Search-Based Procedural Content Generation: A Taxonomy and Survey

Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne

***Abstract*—The focus of this survey is on research in applying evolutionary and other metaheuristic search algorithms to automatically generating content for games, both digital and nondigital (such as board games). The term *search-based procedural content generation* is proposed as the name for this emerging field, which at present is growing quickly. A taxonomy for procedural content generation is devised, centering on what kind of content is generated, how the content is represented and how the quality/fitness of the content is evaluated; search-based procedural content generation in particular is situated within this taxonomy. This article also contains a survey of all published papers known to the authors in which game content is generated through search or optimisation, and ends with an overview of important open research problems.**

***Index Terms*—Computer graphics, design automation, evolutionary computation, genetic algorithms.**

I. Introduction

THIS paper introduces the field of *search-based procedural content generation*, in which evolutionary and other stochastic and metaheuristic search techniques generate content for games. As the demand from players for ever more content rises, the video game industry faces the prospect of continually rising costs to pay for the artists and programmers to supply it. In this context a novel application for AI has opened up that focuses more on the creative and artistic side of game design [1]–[6] than on the tactical and strategic considerations common to NPC AI [7]–[10]. Algorithms that can produce desirable content on their own can potentially save significant expense. Moreover, the possibilities in this area are largely uncharted; the breadth of content potentially affected is only beginning to be understood, raising the question of whether computers will ultimately yield designs that compete with human imagination.

This review examines the first steps that researchers have taken towards addressing this question as this nascent field begins to coalesce. The aim is to investigate what can and cannot be accomplished with such techniques and to outline some of the main research challenges in the field. Distinctions will be introduced between different approaches, and a comprehensive survey of published examples will be discussed and classified according to these distinctions. First, the overarching area of procedural content generation is introduced.

Procedural content generation (PCG) refers to creating game content automatically, through algorithmic means. In this paper, the term *game content* refers to all aspects of the game that affect gameplay other than nonplayer character (NPC) behavior and the game engine itself. This set includes such aspects as terrain, maps, levels, stories, dialogue, quests, characters, rulesets, dynamics, and weapons. The survey explicitly excludes the most common application of search and optimization techniques in academic games research, namely NPC AI, because the work in that area is already well-documented in the literature [7]–[10] while other areas of content are significantly less publicized. The review also puts less weight on decorative assets such as lighting, textures, and sound effects, insofar as they do not directly affect gameplay. (It should be noted that there is a rich literature on procedural generation of textures [11], e.g., for use as ornamentation in games [12], which will not be covered here.) Typically, PCG algorithms create a specific content instance from a short description (parameterisation or seed), which is in some way much smaller than the "expanded" content instance. The generation process is often, but not always, partly random.

There are several reasons for game developers to be interested in procedural content generation. The first is memory consumption—procedurally represented content can typically be compressed by keeping it "unexpanded" until needed. A good example is the classic space trading and adventure game *Elite* (Acornsoft 1984), which managed to keep hundreds of star systems in the few tens of kilobytes of memory available on the hardware of the day by representing each planet as just a few numbers; in expanded form, the planets had names, populations, prices of commodities, etc.

Another reason for using PCG is the prohibitive expense of manually creating game content. Many current generation AAA titles employ software such as *SpeedTree* (Interactive Data Visualization, Inc.) to create whole areas of vegetation based on just a few parameters, saving precious development resources while allowing large, open game worlds. This argument becomes ever more important as expectations about the amount and level of detail of game content continue to increase in pace with game hardware improvements.

A third argument for PCG is that it might allow the emergence of completely new types of games, with game mechanics

Manuscript received September 14, 2010; revised December 08, 2010; accepted March 18, 2011. Date of publication April 29, 2011; date of current version September 14, 2011. This work was supported in part by the Danish Research Agency, Ministry of Science, Technology and Innovation; project *AGameComIn* (number 274-09-0083).

J. Togelius and G. N. Yannakakis are with the IT University of Copenhagen, 2300 Copenhagen, Denmark (e-mail: julian@togelius.com; yannakakis@itu.dk).

K. O. Stanley is with the University of Central Florida, Orlando, FL 32816 USA (e-mail: kstanley@eecs.ucf.edu).

C. Browne is with the Imperial College London, London, SW7 2AZ, U.K. (e-mail: cameron.browne@btinternet.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2011.2148116

built around content generation. If new content can be generated with sufficient variety in real time then it may become possible to create truly endless games. Further, if this new content is created according to specific criteria, such as its suitability for the playing style of a particular player (or group/community of players) or based on particular types of player experience (challenge, novelty, etc.), it may become possible to create games with close to infinite replay value. Imagine a game that never ends—wherever you go, whatever you do, there is always something new to explore, and this new content is consistently novel while at the same time tuned to your playing style and offering the type of challenges you want. Persistent-world games in which players demand a continual stream of new content in particular can benefit from such a capability. While PCG technology is not yet impacting commercial games in this way, this vision motivates several researchers within search-based PCG, as discussed in this article.

Finally, PCG can augment our limited, human imagination. Not every designer is a genius, at least not all the time, and a certain amount of sameness might be expected. Offline algorithms might create new rulesets, levels, narratives, etc., which can then inspire human designers and form the basis of their own creations. This potential also motivates several search-based PCG researchers, as discussed in this paper.

II. Dissecting Procedural Content Generation

While PCG in different forms has been a feature of various games for a long time, there has not been an academic community devoted to its study. This situation is now changing with the recent establishment of a mailing list,[1] an IEEE CIS Task Force,[2] a workshop,[3] and a wiki[4] on the topic. However, there is still no textbook on PCG, or even an overview paper offering a basic taxonomy of approaches. To fill this gap, this section aims to draw some distinctions. Most of these distinctions are not binary, but rather a continuum wherein any particular example of PCG can be placed closer to one or the other extreme. Note that these distinctions are drawn with the main purpose of clarifying the role of search-based PCG; of course other distinctions will be drawn in the future as the field matures, and perhaps the current distinctions will need to be redrawn. More distinctions are clearly needed to fully characterize, e.g., PCG approaches that are not search-based. Still, the taxonomy herein should be useful for analyzing many PCG examples in the literature, as well as published games.

A. Online Versus Offline

The first distinction is whether the content generation is performed *online* during the runtime of the game, or *offline* during game development. An example of the former is when the player enters a door to a building and the game instantly generates the interior of the building, which was not there before; in the latter case an algorithm suggests interior layouts that are then edited and perfected by a human designer before the game is shipped. Intermediate cases are possible, wherein an algorithm running on, for example, a real-time strategy (RTS) server suggests new maps to a group of players daily based on logs of their recent playing styles. Online PCG places two or three main requirements on the algorithm: that it is very fast, that it has a predictable runtime and (depending on the context) that its results are of a predictable quality.

B. Necessary Versus Optional Content

A further distinction relating to the generated content is whether that content is *necessary* or *optional*. Necessary content is required by the players to progress in the game—e.g., dungeons that need to be traversed, monsters that need to be slain, crucial game rules, and so on—whereas optional content is that which the player can choose to avoid, such as available weapons or houses that can be entered or ignored. The difference here is that necessary content always needs to be correct; it is not acceptable to generate an intractable dungeon, unplayable ruleset or unbeatable monster if such aberrations makes it impossible for the player to progress. It is not even acceptable to generate content whose difficulty is wildly out of step with the rest of the game. On the other hand, one can allow an algorithm that sometimes generates unusable weapons and unreasonable floor layouts if the player can choose to drop the weapon and pick another one or exit a strange building and go somewhere else instead.

Note that it depends significantly on the game design and the game fiction whether content is categorized as necessary or optional, and to what extent optional content is allowed to "fail." The first-person shooter *Borderlands* (Gearbox Software 2009) has randomly generated weapons, many of which are not useful, yet exploring these items is a core part of the gameplay and consistent with the game fiction. On the other hand, a single poorly designed and apparently "artificial" plant or building might break the suspension of disbelief in a game with a strong focus on visual realism such as *Call of Duty 4: Modern Warfare* (Infinity Ward 2007). Also note that some types of content might be optional in one class of games, and necessary in another (see, e.g., optional dungeons). Therefore, the analysis of what content is optional should be done on a game-for-game basis.

C. Random Seeds Versus Parameter Vectors

Another distinction concerning the generation algorithm itself is to what extent it can be parameterised. All PCG algorithms create "expanded" content of some sort based on a more compact representation. At one extreme, the algorithm might simply take a seed to its random number generator as input; at another extreme, the algorithm might take as input a multidimensional vector of real-valued parameters that specify the properties of the content it generates. For example, a dungeon generator might be called with parameters specifying such properties as the number of rooms, branching factor of corridors, clustering of item locations, etc. Another name for the random seed-parameter vector continuum is the number of *degrees of control.*

[1]http://groups.google.com/proceduralcontent

[2]http://game.itu.dk/pcg/

[3]http://pcgames.fdg2010.org/

[4]http://pcg.wikidot.com

D. Stochastic Versus Deterministic Generation

A distinction only partly orthogonal to those outlined so far concerns the amount of randomness in content generation. The right amount of variation in outcome between different runs of an algorithm with identical parameters is a design question. It is possible to conceive of deterministic generation algorithms that always produce the same content given the same parameters, but many algorithms (e.g., dungeon-generation algorithms in rogue-like games) do not. (Note that the random number generator seed is not considered a parameter here; that would imply that all algorithms are deterministic.)

Completely deterministic PCG algorithms can be seen as a form of data compression. A good example of this use of PCG techniques is the first-person shooter *.kkrieger* (.theprodukkt 2004), which manages to squeeze all of its textures, objects, music and levels together with its game engine in 96 kb of storage space. Another good example is *Elite*, discussed above.

