Mixed-Initiative Procedural Generation of Dungeons using Game Design Patterns

Alexander Baldwin, Steve Dahlskog, Jose M. Font and Johan Holmberg
Faculty of Technology and Society, Malmö University
Email: {alexander.baldwin, steve.dahlskog, jose.font, johan.holmberg}@mah.se

***Abstract*—Procedural Content Generation (PCG) can be a useful tool for aiding creativity in the process of designing game levels. Mixed-initiative level generation tools where a designer and an algorithm collaborate to iteratively generate game levels have been used for this purpose. However, it can be difficult for designers to work with tools that do not respond to the common language of games: game design patterns. We present the *Evolutionary Dungeon Designer*, the first step towards a mixed-initiative dungeon design tool which evolves dungeon rooms using game design patterns, as well as several metrics regarding the placement of treasures and enemies, in the fitness function of a genetic algorithm. Our results show that we are able to control the frequency, shape and type of design patterns, as well as properly place enemies and treasures in the generated rooms, using design pattern-related input parameters.**

***Index Terms*—Procedural Content Generation, Evolutionary Algorithms, Game Design Patterns, Dungeons.**

I. INTRODUCTION

As the expectations of consumers increase, computer games are constantly becoming larger and more time consuming to produce [1]. A solution that has been suggested for this problem is the algorithmic creation of game content: procedural content generation (PCG). PCG has already been applied in many areas, both in the game industry and in academic research for generation of weapons, maps, levels and even rules for entirely new games [1].

An area that has attracted researcher interest is the generation of dungeons, *i.e.* confined game levels usually consisting of a network of rooms and corridors filled with treasure and enemies [2]. While many approaches to dungeon generation have proven successful, each approach brings its own limitations. Constructive approaches [3] rely on an algorithm that is carefully tuned before generation – a task that can be difficult for game designers if there is not a clear mapping between familiar game concepts and the algorithm's control parameters.

Several efforts have been made to make procedural generation of levels more designer-friendly, often focusing on using the concepts and language of game design [4], [5], [6], [7]. These concepts can be described as game design patterns: "commonly reoccurring parts of the design of a game that concern gameplay" [8]. Game design patterns provide a common vocabulary with which to describe game elements that may be an aid to game designers if integrated into PCG level generation. So far, most of the use of gameplay-related concepts in level generation has treated these concepts as building blocks or inputs to generators, but Dahlskog et al. argue that there may be more value in seeing game design patterns as objectives for generators and the authors describe a search-based approach to level generation using game design patterns [9].

Search-based approaches to generating levels such as evolutionary algorithms can also be integrated into mixed-initiative generation tools where a game designer and a PCG algorithm takes turns to create levels based on desired gameplay, as shown by Liapis et al. in their application *Sentient Sketchbook* [10]. Such approaches may enhance overall creativity in the level generation process while giving designers more control. Attaining a better understanding of how this might be applied to the area of mixed-initiative dungeon generation using game design patterns is the motivation for this research and our goals are presented in the following section.

II. RELATED WORK

In this section, we present an overview of the recent literature regarding search-based methods for procedural level generation, mixed-initiative level generation, and the procedural generation of dungeon-like levels. In the last part of the section we bring up the Evolutionary World Designer [11], to which the work presented on this paper acts as a continuation.

A. Search-Based Procedural Content Generation

Since the first successful implementations in games like *Rogue* [12] and *Elite* [13], PCG has become a very popular tool for reducing the cost taken to develop computer games [1]. Beside cost reduction, PCG allows content to be generated online as players play a game [3]. Content can then be tailored to suit the way they like to play, leading to a more personalized experience [14]. PCG has even been used to generate the rules for entirely new games [15]. To a broader extent, PCG approaches involving agent *intentionality* during their creation have been recently framed within the field of Computational Creativity [16].

Togelius et al. introduce the term *search-based procedural content generation* to describe a set of PCG techniques using search-based methods, such as evolutionary algorithms (EA). EA allow for content to be generated and refined towards a predefined set of goals, using a construct known as a *fitness function* or *evaluation function* [3]. Typically, a set of content artifacts are generated, altered by means of non-deterministic mutation and interbreeding, and then evaluated according to the fitness function [17].

B. Mixed-Initiative Level Generation

A computer, not being held back by limited human imagination, has the potential to produce novel material at a semi-constant rate. However, while PCG may be sufficient in some cases, there exists a desire for human input in the process [3], [18], [19]. This is the motivation behind mixed-initiative PCG, where input from designers and generators is combined.

An example of a mixed-initiative level generator is Sentient Sketchbook [6], which uses EA to generate maps for strategy games. The fitness functions used are inspired by game design patterns such as *resource fairness* and *base safety* [6]. By combining EA with an interactive level authoring tool, designers can actively and iteratively influence the generative process. In order to increase the diversity of the population of potential solutions to their EA, Sentient Sketchbook employs Kimbrough et al.'s *Feasible-Infeasible Two-Population Genetic Algorithm* (FI-2Pop GA) paradigm[20]. Individuals in a population that do not constitute playable maps are placed into a separate infeasible population. Some of the infeasible individuals are allowed to procreate based on the notion that some of them may eventually produce a playable artifact that can re-enter the feasible population.

