Understanding Procedural Content Generation: A Design-Centric Analysis of the Role of PCG in Games

Gillian Smith
Northeastern University
Playable Innovative Technologies Group
Boston, Massachusetts, USA
gillian@ccs.neu.edu

ABSTRACT
Games that use procedural content generation (PCG) do so in a wide variety of ways and for different reasons. One of the most common reasons cited by PCG system creators and game designers is improving replayability—by providing a means for automatically creating near-infinite amounts of content, the player can come back and replay the game and refine her strategies over a long period. However, this notion of replayability is both overly broad and incomplete as a motivation. This paper contributes an analytical framework and associated common vocabulary for understanding the role of PCG in games from a design standpoint, with an aim of unpacking some of the broad justifications for PCG use in games, and bringing together technical concerns in designing PCG systems with design concerns related to creating engaging playable experiences.

Author Keywords
Procedural content generation; game AI; MDA framework; game design; game design theory.

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation: Miscellaneous; I.2.1 Artificial Intelligence: Applications and Expert Systems—Games

INTRODUCTION
Procedural content generation (PCG) is a growing area of research and practice in game design. Researchers in artificial intelligence are creating more efficient and more expressive methods for having a computer create levels, weapons, terrain, and even game rules [17,56]. Simultaneously, game designers from both industry and research backgrounds are increasingly turning to the computer to create massive, varied worlds for players to explore, at a fraction of the cost of human authorship [10]. Furthermore, recent work at the intersection of HCI and AI has been examining using PCG to create intelligent user interfaces for game designers, while lacking a design vocabulary for how PCG can be used [25,47,52].

In order for further advancements in PCG research to be made, it is vital that both AI researchers and designers have a common vocabulary for understanding not just what PCG is but how it can be used to induce particular experiences and what it uniquely offers to game design. Thus, what is needed is a framework for understanding and communicating about how PCG is taking a role in game design. This framework needs to address both the ways in which the PCG system works to create content, which is of primary concern to the PCG researcher, and the experience the player (or user, in the case of AI-based tools) has that is shaped by that system, which is of primary concern to a game designer [19,42]. The framework presented in this paper was created by analyzing several PCG games and research projects through the lens of the Mechanics, Dynamics, and Aesthetics (MDA) framework [19].

PCG is used in a variety of ways in games, with the common justifications of replayability (as described in [56]) and tailoring content for an individual player. The promise of personalized content generation is especially compelling for educational and training games (e.g. [48]). This paper argues that "replayability", in particular, is a nuanced concept that requires unpacking through an examination of the designs of both the PCG system and the game that the system is embedded within. This overly-broad justification is examined more deeply through a design-centric analysis of how PCG has been used in games thus far. While the framework introduced in this paper focuses largely on PCG's implementation in game design, it is also important to note that the representation and mechanics portions of the framework can also be used to describe the role that PCG can play in game design tools, offering more detail than previous work in PCG design metaphors [23].

This paper has two main contributions. First, a conceptual framework for analyzing the role of PCG in game design (both for games and for game design tools), and second, a resulting vocabulary for both practitioners and researchers—from both artificial intelligence and game design—to discuss the affordances of PCG. The framework is illustrated using examples from several games, design

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

CHI 2014, April 26 - May 01 2014, Toronto, ON, Canada
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2473-1/14/04…$15.00.
http://dx.doi.org/10.1145/2556288.2557341

tools, and research projects, and a longer worked example is also provided. However, this is not intended to be a full survey of PCG, and only illustrative examples are used. Finally, the paper closes with a discussion of future work and a reflection on the unique experiences that can come from incorporating PCG in a game's design.

BACKGROUND

This section begins with an overview of technical approaches to procedural content generation and their design-relevant tradeoffs, then discusses the background in the two main contribution areas.

Approaches to Procedural Content Generation

There are many different approaches taken in procedural content generation, each with different affordances and tradeoffs for design. All approaches can be used in both a game and design tool setting. The five main categories that a content generator can fall into are: generation as **optimization**, generation as **constraint satisfaction**, generation with **grammars**, generation as **content selection**, and generation as a **constructive** process.

Optimization

Optimization approaches to content generation treat the design process as a search for the combination of elements that best fit some criteria, which can be either specified mathematically by the system creator, or judged and curated by a human. Optimization-based generators explicitly and externally model the desired qualities of generated content. A great deal of research in PCG uses evolutionary algorithms (this is sometimes referred to as "search-based PCG" [56]), which is an optimization approach. Optimization approaches to PCG are often (though not always) computationally expensive, making it difficult to use them in games that require a highly responsive PCG system. They are often used with a human-in-the-loop for the evaluation function [16,38,43] and to create personalized content [45] offline.

Constraint Satisfaction

This approach involves the declarative specification of properties of and constraints on the content that will be created. For example, levels in the math education game *Refraction* are generated by searching for solutions for a set of design constraints [48]. Declarative representations have the strength that a designer can specify knowledge about how the content should appear without needing to specify how the underlying search algorithm should perform. The challenge with constraint-based approaches comes largely in determining an appropriate representation for facts about the generated content, and with debugging a set of complex, interrelated constraints. This approach has been used extensively in tools for designers [4,46,52].

