The application of procedural content generation in video game design

University of Oulu
Department of Information Processing Science
Bachelor's Thesis
Henri Bomström
01.04.2016

Contents

1. Introduction

Video games have started to become a daily part of our lives. In 2015 the Entertainment Software Association reported that 155 million Americans play video games and 42 percent of those play video games regularly, spending at least three hours per week playing. In 2015 the American video game industry generated $22.41 billion in revenue. (Entertainment Software Association [ESA], 2015.) As the demand of content in video games continues to rise, the amount of resources used by game studios is increasing to meet the demand. The scaling of manual content creation in video games is becoming a problem and automatic content creation has been suggested to increase scalability (Iosup, 2011.) Procedural content generation tries to solve this problem by programmatically creating varying and personalised content on demand.

Procedural content generation is intended to create content automatically in video games, or help in the creation process of content. The applications of procedural content generation in game design and development ranges from integrated solutions that automatically generate content, to pure inspiration and tool applications. (Togelius, Yannakakis, Stanley, & Browne, 2011.) Smith, Gan, Othenin-Girard and Whitehead (2011) have also proposed that video games can be procedural content generation-based, which creates new playable experiences through extensive usage of procedural content generation. This means that a completely new type of games is possible, with game mechanics built around content generation.

Traditionally procedural content generation has been used in single sub-categories of game design, such as map generation to increase replayability. The problem with procedural content generation is that the content might not always be desirable or enjoyable. If the content generated is not for example playable at all, it is considered as a catastrophic failure. Catastrophic failures are one of the reasons why procedural content generation isn't more widely used in commercial game development, as it may render the game impossible to complete. (Zafar & Mujtaba, 2012.)

The usage of procedural content generation in video games is still not a very well researched subject, even though it has been used for a long time. This means that potential applications of PCG in the gaming industry might not have been all realized yet. (Togelius et al., 2011.) Hendrikx, Meijer, Van Der Velden, & Iosup (2013) released a comprehensive survey on procedural content generation and found that the related literature is distributed over multiple disciplines and co-evolves with multiple different areas in science. According to Yannakakis and Togelius (2014) PCG is an active research area and is heavily influenced by other AI research areas. Besides player modelling, PCG was found to be research area that affected the player's gameplay experience the most.

Even though procedural content generation has raised significant research interest in recent years, it has not been yet fully adopted to the practise of game design and should be contextualized to a more design centric perspective. (Khaled, Nelson, & Barr, 2013.) According to Smith (2014) it is essential that both the AI researchers and game designers have a common vocabulary when discussing procedural content generation systems. The focus of this thesis is to study PCG and its relation to game design. This thesis tries to answer the questions *what is Procedural Content Generation* and *how Procedural Content Generation is used in game design*. The research method selected is a literature review, where no new empiric knowledge is created. When referring to PCG and its related methods, the context is limited to video games. The in-depth study of different procedural methods, such as Perlin noise and L-systems, is left out of this study.

The structure of this thesis is built on studying PCG and then exploring its application on video game design. The second chapter defines procedural content generation, search based procedural content generation, experience driven procedural content generation and the terminology related to them. The third chapter continues on the second chapter and explains why and how procedural content generation is used in game design. Finally discussion about the topic is presented with the conclusions.

2. Procedural Content Generation

The traditional way of producing content in video game projects is to use programmers and artists to manually create it. As the demand for quality content and the size of projects grow, the traditional model faces problems in scalability. Rising production costs of content take resources from developing other features of the game and eventually the added costs fall on the consumer. (Kelly & McCabe, 2006.) PCG has been suggested as a solution to this problem, as it can be used to ease the burden of game content designers while they still retain control on the design process. Togelius et al. (2010, 2011) describe procedural content generation (PCG) as creation of game content automatically, through the means of algorithms. The application of procedural content generation techniques is still not very common in commercial games, because the existing techniques are not general enough to use consistently. (Hendrix et al., 2013.)

This section begins by explaining the concept of game content and it's relation to PCG. Following that, PCG is divided into distinct categories where the core concepts of PCG, search-based PCG, and experience-driven PCG are introduced.

2.1 Game content

Togelius, Yannakakis, Stanley, & Browne (2010) define game content in the context of PCG as all aspects of a game that have some effect on gameplay, excluding NPC behaviour and the game-engine itself. Liapis, Yannakakis and Togelius (2014) note that games can be seen as multi-faceted entities, where content ranges from visuals and sounds to plot. Each of these facets affects the player's gameplay experience differently and have to be thought of as a whole.

Hendrikx et al. (2013) have surveyed the field of PCG and classified procedurally generatable content into a hierarchical model composed of six main categories; game bits, game space, game systems, game scenarios, game design and derived content. As can be seen in Figure 1, the classes of content are ordered by complexity, where each layer can be built from the layers below it. Game bits are used as basic building blocks of a game, but never usually independently act with the player. Game bits are placed in game space, a player navigable environment. Game systems are collections of organized things, for example a road network, that can simulate parts of the game world or act as structure for environments. Game scenarios are used to drive the player further in the game and determine in which order events are presented to the player. Game design refers to the design of the game, such as rule sets and goals. Derived content is the by-product of the game world and its events, and can be also known as the metagame. (Hendrikx et al., 2013.)

Hendrix et al. (2013) found that the level of maturity in PCG techniques related to specific content category layers have an inverse relationship. This means that methods used to generate content on the lower level of Figure 1 have reached a higher level of maturity than the methods used to create content in higher categories. Hendrikx et al. (2013) also define categories of methods, called fundamental methods that can be used to generate content across multiple content categories. These categories are pseudo-random number

generators, generative grammars, image filtering, spatial algorithms, modelling and simulation of complex systems, and artificial intelligence.

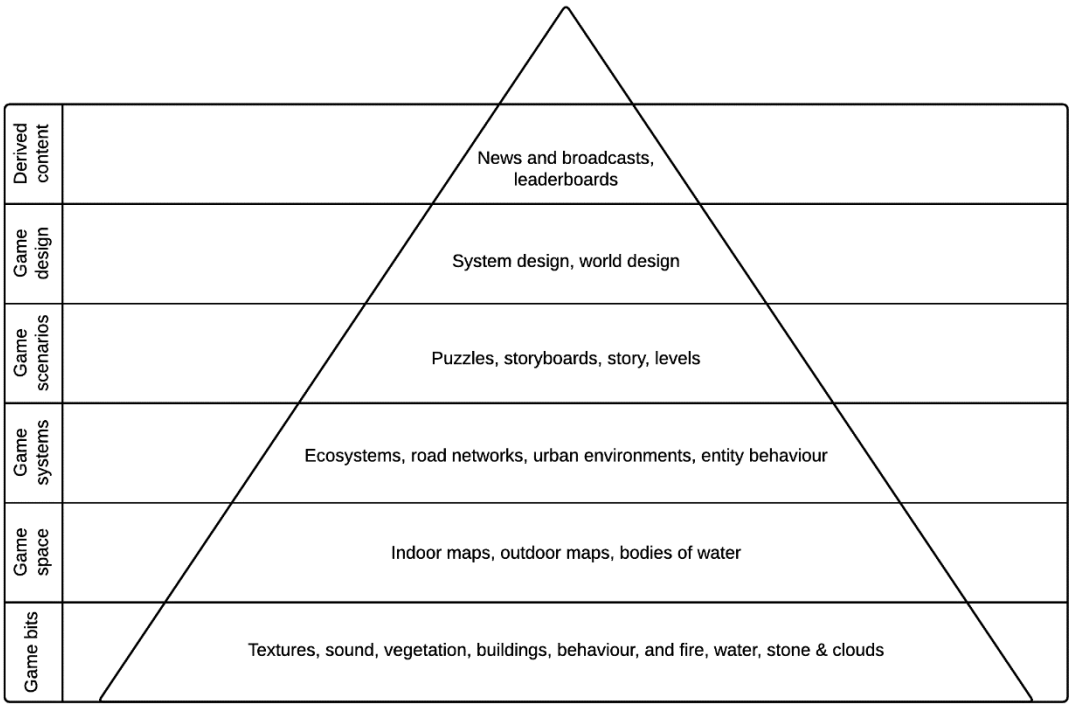

Figure 1 Game content types by Hendrikx et al. (2013).

Yannakakis & Togelius (2011) describe that currently PCG is almost exclusively used in specialized roles and almost always in game development phase. The reason for this is that not all types of content can be generated with the wanted quality, variance and reliability using traditional techniques. Another reason is that the traditional PCG techniques are not deemed controllable enough, resulting in content that doesn't necessarily fit in with the rest of the game.

Merrick, Isaacs, Barlow, & Gu (2012) have also found that existing work on PCG usually focuses on specialized subsets of game content, such as game ecosystems or story. These specialized subsets also define content as subsets of wider game content taxonomies as a base for PCG. (Merrick et al., 2012.)

In their book Angelides & Agius (2014) describe that PCG systems are ideal for generating content with a unifying visual theme, game mechanic or design aesthetic. They identify the most common key content production areas for PCG as terrain generation, architecture, and narrative and plot. Terrain and architecture are usually one of the largest production elements, which makes them very appealing for automation and compression offered by PCG.

The necessity of game content can be used to further divide content into distinct categories. Necessary content is essential for the player to progress in the game, such as mandatory dungeons and enemies, and the most important game rules. The player can avoid optional content without it affecting essential gameplay. The difference with these two categories is that necessary content always needs to be correct, otherwise making it impossible to complete the game. Optional content can be unusable, since the player can opt not to interact with it. (Togelius et al., 2010, 2011.)

Catastrophic failure is the concept of unplayable content. The player is presented with something that he cannot pass to continue advancing in the game. Zafar and Mujtaba (2012) identify catastrophic failures from offline generation in “Infinite Mario”. Some of these failures are not game stopping, for example if the content is aesthetically displeasing or too simple. While catastrophic failure can be seen as a show-stopper in necessary content, optional content can tolerate a critical failure. The critical failure will still render the content unusable and affect the player's opinion of the game.

Togelius et al. (2011) also note that the definition of necessary game content varies on a per game basis. Some games will require a content type to be optional, when it is necessary in other types of games. The degree of tolerance for content failure also varies, since different games tolerate content failures depending on their game setting.

2.1 Categorizing Procedural Content Generation

PCG has been around for a long time. Elite (Acornsoft, 1984) has been regarded as one of the first games making use of PCG, representing a vast amount of unique galaxies and trade items as parameter values. This allowed game content to be represented in very compact way, overcoming hardware restrictions at the time. According to Togelius et al. (2010) all PCG algorithms create content by expanding a smaller representation of it. Parameters can be used to control the properties of generated content, and they can range from a single seed value to a multidimensional vector of numbers.

Games such as Rogue (Toy, Wichman, Arnold, & Lane, 1980) feature level generation that has sometimes been regarded as random generation. In the context of PCG, the word random shouldn't mean that content is created and distributed without any consideration for structure. These levels would be unplayable, and most content generators feature strong constraints on what can be generated. Between the constraints, generated content can differ according to some pseudo-random process. (Togelius et al., 2011.)