C. Game Design Patterns

Design patterns are usually described as solutions to commonly occurring problems [21] and have been applied in architecture and widely in software development. Game design patterns are defined as semiformal interdependent descriptions of commonly reoccurring parts of the design of a game that concern gameplay [8]. Björk and Holopainen present a large collection of general game design patterns and a number of additional efforts have been made to collect patterns for specific game genres, including platform games [9], first person shooters [22], and role playing games [2].

Recent research has started to combine game design patterns with PCG. Pantaleev used game design patterns as inputs to a search-based algorithm with the goal of discovering new patterns for RPG skill systems [18]. Smith et al. used design patterns in a mixed-initiative tool for the design of 2D platform game levels, where the patterns were limited to those describing the geometry of the level's terrain [23]. While the previous examples use design patterns as inputs to an algorithm or components to be combined, Dahlskog and Togelius [24] argue that, rather than using design patterns as building blocks for levels, they can be used as objectives for a PCG generator. The authors present a method of generating 2D platform game levels using an evolutionary algorithm, where fitness is based on the number of patterns found in generated levels.

D. Generating Dungeons

The dungeon is a popular level design archetype found in games spanning multiple genres including action-adventure and role playing games such as the *Legend of Zelda* [25] series and *Diablo* [26]. Linden et al. describe dungeons as "labyrinthic environments, consisting mostly of interrelated challenges, rewards, and puzzles, tightly paced in time and space to offer highly structured gameplay progressions" [27]. In practice this is usually implemented as a set of rooms, connected by corridors or hallways [1].

Dungeons have been a popular focus for PCG research and numerous different approaches have been presented for generating dungeon levels, including both constructive and search-based approaches: space partitioning, agent-based methods [1], cellular automata [28], grammars [4], [7], and mutation-based graph evolution [29].

These methods often require that designers understand the generation process in order to either map control parameters to gameplay goals or to decide which fitness function to use in a search-based approach. However, in a search-based dungeon generation tool, gameplay-informed fitness functions can offer PCG solutions that are easier for game designers to use [27]. Using this approach, game designers hold more control over the PCG process, so that the generated spaces in games result from the desired gameplay [7].

E. The Evolutionary World Designer

A formalized way of describing and evolving maps for adventure-like games using context-free grammars is proposed in the *Evolutionary World Designer* [11]. This work presents a search-based approach that avoids generating infeasible individuals. This is achieved by using grammars to define search spaces composed only by feasible individuals. The system is a mixed-initiative approach that guarantees solvability of the levels, and allows the designer to control several features for the generated maps.

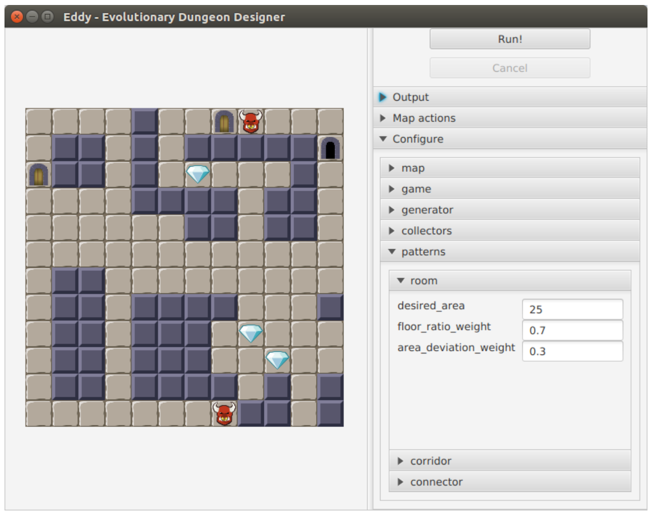
A two-step approach is used to generate an adventure game world in two levels of abstraction: a high-level acyclic graph of the world where each node represents a section of it, and then a set of cyclic graphs, each of them describing a particular dungeon for each section.

The high-level graphs are evaluated according to level branching, recoil, and reward/challenge progression given the designer's desired amount of game objects (enemies and treasure). Low-level dungeons are evaluated based on how interesting the rooms are (in terms of the objects they contain) inside and outside of the critical path (the shortest path from the entrance to the exit), as well as the distances (in number of rooms) between pairs of game objects of the same kind. This allows evaluation of large maps and individual dungeons separately.

III. THE EVOLUTIONARY DUNGEON DESIGNER

The Evolutionary World Designer (EWD) [11] is an ongoing project. In this paper we present the next step in this research, the **Evolutionary Dungeon Designer** (EDD), in which a third level of abstraction is added to generate the specific content for each of the rooms in the generated dungeons.

