Procedural Content Generation for Computer Games

Thomas Smith
Electronics and Computer Science
University of Southampton
taes1g09@ecs.soton.ac.uk

ABSTRACT
Modern computer games make use of a wide variety of procedural content generation (PCG) techniques that serve a number of purposes during both the development and execution of a game. This paper provides a high-level classification of a range of techniques used for different types of content, and attempts to discern common aspects and transferable approaches that can help to promote a more unified, standard approach to content generation for games.

Categories and Subject Descriptors
K.8.0 [**Personal Computing**]: General—*Games*

General Terms
Algorithms, Design, Standardization

Keywords
Procedural generation, game development

1. INTRODUCTION
Though procedural content generation (PCG) techniques have been used since some of the earliest computer games [2], they are now becoming increasing relevant as AAA games become larger and more detailed and indie games teams become smaller. PCG techniques can easily amplify production efforts by automating asset generation or augmenting manual content production, and they can be a powerful tool to allow developers to create and populate varied and believable playspaces. They can also play a role in customising each player's experience to improve engagement and entertainment.

However, the wide range of possible applications for PCG has lead to an extraordinary variety of approaches from both academia and industry. Despite the extensive research undertake over three decades, there are still no general purpose procedural game content generators, or even standard approaches to common requirements. Typically, individual procedural content generators are bespoke approaches used to provide a specific type of content for a particular application, though as this paper shows there are a number of particular algorithms often used as starting-points for these solutions.

1.1 What is PCG?
One of the problems facing researchers into PCG is the difficulty in defining the field itself. Hendrikx et al. describe it as "the application of computers to generate game content, distinguish interesting instances among the ones generated, and select entertaining instances on behalf of the players." [11], however this definition does not cover simple unevaluated or human-evaluated content such as deterministic methods, or tools used by artists during development, and does not suggest how entertainment may be evaluated. Another definition is provided by Togelius et al.: "[PCG is] the algorithmical creation of game content with limited or indirect user input." [34]. They also note that this definition does not specify the presence or absence of adaptivity or randomness, as PCG methods might be both, either or none. This definition hinges on what is understood by the phrase "game content", as many modern approaches to PCG could be said to generate more general game 'experiences' than the traditionally-understood 'content'.

1.2 Online vs. Offline
One of the most easily available classifiers of PCG approaches is whether the content generation is performed *offline*, during the development of the game or as it loads, or *online* as a result of player actions [36]. Originally, PCG algorithms ran offline due to the processing load they incurred. If used during development, the results can be baked into the game's data and shipped - this saves artists' and designers' labour during development, while still allowing fine-tuned control over the final output. Alternatively, content could be generated as a game or level loaded - meaning that the result could vary between different players and different playthroughs, and reducing the storage needed for game content. However, progress has led to faster algorithms that are capable of running during gameplay and generating new content on-the-fly; for example to generate more content whenever the boundaries of existing content are approached. The development of online PCG is more demanding than offline versions, as the available processing resources are more constrained, and algorithms must have a predictable runtime, however it brings advantages in the form of the ability for the PCG to adapt to varying game states.

1.3 Methodology

Given the broad range of procedural content techniques, there are a number of different categorisations possible. Hendrikx et al. provide a taxonomy that describes content in terms of how it is formed - a heirarchy is presented in which upper classes may be built using elements from lower ones (Table 1). This review organises approaches first by the primary user of each method - Artists, Designers and finally Users. This leads to a natural progression from primarily-offline approaches to more recent online algorithms. Each section details a number of the main purposes that PCG is used for within that category, along with examples and academic literature in that area. Since in some areas there are an abundance of similar methods references to surveys have been preferred where possible, with papers on specific techniques occasionally used if they also provide a broader overview of their area, or illustrate an uncommon approach.

