Procedural Content Generation: Goals, Challenges and Actionable Steps

Julian Togelius[1], Alex J. Champandard[2], Pier Luca Lanzi[3], Michael Mateas[4], Ana Paiva[5], Mike Preuss[6], and Kenneth O. Stanley[7]

1 **Center for Computer Games Research, IT University of Copenhagen, Copenhagen, Denmark**
julian@togelius.com
2 **AiGameDev.com KG, Vienna, Austria**
alexjc@aigamedev.com
3 **Department of Electronics and Information, Politecnico di Milano, Milano, Italy**
lanzi@elet.polimi.it
4 **Center for Games and Playable Media, University of California, Santa Cruz, California, USA**
michaelm@cs.ucsc.edu
5 **Intelligent Agents and Synthetic Characters Group, INESC-ID, Lisboa, Portugal**
ana.paive@inesc-id.pt
6 **Department of Computer Science, Technical University of Dortmund, Dortmund, Germany**
mike.preuss@cs.tu-dortmund.de
7 **Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, USA**
kstanley@eecs.ucf.edu

Abstract

This chapter discusses the challenges and opportunities of procedural content generation (PCG) in games. It starts with defining three grand goals of PCG, namely multi-level multi-content PCG, PCG-based game design and generating complete games. The way these goals are defined, they are not feasible with current technology. Therefore we identify nine challenges for PCG research. Work towards meeting these challenges is likely to take us closer to realising the three grand goals. In order to help researchers get started, we also identify five actionable steps, which PCG researchers could get started working on immediately.

1998 ACM Subject Classification I.2.1 Applications and Expert Systems: Games

Keywords and phrases procedural content generation, video games

Digital Object Identifier 10.4230/DFU.Vol6.12191.61

1 Introduction

Procedural content generation (PCG) refers to the algorithmic generation of game content with limited or no human contribution. "Game content" is here understood widely as including e.g. levels, maps, quests, textures, characters, vegetation, rules, dynamics and structures, but not the game engine itself nor the behaviour of NPCs. PCG has been part of published games since the early eighties, with landmark early examples being the runtime

Artificial and Computational Intelligence in Games. *Dagstuhl Follow-Ups*, Volume 6, ISBN 978-3-939897-62-0.
Editors: Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius; pp. 61–75

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

generation of dungeons in *Rogue* and the extremely compressed representation of hundreds of star systems in *Elite*. Prominent recent examples of PCG include the ubiquitous *SpeedTree* system, used for generation of trees, grass and other types of vegetation in hundreds of commercial games, and the generation of dungeons, weapons and items in the *Diablo* series of games. PCG can be used for a variety of reasons, including providing variety, reducing development time and development costs, saving space in transmission or on disk, augmenting human creativity and enabling adaptivity in games.

The academic research community around PCG has formed only within the last few years. While a number of important papers were published in venues dedicated to AI or graphics research, the first workshop devoted entirely to PCG took place in 2009 and the first journal special issue devoted to the topic came out only in 2011[1]. The solutions to content generation problems that have been explored by academic researchers have tended to be focused more on adaptable and controllable (and sometimes highly complex) algorithms, whereas the types of algorithms that have so far been used in published games tend to be simpler, faster and less controllable.

The workgroup on PCG decided to look at the field from the perspective of what research would be most important to do in the future in order to ensure that real breakthroughs are made, and modern PCG techniques will be able to add value to the games industry by enabling new types of games as well as new ways of developing games. We took a top-down perspective and started with thinking about what sort of things we would ultimately want PCG to be able to achieve; the grand *goals* of the research field. We then tried to identify the most important *challenges* that need to be overcome in order to be able to realise these goals, and finally identified a handful of *actionable steps* that could make progress towards overcoming these challenges, and which could be taken already today by someone – perhaps you? – interested in contributing to the field.

2 Goals

The following is a list of what we consider the most important goals of PCG research. In each case, the goal is currently not obtainable and it would require significant further research effort leading to some sort of breakthrough in order to be able to realise the goal. However, in each case there is already some work done that directly addresses this goal, so for each item we list what we consider the most relevant previous work.

2.1 Multi-level Multi-content PCG

Imagine being able to push a button and generating a complete game world: terrain, vegetation, roads, cities, people, creatures, quests, lore, dialogue, items, vehicles; polygons, graphs, textures, text. For example of such a fully fledged game world, think of the *Elder Scrolls V: Skyrim* (Bethesda) game world, complete with all that which makes it so immersive and exciting – except for the underlying game engine. And then press the same button again, and you get a fresh new world, different from the previous world in every respect: the details, the overall structure, the look and feel. Maybe it's a sci-fi world threatened by invasion, or a murder mystery in a contemporary industrial city. The only limits for the expressive space

[1] The PCG workshop runs annually since 2010, co-located with the Foundations of Digital Games Conference. The autumn 2011 issue of IEEE Transaction on Computational Intelligence and AI in Games is entirely devoted to PCG. A discussion group for the PCG community can be found at `https://groups.google.com/group/proceduralcontent/`

are the capabilities of the underlying game engine, which provides primitives for movement, social interactions, combat, scoring etc. The system we imagine should be so flexible so that it would be possible to plug in a new game engine, complete with a new set of rules and dynamics, and go on to generate complete game worlds that fit with that engine.