This is achieved by a FI-2Pop GA. This algorithm is intended to operate over the map generated by the *EWD*, producing optimal suggestions for distributing all the elements



desired_area 25
floor_ratio_weight 0.7
area_deviation_weight 0.3

Fig. 1: Screenshot of the *EDD*, displaying a rendered individual (left) and the configurable options for the EA (right).

in every room. This includes the placement of treasures, enemies, doors, and walls.

The system works on one room at each time. These rooms are evolved from a randomized starting population based on a user-defined configuration of input parameters relating to the type and quantity of design patterns desired in the generated room, as well as the target level of difficulty, and the quantity of treasure and enemies. Figure 1 shows a screenshot of the *EDD*, displaying a sample room on the left and a subset of the configuration parameters for the GA on the right.

In the following subsections we describe the specifics of the codification system, the GA, and the fitness functions that have been specifically developed for this purpose.

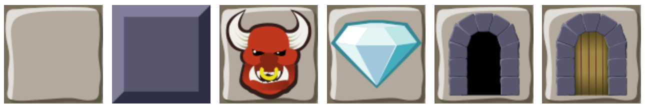
A. Encoding Dungeon Rooms as Individuals

Rooms in the *EDD* follow the structure of those found in *The Legend of Zelda* [25]. Therefore, a room is defined by a rectangular $m \times n$ grid of tiles chosen from six types: floor, wall, enemy, treasure, entrance and door, shown in Figure 2. Walls are impassable, whereas all other tile types are considered passable.

A given room has between one and four doors, of which one is the entrance: the door a character will enter the room through when reaching the room for the first time. For a room to be playable, there must exist at least one treasure and one enemy, as well as paths between the entrance and all other doors, treasure and enemies. Every room is encoded as a two-dimensional $m \times n$ array of integers, in which each integer (from 0 to 5) corresponds to a tile type.

B. Two-population Genetic Algorithm

The genetic algorithm evolves two populations of individuals in parallel: one for the feasible individuals and another one for the infeasible individuals. Each of these populations evolves in a separate genetic algorithm, with its own evaluation, selection, crossover, mutation, and replacement operations. This interbreeding increases the occurrence of feasible individuals and boosts the feasible population's diversity [6]. Since maps that do not satisfy the playability constraint may still contain useful data (such as design patterns), that may later re-enter the feasible population.



(a) Floor (b) Wall (c) Enemy (d) Treasure (e) Entrance (f) Door

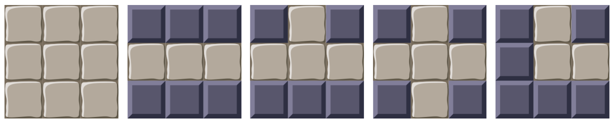
Fig. 2: The graphical representation of the six different tile types used in the generated rooms.

Our implementation of the FI-2Pop GA is largely based on the method Kimbrough et al. describe [20], though we instead use two-point crossover and a very high rate of mutation: 90%. Each mutation has a 20% chance to rotate the room 180º, inspired by Liapis et al. [10], or an 80% chance to mutate a single tile into a random different tile type. The rotation mutation serves to increase the chance of rotationally symmetrical features appearing in maps. Additionally, we end the algorithm after a fixed number of generations, rather than on the basis of a fitness threshold, to ensure that the application's response time is consistent across different sets of input parameters.

C. Pattern Detection

The evaluation of individuals is partly based on the detection of the occurrence of the following fundamental design patterns [2]: micro-patterns *corridor*, *connector* and *room*, though we refer to room as *chamber* to avoid confusion with our previous use of room. The pattern *space* is equivalent to what we refer to as passable tile.

Figure 3 shows several sample expressions of those patterns that may occur in the generated individuals. These are chambers (a), corridors (b), and two types of connectors: joints (c and d), and turns (e). Each detected pattern is assigned an associated *quality* value between 0 and 1 as a measure of how well it conforms to some desired control parameters. Detailed descriptions for these patterns and quality metrics follow.



(a) Chamber (b) Corridor (c) Joint (d) Joint (e) Turn

Fig. 3: Examples of each micro-pattern detected by the generator: (a) the minimal chamber, (b) a three tile long corridor, (c) a three-way joint (the central tile), (d) a four-way joint (the central tile), (e) and a turn (the central tile).

1) Corridors:

A corridor is a horizontal or vertical series of at least two passable tiles, whose end points can be connected to other spaces. Corridors are enclosed by impassable tiles (walls) on either side (Figure 5b).

The quality $Q_{corridor}(c)$ of a given corridor c is calculated as:

$$Q_{corridor}(c) = \min\left\{1.0, \frac{Area(c)}{T_{corridorlength}},\right\} \quad (1)$$

where $Area(c)$ returns the number of tiles included in a pattern (for a corridor this is equivalent to the length), and $T_{corridorlength}$ is the target corridor length specified by the user. Corridors are rewarded for approaching the user defined target length.

2) Connectors: A connector is a passable tile that creates *turns* or *joints* by being connected to two or more corridors. A *joint* is a passable tile with exactly three or four passable neighbors to the north, east, south, or west (Figures 5c & 5d). A *turn* is a passable tile with exactly two passable neighbors in orthogonal directions (Figure 5e).

