Towards Pattern-Based Mixed-Initiative Dungeon Generation

Alexander Baldwin
Faculty of Technology and Society, Malmö University
Nordenskiöldsgatan 1
Malmö, Sweden 20506
alexander.baldwin@mah.se

Steve Dahlskog
Faculty of Technology and Society, Malmö University
Nordenskiöldsgatan 1
Malmö, Sweden 20506
steve.dahlskog@mah.se

Jose M. Font
Faculty of Technology and Society, Malmö University
Nordenskiöldsgatan 1
Malmö, Sweden 20506
jose.font@mah.se

Johan Holmberg
Faculty of Technology and Society, Malmö University
Nordenskiöldsgatan 1
Malmö, Sweden 20506
johan.holmberg@mah.se

ABSTRACT

Mixed-initiative Procedural Content Generation uses algorithms to assist human designers in the collaborative creation of game content. Different mixed-initiative approaches use different methods to engage with the design material while supporting the designer's intentions. However, the designer runs the risk of misunderstanding the system's abilities and how to control them. In order to limit miscommunication during the design process, heuristics could be applied.

In this paper we present a mixed-initiative tool for evolving dungeons with the aid of game design patterns as heuristics. The tool, the Evolutionary Dungeon Designer, uses a genetic algorithm that searches for levels containing game design patterns on two hierarchical levels of abstraction to express more complex gameplay in the game level.

We evaluate the tool through a series of lab experiments and a user study conducted with professional game developers. Our results demonstrate that we are able to control the generation of the different patterns with the aid of design pattern-related input parameters, as well as identifying a number of features a design pattern-based mixed-initiative tool could benefit from.

CCS CONCEPTS

• Theory of computation → Evolutionary algorithms; **• Applied computing → Computer games**; **• Software and its engineering → Interactive games**; **• Computing methodologies** → *Search methodologies*;

KEYWORDS

Procedural Content Generation, Evolutionary Algorithms, Game Design Patterns, Mixed-Initiative Design.

ACM Reference format:
Alexander Baldwin, Steve Dahlskog, Jose M. Font, and Johan Holmberg. 2017. Towards Pattern-Based Mixed-Initiative Dungeon Generation. In *Proceedings of FDG'17, Hyannis, MA, USA, August 14-17, 2017,* 10 pages.
https://doi.org/10.1145/3102071.3110572

1 INTRODUCTION

Since its first implementations in the early 1980's [6, 34], Procedural Content Generation (PCG) has already become one of the common techniques applied to the automatic generation of content for games [28], such as weapons [16], maps [22], race tracks [32], levels [13], and even game rules [7]. Cost reduction, in-game content generation, and user-tailored content can be found among its main advantages [29]. Search-based PCG is defined as the family of PCG methods using a search-based approach that is guided towards a predefined set of goals specified by a *fitness* or *evaluation function* [33].

In particular, PCG research has produced several different approaches for generating dungeon levels, a popular level design archetype found in games such as the *Legend of Zelda* [25] series and *Diablo* [5]. These approaches include both constructive and search-based approaches: space partitioning, agent-based methods [28], cellular automata [18], grammars [13, 35], and mutation-based graph evolution [19].

Since game and level designers are potential users of PCG-tools, mixed-initiative approaches arose as a way of involving human input in the automated generation process [26, 33, 36]. Designers act as an active part of the search process. In some cases, such as in *Galactic Arms Race* [16] or *Unexplored* [14], players act as designers implicitly as they play the game. On the other hand, authoring tools like *Ropposum* [27], *Tanagra* [31], or *The Sentient Sketchbook* [22] ask the user to explicitly input their design criteria at different stages of the evolutionary process.

Some methods have combined PCG with Game Design Patterns (GDP) [4], as semi-formal interdependent descriptions of commonly reoccurring parts of the design of a game that concern gameplay. Specific taxonomies to different game genres have been developed, including platform games [9], first person shooters [17], and role playing games [8]. Rather than using design patterns as building blocks for content creation, Dahlskog and Togelius [10] suggest applying design patterns as objectives for a PCG generator. Such approaches may provide game designers a better understanding of the mixed-initiative generation, enhancing creativity while giving designers more control over the whole process.

In this paper we present the current development of the Evolutionary Dungeon Designer (EDD) [2], a pattern-based mixed-initiative PCG tool where the user takes the role of a level designer to design dungeons for fantasy role playing games. Section 2 summarizes the previous contributions to this work, Section 3 describes the current status of the EDD, while Section 4 presents the results from both laboratory experiments and a user study. Our conclusions and suggestions for future work are presented in Section 5.

2 PREVIOUS WORK

In this section, we present a summary of the preceding research on game design patterns and mixed-initiative level generation, to which the presented current state of the Evolutionary Dungeon Designer acts as a continuation.

2.1 Game Design Patterns

Initially derived from *Design Patterns* [1] in architecture, Kreimeier [21] proposed the use of Game Design Patterns (GDP) as a tool to design and understand games. Björk & Holopainen gathered a pattern collection bordering on 300 patterns [3, 4] covering a variety of aspects including goals and interaction patterns in games. Over the years GDP have been explored in more specific domains, like the game genres First Person Shooters [17] and Role-Playing Games [30]. Practical uses of GDP include teaching and communicating design aspects of games. However, GDP can of course be used in more concrete design activities like brainstorming, exploration and fine-tuning of actual design decisions.

Björk & Holopainen argue that patterns collectively create hierarchies, due to the presence of patterns that instantiate or are instantiated by other patterns. Other effects in a given design instance may cause patterns to modulate other patterns. As a consequence Björk & Holopainen use the terms: "higher-level" and "lower-level" patterns to indicate these relations [4].

Previously, PCG and GDP have been used together to generate levels for the 2D-platform game *Super Mario Bros.*(SMB) [10–12]. An approach using a hierarchy of patterns was suggested [11], in order to generate content that is similar to the orginal game. The hierarchy consist of three levels: *micro-*, *meso-*, and *macro-*patterns, where higher level patterns consist of combinations of low level patterns and express more complex gameplay scenarios. This provides structure through a rather open design heuristic when using the GDP as objectives in a search-based PCG approach. Dahlskog et al. define a collection of patterns from the dungeon domain [8], with the goal of providing a sufficient level of detail to use them in a generator – one example of such generator is our previous work on EDD [2].

2.2 The Evolutionary World Designer

The *Evolutionary World Designer* (EWD) [15] presents a grammatical evolution system as a way of describing and evolving world maps for adventure-like games using context-free grammars. This search-based approach that avoids generating infeasible individuals, guaranteeing solvability of the levels and allowing some level of control to the designer.

A world is represented as an acyclic graph whose nodes are interconnected sections of it. Each node then encloses a specific dungeon represented as a separated cyclic graph, whose nodes are the rooms that compose it. The world graph is first evolved, then a separate evolutionary process is run for each of the dungeon graphs to be generated.

