PCG-Based Game Design: Creating *Endless Web*

Gillian Smith, Alexei Othenin-Girard, Jim Whitehead, Noah Wardrip-Fruin
Center for Games and Playable Media
University of California, Santa Cruz
{gsmith, ejw, nwf}@soe.ucsc.edu aothenin@ucsc.edu

ABSTRACT
This paper describes the creation of the game *Endless Web*, a 2D platforming game in which the player's actions determine the ongoing creation of the world she is exploring. *Endless Web* is an example of a PCG-based game: it uses procedural content generation (PCG) as a mechanic, and its PCG system, Launchpad, greatly influenced the aesthetics of the game. All of the player's strategies for the game revolve around the use of procedural content generation. Many design challenges were encountered in the design and creation of *Endless Web*, for both the game and modifications that had to be made to Launchpad. These challenges arise largely from a loss of fine-grained control over the player's experience; instead of being able to carefully craft each element the player can interact with, the designer must instead craft algorithms to produce a range of content the player might experience. In this paper we provide a definition of PCG-based game design and describe the challenges faced in creating a PCG-based game. We offer our solutions, which impacted both the game and the underlying level generator, and identify issues which may be particularly important as this area matures.

Categories and Subject Descriptors
I.2.1 [**Artificial Intelligence]** Applications and Expert Systems – *Games*. K.8.0 [**Personal Computing]** General – *Games*.

General Terms
Design.

Keywords
Game design, procedural content generation.

1. INTRODUCTION
The goal of procedural content generation (PCG) is often to mimic a human author as well as possible. Indeed, there has been a great deal of research in how to produce believable results in a wide variety of game content domains, from small elements such as rocks and trees [7,12] to levels and environments [14,18] to quests and stories [2,6]. With few exceptions, these systems are used in games to replace or augment human-authored content as seamlessly as possible. Many games that incorporate PCG are designed as though all the content could have been authored by humans, given sufficient development time.

However, PCG offers additional opportunities for game design: a PCG system can be employed as an on-demand game designer, capable of crafting a unique experience for each player. The most established tradition of such games is known as *Roguelikes,* after the game *Rogue* created at UC Santa Cruz and UC Berkeley around 1980 [23]. This is an unusual game category, because it is not defined by an element of the moment-to-moment gameplay, the game audience, the game platform, or the game theme. Rather, Roguelikes are defined by the fact that they use PCG to create content (generally levels) that the player may encounter through a wide variety of mechanics (e.g., the RPG action of *Diablo* [3] or the platforming of *Spelunky* [25]). The argument for PCG in Roguelikes is not that they enable a new form of gameplay, but that they can provide variety, replayability, and (especially important for early Roguelikes) compact representation of a wide variety of potential content.

However, it is possible to go further. Rather than using PCG to create fresh levels for fixed gameplay systems, one can seek to make the PCG system itself a focus of gameplay, reacting rapidly to player behavior — something the player comes to understand and manipulate through play. Creating such a game requires the careful co-design of both game and PCG system, as each has affordances, limitations, and requirements that influence the design of the other. For example, any bias in content created by the generator must be worked into the design of the game. Similarly, the game's design is likely to push additional requirements on the content that the generator should be able to create. The PCG system is so deeply linked to the mechanics and aesthetics [11] of the game that the player can form strategies around the generator's actions and PCG-based dynamics emerge. We call these *PCG-based games.*

Endless Web is a 2D platforming game designed as an exploration of PCG-based game design. It was originally designed to use the *Launchpad* rhythm-based level generator [20], which was heavily modified over the course of the game's development to meet new design requirements. Launchpad supports the tuning of its anticipated output with variables that have a clear impact on the levels it designs, including the appearance of different components and level pacing. *Endless Web* uses Launchpad to generate a world that adapts to choices the player makes throughout the game, as they seek goals that are hidden in layers of Launchpad's generative space.

Most games permit designers and art teams to have complete, fine-grained control over the structure and appearance of their content (e.g., the progression of levels and encounters within them). The systems that produce emergence within games, such as interacting combat mechanics or storytelling choices, are introduced and exercised through carefully-constructed content combinations. Building a PCG-based game instead involves relinquishing direct control over content and treating content generation as a game mechanic. The design of a PCG-based game introduces a number of new design problems, for both the game and the PCG system itself. Game design issues, from the moment-to-moment pacing of a level to difficulty curves, can no longer be solved through the direct manipulation of game content. The

FDG '12, May 29-June 1, 2012 Raleigh, NC, USA.

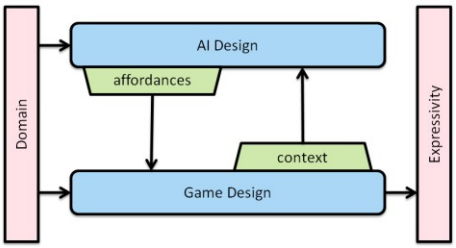

Figure 1. A diagram describing the AI-based game design process used when creating Endless Web, where Launchpad is the AI system.

designer must instead design entire ranges of content that can be meaningfully controlled by both the designer and the player.

