A Preliminary Study on Integrating Procedural Content Generation into Game Development Process

Pratama Wirya Atmaja[1]
Department of Informatics Engineering, Faculty of Computer Science
University of Pembangunan Nasional "Veteran" Jawa Timur Surabaya, Indonesia
pratama_wirya.fik@upnjatim.ac.id

Rizky Parlika[2]
Department of Informatics Engineering, Faculty of Computer Science
University of Pembangunan Nasional "Veteran" Jawa Timur Surabaya, Indonesia
rizkyparlika.if@upnjatim.ac.id

Sugiarto[3]
Department of Informatics Engineering, Faculty of Computer Science
University of Pembangunan Nasional "Veteran" Jawa Timur Surabaya, Indonesia
sugiarto.if@upnjatim.ac.id

***Abstract*—Digital game industry continues to grow and reap enormous profit. On the other hand, game development is still a risky and costly endeavor, and researches on reducing its risks and costs continue to be important. Procedural content generation or PCG is the state-of-the-art method to speed up and automate the production of various game contents, therefore reducing the said costs and risks. However, how to integrate the method into the lengthy process of game development is still not well understood. In this paper we present a preliminary study on the integration. For the development process, we combine MDA or Mechanics-Dynamics-Aesthetics framework with SCRUM-based Agile methodology. The PCG method for our study is for generating levels of platformer games. We study how the PCG method would be developed in pre-production and production phases with the help of two common development tools, Game Design Document (GDD) and user stories. We discuss our findings and possible directions of future researches.**

***Keywords*—MDA framework; game development; Agile methodology; procedural content generation**

I. Introduction

The upward trend of digital game industry cannot be overstated [1]. It is hard to deny the tremendous financial and cultural impact of the industry to the world, yet behind the positive outlook lies a harsh reality of companies and people who try their best to simply survive in an uncertain and risk-laden environment. Developing software has never been a smooth-sailing process for even the largest and most experienced developers, but it is doubly so when the software in question is digital games. The need for multi-disciplinary teams and the prominence of "fun factor" and other non-functional requirements in digital games prevent game development from being controlled and measured properly [2].

One contributing factor to the cost of game development is game contents. As digital games grow ever more complex, players too demand more and more contents from their favorite titles. Developing the contents manually is increasingly becoming inefficient, and this is why procedural content generation (PCG) is seen as the only way forward. With PCG, content generation and production can be automated, although the difficulty of implementing PCG increases sharply with the complexity of the contents. Level structures [3, 4, 5], object behaviors [6], object formations [7], graphical assets [8], and other contents commonly encountered by players have been generated successfully with PCG.

It is undeniable that PCG is an important tool for game developers, yet its applications in real game development projects are still far from optimal and systematic. Many developers in digital game industry still see PCG as yet another tool to add to their repertoires. We argue that this point-of-view needs to change as content production is a fundamental part of a game development process, and therefore a method that is able to dramatically change the nature of the production should be integrated more firmly into the development process. Treating PCG as "just another tool" can affect the resulting game products negatively as it will make it hard for developers to understand and measure its use [9, 10]. This is evidenced by oppositions to PCG from players who think that PCG ruin the "fun factor" of their favorite games or even dislike the very notion of "procedural contents" [11].

In this paper we present a preliminary study on how to integrate PCG into game development process. To narrow our scope, we select a specific PCG method for a specific game genre and content type, which is for generating levels of platformer games. The design of the method follows Mechanics-Dynamics-Aesthetics (MDA) framework and the development process is done under SCRUM framework of Agile methodology.

A. Agile Methodology and SCRUM

Facing the aforementioned hurdles of game development process, developers have tried to adapt by implementing less rigid development methodologies such as Agile [12]. The highly mutable and subjective requirements of game software is accommodated by Agile methodology's spirit and ethics, and the rise of small and independent game developers also increases the need for the methodology. SCRUM, Kanban, user stories, and other methods and tools have been helping Agile practitioners over the years and success stories can be heard anywhere [13].