In other words we are here envisioning a system that can generate multiple types of quality content at multiple levels of granularity in a coherent fashion while taking game design constraints into consideration. Nothing like this exists yet, and while it might not be possible to achieve such a system in the foreseeable future, we see much room for progress in this direction.

Almost all existing approaches to content generation generate a single type of content for a single game, and the results all too often look uninspired and generic. There are some interesting examples of trying to generate several types of content so that they fit in with each other. *Dwarf Fortress* (Bay 12 Games) generates many aspects of the game world, including its geology and backstory, all the way down to the socks on each foot of individual dwarfs. However, for each type of content the generation process is very simple, and the generated game worlds show little variation. A similar problem is tackled by Hartsook et al., though that paper employs a "waterfall" model where one type of content (stories) are generated first and the second (maps) afterwards, with no feedback between process [9]. Another approach to multi-level multi-content generation is that of the *Sketchaworld* system by Smelik et al. [23]. This system allows for mixed initiative generation of various aspects of a landscape, including topology, vegetation, and the placement of roads and buildings. All objects have semantics and defined interdependencies, which together with a conflict resolution system allows editing on one level (e.g. changing the flow of a river) to have effects on entities on other levels (e.g. a bridge is automatically created over the river to allow a pre-existing road to pass through). However, that system does not address game-specific design considerations such as balance or challenge.

2.2 PCG-based Game Design

Imagine a game where procedural content generation was a central mechanic, without which the game could not exist at all; in fact, the whole genre to which the game belongs could not exist without procedural content generation. Imagine that the game was truly endless, and that exploration of an infinite range of content was a central part of the gameplay, indeed the main reason the game was so appealing.

Almost all existing approaches to PCG focus on generating content for an existing game, where the core game itself could exist without the PCG mechanism. PCG is done to facilitate development, or to allow for runtime adaptation. Even in games such as *Rogue*, *Spelunky* and *Diablo*, where a key feature of the game is the endless variation in game content, all of the levels could in principle have been generated offline and presented to the player without taking player choice into account. Creating games where a PCG algorithm is an essential part of the game design requires innovations both in game design, where parameters to the PCG algorithm need to be meaningfully based on player actions, and in PCG, where the algorithm needs to be reliable and controllable beyond the capacities of most current algorithms.

A few games have made progress towards realising this vision. *Galactic Arms Races* and *Petalz* both feature the evolution of new content as a core part of the game, where the players respectively evolve weapons to defeat enemies or evolve flowers to impress friends as the game is being played [10, 19]. *Endless web* is a platform game that lets the player explore different dimensions of content space, and generates new levels in response to the

player's actions[26]. In *Infinite Tower Defence*, the role of PCG is to create new levels and enemies that match the current strategy of the player so as to force the player to explore new strategies [2]. Another example is *Inside a Star-filled Sky*, which features a "zooming" mechanic, where the player character can enter enemies and explore new levels inside them; this yields an apparently endless hierarchy of nested levels. While these games are addressing the challenge of PCG-based game design, they are still variations on well-known genres rather than examples of genres that could only exist because of PCG.

2.3 Generating Complete Games

Imagine a PCG system that could create complete games. Not just content for an existing game, but the whole game from scratch, including the rules and game engine. At the press of a button, starting from nothing, the system will create a game no-one has played before and which is actually enjoyable to play. This would involve automating the arguably most central and "AI-complete" aspects of game design, including estimating how a human player would experience interacting with a complete system of rules and affordances. Perhaps the system accepts parameters in the form of a design specification for the game. For the example, a human might task the system with designing a game that features fog of war, that promotes collaboration or that teaches multiplication.

Several attempts have been made to generate game rules, sometimes in combination with other aspects of the game such as the board. Of particular note is Cameron Browne's *Ludi* system which generated a board game of sufficient novelty to be sold as a boxed product through searching through a strictly constrained space of board games [4]. This system built on evolutionary computation, as did earlier [32] and later [6] attempts to evolve simple arcade-style games. Other attempts have build on symbolic techniques such as logic programming [25, 34]. While these examples have proven that generation of playable game rules is at all possible, they all generate only simple games of limited novelty.

3 Challenges

Analysing what would be needed in order to reach the grand goals discussed above, the workgroup arrived at a list of eight research challenges for procedural content generation. Successfully meeting any of these challenges would advance the state of the art in PCG significantly, and meeting all of them would probably render the goals described above attainable. Addressing any of these challenges could make a good topic for a PhD thesis.