The designer sets up initial parameters regarding the number of world sections, as well as the number of enemies, treasures, and rooms per section. These rooms are automatically populated with enemies, treasures, and keys that grant access to the subsequent world sections, but the exact placement of these elements inside the room is missing, as well as the placement of any kind of wall structures or aesthetic components.

2.3 The Evolutionary Dungeon Designer

The Evolutionary Dungeon Designer (EDD) [2], goes one step further than the EWD, generating the specific content for each of the rooms in the generated dungeons. This is implemented by a FI-2Pop GA [20], where game design patterns are used both as input parameters and objectives. The first release of this tool involves the following micro-patterns in the generated rooms: enemy, treasure, chamber, corridor, connector, entrance, and door. Additionally, a *playability constraint* is defined so that a playable room must contain at least one treasure and one enemy, as well as paths between the entrance and all other doors, treasure and enemies.

Given this, the generated rooms are evolved in two different populations (feasible and infeasible), which are evaluated according to different fitness functions. Feasible rooms are evaluated by assigning a quality measure to each pattern, which allows designers to achieve finer control over the nature of the patterns found in the generated room. Infeasible rooms are evaluated according to how close they are to fulfilling the playability constraint. The experiments on this version were satisfactory at validating the system in terms of fitness optimization, pattern detection, and solution diversity. A sufficient level of control over the results was also provided to the designer.

In this work, we present an extended version of EDD which involves meso-patterns as a core feature of the room generation process, and also presents the generated rooms in a mixed-initiative tool that allows a greater degree of interaction between the designer and the generator in the creation process.

3 THE MIXED-INITIATIVE EVOLUTIONARY DUNGEON DESIGNER

3.1 System Overview and Workflow

EDD is a mixed-initiative dungeon generator that takes advantage of the combination of artificial intelligence techniques with the natural intuition and creativity of human designers. Simplicity was a primary consideration in the design of this tool, in order to reduce the need for laborious and potentially unintuitive tweaking of the underlying generation algorithms.

For this reason, EDD provides a workflow in which the user is presented with a number of widely varied suggestions of possible

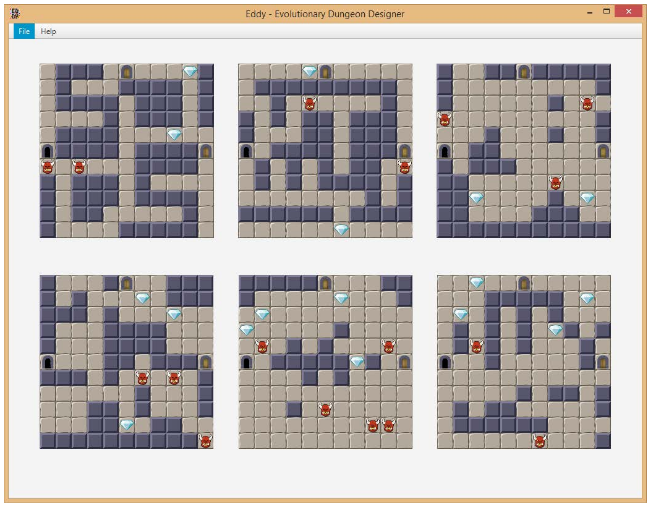

Figure 1: The room suggestion view.

maps and can then follow an iterative process of regeneration and manual refinement to create a room that satisfies their expectations. The tool is divided into two distinct interfaces: the room suggestion view (Figure 1) and the room editing view (Figure 2).

After opening EDD, the room suggestion view presents six rooms generated using different configuration files selected from a small pool of diverse candidates, with the goal of acting as a starting point for the mixed-initiative generation process. If none of the generated rooms are interesting, six new ones can be generated. After clicking on one of the suggestions, it is loaded in the room editing view.

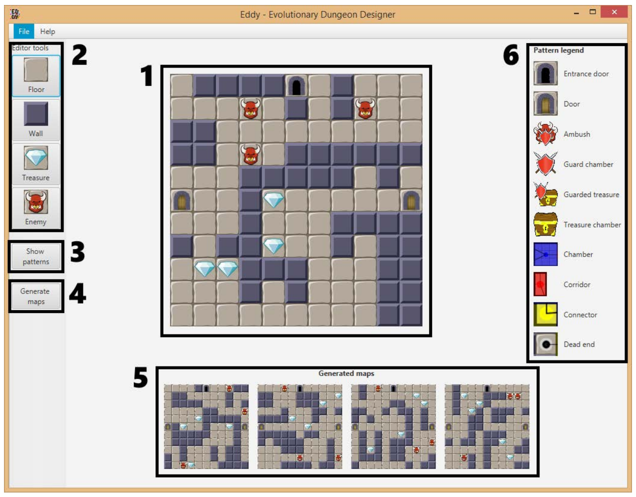

Figure 2: The room editing view.

The editing view is the main interface for generating new room suggestions, manually editing rooms and viewing the game design patterns contained in a generated or edited room. The basic functionality of this view is highlighted in Figure 2, including:

(1) A larger, manually editable version of the selected map.
(2) A toolbox of tiles that can be painted onto the map. Note that only floor, walls, treasure and enemies can be placed in this fashion. Door positions are fixed and cannot be changed.
(3) The *Show patterns* button overlays the map with a view of the patterns it contains – see Figure 3 for more details.
(4) The *Generate maps* button will generate four maps based on the map being edited.
(5) Newly generated maps appear here and can be selected to replace the currently chosen map in order to edit them.
(6) A legend explaining the meaning of the icons shown in the pattern overlay.

The four suggested rooms are generated by following two strategies based on the current map. Two maps are generated using a new configuration based on the proportion and type of design patterns detected in the current map. The remaining two suggestions are also generated using a configuration computed in the same way, but each run is seeded with individuals that are mutated versions of the current map.

By employing these two types of mutation we aim to produce two different types of suggestions: rooms containing a similar distribution of game design patterns to the original room for the former approach and rooms that visually resemble the original room and contain a similar distribution of game design patterns for the latter approach.

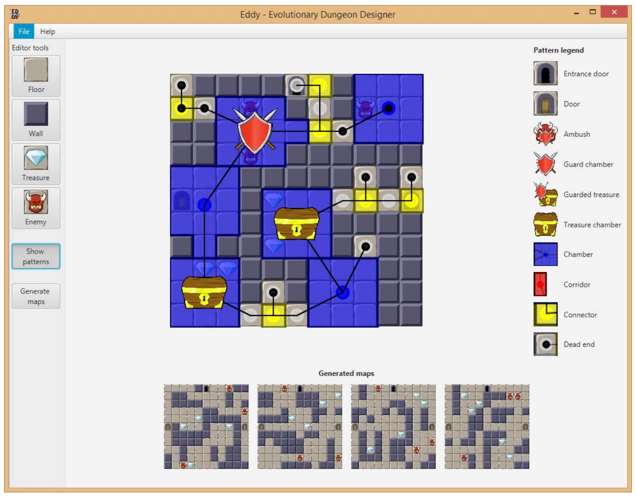

Figure 3: The room editing view with pattern overlay enabled. Spacial micro-patterns are displayed as blocks of colour, blue for chambers, red for corridors and yellow for connectors. Black lines show the pattern-graph for the room, where meso-pattern icons are overlaid.

