Chapter 1
Introduction

Julian Togelius, Noor Shaker, and Mark J. Nelson

Abstract This chapter introduces the field of procedural content generation (PCG), as well as the book. We start by defining key terms, such as game content and procedural generation. We then give examples of games that use PCG, outline desirable properties, and provide a taxonomy of different types of PCG. Applications of and approaches to PCG can be described in many different ways, and another section is devoted to seeing PCG through the lens of design metaphors. The chapter finishes by providing an overview of the rest of the book.

1.1 What is procedural content generation?

You have just started reading a book about Procedural Content Generation in Games. This book will contain quite a lot of algorithms and other technical content, and plenty of discussion of game design. But before we get to the meat of the book, let us start with something a bit more dry: definitions. In particular, let us define Procedural Content Generation, which we will frequently abbreviate as PCG. The definition we will use is that *PCG is the algorithmic creation of game content with limited or indirect user input* [32]. In other words, PCG refers to computer software that can create game content on its own, or together with one or many human players or designers.

A key term here is "content". In our definition, content is most of what is contained in a game: levels, maps, game rules, textures, stories, items, quests, music, weapons, vehicles, characters, etc. The game engine itself is not considered to be content in our definition. Further, non-player character behaviour—NPC AI—is not considered to be content either. The reason for this narrowing of the definition of content is that within the field of artificial and computational intelligence in games, there is much more research done in applying CI and AI methods to character behaviour than there is on procedural content generation. While the field of PCG is mostly based on AI methods, we want to set it apart from the more "mainstream"

use of game-based tasks to test AI algorithms, where AI is most often used to learn to play a game. Like all definitions (except perhaps those in mathematics), our definition of PCG is somewhat arbitrary and rather fuzzy around the edges. We will treat it as such, and are mindful that other people define the term differently. In particular, some would rather use the term "generative methods" for a superset of what we call PCG [8].

Another important term is "games". Games are famously hard to define (see Wittgenstein's discussion of the matter [36]), and we will not attempt this here. Suffice it to say that by games we mean such things as videogames, computer games, board games, card games, puzzles, etc. It is important that the content generation system takes the design, affordances and constraints of the game that it is being generated for into account. This sets PCG apart from such endeavours as generative art and many types of computer graphics, which do not take the particular constraints and affordances of game design into account. In particular, a key requirement of generated content is that it must be playable—it should be possible to finish a generated level, ascend a generated staircase, use a generated weapon or win a generated game.

The terms "procedural" and "generation" imply that we are dealing with computer procedures, or algorithms, that create something. A PCG method can be run by a computer (perhaps with human help), and will output something. A PCG *system* refers to a system that incorporates a PCG method as one of its parts, for example an adaptive game or an AI-assisted game design tool. This book will contain plenty of discussion of algorithms and quite a lot of pseudocode, and most of the exercises that accompany the chapters will involve programming.

To make this discussion more concrete, we will list a few things we consider to be PCG:

- A software tool that creates dungeons for an action adventure game such as *The Legend of Zelda* without any human input—each time the tool is run, a new level is created;
- a system that creates new weapons in a space shooter game in response to what the collective of players do, so that the weapons that a player is presented with are evolved versions of weapons other players found fun to use;
- a program that generates complete, playable and balanced board games on its own, perhaps using some existing board games as starting points;
- game engine middleware that rapidly populates a game world with vegetation;
- a graphical design tool that lets a user design maps for a strategy game, while continuously evaluating the designed map for its gameplay properties and suggesting improvements to the map to make it better balanced and more interesting.

In the upcoming chapters, you will find descriptions of all of those things described above. Let us now list a few things that we do not consider to be PCG:

- A map editor for a strategy game that simply lets the user place and remove items, without taking any initiative or doing any generation on its own;
- an artificial player for a board game;
- a game engine capable of integrating automatically generated vegetation.

Several other authors have tackled the issue of surveying PCG or part of the field we call PCG, though the overlap is far from complete [12, 25].

1.2 Why use procedural content generation?

Now that we know what PCG is, let us discuss the reasons for using and developing such methods. It turns out there are a number of different reasons.

