A Procedural Approach for Infinite Deterministic 2D Grid-Based World Generation

Tanel Teinemaa
IT University of Copenhagen
Rued Langgaards Vej 7
Copenhagen, Denmark
ttei@itu.dk

Till Riemer
IT University of Copenhagen
Rued Langgaards Vej 7
Copenhagen, Denmark
tilr@itu.dk

Noor Shaker
Center for Computer Games Research
IT University of Copenhagen
Rued Langgaards Vej 7
Copenhagen, Denmark
nosh@itu.dk

ABSTRACT

In this paper we introduce a constructive approach for infinite deterministic content generation for 2d grid-based world games. We present our game *Crowdbeam* and the underlying framework implemented to generate its content. Our approach relies on a generic layer-based framework that eases implementation and future extension. In the current version, the system is composed of four layers that allow real time generation of a wide range of content variations while preserving deterministic outcomes when seeded with the same parameters. We rely on a combination of methods such as agents and Cellular Automata to smooth the content and to handle soft transitions between the parts of the world. We present the framework and the methods used and we discuss the integration into the game. We finally provide a preliminary analysis of the results.

Keywords

Procedural Content Generation, Video Games, Infinite World Generation, Constructive Methods, Agent-based Methods, Cellular Automata

1. INTRODUCTION

Procedural world generation is a concept that has been used in many games to automatically create variations of game levels for players to explore. Several methods have been discussed for this purpose with varying degrees of satisfactory results. With the rise of open sandbox games however, as for instance *Minecraft* by Mojang [8] or *Terraria* by Re-Logic [10], the importance of efficient, *infinite* world generation that is *deterministic*, but also *non-repetitive* and interesting has come into focus.

During the last years, the rise of online games that can be played on the majority of consoles and devices has become apparent. While this gives many players the possibility to start a quick game session at any time or place, the challenges to create huge and persistent worlds in real-time are increasing equally. Sandbox games require a semi-infinite-sized world that does not necessarily need to be deterministic. Multiplayer games on the other hand can not be stochastic as players sharing the same session need to have the same view of the environment. Moreover, when targeting mobile devices and browsers as platforms, one will be faced with streamlined technical performance requirements that do not allow for a transformation of pre-generated game assets.

This paper proposes a system to tackle the challenges of the *online* creation of *infinite deterministic* game worlds. For this purpose, we implemented a world generator for a proof-of-concept game named *Crowdbeam*, which, by making use of a sophisticated layered system, contextual abstraction of the layers and multiple types of constructive methods, fulfills these challenges and results in a system with great potential for future extensions.

The paper is organized as follows. The next section describes the theoretical background of the research. Following up, the proposed game idea on which the concept is examined is introduced in Section 3. In Sections 4 and 5 we explain the methods and algorithms and their integration into the concept. Section 6 presents the results and put them in relation to the described goals. Finally, Section 7 gives an overview of possible future directions.

2. BACKGROUND

Procedural Content Generation (PCG) is a flourishing area of research with many exciting applications and directions. The work in this area started as an approach to overcome storage and memory limitations of early games hardware. Recently, however, PCG methods are being developed for many other reasons such as providing variations, personalized and infinite content. The field thrives with several successful implementations of PCG methods in different research and commercial areas. Hastings et al. [6] present a system for automatic generation of personalized weapons in a *Galactic Arms Race* game while in [18] the method is used to generate personalized tracks for a car racing game and in [13, 12] to personalize content generation in *Super Mario Bros.*

Different techniques have been explored to automatically generate different aspects of content [11] and some of them achieved remarkable results in commercial games. *Diablo*[3], for instance, features procedural generation for creating the maps, the type, number and placement of items and monsters. *Civilization IV* [5] allows unique gameplay experiences by generating random maps and *Minecraft* [8] is one of the recent popular indie games that features extensive use of PCG techniques to generate the whole world and its content. *Spelunky* [20] is another notable 2D platform rogue-like indie game that utilizes PCG to automatically generate variations of game levels.

The recent literature on PCG identifies PCG-based game design as

one of the promising visions for future PCG systems [11]. This subfield refers to systems that do not function without PCG which acts as the core for content creation. Very few examples related to this category can be found in previous research. Examples include *Galactic Arms Race* [6] and *Endless Web* [14]. These games however could still function without PCG.