Stochasticity is the aspect of randomness in the generation process, as it determines how much the content varies between runs with identical parameters. The stochasticity of an algorithm is a design question and most of PCG methods are not deterministic. The generation process can also be completely deterministic, making it behave as a packaging method seen in the game .kkrieger (Farbausch, 2004), which uses deterministic methods to pack game content into a very small fraction of the normally required space. Another one of the most famous usages of PCG can be found in the game Rogue (Toy, Wichman, Arnold, & Lane, 1980), which featured nondeterministic generation of dungeon levels, providing a different player experience each time the game was played.

Togelius et al. (2010) continue dividing PCG into categories by separating PCG by the generation time of content, forming two distinct categories known as online and offline generation. Online generation creates content during the runtime of the game, while offline generation usually takes place during the game development or loading screens. Intermediate solutions combining online and offline generation are also possible, such as generating the content offline but making it react to player performance. (Togelius et al., 2010.)

The final division of PCG by Togelius et al. (2010, 2011) is the distinction of algorithms using constructive or generate-and-test methods. Constructive methods don't perform any search to find good candidate solutions, but instead create the content once, using operations that by the rules will not produce broken content. Constructive methods have a predictable and usually short runtime, but lack the ability to create varying and rich

content without the risk of generating unusable content known as catastrophic failure. The generate-and-test method validates the content after it has been created against some chosen criteria. If the content, or a part of it, doesn't fulfill the criteria, it is discarded and the process will start from beginning. This is repeated until the assessment criteria is met. (Togelius et al., 2010.)

2.2 Replayability and adaptability

Rogue (1980) has been considered one of the first games to utilize PCG, providing the player with new levels on each play through (Smith et al., 2011). Most of these earlier games featuring PCG used it as a way to improve replayability, having more worlds or levels that a human designer could author given the timetable and hardware constraints. Additional and more modern examples of such games include Diablo (Blizzard North, 1996) and Civilization IV (Firaxis Games, 2005). (Smith, Whitehead, & Mateas, 2011.) These games use PCG in the start of the game to provide new experiences for players. Even while the games feature PCG in an important role of player experience, large portions of their player strategies remain mostly unaffected by the content generator. The player strategies are more dependent of the static rule systems defined by the human designer. Smith notes that there are games where replayability is a central element of the play experience. Without the PCG system players would choose different strategies and the goal of the game would be different. She gives an example of Robot Unicorn Attack and Cananbalt, 2D platformers where the content generation mechanism relies on producing new and surprising content for each individual play through. (Smith, 2012.)

Adaptability is a related concept to replayability, where PCG is used to adjust content in relation to player skill or actions. Adaptability can improve replayability and potentially diversify the player base by encouraging different play styles or skills. (Smith et al., 2011.) Togelius et al. (2011) describe adaptive or player driven PCG as a form of parametrizable PCG, where the parameters depend on previous player actions. They note that adaptable PCG has not been used almost at all in commercial games. While adaptation can offer much, it can also lead to difficult to detect failures in game design. For example, if the player gets stuck in an adaptive game, he may not be able to find information from wikis and other shared knowledge bases on how to get unstuck. (Smith, A., 2012.)

Togelius et al. (2011) also argue that adaptable PCG might not meet the definition of being PCG at all, depending on the timescale and locality of adaptability. Intention is the key term that determines the distinction. If the player is able to predict the generated content he can use the adaptation mechanism as a gameplay device instead. The authors argue that in this case, adaptation can't be called PCG anymore. (Togelius et al., 2011).

2.3 Search-based Procedural Content Generation

Search-based PCG (SBPCG) is a special case of the generate and test method. The most important distinctions are that the generated content is evaluated and assigned a fitness value, using one or more real numbers. Generation of new content aims to create better instances of previously generated ones, using the fitness value for comparison. Most SBPCG methods use evolutionary algorithms as their search mechanism. Other stochastic and metaheuristic search techniques can also be used, since SBPCG doesn't have to be tied into evolutionary computation. (Togelius et al., 2010.)

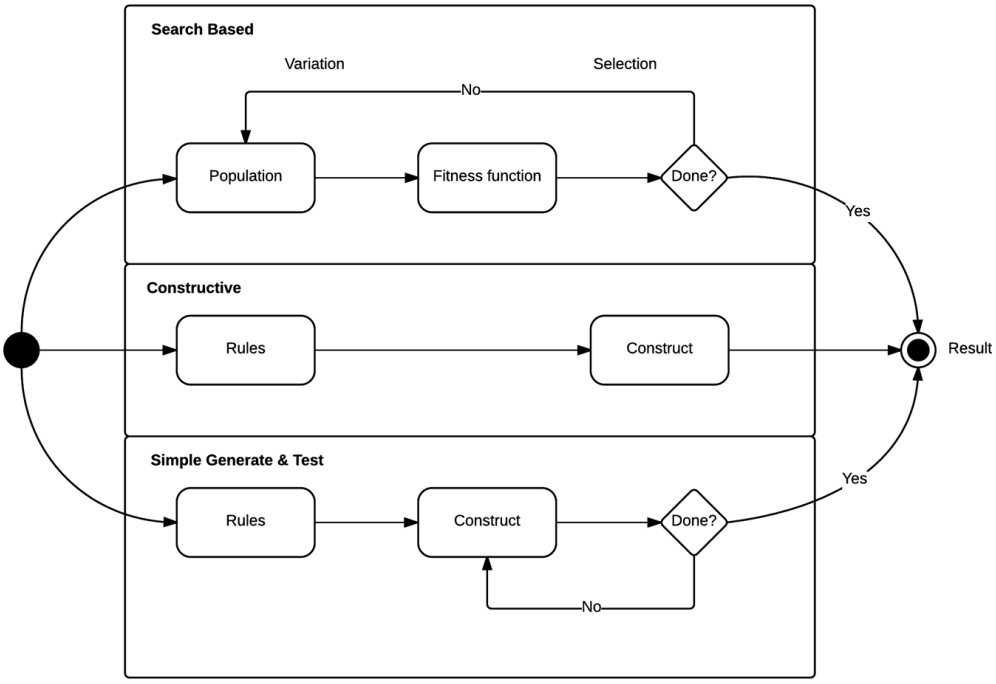

Figure 2 Types of PCG by Togelius et al. (2011).

2.3.1 Search space and content representation

According to Togelius et al. (2011), evolutionary algorithms are the most dominant choice of search algorithms when it comes to SBPCG. In the context of evolutionary algorithms, Mitchell (1998) defines search space as a collection of candidate solutions to a certain problem with a measurable distance between them. Candidate solutions are evaluated and assigned a fitness value, which tells how well the candidate solution solves the problem at hand. The term fitness landscape is defined as a representation of all possible solutions and their fitness values.

When using genetic and evolutionary algorithms for optimization problems, some sort of representation is required in order to encode all the solutions. Evolutionary algorithms mimic nature by applying genetic operations such as selection, crossover and mutation to individuals. Nature also makes a distinction between the genetic representation of an individual, which is called a genotype, and the outward appearance of an individual, called the phenotype. Chromosomes are used to encode individuals in gene level and often uses more than one chromosome, but most evolutionary algorithms use one chromosome to encode the genotype. A chromosome is a string that stores all the genetic information of the individual. Alleles are the smallest pieces of the chromosome, and describe some properties of the individual. If the phenotypic property of an individual is represented by one or more alleles, the group of alleles is then called a gene. (Rothlauf (2006))

An individual in a population is evaluated based on it's phenotype, but when it reproduces it's phenotype is not inherited, but only the genotypic information related to its phenotypic properties. This means that genetic operations are performed on the genotype, while the

evaluation of the individual is performed on the phenotype. Every fitness function that evaluates a genotype can be decomposed into genotype-to-phenotype and phenotype to fitness mappings. When all the genotypic properties are transferred to the phenotype, the mapping is said to be direct. If the genotype won't map linearly to the phenotype, the mapping is said to be indirect. The term locality describes how neighbouring genotypes correspond to neighbouring phenotypes. If the locality is high, small changes made to the genotype should result in small changes to the phenotype. If the neighbouring phenotypes won't correspond to the neighbouring phenotypes, the locality is then low. Low locality results in low performance with evolutionary algorithms, making high locality of content representation a desirable feature. High locality makes neighbouring phenotypes have similar properties, making their fitness values correlated. If the locality is low, evolutionary algorithms won't be able to properly use crossover and mutation, since the offsprings of individuals won't be similar to their parents. (Rothlauf, 2006.)

In SBPCG the representation of content and the size of search space must be coordinated with the problem at hand. Direct encodings with high locality can suffer from the curse of dimensionality, as genotype gets larger. On the other hand indirect encodings handle curse of dimensionality well, but lack the locality and thus making it harder for evolutionary algorithm to search for good solutions. (Togelius et al., 2011.)

Since many content generators are nondeterministic, the term generative space or design space is proposed to define the group of artefacts the generator will eventually produce, given some input. Smith and Mateas (2011) claim that PCG consists of two main problems related to design space. The first problem is the generation of content, which should be done automatically by machines, using some search method. The second one is the meta-level problem of production of generative procedures with desirable properties that should be solved by human designers. These problems coalesce into the central problem of selecting a design space simple enough to sample with algorithms, while at the same time containing only the desired artefacts. (Smith & Mateas, 2011.)

Smith and Mateas (2011) note that the design space of procedural content generators is often only implicitly defined and suggest an alternative way of approaching PCG by explicitly defining the design space for generation. In the context of game design, the problem with undefined design space is that it might produce undesirable results, negatively affecting player experience. They also mention that PCG naturally inherits a design problem of ill-defined requirements, the lack of absolute knowledge of what exactly the designer is looking for. To counter this, Smith and Mateas used Answer Set Programming (ASP) to generate the design space. ASP uses logical statements that must be fulfilled in the answer set, producing all the combinations of content that fulfil the requirements of the designer.

2.3.2 Basic structure of evolutionary algorithms

The generic structure of a genetic algorithm is usually the following. First an initial population of candidate solutions is created and all the individuals in the population are assigned a fitness value. When the entire population is evaluated, the population starts to reproduce. First the two parent chromosomes are picked using the method called selection. Selection means selecting certain chromosomes from the population based on their fitness value. The most fit organism is not guaranteed to reproduce, but is the most likely to. A single chromosome can also be selected multiple times. The two parents then produce two offspring with the crossover method. Crossover switches genetic material between two chromosomes from a certain locus, which means the position of the allele in the chromosome. After the offsprings are created, mutation is applied to them. Mutation

changes bits from each locus in some probability, which is usually very low. These reproductive steps are then repeated until the new population is of a certain size. After that, the old population is replaced with the new population and the process is iterated as many times as required. Each of these iterations is a new generation of solutions. (Mitchell, 1999, p. 8-9.)

2.3.3 Evaluation functions

When a candidate solution for a problem is generated, it has to be assigned a fitness value. The fitness function must reflect the usefulness of content in the specific context. Designing the evaluation function is difficult, since the designer must first know exactly what to optimize and then how to formally represent it. In the context of PCG, three main types of fitness functions exist. (Togelius et al., 2010.)

The first type is the direct fitness function, where some features of the generated content are mapped directly to the fitness value. This mapping doesn't have to be linear, but won't typically require large amounts of computation. The mapping is usually tailored to specific games or content types. The mapping might also take into account player preferences or other characteristics, making it possible to personalize content. Direct evaluation is further divided into two subcategories, called theory-driven and data-driven functions. The theory driven function is guided by some qualitative metric derived from the theory of player experience, or just plain intuition. Data-driven functions operate on collected data, what can be for example collected from questionnaires or physiological measurements. The features to fitness mapping is then tuned by some automated process. (Togelius et al., 2010.)