Perhaps the most obvious reason to generate content is that it removes the need to have a human designer or artist generate that content. Humans are expensive and slow, and it seems we need more and more of them all the time. Ever since computer games were invented, the number of person-months that go into the development of a successful commercial game has increased more or less constantly.[1] It is now common for a game to be developed by hundreds of people over a period of a year or more. This leads to a situation where fewer games are profitable, and fewer developers can afford to develop a game, leading in turn to less risk-taking and less diversity in the games marketplace. Many of the costly employees necessary in this process are designers and artists rather than programmers. A game development company that could replace some of the artists and designers with algorithms would have a competitive advantage, as games could be produced faster and cheaper while preserving quality. (This argument was made forcefully by legendary game designer Will Wright in his talk "The Future of Content" at the 2005 Game Developers Conference, a talk which helped reinvigorate interest in procedural content generation.)

Of course, threatening to put them out their jobs is no way to sell PCG to designers and artists. We could therefore turn the argument around: content generation, especially embedded in intelligent design tools, can augment the creativity of individual human creators. This could make it possible for small teams without the resources of large companies, and even for hobbyists, to create content-rich games by freeing them from worrying about details and drudge work while retaining overall directorship of the games.

Both of these arguments assume that what we want to make is something like the games we have today. But PCG methods could also enable completely new types of games. To begin with, if we have software that can generate game content at the speed it is being "consumed" (played), there is in principle no reason why games need to end. For everyone who has ever been disappointed by their favourite game not having any more levels to clear, characters to meet, areas to explore, etc., this is an exciting prospect.

Even more excitingly, the newly generated content can be tailored to the tastes and needs of the player playing the game. By combining PCG with player modelling, for example through measuring and using neural networks to model the response of players to individual game elements, we can create player-adaptive games

[1] At least, this is true for "AAA" games, which are boxed games sold at full price worldwide. The recent rise of mobile games seems to have made single-person development feasible again, though average development costs are rising on that front too.

that seek to maximise the enjoyment of players. The same techniques could be used to maximise the learning effects of a serious game, or perhaps the addictiveness of a "casual" game.

Another reason for using PCG is that it might help us to be more creative. Humans, even those of the "creative" vein, tend to imitate each other and themselves. Algorithmic approaches might come up with radically different content than a human would create, through offering an unexpected but valid solution to a given content generation problem. Outside of games, this is a well-known phenomenon in e.g. evolutionary design.

Finally, a completely different but no less important reason for developing PCG methods is to understand design. Computer scientists are fond of saying that you don't really understand a process until you have implemented it in code (and the program runs). Creating software that can competently generate game content could help us understand the process by which we "manually" generate the content, and clarify the affordances and constraints of the design problem we are addressing. This is an iterative process, whereby better PCG methods can lead to better understanding of the design process, which in turn can lead to better PCG algorithms.

1.3 Games that use PCG

Overcoming the storage limitations of computers was one of the main driving forces behind the development of PCG techniques. The limited capabilities of home computers in the early 1980s constrained the space available to store game content, forcing designers to pursue other methods for generating and saving content. *Elite* [4] is one of the early games that solved this problem by storing the seed numbers used to procedurally generate eight galaxies each with 256 planets each with unique properties. Another classical example of the early use of PCG is the early-1980s game *Rogue*, a dungeon-crawling game in which levels are randomly generated every time a new game starts. Automatic generation of game content, however, often comes with tradeoffs; roguelike games can automatically generate compelling experiences, but most of them (such as *Dwarf Fortress* [1]) lack visual appeal.

Procedural content generation has received increasing attention in commercial games. *Diablo* [2] is an action role-playing hack-and-slash videogame featuring procedural generation for creating the maps, and the type, number and placement of items and monsters. PCG is a central feature in *Spore* [15] where the designs the players create are animated using procedural animation techniques. These personalised creatures are then used to populate a procedurally generated galaxy. *Civilization IV* [10] is a turn-based strategy game that allows unique gameplay experience by generating random maps. *Minecraft* [19] is a massively popular game featuring extensive use of PCG techniques to generate the whole world and its content. *Spelunky* [39, 38] is another notable 2D platform roguelike indie game that utilizes PCG to automatically generate variations of game levels (Figure 1.1). *Tiny Wings*

Fig. 1.1: Screenshot from *Spelunky*

[13] is yet another example of a mobile 2D game featuring a procedural terrain and texture generation system giving the game a different look with each replay.