In the system we propose in this paper, PCG plays an essential role in content generation without which the implemented game could not exist. While doing our research, we looked closely at how two of the big names in modern sandbox games, *Minecraft* and *Terraria*, have implemented their solutions. However, finding technical descriptions of the solutions used in commercial games is not easy, as most game studios use in-house proprietary solutions. There seems to be a combination of different approaches apparent in both and similar titles, but details on their methods were not made publicly available. Therefore, this paper strives to be an example for a sandbox world generation approach for developers interested in implementing similar concepts, and for researchers interested in further advancing the state-of-the-art of the use of PCG in games.

However, outside of big game development studios, a number of research proposals are available that describe general methods of implementing world generators. Many adaptations of Minecraft-like level generation have used methods such as fractals for the creation of small regions, as described by Tippetts [15], or for cave generation as presented by Cui et al. [4]. Tippetts also describes the division of chunks into regions with a certain configuration of block types, as is probably used in Minecraft [17, 16]. The work in this paper is inspired and informed by this work and builds on it to provide a full description implemented in a playable game.

3. GAME DESIGN: CROWDBEAM

The motivation of this project mainly comes from the need for a procedural world generation method for a game called *Crowdbeam*[1]. Crowdbeam is a 2D voxel-based sandbox game with soft-body physics. Its gameplay is influenced by *Terraria* and *Minecraft*, but it adds an extra layer of challenge through the physical and destructible world. The player controls a character through the world and can mine different types of blocks and place them somewhere else, similar to Terraria, but the danger of falling through unsafe ground or being buried under collapsing constructs is always apparent.
The game uses PCG methods for generating the infinite world and as it is destined to be expanded for a multiplayer version, a deterministic world generation approach is required, i.e. the game world should be generated with pseudo-random seeds (which preserve endless content), but still offer the same content to all players. This is also a common requirement in commercial games that feature PCG [7]. The generation method should also permit ease of extensions as the game is still under development and new concepts are being added over time.

4. METHODOLOGY

In what follows, we describe the two main methods implemented for world generation in our game. Details about their implementation are given in Section 5.

4.1 Chunk-Based Generation

A typical method for infinite world generation, that has been employed in a number of infinite sandbox games, is a solution that divides the world into chunks and then generates individual chunks at run-time as needed [15]. Such systems usually use some form of overarching system that keeps neighboring chunks similar and ensures smoothness of the adjacent edges so that the player will not feel a sudden change. Example approaches include applying a series of noise maps or other forms of generated dividing structures — typically using Perlin Noise, fractals, Voronoi clusters or Cellular Automata (CA) — that are usually used as a base for that particular chunk or series of chunks. Further noise maps or smoothing functions can later be applied. Objects and other features can then be added based either on noise maps or on some other forms of randomization.

The chunk system can further be implemented as multi-layered, where each chunk can be made up of other smaller chunks that can then represent features within the main chunk. For example, a chunk designed to be an underground chunk may contain other more detailed ones designed as iron ore and/or a cave chunk. In this way, arbitrary compositions of methods working on different layers becomes possible, an advantage already suggested by Togelius et al. [19].

4.2 Agent-Based Generation

Using Agent methods to generate content for game levels is generally implemented using a single or multiple agents that traverse an initially empty level area and "dig" out space based on predefined behaviors [9]. There are many different ways to implement such agents and there are just as many AI behaviors to consider. In all approaches however, the agent starts at a certain point on the map and moves either randomly or in directed ways while applying changes to the map. The behavior defined in most of these methods is typically rather unpredictable, but it can result in more organic looking regions, depending on how it has been defined.

5. IMPLEMENTATION

This section provides details about our implementation of the different methods used to generate content for our game.

5.1 Level of Detail

In order to enable multiple abstraction layers of content and context granularity, we introduced a layered system that eases world generation and future extensions. Our system constitutes of four independent layers. The basic layer is composed of the smallest element of the world, the single *block*. A collection of blocks then forms a *cluster*. Clusters of different types form *chunks* which are in turn grouped to form a *region*. Finally, the world is a collection of regions.