Grammars

These systems involve the specification of a grammar that the algorithm should expand upon to create content. These are not simply representational grammars used by an optimization-based generator (e.g. [44]). Grammars can be used purely as production rules to drive generation, without any regard for an external measure of the quality of the level. In such cases, level quality is implicitly baked into the grammar based on how the production rules are authored. Grammar-based methods attempt to strike a balance between designer-specified rules for how content components should fit together and computer exploration of the design space through expanding the grammar. Grammars have been used in offline content generation for games [8,51] and in tools for designers [30].

Content Selection

There is some contention about whether or not selecting content from a library to piece it together is complex enough to qualify as content generation [54,55]. This paper takes the position that content selection, however simple, is a form of PCG when it is used to procedurally create an environment for the player to explore or content for the player to experience within the context of different mechanics. Content selection is a very simple form of PCG that is susceptible to players recognizing large-scale patterns; however, it is also one of the fastest methods for generating content and is typically used in games where the generator must run during play time, such as "endless runner" games [1,41].

Constructive

A constructive generator is one that builds content in an ad-hoc manner by piecing together customized building blocks (see next section). It typically has all of its design knowledge baked into the algorithm; while it may perform some amount of search internally, it does not test the results of the level against some external heuristic to help guide the search process. Constructive generators are often quite game-specific. Examples of constructive generators are those used in *Rogue*-like games [39]. Constructive generators can be seen as content selection-based generators that use smaller pieces of content.

What is Procedural Content Generation?

There are two main survey and taxonomy papers for PCG in games. The first, from Togelius et al., focuses on the technical approaches to creating optimization-based generators, written from the perspective of PCG researchers for readers with a background in AI [56]. The search-based PCG taxonomy does identify two design-relevant characteristics of PCG systems, however: 1) necessity, and 2) whether the generator runs offline or online. The first of these characteristics is not discussed in this paper, as the "necessity" of generated content for the player to complete the game is not relevant to how the player experiences the content. The second property also appears in our framework as a property of the game's mechanics, in terms of how the user or player has control over the generator. This taxonomy also mentions a scale of directness-of-representation, as we do, but only in the context of evolutionary approaches to content generation. The second

survey focuses on the granularity of the content being created—from textures and small "game bits" to NPC behaviors and entire rulesets—and the maturity of the AI techniques used to create them [17]. It does not address issues of player experience or how the player interacts with the content generator within the game.

Togelius et al. have also written on what exactly constitutes PCG, and where the line can be drawn between randomness, player-generated content, and procedurally-generated content [55]. They concluded that PCG is defined as "the algorithmic creation of game content with limited or indirect user input". This paper takes a somewhat broader view of content generation, classifying games that do not provide any user input over the generator as still containing PCG, and also contributes a more nuanced set of attributes for how to describe the role of PCG in games.

Finally, Khaled et al. have written on creating "design metaphors" for PCG systems, to make PCG research from the AI community more accessible to those in HCI and design communities [23]. Their metaphors—which include considering the system as a *tool*, *material*, *designer*, and *domain expert*—all are broad categories for how a PCG system can be seen by a designer. This paper has similar goals in its aim to identify design-relevant properties of PCG systems and better understand the role that PCG can play in a game's design, but takes a more detailed look at the way the systems are used.

Frameworks for Game Design

Doug Church, in his call for more work in creating "formal abstract design tools", stated that "the primary inhibitor of design evolution is the lack of a common design vocabulary" [5]. Since this call for a genre-agnostic design language, there have been several attempts at creating languages and frameworks for understanding a game's design, the framework in this paper being among them.

One thread of design vocabulary research is in game design patterns (e.g. [2,18,28]). The framework contributed in this paper can also be considered as a set of "design patterns" for the role of PCG in games. Bjork and Holopainen's work is among the most comprehensive of these efforts, and includes a pattern for "Procedurally Generated Game Worlds" [3], but does not offer detail into the role that these worlds play in the player's experience, or different ways that they might be implemented in the game's rules.

Other efforts for building a design vocabulary include building languages to describe specific aspects of game design, such as Dan Cook's work on skill atoms [6] or Joris Dormans's "machinations" framework for abstractly representing and prototyping game mechanics [9]. These two frameworks sit at opposite ends of a spectrum of formality. Cook's framework, which focuses on designing for player learning and progression, is not at all formally defined. Dormans's framework, on the other hand, is a highly formalized, computational model for expressing game mechanics and subsystems. The framework presented in this paper aims to sit closer to the middle of this spectrum: it is not formal or detailed enough to be used to immediately synthesize a new game, but offers a strong enough vocabulary and set of examples that it can be easily used by designers to describe and guide the design of their own games.

Hunicke et al.'s MDA framework is another example of a language for describing and designing games [19], and forms the foundation of the analysis presented in this paper. In this framework, the design of a game is split into three main components: 1) the mechanics, or rules and systems in terms of algorithms and representation; 2) the dynamics, which are interactions between mechanics and the player's input; and 3) the aesthetics, or the "desirable emotional responses" that are experienced by the player. The framework is useful for the analysis performed in this paper because of its dual focus on design and experience. In order to understand the role that PCG takes in different games, it is crucial to understand it from both the system designer's perspective and the player's perspective.