1.4 Visions for PCG

As we have seen, procedural content generation has been a part of some published games for three decades. In the past few years, there has also been a surge in academic research on PCG, where researchers from very different academic backgrounds have brought their perspectives and methods to bear on the problems of game content generation. This has resulted in a number of new methods, and variations and combinations of old methods, some of which are in need of further research and development before being useful in actual games. The chapters of this book will present many of the most significant contributions of recent years' research.

To guide the research being done, it is useful to have some visions of where we might be going; this is analogous to lists of "unsolved problems" in some research fields such as mathematics and physics. The authors of a recent survey paper defined three such visions for procedural content generation [31]. These are things that we cannot do with current technology, and might never be possible to achieve exactly as stated, but serve to point out limitations of the state of the art and by extension interesting problems to work on.

1. *Multi-level, multi-content PCG* refers to a content generator that, for a given game engine and set of game rules, would be able to generate all of the content for the game such that the content is of high quality and fits together perfectly.

For example, given the engine and ruleset for the popular computer role-playing game *Skyrim*, this imaginary software would generate backstory, quests, characters, items, weapons, vegetation, terrain, graphics, etc. in such a fashion that it all becomes a coherent, believable new world and an enjoyable game to play.

2. *PCG-based game design* refers to creating games that do not only rely on procedural content generation, but for which PCG is an absolutely central part of the gameplay, so that if you took the content generation part away there would not be anything recognisable left of the game. Some progress has been made towards this, notably in games such as *Galactic Arms Race* [11] and *Endless Web* [28], but these games are still based on established game genres and core parts of the games could function without PCG.
3. *Generating complete games* refers to a generator capable of generating not only content for a given game, but the game itself. This means the rules, reward structures and graphical representation as well as the levels, characters, etc. Some work has been done in this direction, mainly to generate rules for different kinds of games [33, 7, 20, 9], but the rules generated are so far rather simplistic.

Much of the work described in the upcoming chapters can be seen as making progress towards one or several of these visions, but, as you will see, there is much work to be done. At the same time, it is important to keep in mind that it is equally worthwhile to develop generators for more narrowly defined tasks.

1.5 Desirable properties of a PCG solution

We can think of implementations of PCG methods as *solutions* to *content generation problems*. A content generation problem might be to generate new grass with a low level of detail which does not look completely weird within 50 milliseconds. It might also be to generate a truly original idea for a game mechanic after days of computing time, or it might be to polish in-game items to a perfect sheen in a background thread as they are being edited by a designer. The desirable—or required—properties of a solution are different for each application. The only constant is that there are usually tradeoffs involved, e.g. between speed and quality, or expressivity/diversity and reliability. Here is a list of common desirable properties of PCG solutions:

- *Speed:* Requirements for speed vary wildly, from a maximum generation time of milliseconds to months, depending on (amongst other things) whether the content generation is done during gameplay or during development of the game.
- *Reliability:* Some generators shoot from the hip, whereas others are capable of guaranteeing that the content they generate satisfies some given quality criteria. This is more important for some types of content than others, for example a dungeon with no exit or entrance is a catastrophic failure, whereas a flower that looks a bit weird just looks a bit weird without this necessarily breaking the game.

- *Controllability:* There is frequently a need for content generators to be controllable in some sense, so that a human user or an algorithm (such as a player-adaptive mechanism) can specify some aspects of the content to be generated. There are many possible dimensions of control, e.g. one might ask for a smooth oblong rock, a car that can take sharp bends and has multiple colours, a level that induces a sense of mystery and rewards perfectionists, or a small ruleset where chance plays no part.
- *Expressivity and diversity:* There is often a need to generate a diverse set of content, to avoid the content looking like it's all minor variations on a tired theme. At an extreme of non-expressivity, consider a level "generator" that always outputs the same level but randomly changes the colour of a single stone in the middle of the level; at the other extreme, consider a "level" generator that assembles components completely randomly, yielding senseless and unplayable levels. Measuring expressivity is a non-trivial topic in its own right, and designing level generators that generate diverse content without compromising on quality is even less trivial.
- *Creativity and believability:* In most cases, we would like our content not to look like it has been designed by a procedural content generator. There is a number of ways in which generated content can look generated as opposed to human-created.

