What is Procedural Content Generation? Mario on the borderline

Julian Togelius, Emil Kastbjerg, David Schedl and Georgios N. Yannakakis
IT University of Copenhagen
Rued Langgaards Vej 7
Copenhagen, Denmark
{juto, eeka, dasc, yannakakis}@itu.dk

ABSTRACT

We try to clarify the concept of procedural content generation (PCG) through contrasting it to other forms of content generation in games with which it could easily be mistaken, and through discussing some properties of PCG which are sometimes thought of as necessary but are actually not. After drawing up some clear demarcations for what is and what is not PCG, we present two versions of a content generation system for Infinite Mario Bros which is intentionally designed to question these same demarcations. We argue that, according to our own definition, one version of the system is an example of PCG while the other is not, even though they are mostly identical. We hope that this paper answers some questions but raises others, and inspires researchers and developers to thread some less common ground in developing content generation techniques.

1. INTRODUCTION

Procedural content generation (PCG) in games refers to the creation of game content automatically using algorithms. Some famous examples are the dungeon generation in *Rogue* (AI Design 1980) (and successors such as *Diablo* (Blizzard 1996), the map generation in *Civilization* (MicroProse 1991), the weapon generation in *Borderlands* (Gearbox 2009) and the vegetation generation by the *SpeedTree* (Interactive Data Visualization 2003) software, included in many modern games. However, this definition is far from precise (could be both too exclusive and too inclusive) and the examples cited are not representative of the most interesting research on PCG that goes on today.

It would be futile to hope to come up with a definition of procedural content generation in games that everybody agrees on. PCG has been attempted by too many people with too many different perspectives for this to happen. A graphics researcher, a game designer in the industry and an academic working on artificial intelligence techniques would be unlikely to agree even on what "content" is, and much less which generation techniques to consider interesting. We argue that PCG is a concept with fuzzy and unclear boundaries. Besides, exact definitions of concepts are common only in mathematics.

This, however, is no reason to try to clarify the concept of procedural content generation and delineate the field a bit. Probing the borders of the field could generate useful discussion, and maybe identify understudied topics and new methods.

The current paper tries to do exactly this. Our strategy is to first give a few examples of what PCG is *not*, and some examples of what PCG is sometimes taken to be, but is not *necessarily*. It is expected that some readers will disagree with these statements, and hoped that these disagreements will seed interesting discussions. The discussion here is meant to complement the taxonomy of PCG given in [9, 10], and we believe the distinctions drawn here to be fully compatible with that taxonomy. PCG algorithms are classified in four main clusters according to the taxonomy presented in this paper.

Next, we describe a small PCG-like system that was built around the popular Infinite Mario Bros framework used in the Mario AI Championship, and designed intentionally to uneasily straddle the borders drawn up in the previous discussion. To make matters worse, two versions of this system were constructed, which differ in important respects related to these examples. It is entirely possible to argue that both versions of the system presented here are procedural content generators, that only one of them is (even though they are very similar in most respects), or perhaps that none of them is.

1.1 What PCG is not: offline player-created content

Many games contain, or come with, some form of content editors. These could be map editors, level editors, character editors (both for appearance and for traits relevant to the game mechanics), item editors and so on. Sometimes there can be much satisfaction to be had from the assemblage and editing of the content itself, quite regardless of its use in the game; a good example of this is *Spore* (Electronic Arts 2009), a game which was preceded by a separate release of a limited version of its *creature creator*. The creature creator was free to download and became a very popular game-like toy in its own right; likely, many more people "played" the creature creator than those who played the full game, which features the creature creator.

Some game editors are connected to online systems for sharing the player-created content. A recent very successful

PCGames 2011 June 28, 2011, Bordeaux, France

example of this is *LittleBigPlanet* (Sony 2009), where the level editor allows uploading of completed levels to a central server, and another interface allows the browsing, downloading and playing of levels created by other players. However, the history of sharing user-created levels goes back at least to *Doom* (id Software 1993). Spore also features mechanisms for sharing player-created content, including creatures and whole planets. (A non-trivial amount of the uploaded Spore creatures bear clear resemblance to human male external reproductive organs, a fact which might explain why some game developers are reluctant to include mechanisms for sharing of player-created content.)