Content class	Content type examples
Derived Content	News and Broadcasts, Leaderboards
Game Design	System Design, World Design
Game Scenarios	Puzzles, Storyboards, Story, Levels
Game Systems	Ecosystems, Road Networks, Urban Environments, Entity Behaviour
Game Space	Indoor/Outdoor Maps, Bodies of Water
Game Bits	Textures, Sound, Vegetation, Buildings, Behaviour, Fire, Water, Stone & Clouds

Table 1: Taxonomy provided by Hendrikx et al. [11]

2. ARTISTS

Artists have benefited greatly from the development of PCG techniques, as they can be used work more efficiently by automating generation of many content types, and can produce a wider range of outputs by automating the creation of variants. A number of techniques have arisen to generate realistic non-repeating textures, assemble a range of varied models from component parts, intelligently animate character skeletons, produce realistic and varied visual effects, and even supply cue-dependent responsive music. Many of these methods are not unique to the field of computer games, and have been adopted to produce computer-generated graphics for all kinds of media, from animated movies to photo-realistic CGI backgrounds in advertisements and print media. Part of the reason for these techniques' popularity in other fields is the fact that they are about automating the construction of essentially static resources - all either run offline during development, or have no effect on gameplay.

2.1 Content types

Artists make use of PCG to streamline the workload involved in producing a wide range of content types, and common approaches have emerged for some of the most typical use cases, as detailed below:

2.1.1 Textures

Some of the most common approaches for procedural texture construction are pseudo-random number generators such as Perlin noise, algorithmic approaches for specific repeating effects, and image filtering for pattern-based textures. Games in particular have a long history of using Perlin noise and similar effects for procedural material textures. Originally developed by Ken Perlin for the motion picture industry, it provides natural appearing textures via a deterministic process. With careful tuning of parameters, it may be used for a range of applications, from clouds to stone to wood to marble [23]. In contrast, a number of other algorithmic approaches have been developed for generating repeating textures such as the windows on high-rise tower blocks, segments of steel plating or stretches of road. Some of these have been adapted to provide non-repeating high-resolution texturing for rolling natural landscapes and backdrops via image filtering [16]. While procedural texture generation is most usually undertaken offline for artists during development, online algorithms exist that allow comparatively unskilled players to automatically generate suitable textures with limited interaction [9].

2.1.2 Models

Creating a modular 'kit' of compatible reusable model components is a common technique to reduce artist workloads and increase available variety, however with the addition of procedural arrangement algorithms large parts of the process can be automated and hundreds of unique 'individuals' created with comparatively little effort. Such approaches often define generative grammars such as L-systems to ensure individual validity, and include refinement passes to improve the quality of generated output [7]. One popular example of these systems is Speedtree - a commercial middleware product capable of generating many varieties of realistic foliage, from single bushes to entire forests [8].

2.1.3 Animation

The production of skeletal animations for modern 3D games is a time-consuming process – modern games have thousands of animations, and often use PCG techniques in order to automate parts of the work involved in ensuring characters behave realistically. Given known finish and start states for key animations provided by hand or motion capture, a combination of interpolation and physics-aware PCG methods can be used to generate all of the transitions between pairs of animation states [5]. In addition to the offline process, some games also use an online PCG system that intelligently blends animations and ragdoll behaviours in order provide entities with fluid and realistic reactions to unexpected situations. One of the best known such systems is Euphoria, an animation engine that uses a skeletomuscular simulation to generate realistic animation of characters at runtime [21]. A number of approaches also use PCG to develop animations for contexts and conditions that could not have been known at design time, typically in cases where there is a degree of user-driven content generation [12, 24].

2.1.4 Effects

In contrast to the typically-offline approaches presented so far, many PCG techniques exist that are designed to display interseting visual effects at runtime. The artist's role when using these systems will be to tune parameters of the PCG system in order to produce the desired effect, such as fire, water, smoke or clouds. The use of PCG to generate and tune such effects provides believable variety, and can give an additional degree of control in comparison to pure particle systems – which in themselves are a form of stochastic procedural modelling [27]. By specifying the generation of such

effects at runtime, it is possible to create effects that respond dynamically to their contexts, or evolve based upon external factors [10]. Another class of effects possible via PCG is the procedural manipulation of the scene during rendering in order to create graphical styles that are radically different from traditional photo-realistic techniques – such as the tufted truffala trees of Dr. Seuss [15].