The second type of evaluation functions for PCG is the simulation-based function. This approach is useful when no direct evaluation method can be properly used to evaluate content, as it might require further interaction to be fully understood. As the name suggests, a simulation using an artificial agent is run that interacts with the generated content. Content is evaluated based on features extracted from the performance of the artificial agent. This type of evaluation function can be further divided into static and dynamic simulation-based evaluation functions. Static evaluation assumes that the agent playing the game doesn't change while playing. Dynamic evaluation takes into account changes in the agent and reflects those changes to the final fitness value. This can be used for example to measure learnability of the content. (Togelius et al., 2010.)

The third type of evaluation functions for PCG is the interactive evaluation function. This sort of evaluation function directly interacts with the player during gameplay. The data for the evaluation can be gathered explicitly, by using questionnaires or input data, or implicitly by measuring for example how often the player interacts with certain content. (Togelius et al., 2010)

2.4 Experience driven procedural content generation

As the general gamer population continues to grow and diversify, games will need to cater to a wide range of players with different skill sets, preferences and emotions. In order for games to offer the same gameplay experience to all of the players, personalised game content will be needed to optimize the individual gameplay experience. Yannakakis & Togelius (2011) introduced the framework of Experience Driven Procedural Content Generation (EDPCG) as a way of optimizing player experience in games, driven by computational models of player experience. (Yannakakis & Togelius, 2011.)

Games offer rich forms of interaction and induce great amounts of varying emotional responses from the player (Yannakakis & Togelius, 2011). Affective computing deals with the concept of affective loop, which involves the player expressing their emotions through some physical interaction involving their body with an artefact, such as a keyboard or a mouse. The system then responds to the player through affective expressions, such as images and colours. These affective expressions then elicit emotions on the player, making them feel more involved with the system. (Höök, 2008.) In order to successfully close the affective loop, games should be personalised to individual affective responses, have fast adaptation and feature rich forms of affective interaction (Yannakakis & Togelius, 2011.).

In the context of EDPCG, Togelius et al. (2011) take a bit different approach to game content when compared to other types of PCG. They describe that games are built from content, while acting as enablers of player experience. This indirect relation makes game content a building block of player experience. When generating content that is optimized for player experience, the main components of an EDPCG content generator are player experience modelling, content quality, content representation and the content generator. Player experience model is given as a function of game content and player, the latter measured with play style, cognitive and affective responses to gameplay stimuli. Content quality is estimated and linked to the modelled player experience profile. Content is represented with content generator efficiency, performance and robustness in mind. The content generator searches the content space while trying to optimize the player experience related to the modelled player model.

Player experience modelling has three main approaches to data gathering for player experience model creation; subjective, objective, and gameplay-based player experience modelling. Subjective player experience modelling relies on players reporting their experiences through free-response while playing, or by forced data retrieval through some sort of forms. Subjective PEM doesn't rely on external observers or experts for modelling. Free-form data gathering results in rich feedback, but is hard to analyse. Forced data gathering can be simplified to answering to ratings and preference evaluations. Objective PEM measures emotional responses by measuring physiological signals, facial expressions or speech. This approach can be model-based or model-free. Model-based approaches are built on emotion theory, which map certain bodily signs to emotions. Model-free PEM constructs and unknown mapping between data and bodily signs, and are mostly built with statistical analysis. Gameplay-based PEM builds models from player actions during gameplay and can be model-based, model-free or a hybrid between them. (Yannakis & Togelius, 2011.)

Player models are mainly used for evaluating the quality of generated content. The quality of content in the context of EDPCG is measured by evaluating how suitable the generated artefact is and it's capability of producing the desired affective state. The evaluation functions used should also be able to reflect how well each part of the content contributes to eliciting each affective state. (Yannakis & Togelius, 2011.) EDPCG uses the same evaluation functions as SBPCG.

Creating content with player models needs to take into account the questions of how much and how often the person wants content to be generated for him. Some players dislike adaptive generation, while other enjoy it. These differences need to be distinguished and somehow dealt with. The generation process acts as a closed loop, where player experience represents feedback given from content. EDPCG is challenging since the emotional response provided by certain content is hard to evaluate before actually creating it and receiving feedback from the player. (Yannakakis & Togelius, 2012.)

3. Applying procedural content generation to game design

The motivation of generating content in games has largely been unchanged during decades, featuring common themes in replayability, surprising the player, and providing variety (Smith 2015). When compared to conventional games, the development process of a game featuring PCG differs in the aspect that some gameplay experiences may not be hand authored any more. The designer grants the PCG system some autonomy in making decisions related to content generation. (Khaled, Nelson, & Barr, 2013.)

Smith et al. (2011) note that the border of a PCG-based and a non PCG-based game is unclear. They describe the differences as a spectrum where games can be measured according to their relation to replayability, adaptivity, player's control over content, and game mechanics and dynamics.

This section addresses the design related aspects of developing games with PCG. First a brief history of PCG is introduced. Following that, PCG is examined with its relation to game design, how it can be used in game design and what sort of challenges can be expected from using PCG. Finally the concept of mixed-initiative co-creation is introduced, giving insight on combining human and computational creativity.

3.1 Analog procedural content generation

According to Smith (2015) PCG has been considered as a feature of digital games. She argues that if PCG is construed in a more broad way, it only means that content is generated by following a formal procedure. Thus the entity performing the generation does not necessarily need to be a computer. Smith has analysed the analog history of PCG, studying where the foundations of digital games lie and how they have affected PCG in digital games. Earlier motivations of PCG, besides replayability, featured important aspects in aiding creativity. Table top roleplaying can be seen as an example of PCG aided creativity, where the Dungeon Master (DM) needed to act as an on-demand game designer. When early digital games emerged, the use of PCG received two more motivational aspects; the use of computers as unbiased agents and making multiplayer games available for single player mode. (Smith, 2015.) Randomness has been considered as a core feature of PCG, and as such most PCG games are not deterministic. From the perspective of game design uniform randomness in procedural systems should be substituted by more directed approaches. Smith gives two examples of directed randomness; careful authoring of individual instances of content used in the creation process and altering the creation algorithm. (Smith, 2015.)

PCG systems should take care that the content produced is playable. Analog games enforced this with modular design, where all pieces fit together with each other. Smith (2015) gives an example of modular design by describing “Dungeon geomorphs” from Dungeons & Dragons, where different sized tiles were used to create dungeon layouts. The alteration of the creation process is an alternative approach to directing randomness in content creation. In this approach complexity is layered in the randomly-driven creation algorithm. An example of this approach is used in the terrain generation of Advanced Dungeons and Dragons (AD&D), where multi-layered lookup tables and dice rolls are used to create varying game content. Smith also gives examples of games where randomness is substituted or augmented with player control, such as tile-based games

Carcassonne and Betrayal at house on the hill. The player co-determined generation procedure resembles that of games using PCG. (Smith, 2015.)

Smith notes that older and current usages of PCG still face the same problem of creating meaningful play experiences. Formalism and rigid rules combined with random content generation take a toll on player experience. PCG system should try to take into account the qualities of generated content regarding player experience. (Smith, 2015.)

3.2 Procedural content generation and the Mechanics, Dynamics and Aesthetics framework

Smith (2014) has introduced a conceptual framework based on the mechanics, dynamics and aesthetics (MDA) framework by Hunicke, LeBlanc and Zubek (2004) for understanding the role of PCG in game design. She notes that it is important for AI and game design researchers to have a shared vocabulary to communicate what PCG is and what unique experiences it can offer to game design. PCG research has focused on the aspect of how PCG systems create content, while game design research is focused on the play experience provided by the game system. Smith argues that in order to understand the role of PCG in games, the chosen framework should include both the game design and player experience viewpoints. Smith's framework tries to bind these goals together and offers ways in understanding the role of PCG in game design and the unique benefits it has to offer.

The MDA framework has been proposed by Hunicke et al. (2004) as a formal approach to game design and research. The framework treats games as artefacts produced by designers to be consumed by players. The authors emphasize that consuming games differs from consuming other entertaining products, such as books and movies, by being more unpredictable. Events and their outcomes are unknown before actual gameplay, making behaviour built by interaction primary content rather than the media that is streamed towards the player. (Hunicke et al., 2004.)

Consumption of games is formalized by breaking it to components; rules, system, and fun. These components of consumption are mapped to their game design counterparts; mechanics, dynamics, and aesthetics. The MDA framework defines game mechanics as the various actions, behaviours, and control mechanics made available to the player in the game. When game mechanics are combined with game content, they form a supporting base for gameplay dynamics, which reflect the dynamic behaviour of the system based on the underlying game mechanics. Aesthetics are defined as "desirable emotional responses evoked in the player, when she interacts with the game system." Instead of describing aesthetics with vague terms like "fun", the MDA framework divides aesthetics into multiple goals including sensation, fantasy, narrative, challenge, fellowship, discovery, expression, and submission. The combination of these aesthetic goals, each creating their respective player experiences, help describe games and reveal why different games appeal to different players. (Hunicke et al., 2004.)

Figure 3 PCG's relation to Game mechanics and dynamics by Smith (2015).

Mechanics, dynamics and aesthetics can be thought of as layers built on each other, and changes in one layer reflects to the other two. The framework promotes iterative movement between the three design aspects to observe how changes in a specific layer affect the other layers. According to Hunicke et al. mechanics, dynamics and aesthetics can be viewed from both designer and player viewpoints. To a designer mechanics are the platform that produces game dynamics, which in turn lead into certain aesthetics. From a player perspective, aesthetics set the overall feeling of the game that stems from underlying game dynamics born from underlying game mechanics. Hunicke et al. note that thinking about the player perspective promotes experience-driven development instead of more static feature-driven approaches. (Hunicke et al., 2004.)

Smith divides PCG's game mechanical aspects into four categories; building blocks, stages of game, player interaction types, and player experience. The categories represent concerns from both PCG and game design researchers, answering questions such as how the generator works and how it can be used. Game dynamic aspects of PCG in Smith's framework consist from PCG's relation to other mechanics, memorization vs. reaction, strategizing, searching, practising, and interacting. These dynamics help to understand how PCG based mechanics enable different game dynamics, and what kind of unique dynamics are provided by PCG. As for aesthetics, Smith's framework includes discovery, challenge, and fellowship, which can be also found from the original MDA framework. Smith notes that the aspects and properties presented in the framework are not meant to be mutually exclusive, a game might have multiple kinds of control over its PCG system depending on the context. This framework by Smith can be seen in Figure 3.

In the context of game design, Smith (2014) claims that replayability is common, and overly broad, justification for PCG in games. Smith examined replayability on how it affected play experience and found three primary game dynamics that lead to different kinds of replayability; reacting in surprising environment, building generator strategies,

and practising in different environments. The dynamic of reacting in surprising environments leads to replayability where the game features new content on each attempt, usually in order to beat previous high scores. Practising in different environments leads to replayability by providing new content that supports replayability with more traditional mechanics. Both of these dynamics can be achieved with large amounts of hand authored content, but final dynamic of building generator strategies can only be achieved through the use of PCG. The building generator strategies dynamic is described as experiencing new content on each play through and having the opportunity to form strategies around it. (Smith, 2014.)