1.6 A taxonomy of PCG

With the variety of content generation problems and methods that are now available, it helps to have a structure that can highlight the differences and similarities between approaches. In the following, we introduce a revised version of the taxonomy of PCG that was originally presented by Togelius et al. [34]. It consists of a number of dimensions, where an individual method or solution should usually be thought of as lying somewhere on a continuum between the ends of that dimension.

1.6.1 Online versus offline

PCG techniques can be used to generate content online, as the player is playing the game, allowing the generation of endless variations, making the game infinitely replayable and opening the possibility of generating player-adapted content, or offline during the development of the game or before the start of a game session. The use of PCG for offline content generation is particularly useful when generating complex content such as environments and maps. An example of the use of online content generation can be found in the game *Left 4 Dead* [35], a recently released first-person shooter game that provides dynamic experience for each player by analysing

player behaviour on the fly and altering the game state accordingly using PCG techniques [3].

NERO [30] is an example of the use of AI techniques to allow the players to evolve real-time tactics for a squad of virtual soldiers. *Forza Motorsport* [17] is a car racing game where the Non-Player Characters (NPCs) can be trained offline to imitate the player's driving style and can later be used to drive on behalf of the player. Another important use of offline content generation is the creation and sharing of content. Some games such as *LittleBigPlanet* [16] and *Spore* [15] provide a content editor (level editor in the case of *LittleBigPlanet* and the *Spore* Creature Creator) that allows the players to edit and upload complete creatures or levels to a central online server where they can be downloaded and used by other players.

1.6.2 Necessary versus optional

PCG can be used to generate necessary game content that is required for the completion of a level, or it can be used to generate auxiliary content that can be discarded or exchanged for other content. The main distinctive feature between necessary and optional content is that necessary content should always be correct while this condition does not hold for optional content. An example of optional content is the generation of different types of weapons in first-person shooter games or the auxiliary reward items in Super Mario Bros. [21]. Necessary content can be the main structure of the levels in Super Mario Bros., or the collection of certain items required to pass to the next level.

1.6.3 Degree and dimensions of control

The generation of content by PCG can be controlled in different ways. The use of a random seed is one way to gain control over the generation space; another way is to use a set of parameters that control the content generation along a number of dimensions. Random seeds were used when generating the world in *Minecraft* [19], which means the same world can be regenerated if the same seed is used [18]. A vector of content features was used in [24] to generate levels for *Infinite Mario Bros.* [22] that satisfy a set of feature specifications.

1.6.4 Generic versus adaptive

Generic content generation refers to the paradigm of PCG where content is generated without taking player behaviour into account, as opposed to adaptive, personalised or player-centred content generation where player interaction with the game

Fig. 1.2: Three example weapons created in the *Galactic Arms Race* game for different players. Adapted from [11]

is analysed and content is created based on a player's previous behaviour. Most commercial games tackle PCG in a generic way, while adaptive PCG has been receiving increasing attention in academia recently. A recent extensive review of PCG for player-adaptive games can be found in [37].

Left 4 Dead [35] is an example of the use of adaptive PCG in a commercial game where an algorithm is used to adjust the pacing of the game on the fly based on the player's *emotional intensity*. In this case, adaptive PCG is used to adjust the difficulty of the game in order to keep the player engaged [3]. Adaptive content generation can also be used with another motive such as the generation of more content of the kind the player seems to like. This approach was followed in the *Galactic Arms Race* [11] game where the weapons presented to the player are evolved based on her previous weapon use and preferences. Figure 1.2 presents examples of evolved weapons for different players.

1.6.5 Stochastic versus deterministic

Deterministic PCG allows the regeneration of the same content given the same starting point and method parameters as opposed to stochastic PCG where recreating the same content is usually not possible. The regeneration of the galaxies in *Elite* [4] is an example of the deterministic use of PCG.

1.6.6 Constructive versus generate-and-test

In constructive PCG, the content is generated in one pass, as commonly done in roguelike games. Generate-and-test PCG techniques, on the other hand, alternate generating and testing in a loop, repeating until a satisfactory solution is generated. *Yavalath* [5] is a two-player board game generated completely by a computer program using the generate-and-test paradigm [7].