A number of design challenges arose while creating *Endless Web* from the use of PCG as the core mechanic. The primary challenge came in determining how to balance this control over the generator between the player and the designer, so that the player makes meaningful decisions but all of those decisions lead to an engaging, designed experience. Other design problems include teaching the player to explore a generative space, providing well-placed goals to encourage this exploration, and art and audio issues that arose from having no knowledge at design time of how levels would be structured during play.

The primary contributions of this paper include a description of the design challenges arising from designing a PCG-based game and solutions to these problems in terms of both the game and its underlying PCG system. These challenges are explored via an examination of the design decisions made in *Endless Web* and how Launchpad was modified to accommodate the game's design. We also provide a definition of PCG-based games and an overview of the current use of PCG in games.

2. THE DESIGN OF ENDLESS WEB

The design process for *Endless Web* followed the principles of AI-Based Game Design [8], with Launchpad as the AI system. Launchpad's existing design offered two main design affordances: variables that could be exposed during gameplay and a rhythm model that supported altering pacing. The design process began by examining these affordances (Figure 1); however, designing the game quickly uncovered further requirements for Launchpad. This section describes the design of Endless Web and how it was informed by Launchpad, and vice versa.

2.1 Endless Web

Endless Web is a 2D platforming game in which the player takes the role of a member of a fictional race called Eidolons. Eidolons inhabit humanity's collective Dream; when humans fall asleep and dream, we wake up in their world. Humans and Eidolons share a symbiotic relationship, as nightmares disrupt the fabric of the Eidolons' world — they must seek out and wake up people who are trapped in their nightmares, thus releasing dreamers from their fears.

The goal of the game is for the player to rescue six dreamers who are trapped in their nightmares by exploring the Dream to find them. The six different nightmares are: *Body Horror, Broken Hearts, Creepy Crawlies, Dolls and Roses, Faceless Crowds,* and *Time and Death.* Exploration is presented to the player as physical exploration: jumping onto springs, falling through tubes, or ascending in beams of light to discover new areas of the world. If the player continues moving to the left or right, an infinite world of gameplay unfolds in front of them. However, players are simultaneously exploring the *generative* space, i.e. the many different kinds of levels that Launchpad is capable of producing by tuning different parameters. Each time the player interacts with a special, glowing "tuning portal" the game propels her into a new part of the world that has been generated for her based on the choices she has made so far and how far she has progressed through the game. For example, the *enemies* tuning portal involves killing a special glowing enemy and then ascending in a beam of light to newly generated terrain. Depending on whether the portal was configured to strengthen or weaken its associated challenge (a setting the player can decide before entering the portal), the level will contain either a greater or lesser frequency of enemies at a greater or lesser difficulty level. These choices lead players to a wide range of content during the course of the game; Figure 2 shows three drastically different configurations of the world based on the choices the player has made.

There are six different tuning portals in the world that correspond to each challenge type (Figure 3). Each challenge type represents a different fear or negative emotion. Each fear has three different tiers of difficulty, which the player will encounter in order as they continue exploring deeper into a particular fear. For example, the *enemies* challenge type can be realized as (in order of increasing difficulty): a moving enemy on a platform, an enemy who jumps out at the player, and an enemy that hurls projectiles. Each dreamer is hidden at a combination of tiers for three different fears; for example, the dreamer having a nightmare about creepy crawlies is trapped at the intersection of the fears of losing control (tier 1), stress (tier 1), and conflict (tier 2). When the player reaches that intersection, the game presents her with a human-authored level representative of an individual dream that she must traverse to find the dreamer at the end of it. These six levels associated with the dreamers are the only levels in the game authored by a human as opposed to the level generator. Human authoring means that these level segments can be more surprising to the player and offer more challenging combinations of content – they contain configurations of geometry that are outside the

Figure 2. Three different configurations of the Dream. The leftmost screenshot shows two stompers and an enemy patrolling the long, unbroken platform. The middle shows a world with a lot of springs from all the tiers of difficulty. The rightmost has platform hazards and gaps. The background color of the world shifts colors to reflect the predominant challenges and difficulty tiers.

capability of the generator, such as overlapping platforms or platforms made entirely out of springs.

In addition to the primary goals of finding all the dreamers, there are six powerups scattered throughout the generative space as secondary goals. The powerups are hidden at a combination of tiers for only two different fears, so they are easier to reach while exploring to find the dreamers. Collecting a powerup unlocks a special ability that eases level exploration. Each powerup is related to a challenge type: *enemies/shield, platform hazards/place block, springs/float, gaps/double jump, moving platforms/time slow,* and *stompers/dash*. Because of the relationship between challenges and powers, each powerup is placed at the second tier of its challenge and the first tier of a different challenge. Figure 4 shows the visual representation of the generative space and both the primary and secondary goals; this visualization is presented to the player as a map that is accessible at any time during the game. The player also sees a "mini-map" in the top right corner of the screen during the game that reflects the player's current position in each of the color-coded fears

The player's progress towards the different dreamers is reflected in both the art and music. Each level component has seven different art representations: one for each of the dreamers, and one for the undisrupted world (Figure 5 shows the seven different art

Figure 5. The six different kinds of tuning portals, left to right: *enemies/conflict, platform hazards/betrayal, springs/entering the unknown, gaps/failure, moving platforms/losing control*, and *stompers/stress*.