E. Constructive Versus Generate-and-Test

A final distinction is between algorithms that can be called *constructive* and those that can be described as *generate-and-test*. A constructive algorithm generates the content once, and is done with it; however, it needs to make sure that the content is correct or at least "good enough" while it is being constructed. This can be done through only performing operations, or sequences of operations, that are guaranteed to never produce broken content. An example of this approach is the use of fractals to generate terrains [13].

A generate-and-test algorithm incorporates both a generate and a test mechanism. After a candidate content instance is generated, it is tested according to some criteria (e.g., Is there a path between the entrance and exit of the dungeon, or does the tree have proportions within a certain range?). If the test fails, all or some of the candidate content is discarded and regenerated, and this process continues until the content is good enough.

The Markov chain algorithm is a typical constructive method. In this approach, content is generated on-the-fly according to observed frequency distributions in source material [14]. This method has generated novel but recognizable game names [3], natural language conversations, poetry, jazz improvisation [15], and content in a variety of other creative domains. Similarly, generate-and-test methods such as evolutionary algorithms are widely used for PCG in nongame domains, for example in the generation of procedural art; the evaluation function for this very subjective content may be a human observer who specifies which individuals survive each generation [16], [17] or a fully automated process using image processing techniques to compare and judge examples [18]. Although PCG has been successfully applied to a range of creative domains, we shall focus on its application to games in this survey.

III. Search-Based Procedural Content Generation

Search-based procedural content generation is a special case of the generate-and-test approach to PCG, with the following qualifications.

- The test function does not simply accept or reject the candidate content, but grades it using one *or a vector of* real numbers. Such a test function is variously called a *fitness*, *evaluation*, and *utility function*; here, we will use "evaluation function" and call the number or vector it assigns to the content the *fitness* or simply the *value* of the content.
- Generating new candidate content is contingent upon the fitness value assigned to previously evaluated content instances; in this way the aim is to produce new content with higher value.

While most examples in this article rely on evolutionary algorithms, we chose the term "search-based" rather than "evolutionary" for several reasons. One is to avoid excluding other common metaheuristics, such as simulated annealing [19] and particle swarm optimization [20], or simple stochastic local search. Our definition of "search-based" explicitly allows all forms of heuristic and stochastic search/optimisation algorithms. (Some cases of exact search, exhaustive search, and derivative-based optimization might qualify as well, though in most cases the content evaluation function is not differentiable and the content space too big to be exhaustively searched.) Another reason is to avoid some connotations of the word "evolutionary" in the belief that search-based is more value-neutral. Finally, the term search-based for a similar range of techniques is established within search-based software engineering [21], [22]. The over-representation of evolutionary algorithms in this paper is simply a reflection of what papers have been published in the field.

Almost all of the examples in Section IV use some form of evolutionary algorithm as the main search mechanism, as evolutionary computation has so far been the method of choice among search-based PCG practitioners. In an evolutionary algorithm, a population of candidate content instances are held in memory. Each generation, these candidates are evaluated by the evaluation function and ranked. The worst candidates are discarded and replaced with copies of the good candidates, except that the copies have been randomly modified (i.e., *mutated*) and/or recombined. Fig. 1 illustrates the overall flow of a typical search-based algorithm, and situates it in relation to constructive and simple generate-and-test approaches.

As mentioned above, search-based PCG does not need to be married to evolutionary computation; other heuristic/stochastic search mechanisms are viable as well. In our experience, the same considerations about representation and the search space largely apply regardless of the approach to search. If we are going to search a space of game content, we need to represent the content somehow, and the representation (and associated variation operators) shapes the search space. Regardless of the algorithm chosen we will also need to evaluate the content, and the design of the evaluation function is another key design decision. Some terminology from evolutionary computation will be used in this section, simply because that field has a well-developed conceptual apparatus suitable for adapting to our purposes.

A. Content Representation and Search Space

A central question in stochastic optimization and metaheuristics concerns how to represent whatever is evolved. In other words, an important question is how *genotypes* (the data structures that are handled by the evolutionary algorithm) are mapped to *phenotypes* (the data structure or process that is

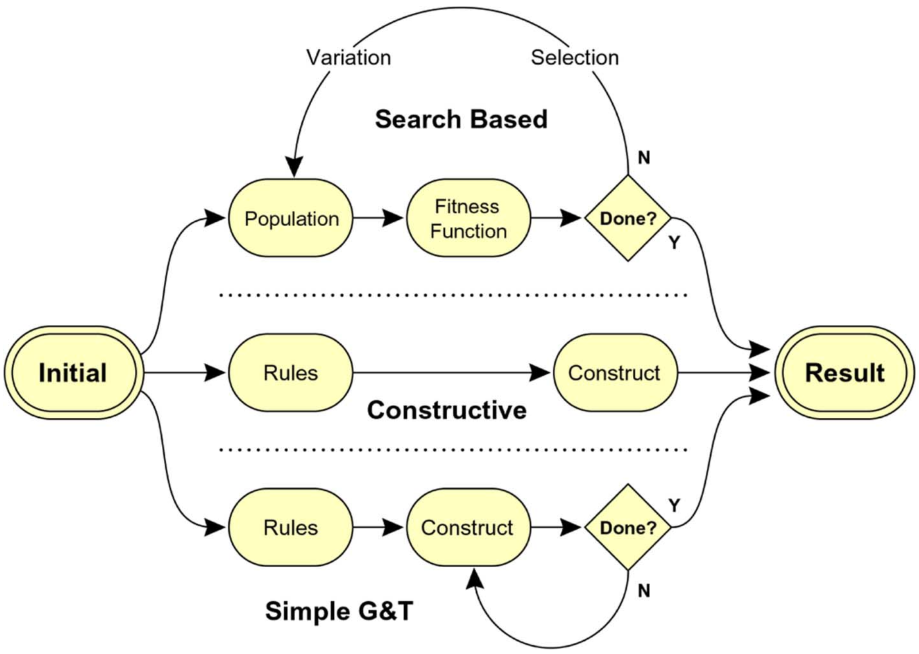


Fig. 1. Three approaches to procedural content generation: constructive, simple generate-and-test and search-based. Note that not all search/optimization algorithms suitable for PCG keep a population of candidate content, but most of the commonly used ones do.

evaluated by the evaluation function).[5] The distinction between genotype and phenotype can be thought of as the distinction between a blueprint and a finished building, alternatively as between an algorithm and the output of the algorithm. In a game content generation scenario, the genotype might be the instructions for creating a game level, and the phenotype the actual game level. We can always talk of a genotype/phenotype distinction when stochastic search is employed, even in simple cases such as searching for the roots of an equation; in this case, the variable values are the genotype, the result of substituting these values for the variables the genotype and the calculation of the left-hand side of the equation the genotype-to-phenotype mapping.

An important distinction among representations is between *direct encodings* and *indirect encodings*. Direct encodings imply relative computational simplicity in the genotype-to-phenotype mapping, i.e., that the size of the genotype is linearly proportional to the size of the phenotype and that each part of the genome maps to a specific part of the phenotype. In indirect encodings, the genotype maps nonlinearly to the phenotype and the former need not be proportional to the latter; often, complex computation is necessary to create the phenotype from the genotype ([23]–[25]; see [26] for a review).

The study of representations is a broad research field within evolutionary computation, and has produced several original concepts that are relevant to search-based PCG [27]. A particularly well-studied case is that in which candidates are represented as vectors of real numbers. These can easily be analyzed, and many standard algorithms are more readily applied to such representations compared to more unusual representations. In order to search the space effectively, the vector should have the right dimensionality. Short vectors that are incapable of properly representing the content (or that introduce the wrong bias in search space) should be avoided, while at the same time avoiding the "curse of dimensionality" associated with vectors that are too large (or, alternatively, the algorithm should find the right dimensionality for the vector [28]). Another principle is that the representation should have a high *locality*, meaning that a small change to the genotype should on average result in a small change to the phenotype and a small change to the fitness value.

Apart from these concerns, it is important that the chosen representation is capable of representing all the interesting solutions. However, this ideal can be hard to attain in practice for indirect encodings, for which there might be areas of phenotype space to which no genotypes map, and no simple way of detecting this. With direct encodings, it is in general easy to ascertain that any particular area of solution space could in principle be found by the search process.

These considerations are important for search-based PCG, as the representation and search space must be well-matched to the domain for the process to perform at a high level. There is a continuum within search-based PCG between direct and indirect representation. As a concrete example, a maze (for use, e.g., in a "roguelike" dungeon adventure game) might be represented:

1) directly as a grid where mutation works on the contents (e.g., wall, free space, door, monster) of each cell;
2) more indirectly as a list of the positions, orientations and lengths of walls (an example of this can be found in [29]);
3) even more indirectly as a repository of different reusable patterns of walls and free space, and a list of how they are distributed (with various transforms such as rotation and scaling) across the grid;

[5]This terminology is taken from evolutionary computation, but similar distinctions and considerations can be found in other forms of optimisation.

4) very indirectly as a list of desirable properties (number of rooms, doors, monsters, length of paths and branching factor);
5) most indirectly as a random number seed.

These representations yield very different search spaces. In the first case, all parts of phenotype space are reachable, as the one-to-one mapping ensures that there is always a genotype for each phenotype. Locality is likely to be high because each mutation can only affect a single cell (e.g., turn it from wall into free space), which in most cases changes the fitness of the map only slightly. However, because the length of the genotype would be equal to the number of cells in the grid, mazes of any interesting size quickly encounter the curse of dimensionality. For example, a 100×100 maze would need to be encoded as a vector of length 10 000, which is more than many search algorithms can effectively approach.