3.1 Non-generic, Original Content

Generated content generally looks generic. For example, looking at the dungeons generated for a roguelike game such as those in the *Diablo* series, you easily get the sense that this is just a bunch of building blocks hastily thrown together with little finesse – which is just what it is. Re-generate the level and you get a superficially very different but ultimately equally bland level. Most generated levels lack meaningful macro-structure and a sense of progression and purpose. Very rarely would you see a generated level about which you could say that it was evidence of skill or artfulness in its creator, and even more rarely one which showcases genuine design innovation. Compare this with the often masterfully designed levels in comparable games such as those in the *Zelda* franchise. The Zelda levels are aesthetically pleasing in several ways, providing a clear sense of place and progression, and often offering some original and unique take on the design problems of action adventure games.

There are a few examples of PCG systems having come up with what could be called genuine inventions. In the *Galactic Arms Race* game, several of the weapons that were generated (such as the tunnel-maker and hurricane) were surprising to players and designer alike and unlike anything the designers had seen before, yet were effective in the game and opened up for new playing styles, as if they had been designed by a human designer [10]. As discussed above, Browne's *Ludi* system managed to come up with a game (*Yavalath*) that was sufficiently novel to be sold as a boxed product. However, we do not know of a system that has exhibited sustained creativity, or that (in the language of Margaret Boden [3]) has displayed transformational rather than just exploratory creativity.

The challenge, then, is to create content generators that can generate content that is purposeful, coherent, original and creative. This challenge is quite broad, as interpretations of these adjectives could vary – this only means that there are many different ways of approaching the challenge.

3.2 Representing Style

Directly connected to the previous challenge but somewhat more specific is the challenge to create a content generator that can create content in a particular style that it has somehow learned or inferred. Human artists of various kinds can observe artefacts produced by another artist, and learn to imitate the style of that artist when producing new artefacts – e.g., a skilful painter could study a number of Picasso paintings and then produce new paintings that were recognisably in the same style (while presumably not exhibiting the same creativity), and then go to on to study Mondrian paintings and produce Mondrian-like paintings of her own. An analogous capacity in PCG could be a level generation system that could study the seminal level designs of Shigeru Miyamoto in the *Zelda* and *Super Mario* series, and produce similar designs automatically; the same generator could, after being presented with John Romero's significantly different designs for *Doom* levels, learn to imitate that style too. Perhaps the generator could be presented with artefacts that were not game levels, for example architectural designs by Frank Lloyd Wright, and learn to reproduce that style within the constrained design space of levels for a particular game. Similarly competent texture generators, game rule generators and and character generators could also be imagined. It is important to note that the challenge is not only to imitate the surface properties of a set of designs (such as recurring colours and ornamentation) but also the deeper features of the design, having to do with expression of ideas and emotions in interplay with the player and game design.

Some attempts have been made to model player preferences in level generators [16] or designer preferences in interactive evolution for content generation [15]. However, the preferences modelled in these experiments are still very indirect, and the generated artefacts still exhibit the style of the generator more than that of the player or designer.

3.3 General Content Generators

Almost all existing PCG algorithms generate a single type of content for a single game. Reusability of developed PCG systems is very limited; there is no plug-and-play content generation available. This can be contrasted with the situation for other types of game technology, where game engines are regularly reused between games and even some aspects of game AI (such as pathfinding) now being available as middleware. The only PCG middleware that is actually in use in multiple games is *SpeedTree*, but this again generates only a single type of content (vegetation) and that type is of little functional significance in most games,

meaning that the risks of generating content are rather low; ugly shrubbery is ugly but tends not to break the level. The lack of readily available PCG systems that could be used without further development to generate for example levels or characters for a new game is probably holding back adoption of PCG techniques in the game industry.

It is already the case that certain techniques underly a number of different PCG systems. For example, L-systems [18] are at the core of both *SpeedTree* and techniques for creating landscapes [1] and levels [7]. Similarly, compositional pattern-producing networks (CPPNs) [29] are the basis for both the spaceships in Galactic Arms Race [10], the flowers in Petalz [19] and the pictures in PicBreeder [20]. However, in each case significant engineering effort was required to make the method work with the particular type of content in the particular game.

A general content generator would be able to generate multiple types of content for multiple games. The specific demands, in term of aesthetics and/or in-game functionality, of the content should be specified as parameters to the generator. A game designer armed with such a tool would just need to properly specify the requirements for the content that should be part of a new game in order to promptly have a means of automatically generating content for it. One idea for how to achieve this is to treat PCG algorithms as content themselves, and generate them using other PCG methods so as to fit the particular content domain or game they are meant to be applied to [14].