2.1.5 Music

As with other content types discussed thus far, the application of PCG techniques to existing approaches to producing music for videogames allows a vast increase in the variety of content available for the same initial investment. Music in games is often cued to player actions, however storage available for individual themes is finite, and so melodies may become repetitive if the same track is triggered many times throughout the course of a game [6]. One approach to solving the occurrences of repetitive music is the use of online transformational algorithms that are able to restructure tracks or alter overall pitch or tempo at runtime, often in response to specific aspects of the context in which the track is played. Another more complex technique involves building bespoke compositions for individual situations from a selection of precomposed components. By providing a generative grammar and a library of smaller tune fragments tagged to fit specific themes, sound designers are able to produce far greater expressive variety and avoid the issues caused by limited musical palette. This approach also lends itself to greater integration between the musical soundscape and the players actions, as the music is able to reflect several aspects of the context simultaneously [25].

2.2 Approaches

Though there are a wide range of techniques, including many unique cases that defy categorisation, it can be seen that the PCG systems used by artists fall broadly into two principal classes. On the one hand are the approaches that are purely algorithmic, as with the Perlin-generated textures and the particle-based effects. These achieve efficiencies in both workflow and storage requirements by specifying a reusable generation system and then only the sets of parameters needed to generate the specific desired content. Contrast to these the approaches that begin with a kernel of hand-generated components, and procedurally assemble these modules and refine the output. Examples of this class are the pattern-based textures, grammar-based model constructors and blended music. These approaches have the effect of magnifying the small pool of original content into a range of varied outputs. Despite the differences in implementation, the common underlying intention of the two approaches leads to a number of similarities between them. None of the methods described have any automatic evaluation component, as it is assumed that human evaluation by the artists using the techniques will always be possible, and have superior results. The majority of them are designed to be used offline during development – where this is not the case, they are either tuned during development and have no impact on gameplay, as is the case with procedural effects and music, or where they are designed for user interaction they are typically made as foolproof as possible, and have minimal effect on gameplay.

2.3 Benefits

Two of the main benefits associated with artists' use of PCG techniques are the reduction in labour required to make a large variety of variations on a them, and the reduced storage requirement to represent this variance. By replacing hundreds of textures, models or animations with a set of components and a system for assembling them into content, the size of necessary assets can be dramatically reduced. '.kkrieger', a demoscene FPS game, uses entirely procedurally generated assets and requires under 100KB of storage (.theprodukkt, 2004). This is becoming increasingly relevant as games become larger and the status quo shifts from physical media to digital distribution - smaller games means faster delivery and less bandwidth costs.

2.4 Future Work

Though PCG is currently effectively used by artists both in the games industry and other digital media fields, there are a number of known issues and other ways in which existing approaches may be refined or improved upon. Amongst the algorithmic approaches, it can be difficult for artists to correct a generated texture that is almost but not quite fit for purpose. Exposed algorithm parameters do not usually provide the fine degree of control that might be necessary, and while manual correction of the texture is always possible during development, such edits would be lost if the texture is regenerated or needs to be generated during runtime. Another issue for some content types is the challenge of producing interesting, unique content. For many content types such as textures and foliage, a range of slightly varied repetitions on a theme are precisely what is needed. However for in-game audio the goal is to combine the available fragments in fresh ways to avoid repetition – the difficulties in doing so effectively are the main reason transformational approaches are currently preferred [6]. Finally, a opportunity for the field in general is the standardisation of approaches to generative grammar PCG systems. Though SpeedTree provides a commercially-available middleware solution, it is restricted to the generation of foliage only, and other more typical solutions in this area are bespoke approaches – highly dependant upon the nature of the problem being solved, the flavour of generative grammar chosen, and the domain-specific grammar itself.

3. DESIGNERS

In contrast to the procedurally generated content developed by artists, which typically has no direct effect on gameplay, the PCG techniques used by designers are generally oriented towards producing content that will have direct in-game effects during runtime. Though PCG techniques can be used to generate swathes of content in a more efficient manner than was previously possible, it also brings other, more valuable benefits to designers. The addition of a stochastic component to most approaches means that output may vary dramatically between one execution and the next, providing greater replayability. However, this comes at the cost of being unable to fully evaluate the output of the generator during development. Unless the system is simple enough that its entire expressive range is known to be usable, there is a risk of *catastrophic failure* [36]. To combat this, online PCG approaches that can impact gameplay typically follow a 'generate-and-test' pattern, whereby each generator includes an evaluator for valid content, and output that fails evaluation is rejected and regenerated.