At the other end of the spectrum, option 5 does not suffer from high search space dimensionality because it searches a one-dimensional space. The question of whether all interesting points of phenotype space can be reached depends on the genotype-to-phenotype mapping, but it is possible to envision one where they can (e.g., iterating through all cells and deciding their content based on the next random number). However, the reason this representation is unsuitable for search-based PCG is that there is no locality; one of the main features of a good random number generator is that there is no correlation between the numbers generated by neighbouring seed values. All search performs as badly (or as well) as random search.

Options 2–4 might thus all be more suitable representations for searching for good mazes. In options 2 and 3, the genotype length would grow with the desired phenotype (maze) size, but sublinearly, so that reasonably large mazes could be represented with tractably short genotypes. In option 4, genotype size is independent of phenotype size, and can be made relatively small. On the other hand, the locality of these intermediate representations depends on the care and domain knowledge with which each genotype-to-phenotype mapping is designed; both high- and low-locality mechanisms are conceivable.

B. Evaluation Functions

Once a candidate content item is generated, it needs to be evaluated by the evaluation function and assigned a scalar (or a vector of real numbers[6]) that accurately reflects its suitability for use in the game. In this paper the word "fitness" has the same meaning as "utility" in some optimization contexts, and the words could be used interchangeably. Another term that can be found in the optimization literature is "cost"; an evaluation function, as defined here, is the negative of a cost function.

Designing the evaluation function is ill-posed; the designer first needs to decide what, exactly, should be optimized and then how to formalize it. For example, one might intend to design a search-based PCG algorithm that creates "fun" game content, and thus an evaluation function that reflects how much the particular piece of content contributes to the player's sense of fun while playing. Or, alternatively, one might want to consider immersion, frustration, anxiety, or other emotional states when designing the evaluation function. However, emotional states are not easily formalized, and it is not entirely clear how to measure them even with multiple modalities of user input (such as physiological measures, eye-gaze, speech, and video-annotated data) and a psychological profile of the player. With the current state of knowledge, any attempt to estimate the contribution to "fun" (or affective states that collectively contribute to player experience) of a piece of content is bound to rely on conflicting assumptions. More research within affective computing and multimodal interaction is needed at this time to achieve fruitful formalizations of such subjective issues; see [31] for a review. Of course, the designer can also try to circumvent these issues by choosing to measure narrower and more game-specific properties of the content.

Three key classes of evaluation function can be distinguished for the purposes of PCG: *direct*, *simulation-based*, and *interactive* evaluation functions. (A more comprehensive discussion about evaluation functions for game content can be found in a recently published overview paper [32]).

1) Direct Evaluation Functions: In a *direct* evaluation function, some features are extracted from the generated content and mapped directly to a fitness value. Hypothetical features might include the number of paths to the exit in a maze, firing rate of a weapon, spatial concentration of resources on an RTS map, material balance in randomly selected legal positions for board game rule set, and so on. The mapping between features and fitness might be linear or nonlinear, but ideally does not involve large amounts of computation and is likely specifically tailored to the particular game and content type. This mapping might also be contingent on a model of the playing style, preferences or affective state of the player, which means that an element of *personalization* is possible.

An important distinction within direct evaluation functions is between *theory-driven* and *data-driven* functions. In theory-driven functions, the designer is guided by intuition and/or some qualitative theory of player experience to derive a mapping. On the other hand, data-driven functions are based on data collected on the effect of various examples of content via, for example, questionnaires or physiological measurements, and then using automated means to tune the mapping from features to fitness values.

2) Simulation-Based Evaluation Functions: It is not always apparent how to design a meaningful direct evaluation function for some game content—in some cases, it seems that the content must be sufficiently experienced and manipulated to be evaluated. A *simulation-based* evaluation function, on the other hand, is based on an artificial agent playing through some part of the game that involves the content being evaluated. This approach might include finding the way out of a maze while not being killed or playing a board game that results from the newly-generated rule set against another artificial agent. Features are then extracted from the observed gameplay (e.g., Did the agent win? How fast? How was the variation in playing styles employed?) and used to calculate the value of the content. The artificial agent

[6]In the case of more than one fitness dimension, a multiobjective optimizations algorithm is appropriate [30], which leads to considerations not discussed here.

might be completely hand-coded, or might be based on a learned behavioural model of a human player, making personalization possible for this type of evaluation function as well.

Another key distinction is between *static* and *dynamic* simulation-based evaluation functions. In a static evaluation function, it is not assumed that the agent changes while playing the game; in a dynamic evaluation function the agent changes during the game and the evaluation function somehow incorporates this change. For example, the implementation of the agent can be based on a learning algorithm and the fitness value can be dependent on *learnability*: how well and/or fast the agent learns to play the content that is being evaluated. Learning-based dynamic evaluation functions are especially appropriate when little can be assumed about the content and how to play it. Other uses for dynamic evaluation functions include capturing, e.g., order effects and user fatigue. It should be noted that while simulating the game environment can typically be executed faster than real-time, simulation-based evaluation functions are in general more computationally expensive than direct evaluation functions; dynamic simulation-based evaluation functions can thus be time-consuming, all but ruling out online content generation.

3) Interactive Evaluation Functions: Interactive evaluation functions score content based on interaction with a player in the game, which means that fitness is evaluated during the actual gameplay. Data can be collected from the player either *explicitly* using questionnaires or verbal cues, or *implicitly* by measuring, e.g., how often or long a player chooses to interact with a particular piece of content [2], [33], when the player quits the game, or expressions of affect such as the intensity of button-presses, shaking the controller, physiological response, eye-gaze fixation, speech quality, facial expressions, and postures.

The problem with explicit data collection is that it can interrupt the gameplay, unless it is well integrated in the game design. On the other hand, the problems with indirect data collection are that the data is often noisy, inaccurate, of low-resolution and/or delayed, and that multimodal data collection may be technically infeasible and/or expensive for some types of game genres—e.g., eye-tracking and biofeedback technology are still way too expensive and unreliable for being integrated within commercial-standard computer games. However, such technology can more easily be deployed in lab settings and used to gather data on which to base player models that can then be used outside of the lab.

C. Situating Search-Based PCG

At this point, let us revisit the distinctions outlined in Section II and ask how they relate to search-based PCG. In other words, the aim is to situate search-based PCG within the family of PCG techniques. As stated above, search-based PCG algorithms are generate-and-test algorithms. They might take parameter vectors or not. If they do, these are typically parameters that modify the evaluation function, such as the desired difficulty of the generated level. As evolutionary and similar search algorithms rely on stochasticity for, e.g., mutation, a random seed is needed; therefore, these algorithms should be classified as stochastic rather than deterministic. There is no way of knowing exactly what you will get with search-based PCG algorithm, and in general no way of reproducing the same result except for saving the result itself.

As there is no general proof that any metaheuristic algorithms ultimately converge (except in a few very simple cases), there is no guaranteed completion time for an search-based PCG algorithm, and no guarantee that it will produce good enough solutions. The time taken depends mostly on the evaluation function, and because an evaluation function for a content generation task would often include some kind of simulation of the game environment, it can be substantial. Some of the examples in the survey section below take days to run, others produce high-quality content in under a second. For these reasons it might seem that search-based PCG would be less suitable for online content generation, and better suited for offline exploration of new design ideas. However, as we shall see later, it is possible to successfully base complete game mechanics on search-based PCG, at least if the content generated is optional rather than necessary.

We can also choose to look at the relation between indirect representation and search-based PCG from a different angle. If our search-based PCG algorithm includes an indirect mapping from genotype to phenotype, this mapping can be viewed as a PCG algorithm in itself, and an argument can be made for why certain types of PCG algorithms are more suitable than others for use as part of an search-based PCG algorithm. In other words, this "inner" PCG algorithm (the genotype-to-phenotype mapping) becomes a key component in the main PCG algorithm. We can also see genotype-to-phenotype mapping as a form of data decompression, which is consistent with the view discussed in Section II that deterministic PCG can be seen as data compression. It is worth noting that some indirect encodings used in various evolutionary computation application areas bear strong similarities to PCG algorithms for games; several such indirect encodings are based on L-systems [34], as are algorithms for procedural tree and plant generation [24], [35].

IV. Survey of Search-Based PCG

Search-based PCG is a new research area and the volume of published papers is still manageable. In this section, we survey published research in search-based PCG. While no review can be guaranteed to cover all published research in a field, we have attempted a thorough survey that covers much of the known literature.

The survey proceeds in the following order: it first examines work on generating necessary game content, and then proceeds to generating optional content, following the distinction in Section II-B. Of course, this distinction is not always clear cut: some types of content might be optional in one game, but necessary in another. Within each of these classes we distinguish between types of content: rules and mechanics, puzzles, tracks, levels, terrains, and maps are deemed *necessary*, whereas weapons, buildings and camera placement are deemed *optional.* Within each content type section, we discuss each project in approximate chronological order, based on the years in which the relevant papers were first published.