At this point, it would be easy to say that player-created content is not procedurally generated content because the content is created by human players. However, there are PCG algorithms that feature significant human input. In the mixed-initiative paradigm, a human and an algorithm or a set of algorithms cooperate on creating content, e.g. through taking turns. Examples include the *Tanagra* system for level generation in 2D platform games, where a human designer draws part of a level and a constraint satisfaction algorithm is used to generate parts of the level that accommodate the human-generated part while retaining playability [6]; and the *Sketchaworld* framework for creating landscapes and cityscapes on multiple levels of detail, where a set of PCG algorithms ensure the consistency of the world on levels below that currently being edited [5].

The difference between mixed-initiative PCG and level editors seems to be the *directness* of the editing. In a typical level editor, there is a *direct*, *immediate* and typically *local* connection between what the user does and how the content changes. In contrast, a (mixed-initiative) PCG system involves a non-trivial amount of computation (the "procedural" part of PCG) between the user input and the content change; the change might not be immediate, not local and not direct in the sense that it is not straightforward for the human to predict the exact changes that will come about as a result as a particular input.

1.2 What PCG is not: online player-created content

Many games are about building things. In particular, this goes for strategy games and God games. An extreme example is *Sim City* (Maxis 1990), where all of the gameplay is about building and maintaining a city. There are no predetermined goals, and the game even includes functions for destroying the city in fanciful ways, such as earthquakes and monster invasions. Given the lack of goals, it could be argued that Sim City is not really a game, and in fact its creator Will Wright has claimed that he is building toys rather than games. (An even more extreme version of this freeform building approach to games/toys is the recent indie game phenomenon *Minecraft* (Markus Persson 2010).)

However, even strategy games with clear goals, such as Civilization, feature building as a key element of the game. In Civilization, the player has considerable freedom in steering his civilization in whatever direction he or she wants, building and naming cities, building facilities and troops in cities and connecting them with roads, and guiding scientific development. For many players, the building process is the main and most enjoyable part of gameplay, and military conflict is of secondary importance; the military conflict is based on the civilization that the player has already built, and the building phases of the game can thus be considered as content generation for the military conflict phases. (Though there is no clear separation between phases, and city/road building can be performed for tactical as well as strategical reasons.)

Further, the effects of the player's action are rather indirect: the player can control where to place a city, but cannot directly control how fast it will grow or how much it will produce (this depends on game mechanics specified by the designer and opaque to the player). A similar situation applies to Sim City, and the non-directness is arguably what turns Sim City into a successful game/toy rather than an editor. It would, therefore, qualify as PCG according to the discussion in the previous section.

For precisely this reason we propose to rather arbitrarily stipulate that if the human input to the content generator is part of a game, and *the player directly intends to create content in the game*, it is not procedural content generation. We draw this line simply to make sure that PCG is something distinct from strategy game gameplay. This is of course not to say that PCG could not exist in strategy games: in fact, the (non-interactive) map generation at the beginning of a Civilization game is a good example of PCG. It could also be argued that the development of the non-player civilizations is an example of online PCG, as their development is informed by the player's actions, but the player cannot predict in detail how non-player civilizations will respond. To put it succinctly, a player can decide where to place his/her own city, but not where the opponent places it's city.

1.3 What PCG might and might not be: random

The Civilization map generator is often casually described as a *random* map generator. Likewise, the dungeon generation in Rogue and the level generation in *Infinite Mario Bros* (Markus Persson 2009) are often referred to as random map generation. These are somewhat problematic statements — what does "random" mean in this context?

It presumably does not, or at least should not, mean that the generated content is an assortment of design elements (islands, walls, monsters, platforms) spread more or less uniformly over some play area without any regard to structure. Such content would simply be unplayable, and this is in fact not how the aforementioned content generators work. In fact, almost all existing content generators (including those of Rogue and Infinite Mario Bros) include provisions to ensure playability and promote engagement by creating content that somehow adheres to rules (such as that there should be a path from entrance to exit, all islands should have some resources etc).

A much better interpretation of randomness in this context is that the generators include some stochasticity: there are strong constraints on what kinds of content can be generated, but that within these constraints the content can vary according to some pseudo-random process. This is true for most PCG implementations that we know of.

However, there is no reason that a content generator needs to include stochasticity. (Though note the dissenting opinion of Roguelike developer Andrew Doull, whose definition of PCG includes randomness [1].) One interesting example of a deterministic content generator is the representation of star systems in space trading/fighting game *Elite* (Acornsoft 1984). Due to memory limitations on home computers of the

day, it was impossible to explicitly store all the star system information (names, coordinates and sizes of stars and planets, prices of common resources, frequency of pirates etc) in working memory. Therefore, a PCG algorithm was developed that generated all of this information deterministically from a single seed value, and the game universe was simply stored as a list of seed values. PCG is here used as a form of data compression.

