Procedural Content Generation in the Game Industry

Alba Amato

Abstract Game content construction and generation are laborious and expensive. Procedural content generation (PCG) aims at generating game content automatically using algorithms, reducing the cost of game design and development. PCG systems have the potential to act as "on-demand game designers," but need to be as flexible as possible while creating content that meets the needs of designers and players. The aim of the chapter is to (1) analyze critically the utilization of PCG in the Game Industry; (2) present a taxonomy for PCG games; (3) address problems in applying PCG and discuss aspects of various solutions.

1 Introduction

Game content construction and generation are laborious and expensive. Procedural content generation (PCG) aims at generating game content automatically using algorithms, thus reducing the cost of game design and development (Risi et al. 2014). Moreover, PCG can also provide a way to generate personalized games that can adapt content according to a player's preference and optimizing their gaming experience. For those reasons, PCG is getting increasingly popular in game development field (Yannakakis and Togelius 2015).

The overall aim of this chapter is to analyze critically the utilization of PCG in the game industry, addressing problems and discussing the important aspects of the different solutions. However, I will not cover all those aspects. Yannakakis and Togelius (2015) arrived at a list of eight research challenges for procedural content generation:

A. Amato (✉)
Second University of Naples, Caserta, Italy
e-mail: alba.amato@unina2.it

O. Korn and N. Lee (eds.), *Game Dynamics*,
DOI 10.1007/978-3-319-53088-8_2

- to create content generators that can generate content that is purposeful, coherent, original and creative
- to create a content generator that can create content in a particular style that it has somehow learned or inferred
- to generate multiple types of content for multiple games as almost all existing PCG algorithms generate a single type of content for a single game, the reusability of developed PCG systems is very limited and there is no plug-and-play content generation available
- to create tools to aid in adapting a particular representation for a particular class of content so that a particular representation could be adapted to a particular content class more easily
- to devise workable methods for communication and collaboration between algorithms working on generating different aspects or layers of the same artifact;
- to create believable animations even if the underlying characters are not procedurally generated;
- to establish game/music systems that interact in a much more complex way than what we are used to see in most games;
- to define theory and taxonomy of PCG in order to explain the relative advantages of different approaches, why some content generation problems are harder than others, and which approaches are likely to work on what problems.

In this chapter, I will mainly focus on the last challenge as there are diverse approaches for generating content for games, which are used in different genres and, due to this wide variety, it is difficult to compare the approaches without a standard taxonomy.

In Sect. 2, I will present technical background of PCG. In Sect. 3, a taxonomy of approaches will be presented, as well as an attempt to provide a common terminology and classification. Section 4 will focus on the history of the utilization of PCG in game industry, overviewing some of the most relevant and common algorithms and techniques used to develop PCG systems, highlighting the advantages of the different solutions. The final part of this chapter will be dedicated to questions and challenge relevant for the future of PCG in the game industry.

2 What Is PCG and Why It Is Used in the Game Industry

Procedural content generation refers to the creation of game content automatically (or semi-automatically) through algorithmic means (Yannakakis and Togelius 2015). It is a methodology for automatic generation of content of an entity, typically a game using algorithms or processes which can produce, due to their random nature, a very wide range of possible content related to the considered entity. The keystone of this methodology is the concept of randomness: Using a few parameters, the application of PCG ensures the creation of a high number of possible contents of a game, all differing from each other. It is important to define the

meaning of the word "content" because according to its definition we can distinguish whether a content falls within the domain of the PCG or in other domains. The results from the application of PCG algorithms can be all kinds of elements affecting the gameplay: terrain, maps, layers, stories, dialogues, quests, characters, rules, dynamics, or weapons.

There is still not a clear distinction between PCG and PG (procedural generation), and often the two terms are used as synonyms, since they are based on the same algorithms to achieve different results. The main difference lies in the results generated: PCG generates content consisting of several components, while each individual component can be generated from the PG. PG also creates content that does not affect the gameplay directly, for example, procedural textures and procedural animation. A texture created by procedural generation techniques relates to PG, while the set of textures that form the setting of a level created by procedural generation techniques relates to PCG.

PCG has the potential to radically change how we conceptualize games, but also faces significant challenges regarding its integration into design practice. In recent years, it has become a mainstay of game AI, and significant research is directed toward the investigation of new PCG systems, algorithms and techniques (Khaled et al. 2013). In fact, there are a number of reasons for using and developing PCG techniques in games. The production of a game with high-level content implies a big effort in terms of cost and time. Normally one would expect that, with the growth of technology and the introduction of ever faster and more functional applications, this process of creation had at least speeded up, or even at least partially been automated. However, while technology in videogames is highly advanced, the creation of realistic 3D environments, rich dynamics and details of high-level graphics and content's creation in general are still largely manual. Usually, a team of people from different production departments of the game (programmers, sound engineers, artists, etc.) is responsible for the manual creation of all game contents. This has a significant effect on the budget available for the development, especially when there is the necessity to create a high quality game; as a result, content creation is viewed as the "bottleneck" for the total budget of a game.