Figure 1 shows a typical SCRUM workflow [14]. Together with stakeholders, the developer gathers the game's requirements in pre-production phase, which are commonly documented in a Game Design Document (GDD) and constitute the game's *product backlog*. The requirements can also be translated into *user stories* to help stakeholders and members of the development team understand them. The production phase is composed of short iterations—several weeks each—called *sprints* where the developer conducts not only the implementation but also the testing and design of the game. The product backlog is translated into *sprint backlog*, which contains specific development tasks to bring user stories into reality. New user stories may be added to the product backlog as the development team learns new things during the game's development process.

B. Mechanics-Dynamics-Aesthetics Framework

The Mechanics-Dynamics-Aesthetics (MDA) framework has been widely used in game design and development [15]. Under the framework, the design of a game starts from its least visible aspect, the mechanics, to its most visible, the aesthetics. The mechanics describe the underlying gameplay rules and object types, the dynamics describe how the rules and the objects come into play under specific spatial and temporal conditions, and the aesthetics describe how players will observe the dynamics with their senses.

A GDD implementing MDA framework has been proposed by Mitre-Hernandez et al. [16, 17]. The structure of the document contains five important chapters as seen in Table 1.

C. Procedural Content Generation of Game Levels

Levels have been one of the most popular content types to be generated procedurally, dating back to *Rogue* in 1980. Different game genres have levels with different characteristics, although they can be generalized to some extent. There are many procedural generation methods for levels, which fall into two basic categories: search-based [18] and constructive [19]. The first tries to find level arrangements with good quality based on certain fitness criteria, by utilizing genetic algorithm or other meta-heuristic approaches. The second, on the other hand, constructs a level one small part at a time, avoiding the need for finding and evaluating a large number of solutions from a vast search space. Other way to classify PCG methods are by looking at whether a method is performed at run-time for end-users or within development process to assist the developer in creating levels. The latter is *mixed-initiative* [20] and the end-product of the game may actually have fixed, non-procedural levels.

From a higher point of view, the generation of a level can be seen as a two-step process where constructing the level spatially is the second step. The step proceeding the spatial construction is designing the player's possible progression in the would-be level, which can be done with design tools such as graphs [21]. Figure 2 shows an example of this two-step process. All things considered, understanding the steps taken in procedural generation of levels is crucial in integrating the generation into the game's development process.

D. Platformer Games

Platformer games are simple yet addictive and their levels provide real-time navigational challenges to players. Even in the age of photorealistic 3D games, 2D platformers are still widely produced and played. A platformer level consists of several basic element types, from platforms themselves to special objects that can be triggered to affect the level somehow [22]. The spatial structure of a platformer level typically exhibits "rhythms" not unlike musical rhythms with rising and descending parts, and this characteristic has been exploited in PCG . Rhythmical parts of a platformer level can be seen as "design patterns" that can be applied to create a wide range of levels with good quality. Three examples of successful platformer games with procedurally generated levels are *Spelunky*, *Terraria*, and *Dead Cells*.

II. Gathering Requirements in Game Design Document

For the purpose of our study we use an example of a platformer game with procedurally generated levels, which will be developed under MDA and SCRUM frameworks. At the start of the pre-production phase the developer writes down the requirements of the game in a GDD. Following MDA framework, requirements for the level PCG method are twofold: mechanics and dynamics requirements. The mechanics requirements specify what the PCG method can do—what kind of levels it can generate—and how it will be evaluated based on its outputs. The dynamics requirements specify the specific characteristics of each level in the game; even if every one of them will be generated with the same PCG method, there will still be differences between them. The aesthetics requirements, on the other hand, are not specified because the PCG method will only generate the arrangements of levels and not the graphics and sounds of the objects in the levels.