Finally, it should be pointed out that "random" is not the same as "statistically uniform". PCG algorithms could take parameters that affect either the constraints on generated content (e.g. the maximum damage of a weapon) or the statistical properties of the stochastic process (e.g. the probability of requiring non-standard ammunition) or both. Biased or non-biased, randomness can act upon nominal, ordinal or numerical scales.

1.4 What PCG might and might not be: adaptive

A particular case of parametrizable PCG is where the parameters somehow depend on the player's (or group of players') previous behaviour. This yields *adaptive* or player-driven PCG, which is currently an active research direction in academia [7, 2, 4] (see also [10] for an extensive review), but is so far more or less absent from PCG in commercial games. Adaptive PCG could have various motivations, such as adjusting the difficulty level of the newly generated content to suit the estimated playing skill of the player, or to generate more content similar to content the player seems to have liked in the past.

The timescale at which adaptation occurs could conceivably vary immensely: between games, between levels or even on a second-to-second level, though we do not know of any example of the latter. (There are examples of second-to-second non-adaptive PCG though, such as the iPhone shooter *Phoenix* (Firi Games 2010). There are also many games that feature online dynamic difficulty adjustment, but this is usually much too simple to qualify as content generation.) Further, just as PCG in general could be deterministic, it would certainly be possible to design a deterministic adaptive PCG game, where the character of the newly generated content depends solely on the player's previous action.

Let us now imagine a game with deterministic adaptive PCG on a second-to-second scale. This means that new game content is continuously created based solely on the player's recent actions. What differentiates this form of content generation from "content created during gameplay", such as the placement of cities and roads in Civilization, which we argued in section 1.2 was not PCG?

Our answer is again to do with intention: when a player places a city in Civilization, it is because he or she has an intention of placing a city there; were a PCG algorithm to place a city, it would probably be as a result of something else the player did. Of course, if the actions of the content generator are easy enough for the player to predict, the player could use it as a gameplay device, and it would then cease to be PCG.

2. DIRECT LEVEL GENERATION IN INFINITE MARIO BROS

We now turn our attention from dichotomizing in the abstract towards a very concrete project. Below, we will describe two versions of a method that was developed to do PCG in ways which it is not usually done, and which uneasily straddles the border between PCG and other forms of content creation.

The implementation is based on Infinite Mario Bros, an open-source Java clone of Nintendo's classic platform game *Super Mario Bros.* We have previously used this game as a basis for a series of AI competitions [8] and also for a series of experiments in adaptive level generation [3, 4]. Infinite Mario Bros is very well suited to these kinds of experiments, as the game mechanics and iconography are among the most well-known of any game ever, and levels are natively represented as straightforward grids of blocks; each block is 8×8 pixels and a standard screen is 22×15 blocks. In the experiments presented in [3, 4], models were created of how enjoyable players with different playing styles found different levels. (Neural network models using level features and gameplay features as input and delivering predicted player engagement as output were trained based on forced choice preference data from hundreds of players.) These models were then used to automatically find level parameters that were predicted to be as fun as possible for individual players, based on the playing style exhibited during the previous levels.

In other words, in our previous experiments new levels were created based on the playing style of the previous level. This means a timescale of about a minute, given that relatively short levels were produced. The generation was stochastic and highly indirect: a player (even one who understood the general architecture of the content generator) would be unlikely to understand what aspects of his playing style in the previous level gave rise to any particular feature of the new level (such as the frequency of gaps, much less why there was a gap at a particular location).

The level generator we present here starts with doing away the indirectness and almost all of the randomness that feature in our previous level generators. The game featuring the level generator is playable online[1] and we recommend the reader to take a few minutes to play with it in connection with reading this paper.

2.1 Offline version

The offline version of the direct level generator starts with letting the player play an ordinary level (of about two to three minutes length) of the game, which has been randomly generated using the standard level generator included with Infinite Mario Bros. This level is meant to be easy to complete, and merely make sure that there is some diversity among the player's actions. Initially completely flat levels were used, but this meant that some players would only walk from left to right and never press the jump or shoot button (see figure 1). All actions taken by the player during gameplay are recorded with frame resolution (the game runs at 24 frames per second).