The bottleneck created by the high costs and long development times of the game represents a barrier to the artistic and technological progress of games on any platform. Another reason for increasing PCG utilization is that it can potentially create infinite games. This happens when it is used for the generation of content in real time with a sufficiently high degree of variety. The history of games shows that this mechanism can lead to success since the early 1980s, when the *Rogue* videogame opened the way for PCG. *Rogue* is generally credited with being the first "graphical" adventure game. Its biggest contribution, and one that still stands out to this day, is that the adventure in *Rogue* was generated algorithmically. Every time you played, a new adventure was created. That is really, what made it so popular for several years in the early eighties. The vision of creating endless games attracted many developers, and this method was imitated numerous times over the years, for example, in the hit game series *Diablo*.

The use of mass storage is another point in favor of the use of procedural content generation. This problem was most significant in the early 1980s, when memory limitations of the existing end-user platforms did not allow the distribution of large amounts of predesigned content such as game levels. This problem reappeared 20 years later with the first mobile games. The advantage of using PCG is that content represented procedurally is "compressed" as long as it is not required by the gameplay. The adventure game *Elite* is the example par excellence: It could handle thousands of star systems in a dozen kilobytes of memory, representing only a few numbers of planets in compressed form. In expanded form, each planet had a name, population, prices of raw materials and so on. Even today, the use of storage should always be taken into account: Despite the large increases in modern hard drive capacities and the decrease in the price per unit, just a few video games with many GB each force many users to uninstall some games or to buy new storage devices to make space for new ones.

An example that addresses this memory problem is the game. *.kkrieger 1*, a 3D first person shooter, similar in kind to *Halo 3*. Unlike the latter, it uses procedural generation techniques to create textures, meshes and sounds which then are combined to form complete setting. *.kkrieger* makes extensive use of procedural generation methods: Textures are stored via their creation history instead of a per-pixel basis, thus only requiring the history data and the generator code to be compiled into the executable, producing a relatively small file size. Meshes are created from basic solids such as boxes and cylinders, which are then deformed to achieve the desired shape—essentially a special way of box modeling. These generation processes account for the extensive loading time of the game, as all assets relevant for the gameplay are produced during the loading phase. The entire game uses only 97,280 bytes of disk space—about 3 to 4 orders of magnitude less than in a similar game like *Halo 3*. According to the developers, *.kkrieger* itself would take up around 200 GB of storage space if it was stored in a conventional way.

Finally, PCG can increase the limits of human imagination. The algorithms may create new rules, levels, stories, which, in turn, can inspire designers to create new games (Fig. 1).

3 A Taxonomy of PCG

PCG has probably been used by too many people with too many different perspectives to arrive at a definition of procedural content generation that everybody agrees on. A graphics researcher, a designer in the game industry and an academic working on artificial intelligence techniques would be unlikely to agree even on what "content" is, and even less on which generation techniques are interesting. We can group, according to the latest classification of Hendrikx et al. (2013), the type of content that can be produced by procedural generation techniques into five main classes:

Fig. 1 Screen-shot of.kkrieger

- **Game Bits** are elementary units of the game content that do not affect the player's gameplay when considered independently. Included in this category: textures, sounds, vegetation, structures, behaviors, fire, water, stone, or clouds.
- **Game Space** represents the environment in which to play. It consists of different units Game Bits. Included in this category: indoor maps, outdoor maps, water bodies such as seas, lakes, or rivers.
- **Game System**, for example, ecosystems, road networks, urban planning.
- **Game Scenarios**, in which events occur, e.g., puzzle, storyboard, the history, the concept of levels.
- **Game Design**, including rules and objectives.

The type of algorithm suited for the creation of an element of a class depends on the type of element. Different algorithms can be used, starting with a simple pseudorandom number generator, to generative grammars up to the most advanced evolutionary programming techniques and artificial neural networks.

In Togelius et al. (2011a, b) another taxonomy for procedural content generation is proposed, centering on what kind of content is generated, how the content is represented and how the quality/fitness of the content is evaluated; search-based procedural content generation in particular is situated within this taxonomy. This work also contains a survey of papers in which game content is generated through search or optimization and ends with an overview of important open research problems. Shaker et al. (2015) present a revision of the above taxonomy that currently represents an important point of reference for many researchers.