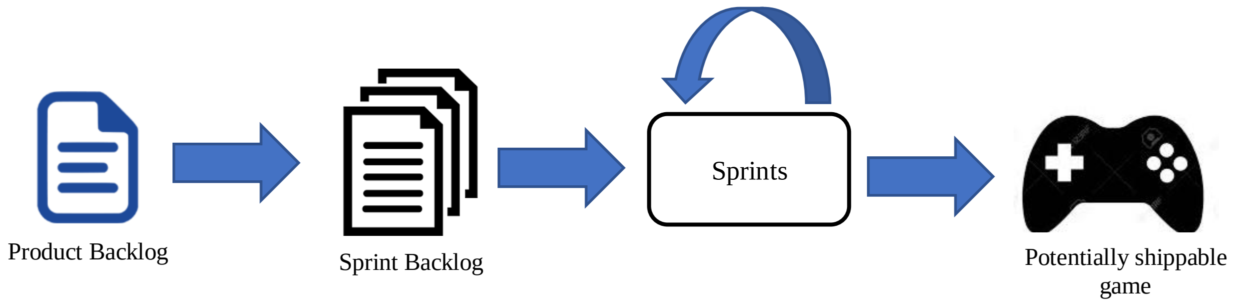

Fig. 1. A typical SCRUM workflow.

TABLE I. STRUCTURE OF AN MDA-BASED GDD

Chapter Number	**Chapter Title**	**Purpose**
1	Overview	To describe the game in general
2	Mechanics	To describe the mechanics of the game.
3	Dynamics	To describe the dynamics of the game.
4	Aesthetics	To describe the aesthetics of the game.
5	Player Experience	To describe the game's quality aspect; how the player should experience the game.

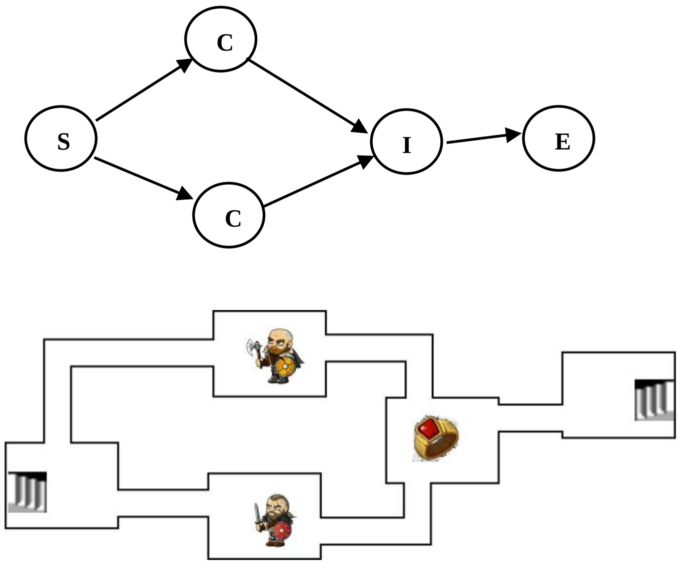

Fig. 2. An example of a two-step PCG method using mission graphs for a dungeon crawler game. The graph nodes represent actions or missions that the player may undertake, whereas the edges represent ways of moving between missions. S represents the starting point, C represents a combat, I represents an item, and E represents an exit. The graph is then converted into a game map as seen below it, where each mission occupies one room and the rooms are connected together with corridors.

A. Mechanics Requirements of Level PCG

Using the GDD format as seen in Table 1, the mechanics requirements for the example game are to be put in Chapter 2. We define that the level PCG method in the game should possess these characteristics:

1) The method generates levels at *run-time* without any interferences from players nor the developer;
2) The method should be easy to analyze and modify during the development of the game.

The PCG method will perform two-step level generations where the first step will be constructing a mission graph. The graph specification becomes another part of the mechanics requirements. In the context of a platformer game, the mission graph nodes may represent the objects in the game. We base the example platformer game on classic style games such as Super Mario Bros. The player's character may walk, run, jump, and shoot a weapon. Touching enemies hurts the player's character, he loses a life when his health is zero and the game is over when the player's lives has been depleted.

We define four object types of the game to be generated procedurally: platforms, enemies, collectible items, and obstacles. The object specification can be seen in Table 2 and the nodes of the missions graphs will represent the object types.