Following completion of the first level, a new level is generated for the player to play. The generation process starts with a copy of the previous level. The level generator then steps through the recorded actions of the player, and modifies the new level according to the actions taken. A simple set of rules determines how to modify the level at each position depending on what action the player took at that position. If there can be said to be an overarching princi-

[1] http://www.itu.dk/people/eeka/PCGMario.jar

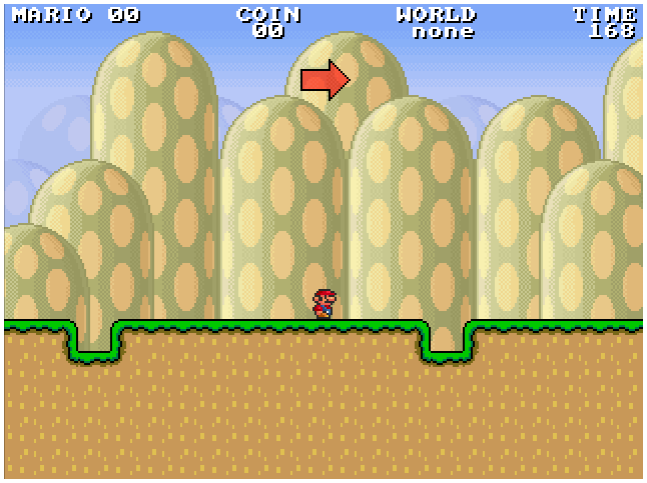

Figure 1: An almost flat part of a level used as initial level. The tiny gaps require the player to at least occasionally press either the jump or run button.

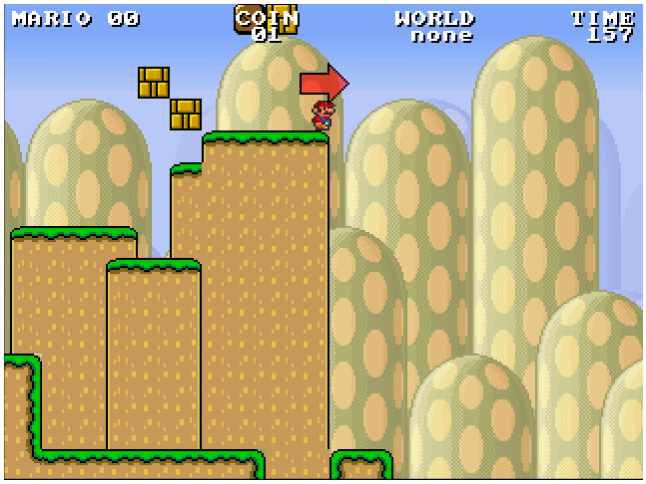

Figure 2: Ground raised and platforms created in response to recorded presses and releases of the jump button. Note the large gap to the right of Mario, the width of which matches the raised and/or created segments to the left.

ple for the generator, it is *balance* or "a reaction for every action". The whole level generation process happens virtually instantly, and is invisible for the player. Each recorded action is interpreted as follows:

- Jump button pressed: at positions where the jump button was pressed, a block is created 64 pixels (4 blocks) above Mario. This block is randomly selected to be either a question mark block with a power-up inside, a question mark block with a coin inside or an empty block.
- Jump button released: where the jump button was released, the ground is modified. The modification depends on Mario's height and direction when the button was released. If Mario was in the lower half of the screen, the ground of the level is raised below the current position up to where Mario was standing. If Mario was instead in the upper half of the screen, a new platform is created below Mario. For the sake of balance, a lethal gap in the ground is created in front of every newly created stretch of ground or platform; the width of the gap equals the height of the created ground or platform. See figure 2 for an example of the type of ground and platform modification performed by the generator in response to jumping.
- Speed/fire button pressed: Where the speed/fire button was pressed, enemies are created a number of blocks to the right of Mario (in the direction of Mario's movement). The enemy type is randomly selected among all available types (Koopas, Goombas etc).

After each modification, numerous corrective actions are performed by the generator to maintain consistency. This includes removing pipes that miss one side because of elevation changes and extending the supporting pillar for cannons that suddenly find themselves floating in thin air. All of these actions are deterministic.

After the player has completed the newly generated level, a new level is generated based on the previous level and the actions taken by the player. This goes on for as long as the player wants, potentially leading to ever more intricate levels.