3.3 Approaching procedural content generation from a game design perspective

Even though PCG has raised significant research interest in recent years, it has not been yet fully adopted to the practise of game design (Khaled, Nelson, & Barr, 2013). Togelius, Justinussen and Hartzen (2012) describe the form of a PCG solvable problem as generating one or more content artefacts for a given game, player and context, while being good enough when evaluated by certain criteria. They also mention, that similar to other problems, PCG problems can be represented as search problems. Instead of viewing PCG as an artificial intelligence related problem of a computer generating content to a game, Smith (2015) notes that PCG problems can be viewed as problems of formalizing design processes. Smith also notes that PCG overlaps with game design, as it "… implicitly encodes a formal theory for both the game design process and the product that is being procedurally created."

3.3.1 How procedural content generation can be used by game designers

Khaled et al. (2013) argue that PCG has to be contextualized to a more design-centric perspective, as opposed to AI or engineering perspectives. They tried to bridge the gap between PCG and HCI communities by providing design metaphors that describe the potential relationships between PCG and a designer. The four metaphors for usage of PCG in game design are the tool, material, designer, and expert metaphors. Each of these metaphors describe PCG by its relation to design process and not the actual technical approaches, instead trying to answer the questions of who and what PCG is for. (Khaled et al., 2013.)

The first metaphor is the tool metaphor. If viewed in this context, PCG is used as a tool to reach a certain game design goal by extending or enhancing the abilities of the designer. The tool and designer take an active role in creating or changing content, but the final responsibility of using the produced content lies with the designer. This means that the designer has to know whether or not to use PCG techniques for the specific design goal. Monitoring content quality with tool usage is easier when done offline, when the designer better control over the content. If used with online content generation, the game should incorporate periods for monitoring the produced game content, in case it needs to be revised. (Khaled et al., 2013.)

The second metaphor is the material metaphor. Materials can be seen as reconfigurable and dynamic substances that the designer can deploy or mould. In the context of PCG, materials can also perform or be the target of computation. Materials are usually in a passive role, being acted on. Materials are primarily generated offline or just in time for designer revision, since the eventual responsibility of material usage always falls to the

designer. The designer is also responsible for articulating a set of property restraints that describe the set of desirable material from the design perspective. (Khaled et al., 2013.)

The designer metaphor is used for PCG algorithms that are given the task of autonomously solving certain design problems or tasks with little or no need for the designer to intervene. The amount of autonomy granted to the designer makes it take either the role of an assistant designer, or the lead designer. Assistant designer solves problems in smaller design spaces, mostly pre-determined by the human designer. Leading designer takes care of most of the designing, and the human designer takes the role of managing the leading designer. The chosen role of the designer depends on multiple factors, the most practical being the ability to create all the relevant content for the design task. The amount of trust also affects the chosen role of the designer. There might not be any guarantees that the designer can consistently produce good content that will also live up to the standards of the human designer. The opacity of the underlying generation methods can affect the trust relation of the human designer, since it might be very difficult to predict what kind of content, for example, a neural network might produce. A more intermediate solution given to this division is the co-designer approach, where the human designer works together with the PCG designer. (Khaled et al., 2013.)

The final design metaphor is the commonly used expert. The expert is compared to the AI-domain's expert systems, but more focused on solving game related problems rather than decision-making. The expert usually analyses and interprets data gathered from gameplay. The adaptation of game experience falls under the designer metaphor, while the expert provides something beyond regular game design objectives. The designer focuses on creating solutions to actual game design problems and the expert provides input to be taken into account while trying to answer game design related problems. This mean the designer and the expert work in tandem. The expert can be divided into two subclasses called the player expert and the domain expert. As per it's name, the player expert focuses on analysing data related to the player's experience. (Khaled et al., 2013.)

When PCG is used to create game content, the human designer gives up some control of the design process to the PCG system. Depending on the level of autonomy of the PCG system, high levels change the designer's role more into choosing, editing and organizing the outputs of the PCG system. Current uses of PCG won't necessarily map correctly to the needs of typical game designers, since most PCG systems expect knowledge of the underlying systems or were created with different goals in mind. Game designers should also be able to maintain and configure PCG systems on their own, avoiding unfitting content and altering the generation processes to better suit their needs. (Khaled et al., 2013.)

3.3.2 What procedural content generation can be used for in game design

Smith (2015) has divided approaching PCG from designer's viewpoint into five main categories: generation as optimization, generation as constraint satisfaction, generation with grammars, generation as content selection, and generation as constructive generation.

Generation as optimization searches for combinations of game elements in a way that the result provides optimal fit for specific evaluation criterion (Smith, 2014.) Search-based PCG is an example of this approach, where most of the implementations use evolutionary algorithms or other methods of stochastic optimization. Togelius et al. (2010, 2011) mention that there is no general proof that evolutionary algorithms converge, so their

completion time cannot be guaranteed. Search-based algorithms usually spend most of their runtime in evaluation functions, which can require heavy resource usage such as simulation of content or human input. There is also no guarantee for the quality of output for evolutionary algorithms. These reasons make search-based methods better suited for offline generation and exploration of content. Optional content generation might also be preferred over necessary content, since it can tolerate more variation in quality of content compared to necessary content. Search-based PCG also lends itself well for personalized content adaptation. (Togelius et al., 2010, 2011.)

Generation as constraint satisfaction uses declarative constraints and rules to describe content for the generation process. Smith (2015) notes that the main strength of constraint satisfaction to the designer is the ability to specify requirements on content, without knowing how the inner generation algorithm performs. The drawback of the approach is that correctly describing content might be very difficult to begin with and debug. Smith notes that generation as constraint satisfaction is used extensively in tool programs for designers. (Smith, 2014.)

Generation with grammars is an approach that uses a formal grammar to expand content in the generation process. Grammars are processed purely as rules driving the generation process and they will not pay any attention of content quality. Creative grammars are a midway-point between letting designers specify their own rules and computer driven exploration of design space. Generative grammars are used in offline generation and tools for designers. (Smith, 2014.)

Content selection is a form of PCG that puts together pieces created by some other library or generator. This approach essentially selects pieces of content and creates new environments and experiences out of them. The approach can be used in applications that require great speed and must be run during play time. A drawback in content selection is that the player might notice patterns in the greater scale. (Smith, 2014.)

Constructive generation builds content by following a certain set of rules, piecing together elements in an improvised manner. Procedural methods usually don't perform any evaluation on content and it's quality. This makes all game-design related decisions a part of the algorithm, making constructive methods very game-specific to use. (Smith, 2014.)

3.4 Procedural content generation based game design

According to Smith, Othenin-Girard, Whitehead and Wardrip-Fruin (2012) the goal of PCG is often to replicate or replace human authors in content generation. Beyond this, the authors note that PCG can also be used as an on-demand game-designer with the ability to create unique play experiences. As an example of this, they mention the genre of roguelikes. The genre is named after the game Rogue (1980), and is mostly known by the usage of PCG as the generation mechanism of content. The goal of PCG in roguelikes is to provide variety, replayability, and compact content representation. This approach does not create any new forms of gameplay, so Smith et al. (2012) argue that one could make the PCG system itself the focus of gameplay. Smith et al. (2012) define PCG-based games as follows:

> “A PCG-based game is one in which the underlying PCG system is so inextricably tied to the mechanics of the game, and has so greatly influenced the aesthetics of the game, that the dynamics— player strategies and emergent behavior — revolve around it.”

Most games allow fine-grained control over content structure and player experience, making the systems producing emergence in games controllable by careful combinations of game content. The approach is different in PCG-based games, since the designer has to give up some autonomy of content creation and treat PCG as a core game mechanic. Instead of directly manipulating content, the designer must create ranges of meaningfully controllable content. (Smith et al., 2012.)

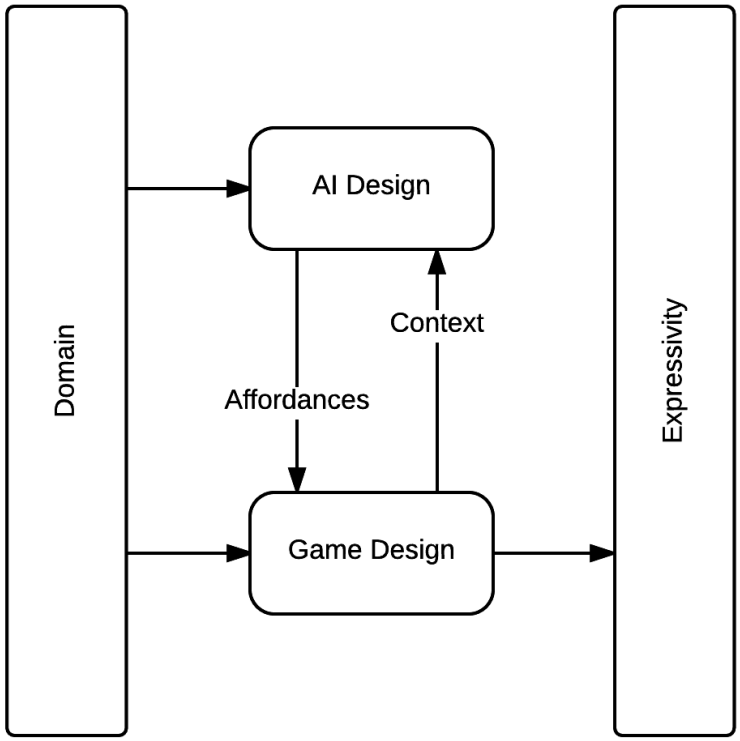

Figure 4 AI-based game design by Eladhari et al. (2012) .

In her dissertation, Smith (2012) describes PCG-based game design as a specific instance of AI-based game design by Eladhari, Sullivan, Smith and McCoy (2011). According to Eladhari et al. the current set of design tools and methods are not able to reach large portions of possible space of games. They note that in the past technological advances have driven the discovery of new types of gameplay, but still have not made very significant steps in producing new gameplay experiences. They propose AI-based game design as a solution for aiding in the exploration of currently unreachable space of games. AI-based game design is an iterative practise that integrates affordances provided by AI-systems to the context of game design. The authors define an AI-based game as:

> "An AI-based game is one that has an AI system closely integrated into its core mechanics and aesthetics, and also into the setting and story."

In the context of AI and game design, some games only create an illusion of intelligence. These games can suffer from the Eliza-effect (Wardrip-Fruin, 2009), where players credit the AI with more intelligence than the system can actually support. Players assign intentionality to AI systems for their behaviour, abstracting away the complexity of its underlying mechanisms. If this intentionality is not complex enough, the player's illusion of the AI-system's life-likeness breaks. According to Eladhari et al. (2012) these kinds of AI's are not able to support new forms of gameplay. They note that the AI-system must

be robust enough to withstand player experimentation and exploration, while games take full advantage of those systems.