B. Dynamics Requirements of Level PCG

Following the convention of Super Mario Bros, the example game will be composed of "worlds", each contains a number of levels. Each world has a unique theme which dictates the look and feel of its levels, which in turn influences the level PCG. The example game contains the following three worlds:

1) The Grassland, where the player starts and everything is relatively simple and mild;
2) The Jungle, where the levels are more intricate and confusing and enemies are more difficult;
3) The Mountain, where the levels are more vertically-inclined and falling down is much more dangerous.

The more detailed specification can be seen in Table 3. The specification becomes the dynamics requirements that are written in chapter 3 of the GDD.

C. Quality Requirements of Level PCG

With the complex nature of a PCG method, it is important that the development team states, from the beginning of the development process, how the quality of the level PCG will be measured. For the example platformer game, two characteristics influence the general quality of the game's level PCG:

1) The run-time level generation should be fast enough that players would not notice it;
2) The generated levels should always be playable (players should never get stuck in them), properly challenging, diverse, and look and feel as natural as possible (like hand-crafted levels).

The developer may also use more detailed quality metrics of procedurally generated platformer levels such as leniency, linearity, and pattern density [23]. These quality requirements should be put in Chapter 5 in the GDD.

III. Creating User Stories and Sprint Backlog

For a two-step level generation, two modules are necessary: *mission graph generator* and *spatial level generator*. To generate a mission graph for a level of a certain world, the generator first takes as an input mission pattern data of said world. The patterns reflect the dynamics requirements of the world. The resulting mission graph then becomes an input for the spatial level generator, which will translate the graph into an actual, playable level.

The overall structure of the PCG system can be seen in Figure 3. The main parts of the system are *level generator module*, which performs the level generation itself, and *level generation analyzer*, which allows the developer to analyze and evaluate the level generation. With the complex nature of a PCG method, the analyzer tools are important assets to the development team.

A. User Stories and Sprint Backlog of Level PCG System

To write user stories of the level PCG system, the developer needs to understand not only the structure but also the users and the module dependencies of the system. Following INVEST principle [12], Agile developers tend to make user stories as independent to each other as possible. Fortunately, a typical level PCG system can be treated separately from other parts of the game development process. The level PCG system does not need to wait for actual audio and visual assets of game objects because any placeholder assets will do. Figure 4 shows the dependencies between modules in the PCG system. We can see that only one module, the analysis report creator tool, needs to wait for the completion of another module before its development may start. This is because the report creator requires analyzer tool to feed it analysis data. Every other module depends not on another module but two encoding conventions: mission graph and spatial level symbols. These define how the mission graphs and spatial level constructions are encoded for cross-module uses. The mission graph nodes can be encoded as ASCII characters whereas edges between nodes can be represented by two-dimensional arrays. As defining the encodings may take very short time, it does not have to have its own user story.

Another concern in an Agile development is choosing the correct users for user stories, because the stories are supposed to state real values of the products being developed. Table 4 shows the intended users of the modules of the level PCG system. Of note are the non-human users of the mission graph generator, the spatial level generator, and world mission patterns. The two analysis modules, on the other hand, are meant to be used by the person in charge of the level PCG method quality, who may be the game's lead designer.

The complete user stories, along with their respective sprint backlog items, for developing the example game's level PCG system can be seen in Figure 5 to 9.

TABLE II. Object Specification of the Developed Game

Type	Sub-Type	Characteristic
Platform	Solid	Nothing can pass through it.
	Jump-through	The player's character may jump or fall through it.
Enemy	Ground	Walks on platforms, is affected by gravity.
	Flying	Is not affected by gravity.
	Shooting	Shoots projectiles but does not move at all.
Collectible item	Coin	Increases player's score.
	Bullet	Refills player's ammunition.
	Health item	Add one health point to player's character
	Life	Add one life to player's character.
Obstacle	Bottomless pit	Is placed at the bottom of the level, kills the player's character instantly.
	Spike	Damages the player's character.