UNDERSTANDING PCG IN GAMES

This section describes each aspect of the framework for understanding the role of PCG in games, categorizing each property by whether or not it is related to the game's mechanics, dynamics, or aesthetics. Each category in the framework is illustrated with an example from either a game or research project. It is important to note that aspects and properties in the framework are **not** intended to be mutually exclusive from each other—for example, a game might provide multiple kinds of control over its PCG system in different contexts.

Mechanics

There are four main mechanical aspects of the PCG system that are considered in this framework, summarized in Table 1. These aspects were chosen as a combination of concerns of the AI programmer (how the generator works) and concerns of the game designer (how the generator is used). Each of these mechanical aspects can be used to describe the role that a PCG system plays in a game design tool, as well as in games themselves.

Building Blocks

How design knowledge is represented to the generator is a crucial aspect of procedural content generation, impacting the range of content that can be produced. This framework contains four different categories for the building blocks used by generators: **experiential chunks**, **templates**, **components**, and **subcomponents**. These categories are actually discrete points on what is close to a continuous spectrum, from highest to lowest authorial burden.

Experiential Chunks. Experiential chunks are large building blocks for content that are designed by a human and are sufficiently large that, standing on their own, they can be experienced by the player. An example of these

Building Blocks	Game Stage	Interaction Type	Player Experience
Experiential Chunk Large, human-authored	*Offline* Before game	*None* No human influence	*Indirect* No direct experiential control
Template Computer fills in blanks	*Online* During game	*Parameterized* Indirect, human sets values	*Compositional* Human influences available components
Component Pattern Small, human-authored		*Preference* Human selects good products	*Experiential* Human influences player experience
Subcomponent Internal representation		*Direct Manipulation* Human manipulates product	

Table 1. Summary of the mechanical aspects of PCG. Each aspect has several potential values it can take on.

chunks include the rhythm groups that are stitched together by the *Polymorph* level generator [21][1].

Templates. Templates are a generalized form of experiential chunk; a human has still authored a great deal of the content, enough to dictate the kind of experience that will be had by a player, but has left some "blanks" (which may be further constrained) for the computer to fill in. Examples of these include the quest templates used in the *Scriptease* authoring tool [7].

Component Patterns. Component patterns are patterns that are large enough to be identifiable as human-designed in the game, but small enough that, on their own, they do not greatly dictate the experience a human will have. For example, patrolling enemies in platformer levels (e.g. Goombas in the *Infinite Mario Bros.* generator [34]) have a general behavior dictated by the human designer, but that behavior is meaningless to the player unless it is placed in context alongside other level components.

Subcomponents. Generators that operate at the subcomponent level are using the same building blocks as a human designer would if she were designing the content herself. Subcomponents exist at the layer of how the game is represented internally, such as individual tiles within a tile grid. *Galactic Arms Race* [16] operates at this layer, with it constructing weapon behavior to guide individual particles along a path.

Stage of Game

Togelius et al. make the distinction between generation that happens offline vs. online [56]; that distinction is continued here. Note that it is possible for a game to incorporate both kinds of content generation, if the game begins with some generated content, but that content might be procedurally altered or added to during play (e.g. *Galactic Arms Race* weapon generation).

[1] These chunks are, themselves, generated and ranked offline by the Launchpad level generator [51]; however, who owns authorship for those chunks is irrelevant to the Polymorph game and its generator, thus it is classified as a generator that uses experiential chunks.

Offline. Offline content generation happens before a unit of play experience begins. Thus, the generator used in *Civilization IV* [13] is classified as an offline generator, as the entire map is created before the main game activity begins. It is important to note that level generation in *Rogue* [57] is also considered to be offline—even though new maps are generated when the player progresses to the next level, the entire dungeon is created before the player begins exploring it and does not base generation on prior player behavior.

Online. Online content generation happens during the play experience, optionally in response to how the player is acting. Examples of online content generation include *Canabalt*'s placement of level chunks in front of the player just off-screen, or *Endless Web*'s [50] generation of new level geometry whenever the player alters a parameter to the underlying generator.

Player Interaction with Generator

In most games, the player interacts exclusively with the generated content, just as if it were created by a human designer. However, there are some games that offer the player the ability to interact with the generator itself: directly or indirectly requesting new content. Thus, it is useful to understand the different ways that a player can exert control over the generator.

None. With no control, the player interacts solely with the generated content and has no influence whatsoever over the kind of content that will be created by the generator. Examples of games that give the player no control over the generator include *Rogue*-like games [39], "endless runner" games, and *Minecraft* [35].

Parameterized. Parameterized control is a form of indirect control, where the player can provide the value for some set of parameters that influence the content the generator creates. For example, the player is able to manipulate parameters for the *Civilization IV* level generator [13] at the beginning of the game, to request the generator create a particular kind of map (e.g. one with no oceans, or a single large continent surrounded entirely by ocean).

Preference. Preference control is another form of indirect control, where the system is set so that the player can state (intentionally or implicitly) preferences about the content they are seeing. This form of control typically comes in the form of the player evaluating content that has already been generated to give feedback to the generator and help guide its future decisions. For example, *Galactic Arms Race* [16] uses player preferences, inferred from player behavior, to guide the next generation of evolved weapons.