As an aid to the understanding of the gameplay afforded by generated rooms, users can view a pattern overlay, as shown in Figure 3. This overlay is automatically updated as the room is edited, so edits can be made with the overlay visible, providing rapid feedback on how changes to the layout of the room affect the resulting gameplay. Rooms are edited by changing existing tiles to one of the other tile types: floor, wall, treasure and enemy. Door positions are locked and cannot be changed (except by starting a new generation process) or overwritten. In the case that the user creates an unplayable room (that is, the room does not satisfy the *playability constraint* [2]), the room is highlighted with a red border, but new rooms may still be generated based on this unplayable room.

3.2 Pattern Detection

EDD optimizes rooms for the occurrence of micro- and meso-patterns, while excluding macro-patterns, since macro-patterns are generally on the scale of entire levels [8]. As meso-patterns are built upon combinations of micro-patterns, detecting these patterns is a process in multiple stages in which we first detect micro-patterns and then use these to identify the meso-patterns that contain them.

We differentiate between **micro-patterns** that deal with the layout of space (passable tiles) in the level and those that deal with the contents of that space. As mentioned in Section 2.3, in the former category – *spacial micro-patterns* – EDD detects *corridor*, *connector* and *room*, (referred to room as *chamber* for the sake of clarity). The second category – *inventorial micro-patterns* – consists of the remaining passable tile types: *door*, *treasure* and *enemy*. How EDD detects and involves these micro-patterns in the evaluation function is not the focus of the present work, being related in detail in [2].

In order to detect **meso-patterns**, which are defined by how micro-patterns or other meso-patterns relate to each other in space, or by what inventorial micro-patterns they contain, we require a means of analysing these properties of the detected micro-patterns. To do this we construct a pattern-graph (using a breadth-first search starting at the room's entrance), where vertices are spacial micro-patterns and edges represent adjacency between the spacial patterns' tiles. The constructed graph (Figure 3) can be cyclic and two vertices (patterns) may be connected by multiple edges, signifying that there exist multiple direct paths between the two patterns. Each edge is weighted according to the number of adjacent tiles and will generally have a weight of either one or two.

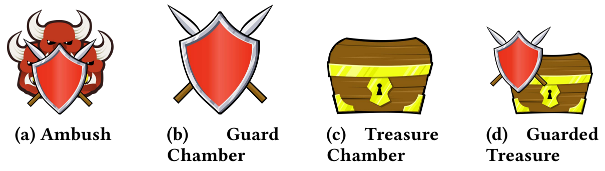

Figure 4: Icons shown in EDD to represent the presence of four of the five meso-pattern types.

The detected meso-patterns are *treasure chamber*, *guard chamber*, *ambush*, *dead end*, and *guarded treasure*. These are highlighted by icons representing the meso-pattern type (Figure 4). The icons are drawn in the location of the chamber contained in the pattern or, in the case of the guarded treasure, which includes multiple chambers, the icon is drawn in the included treasure chamber. Note that dead ends are not represented by an icon, since they can cover a large area of the map. They instead are represented by black dots in the center of the patterns that belong to a dead end.

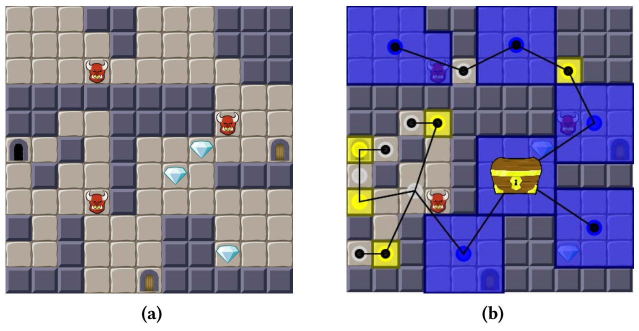

Figure 5: a) A room that contains a treasure chamber with exactly two treasures. b) The pattern-overlay view with the corresponding icon.

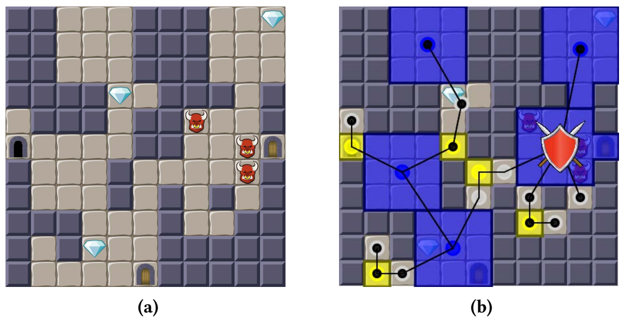

Figure 6: a) A room that contains a guard chamber with three guards. b) The pattern-overlay view with the corresponding icon.

3.2.1 *Treasure Chamber.*

We define a *treasure chamber* as a chamber containing two or more treasure tiles and no enemies. While this can be interpreted as a chamber containing multiple discrete treasures, this pattern could also be used to denote a room containing a single particularly valuable or useful item and could be seen as similar to The Binding of Isaac's item rooms, which are guaranteed to contain a special item for the player to collect [24].

Quality for treasure chambers is defined by how close the number of treasures in the chamber is to a user-defined target number, $T_{chambertreasure}$ and is calculated (for a treasure chamber p) as:

$$Q(p) = \max\left\{0, 1 - \left|\frac{N_t - T_{chambertreasure}}{T_{chambertreasure}}\right|\right\} \quad (1)$$

By adjusting the target number of treasure tiles in the whole room and the desired number of treasures in a treasure chamber, the user is given some control over how treasure is distributed throughout the room.

3.2.2 *Guard Chamber.*

We define a *guard chamber* as a chamber containing two or more enemy tiles and no treasures. As with the treasure chamber pattern, this could be interpreted literally as a

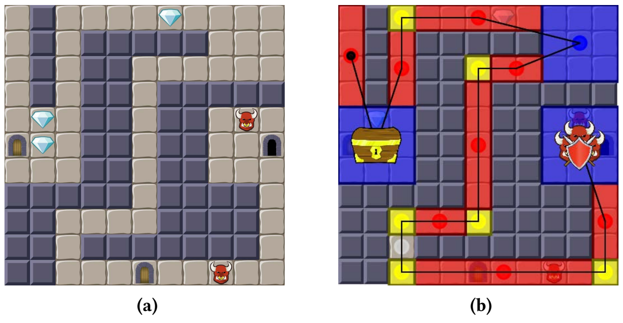

Figure 7: a) A room that contains an ambush: a chamber containing a guard and the room's entrance. b) The pattern-overlay view with the corresponding icon. Note that the map also contains a treasure chamber on the left.

number of discrete enemies or be used to represent particularly challenging enemies (such as bosses or mini-bosses), depending on the number of enemies in the chamber.