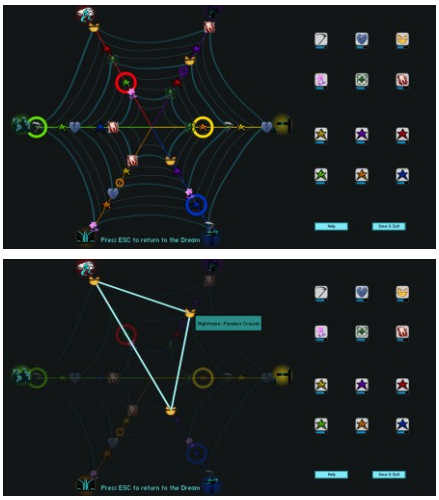

Figure 6. The Web: a visualization of all the different goals available to the player that serves as a map of the generative space. The player has six primary goals to collect the dreamers (upper right) and six secondary goals to collect the powerups (lower right). The six colored circles on each strand of the Web signifies the player's current location – circles are larger when they are at an important location along an axis. When the player mouses-over an icon, all identical icons are highlighted.

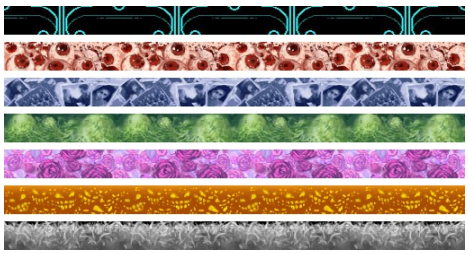

Figure 3. The seven different art assets for non-hazardous platforms. The top asset is for the undisrupted Eidolon world, the remainder correspond to the nightmares (top to bottom): *Body Horror, Broken Hearts, Creepy Crawlies, Dolls and Roses, Faceless Crowds*, and *Time and Death*.

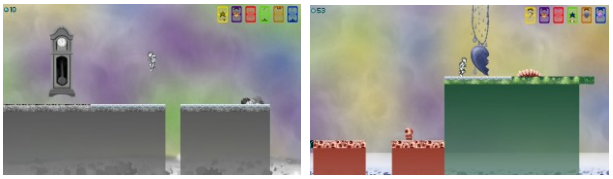

Figure 4. As the player approaches a dreamer, the art style slowly coalesces to that of the nightmare (left: near the *Time and Death* nightmare). When further away from any one dream, the art is more jumbled (right: a level showing art for *Body Horror*, *Broken Hearts*, and *Creepy Crawlies*).

assets for normal platforms). Tile art is procedurally selected at play time based on the player's proximity to the different dreamers. As the player gets closer to dreamers, the visual aesthetic coalesces towards that particular dreamer; for example, when the player is only a short distance away from the *Time and Death* nightmare, the level is likely to look like the screenshot in Figure 6a, but when the player is a far distance from all the dreamers, the art is far more jumbled (Figure 6b). Once a dreamer has been rescued, the art associated with that dreamer is no longer shown and is replaced with the art for the undisrupted world. The music played is also procedurally selected; each time the player makes a choice, the game selects a track to overlay in the game based on that choice. The more challenging the configuration of the generator, the more chaotic the music sounds as there are more tracks that are layered over each other at once.

2.2 Launchpad

Launchpad is a rhythm-based level generator for 2D platforming games that uses grammars to construct both the rhythm that dictates intended player actions and the geometry that matches that rhythm. A designer has input to the generator in the form of parameters that specify the general path that the level should follow and the frequency with which different level components should occur. Launchpad guarantees that levels will be playable with its underlying physics system, which knows the maximum speed the avatar can move, how high it can jump, and the sizes of different level components. The key underlying principle in Launchpad is a separation of the pacing of a level from the geometry in it: it is possible for designers to control level pacing largely independently from the composition of level geometry. A more detailed description of the original version of Launchpad has been published previously [20].

While designing *Endless Web,* Launchpad underwent a number of changes to support game design decisions. The largest change was the introduction of the tiered difficulty system mentioned earlier;

the original version of Launchpad was agnostic about the difficulty of challenges it presented to a player. The new version takes additional parameters for the *tier* of each challenge, which the generator uses to determine the kinds of components it will insert in the level.

The influence of precise timing on level component placement was also relaxed, to create a higher variety of content when certain restrictive elements were needed. For example, Launchpad's expressive range is diminished when asking it for levels composed mainly of springs [21]. By relaxing timing constraints so that the time spent in the air is no longer factored in, it is possible to use springs for a wider range of beats.

3. PCG-BASED GAME DESIGN

Endless Web was designed to be an entirely PCG-based game, which we define 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.

Hunicke et al. describe an approach to game design and analysis called Mechanics, Dynamics, and Aesthetics (MDA) [11]. In this framework, the mechanics are the core rules of the game and the aesthetics are the desired emotional response that should be evoked in the player. The dynamics sit between the mechanics and aesthetics as the "run-time behavior of the mechanics acting on player inputs and each others' outputs over time".