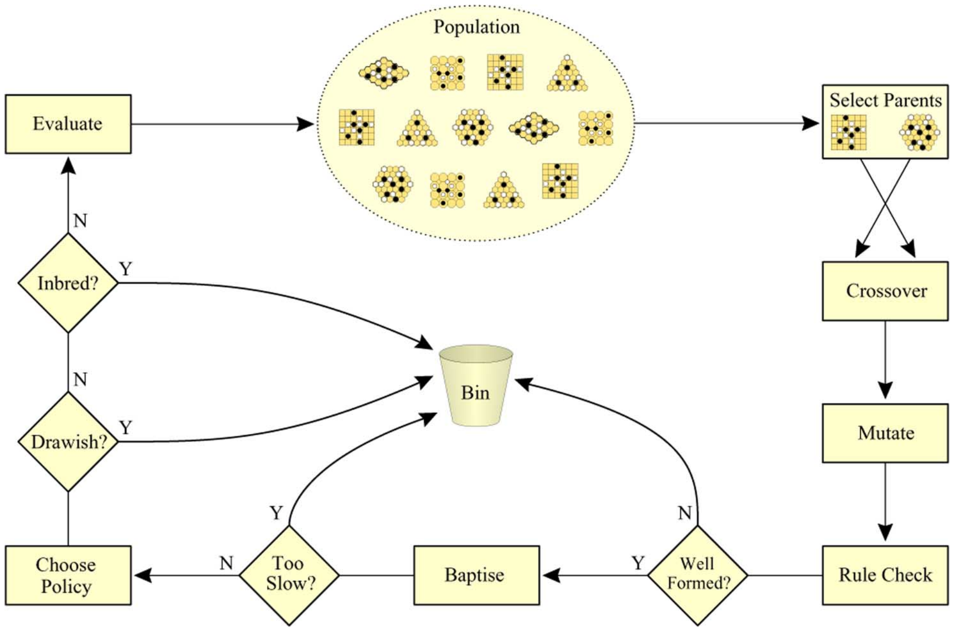


Fig. 2. *Ludi* system for generating game rules.

A. Necessary Content

1) Rules and Mechanics: Game rules and their associated game mechanics might be said to be the most fundamental type of content; it is hard to conceive of a game without any rules. It is also very uncommon to change the rules while a game is being played, though such examples exist. This perspective places rules firmly on the "necessary" end of content that can be generated.

Hom and Marks [36] evolved two-player board game rules for balance. Rules are represented relatively directly as expression trees in the zillions of games (ZOG) game description language [37], with the search space constrained to games similar to Tic-Tac-Toe, Reversi, and Checkers; a total of 5616 games are contained in this space. The evaluation function is simulation-based, static and theory-driven: a game is tested by playing two sides against each other in the ZOG game engine. Quality values are calculated as the negative of the score difference between the two players, assuming that board games are better when there is no advantage for either side.

Togelius and Schmidhuber [38] conducted an experiment in which rulesets were evolved offline for grid-based games in which the player moves an agent, in a manner similar to a discrete version of Pac-Man. Apart from the agent, the grid is populated by walls and "things" of different colors, which can be interpreted as items, allies or enemies depending on the rules. Rulesets are represented fairly directly as fixed-length parameter vectors, interpreted as the effects on various objects when they collided with each other or the agent, and their behaviour. For example, blue things could move clockwise around the grid and kill red things upon collision, but be teleported away and increase the game score when colliding with the agent. A relatively wide range of games can be represented using this vocabulary, and genotype generation is deterministic except for the starting position of objects. The evaluation function is simulation-based, dynamic and theory-driven: an evolutionary reinforcement learning algorithm learns each ruleset and the ruleset is scored according to how well it learned. Games that are impossible or trivial are given low fitness values, whereas those that can be learned after some time score well.

Browne [3] developed the *Ludi* system for offline design of rules for board games using a form of genetic programming. Game rules are represented relatively directly as expression trees, formulated in a game description language that was specially designed for the project. This language allows the representation of a sufficiently wide variety of board games, including many well-known games. The evolutionary process that creates new rule sets is nonstandard in the sense that suboptimal children with poor performance or badly formed rules are not discarded but are instead retained in the population with a lower priority, to maintain a necessary level of genetic diversity. The evaluation function is a complex combination of direct measures and static simulation-based measures: for example, standard game-tree search algorithms are used to play the generated game as part of the content evaluation, to investigate issues such as balance and time to play the game. While hand-coded, the evaluation function is based on extensive study of existing board games and measurements of user preferences for board games that exhibited various features. An illustration of the architecture of *Ludi* is shown in Fig. 2.

Smith and Mateas [5] provide a representation of game rules for simple games that differ from most SBPCG, but which could easily form part of a search-based approach. Game rules and ontologies are represented relatively indirectly as answer sets in answer set programming (ASP), which is a form of constraint programming. Each ruleset is a list of assertions of arbitrary length, in which each assertion can specify the existence of a kind of NPC, the effect of two entities colliding, a winning condition, etc. Using this encoding, many questions about the game (such as "is it winnable?" and "can it be won without shooting anyone?") can be answered through deduction rather than playthrough. Sets of games that correspond to positive answers to stated questions (rules that satisfy stated constraints) can be generated through simply posing the relevant question. In the current implementation of the idea, in which games are generated as part of a metagame in which players explore the space of game mechanics, a simple generate-and-test procedure is used where games that are unplayable are rejected by the user. Depending on the implementation of ASP within the solver, the generation of answer sets that fit the specified constraints might or might not be seen as a search-based (as we use the term) generation process.

Salge and Mahlmann [39] propose a simulation-based evaluation function based on the information-theoretic concept of relevant information, which could be adapted to evaluate game mechanics in a wide range of game types. The relevant information of a game mechanic is defined in this context as the minimum amount of information (about the game state) needed to realize an optimal strategy. This threshold can be approximated by evolving a player for the game in question and measuring the mutual information between sensor inputs (state description) and actions taken by the player. The authors argue that several common game design pathologies correlate to low relevant information.

2) Puzzles: Puzzles are often considered a genre of gaming, though opinion is divided on whether popular puzzles such as *Sudoku* should be considered games or not. Additionally, however, puzzles are part of very many types of games: there are puzzles inside the dungeons of the *Legend of Zelda* (Nintendo 1986) game series, in locations of classic adventure games such as *The Secret of Monkey Island* (LucasArts 1990), and there are even puzzles of a sort inside the levels of *first-person shooter* (FPS) games such as *Doom* (id Software 1993). Usually, these puzzles need to be solved to progress in the game, which means they are necessary content.

Oranchak [40] constructed a genetic algorithm-based puzzle generator for *Shinro*, a type of Japanese number puzzle, somewhat similar to Sudoku. A Shinro puzzle is solved by deducing which of the positions on an 8×8 board contain "holes" based on various clues. The puzzles are directly encoded as matrices, wherein each cell is empty or contains a hole or arrow (clue). The evaluation function is mainly simulation-based through a tailor-made Shinro solver. The solver is applied to each candidate puzzle, and then its entertainment value is estimated based on both how many moves are required to solve the puzzles, and some direct measures including the number of clues and their distribution.

Ashlock [41] generated puzzles of two different but related types—chess mazes and chromatic puzzles—using evolutionary computation. Both types of puzzles are represented directly; in the case of the chess mazes, as lists of chess pieces and their positions on the board, and in case of the chromatic puzzles as 8×8 grids in which a number in each cell indicates its color. The evaluation function is simulation-based and theory-driven: a dynamic programming approach tries to solve each puzzle, and fitness is simply the number of moves necessary to solve the puzzle. A target fitness is specified for each type of puzzle, aiming to give an appropriate level of challenge.

3) Tracks and Levels: Most games that focus on the player controlling a single agent in a two- or three-dimensional space are built around levels, which are regions of space that the player-controlled character must somehow traverse, often while accomplishing other goals. Examples of such games include platform games, FPS games, two-dimensional scrolling arcade games and even racing games.

Togelius *et al.* [42], [43] designed a system for offline/online generation of tracks (necessary or optional content, depending on game design) for a simple racing game. Tracks are represented as fixed-length parameter vectors. A racing track is created from the parameter vector by interpreting it as the parameters for b-spline (a sequence of Bezier curves) yielding a deterministic genotype-to-phenotype mapping. The resulting shape forms the midline of the racing track. The evaluation function is simulation-based, static and personalized. Each candidate track is evaluated by letting a neural network-based car controller drive on the track. The fitness of the track is dependent on the driving performance of the car: amount of progress, variation in progress and difference between maximum and average speed. (Note that it is the track that is being tested, not the neural network-based car controller.) The personalization comes from the neural network previously having been trained to drive in the style of the particular human player for which the new track is being created. This somewhat unintuitive process was shown effective in generating tracks suited to particular players.

Pedersen *et al.* [44] modified an open-source clone of the classic platform game *Super Mario Bros* to allow personalized level generation. Levels are represented very indirectly as a short parameter vector describing mainly the number, size and placement of gaps in the level. This vector is converted to a complete level in a stochastic fashion. The evaluation function is direct, data-driven and personalized, using a neural network that converts level parameters and information about the player's playing style to one of six emotional state predictors (fun, challenge, frustration, predictability, anxiety, boredom), which can be chosen as components of an evaluation function. These neural networks are trained through collecting both gameplay metrics and data on player preferences using variants of the game on a web page with an associated questionnaire.

Sorenson and Pasquier [45] devised an indirect game level representation aimed at games across a number of different but related genres. Their representation is based on "design elements," which are elements of levels (e.g., individual platforms or enemies, the vocabulary would need to be specified for each game by a human designer) that can be composed into complete levels. The design elements are laid out spatially in the genome, to simplify crossover. Levels are tested in two phases: first, the general validity of the level is tested by ensuring, e.g., that all areas of the level are appropriately connected, or that a required number of design elements of each type are present. Any level that fails the test is relegated to a second population, using the FI-2Pop evolutionary algorithm [46] that is specially designed for constraint satisfaction problems. Valid levels are then assessed for fitness by a direct or weakly simulation-based theory-driven evaluation function, which estimates the difficulty of the level and rewards intermediately challenging levels. This function might be implemented as the length from start to finish, or the size of gaps to jump over, depending on the particular game.