3.4 Search Space Construction

If content is to be generated then it must be be situated within a search space. The structure of the search space determines what content can be reached from small perturbations of any particular instance in the search space. For a content generator to be successful, the structure of the search space needs to have certain forms of locality. In general, small perturbations in the underlying representation should not lead to radical changes in the appearance or functionality of the content itself. For example, a table should not turn into a mushroom in a single small mutation. Rather, the table might become a little shorter, a little taller, or a little rounder, but it would remain a recognisable table.

This search space structure is ultimately determined by the selected underlying representation for the class of content being generated. Existing representations include the L-sytems [18] and CPPNs [29] discussed in the previous section, as well as logic-based representations such AnsProlog code snippets [24] and more direct representations where individual numbers correspond to individual features of the artefact. Such representations bias the generator towards producing certain qualitative themes, such as fractals in L-systems or symmetry in CPPNs. These biases are a reflection of the structure of the search space induced by the representations – fractals tend to be reachable from many different parts of a search space induced by L-systems. Thus the representation becomes an implicit means of structuring which types of content neighbour which, and therefore which artefacts can lead to which others.

This relationship between representation and search space structure highlights the significant challenge of designing a generator for a specific type of content: If a type of content is to be generated – say vehicles or buildings – then the engineers designing the generator must carefully construct a representation that induces a reasonable structure on the search space. One would not want airplanes to change in one step into wheelbarrows. To ensure such a smooth a tightly coupled landscape, the designer must intimately understand the relationship between the underlying representation and the structure of the space it induces, a relation which is not necessarily intuitive. For this reason, designing the search space to

make a satisfying generator requires significant skill and insight, and there are currently few general principles known to the research community for undertaking such a task.

In the future, it is possible that tools can be built to aid in adapting a particular representation for a particular class of content. For example, as noted in the previous section, CPPNs have encoded pictures, weapon systems, and flowers by interpreting their outputs and choosing their inputs carefully. It would be most helpful if a tool could help to automate such choices so that a particular representation could be adapted to a particular content class more easily.

3.5 Interfaces and Controllability for PCG Systems

Most existing PCG systems are not easy for a human to interface with and control. Many classic PCG implementations, for example the dungeon generators in many roguelike games, have no parameters of control at all; taking a random seed as input, a level is generated as output. In very many cases, you as a user (or as a game) would need to have more control of the generated artefact. Like controlling how difficult a level is, whether it should be more suited to speed runners or to explorer-type players, how much treasure and how many puzzles it should contain, and whether it should include a green flagpole at a particular location or perhaps five pixels to the left of that position. Or the age of a generated flower, the intuitiveness of a ruleset or the hipness of a car. There are many possible types of control that could be desirable, depending on the game and the designer.

Some classic constructive algorithms such as L-systems offer ways for the designer to specify aspects of the generated content, such as the "bushiness" of a plant. Search-based approaches allow the designer to specify desirable properties of the content in the form of objectives, but encoding the desired qualities in a fitness function is often far from straightforward and there is no guarantee that content with high values on these objectives can be found in the search space. Other PCG paradigms such as solver-based PCG using e.g. Answer Set Programming [24] offer complementary ways of specifying objectives, but again, it is not easy to encode the desired qualities for a non-expert. The mixed-initiative PCG systems Sketchaworld [23] and Tanagra [27] explicitly address this problem by allowing the user to interact with the PCG system by moving and creating objects in physical space, and thus imposing constraints on how what can be generated where. These systems clearly show a viable way forward, but so far only some aspects of control has been achieved (physical location) at the cost of some limitations in what sort of underlying PCG methods can be used.

What would it mean to allow users (be they designers or players, or perhaps some other algorithm such as a challenge balancing system) complete control over the content generation algorithm? Presumably it would mean that they could at any point during the generation process change any aspect of the content: making the level more blue or less scary or just making all of the gaps except the fifth one contain apples. Then the generator responds by implementing the changes, perhaps introducing new features or removing others, but still respecting what the user has specified wherever possible, and intelligently resolving conflict between specifications (e.g. the apples in the gaps could make the level more difficult). One could imagine something like *Adobe Photoshop*'s extreme array of expressive tools, including brushes, filters and abilities to only have modifications apply to particular layers, but all the way taking game design into account and autonomously generating appropriate content suggestions. It should be emphasised that this is not only a formidable challenge for PCG algorithms, but also for human-computer interaction. It is not even obvious how to represent aspects of game content that depend on being played to be experienced (such as game rules)

in an editor; a recent attempt at mixed-initiative generation of game rules mostly highlighted the problem [30].