A block represents the material information of a single geographic entity. Clusters are a small set of blocks that share the same ground material, e.g. there can be clusters of rock, mud or emptiness. The actual appearance of the blocks in the game is driven by the combination of the properties of the region and the cluster they belong to - for instance, a cluster of trees in a desert region looks different than one in an arctic region. Structuring the game in this hierarchy allows sub layers to inherit the properties and information from the layers in the upper levels. This structure also permits easy extensions in the future by simply plugging new layers in.

In the next section, we provide more details about each layer and how they are combined to construct the world.

5.2 World Generation

[1]The current version of the game can be found at http://www.crowdbeam.net - the source code of the world generation part is available at http://github.com/teinemaa/worldgen2d

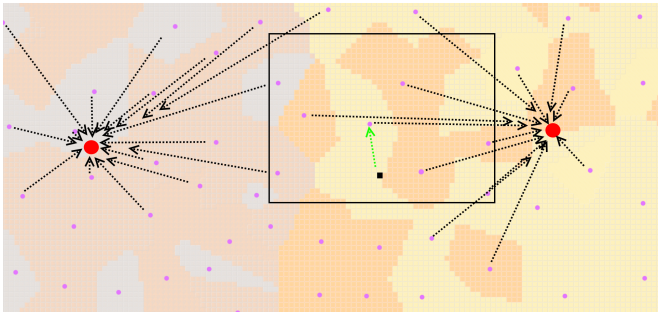


Figure 1: A screenshot from the game showing the different layers. The small black block marks the block that is requested for rendering, which initiates the chunk generation process (marked with the black rectangle). The red dots are the regions' center points from the first layer. The smaller violet points are the clusters' center points from the second layer. Clusters are assigned to regions, as visualized by the black arrows, and blocks are assigned to a cluster, as indicated by the green arrow. On top of that runs the clipping function, the CA and agents, which are not visualized.

As discussed earlier, in order to handle infinite world generation, the approach handles the creation of only a small portion of the world and extends it as needed. This requires a method to handle world division, i.e. dividing the world into regions that can be built individually and independently of the rest of the world; yet can be effectively combined to form the complete world. In our implementation, we use Voronoi diagrams as a segmentation method applied on a number of predefined layers. The boundaries of each region (with all its sub layers) are defined by the Voronoi cells.

Because the world is organized in layers, world generation can be done starting at the smallest element, the *block*, and recursively moving up in the layered hierarchy until we reach our largest element, the *regions*. More specifically, whenever a block is requested for rendering, we initiate the *chunk* generation process which defines the boundary of the area we are currently creating. In order to know the exact characteristics of each block to be rendered, we refer to the *cluster* and the *region* to which the block belongs.

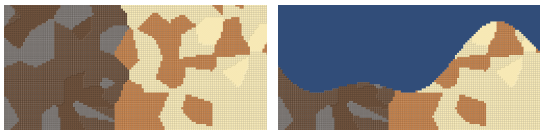
Notice that in this process, the defined layers in our hierarchy serve for two separate purposes when generating the world; while the *chunks* draw the actual layout boundaries, the *clusters* and the *regions* define invisible outlines and they are used to store and organize property information of the inherited layers. If we are to dig into more details, the world generation process can be described as in Algorithm 1 (a visual illustration can be seen in Fig. 1):

The first step ensures deterministic outcome when generating different chunks and when regenerating the same ones. A predefined size is given to a chunk and in order to find the closest cluster and region, we rely on distances to generated Voronoi points that define the centers of regions and clusters. Note that whenever we generate a chunk, we also initiate the process of generating the directly adjacent neighboring chunks to handle the transition between them and to ensure smoothness.

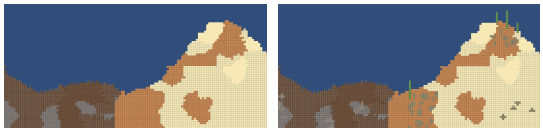
The method also integrates a number of global functions, which get executed after determining the regions and the clusters and during the generation of a chunk. These can be used for defining ground curves according to certain desired features. As an example, there is a function implemented that clips out the ground level of the world