According to Eladhari et al. (2012) AI and games should not be designed in isolation, and define AI-based game design as iterative co-formation of AI and game design. AI-based game design features an iterative loop (Figure 4), where AI-design and game design inform each other. In this loop, game design provides the context as the game world where the AI-system must operate in. Besides operating in a certain context, the AI-system is also bound by the constraints provided by the game designer. If the AI-system operates with no regard to its environment, it can seem to act autonomously or schizophrenically and the player will not be able to read intentionality to its actions. To maximize the intentionality of the AI-system, game design should also be affected by the affordances provided by the AI-system. If this is omitted, the game can suffer from the Tale-spin effect where the player is unable to understand the reason behind the AI-systems actions, breaking the image of AI's complexity. An opposite effect is the SimCity effect, where the player is able to understand the actions of the AI-system, which can only be achieved when game design is informed by affordances provided by the underlying system. The authors mention that even while the player needs information on how the AI system reached its decision, he must not be overwhelmed by an abundance of information. The player should be able to see details the AI system uses to make decisions, but the overall state of the system should be provided as an aggregated representation of all the details. (Eladhari et al., 2012.)

Besides the game and the AI-system being informed of game design, they are also informed of the project domain. Eladhari et al. (2012) mention the three most typical domains affecting game and AI-systems as AI architectures, game design conventions, and knowledge domains. Eladhari et al. give examples of these domains from AI-based games developed by the authors, which are Pataphysic Institute, Mind Music, Rathenn, Prom Week, and Mismanor. The games Pataphysic Institute and Mind Music both share the common AI-system Mind Module. Mind Module features an AI architecture based on the spreading-activation theory. The AI system consists from a network of nodes that determine the game characters mood. Mood of the character directly affects game mechanics in both of the games, Pataphysic Institute choosing what spells can be cast from mood and Mind Music sets its groove and harmony settings from it. As mentioned before, the design of the AI architecture affects game design decisions by shaping the generative space. Knowledge domains represent the different speciality subjects and theories on which the game design is based on. Mind Module is based on personality psychology, affect theory, and basic emotions domains. Game design conventions describe features and ways of doing things in a certain genre of games. The authors mention Rathenn as an example, where its design decisions were greatly affected by the conventions of 2D platformers. (Eladhari et al., 2012.)

Eladhari et al. (2012) note that emergence is an elemental aspect of AI-based game design. In this context the authors raise two qualities of emergence above else, emergent gameplay and using AI-based game design to study emergence within the AI-system itself. Emergent gameplay is described as interaction that the game was not specifically designed for. AI-based game design results in emergent gameplay, when the AI-system underlying game mechanics reacts intelligently to the players actions. Intelligent and fast reactions from the AI-system allow emergent gameplay to form without necessarily requiring emergence from the AI-system itself. The resulting gameplay can be studied to further develop an understanding of the expressivity of the AI-system. Considering the design loop of AI-based game design, more knowledge of the AI-system and its expressive range helps in turn to make game design decisions. The AI-system usually

affects game mechanics, aesthetics, and indirectly game dynamics. Game mechanics and aesthetics are characteristics that define game genres, and when affected by the AI-system, can lead to new game genres. (Eladhari et al., 2012.)

3.5 Procedural content generation based games

PCG has been traditionally used with various motivations, such as improving replayability, circumventing technical limitations, and providing game worlds so large they couldn't have been hand authored. (Smith et al., 2012, May.) The definition of a PCG-based game requires the game's underlying PCG system to be bound to its mechanics while shaping the game's aesthetics. Smith et al. (2012, May) used this definition as a lens to study games containing PCG elements, trying to define if they fulfill the definition of a PCG-based game or not.

The authors give an example of replayability enhancement from the game Rogue and its descendants Diablo and Spelunky. While successful in creating new play experiences, the use of PCG is not related to the game's mechanics or aesthetics. Another example is the game series Civilization. Civilization features PCG in terrain generation, affecting player strategies by making the players react to generated content. This is shadowed by other game mechanics, such as technologies and resources, making the most of the gameplay experience formed by something else than the PCG system. As what comes to circumventing technological restrictions Elite and .kkrieger take a different approach on PCG. The games use PCG to expand content from minimal representation with deterministic methods. This allows them to pack content to a magnitudes smaller level that would otherwise be required to store the full content. Minecraft and Terraria feature PCG in the role of replayability and exploration of large worlds. The player experience if formed by the creation of content, but the player can't affect the generation process in any way, barring new game dynamics from arising. (Smith et al., 2012, May.)

Smith et al. (2012, May) note that the current number of PCG-based games is quite small. As an example, the authors mention Galactic Arms Race and Inside a Star Filled Sky. Galactic Arms Race is an experimental multiplayer game by Hastings, Ratan, Guha, and Stanley (2009) where the player commands an armed spaceship. The player can defeat enemies, earn experience and money, and find new weaponry generated by the content generating NeuroEvolution of Augmenting Topologies (cgNEAT) algorithm. The cgNEAT is an evolutionary algorithm that works in real time, generating new content based on player statistics. Usually evolutionary algorithms related to content generation require some sort of feedback about content quality from the player, but in this case the cgNEAT system reads implicit feedback from player's behaviour. (Hastings et al., 2009) Inside a Star Filled Sky is a game by Rohrer (2011), where the player proceeds through nested levels created by seed values from previous levels. When the player makes decisions that affect the state of the current level, the changes will also affect other levels.

Smith et al. (2012, May) have created the games Rathenn and Endless Web as PCG-based games. The authors argue that while Galactic Arms Race and Inside a Star Filled Sky are PCG-based games that use PCG as a game mechanic, Endless Web differs from them. The reason for this is the player's direct control over generated content. Endless Web doesn't allow direct control over control creation, but instead encourages the player to plan and strategize around the content creation in order to reach his goals. Togelius, Schedl and Yannakakis (2011) examined the definition of PCG, arguing that purely player created content cannot be considered as procedurally generated content. For example a level editor features very direct, immediate, and local ways to interact with the content being generated. Mixed methods of PCG including significant amount of human input

exist, but the difference in generation comes from the ability of not being able to predict the exact changes made to the generated content. They also propose that if human input to the content generator is a part of the game, it cannot be called procedural content generation. Smith, G, (2015) challenges this definition by stating that content can be created procedurally if the human follows a clearly defined procedure. A distinction that also affects the classification is whether the player has complete control over the generation process, or if the process is bound by rules of the game system.

3.6 Design challenges in procedural content generation based games

According to Eladhari et al. (2012) emergence can be usually seen as a positive aspect in games, as it inherently creates new gameplay experiences and greatly expands the expressive range of content creation systems. These positive effects might also result in problems with game design, since emergent gameplay by definition results from interaction not specially designed for. AI-based game design faces challenges in the loss of authorial control and large expressive range of content and behaviour, making it difficult to design player experiences. The game designers should know the expressive ranges and be familiar with their AI-systems, in order to somehow control emergence. This leads to longer testing cycles, since it might be hard to foresee potential gameplay interactions that lead to unwanted player strategies. The authors also note that by their experience it is very important to iterate between AI and game design as early as possible in the design process. Without the rough design of the AI system, it is impossible to create the base for the mechanics of a game. And conversely, it is impossible to determine what to model with the AI-system without the rough design of the game system. (Eladhari et al., 2012.)

Eladhari et al. (2012) give examples of three main techniques they use when trying to direct emergence as adapting to player preferences, explicit guidance on the system's generative space, and setting goals that the player is free to fulfil in any way he can. In the game Mind Music the authors noted that creating novel but pleasant adaptive content was difficult, as the generative space was very large. They note that while smaller generative spaces are easier to control, it may lead to predictability. The size of generative space of an AI-system should be “just right” in regard to its context. The authors also note that in the future constraining emergent patterns might be better achieved by adding more refined filtering techniques to content selection, rather that limiting the size of the generative space. The technique of providing explicit guidance on generative space can be seen in Rathenn. A 2D map of the generative space is shown during gameplay, from which the player can make exploratory decisions on. The game places story goals in pre-determined interesting areas of generative space in order to encourage full exploration. The last example of setting goals can be seen in the game Prom Week, where gameplay is focused around resolving interpersonal conflicts. Goals are set as social conditions that must be met, forming a story as the player solves these puzzles. The game also features chaining sets of rules that when met unravel sub-stories in a way that feels like a natural progression of events. Settings these goals that the player is free to reach in any way he wants creates sequential gameplay that is more focused on narrative story telling than inconsequential events. (Eladhari et al., 2012.)

Smith et al. (2012, May) created the game Endless Web as an experiment in PCG-based game design. Endless Web is a 2D side-scrolling platformer, where the player takes the role on an Eidolon that lives in the dreams of humans. Eidolons must explore the dreams of men, and release them from their nightmares. The game features an AI-system called

Launchpad, which is a rhythm-based level generator for 2D platformers. (Smith et al., 2012, May.) Launchpad is an autonomous level generator based on formal models from 2D platformers by Smith, Whitehead, Mates, Treanor, March, and Cha (2011). Launchpad builds levels from segments called rhythm groups, generated with grammatical methods. Rhythm in 2D games relates to the timing and repetition of user actions, where the player must time his actions correctly. The generated segments are put together and evaluated by a set of heuristics. Launchpad also guarantees that the created levels are playable within the restrictions of current physics system settings. One of the most important features of Launchpad is it's separation of level pacing and geometry, allowing designer control over certain aspects like difficulty, while allowing players to explore the generative space of physical level elements. (Smith et al., 2011.)

Smith et al. (2012, May) faced multiple design problems on creating Endless Web, which mainly rose from treating PCG as a core game mechanic. The authors note that it is important to maintain balance on who controls content generation. Using PCG as a game mechanic requires the player to have some control over content generation to form strategies around the generative process. In order to maintain control of the overall gameplay experience, player control must be bound by the game designer. Examples of design choices that should be controlled by the game designer include pacing and difficulty. Endless Web uses Launchpad as its AI-system, which chooses components based on probabilities specified by the designer. While the player can affect what components are created in the game world, the designer maintains control over the general pacing of the game, which is also related to game difficulty. Designing difficulty in a PCG-based game is more challenging, as the author no longer has total control over the flow of the game. Smith et al. found that their previous iteration of Endless Web didn't feature enough variation on content created and lacked a progressing difficulty level. This was resolved by including a tiered difficulty system into Launchpad. The system keeps track of player performance and assigns a components difficulty level independent of its type. The player is able to affect difficulty level by visiting certain tuning portals, which scale difficulty to the selected direction. Having multiple difficulty levels of components raises the number of total components, which solved the problem of variation. The authors note that while content variation and unpredictability is a good thing, variability must still be controlled in a way that the resulting gameplay experience is fair and engaging with a proper difficulty level. (Smith et al., 2012, May.)