- **Online—Offline Generation**: The first distinction concerns whether the content is generated online (while the game is running) or offline (during development of the game). An example of online PCG is when the player opens a door of a structure and the game instantly generates the interior: rooms, walls,

decorations, etc. An example of offline PCG is when an algorithm creates the basic internal layout of a structure, which is then modified and refined by a designer before the game is completed. Clearly, the online PCG must comply with the basic requirements for creating valuable content: It must be fast, and the content must be qualitatively acceptable (depending on context).

- **Necessary—Optional Content**: The second distinction refers to the importance of the content, if it is necessary or optional for the particular game. Necessary content is required by players to progress through the game. For instance, dungeons that have to be crossed, monsters that must be defeated, crucial rules of the game, etc., fall into this category. Optional content is everything that the player can choose to not consider, such as weapons or facilities that can be ignored. Another difference between these two variants is the required quality: Necessary content must always be of excellent quality and functionally correct. It is not acceptable to create an unpassable dungeon, incorrect rules or unbeatable monsters if these anomalies make it impossible to continue the game. It is also not acceptable to generate the content that does not have a fair difficulty compared to the rest of the game. On the other hand, for optional content, it is principally allowed that an algorithm also produces unusable weapons or an unreasonable dungeon layout if the player can choose to discard the weapon or leave the dungeon.
  Clearly, the importance given to a content is dependent on the design of the game and its storyline. For example, the first person shooter *Borderlands* has a random generation mechanism of weapons, many of which are not useful, but analyzing these weapons is part of the heart of the gameplay. On the other side, a structure with a simple and low quality level of detail seems artificial and can make no credible setting in a game that as a main objective has visual realism, such as *Call of Duty 4: Modern Warfare*. It is interesting to note that some content may be optional in a category of games and necessary in others (for example, dungeons). Thus, the analysis of what is "optional" in a game needs to be made on a case-by-case basis.
- **Control Degrees**: This distinction depends on the type of generation algorithm that is used and how it can be parameterized. At one extreme, the algorithm can simply use a randomly generated number as an input; at the other extreme, the algorithm can use as a multi-dimensional vector containing the parameters that specify the properties of the content to be generated. In the case of the creation of a dungeon, the algorithm can use as input parameters the number of rooms, corridors branching factor, treasures, etc. The more degrees of control there are, the more the generated content can be customized and controlled.
- **Generation Deterministic or Stochastic**: This distinction relates to the degree of randomness in the build process. It is possible to conceive deterministic algorithms that either generate the same content with the same input parameters, or different content even with identical input parameters, as, for example, the *Rogue* dungeon generation algorithm.

Table 1 Revised taxonomy as presented by Shaker et al. (2015)

Online–offline
Necessary content—optional
Control degrees
Generation deterministic or stochastic
Constructive or generative algorithms-with-test

- **Constructive or Generative-with-test**: This last distinction refers to the output of the algorithm. Constructive algorithms generate the content and end their execution, producing the output result. For example, the algorithm of the Markovian process is typically a constructive PCG algorithm that produces stochastic content. However, it is necessary that there is a form of control in such a way as to avoid the production of unsuitable content. This can be done by operations or sequences of operations, which ensure that it cannot produce non-usable material. An example is the use of fractals to generate terrains. A "generative-with-test algorithm" incorporates a mechanism of generation and a test. This means that during the execution of the algorithm, each content instance is tested according to some criteria (which depend on the design of the game). If the candidate fails the test, it is fully or partially discarded and regenerated, and the process continues until the content does have sufficient quality (Table 1).

4 History of PCG in Game Industry

As described in Sect. 2, PCG was born as a way to compress data. *Akalabeth: World of Doom*, designed by Richard Garriott, was the first game to use a seed to generate the game world. It is a roleplaying game published by California Pacific Computer Company for the Apple II in 1980. The famous game *Rogue* (1980) is an open-source, free-to-change game that spawned popular free-to-use games such as *Moria* (1983), *NetHack* (1987) or *Angband* (1990). It is considered the ancestor of the Rogue-like games, which are one of the main game genres to use PCG in modern videogame to create an "endless" game experience (Fig. 2).

The Sentinel (1985) used PCG to create 10,000 unique levels stored in only 48 and 64 kilobytes. A procedural generation algorithm is used which generates each landscape from a small data packet: The 8-digit number given at the completion of a previous landscape. The number of landscapes was quite arbitrary, given the generation algorithm and was chosen as a balance between giving the players good value while not overwhelming them with an unreachable goal. Not all the landscapes were actually tested, but it was always possible to skip a difficult level by completing an earlier one with a different amount of energy.

