an enemy will spawn, and to them it feels random, even though it is somewhat controlled by their actions in-game. With the control of pacing, the AI Director took advantage of narrative anticipation and player unpredictability to create unique

experiences. The player's never know when the next spike in intensity is going to occur. For a player in the game, the performance of their teammates directly affects how often the spikes do occur, and that unpredictability contributes to each play session feeling unique.

Spelunky

In 2008, Derek Yu released a small "freeware" game named *Spelunky* (2008). It is a "roguelike-inspired" 2D platformer where the goal is to explore underground tunnels, gathering as much treasure as possible and not dying to spikes, snakes, and other hazards. The player is equipped with bombs, ropes, and a whip by default, which can be used to both navigate the tunnels and fight off enemies. Outside of hazards and treasure, and the default weapons, players can find other items to assist in cave navigation. These can include a cape to avoid falling damage, spiky shoes to climb up the sides of walls, and spiders' webs to allow bombs to stick to surfaces. *Spelunky* (2008) is a game notorious for its incredible difficulty and superb random level generation. Critics praised the sense of discovery, and the difficulty is eased by potential to find new and interesting things. One critic notes: "Even after you splay your guts on a patch of insta-death spikes, ruining a long session of spelunking, the prospect of a new discovery is ready to entice you back for more."

To generate its levels, *Spelunky* (2008) uses a domain-specific chunk-based approach. Simply put, it splits the level into a grid, with each cell being an individual room or chunk. Each room has is generated based off of a template created by the designer (Kazemi, 2013). Rooms are selected from a pool based on criteria that ensures that it is possible to get from the start room to the exit. Rooms can be split into 4 types: type 0, a side room that is not on the solution path (the solution path being a traversable path from the start room to the exit); type 1, a room that has an

exit on the left and the right; type 2, a room that has exits on the top, right, and bottom; and type 3, a room that has exits on the left, right, and top (Kazemi, 2013). The criteria is as follows. First, the level grid is considered. For the sake of this discussion, the level is made up of 16 rooms in a 4x4 grid. The first thing that is done is that a start room is placed somewhere on the top row. This room is usually a type 1 or type 2 room. To decide where to place the next room, a simple algorithm is followed. A random number is chosen from 1 to 5. If the number is 1 or 2, the next room is placed to the left. If the number is 3 or 4, the next room is placed to the right. If the number is 5, then the next room is placed below. If the next room is placed off of the grid to the left or the right, it instead creates a room below the last room placed. The new room is always a type 1 room. If the new room is placed underneath that last placed room, then the last placed room is converted to a type 2 room, and the new room is a type 2 or 3. The final condition is that if the new room is placed off the grid to the bottom, then the previous room placed becomes the exit room and no new room is created (Kazemi, 2013). After this convoluted series of events, the "solution path" has been created. All additional empty grid cells are filled with type 0 rooms.

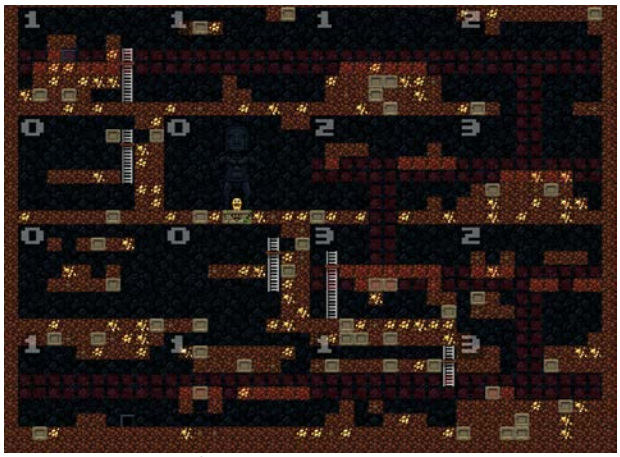

Figure 6. An example of room type distribution in *Spelunky*.