One of the most important aesthetics of Endless Web is the aesthetic of exploration. Exploration in Endless Web consists of examining both the physical game world and the generative space of Launchpad. The player is encouraged to explore the generative space with tuning portals that change the percentage chances of component selection used in creating the physical game world. The authors note that this aesthetic was achieved by using PCG to generate an endless world, something that could not be done with pure hand authoring of content. The player is given tasks of achieving certain goals that are placed around the generative space. In order for the player to complete all these tasks, he must visit tuning portals to reach the specific areas of generative space. This makes the player form his strategies around tuning content generation in ways he likes, as there might be multiple ways of reaching the points defined in generative space. The authors noted that one of the most difficult things in navigating the generative space was the lack of any clear land marks for tuning portal locations. Endless Web levels are divided to multiple segments, which each hold one tuning portal. Previously tuning portals were distributed randomly in the level, but this affected player experience negatively as players couldn't find the portals they needed. The authors solved this problem by making the generation process assist the player by placing tuning portals in a probabilistic fashion based on how much the player needs each portal. The probability of a portal is evaluated by which one

of them is needed to reach the nearest goal. More portals will still be available if the player continues to explore further. When the player enters a portal, the available portals are reset. Endless Web tuning portals were designed to be familiar components in a changing world and transitional points between levels. In order to avoid the Tale-Spin effect where the player can't understand the reasoning behind the AI's actions, the player should be able to understand what the AI-system is doing. As Launchpad features no explicit input from player, such as having them analyse preferred geometries, the authors decided that the player's actions should result in predictable effects on the generation process. This approach still faces problems when the probabilities of components are tuned to be very similar, as immediate changes can be hard to see. The players also had difficulties in adjusting to new rules in an already established game genre. As Endless Web is a side-scrolling platformer, players were used to moving from left to right and avoiding falling down. Some of the tuning portals in Endless Web featured moving or falling down a gap, which caused confusion amongst players. Another problem regarding genre conventions was that the web they are exploring refers to the generative space of the generator and not the physical map of the world. (Smith et al., 2012, May.)

Endless Web is situated in a dream world and the authors wanted the game world to feature familiar nightmarish symbols. For this reason, the art assets in Endless Web are hand authored instead of procedurally generated. Even though the assets were designed to be modular, the design team faced problems in creating the art assets as it was impossible to know beforehand when they would be used. Interestingly enough, the art assets also conflicted with the game's physics system. Endless Web features changeable physics parameters, which affect how high the character can jump. If the physics system limited the player's ability to jump, he wouldn't have been able to jump over or on top of enemies. (Smith et al., 2012, May.)

Fernandez-Vara (2014, October) created the game Symon as an experiment of using PCG to create narrative puzzles. The author notes that Symon uses PCG as a similar expressive device as Endless Web, but in a different genre. Symon is a game that is situated in the dreams of the main character Symon, a paralyzed patient in a hospital bed. The game is bound by the genre conventions of point-and-click adventure games, where puzzles are connected to the game environment and solved by interacting and manipulating with the environment and its items. In order to put Symon to rest, the player is given the task of retrieving a set of items. To retrieve these items the player is required to complete prior puzzles that eventually lead to the final items. Puzzles are located in connected rooms that represent different memories of Symon's life, with an NPC character associated with the particular memory. The PCG system in Symon is based on the GRIOT (Harrell, 2005) poetry generation system, which generates structures for poems. GRIOT first decides the overall high-level structure of the poem and fills it with suitable content. Symon's PCG system operates the same way, splitting the main areas of game design into creating puzzle map structures and selecting suitable pieces content to fill the requirements set in individual puzzles. (Fernandez-Vara, 2012, October.)

Symon might not fulfil the description of a PCG-based game by Smith et al., but it highlights the same design challenges the authors faced when creating Endless Web. The largest problem in the design process of Symon was the workflow created by PCG. The designer was unable to directly affect puzzle map structures, as each of them was generated procedurally. The designer only had direct access to the items and NPC's used by the PCG system, which made iteration between the AI-system and game design a large issue. The author notes that while the game pipeline prevented the creation of more complex puzzle patterns, it was more of a design centric problem than an algorithmic one.

Development of tools that give the game designer more control over the game pipeline became the most important future work goal. (Fernandez-Vara, 2012, October.)

Symon was released as a browser game and received lots of attention from players and reviewers. From the game's usage data the author could notice that players had difficulties in orienting themselves to the replayable nature of Symon, as they were expecting Symon to behave the same way as other point-and-click adventures. When further examining Symon, the AI-system used to create content was not made visible to the player in any way, and caused confusion to players as they tried to seek help from forums and other means of community communication. As players didn't realize that the puzzles were generated, they couldn't directly help each other's. The author notes that the confusion might be caused genre conventions learned from prior point-and-click adventures, where players don't expect replayability and variability in the puzzles they are facing. Related to the expectations from game genre conventions, the players didn't mind the repetitive nature or randomness of puzzle patterns, as they are one of the genre-defining traits experienced in other games of the same type. Most of the player problems were related to manipulating items in a way that resulted in the puzzle becoming impossible to complete. This was resolved be adding a game mechanic of resetting items to their original state prior to any manipulation. (Fernandez-Vara, 2012, October.)

The main lesson noted in designing Symon was the difficulty in incorporating PCG to the game design process. Initially the author used PCG as an expressive method to create new game mechanics in a dream-logic based adventure. Later in the design process it was found that PCG was not a solution to problems, but a design choice that required the support of other design choices made. The author also noted that the hardest part of PCG was not creating puzzles, but making content both playable and meaningful. (Fernandez-Vara, 2012, October.)

The authors of GAR discussed the use of PCG in games and saw it as a trade-off between novelty and game design control. PCG might not fit every game, such as linear first person shooters, where the player experience relies on hand authored content. The authors also note that ideal parametrizations options for evolutionary algorithms should be left as open ended, in a way that content can be evolved beyond initial intentions set by game designers. Game designers should aim for open-endedness with their generative systems to allow exploration, even while the initial effort of parametrization for each content class might require much effort. The particle weapons evolved in GAR used implicit evaluation functions based on the use of weapons. The player received new evolved weapons based on the previous weapons he used the most. The authors noted that this sort of implicit evaluation also acted as a balancing factor between different weapons, as variants of popular weapons were distributed to other players. The authors also discussed the topic of novelty in created content. They note that some game designers and players might think effectiveness of content matters the most, but some players behave otherwise. The implicit evaluation of content by usage allows game designers to bypass these philosophical questions and let the generative system create content based on player behaviour. (Hastings et al., 2009.)

3.7 Mixed initiative co-creation

Due to increased popularity, digital games attract various persons with different backgrounds wanting to contribute to existing games with user created content. To encourage community participation game companies sometimes release easy-to-use tools, such as in-house scripting languages and game editors. The problem with these tools though is that they still require ingenuity, drive and experience with games from the

end-user to achieve proper results. (Liapis, Yannakakis, & Togelius, 2014.) Games such as Rogue, Civilization, and Diablo feature PCG in a role that automatically handles the generation of content without much interaction between the PCG system and the human designer. Most of the level design is implicitly encoded to the generation algorithm, limiting designer contribution to adjusting parameters for the generator. This way of interacting with the generator can be unintuitive and time consuming, depending on the locality of the changes provided by parameters. (Smith, Whitehead, & Mateas, 2011.)

Yannakakis, Liapis, & Alexopolous (2014) describe the concept of mixed-initiative co-creation (MI-CC) as "… the task of creating artefacts via the interaction of a human initiative and a computational initiative." In MI-CC the human designer and the computer both participate to the creation process proactively. MI-CC differs from other forms of co-creation, since only MI-CC features proactive participation from the computer. The authors mention level editors that speed up the creation process, but limit the computer's participation to supportive tasks such as pathfinding and rendering. The creative force behind the process still comes from the human designer. On the other end, PCG systems such as SpeedTree are adept at creating various content, but restrict human contribution to adjusting parameters to the creation process between runs of the tool. MI-CC aims to not only assist human creativity, but fosters it. The authors suggest that the potential for human design ideas are usually not fully realized, as the users and designers might lack the suitable co-creation tools. The difference in MI-CC and traditional computer aided design (CAD) tools is that MI-CC features an autonomous system that is able to independently explore the generative space, guided by the human partner. (Yannakakis et al., 2014.)

Smith, Whitehead & Mateas (2011) have introduced Tanagara as a mixed-initiative level design tool that supports the human designer with procedurally generated content. Tanagara is designed to allow direct interaction between the designer and the underlying content generator. Tanagara features on-demand content generation, playability checks for levels, and the ability to modify level pacing without affecting its geometry. The aim of Tanagara is to reduce the human designer's authorial burden, while allowing him to be creative and use his aesthetic decisions on created content. (Smith, Whitehead, & Mateas, 2011.)

Liapis, Yannakakis and Togelius (2013) introduced the Sentient Sketchbook, a CAD tool that assists game designers in level creation. Sentient Sketchbook features creation of levels by allowing the designer to sketch low-resolution abstractions from strategy game maps. The advantage of minimal abstractions is that the user is able to quickly design a map and let the editor convert it to a playable 2D or 3D map with minimal time investment. Sentient Sketchbook is designed to address limitations of mixed-initiative design, such as user fatigue and designer bias, by limiting the work items to simple sketches. Strategy game maps usually consist from tiles and feature resources to gather and locations for player bases. While the Sentient Sketchbook is a tool that the designer uses to sketch levels, it also acts as an assistant by suggesting alternative map designs and offering real-time feedback from the level. The authors assume that designers are driven by objectives of gameplay related factors, such as balance and game pace, but not necessarily all of them. The authors argue that in such cases the editor's novel suggestion will be useful. The editor's feedback includes data related to level playability, balance, navigation and gameplay evaluation. (Liapis et al., 2013.)

Besides offering helping data, the Sentient Sketchbook aims to foster designer creativity by offering novel alternatives to level features, created by genetic algorithms. Generally genetic algorithms rely on evaluation functions to optimize the population towards the

most fit individuals. Evaluation functions can be difficult to write as the requirements for content may be hard to specify and might be very subjective. Sentient Sketchbook features two different approaches to genetic computation, which are the novelty search and feasible-infeasible two population genetic algorithm (FI-2pop). The Novelty search focuses on maximizing population diversity instead of quantifiable objectives of content. The argument behind this approach is that more thorough exploration of search space might discover unseen high-quality solutions. The FI-2pop has two populations of individuals, which hold the feasible and infeasible solutions. The algorithm optimizes the infeasible individuals towards the border of being feasible, with the argument of driving the generation towards feasible individuals. (Liapis et al., 2013.)

In order to map the affordances provided by the Sentient Sketchbook Liapis et al. (2013) conducted a small user study with five expert users, who were told to interact with the tool as they pleased. The authors found that one of the most important features of Sentient Sketchbook was the ability to rapidly create map suggestions. Overall reception of Sentient Sketchbook was positive, even while alternative map suggestions were rarely picked. Map alternatives created for novelty were picked early in the design process, when the map had little interesting features and the designers wanted large scale changes to the map. Object based optimization suggestions were picked most on elaborate hand authored maps, and were picked overall more than novelty suggestions. The authors explain this with the users being experts on map creation, resulting in good initial map quality. Game balance related suggestions were also omitted by the designers, as their designs were highly symmetric, thus ensuring good initial balance. One the authors' ideas for future work relates to using novice designers as test subjects to properly gauge the creative aspects of the tool. (Liapis et al., 2013.)

Liapis, Holmgård, Yannakakis and Togelius (2015) introduced a method of evaluating game levels with procedural personas with the MiniDungeons game. Procedural personas are artificial agents that are constructed to able to reproduce the decision making process of human players. MiniDungeons is a turn-based roguelike game and its levels are comprised of 12x12 grids, where the tiles are occupied by walls, treasures, enemies, or level exits. The authors note that five procedural persona types have been previously identified as monster killer, treasure collector, “baseline” persona, speedrunner, and a survivalist. MiniDungeons creates similar small levels with the same FI-2pop approach as Sentient Sketchbook, but the levels differ as they are now directly playable. MiniDungeons creates levels with simulation evaluated SBPCG, where the various personas must be able to complete the level and their play experience is used to evolve the levels accordingly. This process makes the personas act as critics on playability and level quality. (Liapis et al., 2015.)