3.6 Interaction and Opportunistic Control Flow Between Generators

Closely related to the previous challenge, and crucial for the goals of multi-level PCG and generating complete games, is the challenge to devise workable methods for communication and collaboration between algorithms working on generating different aspects or layers of the same artefact. For example, when a system generates the rules for a game, the physical environments for the same game and the characters or creatures that feature in it, the various generative algorithms must be able to communicate with each other. The simplest way of implementing this would probably be a "waterfall" model where the rules are generated first, positing requirements on the terrain/levels generator, in turn further constraining the space available for the creature/character generators. But this rules out any innovations in rules which are dependent on, and initiated by, uncommon solutions and crazy ideas in level or character design. In fact, as the rule generator cannot know what sort of characters the character generator will be able to produce (unless the latter's search space is severely constrained), the rules will have to be constrained to be very bland and workable with pretty much any characters. For these reasons, games developed by teams of humans are often developed in a much more opportunistic way, where opportunities or problems discovered at any content layer could affect the design of the other layers (e.g. the invention of a new type of enemy spurs the invention of new rules).

How can we replicate such an opportunistic control flow in a completely (or mostly) algorithmic environments, where algorithms (or perhaps some algorithms and some humans) collaborate with each other? One could imagine a system where constraints are posted in a global space, but this requires that a language and/or ontology be constructed to make such constraints comprehensible across generators, and also that a system is devised for working out priorities and solving conflicts between constraints. Going further, one could imagine equipping the content generators with models of themselves so as to provide a level of introspection, allowing them to exchange models of the bounds of their generative spaces.

3.7 Overcoming the Animation Bottleneck

In modern 3D computer games, animation is a major concern. Almost everything needs to be animated: creatures, characters, vehicles and features of the natural world such as vegetation and water. Creating believable animations even if the underlying characters are not procedurally generated is a huge challenge. In particular:

- Motion capture or hand animation is very expensive to acquire, and requires either the use of specialised motion capture facilities or an extensive team of animators.
- Data-heavy forms of animation such as motion capture or hand-animation also costs a significant amount of time, and are often a bottleneck for improving character behaviour.
- Animation systems based on data require significant runtime overheads for shifting around the data, decompressing it and generating runtime poses via blending.

This makes current animation techniques a bottleneck in three different ways, each as important as the other. Procedural techniques are already starting to resolve each of these three different issues, and the games industry is highly interested in the results [5].

There are many domain-specific problems to generating compelling animations for characters that were hand defined, but harder still is the problem of animating procedurally

generated creatures. Being able to generate an artefact does not mean that one automatically is able to animate it, and if one is not able to convincingly animate an artefact it is more or less useless in-game, as it would break the suspension of disbelief.

The big challenge of procedural animation is to match the high expectations of human observers, without having to resort to stylisation as a solution. This solution will involve subtle combinations of data and code that are crafted and assembled together masterfully by skilled technical animators, and new algorithms to make this possible.

3.8 Integrating Music and Other Types of Content

While most computer games feature music, the whole audio component usually only serves the task of supporting the game flow or emphasising the general atmosphere of the game. This is often done either by producing complete sound tracks as in the movie industry or by designing a very simple generative system (as in the Google app *Entanglement*) that repeatedly recombines single parts in order to stretch the available music and prevent boredom. Games that actively use the music as source of information to create game content or, vice versa, use game events for adjusting or even creating music are still rare.

In many well-known games based on music (e.g. Guitar Hero or SingStar), the interaction between music and game events is completely fixed and has been manually created. An overview of the different possibilities is given in [17]. Some examples of more complex interaction are:

- Rez (Sega)[2] from 2001, a rail shooter that tightly binds the visual impression (and appearance of enemies) to the composed music and also feedbacks user actions acoustically.
- Electroplankton by Nintendo[3] from 2005, where the player interacts in various ways with the game to create audio and visual experiences.
- The turntable-like game Scratch-Off [8] that requires user interaction matching the rhythm of the music while blending over to another piece of music.
- The Impossible Game is a graphically simple but challenging platform game for consoles and mobile phones that requires the user to cope with very different obstacles that are generated in accordance with current music events.
- The Bit.Trip series by Gaijin Games features the main character Commander Video who has to cope with game events mostly based on the rhythm of the played music in 6 very different games (rhythm shooter, platformer).
- Wii Music automatically generates melodies based on how the player moves the controller.
- BeatTheBeat [12] features three mini-games (rhythm-based tap game, shooter, and tower defence) that each use the available music as background and its analysed feature events as source for game event creation.

Music informatics has made great strides in recent years, including the emergence personalised music selection systems (including learning personal preferences) as well as complex music generation systems. Given this progress, it should be possible to establish game/music systems that interact in a much more complex way than what we are used to see in most games. In principle, one could take an existing simple game and attach it to a recommendation or generating system. Some thought has to be put into designing clever interfaces, but the problem appears per se as solvable. Using existing music as input source

[2] http://www.sonicteam.com/rez
[3] http://www.mobygames.com/game/electroplankton

for realtime production of game content just becomes possible now as modern computers are powerful enough to analyse the music online. This scheme can enhance a game considerably because the user can modify it by selecting the music that best reflects the current mood or just some very different music to create a new experience.