Quality for guard chambers is defined by how close the number of guards in the chamber is to a user-defined target number, $T_{chamberenemies}$ and is calculated (for a guard chamber p) as:

$$Q(p) = \max\left\{0, 1 - \left|\frac{N_e - T_{chamberenemies}}{T_{chamberenemies}}\right|\right\} \tag{2}$$

3.2.3 Ambush. We define an *ambush* as a chamber containing both the room's main entrance and at least one enemy. Conceptually, this represents a room in which the player will have to engage in combat (or flee) immediately upon entering and, depending on the number of enemies, may significantly increase the challenge of the room.

Quality for ambushes is defined by how close the number of guards in the chamber is to a user-defined target number, $T_{ambushenemies}$ and is calculated (for an ambush p) as:

$$Q(p) = \max\left\{0, 1 - \left|\frac{N_e - T_{ambushenemies}}{T_{ambushenemies}}\right|\right\} \tag{3}$$

3.2.4 Dead End. Dahlskog et al. define dead ends as "locations from which players must move through previously explored areas." [8]. While this definition invites a number of potential interpretations, we consider a spacial micro-pattern to be part of a dead end if and only if every path from that pattern to the room's critical path passes through a single pattern (where the critical path for the pattern graph is the shortest path connecting all the room's doors). To find dead ends, we first calculate the pattern graph's critical path. Then, for every pattern in the critical path, we execute a breadth first search of adjacent patterns not in the critical path. For each such pattern, if the edge traversed to reach it is not in a cycle, then all other connected unvisited patterns not in the critical path make up a dead end.

Dahlskog et al. also note that an area might be considered less of a dead end if it contains interesting gameplay [8]. We encapsulate this idea in the quality measure for dead ends by rewarding dead ends that contain other meso-patterns (such as a dead end that

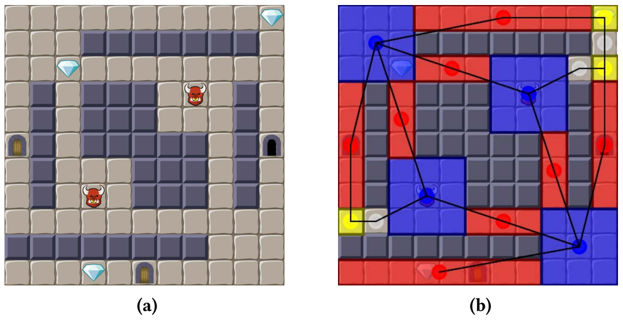

Figure 8: a) A room generated with a high penalty for dead ends. The pattern graph for this room, shown in b), indicates that the room contains no dead ends. Multiple possible paths exist connecting the three doors, giving the player the option of bypassing the enemies.

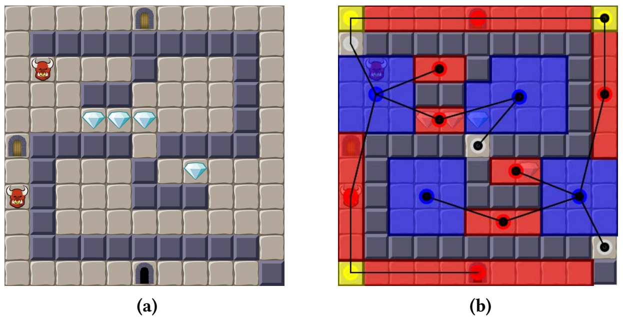

Figure 9: a) A room generated with no penalty for dead ends. The pattern graph for this room, shown in b) indicates that most of the room's patterns are included in dead ends (shown by the black dot in the centre of these patterns). There is only one path between the three doors and large parts of the room are optional.

contains a treasure chamber or a guard chamber). However, it may in some cases be desirable to create dead ends that do not contain anything interesting, in order to create a maze-like room layout. For this reason we allow the user to specify the proportion of patterns in a dead end that should also be contained in other meso-patterns and use this to define the function $Density(p)$ for a dead end p as:

$$Density(p) = 1 - \frac{1}{\max\{T_{DEdensity}, 1 - T_{DEdensity}\}} \cdot \left|\frac{N_{mp}}{N_{sp}} - T_{DEdensity}\right| \tag{4}$$

where N_{mp} is the number of meso-patterns present in the dead end and N_{sp} is the total number of spacial patterns present in the dead end.

The presence of dead ends in a room may also dictate whether or not there are alternative paths between the doors. A room with a high proportion of dead ends (such as Figure 9) may have a single path connecting the doors, while a room with no dead ends (such as Figure 8) may present alternative paths, potentially allowing players to bypass difficult sections while still traversing the room. To provide control over how likely dead ends are to appear in a room at all, we employ a user-defined penalty $P_{de} \geq 0$ to be applied to detected dead ends.

Combining these elements, the quality of a dead end is defined (for a dead end p) as:

$$Q(p) = \frac{1}{2}(1 + Density(p) - P_{de}) \tag{5}$$

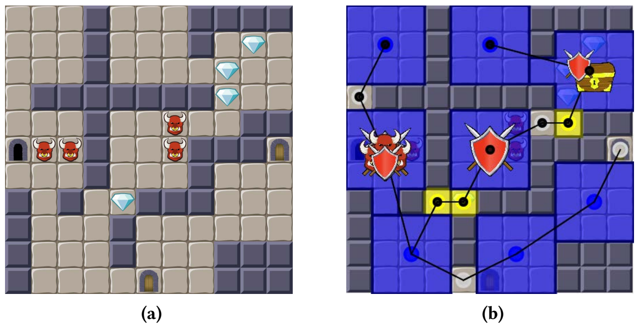
Figure 10: a) A room containing the guarded treasure meso-pattern: a treasure chamber in a dead end such that the player must pass through a guard chamber to access the treasure chamber. b) The pattern overlay view, where the guard chamber in the center and the treasure chamber in the upper right (here marked with the guarded treasure icon) make up a guarded treasure pattern. The black dots in the graph indicate that these patterns are part of a dead-end.

3.2.5 Guarded Treasure. In addition to the previous meso-patterns based on Dahlskog et al.'s definitions, we define one original meso-pattern, *guarded treasure* with the goal of investigating a meso-pattern constructed purely from other meso-patterns, rather than micro-patterns or a combination of the two. Patterns like this are a step closer to Dahlskog et al.'s more abstract macro-patterns, which describe extended sections of gameplay dependent on multiple lower-level patterns [8]. A guarded treasure is a treasure chamber situated in a dead end, such that all paths from the treasure chamber out of the dead end pass through at least one guard chamber (Figure 10). The pattern encapsulates the idea of risk vs. reward: the player does not have to enter the dead end in order to access the room's doors, but can choose to take a risk fighting the enemies in the guard chamber in order to collect the treasure.