The user defines the joint/turn balance found in a room by manually setting the quality of each connector type: Q_{joint} and Q_{turn}. This lets the user choose whether evolution will favor the occurrence of joints, turns or both.

3) Chambers: A room is a set of several passable tiles that is wider than a corridor and allows for more freedom in movement. As mentioned above, we refer to this concept as *chamber* rather than room. For pattern detection, we have followed the more rigid definition given by Ashlock et al. [30], that defines a chamber as a core set of 3×3 passable tiles (Figure 5a), which is iteratively expanded. We detect chambers by locating a core block and then expanding that block by using a four-way flood fill algorithm.

The quality $Q_{chamber}(k)$ of a chamber k is determined by a linear combination of two functions, $Squareness(k)$ and $Size(k)$, in the form

$$Q_{chamber}(k) = w_{sq} \cdot Squareness(k) + w_{size} \cdot Size(k), \quad (2)$$

where $w_{sq} + w_{size} = 1$, and

$$Squareness(k) = \frac{Area(k)}{BoundingBoxArea(k)}, \quad (3)$$

measures how square a chamber is as the proportion of the tiles in the chamber's bounding box ($BoundingBoxArea(k)$) that are also in the chamber ($Area(k)$), and

$$Size(k) = \max\left\{0, 1 - \left|1 - \frac{Area(k)}{T_{chamberarea}}\right|\right\} \quad (4)$$

measures how far a chamber's area is from the user-defined target chamber area, $T_{chamberarea}$. By tuning w_{sq} and w_{size}, the user may choose the importance of these features while calculating the quality of a chamber. Figure 6 illustrates how varying these parameters can affect the appearance of chambers in generated rooms.

D. Evaluating the Fitness of Feasible Individuals

We define the fitness of a **feasible individual** (room) r as a linear combination of the enemy and treasure fitness, $f_{ET}(r)$, and the pattern fitness $f_{PAT}(r)$. This allows us to evaluate the individual both on the use of space (patterns) and the positioning of the things found in that space (enemies and treasure).

1) Enemy and treasure fitness: $f_{ET}(r)$ is a linear combination of several features that evaluate the placement of enemies and treasure in a dungeon room. Each feature is optimized to decrease its distance from a target value, based on the user-defined *difficulty* of the room, which can be *easy*, *medium* or *hard*. These features are:

a) Entrance safety fitness: a representation of the size of the largest circular area centered on the main entrance that contains no enemies. Easier rooms provide larger safe areas, while harder rooms usually place enemies next to the main entrance, thus providing tiny safe areas. The safe area of a room $Safe(r)$ returns the number of tiles traversed by a flood fill algorithm starting at the main entrance before an enemy is found. Then we calculate the entrance safety fitness as

$$f_{esafe}(r) = \left|\frac{Safe(r)}{N_P} - T_{esafe}(difficulty)\right| \quad (5)$$

where N_P is the number of passable tiles in the room and $T_{esafe}(difficulty)$ is the user-defined target safe area for a given difficulty level.

b) Entrance greed fitness: analogously to the entrance safety fitness, this measures how close to the main entrance treasures are placed. $Greed(r)$ returns the number of tiles traversed by a flood fill algorithm starting at the main entrance before a treasure is found. We calculate the entrance greed fitness as

$$f_{esafe}(r) = \left|\frac{Greed(r)}{N_P} - T_{egreed}(difficulty)\right| \quad (6)$$

where $T_{egreed}(difficulty)$ is the user-defined target entrance greed for a given difficulty level.

c) Enemy density fitness: the rate of enemies according to the number of passable tiles in the room.

$$f_{enemy}(r) = \left|\frac{N_E}{N_P} - T_{enemy}(difficulty)\right| \quad (7)$$

where N_E is the number enemies in the room and $T_{enemy}(difficulty)$ is the user-defined target enemy density for a given difficulty level.

d) Treasure density fitness: analogously to the enemy density, it calculates the proportion of treasures in the room according to the size of its passable area.

$$f_{treasure}(r) = \left|\frac{N_T}{N_P} - T_{treasure}(difficulty)\right| \quad (8)$$

where N_T is the number of treasures in the room and $T_{treasure}(difficulty)$ is the user-defined target treasure density for a given difficulty level.

e) Treasure safety fitness: We define $Safe_{t,d}(r)$ as the measure of how safe a treasure tile t is, regarding the distance (calculated using the A* search algorithm) to the main door $d_{t,d}$ and to every enemy tile $d_{t,e}$ in r. A treasure can be considered safe for the player to get when it is close to the main entrance and far from every enemy in the room, and vice versa. This is calculated as

$$Safe_{t,d}(r) = \min_{1 \le e \le N_E} \left\{ \max\left\{ 0, \frac{d_{t,e} - d_{t,d}}{d_{t,e} + d_{t,d}} \right\} \right\} \quad (9)$$

This metric has been adapted from the *safety metric* in [6], translating concepts from strategy maps (*resources* and *player bases*) to dungeon rooms (*main door* and *enemies*). Here, $Safe_{t,d}(r)$ provides a value closer to 0 the closer treasure t is to the enemies and the further it is from the main entrance. When the treasure is close to the main entrance and far from enemies, the value approaches 1.