However, this strategy is only applicable for simple games with a restricted number of (partly repetitive) game components. For more complex games, interaction in the other direction (game events influence the music played) may make more sense. The components to achieve this are available, simple forms of modulation would be changing volume, speed, or key of the music. One could also think of online remixing by changing the volume of single instrument tracks, an approach that has been tried some years ago by several musicians by means of online tools (e.g. Peter Gabriel, William Orbit), but not in the games context. The feasibility of this approach highly depends on access to the single music and games components, but technically appears to be rather simple. A more sophisticated approach would apply a music generating system in order to modify or re-create the currently played music. However, for achieving a believable connection between game events and music, the semantic of the game events needs to be defined in a transferable fashion, for example through their effects on the player's mood.

3.9 Theory and Taxonomy of PCG Systems

While PCG research has been steadily increasing in volume in the last few years, there has been a lack of unifying theory to guide the research or even to help relating the disparate contributions to each other. New algorithms have been proposed and case studies carried out in a number of game domains, but it is seldom clear in what ways a new approach is better (or worse, or just different) to existing approaches. A theory and taxonomy of PCG would 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. It would also help situate and analyse any new algorithms proposed. A recent paper makes an initial attempt to provide a taxonomy for PCG, but covers mostly search-based approaches, in particular those based on evolutionary computation [33].

4 Actionable Steps

The challenges listed above are somewhat abstract and are mostly long-term projects. For some of them, it is not even clear how to approach them. To make matters more concrete, and to provide a set of example projects for anyone wishing to contribute to advancing the state of the art of procedural content generation, we devised a number of *actionable steps*. These are more specific research questions around which projects with a limited scope could be formulated, and for which the technical prerequisites (in terms of algorithms and benchmarks) already exist. You could start working on any of these steps already today.

4.1 Atari 2600 Games

Inventing a system that could generate a complete game with the complexity and scope of a modern AAA game might be a tall task – after all, developing these games often takes hundreds of person-years. Fortunately, not all games are equally complex. Those games that were made for early computers and game consoles were substantially less complex, as available memory size, processing speed, graphics resolution and development teams were all fractions of what they are today. The *Atari 2600* games console, released in 1977, had

4 kilobytes of RAM, a 1.2 MHz processor and was home to classic games such as *Pitfall*, *Breakout*, *Adventure* and *Pac-Man*. All of them had two-dimensional graphics that adhered to certain constraints regarding e.g. the number of movable objects that were dictated by the system's hardware design.

One could realistically try to directly approach the third of the grand goals outlined above, that of generating complete games, working within the much constrained space of games that would be possible on a system such as the Atari 2600. The limited space makes it much more tractable to search for good games, regardless of which search-mechanism would be used. The limited computational requirements of running these games would also make it possible to test them, for example by simulating playthroughs, at a much faster pace than real-time. A successful game generation system for Atari 2600 games should be able to re-invent classic games such as *Breakout* and *Pac-Man*, but also to invent enjoyable games that have never before been seen and which differ in some interesting way from all existing games. Developing such a system would require devising a description language for this type of video games, that is complete enough to cover essentially all interesting such games, but which still enables the space to be effectively searched; for some thoughts on how this could be done, please see the chapter *Towards a Video Game Description Language* in this volume.

Relates directly to challenges: general content generators, search space construction, interaction and opportunistic control flow.

4.2 Procedural Animation for Generated Creatures

One way of approaching the challenge of bringing PCG and procedural animation together is to develop a creature generator which generates creatures together with suitable procedural animation. The most impressive attempt to do something similar is probably the Creature Creator in *Spore*, but that is an editor rather than an automatic content generator, and imposes certain constraints on the generated creatures that should be avoided in order to be able to search a more complete creature space.

Relates directly to challenges: overcoming the animation bottleneck, interfaces for PCG systems.

4.3 Quests and Maps

The computational generation and adaptation of narrative is a specific type of PCG which enjoys its own research field, and a number of promising approaches have been presented, most of them based on planning algorithms [35]. Another domain of PCG which has seen much attention is the generation of maps of different kinds. Maps are automatically generated in games such as *Civilization* and *Diablo*, and a number of recent studies have investigated techniques such as cellular automata [11], grammars [7, 1] and evolutionary computation [31] for generating interesting and/or balanced maps for different types of games.

Generating maps and quests together could potentially be an excellent showcase for multilevel PCG (the first of the grand goals outlined above), as the best-designed games often feature quests and maps that interact and reinforce each other in clever ways – the map helping the game tell the story, and the story helping the player explore the map – and there are workable approaches to generating each type of content separately. However, there is surprisingly little work done on generating quests and maps *together*. One of the few examples is Dorman's use of grammars for generating Zelda-style dungeons and quests together, achieving good results by severely limiting the domain [7]. Another is Hartsook et

al.'s generation of matching maps and quests, by simply generating the map after the quest so that the former fits with the latter [9].