Algorithm 1 World generation process

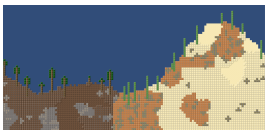
For a given block b_g to be visualised:
Seed the chunk c_i to which the block belongs with a seed and an index
Determine the boundaries of c_i
Generate the centres of the regions within c_i
Generate the centres of the clusters within c_i
for (each block b_i in c_i) **do**
  Check to which cluster l_i the block belongs and determine the type of the block accordingly
  Check to which region r_i the cluster l_i belongs and update the b_i type accordingly
end for
Repeat the above process on the directly adjacent chunks to c_i
Apply a set of clipping functions to define the ground boundaries (separation from the sky)
Apply cellular automata steps inside the chunk and across its borders
Run agents from a set of starting points, determined by the chunk's seed, inside the chunk and across the borders



(a) Initial blocks generation (b) Ground clipping



(c) Cellular automata (d) Agents working on the center and adding trees



(e) agents working across borders and adding trees

Figure 2: The procedure followed for world generation (the figure includes four chunks belonging to different regions).

with a couple of parameters, in order to make the ground more interesting and passable. There could also be other clipping functions added such as various cave structures or a water/ lava ground level similar to those in Terraria.

In the following sections, each of the remaining steps is presented in detail. A figure illustrating the highlights of the world generation can be seen in Fig. 2.

5.3 Cluster Generation

According to the procedure illustrated in Algorithm 1 above, the clusters must be identified in order to correctly assign the properties of each block. Cluster centers are defined when the chunk generation process is initialized. Voronoi diagrams are used to define the boundaries of each cluster (a collection of blocks of the same type). When assigning a type to a block, the distances to the centers of the clusters within the chunk are calculated and the block is assigned the type of the cluster with the smallest distance, as can be seen in Fig. 1.

5.4 Region Generation

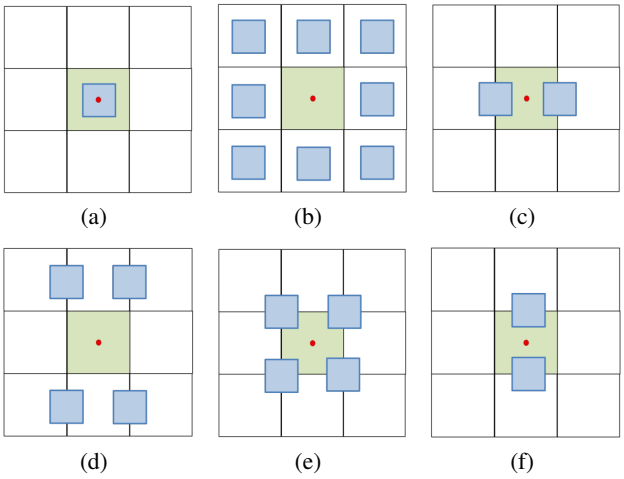


Figure 3: The procedure followed to execute the agents to smooth the the transitions between chunks. A similar concept is used for the execution of Cellular Automata.

An additional layer of Voronoi is used to create regions [2]. Each cluster simply gets assigned the closest region type (see Fig. 1 for illustration).

The possible region types are predefined and called *biomes*, and can have their own parameters, types, cluster materials and cluster probabilities. The Poisson distribution function is used to distribute the Voronoi points of the regions, which allows for a naturally random distribution of points over the whole world space [1]. Each region has assigned a range of different Cellular Automata, which can be used with different probabilities defined by the game designer, in order to create interesting variations and effects such as erosion. The process of generating the regions' Voronoi points is initiated when the chunk generation process starts. Some of the points might already be created during the creation of other neighboring chunks and the current chunk might create some extra ones.

The regions are also equipped with different kinds of agents, which are executed with user-defined probabilities and ranges. Their goal is to further refine the terrain and add high level content. While the previous generation steps mostly shape the main terrain structure, these agents spice up the terrain and can be instantiated and used in various ways, e.g. for generating trees, dungeons, lakes, clouds or just to add some random noise. In the proof of concept game version, we use two types of agents for tree generation and for creating mineral ores.

5.5 Handling Border Areas

The previous discussion explores the generation of individual pieces of the world. However when putting these pieces together, extra care needs to be taken to ensure smooth transition from one piece to another. One should also keep in mind that the outcome should be deterministic.

A chunk is depending on its neighbor chunks and in order to guarantee deterministic results, we implemented a deterministic method relying on agents and Cellular Automata. Specifically, we use parameterized agents that work on the borders and the center of the chunks, and we apply a CA beforehand to smooth the terrain. The agent-based method, as seen in Fig. 3, works as follows:

We start with the current chunk, c_i, that is being generated (Fig. 3.a).