We detect a guarded treasure by locating treasure chambers inside previously detected dead ends. From each such treasure chamber, we execute a breadth-first search on the pattern graph, searching for a node that is not part of the dead end. If all such paths pass through a guard chamber, the treasure chamber is considered a guarded treasure.

Quality for guarded treasure is defined by how close the number of guards that must be passed in order to access the treasure is to a user-defined target number, $T_{deadendenemies}$ and is calculated (for a guarded treasure p) as:

$$Q(p) = \max\left\{0, 1 - \left|\frac{N_e - T_{deadendenemies}}{T_{deadendenemies}}\right|\right\} \tag{6}$$

3.3 Evaluation of feasible individuals

The genetic algorithm optimizes feasible individuals (rooms) to increase their fitness, which is a value between zero and one, one being the maximum possible fitness. This fitness is a linear combination of two functions, $f_{inv}(r)$ and $f_{spacial}(r)$, which evaluate desirable features of dungeon rooms. $f_{inv}(r)$ evaluates inventorial patterns, such as the placement of enemy and treasure tiles in the room, relative to each other, the doors and the overall layout of the room. $f_{spacial}(r)$ evaluates the frequency of occurrence (in terms of tile area) and quality of design patterns detected in the room.

Since meso-patterns do not necessarily correspond to a specific set of tiles, their qualities are instead used to modify the quality of the spacial micro-patterns they contain. After this modification, the quality of a spacial pattern is a linear combination of its own quality and the sum of the qualities of any meso patterns it is in (capped at 1). So, for example, a chamber that is not also part of a meso-pattern has its quality capped at 0.9, while if that chamber is part of the treasure-chamber meso-pattern, it may reach a quality of 1.

3.3.1 Inventorial Pattern Fitness. Inventorial pattern fitness $f_{inv}(r)$ for a room r is calculated as a linear combination of the sums of the qualities of each pattern type.

$$f_{inv}(r) = \frac{1}{5}\sum_{p \in D} Q(p) + \frac{2}{5}\sum_{p \in E} Q(p) + \frac{2}{5}\sum_{p \in T} Q(p) \tag{7}$$

Where D, E and T are the sets of all doors, enemies and treasures, respectively.

3.3.2 Spacial Pattern Fitness. Spacial Pattern fitness $f_{spacial}(r)$ is a measure of the frequency and quality of design patterns detected in a room and is a linear combination of two other functions, $f_{chamber}(r)$, which evaluates the fitness of detected chambers and $f_{corridor}(r)$, which evaluates the fitness of detected corridors and connectors. This fitness can be seen as a measure of how closely a room adheres to user-defined target values for the proportion and quality-parameters of each design pattern. The final quality of a spacial micro-pattern p is a linear combination of its own quality and the quality of the meso-patterns it appears in, $MesoQ(p)$, which is defined as:

$$MesoQ(p) = \min\left\{1, \sum_{m:p \in m} Q(m)\right\} \tag{8}$$

$WChambers(r)$ then calculates the proportion of passable tiles in a room r that are in chambers, weighted by the quality of those chambers and the meso-patterns that include the chambers as:

$$WChambers(r) = \sum_{p_{chamber} \in r} \left(\frac{\frac{9}{10} \cdot Q(p_{chamber})}{N_P} + \frac{\frac{1}{10} \cdot MesoQ(p_{chamber})) * Area(p_{chamber})}{N_P} \right) \tag{9}$$

where $p_{chamber}$ is a detected chamber pattern and N_P is the number of passable tiles in the room. This is used to calculate the **chamber fitness** as the degree to which the proportion of chambers differs from a user-defined target proportion $T_{chamber}$ as:

$$f_{chamber}(r) = 1 - \left| \frac{WChambers(r) - T_{chamber}}{\max\{T_{chamber}, 1 - T_{chamber}\}} \right| \tag{10}$$

$WCorridors(r)$ calculates the proportion of passable tiles in a room that are in corridors or connectors, weighted by the quality of those corridors/connectors as:

$$\begin{aligned} WCorridors(r) = \sum_{p_{corridor} \in r} & \left(\frac{\frac{9}{10} \cdot Q(p_{corridor})}{N_P} + \frac{\frac{1}{10} \cdot MesoQ(p_{corridor}) * Area(p_{corridor})}{N_P} \right) \\ + \sum_{p_{connector}) \in r} & \left(\frac{\frac{9}{10} \cdot Q(p_{connector}}{N_P} + \frac{\frac{1}{10} \cdot MesoQ(p_{connector}) * Area(p_{connector})}{N_P} \right) \end{aligned} \tag{11}$$

where $p_{corridor}$ is a detected corridor pattern and $p_{connector}$ is a detected connector pattern. This is used to calculate the **corridor fitness** as the degree to which the proportion of corridors (including connectors) differs from a user-defined target proportion $T_{corridor}$ as:

$$f_{corridor}(r) = 1 - \left| \frac{WCorridors(r) - T_{corridor}}{\max\{T_{corridor}, 1 - T_{corridor}\}} \right| \tag{12}$$

Combining these two fitness functions (using weights determined through a process of experimentation), we arrive at the **pattern fitness**:

$$f_{spacial}(r) = \frac{3}{10} f_{chamber}(r) + \frac{7}{10} f_{corridor}(r) \tag{13}$$

3.3.3 Feasible fitness. Combining the inventorial pattern fitness with the spacial pattern fitness, the overall fitness of a feasible individual, also named as the **feasible fitness score**, is calculated as:

$$f_{feasible}(r) = \frac{1}{2} f_{inv}(r) + \frac{1}{2} f_{spacial}(r) \tag{14}$$

3.4 Evaluation of Infeasible Individuals

The **infeasible fitness score** evaluates individuals that do not satisfy the playability constraint. Rooms that are closer to fulfilling the playability constraint will be awarded higher fitnesses. The fitness $f_{infeasible}(r)$ is calculated as

$$f_{infeasible}(r) = 1 - \frac{1}{3} \left(\frac{N_{epath}}{N_E} + \frac{N_{tpath}}{N_T} + \frac{N_{dpath}}{D} \right) \tag{15}$$

where N_{epath}, N_{tpath}, and N_{dpath} are the number of enemies, treasures, and doors respectively for which there is no path to the main entrance and the best fitness is 1. D is the number of doors excluding the main entrance.

4 EVALUATION

4.1 Diversity and Controllability

We conducted a set of experiments in order test the extent to which the generator is able to optimize rooms to include the presented patterns. These experiments aim for analyzing the diversity of the solutions reached as well as the degree of controllability offered to the user. Each experiment used a total population of 150 individuals, split evenly into feasible and infeasible populations. Each run was terminated after 150 generations. Rooms were 12x12 tiles. For each set of input parameters, 100 runs were executed using 31 representative combinations of the following input parameters (see [2] for more details on the first five): $T_{chamber}$, $T_{corridor}$, $T_{chamberarea}$, $T_{treasure}$, $T_{enemies}$, and P_{de}. Note that $T_{treasure}$ and $T_{enemies}$ represent ranges and are denoted in the form $T_{treasure} = [a, b]$ for brevity. Since these experiments analyze the expressiveness of the generator, the following results are produced only by means of evolution, without any kind of manual edition.