Within each room, procedural generation of a similar style happens as well. As mentioned before, rooms are generated based off of a template structure. The templates determine the overall room layout; whether the room is wide open, full of platforms, or claustrophobic. A template is a 10x8 grid of tiles. A tile can be a static type or a probabilistic one. A static tile is guaranteed to be a certain type, while a probabilistic one is empty with a chance to be a special tile. Each tile is represented by a character: “0”, an empty space; “1”, 100% chance of a solid block; “2”, a 50% chance of a solid brick; “L”, a ladder; “P”, a top-of-ladder platform, “4”, a 25% chance that there is a block at the top of a ladder, etc (Kazemi, 2013). A level template might look like this:

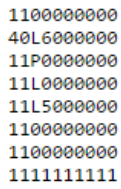

Figure 7. An example of a level template in *Spelunky*.

By combining simple, randomly generated functions, *Spelunky* (2008) creates rooms that are well designed, random, and always traversable. The beauty in simplicity is that it becomes trivial to modify the design without breaking the entire system. It does not require a team of experts to design and implement new levels, and the time it took to develop this algorithm and iterate on it is much less than in ones that use complex algorithms like NEAT. Ultimately, however, the result of *Spelunky's* (2008) level generation is an excellent level of uncertainty and challenge. Each level is tuned to a satisfying level of performative uncertainty and solver's uncertainty, as the players must figure out the correct path of action to maximize their treasure and minimize their chance of death. Additionally, each level has a chance of a special event, like cutting out all of the lights, adding snake pits, adding a spider's lair, or flooding the tunnels. Events like these pepper the landscape of *Spelunky*'s (2008) levels, and emphasize the biggest strength of the game: its use of hidden information to drive uncertainty. The player, even with many hours played, is unaware of all the game's tricks, and each play session introduces something new. Thus, the player is continually engaged and interested.

DISCUSSION

This paper has looked at PCG techniques from certain perspectives. The literature review contained a high-level technical overview of each technique. Next, the techniques were categorized and analyzed in respect to the different types of uncertainty described by Costikyan (2013). Finally, some of the techniques were described in practice in the case studies for *Left 4 Dead* (2008) and *Spelunky* (2008). The purpose of this has been to demonstrate how procedural content generation can have a profound effect on the perception of uncertainty in games. It is useful to look at the games relatively to see how they compare:

Table 1. Games with procedural content generation that have been mentioned in this thesis.

Game	PCG Type	Dominant Uncertainty Type
Minecraft	PRNG	Hidden Information
Dwarf Fortress	PRNG	Analytic Complexity
Binding of Isaac	PRNG	Hidden Information
Infinite Mario	AI	Performative Uncertainty
The Game of Life	GG	Randomness
Desert Golfing	PRNG	Solver's Uncertainty
Façade	AI	Narrative Anticipation
Left 4 Dead	PRNG, AI	Narrative Anticipation
Spelunky	PRNG	Performative Uncertainty
AudioSurf	AI	Solver's Uncertainty

As a note, almost all PCG algorithms affect randomness, narrative anticipation, and hidden information. Therefore, the uncertainty type that a game's PCG had the most influence on

is a more useful metric. PCG proved to be a versatile modifier of uncertainty, with the type of PCG not being an indicator of the dominant type of uncertainty it influenced. The most utilized technique in practice is PRNG. For most of the games that implement it, the complexity of the algorithm does not exceed using simple random number generators to decide what content is to be displayed and where, as in *Spelunky* (2008). PRNG's popularity is due to its simplicity, allowing for quick iteration and a high amount of customization. GGs are not used as much in the industry. While they are commonly used for vegetation, and they can be used for generating levels, they lack the level of control needed for game designers to create an intended experience. Finally, AI techniques are used more often than GGs, but have seen limited commercial use. *Infinite Mario* and *Façade* are just two of many academic games that have been published. In contrast to PRNG techniques, AI techniques are much more difficult to implement and take considerable time to iterate on. Thus, it is less appealing to integrate them into commercial games.