PCG has been used in games for many reasons, including addressing technical limitations, improving replayability, and providing infinite worlds for the player to explore. We use the definition of a PCG-based game as a lens with which to examine existing games that include PCG elements, exploring whether they do or don't satisfy the elements of the definition.

3.1 Non PCG-Based Games

There are three major ways that PCG has traditionally been used in digital game design: to improve replayability, to resolve technical limitations, and to provide players with an environment to explore that is so vast it could not be created by human designers.

As discussed earlier, *Rogue* [23] and its descendent *roguelikes* (e.g. *Diablo* [3] or *Spelunky* [25]) all use PCG to enhance replayability. Both the mechanics and aesthetics of these games are largely unrelated to the PCG system; while players are aware that they have a different experience each time they play, the use of PCG does not affect their playing experience, with the notable exception of making failure "permanent" for any generated world. The *Civilization* games also use PCG to improve replayability; however, it is also forms an important aspect of the early game experience. The maps generated by the game influence strategies that players can take. For example, if a player's starting area does not have a particular important resource, they might choose to expand towards a known source of the resource or negotiate a trade with another civilization. However, while important, the PCG does not make up the core game experience for most players. Most strategies are less related to the generated environment and more about building certain technologies and resource allocation. *Civilization IV* [9], for example, comes with several human-authored maps that still provide a great deal of replayability through these other, non-PCG systems.

Elite [5] is an example of a game that uses PCG as a form of data compression; with a platform that allowed only 22K of memory, it was impossible for the designers to author and store all the content they wished for the game. However, while the galaxies and worlds of *Elite* were algorithmically generated, they were entirely static – the designers had the ability to curate the content that was created [4], and had retained control over the world the players would explore. While the aesthetics of the game—the player's awe at being able to explore a huge universe—were a result of using PCG to compress the world, the mechanics were in no way related to the PCG system. The technique of using PCG for compression is also common in demoscenes. For example, the game .kkrieger [1] is a first-person shooter that uses only 96K of disk space.

Minecraft [16] and *Terraria* [22] are modern examples of using PCG for both replayability and exploration (as in *Elite*). The player is provided with an inconceivably vast world to explore, and the formation of the world guides the player's strategies. However, again, the PCG system is not influenced by the player, so no PCG-based dynamics can arise.

3.2 PCG-Based Games

There are a small number of games that we do consider PCG-based. For example, in *Galactic Arms Race* [10] the players' battle strategies determine the next round of weapons that will be available to them; the creators of the game have shown that there is a large variety in the kinds of weapons players will customize. Because it is a multiplayer game, players can see the different kinds of guns they could have created if they played with different tactics.

Inside a Star-Filled Sky [17] is a another good example of a game whose mechanics are built around PCG. Players navigate a space in which they can zoom into or out of recursively nested levels, each one generated from a seed passed from an object in a higher or lower level. Play events that take place on one level of play affect levels above and below themselves.

Both *Galactic Arms Race* and *Inside a Star-Filled Sky* use PCG as a game mechanic; however, both are different from *Endless Web* in one crucial aspect: the player's direct influence over content. Actions players take in these two games have, at most, an indirect effect on the content generator: the games are designed so that the generators are entirely invisible to the player. *Endless Web* intentionally places the generator as something that players are encouraged to strategize around when determining the path they should take through the generative space to reach different goals.

3.3 Endless Web as a PCG-Based Game

Recalling the definition of a PCG-based game given earlier in this section, understanding *Endless Web* as a PCG-based game requires an analysis in terms of its mechanics, dynamics, and aesthetics. Mechanics-wise, *Endless Web* is consciously positioned within the traditions of the 2D platformer. The basic actions available to the player are those typical of existing platformers, including jumping over gaps, killing enemies, and avoiding stompers. The level elements used in the game are also commonly seen in traditional platformers; enemies patrol back and forth along platforms, stompers descend from the "ceiling" of the level, and springs propel the player to greater heights. However, one of the main core mechanics of *Endless Web* is not one of these platforming mechanics. *Endless Web* is fundamentally a game about manipulating a generative space, and the core mechanic involves the player deliberately choosing to influence the generator in different directions through interacting with the glowing tuning portals. This core mechanic is intertwined

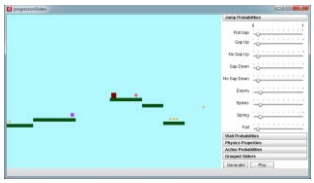

Figure 7. The technical prototype built to understand the impact of altering parameters on the player's experience. Sliders on the right control every parameter of the generator. The area on the left allows the designer to play levels as soon as they are generated.

with the generator; each tuning portal the player interacts with directly changes parameters to the generator.

A key aesthetic in *Endless Web* is a sense of exploration and wonder, both in terms of physical exploration and uncovering the generative space of the game. As the player progresses through the game, the generator provides increasingly challenging combinations of content, and the variety of potential content is quite high. The world the player is exploring is infinite; as mentioned above, the player could choose to keep moving in one direction forever and the generator would continue to provide content appropriate to the choices the player has made. This aesthetic could not be achieved without the use of procedural content generation—there is a limit to the amount of human-authored content that can be made for a game and thus the extent to which players can explore a world, and the generator makes it possible to remove this limitation.