There is plenty of scope for taking this further and generating maps and quests that are perfect fits for each other. The interested investigator could start by taking one of those algorithms that has been proven to work well for one domain (maps or quests) and try to encode the other domain within the first, or by devising a scheme where map and quest generators take turns to respond to each other, or perhaps by trying to invent a completely new algorithm for this task. One could also try to allow human intervention and input at any phase of the quest/map generation.

Relates directly to challenges: interaction and opportunistic control flow, general content generators, interfaces for PCG systems.

4.4 Competent Mario Levels

The *Mario AI Benchmark* is a popular testbed within the CI/AI in games community, based on an open source clone of Nintendo's classic platformer *Super Mario Bros.* The benchmark has been used for a series of competitions focused on developing AI controllers that play the game proficiently [13], but in 2010 and 2012 it was also used for a competition where entrants submitted level generators capable of generating personalised levels for particular players [22]. A number of PCG systems were submitted to this competition, and a number of other PCG experiments using the Mario AI Benchmark have been published following the competition [21, 28].

However, comparing the quality of the generated levels with those that can be found in the real Super Mario Bros game, or with human-designed levels in any other high-quality platformer, makes any PCG aficionado disappointed. The generated levels typically lack a sense of progression, or any other macro-structure for that matter. Unlike the real Super Mario Bros levels, there is no sense that they are telling a story, trying to teach the player a skill, or hiding a surprise. Furthermore, the generated levels frequently feature items and structures that make no sense, unexplainable difficulty spikes and repeated structures that seem to be taken straight from a structure library. A high priority for someone interested in procedurally generating platform levels should be to devise an algorithm that can create levels with a sense of purpose. Using the Mario AI Benchmark as a testbed means that there is no shortage of material for comparisons, both in the form of level generators and in the form of professionally designed levels.

Relates directly to challenges: non-generic content, representing style.

4.5 Player-directed Generation with Model-based Selection

A final intriguing possibility is that player-directed generation in the style of Galactic Arms Race [10] could be enhanced by combining it with model-based selection such as in [16]. In a game like Galactic Arms Race, the game generates new content based on content players have liked in the past (as evidenced by e.g. using it). This idea works to ensure that new content appearing in the world derives from content that players appreciated.

However, as the number of players climbs higher, the amount of content generated will also climb because player behaviour generally leads to new content spawning in the world. With a relatively small population of players, this dynamic poses few problems because the probability of any player in the game eventually experiencing a reasonable sampling of the diversity of generated content is high. However, with many players, the consequent content explosion means that most players will see only a small fraction of the diversity of

content that the game is able to produce. In that situation, the question arises whether the overall search for content might benefit from trying to expose players to content that they are likely to find interesting. That is, the game might try to *model* the type of content that individual players prefer and thereby avoid wasting effort presenting newly-generated instances to players who are unlikely to be interested in them. If such mismatches occur on a large scale, then pockets of the search space that could have been explored fruitfully might be abandoned simply because the players who would have been interested in such content never had to the opportunity to observe it.

By combining player modelling with player-directed content generation, it is a possible that a synergistic effect could accelerate the search and also produce a more diverse set of content. When players are exposed to candidate content that they are likely to find interesting, their discernment in principle can help to explore the subspace of that particular type of content with more fidelity than would be possible through the overall player population.

Relates directly to challenges: representing style, search space construction.

5 Conclusion

This chapter presents three grand goals for procedural content generation, and presents several challenges that should be addressed in order to realise these goals, and a sample of actionable steps that could get you started towards the challenges. Obviously, these are not the only conceivable actionable steps nor even the only challenges for PCG. We believe PCG presents a rich and fertile soil for research and experimentation into new techniques, with obvious benefits both for industry and for the science of game design.

References

1 Daniel A. Ashlock, Stephen P. Gent, and Kenneth M. Bryden. Evolution of l-systems for compact virtual landscape generation. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2005.

2 Pippa Avery, Julian Togelius, Elvis Alistar, and Robert Pieter van Leeuwen. Computational intelligence and tower defence games. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2012.

3 M. Boden. *The creative mind: Myths and mechanisms*. Routledge, 2003.

4 Cameron Browne. *Automatic generation and evaluation of recombination games*. PhD thesis, Queensland University of Technology, 2008.

5 Alex J. Champandard. Procedural characters and the coming animation technology revolution. *AIGameDev.com*, 2012.

6 Michael Cook and Simon Colton. Multi-faceted evolution of simple arcade games. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2011.

7 Joris Dormans. Adventures in level design: Generating missions and spaces for action adventure games. In *Proceedings of the FDG Workshop on Procedural Content Generation*, 2010.

8 N. Gillian, S. O'Modhrain, and G. Essl. Scratch-Off: A gesture based mobile music game with tactile feedback. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Pittsburgh, June 4–6 2009.