Meaningful Play and Uncertainty

Little discussion has been made on how uncertainty directly relates to game design. Concepts like "engagement," "accessibility," and "replayability" have been cited as results of procedural techniques adapting to the player or randomly generating new experiences. This does not mean that any time a PCG algorithm is utilized, engagement or accessibility or replayability occurs. To the contrary, it takes a deliberate and steady use of procedural generation to achieve any noteworthy results. For example, if the generation is too loose and unorganized, excessive uncertainty may result. With excessive uncertainty, the player has a lack of control over the game state. This, for example, would include gambling games, like *Roulette*. In roulette, the

player's fate is in the hands of the spin of the wheel. In a similar sense, procedural techniques can have the same effect on the player. If a game chose to randomize player inputs, it would introduce so much performative uncertainty that it would frustrate almost everyone. Excessive uncertainty also includes poor design decisions. Costikyan (2013) mentions the camera system in Disney's *Epic Mickey* (2010), which would unexpectedly move around the player, drastically increasing the difficulty of the jumping segments and resulting in player frustration. Since the player doesn't have fluid control over the camera, frustration can result. Bad design can result from PCG, too. Generated levels can have no exit or be incredibly tedious to traverse. Difficulty could be dynamically tuned to be too difficult or too easy, introducing too much challenge to the player. Or, the generated systems could be needlessly complex, obfuscating the objectives of the game and confusing the player rather than providing tools to play with.

On the other end of the spectrum, if a procedural technique does not introduce enough variation, a lack of uncertainty arises. Salen and Zimmerman (2004) comment that if a game is completely certain, meaningful play is impossible. One of the dangers of using procedural generation is creating content that all feels similar. For example, in a game where the level generator creates levels that all look the same, players would perceive the same experience over and over, even with variation in the layout. This is why successful games that have used procedural generation either focus on the non-procedural elements or introduce special events to spice up the generative parts. Looking back to the case studies, *Left 4 Dead* (2008) made the levels exciting by dynamically creating spikes of high-intensity generation. *Spelunky* (2008) included rare level modifiers like removing the light from the cave or spawning a snake pit. A game like *Minecraft* (2009), however, uses procedural level generation to create a new blank

canvas for the player to explore, but the true focus of the game is creative crafting and building. Designers should evaluate a game on a case-by-case basis to determine what kind of procedural generation is necessary.

Even if the uncertainty of a game is properly tuned, it is useless without meaningful choices. According to Salen and Zimmerman, meaningful play has two definitions: first, it is a relationship between player action and system outcome, and second, it is what occurs when the relationships between actions and outcomes in a game are both discernable and integrated into the larger context of the game (2004). All that uncertainty does is provide a platform for meaningful play. Meaningful play itself is essential to the design of the game. It gives players context and purpose to the act of playing. Thus, for any game, it is essential to have fundamentally strong gameplay before the benefits of PCG can be applied in an impactful way.

Games of Chance

One of the most common associations of PCG with games is with randomness. Whenever probability is involve so is the concept of luck, or *alea* by Callois' terms (1958). Why is luck such a pervasive element in games? As shown in previous sections, uncertainty in games can be introduced in ways outside of a dice roll. The element of chance is one that is embedded in cultures throughout history and modern times. It goes beyond the vices of gambling or the lottery. To Callois, chance and competition, *alea* and *agon*, are paradoxically linked (1958). *Agon* represents authority, structure, and government (Caillois, 1958). The best player wins, and in a meritorious fashion the champion is crowned. However, the *alea* of life circumstances, Callois argues, provides a counter balance. "Each man is conditioned by environment. He may perhaps ameliorate conditions through merit, but he cannot transcend them. He is unable to

radically change his station in life. From this arises the nostalgia for crossroads, for immediate solutions offering the possibility of unexpected success, even if only relative. Chance is courted because hard work and personal qualifications are powerless to bring such success about" (1958).

Many people can associate with feeling of inadequacy in life. One can entertain the thought of buying a lottery ticket and never having to worry about money again. In this way, there is a comfort in a game of chance, even on a minute level. Being presented with a string of challenges that are followed by an easy encounter satisfies our need for feeling lucky, the essence of Caillois' *alea* (1958). In a way, it feels like cheating. For example, beating a level in *Spelunky* (2008) is normally an intensive and challenging feat. Occasionally, however, the path to the exit is direct and clear of hazards. In this way, the goal was achieved without hard work or much effort at all. Caillois suggests that *alea* "seems like the resistance posed by nature against the perfect equity of human institutional goals" (1958).