Liapis, Yannakakis and Togelius (2013, November) introduced the term designer modelling as a method of identifying designer style, preference, intentions, and affective state. The aim of designer aware CAD tools is to reduce designer effort and speed up the content generation process, while supplementing the designer's creativity by offering novel suggestions that the designer might not have thought of. This can also be thought of as breaking the designers' usual pattern of work, aiming to foster creativity. New suggestions that break the designer model should be created with certain base level of usefulness in order to not seem outright wrong or random. Designer modelling is a demanding task, as the tool is intended to assist the designer, who is in turn trying to adjust the creation process in regard to the player's experience. Additional challenges for adaptive CAD tools are the partly unclear definition of the designer's tasks and the difficulty in measuring them. (Liapis et al., 2013, November.)

When discussing computational creativity (CC), the key characteristics of a creative process are usefulness and novelty. Autonomous generative systems are able to provide exhaustive amounts of novel combinations, but more commonly than not provide uninteresting solutions. For that reason another requirement of creative solutions is that they produce somehow valuable content. (Yannakakis, Liapis, & Alexopolous, 2014.) Liapis, Yannakakis and Togelius (2014) note that studying CC in the context of games differs from prior literacy, because games can be seen as multi-faceted entities, in which each facet evokes different emotions in the player. The most important creative facets in video games feature visuals, narrative, ludus, level architecture, and gameplay. The authors argue that computational game creativity challenges regular CC's theory of evaluating creativity because of the mentioned multi-faceted nature of games and them being highly interactive experiences to the player. The interactiveness of games means that their creativity can't be evaluated by just some spectator, but by the players who bring their own creativity to interact with the game's generative system. Distinguishing designer and player creativity can be hard as the player might be preoccupied by other elements of the game so that he ignores the intricate level design. The reverse situation exists where the designer has provided game mechanics which the player uses in an unexpected way, such as rocket jumping. These effects become even harder to evaluate when the player is given large control over the game world, much like in Minecraft. (Liapis et al., 2014.)

4. Discussion

The discussion presented in this thesis is divided into the most important findings from the literature reviewed. The discussion begins by reviewing research literature revolving around PCG and continues by examining the design choices affecting the development process and how the designer is able to divide content generation control between him and the player.

4.1 Research around PCG is mostly young and divided by viewpoint differences.

PCG holds many promises of reducing authoring costs, speeding up development processes, and offering replayability with personalised and novel content. Even with its alluring benefits PCG has not seen much commercial success, with the exceptions of few selected games that feature content generation as a part of their gameplay. Most of the arguments behind forgoing PCG in commercial games is the lack of reliability and control in the generation process. PCG has been around for a long time, but only during the last decade it has seen a rise in interest.

Yannakakis & Togelius (2014) have reviewed the main research areas in artificial and computational intelligence in the context of games. Based on interconnections between research areas, they identified the most influential areas as game AI benchmarks, NPC behaviour learning, PCG, and general game AI. The authors also found that the research areas most influenced by other research areas are clearly PCG and commercial game AI. The AI methods used in literature related to SBPCG are dominated by evolutionary computation, which has been tied to offline generation because of its computationally demanding nature. When the authors examined online PCG, other AI methods were preferred over evolutionary computation. From all the computational processes reviewed by the authors, player modelling was selected as the research area with most immediate and direct effect on player experience. From other research areas that were not directly connected to player experience PCG was selected as the most influential research area in defining the player's experience. PCG and player modelling have a strong connection, as most contemporary PCG methods include player models in them. (Yannakakis & Togelius, 2014.)

According to the survey by Hendrix et al. (2013) the research revolving around PCG is divided across multiple disciplines and is mostly focused on methods used to create content in specific subcategories. The classification of game content by Hendrix et al. (2013) also indicates an inverse relationship in the maturity of PCG techniques in relation to their position in the taxonomy layers (Figure 2). Another perspective to PCG research is the role of PCG in game design. The MDA framework by Hunicke et al. (2004) is an important aspect of video game research, as it allows people from different fields of study to share a common vocabulary in game design discussion. Smith's framework in Figure 3 is a description of PCG's relationship to game design and bases its analysis on the MDA framework. Smith (2015) agrees with Hunicke et al. that in order to really understand the role of PCG in games, the framework for discussion should include both game design and player experience viewpoints. Smith also brings out that designers should understand how to produce specific experiences through PCG and what unique aspects it has to offer.

It appears that the lack of maturity in PCG methods might not be the only reason PCG is lacking commercial success, as its role in game design might not have been yet fully realised. The literature studying PCG is focused to study the topic from an AI research perspective, and with strong influence from other research areas the design related best

practises have not yet emerged. The strong connection between player modeling and PCG hints that research is currently trending towards bringing new and adaptable gameplay experiences for the player. These goals and motivations still need to be addressed on game design level, within the proper framework of PCG and game design.

As an example of ill-fitting motivations, Smith (2015) brought out replayability as a widely used motivation to use PCG in video games. Smith argues that replayability is too broad of a rationale for PCG in games, as she has identified multiple kinds of replayability resulting from different design choices. Smith notes that while replayability can be achieved with hand authored content, generating player strategies around content creation is a unique dynamic to PCG. The relationships between PCG and mechanics, dynamics and aesthetics should help the designer to make the decisions on how to use PCG and to what purpose.

4.2 The use of PCG should be a conscious design choice, supported by other design choices made.

When incorporating PCG to video games, the designer has to make conscious decisions on the role of PCG and how to use it. Hunicke et al. (2004) mention that everyone regardless of discipline should keep in mind the base game mechanics, overall design goals of the game, and the preferred gameplay experience. Khaled et al. (2013) described the relationship between PCG and the designer with the tool, material, designer and expert metaphors. Smith (2015) divided approaching PCG from the designer's viewpoint to five main categories of generation as optimization, generation as constraint satisfaction, generation with grammars, generation as content selection, and generation as constructive generation. All of the above approaches should also be thought in the context of online and offline generation with pros and cons in algorithmic performance and reliability.

The designer has to also be aware how the game design affects the AI-system and vice versa. Eladhari et al. (2012) note that besides game design, the AI-system is affected by knowledge domains, AI-architectures and game design conventions. These design decisions also affect the shape of the AI-system's generative space, which should be in turn reflected to the game's design.

When players interact with a game AI-system, they assign intentionality to decisions made by the system. If the player credits the AI-system with more intelligence than it can support, the illusion of life-likeness behind the AI-system's decisions breaks due to the Eliza-effect. In order to support new forms of gameplay, the AI-systems used need to be able to withstand player experimentation without breaking. The image of the AI-system's complexity can also be broken by decisions that the player cannot understand, resulting in the Tale-spin effect. When the AI-system reacts intelligently to player actions, it allows for emerging gameplay. (Eladhari et al., 2012.)

Emergence is an important part of PCG-based games, allowing new forms of gameplay to surface. This does not come without a price, since emergent gameplay results from interaction that was not planned for. PCG has been suggested as a tool that shortens development cycles, but in order to avoid unwanted interactions it might actually lengthen them. The developer has to familiarize himself with the system's expressive range. (Eladhari et al., 2012.)

PCG can be hard to use properly, as the designer might not be able to justify all the design decisions during the development process. Eladhari et al. (2012) note that PCG-based games should be designed in an iterative fashion, where the design of the game and its AI-system enforce each other. Even if the designer chooses to utilize PCG in creating

specialised parts of content, the guideline is still useful to make sure the created content fits into existing the existing content.

PCG can be used used in different levels of commitment, ranging from development phase offline generation to core game mechanics with online generation. Some PCG systems are meant to act as a replacement for the human designer, creating content without much interaction or supervision. Tools such as SpeedTree provide great utility for the designer and the development team, as it lowers authoring costs and allows the development effort to focus on creative aspects. MI-CC tools such as Sentient Sketchbook act as designer peers and provide further affordances as the AI-system works in co-operation with the human designer, taking initiative in the design process. Most of offline generation imposes low risks of catastrophic failures, as the generation process doesn't usually have hard time constraints. The more PCG is meant to affect the player's gameplay experience and the more time acts as a constraint to the generation process, the more design choices can weigh in making a successful game.

PCG-based game design represents the most commitment to transferring authorial control to the AI-system and by definition ties the game's mechanics to it, resulting in the game's dynamics to revolve around the PCG-system. The designer approach to the design process changes from hand authoring single experiences to providing ranges of meaningful content and directing gameplay experiences through the PCG-system.

4.3 PCG-based game design requires balancing generation process control between the designer and the player.

One of the designer's main tasks is to control the player's gameplay experience and ensure that it is appealing and scaled to the correct difficulty level (Smith et al., 2012, May). Traditional designer workflow allows precise control over player experience by hand authoring and play testing. Introducing PCG to the design process means giving up autonomy in the creation process and transferring it to the PCG system, allowing it to make decisions on the content it creates. Using PCG in game design requires a shift from hand authoring content to creating ranges of meaningfully controllable content. (Smith et al., 2012.)

An important aspect of content generation is the control exerted over the generation process. PCG-based game design ties the AI-system close to the game's core game mechanics and aesthetics. In order for the player to form gameplay strategies around content generation, the player must have some control over the generation process. Even though the player has some control over the generation process and may navigate the search space at will, the designer must still establish clear boundaries on what aspects of generation the player can control, while keeping the most important design choices under his own control, such as difficulty and pacing. (Smith et al., 2012, May).

When the player is given a chance to affect the generation process, the intention of his actions becomes an interest to the designer. According to Schedl and Yannakakis (2010) purely player created content cannot be considered as procedurally created and mentioned an example of level editors, which allow direct, local, and predictable changes to the content. Togelius et al. (2011) describe adaptability as a form of parametrizable PCG, where the parameters depend on previous player actions. The authors argued that depending on the locality and time scale of changes, the player can abuse the adaptive process and use it as a gameplay device.

Abusing the generation process stems from the player being able to directly predict how his actions affect the end result of content generation. An example of this can be seen in

some games that feature adapting difficulty levels, where the player can purposefully repeatedly die in order to lower the difficulty level. Smith et al. (2012, May) used the Launchpad AI-system in their game Endless Web, which used designer specified probabilities of selecting game components. The player could visit tuning portals to affect these probabilities and thus was able to explore the AI-system's search space. The authors implemented a tiered difficulty system, which assigned component difficulty level based on player performance. The player was also given the chance of visiting difficulty tuning portals, which lowered or raised the difficulty based on player selection. (Smith et al., 2012, May.) This highlights the effect of giving the player the ability of doing something through game mechanics, versus trying to abuse the game system.

Distributing control in a fine-grained manner might be challenging from a designer viewpoint, as Khaled et al. (2013) mention that most PCG systems expect expert knowledge of the underlying systems. The authors also note that game designers should be able to maintain and configure the PCG systems themselves to make sure they fit to their needs.

5. Conclusion

Video games have become a part of daily life and as a result the demand of game content is increasing (Entertainment Software Association [ESA], 2015). Content creation is becoming one of the biggest problems in video game scalability and project resource usage (Iosup, 2011). PCG has been suggested as a solution to these scalability problems with the added benefits of possibly new game genres and personalized game content (Smith et al., 2011).