This aesthetic comes from the player's interactions with the generator using the tuning portals and desire to achieve the goals that are scattered throughout the generative space. The dynamics of the game involve the player building strategies around which direction to push the generator at any given time. To achieve each goal, the generator must be configured to a particular location in generative space, but there are many ways for the player to reach these locations. The player can choose to manipulate the generator in different ways in order to achieve goals in different orders, or to reduce the difficulty of a particular challenge that isn't needed for the current goal. Thus, the player is building strategies around the procedural content generator to make sure that the content seen is of an appropriate challenge and interesting composition.

4. DESIGN CHALLENGES

Building a game around a PCG system introduces some unique design challenges, including how to best integrate the particular PCG system into the game, how to design for always having desirable content while still giving the player control, and how to teach the player to understand entirely new rules within common genre conventions. Some problems arise from the abilities and biases of the generator, while others arise from technical decisions made to support an infinite world. This section discusses these challenges in the context of designing *Endless Web*.

4.1 Using PCG as a Mechanic

The primary design goal for *Endless Web* was to create a game that could not exist without procedural content generation. The design process began with an analysis of the capabilities of Launchpad, with a view to how these capabilities afford new game mechanics. Launchpad's key feature is how the input parameters are directly tied to the levels that it can create, and altering these parameters was identified as a core mechanic. Early design ideas involved altering the parameters indirectly; for example, one proposed idea was a platformer game for Facebook where beat density would be determined from the frequency of status updates and components would be selected based on the tone of posts. However, in keeping with the definition of a PCG-based game as one whose dynamics are strongly influenced by the procedural content generator, we decided instead to provide the player with relatively direct control over Launchpad's input parameters, thus offering players the opportunity to build strategies around the generator.

This also fit in with the desired aesthetics for the game. PCG is well-suited for an exploration game due to its capability for creating new territory for the player to explore infinitely in any direction. By giving players control over Launchpad's input parameters, they are able to explore not only the physical space of the world but also Launchpad's *generative space*.

However, it is important that the parameters made available to the player are not confusing or overwhelming. An early technical prototype for *Endless Web* exposed every parameter to Launchpad as sliders on the right side of the screen to make it easy to rapidly play with different parameter configurations (Figure 7). This prototype was useful in determining which parameters were most obvious when changed, and the relationships between these parameters. For example, there are parameters for the frequency of wait and jump actions in a rhythm that are set separately from the parameters for different geometry components, but these parameters are dependent on each other. While previous work [21] had shown relationships between Launchpad's parameters through an analysis of thousands of generated levels, creating this prototype made it possible to understand the ramifications of these dependencies on the player's experience. The parameters chosen for the player to manipulate were those that provided the greatest variation in player experience while minimizing potential undesirable dependencies (e.g. we do not allow players to modify the physics of the game despite being a noticeable change because it has too many implications on the availability of different level components).

4.2 Balancing Control for a Generative Space

Using PCG as a mechanic requires the player to have control over the kind of content that appears in the game. However, there is a measured amount of control that the designer should retain over the generator as well, to handle design concerns such as difficulty or pacing. In *Endless Web,* the game state determines the value of certain parameters to the generator while the player controls others through play. The player retains a great deal of control over the generator, but within the boundaries set by the game's design.

Launchpad's separation of pacing from level components seemed a natural fit for drawing a boundary between player and designer control. Thus, the player was provided with control over the components that appear in the level, rather than rhythm parameters. Control over the pacing of the level was kept in the game, as level pacing is a global consideration related to difficulty, which we wanted to be able to manipulate ourselves as game designers rather than leave in the hands of the player. Any PCG-based game must have a similar separation in its underlying generator so that the designers still have control over some aspect of player experience.

4.2.1 Designer Control: Difficulty and Pacing

Retaining control over difficulty and pacing was a response to playtesting feedback from an earlier iteration of the game [19]. There was not enough variety in the levels being created, and there was no clear difficulty progression. This problem was addressed in two ways: through the creation of a tiered difficulty

system in Launchpad, and by providing the game with control over the level's pacing.

Launchpad was originally designed to choose from all components according to a probability specified by the designer. The tiered difficulty system altered this by adding more components at different difficulty levels, and having the generator choose the difficulty of a component independently from the component type. *Endless Web* keeps track of player progress into a particular challenge and requests the appropriate tier of difficulty from Launchpad when the player uses a tuning portal. The player can choose whether the difficulty of that challenge will be increased or decreased before entering the portal by hitting a direction box (similar to a question mark box from *Super Mario World* [15]). Raising the difficulty of a challenge type raises the probability that a more difficult component will be chosen. This design decision also solved the problem of variety in levels by tripling the number of components available to the generator while maintaining uncertainty about which components will be chosen.