After calculating $Safe_{t,d}(r)$ for every treasure tile in the room, we calculate the arithmetic mean $\overline{Safe_{t,d}(r)}$ and the variance σ^2. The former provides the average challenge for the players to get the treasures, which is indicative of the average level of difficulty for the room. The later indicates whether there is a variety in the challenge level shown by treasures in the room. Higher variances show that the treasure safety values for a given room are dispersed around the mean, indicating higher levels of variety.

Finally we obtain the **average treasure safety fitness** and the **treasure safety variance fitness** as

$$f_{atsafe}(r) = \left| \overline{Safe_{t,d}(r)} - T_{atsafe}(difficulty) \right| \quad (10)$$

$$f_{vtsafe}(r) = \left| \sigma^2 - T_{vtsafe}(difficulty) \right| \quad (11)$$

where $T_{atsafe}(difficulty)$ and $T_{vtsafe}(difficulty)$ are the user-defined target values for the average and the variance, respectively.

All these metrics are linearly combined (using weights determined through a process of experimentation) to get the **enemy and treasure fitness** as

$$\begin{aligned} f_{ET}(r) = 1 - \bigg(& \frac{1}{10} \cdot f_{esafe}(r) + \frac{1}{10} \cdot f_{egreed}(r) \\ & + \frac{3}{10} \cdot f_{enemy}(r) + \frac{1}{10} \cdot f_{treasure}(r) \\ & + \frac{2}{10} \cdot f_{atsafe}(r) + \frac{2}{10} \cdot f_{vtsafe}(r) \bigg) \end{aligned} \quad (12)$$

2) Pattern Fitness: Pattern fitness $f_{PAT}(r)$ 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.

a) Chamber fitness: We define the weighted chamber ratio as the proportion of passable tiles in a room that belong to a chamber, weighted by the quality of that chamber.

$$ChamberRatio(r) = \sum_{k \in r} \frac{Q_{chamber}(k) * Area(k)}{N_P} \quad (13)$$

for every k chamber detected in the room r. Given this, the **chamber fitness** is the degree to which the chamber rate differs from a user-defined target rate $T_{chamber}$.

$$f_{chamber}(r) = 1 - \left| \frac{ChamberRatio(r) - T_{chamber}}{\max\{T_{chamber}, 1 - T_{chamber}\}} \right| \quad (14)$$

b) Corridor fitness: We define the weighted corridor ratio as the proportion of passable tiles in a room that belong to a corridor or a connector, weighted by the quality of those corridors and connectors, respectively.

$$\begin{aligned} CorridorRatio(r) = & \sum_{c \in r} \frac{Q_{corridor}(c) * Area(c)}{N_P} \\ & + \sum_{conn \in r} \frac{Q_{connector}(conn) * Area(conn)}{N_P} \end{aligned} \quad (15)$$

for every c corridor and $conn$ connector detected in the room r. Given this, the **corridor fitness** is the degree to which the proportion of corridors and connectors differs from a user-defined target proportion $T_{corridor}$.

$$f_{corridor}(r) = 1 - \left| \frac{CorridorRatio(r) - T_{corridor}}{\max\{T_{corridor}, 1 - T_{corridor}\}} \right| \quad (16)$$

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

$$f_{PAT}(r) = \frac{1}{4} f_{chamber}(r) + \frac{3}{4} f_{corridor}(r) \quad (17)$$

Finally, the overall **fitness for a feasible individual** is calculated as

$$f_{feasible}(r) = \frac{1}{5} f_{ET}(r) + \frac{4}{5} f_{PAT}(r) \quad (18)$$

E. Evaluating the Fitness of Infeasible Individuals

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

$$f_{infeasible}(r) = 1 - \frac{1}{3} \left(\frac{N_{ne}}{N_E} + \frac{N_{nt}}{N_T} + \frac{N_{nd}}{D} \right) \quad (19)$$

where N_{ne}, N_{nt}, and N_{nd} are the number of enemies, treasures, and doors respectively for which there is no path to the main entrance, and D is the number of doors excluding the main entrance.

IV. RESULTS AND DISCUSSION

We have carried out a series of experiments in order to test the performance of the Evolutionary Dungeon Designer, focusing on the optimization of the population fitness, the pattern detection, and the diversity of the solutions reached.