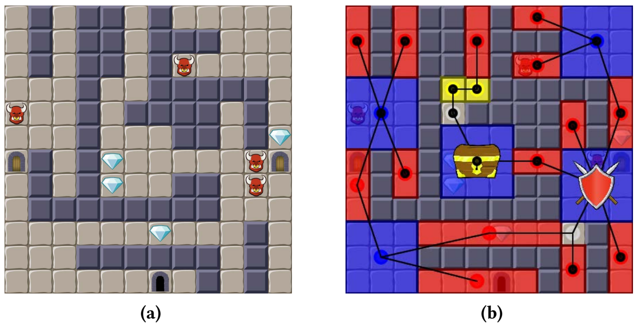

Figure 11: (a) Optimal solution for the configuration: $P_{de} = 0$, $T_{chamber} = 0.5$, $T_{corridor} = 0.5$, $T_{chamberarea} = 9$, $T_{treasure} = [0.04, 0.05]$, $T_{enemies} = [0.04, 0.05]$. **(b) The pattern overlay view for this solution, indicating that a large proportion of the room consists of dead ends.**

Figure 11 shows the results for a series of runs where dead ends were not penalized at all. It was our experience during prototyping that the default behavior of the generator is to create maps with pattern graphs that do not contain cycles (and, as such, are likely to contain dead ends). As illustrated in Figure 12 and Figure 13, by manipulating P_{de} we can control the likeliness for branching paths and *optional areas* to appear.

The results obtained show that, on average, the proportion of dead ends generated is inversely proportional to the value of the

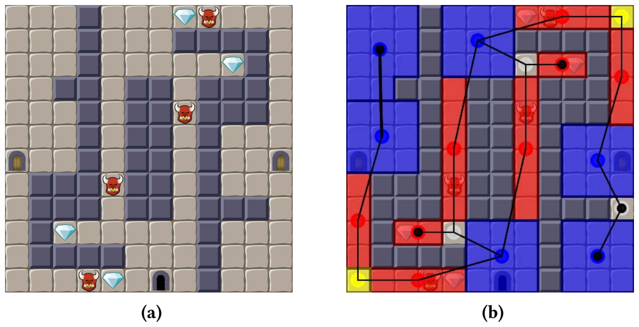

Figure 12: (a) Optimal solution for the configuration: $P_{de} = 3$, $T_{chamber} = 0.5$, $T_{corridor} = 0.5$, $T_{chamberarea} = 9$, $T_{treasure} = [0.04, 0.05]$, $T_{enemies} = [0.04, 0.05]$. **(b) The pattern overlay view for this solution, where, while there are a number of minor dead ends, the majority of the room consists of two branching paths.**

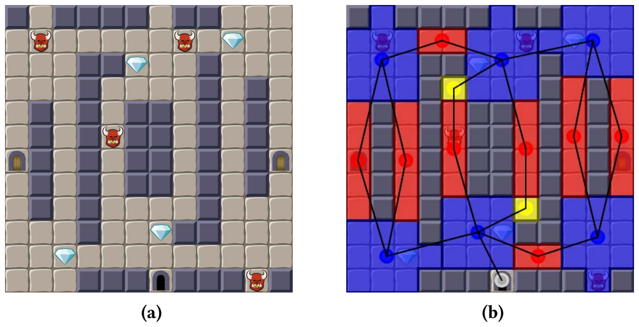

Figure 13: (a) Optimal solution for the configuration: $P_{de} = 10$, $T_{chamber}$ = **0.5**, $T_{corridor}$ = **0.5**, $T_{chamberarea} = 9$, $T_{treasure} = [0.04, 0.05]$, $T_{enemies} = [0.04, 0.05]$. **(b) The pattern overlay view for this solution, indicating that there are no dead ends and there are many alternative routes by which to traverse the room.**

dead end penalty. This varies from a maximum 38% of passable tiles belonging to dead ends when $P_{de} = 0$, to less than 0.05% of tiles when $P_{de} = 10$.

The remaining meso-patterns are evaluated in terms of their occurrence, not being meaningful to consider the proportion of the room that is made up of these patterns, since it will be largely dependent on details of the micro-patterns in the room.

Figure 14 shows the meso-pattern counts for a configuration generating mainly chambers. Including the meso-patterns guard chamber and treasure chamber has encouraged the formation of chambers containing either treasure or enemies rather than a combination of both. In this case, the algorithm increases the number of both guard chambers and treasure chambers, with an average of about 1.42 of each in a room of this configuration. Guarded treasures are also generated by this configuration, with an average of 0.1 per room (since it is very unlikely that two guarded treasures will occur

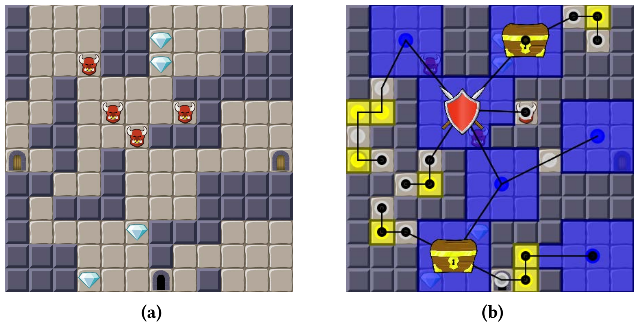

Figure 14: (a) Optimal solution for the configuration: $P_{de} = 3$, $T_{chamber} = 0.5$, $T_{corridor} = 0.5$, $T_{chamberarea} = 9$, $T_{treasure} = [0.04, 0.05]$, $T_{enemies} = [0.04, 0.05]$. **(b) The pattern overlay view for this solution.**

in a single room of this size, this essentially means that 10% of maps generated with this configuration contained one guarded treasure). The average number of ambushes arrives at an average of 0.34 (or one in every three rooms). These four meso-patterns have little impact on the layout of spacial micro-patterns, but rather control the distribution of inventorial micro-patterns, so if the entrance does not happen to be in a chamber, an ambush will not be present.

When the target number of treasures is tripled, from $T_{treasure} = [0.04, 0.05]$ (Figure 14) to $T_{treasure} = [0.12, 0.15]$ (Figure 15), the number of detected treasure rooms increases to an average count of 4.34. This also increases the number of guarded treasures from 0.1 to 0.25. A similar increase in guard chambers occurs when increasing the target number of enemies.