1.6.7 Automatic generation versus mixed authorship

Until recently, PCG has allowed limited input from game designers, who usually tweak the algorithm parameters to control and guide content generation while the main purpose of PCG remains the generation of infinite variations of playable content [39, 7, 1, 2]. However, a new interesting paradigm, has emerged that focuses on incorporating designer and/or player input through the design process. In this mixed-initiative paradigm, a human designer or player cooperates with the algorithm to generate the desired content.

Tanagra [29] is an example of a system where the designer draws part of a 2D level and a constraint satisfaction algorithm is used to generate the missing parts while retaining playability. Another example is the *SketchaWorld* framework [26], an interactive procedural sketching system for creating landscapes and cityscapes where designers can manually edit and tune the generated results while the virtual world model is kept consistent. *Ropossum* [23] is yet another recent example of the use of PCG for completing unfinished designs, suggesting modifications, handling constraints and testing for playability for the 2D physics-based game *Cut the Rope* [40].

1.7 Metaphors for PCG

In the phrase "procedural content generation system", we have discussed what the words "procedural", "content", and "generation" mean. But what about the word *system*? A *PCG system* is the generic term for any piece of software that does PCG. But these systems do different things, are used in different ways, and have quite different relationships to the overall game-design process. Some PCG systems try to help a designer out with a small part of the design process. Others try to provide a new way of working with game content. Some are interactive; others aren't. Some aim to do fully autonomous, creative game design; others aim to automate routine or common aspects of design.

To break this broad term, *PCG system*, into more specific kinds of systems, Khaled et al. [14] proposed four metaphors for thinking about how PCG systems relate to the game-design process. Some PCG systems are *tools*: instruments that give designers enhanced capabilities, in the way that a programmer's development environment or an architect's CAD system do. Others define new kinds of *materials*, allowing a designer to work in a new medium, the way stone, clay, and laser installations are different materials for an artist. Some PCG systems are intended to be *designers* themselves, carrying out fully autonomous design of parts or even entire games, rather than assisting game designers. Finally, some systems are primarily *domain experts*, carrying with them extensive knowledge of game design that can be used to critique or improve designs. Many systems can be viewed through more than one of these lenses, though few will exhibit all of them equally.

PCG *tools*, like non-PCG design tools, aim to improve a designer's workflow, but PCG tools do it by adding a generative component. A common example is a PCG-enhanced level editor. The level-editing tools included with many game engines already improve the level-design process by providing specialised ways of editing and laying out levels, rather than the designer having to do level design in a more generic tool, or entirely in code. A PCG-enhanced level editor adds a generative component to the traditional passive level editor. The *Tanagra* [29] level editor generates levels that fit a theory of rhythmic patterns in platformer games, which the designer can modify and add more constraints to, followed by re-generation of the relevant portions. This back-and-forth pattern, alternating procedural content generation and human editing, is called *mixed-initiative* generation, and is covered in Chapter 11. Among the visions for PCG discussed earlier in this chapter, "multi-level multi-content PCG" can be seen as using a tool metaphor.

PCG systems can also create new generative *materials* that a game designer manipulates and sculpts to produce content. A popular commercial example is *SpeedTree*. In one sense it's a tool for designing trees to place as scenery in videogames. But the way it does this is by turning trees into an interactive generative material: the designer can click and drag them around, add and remove branches, etc., and they always look like a tree, because the trees are procedurally generated in real time as the designer manipulates them. The fractal landscapes discussed in Chapter 4 are also a kind of procedurally generative material, which a designer manipulates to produce their desired landscapes. For the PCG vision of "PCG-based game design", the appropriate metaphor is material.

A procedural content *designer* has less interaction with the human designer, and instead has ambitions of designing content all on its own. In the limit case, a PCG designer turns into a fully autonomous game generator that creates new games, usually in a specific genre. Work on automatic game design is still at an exploratory stage, but promising prototype systems exist [20, 33, 7, 9]. A key challenge for a lead designer is that it must design not only the content *in* a game, but the rules of the game itself. Chapter 6 looks at these systems that generate rules and game mechanics. The PCG vision of "generating complete games" relies on a designer metaphor.