Figure 4: Screenshot highlighting structural differences between two adjacent regions.

The inner blue square marks the area that agents are allowed to be spawned within. The agents are initialized with a maximum radius of $\frac{1}{4}$ of the chunk size to ensure that they remain inside the chunk's borders. Now in order to guarantee deterministic outcomes and smooth transform from one chunk to its neighbors, this step is performed also on the eight directly adjacent chunks to the one we are currently generating (Fig. 3.b). This is followed by spawning border agents on the left and right borders of the current chunk as shown in Fig. 3.c. A combined seed of the current and the neighbor chunks is used by the agents and the process is repeated for each of the adjacent chunks as can be seen in Fig. 3.d. After this step, diagonal border agents are spawned on the four corners of the chunk as illustrated in Fig. 3.e. Finally, the upper and lower areas are smoothed (Fig. 3.f).

The above described procedure results in smoothing all parts of a chunk separately, yet in a specific order and using combined seeds that guarantee deterministic world generation. Notice that when one of the neighbor chunks is later requested for rendering, part of the blocks belonging to that chunk will be already generated and the process continues to create only the missing part.

6. RESULTS

The implemented proof of concept shows that the combination of methods allows for a deterministic, theoretically infinite-scaled world generation, and enables enough variation and control for a commercial-scoped game.

The world generation was tested with different starting locations and screenshots of the same world location were later compared to ensure that different world generation paths would still lead to the same results. Since the results matched, it is likely that the world generation is in fact deterministic. We are considering to implement a stochastic evaluation method in the future to be able to validate the approach more thorough.

In order to analyze the outcome of our method, we also looked at different screenshots from different regions from the world. An example screenshot is shown Fig. 4. The analysis highlighted the visual advantages and disadvantages of using Voronoi for clustering.

The use of Voronoi for the generation of structures allows for a certain degree of variation, however it is not possible to have caves with completely different styles, e.g. long, tiny caves along with big hollow ones. Instead, Voronoi-based structures tend to have cluster-like shapes. Therefore, a more promising alternative for structure generation could be to rely on global functions or more powerful agents with bigger radii that permit more unpredictable results. It could also be possible to have a variable frequency of Voronoi points across the world, to allow for more contrasts. The use of a low frequency of Voronoi points would lead to rather straight lines while a high frequency will result in cluttered landscapes and rock formations.

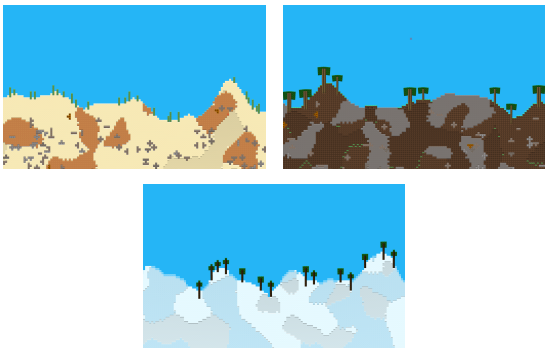


Figure 5: Snapshots taken from the same position in the game generated with different seeds.

In order to give some insight on the variations provided by our method, Fig. 5 presents three snapshots taken from the same position in the game generated using different seeds.

6.1 Computational performance

In order to show the efficiency of our approach, we report some statistics about the time required for world generation. We ran experiments on a PC of the current standard; the PC runs an Intel(R) Core(TM) i7-2630QM CPU 2.00GHz with 6GB RAM and we use Microsoft Windows 7 Professional 64-bit.

For a chunk size of 48 x 48 blocks, the system takes approximately 4 seconds on average to complete the block generation for the generation of the initial chunk, where all neighbors will be partially generated as well. This can thus be seen as the worst case. The execution time for the later generation of chunks was an average of one second, as the neighbors had already been partially generated before. The world generation was also able to keep up with the camera moving 25 blocks per second to the right. The world generation is executed using multiple threads and the next call is triggered as soon as the camera gets close to the borders of the current chunk, which enables for overhead computation in the background and eliminated a possible undesirable decay. The memory usage is constantly low and very reasonable for a standard PC, which is permitted by deleting unnecessary chunks that are far from the current player position. In practice, no performance problems with the approach have been observed on the reasonable modern PC configuration.

