Multi-segment Evolution of Dungeon Game Levels

Antonios Liapis
Institute of Digital Games, University of Malta
antonios.liapis@um.edu.mt

ABSTRACT

This paper presents a generative technique for game levels, focusing on expansive dungeon levels. The proposed two-step evolutionary process creates a high-level overview of the map, which is then used to specify constraints and objectives on multiple constrained optimization algorithms which generate the high-resolution segments of the map. Results show how different types of segments are possible, and how the different connectivity constraints and objectives affect the performance of the algorithm. The modular approach, which allows for a high-level specification of the level first and the subsequent compartmentalized generation of the final map's components, is both scalable and more computationally efficient than a direct encoding, while it allows for more control and user intervention on either level of detail.

CCS CONCEPTS

•Theory of computation → Evolutionary algorithms; •Applied computing → *Computer games;*

KEYWORDS

Level Generation, Rogue-like games, Iterative Refining, Constrained Optimization

ACM Reference format:
Antonios Liapis. 2017. Multi-segment Evolution of Dungeon Game Levels. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071179

1 INTRODUCTION

Evolutionary computation has been extensively used in game content generation in order to evolve puzzles, rulesets, visuals and other types of content [21]. However, the most popular domain for procedural content generation (PCG) within academia and the game industry remains game level generation: this includes the generation of indoors levels — such as dungeons in commercial games from *Rogue* (Toy and Wichman 1980) to *Diablo 3* (Blizzard 2012) and in academia as surveyed in [27] — and the generation of outdoors game worlds [22] such as the galaxy of *ELITE* (Acornsoft 1984) and the earth-like maps of *Civilization VI* (Firaxis 2016). Game levels cover the broadest spectrum of aesthetic and game design choices, as all games regardless of their complexity or target audience take place in one or more levels. To demonstrate the different criteria considered when tackling level generation, generated levels for horror games prioritize a steady increase in tension from the start of the level towards the end, both in the game industry with *Daylight* (Zombie Studios 2014) and in academia with *Sonancia* [14]. While levels for horror games primarily affect the player experience via low visibility, aural or visual cues to scare players, and labyrinthine hallways to confuse their sense of direction, other games such as *Starcraft* (Blizzard 1998) focus on the multi-player competitive aspect: level generation for such games must primarily be concerned with player balance on the functional dimension, such as the distance to resources [25]. In levels for single-player story-driven experiences (e.g. Role-Playing Games), the narrative progression more fundamentally affects the generation of the level so that some locations, NPCs and enemies are encountered first before the story develops further; level generation on such criteria has also been realized both in the game industry with *Moon Hunters* (Kitfox Games 2016) and in academia with *Game Forge* [6].

There has been extensive work in generating game levels, both with carefully scripted constructive methods [19] and with artificial evolution [26]. Due to its very nature, artificial evolution applied to level generation requires a compact representation and well-behaved objective functions. This often limits the granularity of the levels that can be evolved, for instance by forming impassable regions via squares and lines [16, 25] which is ultimately unrealistic to players. The clever design of a representation that can create complex levels can greatly enhance the appearance of the final artifacts but also optimization performance; attempts at evolving grammars [20] or other rewrite rules [1] can craft finer details of levels at the loss of locality for the evolutionary algorithm.

This paper extends the literature on search-based PCG [26] by introducing a two-step generative process for creating complex levels. The paper focuses on generating dungeon levels as a testbed, due to their popularity in the game industry since *Rogue* and *Diablo* (Blizzard 1996), but the algorithm can generate any type of top-down level by changing tilesets and objectives. The novelty of the proposed approach lies in the representation of the game level in two ways: as a high-level *sketch* of the dungeon and as a low-level high-resolution map composed of *segments* generated individually. The dual representation of levels as missions (generated first) and spaces (generated to match the mission) was first introduced by Dormans [4, 5]; however, this is the first instance where both representations are evolved rather than Dormans' grammar-based generation of missions and spaces, or in [7] where evolution was applied only on the mission graph. This sequential process, with the dungeon sketch evolved first to guide the parameters of individual segment generation, allows for faster and more controllable evolution as the representation and evaluation on low-resolution sketches is simpler and easily scalable to more expansive final maps. Moreover, splitting the level into segments similarly lowers the computational