Jennings-Teats *et al.* [47] describe initial work towards creating platform game levels that adapt their difficulty to a human player during runtime. The generation process is based on generate-and-test, but is not completely search-based, as no optimization mechanism is employed. Initially, player data are collected to rank short level segments according to their difficulty. The level is then generated as it is played by composing the segments in front of the player, based on a rhythm-based generation mechanism. The generated level part is then evaluated by a number of "critics" based on the acquired difficulty ratings of the constituent level segments. Level parts that are rejected by the critics are simply regenerated until parts of appropriate difficulty are found. The ensemble of critics can therefore be conceived as a direct, data-driven binary evaluation function.

4) Terrains and Maps: A large number of games are built around terrains or maps, which can be loosely defined as two- or two-and-a-half-dimensional planes with various features situated within them that might or might not be gameplay-related (e.g., types of surface, or impassable rocks or rivers) and possibly a heightmap specifying elevations of various parts of the maps. In particular, many strategy games are heavily dependent on maps and the character of the map can strongly influence the gameplay. The category of terrains and maps partly overlaps with the previous category, as, e.g., FPS levels can also be considered as maps.

Frade *et al.* [48] evolved terrains for the video game *Chapas*. The terrain was represented very indirectly as expression trees, which were evolved with genetic programming using an approach similar to the CPPN [25] encoding. The elevation at each point is determined by querying the evolved expression trees, substituting the current position coordinates for constants in the tree. The evaluation function is direct and theory-driven, based on "accessibility"; this function scores maps depending on the largest connected area of flat or almost-flat terrain (this value is bounded to prevent the evolution of completely flat maps). An interesting result was that whereas the algorithm produced useful maps, they were sometimes visually unpleasant and required human inspection before being used.

Togelius *et al.* [49], [50] designed a method for generating maps for RTS games. Two semidirect representations were investigated, one for a generic heightmap-based strategy game and one for the seminal RTS *StarCraft* (Blizzard 1998). In both representations, positions of bases and resources are represented directly as (x, y) coordinates, but other terrain features are represented more indirectly. For the heightmap-based representation, positions, standard deviations and heights of several two-dimensional Gaussians are evolved, and the height of the terrain at each point is calculated based on those. For the StarCraft representation, mountain formations are drawn using a stochastic (but deterministic) method inspired by "turtle graphics." A collection of direct and lightly simulation-based, theory-driven evaluation functions are used to evaluate the maps. These functions were directly motivated by gameplay considerations and are to a large extent based on the A^* search algorithm; for example, the resource balance evaluation function penalises the difference in the closest distance to resources between the players. The search mechanism in this method differs considerably from most other search-based PCG research because it is based on a multiobjective evolutionary algorithm (MOEA) [30] (the particular algorithm used is the SMS-EMOA [51]). Because each of the evaluation functions is partially conflicting with several of the other evaluation functions, the MOEA tries to find the optimal tradeoffs between these objectives, expressed as a Pareto front. A human designer, or a game-balancing algorithm, can then choose solutions from among those on the Pareto front. An example of a map generated using this method can be seen in Fig. 3.

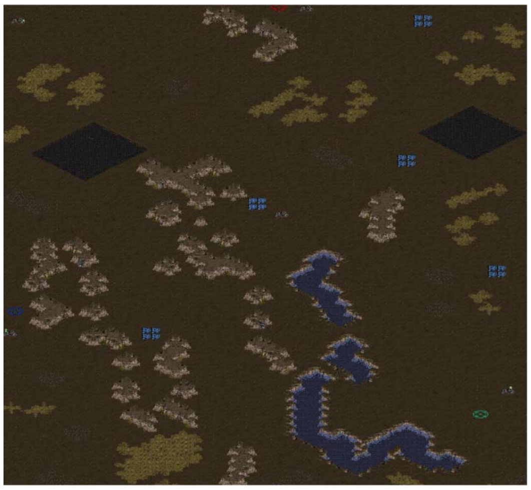
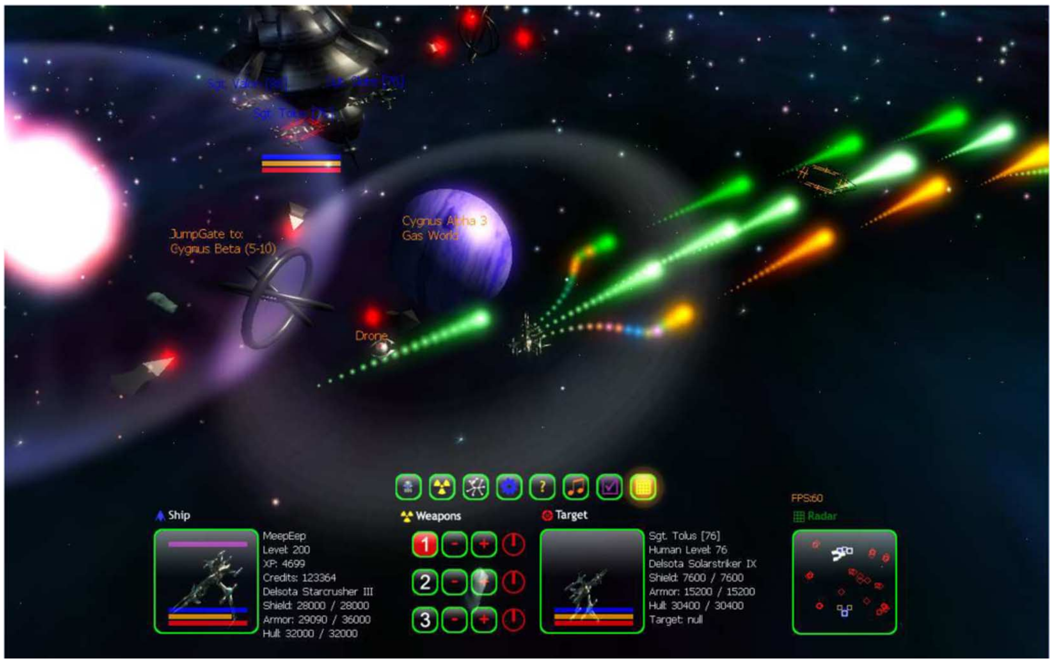


Fig. 3. Evolved map for the *StarCraft* RTS game.

It should be noted that there is a substantial body of literature on constructive methods for generating maps, which is not extensively discussed here as it is not search-based. Terrain generation systems for games based on fractals (such as the diamond-square algorithm [52], [53]), on agent-based simulated erosion [54] or on cellular automata [55] have been proposed previously; while most such algorithms enjoy a short and predictable runtime, they cannot generally be controlled for the level of gameplay properties (e.g., there is no way to guarantee a balanced map, or maybe not even to guarantee one wherein all areas are accessible). An interesting approach is that of *Diorama*, a map generator for the open-source strategy game *Warzone 2100* that uses answer set programming. Some commercial games, such as those in the *Civilization* series, feature procedural map generation, but that is usually accomplished through simple methods, such as seeding islands in the middle of the ocean and letting them grow in random directions.

Ashlock *et al.* [56] proposed an indirect search-based method for landscape generation based on an evolvable L-system representation and used this approach to evolve fractal landscapes to fit specific shapes; however, no concern was given to the suitability of these landscapes for games. Some recent work has focused on integrating various dissimilar terrain generation algorithms into mixed-initiative models [6].



MeepEep
Level: 200
XP: 4699
Credits: 123364
Delsota Starcrusher III
Shield: 28000 / 28000
Armor: 29090 / 36000
Hull: 32000 / 32000

Sgt. Tolus [76]
Human Level: 76
Delsota Solarstriker IX
Shield: 7600 / 7600
Armor: 15200 / 15200
Hull: 30400 / 30400
Target: null

Fig. 4. *Galactic Arms Race* game, featuring online distributed interactive evolution of weapons.

5) Narrative and Storytelling: Many games are built around or contain some form of story/narrative—this is the case for most first-person shooters and platform games, and all role-playing games, but arguably not for many other types of games, such as matching tile games like *Tetris* (Alexey Pajitnov 1984) or *Bejeweled* (PopCap 2007) [57]. (It should be noted that there is a debate about to what degree all games contain or constitute narrative, and some people would argue that *Tetris* is a game with narrative [58].) Attempts to automatically generate narrative for games go back a few decades, and a variety of approaches have been developed for problems that can variously be described as online and offline. Some systems construct background stories and/or playable scenarios from scratch, whereas others are focused on controlling the actions of NPCs in response to the player character so as to fulfill dramatic goals. The approaches taken can variously be described as constructive and generate-and-test. The core mechanism in many of these systems is some version of classical AI planning [59]–[61] (including *Facade*, the most famous example of procedural storytelling [62]), though there are a few examples of search-based approaches [63].

Due to the sheer volume of work in this area, and the fact that most of it is not search-based in the sense we have defined above, we will not survey work on narrative and storytelling as part of this paper. The reader is referred to Wardrip-Fruin's recent book [64] for a history and critique of that field.

B. Optional Content

Because optional content is not always critical to the game (i.e., it is forgiving), it can sometimes support more creative exploration, as discussed in this section.

1) Weapons: Because combat is a common facet of modern games, weapons are well-suited for procedural generation. While most games that feature weapons require their usage for the player to make progress within the game, many let their players carry a number of the weapons at the same time, which means that each particular weapon is optional (i.e., a useless weapon will simply not be used, and exchanged at the next opportunity).