Direct Manipulation. Direct manipulation involves the player actively altering the generated content to provide additional constraints to the generator. This form of manipulation is more commonly seen in PCG-enabled design tools [46,52], rather than games themselves. The *Spore Creature Creator* [26], a design tool offered as part of the game *Spore* [27], offers this kind of interaction.

Awareness of Player Experience

Some generators are designed simply to piece together content haphazardly and let player experience emerge from the results, while others are designed for crafting particular kinds of player experiences. In the case of generators that can be controlled by the player, it is also worth discussing whether or not the player's control extends to all aspects of how the generator designs for player experiences. There are three different ways a generator constructs content to produce an experience.

Indirect. All content generators, by their very definition, impact player experience somehow—it is simply a matter of understanding how intentional the impact is. Generators that exhibit indirect control over desired player experience are ones that do not have any intent baked into their design. For example, the gun generation system in *Borderlands* [14] is simply a matter of combinatorics—new guns are made by compositing different hand-authored gun properties. There is no desire to create specific kinds of guns for specific kinds of players, or even to limit the combinations that can be made based on game constraints.

Compositional. Compositional control involves influencing the appearance of particular components in the content being created, but not directly having experiential control over what is made. For example, *Endless Web* [50] requires players to manipulate parameters that control the frequency of different level components appearing in the generated geometry, but does **not** offer control in the form of demanding a particular difficulty or pattern for pacing. While the configuration of components does lead to a particular kind of experience, compositional control does not let a player explicitly control for experience, it must be left to emerge from the composition.

Experiential. Experiential control means that the generator can directly control for the kind of experience the player will receive from the content generator, and that (optionally) the player can interact with the generator at the experiential level. Yannakakis and Togelius attempt to directly model and capture player experience in their work on experience-driven procedural content generation [58]. A non-optimization based approach to experiential control comes in the game *Warning Forever* [33], in which the player's actions when attacking the boss directly influences how the boss evolves over time.

Dynamics

The results of these kinds of mechanics are several dynamics—ways in which the rules interact with each other and the player during play (see Table 2). When considering how PCG enables these dynamics, it is helpful to compare a game with PCG to a game with similar mechanics that does not have PCG. By doing so, it is possible to see what the unique dynamics are that are brought out by the incorporation of the PCG system, and how the mechanics related to the generator contribute to these dynamics.

PCG Relationship to Other Mechanics

First, in order to understand how the PCG system is situated within the game and the dynamics that emerge from it, it is helpful to think about how the player will be interacting with the generated content and the extent to which the generated content will influence the player's overall game experience.

Core. These games use the procedurally generated content as a core part of the player's experience. For example, *Infinite Mario Bros.* relies entirely on the generated content for the player to be able to experience the game; platformers are heavily reliant on level design to dictate the aesthetic experiences that are core to the game.

Partial framing. These are games that use procedurally generated content to frame some aspect of the player's experience, but it does not make up the entirety of that experience. For example, *Civilization [13]* is a game where map generation heavily influences early stage exploration and decisions about where to build cities, but there are so many other mechanics in the game that the player can build strategies around that, by the later stages of the game, the fact that the map was procedurally generated is far less important.

Decorative. There are many games that use PCG to create content that is intended to be purely decorative; this is especially common in computer graphics, where a concern is how to generate reasonable textures, vegetation, and decorative façades for city buildings [11,20,30]. While this distinction between "decorative" and the other two categories might be seen as similar to Togelius et al.'s declaration that some content is "unnecessary", that is *not* the intent with this category. All content in a game makes up a part of the player's experience, even if it is not directly interacted with, and thus there is no "unnecessary" content from a dynamics point of view.

Memorization vs. Reaction

Games that are heavily dependent upon PCG for their core play, and that use either online or offline generation of

Other Mechanics	Memorization vs. Reaction	Strategizing	Searching	Practicing	Interacting
Core Reliant on PCG	*Memorization* Testing player memory	Player builds strategies for influencing the content generator	Player seeks out new content in a vast world	Player practices game mechanics in new settings	Communities of players discuss differences in game experiences
Partial Framing Partial player experience	*Reaction* Reacting to unforeseen circumstances				
Decorative Not core to game experience					

Table 2. Summary of the dynamics aspects of PCG. There are six dynamic aspects, two of which can take on different values, while the others are properties that games using PCG might have.

content with no direct control from the player are games that are about **reacting to unforeseen circumstances**. Sometimes this reaction must come very quickly, in the case of endless runner games like *Robot Unicorn Attack* [1]. Other times, the reaction may be related to exploring an unfamiliar space, as in the platformer *Spelunky* [59].

In contrast, games such as these that do not have PCG and instead simply have static, pre-authored content tend to lead to dynamics of the player memorizing paths through a level (e.g. *Sonic the Hedgehog* [53]) or where items are hidden (e.g. *Donkey Kong Country 2* [36]), with the player's goal on replay being to beat her best score or test her memory.

It is important to note that the dynamic of reaction emerges simply from the use of randomness; it is possible for generators that perform content selection from a library of experiential chunks or templates to produce these experiences; there is no need for sophisticated generators that incorporate experiential design. Indeed, the dynamic of reaction vs. memorization arises in games without any PCG at all, but that include a random element, such as match-three games. Such games would be drastically different were the gem ordering to be deterministic; players would be able to memorize sequences of moves and practice moves that lead to optimal play.