It is curious, perhaps, that uncertainty is such a source of anxiety in life, and a source of relief in games. For example, the threat of a car accident or a heart attack is ever-present in daily life. In many games, too, the threat of death and failure is pervasive. Costikyan (2013) suggests that human culture has taken uncertainty and transformed it into a series of elaborate constructs that present uncertainty in a fictive and nonthreatening way. PCG techniques in many forms embody this mantra, these elaborate constructs, the endless labyrinths of *Spelunky* (2008) or the endless hordes of zombies in *Left 4 Dead* (2008). For us to come to terms with reality - that not everyone is born equal, that bad things can happen, and most importantly, unexpected things can

happen - is essential to the development of culture. Procedural generation allows us to explore that space in a safe and equal manner, and it is a field that I hope continues to grow in the future.

Considerations for Future Work

This paper approached procedural generation and uncertainty from a high level. The application and development of procedural content generators is so vast that just the potential uses could be the subject of an entire paper. To provide some examples, it would be interesting to see further research in the fields of evolutionary algorithms. These algorithms have the potential to be particularly interactive with players, learning along with them to provide compelling interactions. Additionally, user-generated content is becoming more prevalent in the games industry. How will games, especially those with carefully constructed procedural algorithms, integrate user content in a seamless way?

Within the topic of uncertainty, it would be a logical next step to relate uncertainty to other perspectives on game design. How does a particular genre employ certain types of uncertainty? Perhaps a formalized case study on a game that includes advanced PCG techniques, like *Galactic Arms Race*, a game that takes user feedback to generate new weapons for space ships in real time (Hastings, 2009). Regardless of where the scholarship on uncertainty in games goes, its importance as a fundamental principle in game design makes it deserving of further attention and discussion.

REFERENCES

Adams, T & Adams, Z. (2006). *Dwarf Fortress* [computer software]. Bay 12 Games.

Avedon, E., & Sutton-Smith, B. (1971). *The Study of Games*. New York, New York: John Wiley & Sons.

Booth, M. (2009). *The AI Systems of Left 4 Dead* [PDF]. Retrieved from Valve Software: http://www.valvesoftware.com/publications/2009/ai_systems_of_l4d_mike_booth.pdf.

Caillois, R. (1958). *Man, Play and Games*. Urbana-Champaign, Illinois: University of Illinois Press.

Cellular Automata. From *Wolfram MathWorld*. Retrieved November 19, 2014, from http://mathworld.wolfram.com/Rule30.html. Copyright 2014 by Wolfram Research, Inc.

Conway, J. H. (1970). *The Game of Life*.

Costikyan, G. (2013). *Uncertainty in Games*. Cambridge, Massachusetts: The MIT Press.

Dark Souls [computer software]. (2011). Tokyo: From Software.

De Koven, B. (1978). *The Well-Played Gamed: A Player's Philosophy*. Garden City, New York: Anchor Press.

Epic Mickey [computer software]. (2010). Austin: Junction Point Studios.

Flitterer, D. (2008). *AudioSurf* [computer software]. Invisible Handlebar.

Francillette, Y., Gouaich, A., Hocine, N., & Pons, J. (2012). A Gameplay Loops Formal Language. *2012 17th International Conference on Computer Games (CGAMES)*, 94.

Gips, J. (1999). Computer Implementation of Shape Grammars. In *NSF/MIT Workshop on Shape Computation* (Vol. 55, p. 56).

Guitar Hero [computer software]. (2005). Cambridge: Harmonics.

Gygax, G & Arneson, D. (1974). *Dungeons & Dragons*. Lake Geneva: Tactical Studies Rules.

Half-Life [computer software]. (1998). Fresno: Sierra Software.

Hastings, E. J., Guha, R. K., Stanley, K. O. (2009). Evolving Content in the Galactic Arms Race Video Game. *Computational Intelligence and Games*.

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*, 9(1), 1-22.

Kazemi, D. (2013). *Spelunky Generator Lessons, Part 1: Generating the Solution Path*. Retrieved from: http://tinysubversions.com/spelunkyGen/

Kazemi, D. (2013). *Spelunky Generator Lessons, Part 2: Generating the Rooms*. Retrieved from: http://tinysubversions.com/spelunkyGen2/

Lagae, A., Lefebvre, S., Cook, R., DeRose, T., Drettakis, G., Ebert, D. S., Lewis, J. P., Perlin, K. & Zwicker, M. (2010). A Survey of Procedural Noise Functions. *Computer Graphics Forum*, 29(8), 2579-2600.