GECCO '17, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3071178.3071179

effort due to the smaller size and complexity of each individual segment, and allows the use of custom genetic mappings (i.e. embryogenies) and objectives in each segment which are specified by the high-level sketch. The paper presents how the dungeon sketch specifies the segments' embryogeny [24] (based on its wall segments and connectivity patterns), constraints and tile counts (based on the high-level segment type stored in the sketch). Using constrained optimization techniques on both the dungeon sketch and the individual segments ensures that resulting maps are (a) guaranteed to be playable and (b) exhibit certain designer priorities specified either as additional constraints or as objectives. While the paper is inspired by earlier work on map sketches and their evaluations [12] and the idea of iteratively refining levels in increasing representational accuracy [10], this is the first instance where the final map is segmented and evolved piece-meal, and the first instance where the high-level sketch directly affects the embryogeny and objectives of different segments of the final map.

2 RELATED WORK

In order to provide the necessary background for the algorithms used in this study, this section provides a brief overview of constrained optimization in PCG along with a summary of work on evolving low-resolution levels (such as this paper's dungeon sketch) and the objective functions for general level creation.

2.1 Constrained Search-based PCG

As noted in the introduction, evolutionary computation has often been used within academia to generate game levels under the general principles of search-based PCG [26]. Since levels need to fulfill some playability criteria (e.g. a puzzle game must be solvable and a dungeon must have a path from entrance to exit), many search-based level generators somehow constrain the results to only show playable (i.e. feasible) content. While the naive solution is to assign the lowest fitness score to unplayable levels, in evolutionary computation this has been argued against [15] as it leads to a loss of important information and can lead to random search if no feasible individuals exist in the initial population. Other solutions include penalties on the fitness scores of infeasible individuals [3], which has been used in the level generator of [17]. Another alternative is to co-evolve two populations, one with feasible individuals and another with infeasible individuals [8]: this feasible-infeasible two-population genetic algorithm (FI-2pop GA) evolves infeasible individuals towards minimizing their distance to the feasible space and transfers feasible offspring of infeasible individuals to the feasible population and vice versa. The FI-2pop GA has been used extensively for level generation [12, 23] — also in this paper — as well as for generating game music [18] and art assets [9].

2.2 Evolution of Map Sketches

This work builds on the general level generation framework introduced in [12], which uses constrained optimization to evolve simple game levels named *map sketches*. A map sketch is a two-dimensional low-resolution representation of a final game level. Map sketches include passable and impassable tiles as well as the minimal set of game-specific tiles which suffice to define the functionality of a game or game genre: for instance, real-time strategy game-specific tiles are player base tiles and resource tiles while for shooter games that would be team spawn points, weapon and healthpack tiles [12]. Due to the abstract representation both in terms of map size and tile variety, these map sketches can be evolved (and evaluated) in straightforward ways.

Genotypically, map sketches are represented directly, as an array of integers where each integer is the type of one tile in the level. Map sketches can be recombined via 2-point crossover (swapping areas of the level of two parents) or mutated by adding or removing game-specific tiles or by swapping the types of adjacent tiles. Due to playability constraints, evolution is carried out via the FI-2pop constrained optimization method: infeasible individuals evolve to minimize an infeasible fitness which evaluates their distance from feasibility and can have multiple criteria (i.e. objectives), while feasible individuals evolve to maximize several general evaluations of level quality discussed below.

The evaluation of map sketches, therefore, is on the distance to playability for infeasible sketches and on the level's quality for feasible ones. Primarily, constraints test whether all game-specific tiles are connected with each other via passable paths and calculate distance to feasibility based on the number of disconnected game-specific tiles. Other constraints such as on the number of tiles of one type can also be introduced: there, the distance to feasibility is the difference in the number of tiles of this type from the designer-defined lower or upper bound. For feasible levels, there is a general framework for evaluating a number of level patterns inspired by the notions of *exploration*, *safety* and *balance* [2]. The exploration metric from tile i to a tile j is the ratio of total passable tiles which are covered when a flood fill algorithm starts from i and stops when j is covered. The safety metric of tile j to tile i gives positive values proportional to the ratio of distance between tiles i and j over the distance between j and the closest other tile in the set that i belongs to (if i is not the closest in its set, then safety is 0). Balance is evaluated based on the equality of scores in exploration or safety between tiles of the same set. The objective functions derived from these concepts are summarized below and visualized in Fig. 2, while details of their mathematical formulations can be found in [12]:

Exploration as $f_{exp}(S_N)$ which evaluates the effort made to discover tiles in the set S_N starting from other tiles in the same set, and its balance dimension as $b_{exp}(S_N)$, i.e. if all tiles in S_N are equally difficult to find from each other.