3.1 Content Types

Designers make use of PCG to provide varied and interesting play spaces in a multitude of ways. Sample PCG approaches for some of the most common aspects of play spaces are detailed below:

3.1.1 Enclosed Environments

Many games incorporate some manner of enclosed environments where the gameplay is highly influenced by the topology of the playspace – these could be anything from caves to pinball tables to temples to racetracks. Whether 2D or 3D, it is generally necessary that the environment fulfils a certain set of minimum constraints so that it is fit for purpose – often including the requirement that the final output contain at least one entrance and exit, and that the space between them is ultimately traversable. The specific method used to generate an enclosed environment and then evaluate the output is typically highly domain dependent, though some general approaches do exist. The component-assembly method described in section 2.2 is a common starting-point, often combined with prior route setup [31]. Variants exist that offer 'mixed-initiative' development – the ability for human designers to intervene and modify the developing level at any point without disrupting the PCG algorithm [20]. Another approach suitable to more natural-looking enclosed environments is the use of cellular automata to erode randomly-seeded maps in an iterative fashion, generating extensive, tunable cave systems [14]. For simpler applications, A* search is a basic traversability evaluator, though domain knowledge may allow more efficient approaches in many cases.

3.1.2 Open Environments

Open environments differ from enclosed environments primarily in that they constrain gameplay to a lesser degree, and in turn are less constrained by it. Open environments are generally considerably more sparse and rarely unsatisfiable in any sense – rather, in this context PCG techniques are primarily used to provide believable variety for the form and content of the playspace, and must be of sufficient quality to do so. A wide range of techniques exist to generate many kinds of terrain, and existing noise and network creation techniques may be used to specify land cover features such as woodland, roads and rivers [26, 23]. A number of similar techniques have been developed to populate urban environments with realistic transport infrastructure and buildings [3]. Again, approaches have been developed that allow human designers to influence the generation of content in support of particular goals, either through the introduction of *semantic constraints* that express localised intent [28], or by introducing broader *objectives* that specify purpose for the generated environment [35].

3.1.3 Entity Behaviours

In addition to the generation of more concrete content types, PCG techniques may be used to manipulate the behaviours of entities within the game. This can be done in real time, in order to progressively modify a character's responses to the player, as in the interactive drama Façade [19], or it may be integrated into entity development in order to provide enemy and ally characters with a range of believable fighting styles, as in the FPS Killzone [33]. Some modern games also procedurally generate the behaviour and appearance of non-character entities, particularly weapons. Both Borderlands (Gearbox Software, 2009) and Galactic Arms Race (Evolutionary Games, 2010) make use of runtime PCG techniques in order to provide an almost limitless variety of weapons with a range of effects, potencies and appearances.

3.2 Approaches

There are a wide range of approaches to PCG used by designers for an equally wide range of purposes – including many highly domain-specific solutions not covered here. However, unless the approach is a simple *constructive* PCG technique that is guaranteed never to produce broken content, all of these approaches incorporate some kind of evaluation method to ensure that each piece of content it produces is viable in some sense. Some of the more common approaches include the use of cellular automata, generative grammars for top-down planning or the component-assembly model seen in section 2.2, and the use of evolutionary algorithms for content generation. However, rather than specific algorithmic approaches, two main themes arise from many of the techniques studied.

3.2.1 Search-Based PCG

An extension of the generate-and-test approach, search-based PCG evaluates candidate outputs according to some domain-specific fitness function, and assigns each some appropriate fitness score. The generation of further candidate content is influenced by the fitness of existing content, with the aim of ensuring each iteration has increasing fitness. With a carefully chosen fitness function this means not only that all output from a search-based PCG system will be valid (traversable / interesting), but also that any additional time available may be spent on improving the quality of generated content [36].

3.2.2 Mixed-Initiative Creation

Another common feature of some of the PCG approaches surveyed is the provision for human designers to directly influence the generation of content and therefore the ultimate output of the system. This is known as mixed-initiative creation as it allows the provision of hand-generated set pieces [30], level features [20] or constraints [28] by a human developer, and then uses the PCG system to fill in the blanks and weave the provided parts into a coherent whole. These systems are also capable of allowing live editing after generation, and warning the developer if the resulting level fails any of a number of satisfiability constraints.