Elite (1985) used PCG to generate a universe with 8 galaxies and 256 solar system each. Each solar system has 1–12 planets, each with a space station in its

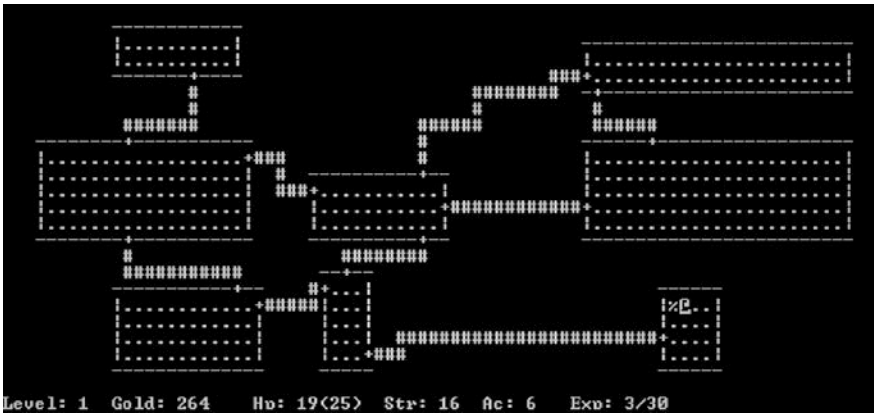

Fig. 2 Screenshot of Rogue

orbit, a name, a terrain, prices of commodities and local details. Due to the limited capabilities of 8-bit computers, these worlds are procedurally generated. A single seed number is run through a fixed algorithm the appropriate number of times and thus creates a sequence of numbers determining each planet's complete composition (position in the galaxy, prices of commodities, name, etc.). Text strings are chosen numerically from a lookup table and assembled to produce unique descriptions, such as a planet with "carnivorous arts graduates". This means that no extra memory is required to store the characteristics of each planet, yet each one is unique and has fixed properties. However, there also were some problems. Some solar systems were poorly connected, and the random name generator sometimes used profanity to name planets or space stations. The new version of *Elite*, *Elite Dangerous* uses PCG to generate a 1:1 replica of the Milky Way with more than 400 billion star systems. If we assume a storage requirement of just 1 KB per star system (very optimistic), without PCG, the full system would occupy more than 400 Terabyte (Aversa 2016).

With the establishment of CD-ROMs in 1980s, developers became able to store more data, so using procedural generation to build large worlds became less essential. During the 1990s, PCG was utilized to automatize designing assets (e.g., trees, rocks, foliage) to increase the amount of contents in a game in order to provide an improved replay value (Aversa 2016). *Diablo* (1996), shown in Fig. 3, was developed 16 years after the release of *Rogue* by Blizzard Entertainment and was one of the first games to bring the Rogue-like genre into the modern age. *Diablo* did a lot of things right, but as far as procedural generation goes, it popularized two specific gameplay elements: random dungeon layouts and random item or "loot" generation. Just like *Rogue* and all of its clones, *Diablo* generated dungeons according to an algorithm with random elements, but it took the process a step further: Instead of simply matching ASCII characters, *Diablo's* dungeons were created using 2D isometric graphics. By applying elements of randomization to its

Fig. 3 Screenshot of Diablo (1996)

item system, it pioneered a type of gameplay still common today, e.g., in *Borderlands* or *Torchlight*. Color-coded tiers of rarity categorized items and each item's stats were generated on-the-fly (Lee 2016): a novelty in commercial videogames.

After 1996, commercial design tools using PCG were established, so PCG was no more confined to roleplaying games (RPG), space games or Rogue-like games. This gave rise to the modern age of PCG. The main idea behind procedural content generation is that game content is not generated manually by human designers, but by computers executing a well-defined procedure (Hendrikx et al. 2013). However, today PCG techniques are used to design level contents, helping designers to create game contents more quickly—and often without being noticed by the player as a design technique. When used more intensely, procedural techniques are an alternative to create game worlds in a limited amount of time without putting a large burden on the content designers. Such an intense use of PCG techniques is the generation of game levels/world while the game is being played or loaded: In this case, PCG instantiates the game objects, such as trees, monsters, characters, items, treasures and so on. Such techniques are used in games like *Elite*, *Minecraft*, *Spelunky*, *Diablo*, and many more (Togelius et al. 2013).