Safe areas as $f_{area}(S_N)$ which evaluates the number of passable tiles much closer to one tile in the set of S_N than other tiles in the same set, and its balance dimension as $b_{area}(S_N)$, i.e. if tiles in S_N have similar-sized safe areas.

Strategic resource control as $f_{saf}(S_N, S_M)$ which evaluates whether tiles in the set S_N are much closer to tiles in the set of S_M than other tiles in S_N, and its balance dimension as $b_{saf}(S_N, S_M)$, i.e. whether each tile in S_N has equal nearby tiles in S_M.

3 METHODOLOGY

The core of this paper is the framework where levels can be represented in a high-level way, and then each segment thereof can be further refined via evolution and be re-inserted into the whole to provide a high-resolution playable level. This iterative evolutionary process is summarized in Figure 1. Focusing on the dungeon

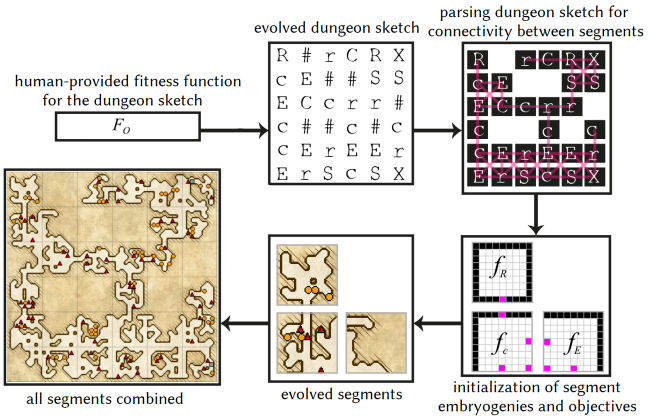


Figure 1: The generative pipeline used in this paper: the dungeon sketch is evolved first towards a specific objective. Each tile on the dungeon sketch, and its connectivity with adjacent tiles, is used to define the embryogeny, constraints and objectives of each segment which is evolved piecemeal. Once all segments are evolved, they are combined together based on the dungeon sketch to create the final dungeon.

generation domain, the sections below describe the tilesets used for each level of detail (the high-level *dungeon sketch* and the low-level *dungeon segment*) and the objectives and constraints for evolving levels in both levels of detail. It should be noted that in this paper, the term *embryogeny* [24] specifically refers to each segment's phenotypic representation as different connectivity constraints require the placement of specific tiles (connections, walls) on the edges of the segment, which can not be modified via evolution.

3.1 The Dungeon Sketch

The dungeon sketch borrows heavily from the concept of a map sketch discussed in Section 2.2: it is a high-level representation of the dungeon which primarily defines the connectivity of different segments (via the introduction of impassable segments) and secondarily the content of the passable segments. Each tile on the dungeon sketch represents a *dungeon segment* in the final map. The types of segments are shown in Table 1; each segment type has different parameters and objectives in the segment evolution process detailed in Section 3.2. Simple and exit segments have a few monsters and treasures, empty segments are passable but have no monsters or treasures, and wall segments are completely impassable[1]. High and sparse challenge segments contain more monsters, while high and sparse reward segments contain more treasure; high challenge and high reward segments contain only monsters and treasures respectively.

In order to evolve the dungeon sketches, they are represented as a 2D array of integers and are optimized via the FI-2pop GA [8] to (a) minimize the distance to feasibility for unplayable dungeon sketches and (b) improve the quality of playable dungeon sketches. Feasibility consists of the following constraints: all segment except empty and walls must be connected to each other via passable paths, there must be exactly two exit segments, two high

[1]Wall segments consist entirely of wall tiles so they do not need to be evolved.