Hastings *et al.* [2], [33] developed a multiplayer game built on search-based PCG called *Galactic Arms Race*, in which players guide a spaceship through the sectors of a galaxy, engaging in firefights with enemies and collecting weapons. A fixed-size array of weapons can be carried, and the player can choose to exchange any particular weapon currently being carried each time a new weapon is found. Weapons are represented indirectly as variable-size vectors of real values, which are interpreted as connection topologies and weights for neural networks, which in turn control the particle systems that underlie the weapons [65]. The evaluation function is interactive, implicit and distributed. The fitness of each weapon depends on how often the various users logged onto the same server choose to fire the weapon relative to how long it sits unused in their weapons cache. This evaluation function is appealing because players in effect indicate their preferences implicitly by simply playing without needing to know the mechanics or even existence of the underlying evolutionary algorithm. The game is illustrated in Fig. 4.

The same authors [66] recently added a method that enables more directly player-controlled weapons generation to *Galactic Arms Race*. Through the same representation as that described above, this feature allows players to perturb individual genes of their choice within the genome of a chosen weapon. This new *weapons lab* feature in effect adds a kind of genetic engineering to the game, which the players must earn the right to access.

An interesting parallel in the world of commercial games is *Borderlands* (Gearbox Interactive 2009), a collaborative online

FPS in which all weapons are procedurally generated. However, there is no search-based process; rather, weapon parameters are simply selected at random, and the approximate efficacy of the weapon is capped at the current level of the player character.

2) Buildings: Martin *et al.* [67] designed a system for interactively evolving buildings for the prototype video game *Subversion*, in development by the commercial video game developer Introversion. In this game, whole cities are procedurally generated by the player, meaning that the individual buildings could be seen as optional content. The buildings are represented relatively indirectly in a custom mark-up language, which describes each building from the bottom up as a stack of three-dimensional objects. Each object is in turn a two-dimensional shape that is vertically extruded, and various transformations can be applied to objects or groups of objects. The explicit interactive evaluation function (which is similar to functions commonly used in evolutionary art [68]), works as follows: each "generation" the user selects two parent buildings and the system produces a new screen of 16 offspring buildings. Variation is achieved through both structurally recombining the parents and mutating the numerical parameters of the offspring.

Substantial work has been done on procedural modelling of architecture within the computer graphics community. For example, shape grammars have been invented that take advantage of the regularity of architectural feature to enable compact and artist-friendly representation of buildings [69], [70], as well as techniques for semiautomatically extracting the "building blocks" for use in such description from archetypical building models [71]. Such description languages, somewhat similar to L-systems, could conceivably be used as representations in future search-based approaches to building and city generation.

3) Camera Control: Many games feature an in-game virtual camera through which the player experiences the world. Camera control is defined as controlling the placement, angle and possibly other parameters of the depending on the state of the game. Camera control is an important content class for many game genres, such as 3-D platformers, where the player character is viewed from a third-person vantage point. We consider camera control to be optional content, as a game is typically playable, though more challenging, with suboptimal camera control. Nevertheless, camera control may be viewed as necessary content for some third-person games since, in certain occasions, a poor camera controller could make the game completely unplayable.

Camera control coupled with models of playing behavior may guide the generation of personalised camera profiles. Burelli and Yannakakis [72], [73] devised a method for controlling in-game camera movements to keep specified objects (or characters) in view and potentially other objects out of view, while ensuring smooth transitions. Potential camera configurations are evaluated by calculating the visibility of the selected objects. A system based on probabilistic roadmaps and artificial potential fields then smoothly moves the camera towards the constantly reoptimized best global position. The quality of camera positions may feed an SBPCG component which, in turn, can set the weighting parameters of various camera constraints according to a metaheuristic (e.g., a player model).

Yannakakis *et al.* [74] introduce the notion of affect-driven camera control within games by associating player affective states (e.g., challenge, fun, and frustration) to camera profiles and player physiology. The affective models are constructed using neuroevolutionary preference learning on questionnaire data from several players of a 3-D Pac-Man like game named Maze-Ball. Camera profiles are represented as a set of three parameters: *distance* and *height* from the player character and frame-to-frame coherence (camera speed in between frames). This way, personalized camera profiles can be generated to maximize a direct, data-driven evaluation function which is represented by the neural network predictor of emotion.

4) Trees: Talton *et al.* [75] developed a system for offline manual exploration of design spaces of 3-D models; their main prototype focuses on trees. This system lets users view a two-dimensional pane of tree models, and navigate the space by zooming in on different parts of the pane. At all times the user can see the tree models that are generated at the selected point as well as nearby point; in other words this approach is an interactive evaluation function. The tree models are represented as fixed-length vectors of real-numbers. As the vector dimensionality is always higher than two, the vector is mapped to the two-dimensional pane through dimensionality reduction and estimation of density. The underlying algorithms is therefore comparable to an estimation of distribution algorithm (EDA) with interactive fitness.

C. A Note on Chronology

The first published search-based PCG-related papers that we know of are Togelius *et al.*'s first paper on racing track evolution [42], published in 2006, Hastings *et al.*'s paper on NEAT Particles (a central part of *Galactic Arms Race*) [65], and Hom and Marks' paper on balanced board games [36]. Two publications on evolving rules for games (the paper by Togelius and Schmidhuber [38] and the Ph.D. thesis of Cameron Browne [3]) appeared in 2008; the work in these two papers was carried out independently of each other and independently of Hom and Marks' work, though Browne's work was started earlier. Most of the subsequent papers included in this survey, though not all, were in some way influenced by these earlier works, and generally acknowledge this influence within their bibliographies.

D. Summary

Of the 14 projects described above in this section that are unequivocally search-based:

- six use direct evaluation functions, four of those are theory-driven rather than data-driven;
- two use interactive and six use simulation-based evaluation functions;
- 12 use a single fitness dimension, or a fixed linear combination of several evaluation functions;
- six represent content as vectors of real numbers;
- four represent content as (expression) trees of some form;
- three represent content directly in matrices where the genotype is spatially isomorphic to the phenotype.

Some patterns are apparent. Most of the examples of evolving rules and puzzles represent the content as expression trees and all use simulation-based evaluation functions. This could be due to the inherent similarity of rules to program code, which is often represented in tree form in genetic programming, and the apparent hardness of devising direct evaluation functions for rules. On the other hand, levels and maps are mostly evaluated using direct evaluation functions and represented as vectors of real numbers. Only two studies use interactive evaluation functions, and only two use data-driven evaluation functions (based on player models). There does not seem to be any clear reason why these last two types of evaluation functions are not used more, nor why simulation-based evaluation functions are not used much outside of rule generation.

V. Outlook

As can be seen in the previous section, there are already a number of successful experiments in search-based PCG. About half of these were published in 2010, indicating that this field is currently drawing considerable interest from within the game AI and computational intelligence and games (CIG) communities.

By classifying these experiments according to the taxonomies presented in this paper, it can be seen both that: 1) though all are examples of SBPCG, they differ from each other in several important dimensions,; and 2) there is room for approaches other than those that have already been tried, both within the type of content being generated and the algorithmic approach to it generation. Indeed, given the large variety of game genres and types of game content out there, there is arguably plenty of low-hanging fruit for researchers interested in this field. At the same time, there are several hard and interesting research challenges. It is important that research is carried out both on those easier problems for which more or less immediate success is probable, and ideally that search-based PCG techniques are included in shipped games. But it is equally important that research continues on hard problems where we are currently nowhere near producing content of a sufficient quality for including in commercial games. Such research could both lead to viable content generators in the future, and help advance the science of game design. Below is an attempt to identify the major research challenges in SBPCG.

- *Which types of content are suitable to generate?* It is clear that some types of content are easier than others to generate using search or optimization algorithms. The overarching question is which types can be generated well enough, fast enough and reliably enough to be used in actual production games rather than the type of research prototypes that most of the papers above are based on. The answer will partly depend on whether the content will be generated offline, in which case generation speed and reliability is less important while quality can be emphasized, or online, in which case speed is very important but some aspects of quality might be sacrificed. The importance of reliability depends partly on whether the content generated is optional or necessary; when generating optional content, larger variations in quality are more acceptable.
- *How can we avoid catastrophic failure?* One of the main arguments against PCG in general from representatives of the games industry, at least when discussing online generation of necessary content, is lack of reliability—or more precisely, the risk of *catastrophic failure* [76]. Given the way most commercial games are designed, any risk of the player being presented with unplayable content is unacceptable. One response to this challenge is to invent new game designs for which all content is to some extent optional, or occasional unplayable content is otherwise tolerable. Another response is to ensure that all content is of sufficient quality. Such a guarantee might be possible through the content representation, though this approach would likely limit the diversity of content that can be generated. The evaluation function can also enforce a guarantee. For many content types, it is likely that better simulation-based evaluation functions can avoid catastrophic failure by automatically playing through the content thoroughly; however, that approach might be computationally prohibitively expensive.
- *How can we speed up content generation?* The computational expense of search-based PCG can be prohibitive for online generation, and sometimes even for offline generation. A key challenge is to speed up the generation process as much as possible. It is well-known that, depending on the representation and shape of the search space, some stochastic optimization algorithms are more efficient than others. For vectors of real numbers there is an especially large assortment of powerful optimizers available, including nonevolutionary techniques such as particle swarm optimization [20]. Regardless of representation, there are algorithms available that take good advantage of characteristics of the search space, such as the presence of constraints.
- *How is game content best represented?* For most content types, multiple representations are possible, as discussed in Section III-A. The most appropriate representation for each content generation problem is likely to vary depending on a number of factors, e.g., the desired novelty/reliability tradeoff. However, when designing a representation for a new problem, it would be useful to have a set of principles and best practices for content representation. The particular type of content being generated will also significantly affect the representational options available. Some types of content are easier to represent than others and the amount of expertise require to parameterize a particular class of content may impact the cost of creating a game around it.
- *How can player models be incorporated into evaluation functions?* With a few exceptions, the experiments discussed above use theory-driven evaluation functions, which assume that some particular feature of some content type provides a good playing experience for players in general. For many applications, it would be advantageous to move to data-driven evaluation functions based on experiments on real players, making it possible to adapt the content to optimize predicted fun for particular classes of players. The *Super Mario Bros* level generation and the Maze-Ball camera profile generation experiments discussed above suggests a way to base direct evaluation functions on recorded player preferences when the content is represented as vectors of real numbers, but how to do that for simulation-based evaluation functions or when content is represented in a less straightforward manner is an open research topic. Player affective and cognitive

models deriving from the fusion of multiple modalities of player input may provide some answers to the problem.