Further, it is plausible that a similar experience about reacting to unforeseen circumstances could be crafted through crowdsourcing level designs from a large pool of users, such that each time the game is played, the player sees content from a different user. Without any input or control from the player, the role of the computer is to be an on-demand, highly productive replacement for a human designer.

Building Generator Strategies

Online, controllable content generators can lead to a dynamic in which the player builds strategies around the generator. This leads to a cycle in which the player informs the generator what should be created next, and the content presented in response informs the next decision that the player will make. Examples include *Warning Forever* [33] and *Endless Web* [50], which is described later in the paper.

There is no clear analog to this dynamic when considering games with static, human-authored content. Building a strategy around the generator is, clearly, only possible with an actual content generator that can respond believably to human input. In fact, generators that have only indirect control over player experience are not as suitable for producing this dynamic, as the player needs to have a mapping between action and generator reaction in order to learn how the system works and build a strategy.

Searching a Vast World

This dynamic arises from a generator's ability to create vast, varied spaces that could never be reasonably made by a single author. Generators that produce this dynamic are those that work either offline or online, at the level of components or subcomponents (experiential chunks and templates have patterns that can be quite easy for players to detect), create either core or framing content, and do not necessarily require any player control over the generator but can design for experience either directly or through components. There are several examples of games that use this dynamic, including *Minecraft* [35], the academic project *Charbitat* [32], and *Inside a Star-Filled Sky* [40].

The key here is that the generator produces content that is somehow surprising to the player as they explore the space. For example, the *Borderlands* [14] weapon generator produces a vast array of guns that the player can be surprised by, but the player might be searching for a particular kind of gun as they hunt through the space.

In order to produce this dynamic in games without content generation, it would require a large authorial burden on a team of human designers. However, there are games that show that this level of authoring is plausible; for example *Animal Crossing: New Leaf* [31] is a game that has a huge cast of characters, each of whom have item preferences, unique home layouts, one of several kinds of personality, and unique verbal cues. It has a large catalog of items for the player to purchase and search for over the course of play. While the physical space of the game is a small town, there is still a massive amount of designed content in the game that produces a similar dynamic of searching and discovering a vast, designed space.

Practicing in Different Environments

There are also games that use content generation to frame choices made by the player, where content is generated offline with or without control from the player. The dynamic that emerges from this form of generation is an ability for the player to practice game strategies in a variety of different environments. *Civilization IV* [13] is an example of a game that allows the player to exert parameterized control over the generator, in order to help guide the environment they will be playing and practicing in. *Diablo 3* [37] does not give the player control over the generator, but still allows the player to practice different combat strategies or character classes in different worlds.

In comparison, games that have only a single environment to play in do not achieve the same sort of dynamic. For example, *Professor Layton and the Curious Village* [24] is a game where the player must progress through a series of hand-authored puzzles. If the player is stuck on a particular puzzle, there are no options other than to ask for hints or seek the solution from another player. The incorporation of procedurally generated puzzles would allow the player to practice different solving strategies in a variety of related contexts. On the other hand, there are puzzle games such as *Picross* [22] which do allow the player to practice and improve upon their overall puzzle solving strategy simply through practicing on a large set of hand-authored levels at increasing difficulty. It is plausible for human-created content to compensate for the content-generator.

Interaction with a Community of Players

The final dynamic identified in this framework is that of how PCG impacts the community of players surrounding a game. Gee describes this community as the "external semiotic domain" for games, and as an area where players engage in a great deal of learning and reflection while talking to each other about game strategies [15]

PCG researchers typically focus on crafting an experience for individual players, only within the game's internal semiotic domain. However, there is evidence that the incorporation of PCG into a game prompts communication about the underlying system and its impact on the play experience within a community of players. For example, *Civilization IV* players engage in long discussions about how different map generation options work, how they impact strategies, and why they prefer certain options [29]. In a point-and-click adventure game that incorporated offline content generation to produce a unique set of puzzles for each player, the designer noticed that players who typically rush to post a walkthrough for adventure games worked together to figure out how the underlying content generator was working [12].

Understanding the community that surrounds games that use PCG is an area ripe for future research, both from a PCG AI point of view (how do we create generators that target a community of players?) and an HCI/game design point of view (how do communities of players communicate about PCG?). There is also an opportunity to study how PCG impacts player learning in games [49].

Aesthetics

The design choices made for the PCG system, in concert with the mechanics of the game, lead to unique dynamics. These dynamics then act in support of several different aesthetic experiences. The three aesthetics mentioned below are among the eight "kinds of fun" identified by Hunicke et al. in the original MDA framework paper [19].

Discovery

PCG acts in support of discovery by providing new environments for the player to explore or new procedural systems for the player to learn about over time. This aesthetic is supported by the dynamics of *Searching a Vast World* and *Building Generator Strategies*. In the first case, discovery comes simply through exploring a large, unknown environment. In the second case, discovery is also cast as a form of exploration for the game's *generative* space.

Challenge

PCG acts in support of challenge through the dynamics of *Memorization vs. Reaction*, *Building Generator Strategies*, and *Practicing in Different Environments*. Again, each of these dynamics leads to a different form of challenge. Games that use PCG to force the player to react quickly are providing a form of twitch challenge, where the player must make moment-to-moment decisions based on a world unfolding in front of them. Whereas games that use PCG to force the player to react, but not necessarily in a time-sensitive way, offer challenge in that the exact content being experienced has not been seen before by the player. Similarly, *Practicing in Different Environments* allows the player to experience new challenges (though, the same *kind* of challenges) through playing the same game in several different environments.