9 Ken Hartsook, Alexander Zook, Sauvik Das, and Mark O. Riedl. Toward supporting stories with procedurally generated game worlds. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2011.

10 Erin J. Hastings, Ratan K. Guha, and Kenneth O. Stanley. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263, 2009.

11 Lawrence Johnson, Georgios N. Yannakakis, and Julian Togelius. Cellular Automata for Real-time Generation of Infinite Cave Levels. In *Proceedings of the ACM Foundations of Digital Games*. ACM Press, June 2010.

12 Annika Jordan, Dimitri Scheftelowitsch, Jan Lahni, Jannic Hartwecker, Matthias Kuchem, Mirko Walter-Huber, Nils Vortmeier, Tim Delbrügger, Ümit Güler, Igor Vatolkin, and Mike Preuss. Beatthebeat – music-based procedural content generation in a mobile game. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, 2012.

13 S. Karakovskiy and J. Togelius. The mario ai benchmark and competitions. In *IEEE Transactions on Computational Intelligence and AI in Games*, volume 4, pages 55–67, 2012.

14 Manuel Kerssemakers, Jeppe Tuxen, Julian Togelius, and Georgios Yannakakis. A procedural procedural level generator generator. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2012.

15 A. Liapis, G. Yannakakis, and J. Togelius. Adapting models of visual aesthetics for personalized content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):213–228, 2012.

16 Chris Pedersen, Julian Togelius, and Georgios N. Yannakakis. Modeling Player Experience for Content Creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):54–67, 2010.

17 Martin Pichlmair and Fares Kayali. Levels of sound: On the principles of interactivity in music video games. In Baba Akira, editor, *Situated Play: Proceedings of the 2007 Digital Games Research Association Conference*, pages 424–430, Tokyo, September 2007. The University of Tokyo.

18 Przemyslaw Prusinkiewicz. Graphical applications of l-systems. In *Proceedings of Graphics Interface / Vision Interface*, pages 247–253, 1986.

19 S. Risi, J. Lehman, D.B. D'Ambrosio, R. Hall, and K.O. Stanley. Combining search-based procedural content generation and social gaming in the petalz video game. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2012.

20 Jimmy Secretan, Nicholas Beato, David B. D'Ambrosio, Adelein Rodriguez, Adam Campbell, and Kenneth O. Stanley. Picbreeder: Evolving pictures collaboratively online. In *CHI'08: Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1759–1768, New York, NY, USA, 2008. ACM.

21 N. Shaker, G. N. Yannakakis, and J. Togelius. Crowd-sourcing the aesthetics of platform games. *IEEE Transactions on Computational Intelligence and Games, Special Issue on Computational Aesthetics in Games, (to appear)*, 2012.

22 Noor Shaker, Julian Togelius, Georgios N. Yannakakis, Ben Weber, Tomoyuki Shimizu, Tomonori Hashiyama, Nathan Sorenson, Philippe Pasquier, Peter Mawhorter, Glen Takahashi, Gillian Smith, and Robin Baumgarten. The 2010 Mario AI championship: Level generation track. *IEEE Transactions on Computational Intelligence and Games*, 2011.

23 R.M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra. A declarative approach to procedural modeling of virtual worlds. *Computers and Graphics*, 35:352–363, 2011.

24 Adam Smith and Michael Mateas. Answer set programming for procedural content generation: A design space approach. *IEEE Transactions on Computational Intelligence and AI in Games*, 2011.

25 Adam M. Smith and Michael Mateas. Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2010.

26 G. Smith, A. Othenin-Girard, J. Whitehead, and N. Wardrip-Fruin. Pcg-based game design: creating endless web. In *Proceedings of the International Conference on the Foundations of Digital Games*, pages 188–195. ACM, 2012.

27 Gillian Smith, Jim Whitehead, and Michael Mateas. Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):201–215, 2011.

28 Nathan Sorenson and Philippe Pasquier. Towards a generic framework for automated video game level creation. In *EvoApplications (1)*, pages 131–140, 2010.

29 Kenneth O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines (Special Issue on Developmental Systems)*, 8(2):131–162, 2007.

30 Julian Togelius. A procedural critique of deontological reasoning. In *Proceedings of DiGRA*, 2011.

31 Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, and Georgios N. Yannakakis. Multiobjective exploration of the starcraft map space. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2010.

32 Julian Togelius and Jürgen Schmidhuber. An experiment in automatic game design. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, 2008.

33 Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation: a taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3:172–186, 2011.

34 Mike Treanor, Bryan Blackford, Michael Mateas, and Ian Bogost. Game-o-matic: Generating videogames that represent ideas. In *Proceedings of the FDG Workshop on Procedural Content Generation*, 2012.

35 R Michael Young, Mark O Riedl, Mark Branly, Arnav Jhala, RJ Martin, and CJ Saretto. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development*, 1(1):51–70, 2004.