- *Can we combine interactive and theory-driven evaluation functions?* Interactive evaluation functions show great promise by representing the most accurate and relevant judgment of content quality, but they are not straightforward to implement for most content generation problems. Major outstanding issues are how to design games so that effective implicit evaluation functions can be included, and how to speed up the optimisation process given sparse human feedback. One way to achieve the latter could be to combine the interactive evaluation of content with data-driven evaluation functions, thereby allowing the two evaluation modes to inform each other.
- *Can we combine search-based PCG with top-down approaches?* Coupled with the right representation and evaluation function, a global optimization algorithm can be a formidable tool for content generation. However, one should be careful not to see everything as a nail just because one has a hammer; not every problem calls for the same tool, and sometimes several tools need to be combined to solve a problem. In particular, the hybridization of the form of bottom-up perspective taken by the search-based approach with the top-down perspective taken in AI planning (commonly used in narrative generation) could be very fruitful. It is currently not clear how these two perspectives would inform each other, but their respective merits make the case for attempting hybrid approaches quite powerful.
- *How can we best assess the quality and potential of content generators?* As the research field of procedural content generation continues to grow and diversify, it becomes ever more important to find ways of evaluating the quality of content generators. One way of doing this is to organize competitions where researchers submit their separate solutions to a common content generation problem (with a common API), and players play and rank the generated content. The recent Mario AI level generation competition is the first example of this. In this competition, participants submitted personalized level generators for a version of Super Mario Bros, and attendants at a scientific conference played levels generated just for them decided which of the freshly generated levels they liked best [77]. However, it is also important to assess other properties of content generators, such as characterizing their *expressive range*: the variation in the content generated a by a specific content generator [78]. Conversely, it is important to analyze the range of domains for which a particular content generator can be effective.

We believe that progress on these problems can be aided by experts from fields outside computational intelligence such as psychology, game design studies, human–computer interface design and affective computing, creating an opportunity for fruitful interdisciplinary collaboration. The potential gains from providing good solutions to all these challenges are significant: the invention of new game genres built on PCG, streamlining of the game development process, and deeper understanding of the mechanisms of human entertainment are all possible.

Acknowledgment

This paper builds on and extends a previously published conference paper by the same authors [1]. Compared to the previous paper, this paper features an updated taxonomy, a substantially expanded survey section, and an expanded discussion on future research directions and challenges.

The authors would like to thank the anonymous reviewers for their substantial and useful comments. They would also like to thank all the participants in the discussions in the Procedural Content Generation Google Group.

References

[1] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation," in *Proc. EvoAppl.*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 2010, vol. 6024.

[2] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Automatic content generation in the galactic arms race video game," *IEEE Trans. Comput. Intell. AI Games*, vol. 1, no. 4, pp. 245–263, Dec. 2010.

[3] C. Browne, "Automatic generation and evaluation of recombination games," Ph.D. dissertation, Queensland Univ. Technol., Queensland, Australia, 2008.

[4] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling player experience for content creation," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 1, pp. 54–67, Mar. 2010.

[5] A. M. Smith and M. Mateas, "Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games," in *Proc. IEEE Conf. Comput. Intell. Games*, 2010, pp. 273–280.

[6] R. M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra, "Integrating procedural generation and manual editing of virtual worlds," in *Proc. ACM Foundations Digital Games*. New York: ACM Press, 2010.

[7] R. Miikkulainen, B. D. Bryant, R. Cornelius, I. V. Karpov, K. O. Stanley, and C. H. Yong, "Computational intelligence in games," in *Computational Intelligence: Principles and Practice*, G. Y. Yen and D. B. Fogel, Eds. Piscataway, NJ: IEEE, 2006.

[8] S. M. Lucas and G. Kendall, "Evolutionary computation and games," *IEEE Comput. Intell. Mag.*, vol. 1, pp. 10–18, 2006.

[9] S. Rabin*, AI Game Programming Wisdom*. Boston, MA: Charles River Media, 2002.

[10] D. M. Bourg and G. Seemann*, AI for Game Developers*. Sebastopol, CA: O'Reilly, 2004.

[11] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley, *Texturing and Modeling: A Procedural Approach (Third Edition)*. San Mateo, CA: Morgan Kaufmann, 2002.

[12] J. Whitehead, "Towards procedural decorative ornamentation in games," in *Proc. FDG Workshop Procedural Content Generation*, 2010.

[13] G. S. P. Miller, "The definition and rendering of terrain maps," *Proc. SIGGRAPH*, vol. 20, 1986.

[14] B. W. Kernighan and R. Pike*, The Practice of Programming*. Reading, MA: Addison-Wesley, 1999.

[15] F. Pachet, "Beyond the cybernetic fantasy jam: The continuator," *IEEE Comput. Graph. Appl.*, vol. 24, no. 1, pp. 31–35, 2004.

[16] K. Sims, "Artificial evolution for computer graphics," *Proc. SIGGRAPH*, vol. 25, no. 4, pp. 319–328, 1991.

[17] J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, and K. O. Stanley, "Picbreeder: Evolving pictures collaboratively online," in *Proc. 26th Annu. SIGCHI Conf. Human Factors Comput. Syst.*, New York, 2008, pp. 1759–1768.

[18] P. Machado and A. Cardoso, "Computing aesthetics," in *Proc. Brazilian Symp. Artif. Intell.*, 1998, pp. 219–229.

[19] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671680–671680, 1983.

[20] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Conf. Neural Netw.*, 1995, pp. 1942–1948.

[21] M. Harman and B. F. Jones, "Search-based software engineering," *Inf. Softw. Technol.*, vol. 43, pp. 833–839, 2001.

[22] P. McMinn, "Search-based software test data generation: A survey," *Softw. Testing Verif. Reliab.*, vol. 14, pp. 105–156, 2004.

[23] P. J. Bentley and S. Kumar, "The ways to grow designs: A comparison of embryogenies for an evolutionary design problem," in *Proc. Genet. Evol. Comput. Conf.*, 1999, pp. 35–43.

[24] G. S. Hornby and J. B. Pollack, "The advantages of generative grammatical encodings for physical design," in *Proc. Congr. Evol. Comput.*, 2001 [Online]. Available: http://demo.cs.brandeis.edu/papers/long.html#hornby_cec01

[25] K. O. Stanley, "Compositional pattern producing networks: A novel abstraction of development," *Genet. Program. Evolvable Mach. (Special Issue Develop. Syst.)*, vol. 8, no. 2, pp. 131–162, 2007.

[26] K. O. Stanley and R. Miikkulainen, "A taxonomy for artificial embryogeny," *Artif. Life*, vol. 9, no. 2, pp. 93–130, 2003.

[27] F. Rothlauf, *Representations for Genetic and Evolutionary Algorithms*. Heidelberg, Germany: Springer-Verlag, 2006.

[28] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, pp. 99–127, 2002.

[29] D. Ashlock, T. Manikas, and K. Ashenayi, "Evolving a diverse collection of robot path planning problems," in *Proc. Congr. Evol. Comput.*, 2006, pp. 6728–6735.

[30] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Hoboken, NJ: Wiley-Interscience, 2001.

[31] G. N. Yannakakis, "How to model and augment player satisfaction: A review," in *Proc. 1st Workshop Child Comput. Interact.*, Chania, Crete, Oct. 2008.

[32] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Trans. Affective Comput.*, 2011, to be published.

[33] E. Hastings, R. Guha, and K. O. Stanley, "Evolving content in the galactic arms race video game," in *Proc. IEEE Symp. Comput. Intell. Games*, 2009, pp. 241–248.

[34] Lindenmayer, "Mathematical models for cellular interaction in development parts I and II," *J. Theor. Biol.*, vol. 18, pp. 280–299, 1968.

[35] P. Prusinkiewicz, "Graphical applications of l-systems," *Proc. Graph. Interface/Vis. Interface*, pp. 247–253, 1986.

[36] V. Hom and J. Marks, "Automatic design of balanced board games," in *Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain.*, 2007, pp. 25–30.

[37] J. Mallett and M. Lefler, "Zillions of Games," 1998 [Online]. Available: http://www.zillions-of-games.com

[38] J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in *Proc. IEEE Symp. Comput. Intell. Games*, 2008, pp. 111–118.

[39] C. Salge and T. Mahlmann, "Relevant information as a formalised approach to evaluate game mechanics," in *Proc. IEEE Conf. Comput. Intell. Games*, 2010, pp. 281–288.

[40] D. Oranchak, "Evolutionary algorithm for generation of entertaining shinro logic puzzles," in *Proc. EvoAppl.*, 2010.

[41] D. Ashlock, "Automatic generation of game elements via evolution," in *Proc. IEEE Conf. Comput. Intell. Games*, 2010, pp. 289–296.