7. DISCUSSION AND CONCLUSIONS

This paper proposes the use of a layer-based system for infinite deterministic world generation. The layers define the level of detail of the world and are organized in a hierarchical order that permits information inheritance. Different methods are implemented to work on one or multiple layers. The methods are properly seeded and executed with predefined order to ensure deterministic outcome when necessary. The results show that infinite variations of content can indeed be generated. The proof of concept game proposed proves that a combination of simple generation methods in an intelligent way can lead to sophisticated results and interesting, varied and sometimes surprising level constructs. The implemented abstraction system and the possibility to add new custom layers make the approach easy to adjust and control for game designers. The implemented prototype is already being used by players and is a valuable inspiration for all kinds of voxel-based 2D games that require infinite or very big deterministic worlds.

There are some main areas with potential future extensions. For instance, it would be desirable to increase the number of regions, Cellular Automata and agents, that go beyond a proof of concept and enrich the game worlds with new or expanded structures. Possible ideas for additional agents could be dungeons, castles, lakes, clouds, items, or even spawn positions of NPCs. Cellular Automata could be added for different kinds of erosion, lava streams or mineral ores. Probabilities of spawning of certain agents could also be contextually adjusted within a region, e.g. flowers could have a higher probability of spawning next to trees. In addition to that, more sophisticated global functions could be added, e.g. for different types of ground lines, interesting cave formations or level-wide style choices (e.g. a candy-level, a Halloween-level, etc.).

8. ACKNOWLEDGMENTS

Many thanks to Trond Glomnes and Michal Krolikowski for the continued development of Crowdbeam.
The research was supported in part by the Danish Research Agency, Ministry of Science, Technology and Innovation; project "PlayGALe" (1337-00172).

9. REFERENCES

[1] A. H. Ang and W. H. Tang. Probability concepts in engineering. *Planning*, 1(4):112–117, 2004.
[2] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, Sept. 1991.
[3] Blizzard North, 1997. Diablo, Blizzard Entertainment, Ubisoft and Electronic Arts.
[4] J. Cui. Procedural cave generation. 2011.
[5] Firaxis Games, 2005. Civilization IV, 2K Games & Aspyr.
[6] E. J. Hastings, R. K. Guha, and K. O. Stanley. Evolving content in the galactic arms race video game. In *Proceedings of the 5th international conference on Computational Intelligence and Games*, CIG'09, pages 241–248, Piscataway, NJ, USA, 2009. IEEE Press.
[7] G. W. Lecky-Thompson. *Infinite Game Universe: Mathematical Techniques*. Charles River Media, Inc., 2001.
[8] Mojang. Minecraft, 2009.
[9] J. T. R. L. Noor Shaker, Antonios Liapis and R. Bidarra. *Constructive generation methods for dungeons and levels (DRAFT)*. 2014.
[10] Re-Logic. Terraria, 2011.
[11] N. Shaker, J. Togelius, and M. J. Nelson. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015.
[12] N. Shaker, J. Togelius, and G. N. Yannakakis. Towards automatic personalized content generation for platform games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE). AAAI Press*, 2010.
[13] N. Shaker, G. N. Yannakakis, and J. Togelius. Towards player-driven procedural content generation. In *Proceedings of the 9th conference on Computing Frontiers*, pages 237–240. ACM, 2012.
[14] G. Smith, A. Othenin-Girard, J. Whitehead, and N. Wardrip-Fruin. Pcg-based game design: creating endless web. In *Proceedings of the International Conference on the Foundations of Digital Games*, pages 188–195. ACM, 2012.
[15] J. Tippetts. More on minecraft-type world gen, 2010.
[16] J. Tippetts. 3d cube world level generation, 2011.
[17] J. Tippetts. More procedural voxel world generation, 2011.

[18] J. Togelius, R. De Nardi, and S. M. Lucas. Making racing fun through player modeling and track evolution. 2006.

[19] J. Togelius, T. Justinussen, and A. Hartzen. Compositional procedural content generation. In *Proceedings of the The Third Workshop on Procedural Content Generation in Games*, PCG'12, pages 16:1–16:4. ACM, 2012.

[20] D. Yu and A. Hull, 2009. Spelunky, Independent.