Each experiment evolves a population of 150 individuals, split evenly into feasible and infeasible populations, during 150 generations, under the same difficulty setting (easy) and room size (12 × 12 tiles, comparable to the 7 × 12 room size in *The Legend of Zelda* [25]). This configuration enables evolutionary runs that take a few seconds to complete (running on 2.69GHz × 2 cores, 16GB RAM), which is suitable for real-time use. A total of 28 different combinations of the following input parameters were tested, each executed 100 times:

- Target chamber ratio: $T_{chamber}$
- Target corridor ratio: $T_{corridor}$
- Target chamber area: $T_{chamberarea}$
- Chamber squareness weight: w_{sq}
- Chamber size weight: w_{size}
- Target corridor length: $T_{corridorlength}$
- Turn quality: Q_{turn}
- Joint Quality: Q_{joint}

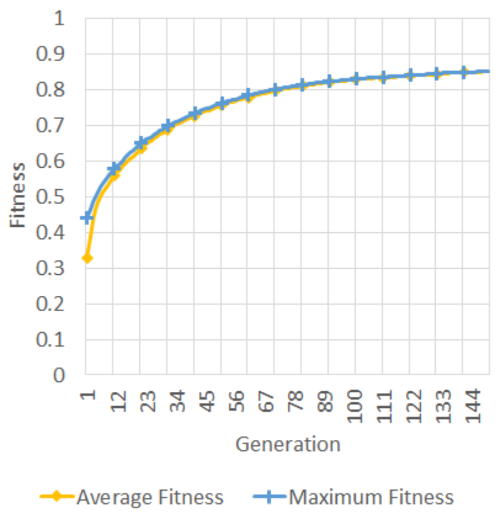


Fig. 4: Average maximum and mean population feasible fitnesses for the configuration $T_{chamber}$ = 0.5, $T_{corridor}$ = 0.5, $T_{chamberarea}$ = 25, w_{sq} = 0.5, w_{size} = 0.5, $T_{corridorlength}$ = 4, Q_{turn} = 0.5, Q_{joint} = 0.5 in 100 runs.

A. Optimization of Feasible Fitness

Figure 4 shows the progression of the average maximum and mean population fitnesses in 100 runs for the parameters $T_{chamber}$ = 0.5, $T_{corridor}$ = 0.5, $T_{chamberarea}$ = 25, w_{sq} = 0.5, w_{size} = 0.5, $T_{corridorlength}$ = 4, Q_{turn} = 0.5, Q_{joint} = 0.5. These results are quite representative of the fitness evolutions shown under all of the 28 tested configurations.

We observe that the population fitness converges around a near-optimal value in less than 150 generations, providing a good optimization speed. In general, the algorithm achieves higher maximum fitness when targeting a high ratio of chamber tiles and lower when targeting corridor tiles, due to a higher weight being assigned to the corridor fitness in the pattern fitness function (Eq. 17).

B. Pattern Generation

Table I presents six groups of tested parameters, primarily focusing on configurations with different corridor and chamber ratios in order to discuss how well the input parameters correspond to the detected patterns. Figure 5 displays six optimal individuals obtained under these parameter settings, that are discussed below.

TABLE I: Configurations used to produce the individuals shown in Figure 5.

	(a)	(b)	(c)	(d)	(e)	(f)
$T_{chamber}$	0	0.2	0.5	0.8	1	1
$T_{corridor}$	1	0.8	0.5	0.2	0	0
$T_{chamberarea}$	25	25	25	25	25	9
w_{sq}	0.5	0.5	0.5	0.5	0.5	0.8
w_{size}	0.5	0.5	0.5	0.5	0.5	0.2
$T_{corridorlength}$	4	4	4	4	4	4
Q_{turn}	0.05	0.5	0.5	0.5	0.5	0.5
Q_{joint}	0.95	0.5	0.5	0.5	0.5	0.5

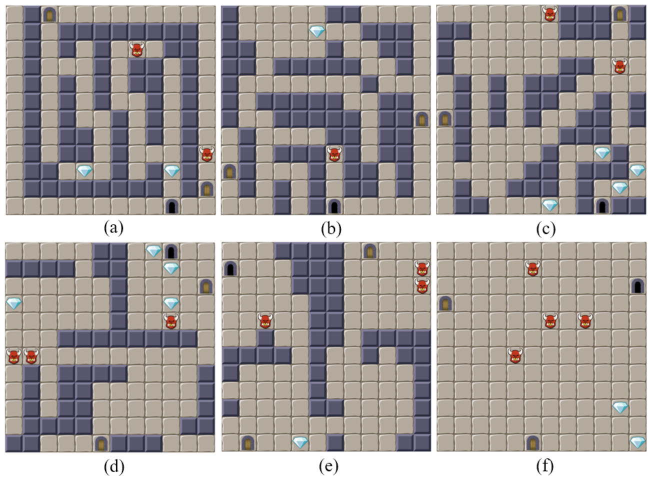


Fig. 5: Optimal individuals obtained in the evolutionary runs under the parameters shown in Table I.