3.3 Benefits

Speed up work - large, believable diverse playspaces

3.3.1 Content scale

Use of PCG allows designers to populate large game spaces with a high level of detail and variety. Middleware packages such as Speedtree[8] allow entire forests to be generated and customised, and similar approaches exist for generating believable cities and landscapes. "A Survey of Procedural Terrain Generation Techniques using Evolutionary Algorithms" [26] A survey of procedural content generation techniques suitable to game development [3]

3.3.2 Replayability

The use of <semi-random> techniques means that playspaces generated using PCG may be different for each player, and for each player's playthrough of a game. The variety ensures that content is fresh each time, and minimises the effects of repetition - players are not able to memorise the precise route through each dungeon and locations of treasures, and so there is a sense of discovery and exploration each time.

3.3.3 Challenge

On of the main factors that affects players' enjoyment of games is their ability to remain in a state of 'flow' - the sensation that their abilities are matched to the challenge provided by the game [4]. In order to cater to the wide range in ability of players that each game may attract, many games have previously offered the option of customising the challenge experienced to one of a number of pre-set 'difficulty levels'. <Read through 3yp for citations on difficulties with self-assessment and mutability>. Rather than <set> the degree of challenge at design time, Lopes et al. show that it is possible to adaptively generate or alter aspects of the game [18] in order to match the observed ability of the player [17].

4. FUTURE WORK

Improve automated critics [11]

5. USERS

close the gap between designer and player - allow modification based on individual information about the player.

[32]

5.0.4 Experience

Valve's AI Director [1] Bethesda's Radiant Storytelling [22]

5.0.5 Agency

Typically, no direct player control over adaptive generation [13] however, in some games that make use of procedural generation it can be a benefit to give the player some degree of direct control over the generation process - for example a recent addition to the GAR was the ''Weapons Lab', a portion of the game where players may spend in-game resources to customise their procedurally-generated weapons. (cite GAR's weapons lab) Radiant Story? GAR's weapons Choose ladders in Rathenn [29] Inside a star-filled sky citation in [29]

5.1 Benefits

Users get a more engaging experience that can be tailored specifically to their preferences and abilities

5.1.1 Future work

6. CONCLUSION

We can see that a variety of <stakeholders> benefit from the improvement of procedural content techniques, and that there are a wide range of existing techniques used for a plethora of different reasons.

7. REFERENCES

[1] M. Booth. The AI systems of Left 4 Dead. In *Keynote, Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE '09)*, Stanford, CA, October 14 - 16, 2009.

[2] D. Braben and I. Bell. *Elite*. (Armstrad CPC), Acornsoft, 1984.

[3] D. M. D. Carli, F. Bevilacqua, C. T. Pozzer, and M. C. dOrnellas. A survey of procedural content generation techniques suitable to game development. In *Games and Digital Entertainment (SBGAMES), 2011 Brazilian Symposium on*, pages 26–35. IEEE, 2011.

[4] J. Chen. Flow in games (and everything else). *Communications of the ACM*, 50(4):31–34, April 2007.

[5] S. Clavet. Procedurally Animating Assassin's Creed III. In *Presentation, Vancouver ACM SIGGRAPH*, Vancouver, CB, October 17, 2012.

[6] K. Collins. An introduction to procedural music in video games. *Contemporary Music Review*, 28(1):5–15, 2009.

[7] I. M. Dart, G. De Rossi, and J. Togelius. Speedrock: procedural rocks through grammars and evolution. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, page 8. ACM, 2011.

[8] A. de la Re, F. Abad, E. Camahort, and M. Juan. Tools for procedural generation of plants in virtual scenes. In *Computational Science – ICCS 2009*, pages 801–810. Springer Berlin / Heidelberg, 2009.