Left 4 Dead [computer software]. (2008). Bellevue: Valve Corporation.

Left 4 Dead Enemy Placement Map. Retrieved March 10th from http://www.valvesoftware.com/publications/2009/ai_systems_of_l4d_mike_booth.pdf. Copyright 2009 by Valve Corporation.

Left 4 Dead Intensity Graph. Retrieved March 10th from http://www.valvesoftware.com/publications/2009/ai_systems_of_l4d_mike_booth.pdf. Copyright 2009 by Valve Corporation.

Lehman, J., & Stanley, K. O. (2011). Abandoning Objectives: Evolution through the Search for Novelty Alone. *Evolutionary Computation*, 19(2), 189-223.

Liapis, A., Yannakakis, G., Togelius, J. (2014). Computational Game Creativity. *Proceedings of the Fifth International Conference on Computational Creativity*. 4(1).

Mateas, M., & Stern, A. (2003, March). Façade: An experiment in building a fully-realized interactive drama. In Game Developers Conference (Vol. 2).

McMillen, E & Himsl, F. (2011). *The Binding of Isaac* [computer software].

Meier, S. (1991). *Civilization* [computer software]. Hunt Valley: MicroProse.

Merrick, K. E., Isaacs, A., Barlow, M., & Gu, N. (2013). A Shape Grammar Approach to Computational Creativity and Procedural Content Generation in Massively Multiplayer Online Role Playing Games. *Entertainment Computing*, 4(2), 115.

Monopoly. (1903). Pawtucket: Hasbro Incorporated.

Nishikado, T. (1978). *Space Invaders* [computer software]. Chicago: Midway.

Perlin Noise. (2014). In *Wikipedia*. Retrieved November 19, 2014, from http://en.wikipedia.org/wiki/Perlin_noise.

Persson, M. (2009). *MineCraft* [computer software]. Stockholm: Mojang.

Picbreeder. (2014). From *Picbreeder*. Retrieved November 19, 2014, from http://picbreeder.org/. Copyright 2007 by University of Central Florida.

Portal [computer software]. (2007). Bellevue: Valve Corporation.

Salen, K., & Zimmerman, E. (2004). *Rules of Play: Game Design Fundamentals*. Cambridge, Massachusetts: The MIT Press.

Commented [TF1]:

Secretan, J., Beato, N., D'Ambrosio, D. B., Rodriguez, A., Campbell, A., Folsom-Kovarik, J. T., & Stanley, K. O. (2011). Picbreeder: A Case Study in Collaborative Evolutionary Exploration of Design Space. *Evolutionary Computation*, 19(3), 373-403.

Shaker, N., Togelius, Nelson, M. J. (2015). *Procedural Content Generation in Games: A Textbook and an Overview of Current Research.* Springer.

Smelik, R. M., De Kraker, K. J., Tutenel, T., Bidarra, R., & Groenewegen, S. A. (2009, June). A Survey of Procedural Methods for Terrain Modelling. *Proceedings of the CASA Workshop on 3D Advanced Media in Gaming and Simulation (3AMIGAS)* (pp. 25-34).

Smith, J. (2014). *Desert Golfing* [computer software]. Vancouver: Captain Games.

Spelunky Level Map. Retrieved March 14th from http://tinysubversions.com/spelunkyGen/.

Spelunky Room Template. Retrieved March 14th from http://tinysubversions.com/spelunkyGen2/.

Stiny, G., Gips, J. (1972). Shape Grammars and the Generative Specification of Painting and Sculpture. In C.V Freiman (Ed.), *Proceedings of IFIP Congress 71*. Amsterdam: North-Holland, 1460-1465.

Togelius, J., Yannakakis, G., Stanley, K., & Browne, C. (2011). Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3), 172-186.

Toy, M. & Wichman, G. (1980) *Rogue* [computer software].

Van der Linden, R., Lopes, R., & Bidarra, R. (2014). Procedural Generation of Dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(1), 78-89.

White Noise. (2014). In *Wikipedia*. Retrieved November 19, 2014, from http://en.wikipedia.org/wiki/White_noise.

World of Warcraft [computer software]. (2004). Irvine: Blizzard Entertainment.

Yu, D. (2008). *Spelunky* [computer software].