In mods, editors or middlewares, users can manually change PCG parameters to generate personalized contents, share these parameters, and so on. Today several middlewares for PCG exist, for instance, *CityEngine* (2016) for the generation of

urban environments or SpeedTree (2016) for the generation of detailed forests. *SpeedTree* is widely used in AAA games, for instance, *The Elder Scrolls IV: Oblivion* by Bethesda or *The Witcher 2*. Finally, PCG techniques are also applied to agent behaviors (Aversa 2016) in order to model dynamic systems such as weather, and group and crowd behavior. For example, S.T.A.L.K.E.R.: The Shadow of Chernobyl, contains one thousand non-scripted characters.

5 Conclusion and Challenges

In this chapter, we presented the most common ways in which PCG is used. We discussed its applications: from early games that used it to deal with strict memory constraints, to recent games that use it to reduce the costs of creating large amounts of content. We presented a literature survey of selected future challenges in PCG, concentrating on themes mentioned in multiple sources where some level of consensus seems to be reached.

Some established game studios are starting to use PCG instead of artists, to produce games faster and cheaper while preserving quality (Shaker et al. 2015). At the same time, for many indie studios, PCG is the only way to produce enough content to create a game with an adequate amount of content and some replay value. PCG can potentially work like on-demand game designers, but needs to be as flexible as possible, rapidly creating content that meets the needs of designers and players while reviewing the quality of that content.

In future, PCG will be highly used for game elements on mobile devices (smartphones and tablets). However, on all platforms automatic techniques for generating contents with an optimal level of difficulty are required to keep the player in a state of flow. While PCG is very strong when there is a dependence on cost and time factors, without appropriate constraints and controls it is unsure if PCG creates the expected play experience. This disadvantage keeps many producers of video games from using PCG. In future research, an important challenge is analyzing the data collected during play sessions in order to detect and correct defects in gameplay and improve the applicability of PCG. It is possible to hypothesize alternative environments that evolve by genetic algorithms, resulting in higher quality variants.

References

Risi, S., Lehman, J., DAmbrosio, D., Stanley, K. (2014). Automatically Categorizing Procedurally Generated Content for Collecting Games. In: *Proceedings of the Workshop on Procedural Content Generation in Games (PCG) at the 9th International Conference on the Foundations of Digital Games (FDG-2014)*

Yannakakis, G.N., Togelius, J. (2015). Experience-driven procedural content generation (extended abstract). In: *ACII, IEEE Computer Society 519–525*

Khaled, R., Nelson, M.J., Barr, P. (2013). Design metaphors for procedural content generation in games. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '13*, New York, NY, USA, ACM 1509–1518

Procedural content generation (2016) Retrieved from: https://pcg.wikidot.com/

Togelius, J., Kastbjerg, E., Schedl, D., Yannakakis, G.N. (2011) What is procedural content generation?: Mario on the borderline. In: *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games. PCGames '11*, New York, NY, USA, ACM (2011) 3:1–3:6

Hendrikx, M., Meijer, S., Van Der Velden, J., Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Trans. Multimedia Comput. Commun. Appl. 9* 1:1–1:22

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

Shaker, N., Togelius, J., Nelson, M.J. (2015). Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer

Elite (2016). Retrieved from: https://en.wikipedia.org/wiki/Elite_(video_game)

Aversa, D. (2016). Procedural contents generation history and techniques used in the modern video-game industry. Retrieved from: www.davideaversa.it/tag/pcg/

Lee, J. (2016). How procedural generation took over the gaming industry. Retrieved from: http://www.makeuseof.com/tag/procedural-generation-took-gaming-industry/

Cityengine (2016). Retrieved from: https://en.wikipedia.org/wiki/CityEngine

Speedtree (2016). Retrieved from: https://en.wikipedia.org/wiki/SpeedTree

Togelius, J., Champandard, A.J., Lanzi, P.L., Mateas, M., Paiva, A., Preuss, M., Stanley, K.O. (2013). Procedural content generation: Goals, challenges and actionable steps. In: *Artificial and Computational Intelligence in Games*. 61–75

Author Biography

Alba Amato is currently a postdoc researcher in computer science at the Second University of Naples. She received her M.Sc. in computer science from the University of Napoli Federico II in 2007. She received her postgraduate degree from the School of Specialization in Computer Science Teaching of the University of Naples Federico II and her Ph.D. from the Second University of Naples.

Her research is focused on multiagent systems, Internet of things, smart grid and cloud computing. She is also interested in big data, artificial intelligence, and the intersection between artificial intelligence and games—in particular, algorithms for the procedural generation of games, levels, narrative, environments, animation, characters, and other game contents together with their historical utilization.

She co-authored more than 50 journal articles, book chapters, and refereed conference papers.

She participated in many national and international conferences and workshops. She is reviewer for several IEEE, Inderscience, and Wiley journals in the areas of computer science and for national and international conferences and workshops.