While variety and unpredictability in content is considered good, that variety must still be carefully structured to provide an engaging and fair experience with an appropriate level of challenge. *Endless Web* controls the rhythm and pacing parameters by slowly increasing them as the player reaches his goals. Level segments start out as short, slower paced segments and change (as dreamers are rescued) to be longer and faster paced. Note that increasing the parameters actually alters the *probability* that a segment will have the appropriate length and density, not the actual frequency itself. This adds some extra variation and can surprise the player with a more challenging than usual section early in the game, but on average the segments meet the intended difficulty curve.

4.2.2 Player Control: Level Structure

While the game retains control over the pacing of the generated levels, the player is given control over the overall composition of the world. Each time the player uses a tuning portal, the probability of the appearance for the corresponding component is altered. Launchpad then generates 50 different candidate level segments and returns the one that most closely meets the parameters for component frequency. This provides players with appropriate feedback about the changes they are making to the generative space, resulting in players feeling like they are controlling what they see in the game.

4.3 Navigating a Generative Space

The aesthetic of exploration in *Endless Web* is designed to feel natural and organic. The tuning portals are designed to be familiar level elements that, while clearly marked as transitions to a new location, still feel like they belong in the game world. Using familiar level components provides a cue to players about what they should expect the generator to do when they are used. We found that one of the most challenging aspects of navigating generative space is the lack of waypoints or landmarks that can be used when navigating a physical space, since all of the content is generated. This issue was further addressed through the use of art assets and audio to provide each configuration of the generative space with a unique "fingerprint" (Sections 2.1 and 4.6).

Important decisions that made generative space navigation less frustrating were made when designing the algorithm for how tuning portals should be scattered throughout the world. In *Endless Web,* to make exploration feel more organic and well-paced, there is only one tuning portal that appears between level segments, allowing the player to keep moving through the level to find more portals.

Originally, these portals were placed entirely randomly. However, the entirely random placement was jarring to players. One player described the experience as "being at the mercy of a random number generator". Players would frequently express frustration at being unable to find the one portal they needed to reach a goal. Tuning portal placement needed to be done *intelligently*, not randomly. Indeed, while *Endless Web* uses random numbers frequently, they are always part of a directed experience.

This problem was addressed by probabilistically placing tuning portals based on how likely the game judges the player is to need them. For example, if the player is only two portals away from reaching a goal, the game is more likely to show the player the portal that he needs, intentionally assisting the player in reaching the nearest goal. However, if the player ignores the portal and continues moving to the left or right, he will see the full sequence of portals before seeing any repeats. The set of available portals is reset whenever the player uses one. This guarantees that the player will always be able to find the portal he wants to use, but is more likely to see the portal that he needs.

4.3.1 Making the PCG Visible

A key issue in building games around AI systems is avoiding the Tale-Spin effect [24]; the underlying AI system must be transparent enough that the player can understand what it is doing. The Tale-Spin effect occurs when there is no "means for interaction that would allow audiences to come to understand the more complex processes at work within the system".

The decision to make exploration completely seamless means that the player never sees Launchpad in action. There is no geometry popping into the screen, and no way for the player to see Launchpad construct levels and choose between different options. Thus, *Endless Web*'s only path to avoiding this effect is making sure the player's choices have a clear and predictable effect on the world. Playtests have shown that this works well at lower tiers, when elements are first introduced. However, it is currently unclear how well this works at higher difficulty tiers, or when all the tiers are at equal values. When this occurs, the probability of each component appearing is so similar to the others that it can be hard to see an immediate impact from choices. We intend to better understand how players interpret the consequences of their choices in future work.

4.3.2 Reusing Genre Conventions

Another challenge faced in *Endless Web* was the player's interpretation of certain genre conventions. Experienced platformer players naturally tend towards moving to the right instead of the left, and assume that falling down gaps leads to death. Since three of *Endless Web*'s tuning portals transport the player downwards, and the player always has the ability to move to either the left or right at the end of a portal, this presented problems in our design. Early versions of the tuning portals involved the player falling down gaps. This issue was addressed by changing the downward moving portals to have the player move through a tube instead of falling in mid-air. Playtests run before and after these changes show that using tubes for downward portals reduces player confusion.

4.4 Goals for Exploring Generative Space

The dreamers in *Endless Web* are scattered throughout the generative space. Goals are placed in generative space rather than physical space to reinforce the design goal that players should be exploring the capabilities of the generator, and that physical

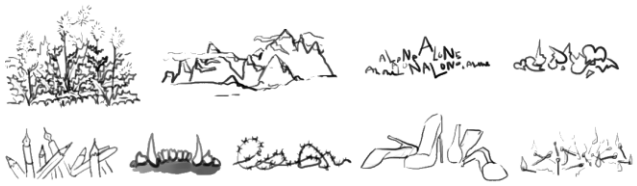

Figure 8. A selection from the 54 original concept art sketches for the spikes challenge.

position in space is immaterial. The tiered difficulty system offered seemingly obvious goal placement; with six challenge types and six dreamers, there was initially one dreamer at the end of each challenge, and the player would be rewarded for finding a dreamer by giving her a powerup.