2.2 Online version

The offline generator works at roughly the same timescale as our previous level generation experiments — one level, or about a minute. Next, we decided to drastically shorten this timescale. In the online version of the generator, the same generation process is carried out as in the offline version. Additionally, two additional rules are added to the generator ruleset:

- Coin collected: where coins were collected, enemies are created to the right of Mario (in the same way as where the speed/fire button was pressed).
- Enemy stomped: where enemies were stomped, coins are placed either to right or left.

The above two rules taken together ensure a balanced game: the more coins the player collects, the more enemies are added to the next level, and the more enemies are added the more coins are collected.

The most important difference, however, is that instead of waiting until the next level is generated the modifications happen instantly on the level currently being played. This means that ground literally appears below the feet of Mario, gaping holes appear in front of him and enemies and coins spawn out of thin air, all in immediate response to the actions taken. Figure 3 illustrates what happens when a long row of coins have just been collected: this greed is punished by unleashing a veritable onslaught of enemies.

2.3 Player reactions

Both versions of the game were tested on a handful of students and academics at the IT University of Copenhagen. The offline level generator was relatively straightforward to

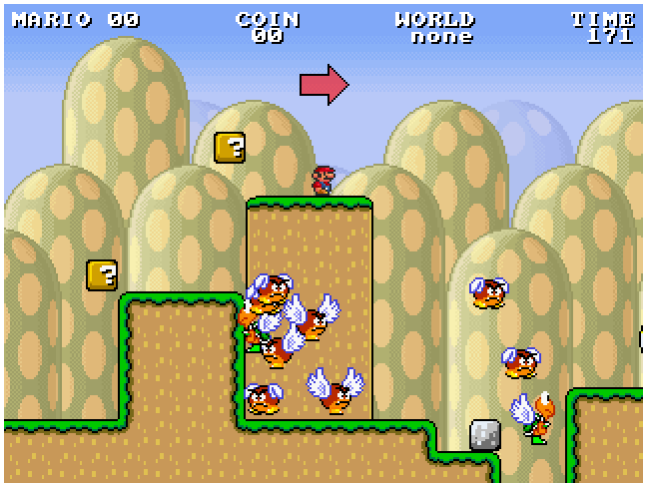

Figure 3: Enemies created in response to coins collected.

play for most subjects. In fact, unless told so, most players did not realize that the second level was created based on their own actions in the previous level. Some players remarked that the first level was boring, and subsequent levels were strange and often unpredictable, with easy segments mixed up with very hard segments.

The online version took all players by surprise. The level changing in real time is nothing you expect from a game, especially not a game strongly resembling a very well-known game for which the player already has a mental model. In particular, nobody expects local changes relating to player actions. Some players burst out laughing at the absurdness of it all. Most players found the online version hard to play, and the rules for content generation hard to comprehend even though they are actually very simple. This was probably both because of the discord between the established mental model of Super Mario Bros and the events on screen, and because there is sometimes very much happening on screen when playing with the online content generator.

2.4 Issues and possible future developments

While the individual modifications are quite simple in nature, the compounded effect of several modifications might have undesirable effects. When several levels are played in sequence with the offline generation mode, the height of ground and platforms tends to rise for each level, due to the effects of the ground being raised and platforms created multiple times above each other. Currently this issue is handled by specifying a maximum height at which platforms can be generated, but a better solution would be to occasionally lower the whole level so as to keep a medium height. For the offline generation one issue is how to deal with few recorded data in cases where the player is dying too early in the previous level. Another issue is that the placement of pipes and cannons can, in conjunction with the raising of ground and opening of gaps, sometimes make levels unplayable. This could be solved simply by adding more constraints on the pipe and cannon placement. It should be noted, though, that the algorithm never generates gaps that are too wide to jump over, as the gap is never wider than the length of the jump that preceded it.

The ideas behind this level generator could be extended to other games. It is quite possible to imagine a first person shooter map where more non-player characters are spawned in places where the player picked up items in the last level and items spawned where the player previously took damage, or a strategy game where the map geometry changes between turns to reflect which parts of the map are used or not used by the units. However, getting these designs to work in actual games is another matter.

2.5 Discussion

The level generation described above is quite different from all PCG examples we know of. So, is it still PCG? We would argue that, according to our discussion in the first part of the paper, the offline version is an example of PCG, whereas the online version is probably a tool for player-created content.