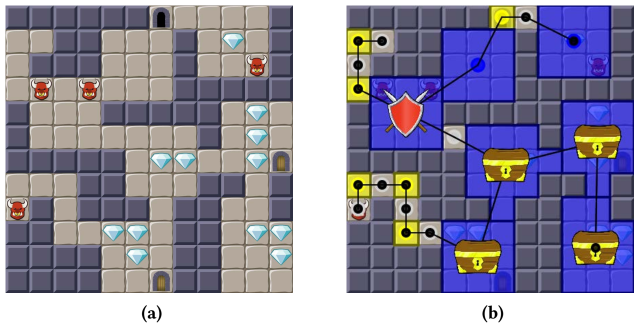

Figure 15: (a) Optimal solution for the configuration: $P_{de} = 3$, $T_{chamber} = 1$, $T_{corridor} = 0$, $T_{chamberarea} = 9$, $T_{treasure} = [0.12, 0.15]$, $T_{enemies} = [0.04, 0.05]$. **(b) The pattern overlay view for this solution.**

Generating rooms with lower proportions of chambers (Figure 16) leads to a lower occurrence of these chamber-centric meso-patterns, settling around an average count of 0.57. Ambushes are also less frequent, as it is less likely that the entrance will be part of a chamber. Guarded treasures were not found in any of the optimal solutions.

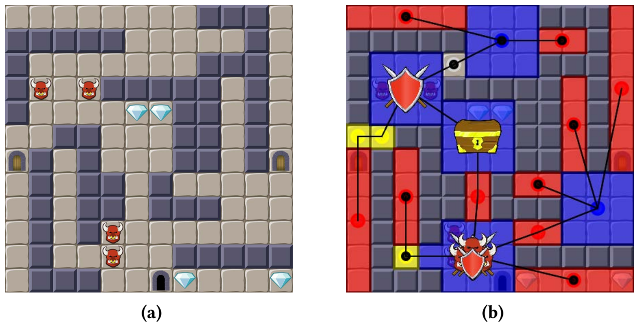

Figure 16: (a) Optimal solution for the configuration: $P_{de} = 3$, $T_{chamber} = 0.5$, $T_{corridor} = 0.5$, $T_{chamberarea} = 9$, $T_{treasure} = [0.04, 0.05]$, $T_{enemies} = [0.04, 0.05]$. **(b) The pattern overlay view for this solution. A lower proportion of chambers in the room has made chamber-based meso-patterns less likely to appear.**

4.2 User Study

A user study was also conducted in order to assess the relevance of the mixed-initiative PCG component. Five representatives from four different branches of the game development industry (level design, engine programming, animation, and user research) participated in the tests. Separately, these participants were introduced to the tool by means of a brief description and a short demo of its functionality. After filling in a pre-questionnaire on their background, each participant was asked to design three rooms that would be part of the same dungeon level showing progression (e.g. in difficulty) between the rooms. Each room was modeled as a square of 11 by 11 tiles with three doors in fixed positions.

The nature of the dungeon and the rooms was left to the participant's imagination. No further information was given on the layout of the individual rooms. Similarly, no information was given regarding the nature of the enemies and treasures in advance. The enemies and treasures were considered generic place holders, giving the participant the opportunity to make of them what they wanted. All the generated maps were saved for an analysis and a discussion conducted in a structured interview with the participant after this phase. An observer kept notes of what the participant was doing, providing additional data for the later analysis.

From the pre-questionnaire it is extracted that the term game design pattern was only familiar to one of the participants, suggesting a need to more deeply investigate the principles and language used in professional level design situations in order to communicate these concepts in a language level designers understand. Besides, only one of the participants had prior experience of using artificial intelligence-assisted level design tools.

The approach taken to accomplish the task varied considerably between the participants. Whereas the designers tended to use the initially selected maps as templates for further development, the other participants tended to select maps based on emptiness in order to start off each room design with a clean slate. The workflow followed by the programmer most closely matched the kind of workflow the EDD tool was originally intended to cater to, described in Section 3.1.

Four out of five considered EDD as an interesting and time-saving tool for designing dungeons, and also expressed the importance of visualizing and working with patterns for this task, specially the dead end. Nevertheless, one participant actively tried to construct rooms containing patterns not previously described, which the tool then failed to recognize. Two of the participants expressed a wish to being able to define new patterns or redefine existing ones, e.g. the dimensions of a corridor or the constitution of a guarded treasure, which most participants interpreted as being a treasure with an adjacent enemy. All in all, four out of five participants considered the pattern detecting capabilities important, with the dead end pattern being deemed extra important.

When asked about what is missing from the tool, four participants mentioned the importance of displaying a graph-like view that connects all the designed rooms into one large dungeon. Three of them considered that the manually edited content should not be altered by the evolved suggestions. Interestingly, two of these considered aesthetics as a very important criterion while designing levels, while the others disregarded this in favor of gameplay. These two also emphasized the importance of preserving a manually designed symmetry in the suggestions offered, arguing that the evolved suggestions appeared to be unrelated to the edited room. The level designers mentioned that they lacked information regarding the game type and game mechanics, e.g. the nature and behavior of the enemies or the meaning of the treasures, which is highly relevant when designing levels.

None of the participants used the pattern overlay frequently. Some of them expressed a perceived lack of usefulness for pattern visualization in the relatively small rooms they were expected to work with. Nonetheless, the same participants expressed that the pattern display would be useful in larger rooms to more easily grasp the layout and nature of those larger rooms, particularly in terms of the possible paths through the room.

5 CONCLUSIONS AND FUTURE WORK

We have presented the latest iteration of the Evolutionary Dungeon Designer, in which we explore how game design patterns can be used in a procedural dungeon generator in order to generate levels based on the type of gameplay they contain, as well as to examine how this can be used in a mixed-initiative level generator as a means of facilitating collaboration between human designers and PCG algorithms.

We have extended the results in [2] to the detection and generation of meso-patterns based on micro-patterns, showing that the level of diversity and controllability achieved by EDD is sufficient to drive an evolutionary mixed-initiative level design tool. While doing this, as an additional contribution, our method refines the pattern definitions presented in [8] to the level of detail necessary to build a pattern-based dungeon generator. The presented graph-based solution provides an understandable way of presenting micro and meso-patterns, as well as a useful framework for working with them in the context of search-based problems.

As a mixed-initiative level generator, EDD provides a workflow in which, through the use of game design patterns, room suggestions are generated to provide a starting point to work with that is of a higher quality than a suggestion simply generated at random or a blank canvas. Users stated that they felt that this approach provides inspiration and could save time, since designers do not need to manually alter as much of the room as they otherwise might.

We notice that users expect rooms suggested based on one of their own designs to preserve the design's essence to a higher degree, both in terms of gameplay/game design patterns and the aesthetics in the room. To facilitate this, the generator could allow designers to define their own patterns, so that they can be later detected, preserved, and promoted. As suggested by Liapis et al. [23], designer modeling could also be used in a mixed-initiative tool in order to better tailor generated content to the user by, for example, emphasizing aesthetics when being used by a designer who favors these features. Alternatively, automatically detecting new patterns based on the user's manual design choices could provide more meaningful suggestions from the user's perspective, as well as enhancing the utility of the tool's pattern visualization function.