However, placing goals at the end of the web strands was not interesting enough. It provided the player with no motivation to actually *explore* the generative space because they were always simply aiming to max out each axis. This problem prompted us to move goals to be hidden within "layers of the Dream" at different world configurations, and de-coupled powerups from the dreamer locations. While this added more complexity to the game, requiring the player to learn another system (section 4.5), this did successfully encourage players to explore the generative space and see more interesting configurations of content. The powerups became secondary goals that helped the player through the game, although they are not required. Since Launchpad has no knowledge of the powerup system, there is never a level that cannot be completed without a powerup.

4.5 Teaching the Player

The hardest aspect of designing *Endless Web* was teaching the player how to understand an entirely new game genre that, on the surface, appears identical to other, more familiar, platforming games. After the main part of the game was designed, we introduced a tutorial series that guided the player through the different systems of the game. The first phase of the tutorial teaches the player the controls and core platforming concepts, and requires the player to take the platform hazards tuning portal. This section of the tutorial is hand-authored. While it could have been made with a different level generator that would sufficiently control the content and placement of instructions, we don't expect players to replay the tutorial level once they understand the mechanics so a procedurally generated tutorial would offer little value over a hand-authored one.

After the player uses the platform hazards tuning portal, the game continues with procedurally generated content. The Eidolon is given a small goal to find his elder tutor, who is waiting at a minor configuration of the world. The player cannot see the entire Web at this point, only a small subsection of it. The tutorial familiarizes the player with the concept of using tuning portals to change the world configuration and aligning the world to a specific configuration. After the player finds the Elder, the rest of the game opens up and the player is given the main goal of finding all the dreamers.

Designing the Web was particularly challenging, as players expected it to be a physical map of the world. Many early playtesters assumed that the positions on the web corresponded to physical positions in the world, and that by moving upwards in space they would move "up" on the Web. This is another example of genre standards confusing players, in this case by repurposing physical direction to mean something completely different. It is not immediately clear to players that the world they are exploring is endless, as it looks identical to other platformers they have played in the past. To solve this problem, the Web was never referred to in the game or any tutorial text as a "map" and additional tutorial information was added to the Web screen itself.

4.6 Art Influence and Challenges

Having a visually compelling world was critical to our design goals; we wanted players to have the sense of being in a surreal dreamscape, surrounded by familiar nightmare motifs. Therefore there is human-created art incorporated into the game, rather than procedurally generated assets. These assets were designed to be modular so that they could be procedurally selected during play. Modular art design introduced a large challenge for the art team, as there is no way of knowing ahead of time the exact placement of platforms or enemies.

The art direction had a large influence on the final story of the game. One round of early concept art included 26 ideas for what spikes might look like – from broken hearts and spiky high heels to teeth and electric fences (Figure 8). The diversity in art we saw for spikes influenced our decision to introduce the nightmares to the game, and the role of the player shifted from being a single human dreamer to a creature who is trying to find and rescue multiple dreamers trapped in these nightmares.

Designing new art assets also revealed implicit assumptions and inconsistencies in Launchpad's physics system. Launchpad was designed to support changing physics parameters (e.g. the player's movement speed or the sizes of level components) without altering the generator at all. However, the original design for Launchpad had not accounted for constraints among these parameters, and when new sizes of art assets were added, the physics system in Launchpad broke due to failing to enumerate these constraints. For example, the height of enemies must be set such that they are no taller than the player's maximum jump height, otherwise the player will not be able to jump over them or jump on top of them. Using art assets that had a different ratio from the originally intended size for Launchpad revealed such inconsistencies, and prompted a redesign of the physics system such that these constraints are taken into account.

4.7 Technical Design Decisions

In addition to the game design issues detailed above, there were also two key technical design decisions that guided the creation of *Endless Web*: memory limitations for an infinite world and the potential for delayed response from the generator.

4.7.1 Memory Limitations

With an infinite world, it is impossible to store all of the generated content in memory. Although a stored seed could have re-generated content on demand, simply deleting all off-screen content provided a much simpler architecture and reduced any need for worrying about memory management. This architecture decision played a large role in initial story concept development. Endless Web's setting—the surreal landscape of dreams—also grew from the incorporation of procedural content generation and the limitation that content is generated off-screen and deleted off-screen, meaning that if the player turns around they will see different level geometry than was there before. Dreams provided a good setting that could explain these issues, since the landscape in dreams frequently shifts in unexpected ways.

4.7.2 Client/Server Split

While Launchpad is capable of producing levels extremely quickly—usually requiring less than a second—any lag in response from the generator must be handled in the game. This lag

was compounded by the use of a client/server architecture; Launchpad is a web service that is called by *Endless Web* whenever a new level segment is needed. A new segment is required once every 5-20 seconds. Having Launchpad as a web service provides flexibility in updating the game and testing it with different PCG techniques, and also permits gathering of gameplay metrics. However, it does introduce a potentially significant problem in handling server lag and lost connections.