TABLE III. Worlds Specification

World	Theme	Characteristic
1	Grassland	Bottomless pits and traps are rare.
		Platform arrangements are simple and paths to level exits are easy to find.
		Walking enemies are numerous and other enemy types are rare.
2	Jungle	Dead ends and traps are more numerous
		Items are more often hidden and/or hard to reach.
		Shooting enemies are more frequent.
3	Mountain	Platform arrangements are much more vertical.
		Bottomless pits are much more numerous and wide.
		Flying enemies are much more numerous.

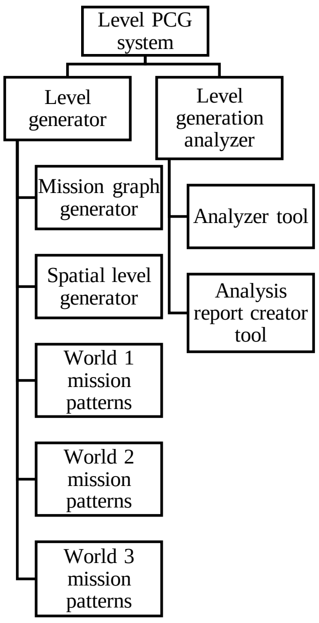

Fig. 3. Structure of the level PCG system.

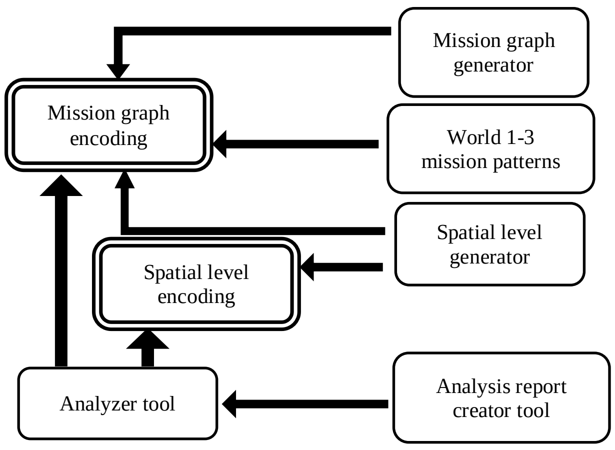

Fig. 4. Module dependencies in the level PCG system.

TABLE IV. USERS OF THE LEVEL PCG SYSTEM'S MODULES

Module	**User**
World 1-3 mission patterns	Mission graph generator
Mission graph generator	Spatial level generator
Spatial level generator	⚙
	Game engine
Analyzer tool	● Lead designer
Analysis report creator tool	

As a spatial level generator, I want to read, as my inputs, mission graphs so I can generate levels based on them.

Conditions of Satisfaction:

- ➢ The mission graphs are always playable.
- ➢ The mission graphs are always properly challenging.
- ➢ The mission graphs always feel as natural as hand-crafted ones.
- ➢ The mission graphs are diverse enough.
- ➢ The mission graph generation is fast enough to be applied at run-time.

Programmer: Implement an algorithm for generation of mission graphs which takes the appropriate world's mission patterns as inputs.

Programmer: Test and tune the mission graph algorithm.

Fig. 5. User story and sprint backlog item of mission graph generator.

As the game engine, I want to present procedurally generated levels to players.

Conditions of Satisfaction:

- ➢ The generated levels are accurate with regards to the mission graphs they are based upon.
- ➢ Level generation is fast enough to be applied at run-time.

Programmer: Implement algorithm for generation of levels based on mission graphs.

Programmer: Test and tune the generation algorithm.

Fig. 6. User story and sprint backlog item of spatial level generator.

As a mission graph generator, I want to read World 1/2/3 mission patterns to guide me in generating mission graphs for the world.

Conditions of Satisfaction:

- ➢ The mission patterns are diverse enough.
- ➢ The mission patterns have good “fun factor”.
- ➢ The mission patterns are properly challenging.

Designer: Design the world’s mission patterns as graphs.

Programmer: Translate the missions pattern graphs into machine-readable data.

Fig. 7. User story and sprint backlog item of mission patterns of worlds.

As the lead designer of the game, I want to analyze and evaluate the level PCG with a GUI-based analyzer tool.