(a) Aiming for no chambers quickly produces a room full of corridors. On average the generator is not able to generate rooms where 100% of the passable tiles are corridors (or connectors), arriving instead at an average of roughly 90%. The corridor fitness remains strictly below 1.0 because connectors are never assigned fitness scores as high as other corridor tiles.
(b) Corridors leave space for smaller chambers in order to fit to the chamber ratio $T_{chamber} = 0.2$. Nevertheless, the obtained chamber ratio usually falls below that target, indicating that corridors have a tendency to dominate in the generated rooms. This can be explained by the fact that corridors must be flanked by wall tiles, which reduces the available floor tiles for chambers.
(c) Equal chamber and corridor ratios ends up close to the target values, though corridors are again slightly overrepresented. The increase in corridor tiles also causes a decrease in the quality of the existing chambers.
(d) Results are similar to those in (c), with corridors being slightly overrepresented at around 25%, while chambers

are slightly underrepresented at around 72%. Corridor fitness is significantly higher than room fitness.

(e) In this case, both chambers and corridors are able to arrive at high fitness values: 0.88 and 1.0 respectively. Almost all of the available passable tiles are chamber tiles, while none are corridor tiles. Without having to compete with corridors, it is clearly easier to optimize the room fitness.

(f) Anomalous cases such as this were produced in 34 out 100 runs with these inputs. The generator can easily satisfy a high squareness property by removing every wall, ignoring the target chamber area and producing an entirely empty room.

Figure 6 shows the impact of setting $T_{chamberarea}$ equal to (a) 9, (b) 25, and (c) 64, while keeping the remaining parameters constant: $T_{chamber}$ = 0.5, $T_{corridor}$ = 0.5, w_{sq} = 0.2, w_{size} = 0.8, $T_{corridorlength}$ = 4, Q_{turn} = 0.5, and Q_{joint} = 0.5. Also shown is the impact of setting w_{sq} equal to (d) 0.2, (e) 0.5 , and (f) 0.8, while keeping the remaining parameters constant: $T_{chamber}$ = 1, $T_{corridor}$ = 0, $T_{chamberarea}$ = 25, $T_{corridorlength}$ = 4, Q_{turn} = 0.5, and Q_{joint} = 0.5. This demonstrates the potential for controlling the size and squareness of the open areas in a room.

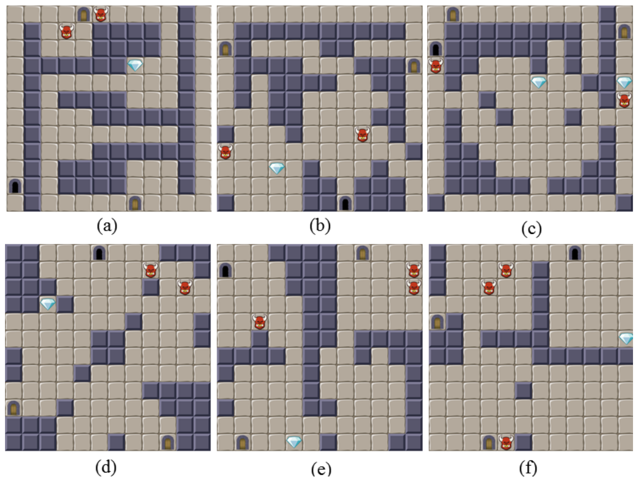


Fig. 6: The impact of varying $T_{chamberarea}$ (a)(b)(c) and w_{sq} (d)(e)(f) values while keeping the remaining settings constant.

C. Diversity of Rooms

While a high fitness and a ratio of pattern tiles closely matching that specified in the input parameters are important, the generator should also produce diverse solutions within these constraints. If the same inputs always result in the same output, the generator becomes useless as a tool for providing novel and diverse room suggestions.

Figure 7 shows nine rooms generated using the same set of inputs, demonstrating that the generator is capable of producing visually and topologically varied outputs for a given set of inputs.

We have also gathered data related to the effects of using FI-2Pop GA, in terms of the number of offspring of infeasible

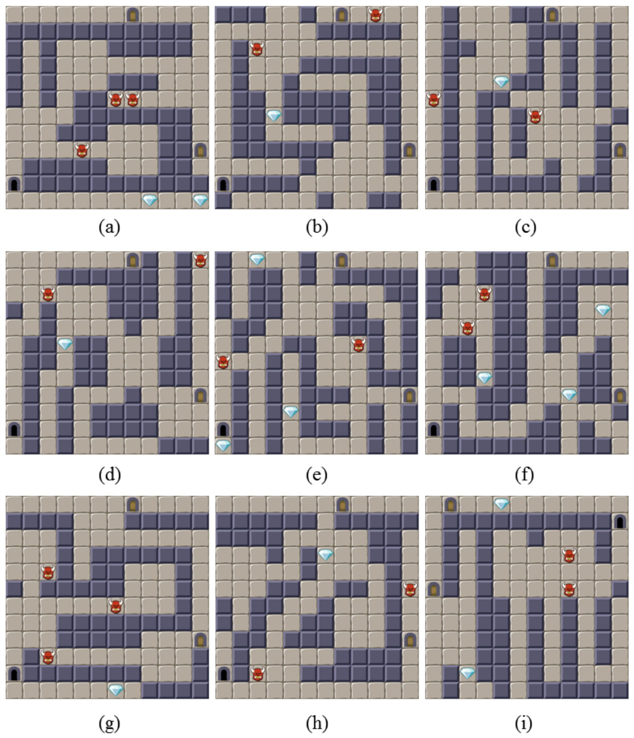


Fig. 7: Nine solutions resulting from the same input parameters: $T_{chamber}$ = 0.5, $T_{corridor}$ = 0.5, $T_{chamberarea}$ = 25, w_{sq} = 0.8, w_{size} = 0.2, $T_{corridorlength}$ = 4, Q_{turn} = 0.5, Q_{joint} = 0.5, showing diverse layouts while containing a similar distribution of design patterns.