We can also consider dynamic difficulty adjustment as related to the *Memorization vs. Reaction* dynamic, in which the player is not consciously making a decision about how to guide the generator, but nonetheless is influencing how the generator will mete out challenges over the course of the game, thus altering the player's path and making it impossible for them to memorize what will come next.

When *Building Generator Strategies*, the game has added challenge through the layer of strategic play that comes in learning how the generator works, what it can create, and how to use that knowledge to the player's advantage.

Fellowship

PCG acts in support of fellowship through creating an emergent system that encourages player communication outside of the game environment, via the *Interaction with a Community of Players* dynamic. Not only does PCG seem to encourage player discussion, but it also shapes the ways that players talk about the game, including discussing

general strategies for different kinds of content configurations.

WORKED EXAMPLE

This section provides a worked example, illustrating how the framework can be used as a language for describing the combination of PCG system and game.

Endless Web

Endless Web has been used as a motivating example for several of the framework options thus far. It is a PCG-based game [50], designed intentionally so that its mechanics, dynamics, and aesthetics are heavily interdependent with the PCG system, and where the PCG system and game were iteratively co-designed. *Endless Web* uses a generator called Launchpad. It is a 2D platforming game that uses entirely procedurally generated content; players are tasked with exploring Launchpad's generative space while simultaneously exploring physical pace.

Mechanics

The Launchpad generator is a *grammar-based* generator. It provides *parameterized* control over both *compositional* and *experiential* aspects of content. The generator uses a *component* representation—it knows about individual level components and some constraints on how they fit together (e.g. enemies walk on top of platforms), but does not know about larger-scale common patterns of geometry. The generator's experiential control comes in the form of being able to manipulate level pacing *parameters*, by specifying a length of time and frequency/pattern of actions that the player takes during that time. Launchpad is used to perform *online* level generation; new level segments are generated whenever the player makes a choice that influences the generator, and in front of the player while she is running through the world. *Endless Web* can theoretically produce an infinite world for the player to explore.

Dynamics

As a platformer, the generated levels make up a *core* aspect of the way players interact in the game. There are two main dynamics that emerge from this combination of mechanics: *Searching a Vast Space*, and *Building Generator Strategies*. By being given control over the generator's content composition, the player is exploring both a physically infinite space and a very large (though not infinite) generative space. She is tasked with searching for specific goals that are hidden at different configurations of this generative space, so she must manipulate the generator to find these goal points over the course of the game. This goal-oriented play also contributes to the dynamic of *Building Generator Strategies*. The player will have to experiment with different configurations of generation parameters that are at an acceptable difficulty level, and can choose to push the generator in different directions to find power-ups that can help through more difficult generated content. Thus, the player is expected to form strategies around which goal to search for next, and which directions to push the generator to both maintain an appropriate difficulty level and move towards the next goal point.

Aesthetics

These two dynamics lead to two aesthetics for the game. The primary aesthetic is *discovery*, brought about by the fact that the game actively encourages the player to search a generative space as well as physical space. The game is about the wonder and confusion of exploring a constantly changing space, which the PCG contributes to heavily. A secondary aesthetic is *challenge*, which emerges from the strategies that the player needs to form and, to a lesser extent, from the platforming challenges themselves.

CONCLUSIONS

This paper has presented a framework for analyzing and discussing the role of PCG in game design and PCG-based design tools. The framework has been illustrated with numerous examples from both industry games and academic research projects, and has helped uncover nuances in the motivations for using PCG in game design, and how the style of control a user has over the PCG system influences their experience.

One of the goals of this research was to unpack the concept of "replayability" and critically examine how PCG is used to impact play experience. The framework has uncovered three primary dynamics that lead to different kinds of replayability: 1) reacting in a surprising environment, 2) building generator strategies, and 3) practicing in different environments. The first of these dynamics leads to replayability in that the entire purpose of the game is to play different content on each attempt, typically in order to beat a high score or progress further than in prior attempts. The third leads to replayability by providing content that supports replay of more traditional mechanics as a way to practice in different scenarios. Both the first and third dynamics could be plausibly attained through the use of large amounts of human-authored content. While the second dynamic does lead to replayability (through experiencing different content on each play through and having the opportunity to build strategies around the generator), it is also a kind of game dynamic that is unique to what PCG can offer.

It is important to note that there are some aspects of how the player controls the content generator that are not prioritized in the framework. In particular, no difference is explicitly drawn between intentional and unintentional control over how the generator is being controlled, it simply provides a language for the style of control from the generator's point of view. Whether or not the player is explicitly provided with control over generated content is a matter for how the game is designed, and irrelevant to the generator. The distinction was not made more explicit because it does not drastically impact game dynamics and overall play experience; in cases such as *Galactic Arms Race* [16], where the player is (supposedly) unintentionally influencing the next generation of weapons, the dynamic of

building generator strategies is still present as the player still takes actions that rely upon the PCG's existence and ability to be directed by the player. One might think that intentional control might be required for a PCG-enabled game design tool, and certainly some aspects of control must be intentional—however, one of the goals of using PCG in design tools is to help the designer brainstorm different design variants, where lack of intention in control and resulting surprise over produced content is a benefit.