[42] J. Togelius, R. De Nardi, and S. M. Lucas, "Making racing fun through player modeling and track evolution," in *Proc. SAB Workshop Adapt. Approach. Optimizing Player Satisfaction*, 2006.

[43] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation in racing games," in *Proc. IEEE Symp. Comput. Intell. Games*, 2007, pp. 252–259.

[44] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling player experience in Super Mario Bros," in *Proc. IEEE Symp. Comput. Intell. Games.*, 2009, pp. 132–139.

[45] N. Sorenson and P. Pasquier, "Towards a generic framework for automated video game level creation," in *Proc. Eur. Conf. Appl. Evol. Comput.*, 2010, vol. 6024, pp. 130–139.

[46] S. O. Kimbrough, G. J. Koehler, M. Lu, and D. H. Wood, "On a feasible-infeasible two-population (fi-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch," *Eur. J. Operat. Res.*, vol. 190, 2008.

[47] M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin, "Polymorph: A model for dynamic level generation," in *Proc. Artif. Intell. Interact. Digital Entertain.*, 2010.

[48] M. Frade, F. F. de Vega, and C. Cotta, "Evolution of artificial terrains for video games based on accessibility," in *Proc. Eur. Conf. Appl. Evol. Comput.*, 2010, vol. 6024, pp. 90–99.

[49] J. Togelius, M. Preuss, and G. N. Yannakakis, "Towards multiobjective procedural map generation," in *Proc. FDG Workshop on Procedural Content Generation*, 2010.

[50] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G. N. Yannakakis, "Multiobjective exploration of the starcraft map space," in *Proc. IEEE Conf. Comput. Intell. Games*, 2010, pp. 265–272.

[51] N. Beume, B. Naujoks, and M. Emmerich, "SMS-EMOA: Multiobjective selection based on dominated hypervolume," *Eur. J. Oper. Res.*, vol. 181, no. 3, pp. 1653–1669, 2007.

[52] A. Fournier, D. Fussell, and L. Carpenter, "Computer rendering of stochastic models," *Commun. ACM*, vol. 25, no. 6, 1982.

[53] J. Olsen, "Realtime procedural terrain generation," Univ. Southern Denmark, Tech. Rep., 2004.

[54] J. Doran and I. Parberry, "Controllable procedural terrain generation using software agents," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 2, pp. 111–119, Jun. 2010.

[55] L. Johnson, G. N. Yannakakis, and J. Togelius, "Cellular automata for real-time generation of infinite cave levels," in *Proc. ACM Found. Digit. Games.*, Jun. 2010.

[56] D. A. Ashlock, S. P. Gent, and K. M. Bryden, "Evolution of l-systems for compact virtual landscape generation," in *Proc. IEEE Congr. Evol. Comput.*, 2005, pp. 2760–2767.

[57] J. Juul, "Swap adjacent gems to make sets of three: A history of matching tile games," *Artifact J.*, vol. 2, 2007.

[58] G. Frasca, "Ludologists love stories, too: Notes from a debate that never took place," in *Proc. Level Up: Digit. Games Res. Conf.*, 2003.

[59] R. Aylett, J. Dias, and A. Paiva, "An affectively-driven planner for synthetic characters," in *Proc. ICAPS*, 2006.

[60] M. O. Riedl and N. Sugandh, "Story planning with vignettes: Toward overcoming the content production bottleneck," in *Proc. 1st Joint Int. Conf. Interact. Digit. Storytelling*, Erfurt, Germany, 2008, pp. 168–179.

[61] Y.-G. Cheong and R. M. Young, "A computational model of narrative generation for suspense," in *Proc. AAAI Comput. Aesthetic Workshop*, 2006.

[62] M. Mateas and A. Stern, "Facade: An experiment in building a fully-realized interactive drama," in *Proc. Game Develop. Conf.*, 2003.

[63] M. J. Nelson, C. Ashmore, and M. Mateas, "Authoring an interactive narrative with declarative optimization-based drama management," in *Proc. Artif. Intell. Interact. Digit. Entertain. Int. Conf.*, 2006.

[64] N. Wardrip-Fruin, *Expressive Process.*. Cambridge, MA: MIT Press, 2009.

[65] E. Hastings, R. Guha, and K. O. Stanley, "Neat particles: Design, representation, and animation of particle system effects," in *Proc. IEEE Symp. Comput. Intell. Games*, 2007, pp. 154–160.

[66] E. J. Hastings and K. O. Stanley, "Interactive genetic engineering of evolved video game content," in *Proc. FDG Workshop Procedural Content Generat.*, 2010.

[67] A. Martin, A. Lim, S. Colton, and C. Browne, "Evolving 3d buildings for the prototype video game subversion," in *Proc. EvoApplications*, 2010.

[68] H. Takagi, "Interactive evolutionary computation: Fusion of the capacities of EC optimization and human evaluation," *Proc. IEEE*, vol. 89, no. 9, pp. 1275–1296, Sep. 2001.

[69] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. V. Gool, "Procedural modeling of buildings," *ACM Trans. Graph.*, vol. 25, pp. 614–623, 2006.

[70] J. Golding, "Building blocks: Artist driven procedural buildings," in *Proc. Present. Game Develop. Conf.*, 2010.

[71] M. Bokeloh, M. Wand, and H.-P. Seidel, "A connection between partial symmetry and inverse procedural modeling," in *Proc. SIGGRAPH*, 2010.

[72] P. Burelli and G. N. Yannakakis, "Combining local and global optimisation for virtual camera control," in *Proc. IEEE Conf. Comput. Intell. Games*, Copenhagen, Denmark, Aug. 2010, pp. 401–403.

[73] P. Burelli and G. N. Yannakakis, "Global search for occlusion minimization in virtual camera control," in *Proc. IEEE World Congr. Comput. Intell.*, Barcelona, Spain, Jul. 2010, pp. 2718–2725.

[74] G. N. Yannakakis, H. P. Martínez, and A. Jhala, "Towards affective camera control in games," *User Model. and User-Adapted Interact.*, vol. 20, no. 4, pp. 313–340, 2010.

[75] J. O. Talton, D. Gibson, L. Yang, P. Hanrahan, and V. Koltun, "Exploratory modeling with collaborative design spaces," *ACM Trans. Graph.*, vol. 28, 2009.

[76] K. Compton, "Remarks during a panel session at the FDG workshop on PCG," 2010.

[77] N. Shaker, J. Togelius, G. N. Yannakakis, B. Weber, T. Shimizu, T. Hashiyama, N. Sorenson, P. Pasquier, P. Mawhorter, G. Takahashi, G. Smith, and R. Baumgarten, "The 2010 Mario AI Championship: Level Generation Track".

[78] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proc. FDG Workshop Procedural Content Generation*, 2010.

Julian Togelius received the B.A. degree in philosophy from Lund University, Lund, Sweden, in 2002, the M.Sc. degree in evolutionary and adaptive systems from University of Sussex, Sussex, U.K., in 2003, and the Ph.D. in computer science from University of Essex, Essex, U.K., in 2007.

He is currently an Assistant Professor at the IT University of Copenhagen (ITU), Copenhagen, Denmark. Before joining the ITU in 2009, he was a Postdoctoral Researcher at IDSIA in Lugano. His research interests include applications of computational intelligence in games, procedural content generation, automatic game design, evolutionary computation, and reinforcement learning; he has around 50 papers in journals and conferences about these topics.

Dr. Togelius is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES and is the current Chair of the IEEE CIS Technical Committee on Games.

Georgios N. Yannakakis received the five-year Dipl. degree in production engineering and management and the M.Sc. degree in financial engineering from the Technical University of Crete, Crete, Greece, in 1999 and 2001, respectively, and the Ph.D. degree in informatics from the University of Edinburgh, Edinburgh, U.K., in 2005.

He is currently an Associate Professor at the IT University of Copenhagen, Copenhagen, Denmark. Prior to joining the Center for Computer Games Research, ITU, in 2007, he was a Postdoctoral Researcher at the Mærsk Mc-Kinney Møller Institute, University of Southern Denmark. His research interests include user modeling, neuroevolution, computational intelligence in computer games, cognitive modeling, and affective computing, emergent cooperation and artificial life. He has published around 60 journal and international conference papers in the aforementioned fields.

Dr. Yannakakis is an Associate Editor of the IEEE TRANSACTIONS ON AFFECTIVE COMPUTING and the IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, and the Chair of the IEEE CIS Task Force on Player Satisfaction Modeling.

Kenneth O. Stanley received the B.S.E. degree from the University of Pennsylvania, University Park, in 1997 and the Ph.D. degree from the University of Texas at Austin, Austin, in 2004.

He is currently an Assistant Professor in the Department of Electrical Engineering and Computer Science at the University of Central Florida, Orlando, FL. He is an inventor of the Neuroevolution of Augmenting Topologies (NEAT) and HyperNEAT algorithms for evolving complex artificial neural networks. His main research interests are neuroevolution (i.e., evolving neural networks), generative and developmental systems, coevolution, machine learning for video games, and interactive evolution.

Dr. Stanley is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES and on the editorial board of the *Evolutionary Computation Journal*. He has won best paper awards for his work on NEAT, NERO, NEAT Drummer, HyperNEAT, novelty search, and Galactic Arms Race.

Cameron Browne received the Ph.D. degree in computer science from the Faculty of Information Technology, QUT, Brisbane, Australia, in 2008.

He is currently a Research Fellow in the Computational Creativity Group at Imperial College London, London, U.K. His research interests include the application of MCTS methods to creative domains such as game design, art, and music.

Dr. Browne was Canon Research Australia's Inventor of the Year for 1998 and won the QUT Deans Award for Outstanding Thesis of 2008.