Auto-saving the game at every transition point, and requiring each tuning portal to be able to extend indefinitely while waiting for server response, addressed this problem. For example, the *enemies* tuning portal involves the avatar being lifted up in a beam of light. This beam extends off-screen until the server has responded to the generation request. This maintains the sense of seamless exploration as long as possible. If the request fails, the player is placed in a human-authored level segment that tells the player they've lost their connection to the dream.

5. CONCLUSION

While the discussion thus far has largely been specific to *Endless Web*, there are a number of general lessons that can be drawn.

Balancing control over content. Dan Kline, speaking of designing the game *Darkspore*, noted that all aspects of the player's experience should be *directed*—purely randomized aspects of content stand out to the player as undesirable [13]. Control over the content generator must be balanced between the player and the game itself. When creating a PCG-based game, the role of the game designer shifts from being a creator of instances of content to an entire, parameterizable range of content. The challenge here comes in ensuring that, while the content the player experiences will be varied, the overall game experience is still appropriately controlled. For example, *Galactic Arms Race* retains control over when new weapons are evolved and seeds the pool with content that is known to be enjoyable.

Keeping the PCG visible to the player. It is crucial for players to understand the effect they have on the world, and this can only be done by making the use and consequences of PCG as visible to the player as possible. Without knowing that the world is being generated around them, players lose agency as the choices they make feel meaningless.

Building support for art. Building complete games requires effort from not only engineers and designers, but also artists and musicians. While procedurally generating art and music are possible and can lead to successful results [16,17], the resulting aesthetic is not always desirable. Thus, it is vital to build in support for art and music to the PCG system, and design the system to be flexible enough to accommodate a changing art or music style over the course of the game's design. For example, Launchpad was designed to create levels for variably sized content, which was helpful when testing different size art assets for *Endless Web*.

This paper has presented the design of the PCG-based game *Endless Web*, the challenges faced during the design process, and how these challenges were addressed. There are still very few PCG-based games in existence, and it is our hope that the lessons we have learned from creating *Endless Web* will be helpful as this field matures and new PCG-based games are created.

6. Acknowledgments

Endless Web was developed by a student team of engineers, designers, and artists. The authors extend thanks to Ari Burnham, Jon Gill, Rob Giusti, Umi Hoshijima, Masami Kiyono, Jameka March, Joshua Ray, Christian Ress, Rob Segura, and Vencenza Surprise, without whom the game would not exist. This work is supported by the National Science Foundation, grant no. 1002852. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

7. References

1. .theprodukkt. *.kkrieger (PC Game)*. 2004.
2. Ashmore, C. and Nitsche, M. The Quest in a Generated World. *Proceedings of the 2007 Digital Games Research Association (DiGRA) Conference: Situated Play*, (2007), 503–509.
3. Blizzard North. *Diablo (PC Game)*. Blizzard Entertainment, 1997.
4. Boyes, E. Q&A: David Braben -- from Elite to today. *GameSpot UK*, 2006. http://uk.gamespot.com/news/6162140.html?part=rss&tag=gs_news&subj=6162140.
5. Braben, D. and Bell, I. *Elite (BBC Micro)*. Acornsoft, 1984.
6. 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.
7. Dart, I.M., De Rossi, G., and Togelius, J. SpeedRock: procedural rocks through grammars and evolution. *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, ACM (2011), 8:1–8:4.
8. Eladhari, M.P., Sullivan, A., Smith, G., and McCoy, J. AI-Based Game Design: Enabling New Playable Experiences. *Technical Report, UCSC-SOE-11-27*, 2011. http://www.soe.ucsc.edu/research/technical-reports/ucsc-soe-11-27.
9. Firaxis Games. *Civilization IV (PC Game)*. 2K Games, 2005.
10. 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.
11. 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).
12. Interactive Data Visualization Inc. *SpeedTree (PC Software)*. Lexington, SC, 2010.
13. Kline, D. and Hetu, L. AI of Darkspore (Invited Talk). *2011 Conference on Artificial Intelligence in Interactive Digital Entertainment*. http://dankline.files.wordpress.com/2011/10/ai-in-darkspore-aiide-2011.pptx.
14. 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.
15. Nintendo EAD. *Super Mario World (SNES)*. Nintendo, 1990.
16. Persson, M. *Minecraft (PC Game)*. 2011.
17. Rohrer, J. *Inside a Star-Filled Sky (PC Game)*. 2011.
18. 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).
19. Smith, G., Gan, E., Othenin-Girard, A., and Whitehead, J. PCG-Based Game Design. *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games, co-located with FDG 2011*, (2011).
20. 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).
21. Smith, G. and Whitehead, J. Analyzing the Expressive Range of a Level Generator. *Proceedings of the Workshop on Procedural Content Generation in Games, co-located with FDG 2010*, (2010).
22. Spinks, A. *Terraria (PC Game)*. Re-Logic, 2011.
23. Toy, M., Wichman, G., Arnold, K., and Lane, J. *Rogue (PC Game)*. 1980.
24. Wardrip-Fruin, N. The Tale-Spin Effect. In *Expressive Processing: Digital Fictions, Computer Games, and Software Studies*. MIT Press, 2009.
25. Yu, D. *Spelunky (PC Game)*. http://www.spelunkyworld.com/, 2009.