The presented framework finds broader relevance in the area of AI-enabled creativity and design support tools outside of the domain of games. For example, the Picbreeder [43] tool for cooperating with a computer to create 2D evolutionary art can be described using the same set of "mechanics" aspects: patterns emerge from the underlying *subcomponent* representation that is not expected to be recognizable to the creator (our "player"), is performed *online* in that creators are building upon each others' work, uses a *preference*-based interaction type, and allows users to exert *compositional* control over its output. In future work, I intend to perform a broader survey of existing computationally creative tools to see how well the framework extends, as well as use the framework as a basis for exploring new potential tools, games, and research projects that use unique—or even undiscovered—forms of mechanical control to create new kinds of user experiences.

REFERENCES

1. [adult swim games]. *Robot Unicorn Attack (PC Game)*. 2010.
2. Bjork, S. and Holopainen, J. *Patterns in Game Design (Game Development Series)*. Charles River Media, 2004.
3. Bjork, S. Procedurally Generated Game Worlds. *Game Design Patterns 2.0*. http://gdp2.tii.se/index.php/Procedurally_Generated_Game_Worlds.
4. Butler, E., Smith, A.M., Liu, Y.-E., and Popovic, Z. A Mixed-Initiative Tool for Designing Level Progressions in Games. .
5. Church, D. Formal Abstract Design Tools. *Gamasutra [Online]*, 1999. http://www.gamasutra.com/view/feature/3357/formal_abstract_design_tools.php.
6. Cook, D. The Chemistry Of Game Design. *Gamasutra*, 2007. http://www.gamasutra.com/view/feature/129948/the_chemistry_of_game_design.php.
7. Cutumisu, M., Onuczko, C., McNaughton, M., et al. ScriptEase: A generative/adaptive programming paradigm for game scripting. *Science of Computer Programming 67*, 1 (2007), 32–58.
8. Dormans, J. Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games (co-located with FDG 2010)*, (2010).
9. Dormans, J. Engineering Emergence: Applied Theory for Game Design. 2012.
10. Doull, A. The Death of the Level Designer: Procedural Content Generation in Games. *ASCII Dreams: A Roguelike Developer's Diary*, 2008. http://roguelikedeveloper.blogspot.com/2008/01/death-of-level-designer-procedural.html.
11. Ebert, D.S. *Texturing & Modeling: A Procedural Approach*. Morgan Kaufmann, 2003.
12. Fernández-Vara, C. Personal communication, creator of Symon game. (2012).
13. Firaxis Games. *Civilization IV (PC Game)*. 2K Games, 2005.
14. Gearbox Software and Feral Interactive. *Borderlands (XBox 360)*. 2K Games, 2009.
15. Gee, J.P. *What video games have to teach us about learning and literacy*. Palgrave Macmillan, New York, NY, 2003.
16. Hastings, E.J., Guha, R.K., and Stanley, K.O. Automatic Content Generation in the Galactic Arms Race Video Game. *IEEE Transactions on Computational Intelligence and AI in Games 1*, 4 (2009), 245–263.
17. Hendrikx, M., Meijer, S., Van der Velden, J., and Iosup, A. Procedural Content Generation for Games: A Survey. *ACM Transactions on Multimedia Computing, Communications and Applications*, (2011).
18. Hullett, K. and Whitehead, J. Design Patterns in FPS Levels. *Proceedings of the 2010 International Conference on the Foundations of Digital Games (FDG 2010)*, (2010).
19. Hunicke, R., LeBlanc, M., and Zubek, R. MDA: A Formal Approach to Game Design and Game Research. *Proceedings of the 2004 AAAI Workshop on Challenges in Game Artificial Intelligence*, AAAI Press (2004).
20. Interactive Data Visualization Inc. *SpeedTree (PC Software)*. Lexington, SC, 2010.
21. Jennings-Teats, M., Smith, G., and Wardrip-Fruin, N. Polymorph: A Model for Dynamic Level Generation. (2010).
22. Jupiter. *Picross DS (Nintendo DS)*. Nintendo, 2007.
23. Khaled, R., Nelson, M.J., and Barr, P. Design metaphors for procedural content generation in games. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, (2013), 1509–1518.
24. Level-5. *Professor Layton and the Curious Village (Nintendo DS)*. Nintendo, 2008.
25. Liapis, A., Yannakakis, G.N., and Togelius, J. Sentient sketchbook: Computer-aided game level authoring. *Proceedings of ACM Conference on Foundations of Digital Games*, (2013).
26. Maxis. *Spore Creature Creator (PC Game)*. Electronic Arts, 2008.
27. Maxis. *Spore (PC Game)*. Electronic Arts, 2008.
28. McNaughton, M., Cutumisu, M., Szafron, D., Schaeffer, J., Redford, J., and Parker, D. ScriptEase:

Generative design patterns for computer role-playing games. *Automated Software Engineering, 2004. Proceedings. 19th International Conference on*, (2004), 88–99.
29. Mortac. The Complete Guide to Map Generation. *Civ Fanatics Forums*, 2007. http://forums.civfanatics.com/showthread.php?t=246788&s=b667c50191bb40c93402f38a7560b7ac.
30. Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. Procedural Modeling of Buildings. *ACM Transactions on Graphics 25*, 3 (2006), 614–623.
31. Nintendo EAD. *Animal Crossing: New Leaf (Nintendo 3DS)*. Nintendo, 2013.
32. Nitsche, M., Ashmore, C., Hankinson, W., Fitzpatrick, R., Kelly, J., and Margenau, K. Designing Procedural Game Spaces: A Case Study. *Proceedings of FuturePlay 2006*, (2006).
33. Ohkubo, H. *Warning Forever (PC Game)*. Hikware, 2003.
34. Persson, M. *Infinite Mario Bros! (PC Game)*. http://www.mojang.com/notch/mario/, 2008.
35. Persson, M. *Minecraft (PC Game)*. 2011.
36. Rareware. *Donkey Kong Country 2: Diddy's Kong Quest (Nintendo DS)*. Nintendo, 1995.
37. Regier, J. and Gresko, R. Random Asset Generation in Diablo 3. *Invited Talk, UC Santa Cruz*, (2009).
38. Risi, S., Lehman, J., D'Ambrosio, D.B., Hall, R., and Stanley, K.O. Combining Search-Based Procedural Content Generation and Social Gaming in the Petalz Video Game. *Proceedings of the 2012 Conference on Artificial Intelligence and Interactive Digital Entertainment*, (2012).
39. Rogue Basin. Articles on Implementation Techniques. http://roguebasin.roguelikedevelopment.org/index.php?title=Articles#Implementation.
40. Rohrer, J. *Inside a Star-Filled Sky (PC Game)*. 2011.
41. Saltsman, A. *Canabalt (PC Game)*. Adam Atomic, 2009.
42. Schell, J. *The Art of Game Design: A book of lenses*. Morgan Kaufmann, 2008.
43. Secretan, J., Beato, N., D'Ambrosio, D.B., et al. Picbreeder: a case study in collaborative evolutionary exploration of design space. *Evolutionary Computation 19*, 3 (2011), 373–403.
44. Shaker, N., Nicolau, M., Yannakakis, G., Togelius, J., and O'Neill, M. Evolving Levels for Super Mario Bros Using Grammatical Evolution. *IEEE Transactions on Computational Intelligence and Games (CIG)*, (2012).
45. Shaker, N., Yannakakis, G.N., and Togelius, J. Towards Automatic Personalized Content Generation for Platform Games. *Proceedings of the Sixth Artificial Intelligence in Interactive Digital Entertainment Conference (AIIDE10)*, (2010).
46. Smelik, R., Galka, K., de Kraker, K.J., Kuijper, F., and Bidarra, R. Semantic constraints for procedural generation of virtual worlds. *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, ACM (2011).
47. Smelik, R.M., Tutenel, T., de Kraker, K.J., and Bidarra, R. Integrating Procedural Generation and Manual Editing of Virtual Worlds. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games (co-located with FDG 2010)*, (2010).
48. Smith, A.M., Andersen, E., Mateas, M., and Popovic, Z. A Case Study of Expressively Constrainable Level Design Automation Tools for a Puzzle Game. *Proceedings of the 2012 Conference on the Foundations of Digital Games*, (2012).
49. Smith, G. and Harteveld, C. Procedural Content Generation as an Opportunity to Foster Collaborative Mindful Learning. *Workshop on Games and Learning, co-located with Foundations of Digital Games 2013*, (2013).
50. Smith, G., Othenin-Girard, A., Whitehead, J., and Wardrip-Fruin, N. PCG-based Game Design: Creating Endless Web. *Proceedings of the International Conference on the Foundations of Digital Games*, ACM (2012), 188–195.
51. Smith, G., Whitehead, J., Mateas, M., Treanor, M., March, J., and Cha, M. Launchpad: A Rhythm-Based Level Generator for 2D Platformers. *IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG) 3*, 1 (2011).
52. Smith, G., Whitehead, J., and Mateas, M. Tanagra: Reactive Planning and Constraint Solving for Mixed-Initiative Level Design. *IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG), Special Issue on Procedural Content Generation 3*, 3 (2011).
53. Sonic Team. *Sonic the Hedgehog (Genesis)*. SEGA, 1991.
54. Sullivan, A. Content Selection vs. Content Generation. *Expressive Intelligence Studio*, 2010. http://eis-blog.ucsc.edu/2010/06/content-selection-vs-content-generation/.
55. Togelius, J., Kastbjerg, E., Schedl, D., and Yannakakis, G.N. What is procedural content generation?: Mario on the borderline. *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, (2011), 3.
56. Togelius, J., Yannakakis, G.N., Stanley, K.O., and Browne, C. Search-Based Procedural Content Generation: A Taxonomy and Survey. *Computational Intelligence and AI in Games, IEEE Transactions on 3*, 3 (2011), 172 –186.
57. Toy, M., Wichman, G., Arnold, K., and Lane, J. *Rogue (PC Game)*. 1980.
58. Yannakakis, G.N. and Togelius, J. Experience-Driven Procedural Content Generation. *IEEE Transactions on Affective Computing 2*, 3 (2011), 147–161.
59. Yu, D. *Spelunky (PC Game)*. 2009.