individuals that become feasible, as well as the number of those offspring that are fit enough to be kept in the feasible population. Around 5% of the population size of the infeasible offspring is moved to the feasible population in the first two generations, but there is no further movement between the populations after that point. This suggests that, over a large number of generations, FI-2Pop GA may be no more effective than a standard GA for increasing population diversity when using our method.

V. CONCLUSIONS AND FUTURE WORK

In this work we described the implementation of a method for automatically generating rooms for a connected-rooms style dungeon using a FI-2Pop GA, where game design patterns are used both as input parameters and objectives. By considering not only the pattern types, but also assigning a quality measure to each pattern, users (game designers) are able to achieve finer control over the nature of the patterns found in the generated room, making it easier to express their own design style when creating levels.

The experiments evaluated the system in terms of fitness optimization, pattern detection, and solution diversity. Our results show that the values of the input parameters specifying target pattern (chamber and corridor) ratios are correlated to the ratios of these patterns found in the generated rooms. Though they do not currently offer precise control over the occurrence of these patterns, we conclude that a sufficient level of control over the results is provided, since the role of a mixed-initiative generator is to provide designers with reasonable suggestions rather than to adhere to strict constraints. The results also show topological diversity in the optimal solutions produced in different executions of the system under the same input parameters, which could help increase designer creativity.

While we have demonstrated the generation of rooms featuring micro-patterns, much of the gameplay possible in a dungeon relies on how these micro-patterns are arranged into higher-level structures, like Dahlskog et al.'s [2] meso- and macro-patterns. As such, the next step in this work is to detect and optimize for meso- and macro-patterns with the goal of generating rooms with more meaningful use of space, treasure and enemies.

Ideas such as Sentient Sketchbook's [10] multiple map suggestions and the ability to edit the generated maps can be used to allow for more interaction, as well as the possibility of re-evolving already generated rooms to create similar rooms. Such changes should be accompanied by a user study to evaluate the tool's usefulness for game designers as well as an expressive range analysis to visualize the space of possible rooms. At a later stage, we will also integrate the Evolutionary Dungeon Designer in its corresponding stage of the Evolutionary World Designer [11]. Further analysis of the potential of FI-2Pop GA for increasing the solution diversity is also needed. The results of such an experiment may necessitate the investigation of alternative genetic algorithm types, such as those based on Novelty Search [31], where the algorithm aims to maximize the diversity of the population.

ACKNOWLEDGMENTS

Authors want to thank The Crafoord Foundation for supporting this project.

REFERENCES

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

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

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

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

[5] K. Hartsook, A. Zook, S. Das, and M. O. Riedl. Toward Supporting Stories with Procedurally Generated Game Worlds. In *2011 IEEE Conference on Computational Intelligence and Games*, 2011.

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

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

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

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

[10] A. Liapis, G. N. Yannakakis, and J. Togelius. Sentient Sketchbook: Computer-Aided Game Level Authoring. In *Proceedings of the 8th Conference on the Foundations of Digital Games*, 2013.

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

[12] M. Toy. Rogue, 1980.

[13] D. Braben and I. Bell. Elite, 1984.

[14] N. Shaker, G. N. Yannakakis, J. Togelius, M. Nicolau, and M. O'neill. Evolving personalized content for super mario bros using grammatical evolution. In *AIIDE*, 2012.

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

[16] D. Ventura. Beyond computational intelligence to computational creativity in games. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2016.

[17] A. Liapis. Map Sketch Generation as a Service. In *Proceedings of the AIIDE workshop on Experimental AI in Games*, 2015.

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

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

[20] 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. *European Journal of Operational Research*, 190(2):310 – 327, 2008.

[21] S. Dahlskog. *Patterns and Procedural Content Generation in Digital Games: Automatic Level Generation for Digital Games Using Game Design Patterns.* Number 2 in Studies in Computer Science. Malmö university, Faculty of Technology and Society, Malmö, 2016.

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

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

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

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

[26] Blizzard North. Diablo, 1996.

[27] R. van der Linden, R. Lopes, and R. Bidarra. Procedural Generation of Dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(1):78–89, March 2014.

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

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

[30] D. Ashlock, S. Risi, and J. Togelius. Representations for search-based methods. In N. Shaker, J. Togelius, and M. J. Nelson, editors, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, pages 159–179. Springer, 2016.

[31] Joel Lehman and Kenneth O. Stanley. Abandoning Objectives: Evolution Through the Search for Novelty Alone. *Evol. Comput.*, 19(2):189–223, June 2011.