A procedural *domain expert* is a slightly different kind of system, full of knowledge about games or players, and able to apply it to critique and modify content. Often it will apply that expertise by being part of a system that also serves as a tool or a designer. A domain expert may have purely formal knowledge of games, such as what makes a particular set of rules elegant [6]. Or it may have extensive knowledge of human players, being able to predict what people will do in a game, and what they will find challenging, fun, or boring. For a PCG-based educational game, the domain expert may have pedagogical knowledge. For example, the procedural level generation in the fraction-teaching game *Refraction* is constrained so that generated levels meet the system's pedagogical goals [27]. Chapter 10 discusses the experience-driven PCG approach, which builds PCG systems that are experts in player behaviour and reactions.

1.8 Outline of the book

This book is structured as a series of chapters, co-written by the main authors of the book and the leading experts on the topic of each chapter. Most chapters are organised so that they introduce both a family of methods (e.g. fractals or grammars) and an application domain (e.g. plants or dungeons). The method is typically introduced through an example in the application domain, and the chapter then also discusses how the same method could be used for other domains or how different methods could be used for that domain. This structure is partly motivated by the interdisciplinary nature of PCG research and practice, where the algorithms used come from numerous different fields (and thus rarely build on each other) and game design knowledge is vital in all cases. Each chapter ends with a summary and typically also with a proposed lab exercise.

In Chapter 2 we present the search-based approach to procedural content generation, which is very versatile and which has recently been used in a large number of academic research projects as well as some released games. In the search-based approach, evolutionary algorithms are used to search for good game content using principles from Darwinian evolution. The two main challenges when building a search-based content generator are the evaluation function, which evaluates candidate content artefacts, and the content representation, which defines the search space for the algorithm. While this chapter contains several examples of content generators based on artificial evolution, there are further such examples scattered in the upcoming chapters.

Chapter 3 discusses the specific example of creating dungeons for roguelike games, and similar levels based on navigating a mostly two-dimensional space—for example, levels for platform games or first-person shooters. A number of fast and constructive algorithms for generating such levels are described. Some of these algorithms come from the game development community and are widely used in roguelikes such as *Diablo*. Others, such as cellular automata, have their origin in physics. We also describe the Mario AI framework, a common testbed for level generation algorithms based on a clone of *Super Mario Bros*.

Chapter 4 describes several algorithms with a background in computer graphics research, namely simple fractal algorithms and other noise algorithms. These are commonly used to produce terrains and complete landscapes, as well as textures and features such as clouds. While these algorithms are fast and reliable, they lack some forms of controllability. Therefore two other approaches to generating landscapes are presented, one search-based and one based on collections of agents.

Chapter 5 is about grammars. Grammars, common to computer science and linguistics, prove to be very useful for creating many types of game content. The chapter starts with the example of creating lifelike plants, which is a very common form of PCG; in fact, hundreds of AAA games from recent years feature procedurally generated vegetation based on grammars. But grammars can also be used for e.g. level generation; the rest of the chapter details how to use grammars for generating levels and missions for Zelda-style action-adventure games, and how to evolve grammars that generate *Super Mario Bros.* levels.

While some of the application domains of the previous chapters may be seen as somewhat peripheral, Chapter 6 addresses the problems of generating the absolutely most central part of any game: its rules. We describe a number of different attempts at generating rules for games, from board games to card games and arcade games. Some of these attempts are constructive, but most of them are search-based in one way or another. The chapter also describes the Video Game Description Language, a way of encoding game rules for simple arcade games of the kind you would find in the early 1980s—one of the purposes of this language is to enable automatic generation of complete games.

Most games feature stories of some kind, either backstories or interactive stories that the player can affect; stories can be seen as content, so Chapter 7 is devoted to the generation of game stories. It turns out that almost all methods of story generation are based on planning algorithms; planning is a classic AI method originally developed for robot control and now widely used in various domains. The chapter also discusses how story generation can be combined with map generation, so that game maps are generated that fit with the generated story.