Comparing the offline version of the current generation to our previous experiments (see [4]), the level generation is still adaptive to the actions of the player in the previous level and arguably even more so, as the adaptation is not only based on high-level metrics (the number of times the player jumped, proportion of time spent running, etc.) but on the position of each particular action. There is still some degree of randomness, though much less than in our previous experiments. It is in theory possible to predict and intentionally shape the next level (as the effects of actions are local and quite direct) by conducting actions in the current level, but we seriously doubt whether human memory capacity is enough for this.

The online version is another matter. The effects of actions on the level are not only local and relatively direct, but also instant. As the results of all actions up until the current one are visible and exploitable by the player, it is possible to intentionally modify the level either in order to progress through the game or to just edit the level as the player sees fit. In other words, if the player has the intention to complete the level, it is most appropriate to see the level generator as a gameplay mechanism and the generated levels as content generated during gameplay, similar to Minecraft or Civilization. If the player has the intention to create an interesting level rather than just play it through to the end, it is more appropriate to think of the level generator as an interactive level editor and the generated levels as player-generated content, similar to the editors in LittleBigPlanet or Spore.

3. WHAT IS PROCEDURAL CONTENT GENERATION, AGAIN?

Let us revisit the sentence that started this paper: *Procedural content generation (PCG) in games refers to the creation of game content automatically using algorithms.* For the reader who agrees with the arguments made so far in the paper, or at least with the conclusions of those arguments, this definition becomes too broad, as it encompasses things that we do not consider PCG — in particular, content created directly by players in an editor or as part of gameplay, but assisted by algorithms. At the same time, the definition could be too narrow: the word "automatic" could suggest that the designer or player's input can not be considered by a PCG algorithm, which would exclude both

adaptive and mixed-initiative approaches to PCG. In fact, some input from a designer or player is typically required for the generation of content, even if it so simple as clicking the "start" button. We can therefore tentatively redefine PCG as *the algorithmical creation of game content with limited or indirect user input*. Note that this definition does not contain the words "random" and "adaptive", as PCG methods could be both, either or none.

4. CONCLUSIONS?

In this paper, we have tried to clarify what procedural content generation is by first talking about what it is not and what it is only sometimes. We have then described two versions of a level generator that was designed to do content generation in a way it is not typically done, and to test the limits of PCG as we have just defined it. It turns out that the way we have defined PCG, one version of the content generator falls inside the demarcation while the other (which works identically but on a different timescale) falls outside. This way, we hope to have helped you and us understand what we talk about when we talk about procedural content.

Of course, one might also take the view that all these demarcations are rather phony — after all, who cares how we choose to apply the rather artificial label "procedural content generation"? What matters is individual algorithms, designs and ideas. This is an entirely reasonable position. However, discussions such as the current can be valuable also as a way of charting the research field and point to overlooked combinations of ideas. Such as the generator presented here, which is the only example we know of a both adaptive and direct level generator, and also the only example we know of an almost deterministic online generator. We hope to have inspired you similarly to explore new directions in PCG, even if you do not agree with the distinctions we have drawn and the arguments we have made.

5. ACKNOWLEDGMENTS

This research was supported in part by the Danish Research Agency, Ministry of Science, Technology and Innovation; project name: Adaptive Game Content Creation using Computational Intelligence (*AGameComIn*); project number: 274-09-0083. Thanks to Noor Shaker for additional input.

6. REFERENCES

[1] A. Doull. The death of the level designer, 2008.

[2] E. Hastings, R. Guha, and K. O. Stanley. Evolving content in the galactic arms race video game. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, 2009.

[3] C. Pedersen, J. Togelius, and G. N. Yannakakis. Modeling Player Experience for Content Creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):54–67, 2010.

[4] N. Shaker, G. N. Yannakakis, and J. Togelius. Towards Automatic Personalized Content Generation for Platform Games. In *Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE'10)*, pages 63–68, Palo Alto, CA, October 2010. AAAI Press.

[5] R. M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra. Integrating procedural generation and manual editing of virtual worlds. In *Proceedings of the ACM Foundations of Digital Games*. ACM Press, June 2010.

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

[7] J. Togelius, R. De Nardi, and S. M. Lucas. Towards automatic personalised content creation in racing games. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, 2007.

[8] J. Togelius, S. Karakovskiy, and R. Baumgarten. The 2009 mario ai competition. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2010.

[9] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-based procedural content generation. In *Proceedings of EvoApplications*, volume 6024. Springer LNCS, 2010.

[10] G. N. Yannakakis and J. Togelius. Experience-driven Procedural Content Generation. *IEEE Transactions on Affective Computing*, 2011. (to appear).