PCG is not a recent concept, with its roots originating from tabletop role playing games and older digital games such as Rogue and Elite. The motivation behind using PCG in games hasn't changed much from those days, featuring replayability, surprising the player and providing variety. (Smith, 2015.) PCG has been used almost exclusively in game development phase of production and on specialized roles and subsets of game content. The reasons behind this include the lack of control in quality and reliability of the content generation process. (Yannakakis & Togelius, 2011., Merrick et al., 2012.) Adaptivity in games is a related concept to PCG and has been considered as an appealing approach to improving replayability and diversifying the game's player base. Despite its benefits, almost no commercial games use adaptable PCG. (Togelius et al., 2011.)

Video games are content-intensive multi-faceted entities and the content created in them ranges from visual art and environments to sound design. They also offer very rich forms of interaction with the player, allowing players to express their creativity through gameplay. (Liapis et al., 2014.) When PCG is examined in a specific role of content creation, different variations of PCG methods offer a trade-off between content novelty and game design control (Hastings et al., 2009). Differences between PCG methods is also a matter of algorithmic performance and the game designer must know which methods suit the game (Togelius et al., 2010, 2011). When the designer chooses to use PCG in content creation, he gives up some control over the design process to the PCG system. The designer's role then shifts more into choosing, editing and organizing the outputs of the PCG system. (Khaled et al., 2013.)

The goal of PCG has often been thought of as replicating or replacing human authors in content generation. PCG has also been used as an on-demand game designer with the ability to create unique play experiences. Even though using PCG in specific content creation roles can lead to new gameplay experiences, following traditional design practices won't allow new forms of gameplay to emerge. PCG-based game design has been introduced as design practice that aims to incorporate content generation and interaction with the generator as a core game mechanic. (Smith et al., 2012.) PCG-based game design is an iterative design practice where the design of the game and its AI-system are reflected on each other as a loop. When developed in tandem, the game design places the AI-system's actions into context and the AI-system provides various affordances to the design the game with. If the game is not designed with its AI-system in mind, or vice versa, the decisions made by the AI-system can seem incomprehensible to the player. (Eladhari et al., 2012.)

PCG has sometimes been described as random generation, but this does not mean that content should be created or distributed without any regard to overall structure, as purely random content would be unplayable (Togelius et al., 2011). The design of the game should offer a context for the AI-system's decisions, so the player may see reason and meaning behind them. From a game design viewpoint, uniform randomness should be replaced by a directed approach in content variance. (Smith, 2015.)

Emergence is a key part of PCG-based game design. Emergent gameplay is described as the kind of interaction the game was not specifically designed for. Emergent gameplay is possible when the game's AI-system reacts intelligently to the player's actions, creating new gameplay experiences and expands the expressive range of content creation systems. Emergence is regarded as a positive aspect in games, but it also raises challenges in designing games, as emergent gameplay by its definition is something the designer did not take into account. PCG-based game design faces challenges in the loss of authorial control and large expressive ranges of the AI-systems. When designing PCG-based games, the designer must be familiar with the AI-system and take into account the size of its expressive range in order to predict the potential gameplay interactions that lead to unwanted player strategies. (Eladhari et al., 2012.)

Directing player experience is a challenging task in PCG-based games, as they rely on generated content, rather than direct hand authoring of content. Treating PCG as a core game mechanic requires the player to have some control over the generation process in order to form strategies around the generative process. Control over the generative process is an important aspect when designing the overall player experience, as it must be bound by guidelines set by the game designer. This means that the player might have influence over what kind of content is being generated, but some design affecting decisions, such as pacing and difficulty, are being controlled by the designer. (Smith et al., 2012, May.)

Some games and design tools offer PCG systems that create content automatically, without much interaction between the PCG system and the game designer. Most of the level design is implicitly embedded to the generation algorithm, leaving the designer to merely tweak its parameters. This way of interacting with content creation might be tedious and unintuitive, depending on the locality of the generators parameters. (Smith et al., 2011.) When creating games with PCG, the designer should have the proper tools to affect the generation process in order to avoid problems with his workflow (Fernandez-Vara, 2012). PCG systems, such as SpeedTree, are good at performing supportive tasks, but the creative force behind content creation process still comes from the designer. The concept of mixed-initiative co-creation (MI-CC) offers proactive participation from the computer in the creative process, aiming to foster designer creativity. MI-CC tools are able to independently explore the system's generative space, while being guided by the human designer. (Yannakakis et al., 2014.) MI-CC tools such as Tanagara and Sentient Sketchbook offer playability checks and other various useful features to help reduce designer authorial burden and allow them to focus on creative aspects (Smith et al., 2011, Liapis et al., 2013). The concept of designer modeling can help create better suited CAD tools for designers. Designer aware CAD tools speed up the development process and bring the designer even closer to the PCG system. (Liapis et al., 2013, November.)

All things considered, PCG is a powerful design concept for game development, bringing benefits in the form of novel content, reducing authorial burden, enabling new gameplay experiences and possibly even allowing new game genres to emerge. Approaching PCG from a designer’s viewpoint changes the traditional hand authoring approach more into interacting with the PCG system and directing gameplay experiences through it. The usage of PCG is an important design choice that requires support from other design choices made.

References

Acornsoft. (1984). *Elite* [PC game].

Angelides, M. C., & Agius, H. (Eds.). (2014). *Handbook of Digital Games*. John Wiley & Sons.

Bentley, P. J., & Kumar, S. (1999, July). Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem. In *GECCO* (Vol. 99, pp. 35-43).

Dormans, J. (2010, June). Adventures in level design: generating missions and spaces for action adventure games. In *Proceedings of the 2010 workshop on procedural content generation in games* (p. 1). ACM.

Eladhari, M. P., Sullivan, A., Smith, & McCoy, J. (2011). *AI-Based game design: Enabling new playable experiences* (Vol. 27). Technical Report, UCSC-SOE-11.

Entertainment Software Association. (2015). *Essential facts about the computer and video game industry: Sales, demographic, and usage data*. Retrieved 13.05.2015, available: http://www.theesa.com/wp-content/uploads/2015/04/ESA-Essential-Facts-2015.pdf

Fernandez Vara, C. (2014, October). Creating Dreamlike Game Worlds Through Procedural Content Generation. In *Seventh Intelligent Narrative Technologies Workshop*.

Harrell, D. F. (2005). Shades of computational evocation and meaning: The GRIOT system and improvisational poetry generation. In *Proceedings, Sixth Digital Arts and Culture Conference* (pp. 133-143).

Hastings, E. J., Guha, R. K., & Stanley, K. O. (2009, September). Evolving content in the galactic arms race video game. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on* (pp. 241-248). IEEE.

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 (TOMCCAP)*, *9*(1), 1.

Hunicke, R., LeBlanc, M., & Zubek, R. (2004, July). MDA: A formal approach to game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI* (Vol. 4).

Höök, K. (2008). Affective Loop Experiences–What Are They?. In *Persuasive Technology* (pp. 1-12). Springer Berlin Heidelberg.

Iosup, A. (2011). POGGI: generating puzzle instances for online games on grid infrastructures. *Concurrency and Computation: Practice and Experience*, *23*(2), 158-171.

Kelly, G., & McCabe, H. (2006). A survey of procedural techniques for city generation. *ITB Journal*, *14*, 87-130.

Khaled, R., Nelson, M. J., & Barr, P. (2013, April). Design metaphors for procedural content generation in games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1509-1518). ACM.

Liapis, A., Holmgård, C., Yannakakis, G. N., & Togelius, J. (2015). Procedural personas as critics for dungeon generation. In *Applications of Evolutionary Computation* (pp. 331-343). Springer International Publishing.

Liapis, A., Yannakakis, G. N., & Togelius, J. (2013, November). Designer modeling for personalized game content creation tools. In *Proceedings of the AIIDE Workshop on Artificial Intelligence & Game Aesthetics*.

Liapis, A., Yannakakis, G. N., & Togelius, J. (2013). Sentient Sketchbook: Computer-aided game level authoring. In *FDG* (pp. 213-220).

Liapis, A., Yannakakis, G. N., & Togelius, J. (2014, August). Designer modeling for sentient sketchbook. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on* (pp. 1-8). IEEE.

Liapis, A., Yannakakis, G. N., & Togelius, J. (2014). Computational game creativity. In *Proceedings of the Fifth International Conference on Computational Creativity* (Vol. 4, No. 1).

Merrick, K. E., Isaacs, A., Barlow, M., & Gu, N. (2012). Computational creativity and procedural content generation. *Unpublished article*.

Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.

Smith (2014, April). Understanding procedural content generation: A design-centric analysis of the role of pcg in games. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems* (pp. 917-926). ACM.

Smith An Analog History of Procedural Content Generation. In *Proceedings of the 2015 Conference on the Foundations of Digital Games (FDG 2015). Monterey, CA, June*.

Smith M. (2012). Expressive Design Tools: Procedural Content Generation for Game Designers.

Smith, A. M. (2012). Mechanizing exploratory game design.

Smith, Gan, E., Othenin-Girard, A., & Whitehead, J. (2011). PCG-based game design: enabling new play experiences through procedural content generation. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games* (p. 7). ACM.

Smith, Othenin-Girard, A., Whitehead, J., & Wardrip-Fruin, N. (2012, May). PCG-based game design: creating Endless Web. In *Proceedings of the International Conference on the Foundations of Digital Games* (pp. 188-195). ACM.

Smith, Whitehead, J., & Mateas, M. (2011). Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *Computational Intelligence and AI in Games, IEEE Transactions on*, *3*(3), 201-215.

Smith, Whitehead, J., Mateas, M., Treanor, M., March, J., & Cha, M. (2011). Launchpad: A rhythm-based level generator for 2-d platformers. *Computational Intelligence and AI in Games, IEEE Transactions on*, *3*(1), 1-16.

Togelius, J., Justinussen, T., & Hartzen, A. (2012, May). Compositional procedural content generation. In *Proceedings of the The third workshop on Procedural Content Generation in Games* (p. 16). ACM.

Togelius, J., Kastbjerg, E., Schedl, D., & Yannakakis, G. N. (2011, June). What is procedural content generation?: Mario on the borderline. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games* (p. 3). ACM.

Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2010). Search-based procedural content generation. In *Applications of Evolutionary Computation* (pp. 141-150). Springer Berlin Heidelberg.

Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2011). Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, *3*(3), 172-186.

Toy, Wichman, Arnold and Lane (1980). *Rogue* [PC game]

Yannakakis, G. N., & Togelius, J. (2011). Experience-driven procedural content generation. *Affective Computing, IEEE Transactions on*, *2*(3), 147-161.

Yannakakis, G. N., & Togelius, J. (2014). A panorama of artificial and computational intelligence in games.

Yannakakis, G. N., Liapis, A., & Alexopoulos, C. (2014). Mixed-initiative co-creativity. In *Proceedings of the 9th Conference on the Foundations of Digital Games*.

Zafar, A., & Mujtaba, H. (2012, December). Identifying Catastrophic Failures in Offline Level Generation for Mario. In *Proceedings of the 2012 10th International Conference on Frontiers of Information Technology* (pp. 62-67). IEEE Computer Society.