The user study also revealed some interesting features to be included in the next iteration of the software. Enemies that behave differently and thus provide different gameplay were found to play a major role in how designers choose to design a level – as such, it would be valuable to include the means of generating/designing multiple enemy types with different behaviour. The inclusion of locked doors and keys would also expand the possibilities for interesting gameplay. A world-like view that presents the designed rooms in an interconnected fashion is also desirable for providing the designer with more context about the entire level. As this work exists as an extension to the previous Evolutionary World Designer project [15], which generates higher level dungeon layouts (above the room level), a next step is to integrate this with EDD so that full dungeon levels can be evolved, starting from the larger context and then moving on to the specifics of each room.

ACKNOWLEDGMENTS

The project was supported by The Crafoord Foundation.

REFERENCES

[1] C. Alexander, S. Ishikawa, and M. Silverstein. 1977. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, New York.

[2] A. Baldwin, S. Dahlskog, J. M. Font, and J. Holmberg. 2017. Mixed-Initiative Procedural Generation of Dungeons using Game Design Patterns. In *Proceedings of the 2017 IEEE Conference on Computational Intelligence and Games.*

[3] S. Björk. 2013. Game Design Patterns 2.0. Web page. (March 2013). http://gdp2.tii.se/

[4] S. Björk and J. Holopainen. 2005. *Patterns in Game Design.* Charles River Media.

[5] Blizzard North. 1996. Diablo. (1996).

[6] D. Braben and I. Bell. 1984. Elite. (1984).

[7] C. Browne and F. Maire. 2010. Evolutionary Game Design. *IEEE Transactions on Computational Intelligence and AI in Games* 2, 1 (March 2010), 1–16.

[8] S. Dahlskog, S. Björk, and J. Togelius. 2015. Patterns, Dungeons and Generators. In *Proceedings of the 10th International Conference on the Foundations of Digital Games.*

[9] S. Dahlskog and J. Togelius. 2012. Patterns and Procedural Content Generation: Revisiting Mario in World 1 Level 1. In *Proceedings of the First Workshop on Design Patterns in Games.* ACM, New York, NY, USA.

[10] S. Dahlskog and J. Togelius. 2013. Patterns as Objectives for Level Generation. In *Proceedings of the Second Workshop on Design Patterns in Games.*

[11] S. Dahlskog and J. Togelius. 2014. A multi-level level generator. In *2014 IEEE Conference on Computational Intelligence and Games.*

[12] S Dahlskog and J. Togelius. 2014. Procedural Content Generation Using Patterns as Objectives. In *Applications of Evolutionary Computation: 17th European Conference, EvoApplications 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers*, A. I. Esparcia-Alcázar and A. M. Mora (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg.

[13] J. Dormans. 2010. Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games.* ACM, New York, NY, USA.

[14] Joris Dormans. 2017. Unexplored. (2017).

[15] J. M. Font, R. Izquierdo, D. Manrique, and J. Togelius. 2016. Constrained Level Generation Through Grammar-Based Evolutionary Algorithms. In *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I*, G. Squillero and P. Burelli (Eds.). Springer International Publishing, Cham, 558–573.

[16] E. J. Hastings, R. K. Guha, and K. O. Stanley. 2009. Evolving content in the Galactic Arms Race video game. In *2009 IEEE Symposium on Computational Intelligence and Games.*

[17] K. Hullett and J. Whitehead. 2010. Design Patterns in FPS Levels. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games.* ACM, New York, NY, USA.

[18] L. Johnson, G. N. Yannakakis, and J. Togelius. 2010. Cellular Automata for Real-time Generation of Infinite Cave Levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games.* ACM, New York, NY, USA.

[19] D Karavolos, A. Liapis, and G. N. Yannakakis. 2016. Evolving Missions to Create Game Spaces. In *Proceedings of the IEEE Conference on Computational Intelligence and Games.*

[20] S. O. Kimbrough, G. J. Koehler, M. Lu, and D. H. Wood. 2008. On a Feasible–Infeasible Two-Population (FI-2Pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research* 190, 2 (2008), 310 – 327.

[21] Brend Kreimeier. 2002. The Case For Game Design Patterns. (2002). http://www.gamasutra.com/view/feature/132649/the_case_for_game_design_patterns.php

[22] A. Liapis, G. N. Yannakakis, and J. Togelius. 2013. Generating Map Sketches for Strategy Games. In *Proceedings of Applications of Evolutionary Computation*, Vol. 7835, LNCS. Springer.

[23] A. Liapis, G. N. Yannakakis, and J. Togelius. 2014. Designer Modeling for Sentient Sketchbook. In *Proceedings of the IEEE Conference on Computational Intelligence and Games.*

[24] E. McMillen and F. Himsl. 2011. The Binding of Isaac. (2011).

[25] Nintendo Research & Development 4. 1986. The Legend of Zelda. (1986).

[26] A. Pantaleev. 2012. In Search of Patterns: Disrupting RPG Classes through Procedural Content Generation. In *Proceedings of the 2012 Workshop on Procedural Content Generation in Games.*

[27] Noor Shaker, Mohammad Shaker, and Julian Togelius. 2013. Ropossum: An Authoring Tool for Designing, Optimizing and Solving Cut the Rope Levels.. In *AIIDE.*

[28] N. Shaker, J. Togelius, and M. J. Nelson. 2016. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research.* Springer.

[29] N. Shaker, G. N. Yannakakis, J. Togelius, M. Nicolau, and M. O'neill. 2012. Evolving Personalized Content for Super Mario Bros Using Grammatical Evolution.. In *AIIDE.*

[30] Gillian Smith, Ryan Anderson, Brian Kopleck, Zach Lindblad, Lauren Scott, Adam Wardell, Jim Whitehead, and Michael Mateas. 2011. Situating Quests: Design Patterns for Quest and Level Design in Role-Playing Games. In *ICIDS.* 326–329.

[31] G. Smith, J. Whitehead, and M. Mateas. 2011. Tanagra: Reactive Planning and Constraint Solving for Mixed-Initiative Level Design. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (Sept. 2011).

[32] J. Togelius, R. De Nardi, and S. M. Lucas. 2007. Towards automatic personalised content creation for racing games. In *2007 IEEE Symposium on Computational Intelligence and Games.* 252–259.

[33] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. 2011. Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (Sept. 2011), 172–186.

[34] M. Toy. 1980. Rogue. (1980).

[35] R. van der Linden, R. Lopes, and R. Bidarra. 2013. Designing procedurally generated levels. In *Proceedings of the 2nd AIIDE Workshop on Artificial Intelligence in the Game Design Process.*

[36] G. N. Yannakakis, A. Liapis, and C. Alexopoulos. 2014. Mixed-Initiative Co-Creativity. In *Proceedings of the 9th Conference on the Foundations of Digital Games.*