Conditions of Satisfaction:

- ➢ The tool’s GUI must be easy to use and understand.
- ➢ The tool must be able to save analysis and evaluation data.
- ➢ The tool must allow the level generation to be tested on [insert metric].

Programmer: Code the GUI of the analyzer tool.

Programmer: Code the functions for testing [insert metric].

Fig. 8. User story and sprint backlog item of analyzer tool. The test metrics in CoS and sprint backlog are taken from the quality metrics in chapter 5 of the GDD.

As the lead designer of the game, I want to read reports on the characteristics and performance of the level PCG.

Conditions of Satisfaction:

- ➢ The report softcopies must be in PDF and spreadsheet formats.
- ➢ The reports must be easy to retrieve.
- ➢ The reports must be printable.
- ➢ The printed reports must be easy to read.

Programmer: Code a tool for translating analysis and evaluation data of level PCG into PDF and spreadsheet formats and printing the data.

Fig. 9. User story and sprint backlog item of analysis report creator tool.

B. Testing and Refining the Level PCG System

The level generator modules need to be tested and Figure 10 shows a user story and sprint backlog item for the test. The lead designer performs analysis and evaluation based on the test results from testers and also by using the analyzer tool module. Because the test is ultimately aimed at improving the level PCG as a whole, the “user” in the story is one who will benefit from it, which in this case is a would-be player of the game.

With each subsequent sprint, the development team understands more and more of how to refine the level PCG until it is ready for release. New level PCG user stories may be created for the next sprints based on the results of analysis and evaluation. For example, if the player’s missions are not diverse enough, the actions to address the problem may be:

1) Increasing the quality and/or quantity of mission patterns of the related worlds;
2) Modifying the mission graph generator so that it produces more diverse missions;
3) Adding a new “spatial level pattern” modules which work on the same principle as the mission patterns but for guiding the spatial level generator;
4) Other actions.

The user story and sprint backlog item for the first action can be seen in Figure 11.

As a player, I want to play procedurally generated levels of good quality.

Conditions of Satisfaction:

- ➢ The generated levels are always playable.
- ➢ The generated levels are always properly challenging.
- ➢ The generated levels always feel as natural as hand-crafted ones.
- ➢ The generated levels are diverse enough.
- ➢ The level generation is fast enough.

Lead designer: Write a test plan for level generator.

Tester: Test the level generator.

Lead designer: Analyze and evaluate test results.

Fig. 10. User story and sprint backlog item of testing the level PCG.

IV. Conclusions

We have presented a preliminary study on how to integrate procedural level generation method into a game development process so that the method may be managed well during the process. We have used a level PCG method for a platformer game as an example and we have studied how the method's mechanics and dynamics requirements may be gathered and documented in GDD and turned into proper user stories in accordance with common user story principles.

As the mission graph generator, I want to read, as my inputs, World 1/2/3 mission patterns.

Conditions of Satisfaction:

- ➢ The mission patterns are twice as diverse as before.
- ➢ The mission patterns have good "fun factor".
- ➢ The mission patterns are properly challenging.

Designer: Design more mission patterns as graphs.

Programmer: Translate the missions pattern graphs into machine-readable data.

Fig. 11. User story and sprint backlog item of refining the mission patterns.

Future researches may explore deeper into the topic by studying a more complete development process of a game with PCG method, under SCRUM or other development methodologies and frameworks such as Extreme Programming and Waterfall. Other kinds of procedural contents such as storylines and game rules are also interesting topics, as well as mixed-initiative PCG methods.

References

[1] J. Batchelor, "Games industry generated $108.4bn in revenues in 2017," Gamer Network, 31 January 2018. [Online]. Available: https://www.gamesindustry.biz/articles/2018-01-31-games-industry-generated-usd108-4bn-in-revenues-in-2017. [Accessed 2 September 2018].

[2] D. Callele and E. Neufeld, "Requirements Engineering and Creative Process in the Video Game Industry," in *Proceedings of the 2005 13th IEEE International Conference on Requirements Engineering*, 2005.