Chapter 8 is focused on a single method, namely Answer Set Programming (ASP). This is a form of logic programming plus constraint satisfaction: a content generation method plus conditions are specified in a language called AnsProlog, and a solver produces all configurations of content that are compatible with the specified conditions. While this might seem rather abstract and mathematical, it has recently been demonstrated that certain PCG problems can be easily stated in AnsProlog form, and the results of the solver interpreted as game content. This yields a highly efficient method for creating some form of game content, for example levels for puzzle-like games.

Chapter 9 returns to the topic of Chapter 2, search-based PCG, and dwells on the question of how to represent the game content. Representation is important as it defines the shape of the search space and the ways in which it can be explored. This chapter demonstrates how a wise choice of representation can alter the style of the generated content as well as enable more effective search for content that better satisfies the evaluation function. Examples include flowers represented as neural networks and level generators represented as collections of agents.

One of the motivations for PCG is that it can enable player-adaptive games. Chapter 10 describes a framework for adapting games to the player, namely that of experience-driven PCG. We describe different methods for creating models of player experience based on data collected from players.

A theme throughout much of the book is that the relationship between procedural content generation and human game designers can be quite varied. PCG can be used in a highly automated way, but it can also be used in close coupling with the designer's own design choices. Chapter 11 looks at this close coupling explicitly, considering mixed-initiative systems, in which a human designer and a procedural content generation system collaborate to produce content.

Finally, Chapter 12 discusses how the quality of a PCG solution can be evaluated once it has been implemented.

1.9 Summary

Procedural content generation (PCG) in games is the algorithmic creation of game content with limited or indirect user input. PCG methods are developed and used for a number of different reasons, including saving development time and costs, increasing replayability, allowing for adaptive games, assisting designers and studying creativity and game design. While PCG algorithms have been used in some commercial games since the early 1980s, they are typically either used in a peripheral role or their scope is highly limited; current research in academia is trying to push the boundaries of what can be generated and with what quality it can be generated. Ideally, a PCG solution should be fast, reliable, controllable, expressive and creative, but in practice there are certain tradeoffs that need to be made between these properties. PCG solutions can be classified according to a relatively extensive taxonomy, which might help to identify their strengths and weaknesses. Another lens through which to understand a PCG system is the metaphor according to which it is used; here we can differentiate between using a system as tool, material, designer or domain expert. PCG algorithms are drawn from a variety of different fields, and this methodological diversity is evident from the table of contents of this book.

References

1. Adams, T.: (2006). Dwarf Fortress, Bay 12 Games
2. Blizzard North: (1997). Diablo, Blizzard Entertainment, Ubisoft and Electronic Arts
3. Booth, M.: The AI systems of Left 4 Dead. In: Keynote, Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE) (2009)
4. Braben, D., Bell, I.: (1984). Elite, Acornsoft, Firebird and Imagineer
5. Browne, C.: Yavalath (2007). URL http://www.cameronius.com/games/yavalath/
6. Browne, C.: Elegance in game design. IEEE Transactions on Computational Intelligence and AI in Games **4**(3), 229 –240 (2012)
7. Browne, C., Maire, F.: Evolutionary game design. IEEE Transactions on Computational Intelligence and AI in Games **2**(1), 1–16 (2010)
8. Compton, K., Osborn, J.C., Mateas, M.: Generative methods. In: Proceedings of the 4th Workshop on Procedural Content Generation in Games (2013)
9. Cook, M., Colton, S.: Multi-faceted evolution of simple arcade games. In: Proceedings of the 7th IEEE Conference on Computational Intelligence and Games (CIG), pp. 289–296 (2011)
10. Firaxis Games: (2005). Civilization IV, 2K Games & Aspyr
11. Hastings, E.J., Guha, R., Stanley, K.: Evolving content in the Galactic Arms race video game. In: Proceedings of the 5th IEEE Conference on Computational Intelligence and Games (CIG), pp. 241–248 (2009)
12. Hendrikx, M., Meijer, S., van der Velden, J., Iosup, A.: Procedural content generation for games: a survey. Transactions on Multimedia Computing, Communications and Applications **9**(1), 1 (2013)
13. Illiger, A.: (2011). Tiny Wings, Andreas Illiger
14. Khaled, R., Nelson, M.J., Barr, P.: Design metaphors for procedural content generation in games. In: Proceedings of the 2013 ACM SIGCHI Conference on Human Factors in Computing Systems, pp. 1509–1518 (2013)
15. Maxis: (2008). Spore, Electronic Arts