[9] D. g. DeBry, H. Goffin, C. Hecker, O. Quigley, S. Shodhan, and A. Willmott. Player-driven procedural texturing. In *ACM SIGGRAPH 2007 sketches*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[10] E. Hastings, R. Guha, and K. Stanley. Interactive evolution of particle systems for computer graphics and animation. *IEEE Transactions on Evolutionary Computation*, 13(2):418–432, 2009.

[11] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup. Procedural content generation for games: a survey. 2012.

[12] I. D. Horswill. Lightweight procedural animation with believable physical interactions. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(1):39–49, 2009.

[13] R. Hunicke and V. Chapman. AI for dynamic difficulty adjustment in games. In *Challenges in Game Artificial Intelligence AAAI Workshop*, pages 91–96, 2004.

[14] L. Johnson, G. N. Yannakakis, and J. Togelius. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, page 10. ACM, 2010.

[15] M. A. Kowalski, L. Markosian, J. Northrup, L. Bourdev, R. Barzel, L. S. Holden, and J. F. Hughes. Art-based rendering of fur, grass, and trees. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 433–438. ACM Press/Addison-Wesley Publishing Co., 1999.

[16] S. Lefebvre and F. Neyret. Pattern based procedural textures. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, I3D '03, pages 203–212, New York, NY, USA, 2003. ACM.

[17] C. Liu, P. Agrawal, N. Sarkar, and S. Chen. Dynamic difficulty adjustment in computer games through real-time anxiety-based affective feedback. *Intl. Journal of Human–Computer Interaction*, 25(6):506–529, 2009.

[18] R. Lopes and R. Bidarra. Adaptivity challenges in games and simulations: a survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(2):85–99, June 2011.

[19] M. Mateas and A. Stern. Writing façade: A case study in procedural authorship. *Second Person: Role-Playing and Story in Games and Playable Media*, pages 183–208, 2007.

[20] P. Mawhorter and M. Mateas. Procedural level generation using occupancy-regulated extension. In *2010 IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 351–358. IEEE, 2010.

[21] Natural Motion Inc. 'Euphoria: core motion synthesis library'. Oxford, U.K., 2006.

[22] B. Nesmith. Radiant Story: Dynamically created content in The Elder Scrolls V: Skyrim. In *Keynote, Sixth Vancouver Game Design Expo (VGDE '11)*, Vancouver, BC, January 21 - 22, 2011.

[23] K. Perlin. Improving noise. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 681–682. ACM, 2002.

[24] K. Perlin, C. Hecker, C. Reynolds, and F. Kirschner. Four views of procedural character animation for computer games. In *Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, Sandbox '08, pages 61–62, New York, NY, USA, 2008. ACM.

[25] D. Plans and D. Morelli. Experience-driven procedural music generation for games. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(3):192–198, 2012.

[26] W. L. Raffe, F. Zambetta, and X. Li. A survey of procedural terrain generation techniques using evolutionary algorithms. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012.

[27] W. T. Reeves. Particle systems—a technique for modeling a class of fuzzy objects. In *ACM SIGGRAPH Computer Graphics*, volume 17, pages 359–375. ACM, 1983.

[28] R. Smelik, K. Galka, K. J. de Kraker, F. Kuijper, and R. Bidarra. Semantic constraints for procedural generation of virtual worlds. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, page 9. ACM, 2011.

[29] G. Smith, E. Gan, A. Othenin-Girard, and J. Whitehead. Pcg-based game design: enabling new play experiences through procedural content generation. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, page 7. ACM, 2011.

[30] G. Smith, J. Whitehead, and M. Mateas. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 209–216. ACM, 2010.

[31] G. Smith, J. Whitehead, M. Mateas, M. Treanor,

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

[32] N. Sorenson and P. Pasquier. Towards a generic framework for automated video game level creation. *Applications of Evolutionary Computation*, pages 131–140, 2010.

[33] R. Straatman, W. van der Sterren, and A. Beij. Killzone's ai: dynamic procedural combat tactics. In *Game Developers Conference*. Citeseer, 2005.

[34] J. Togelius, E. Kastbjerg, D. Schedl, and G. Yannakakis. What is procedural content generation?: Mario on the borderline. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, page 3. ACM, 2011.

[35] J. Togelius, M. Preuss, and G. N. Yannakakis. Towards multiobjective procedural map generation. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, page 3. ACM, 2010.

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