[3] A. Liapis, "Multi-segment Evolution of Dungeon Game Levels," in *Proceedings of the Genetic and Evolutionary Computation Conference*, Berlin, Germany, 2017.

[4] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network," in *GECCO '18 Proceedings of the Genetic and Evolutionary Computation Conference*, Kyoto, Japan, 2018.

[5] M. Stephenson and J. Renz, "Procedural generation of complex stable structures for angry birds levels," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 2016.

[6] K. Siu, E. Butler and A. Zook, "A Programming Model for Boss Encounters in 2D Action Games," in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[7] A. Khalifa, S. Lee, A. Nealen and J. Togelius, "Talakat: bullet hell generation through constrained map-elites," in *Proceedings of the Genetic and Evolutionary Computation Conference*, Kyoto, Japan, 2018.

[8] A. Liapis, "Exploring the Visual Styles of Arcade Game Assets," in *Proceedings of Evolutionary and Biologically Inspired Music, Sound, Art and Design (EvoMusArt)*, 2016.

[9] D. Aversa, "A call for a more meaningful Procedural Content Generation," 27 November 2017. [Online]. Available: https://www.gamasutra.com/blogs/DavideAversa/20171127/310426/A_call_for_a_more_meaningful_Procedural_Content_Generation.php. [Accessed 30 July 2019].

[10] A. Bradley, "Devs weigh in on the best ways to use (but not abuse) procedural generation," 12 March 2018. [Online]. Available: https://www.gamasutra.com/view/news/315400/Devs_weigh_in_on_the_best_ways_to_use_but_not_abuse_procedural_generation.php. [Accessed 30 July 2019].

[11] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," in *2015 International Conference on Affective Computing and Intelligent Interaction*, 2015.

[12] C. Keith, Agile Game Development with Scrum, Addison-Wesley, 2010.

[13] P. Serrador and J. K. Pinto, "Does Agile work? — A quantitative analysis of agile project success," *International Journal of Project Management,* vol. 33, no. 5, pp. 1040-1051, July 2015.

[14] G. S. Matharu, A. Mishra, H. Singh and P. Upadhyay, "Empirical Study of Agile Software Development Methodologies: A Comparative Analysis," *ACM SIGSOFT Software Engineering Notes,* vol. 40, no. 1, pp. 1-6, January 2015.

[15] R. Hunicke, M. Leblanc and R. Zubek, "MDA: A formal approach to game design and game research," in *Workshop on Challenges in Game AI*, 2004.

[16] M. G. Salazar, H. A. Mitre, C. L. Olalde and J. L. G. Sánchez, "Proposal of Game Design Document from software engineering requirements perspective," in *2012 17th International Conference on Computer Games (CGAMES)*, 2012.

[17] H. A. Mitre-Hernandez, C. Lara-Alvarez, M. Gonzalez-Salazar and D. Martin, "Decreasing Rework in Video Games Development from a Software Engineering Perspective," in *4th International Conference on Software Proceess Improvement*, 2015.

[18] J. Togelius and N. Shaker, "The search-based approach," in *Procedural Content Generation in Games*, Springer International Publishing, 2016, pp. 17-30.

[19] N. Shaker, J. Togelius, A. Liapis, R. Lopes and R. Bidarra, "Constructive generation methods for dungeons and levels," in *Procedural Content Generation in Games*, 1st ed., Springer International Publishing, 2016, pp. 31-55.

[20] A. Liapis, G. Smith and N. Shaker, "Mixed-initiative content creation," in *Procedural Content Generation in Games*, Springer International Publishing, 2016, pp. 195-214.

[21] D. Karavolos, A. Liapis and G. N. Yannakakis, "Evolving Missions to Create Game Spaces," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Santorini, Greece, 2016.

[22] G. Smith, M. Cha and J. Whitehead, "A framework for analysis of 2D platformer levels," in *Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, Los Angeles, 2008.

[23] J. R. H. Mariño, W. M. P. Reis and L. H. S. Lelis, "An Empirical Evaluation of Evaluation Metrics of Procedurally Generated Mario Levels," in *Proceedings, The Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2015.