16. Media Molecule, SCE Cambridge Studio, Tarsier Studios, Double Eleven, XDev, United Front Games: (2008). Little Big Planet, Sony Computer Entertainment Europe
17. Microsoft Game Studios: (2005). Forza Motorsport, Microsoft
18. Minecraft Wiki: Minecraft wiki. URL http://www.minecraftwiki.net/
19. Mojang: (2011). Minecraft, Mojang and Microsoft Studios
20. Nelson, M.J., Mateas, M.: Towards automated game design. In: AI*IA 2007: Artificial Intelligence and Human-Oriented Computing, pp. 626–637. Springer (2007). Lecture Notes in Computer Science 4733
21. Nintendo Creative Department: (1985). Super Mario Bros., Nintendo
22. Persson, M.: Infinite Mario Bros. URL http://www.mojang.com/notch/mario/
23. Shaker, M., Shaker, N., Togelius, J.: Ropossum: An authoring tool for designing, optimizing and solving Cut the Rope levels. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. AAAI Press, pp. 215 – 216 (2013)
24. Shaker, N., Togelius, J., Yannakakis, G.N.: Towards automatic personalized content generation for platform games. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), pp. 63–68 (2010)
25. Smelik, R., De Kraker, K., Tutenel, T., Bidarra, R., Groenewegen, S.: A survey of procedural methods for terrain modelling. In: Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS) (2009)
26. Smelik, R., Tutenel, T., de Kraker, K., Bidarra, R.: Integrating procedural generation and manual editing of virtual worlds. In: Proceedings of the Workshop on Procedural Content Generation in Games. ACM (2010)
27. Smith, A.M., Andersen, E., Mateas, M., Popović, Z.: A case study of expressively constrainable level design automation tools for a puzzle game. In: Proceedings of the 7th International Conference on the Foundations of Digital Games, pp. 156–163 (2012)
28. Smith, G., Othenin-Girard, A., Whitehead, J., Wardrip-Fruin, N.: PCG-based game design: creating Endless Web. In: Proceedings of the International Conference on the Foundations of Digital Games, pp. 188–195. ACM (2012)
29. Smith, G., Whitehead, J., Mateas, M.: Tanagra: A mixed-initiative level design tool. In: Proceedings of the Fifth International Conference on the Foundations of Digital Games, pp. 209–216. ACM (2010)
30. Stanley, K., Bryant, B., Miikkulainen, R.: Real-time neuroevolution in the NERO video game. IEEE Transactions on Evolutionary Computation **9**(6), 653–668 (2005)
31. Togelius, J., Champandard, A.J., Lanzi, P.L., Mateas, M., Paiva, A., Preuss, M., Stanley, K.O.: Procedural content generation: Goals, challenges and actionable steps. In: S.M. Lucas, M. Mateas, M. Preuss, P. Spronck, J. Togelius (eds.) Dagstuhl Seminar 12191: Artificial and Computational Intelligence in Games, pp. 61–75. Dagstuhl (2013)
32. Togelius, J., Kastbjerg, E., Schedl, D., Yannakakis, G.N.: What is procedural content generation?: Mario on the borderline. In: Proceedings of the 2nd Workshop on Procedural Content Generation in Games (2011)
33. Togelius, J., Schmidhuber, J.: An experiment in automatic game design. In: Proceedings of the 4th IEEE Symposium on Computational Intelligence and Games (CIG), pp. 111–118 (2008)
34. Togelius, J., Yannakakis, G.N., Stanley, K., Browne, C.: Search-based procedural content generation. Applications of Evolutionary Computation pp. 141–150 (2010)
35. Valve Corporation: (2008). Left 4 Dead, Valve Corporation
36. Wittgenstein, L.: Philosophical Investigations. Blackwell (1953)
37. Yannakakis, G.N., Togelius, J.: Experience-driven procedural content generation. IEEE Transactions on Affective Computing **2**(3), 147–161 (2011)
38. Yu, D.: Spelunky. Boss Fight Books (2016)
39. Yu, D., Hull, A.: (2008). Spelunky, Independent
40. ZeptoLab: (2010). Cut the Rope