E	Empty	#	Wall
S	Simple	X	Exit
C	High Challenge	c	Sparse Challenge
R	High Reward	r	Sparse Reward

Table 1: Types of segments appearing in the dungeon sketch, and their notation.

challenge segments and two high reward segments (otherwise the dungeon is too difficult or too easy). The fitness function for the playable dungeon sketches is a sum of several fitness dimensions on exploration between exits (ensuring that it is difficult to find an exit when starting from the other one), on area distribution of non-empty, non-simple segments, on safety of high reward tiles to high challenge tiles (ensuring that those tile types will be close together) and on their balance counterparts. This fitness is formalized below:

$$F_O = f_{area}(\{X, C, c, R, r\}) + b_{area}(\{X, C, c, R, r\}) + f_{exp}(X) + b_{exp}(X) + f_{saf}(C, R) + b_{saf}(C, R) \quad (1)$$

Evolution is carried out via fitness-proportionate roulette wheel selection, and selected parents can mutate (1-parent mutation) with a 10% chance or else recombine via 2-point crossover to create two offspring which also have a 10% chance of mutating. Initial individuals have empty or impassable tiles, and mutation can change empty tiles into C, c, R or r with a 2% chance and into walls or simple tiles with a 10% chance[2]. While feasible offspring of infeasible parents move to the feasible population, to ensure that there is a large feasible population for an efficient optimization process the *offspring boost* [13] is applied: if the feasible population is smaller than the infeasible one, the number of offspring from feasible parents is increased to half the total population with a relevant decrease in the infeasible population.

3.2 Dungeon Segments

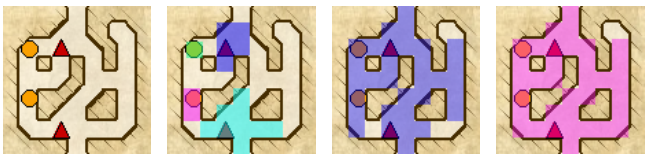
As its name suggests, a dungeon segment is a piece of the dungeon (and a tile of the high-level dungeon sketch). As part of the whole dungeon, the segment needs to obey connectivity requirements with its adjacent segments. Moreover, as the lower-level segments need to specify the distribution of monsters and treasures in the level, the high-level directives of the dungeon sketch are used to ascertain the number of game objects in each segment as well as the objectives it evolves towards. The same naming convention as in Table 1 is used to describe the segments. Based on Table 2, segments have a set of possible tile types; among these tile types, *connection* tiles are used to link different segments together in the full map. High and sparse challenge segments have 3 to 5 monsters, while other segments have 2 monsters except for empty and high reward segments which have no monsters. Similarly, high and sparse reward segments have 3 to 5 treasures, while other segments have 1 or 2 treasures apart from empty and high challenge segments which have no treasures. Only exit segments have an exit tile.

As dungeon segments must be connected to the rest of the dungeon as per the dungeon sketch, the representation of each dungeon

[2]The opposite is also true, so any wall tile has a 10% chance to mutate back to an empty tile, for instance.

.	Empty	#	Wall
n	Connection	x	Exit
m	Monster	t	Treasure

Table 2: Tile types and notations in the dungeon segment.

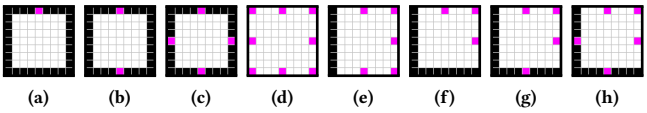


(a) (b) (c) (d)

Figure 2: Fitness functions applied to the dungeon segment of Fig. 2a. The top-most treasure (circle) has a high safety score to its nearby monster (triangle) as that monster is much closer to it than other monsters; the other treasure has a low safety score as it is equally close to two monsters. Safe areas (with a safety score above 0.35) for monsters and treasure tiles are shown in Fig. 2b; the total safe area coverage evaluates f_{saf}. Exploration is calculated via a flood fill algorithm starting from the top connection tile until the bottom connection tile is reached (Fig. 2c) and from bottom to top connection (Fig. 2d); the average explored area from all connections to all connections evaluates f_{exp}. In this level, the safe areas are not equal in size, nor is the exploration from one connection to the other (due to more winding corridors near the bottom connection); this is assessed by the balance dimensions b_{area} and b_{exp} respectively.

segment includes "frozen" tiles which are specified in the initial population and can not be modified during evolution. The frozen tiles are on the edge of the segment and define wall areas and connection tiles. Fig. 3 shows examples of frozen tiles in different segment layouts: 1-neighbor segments (Fig. 3a) have only one neighboring passable segment in the dungeon sketch (upwards), 2-neighbor segments (Fig. 3b) have two (downwards and upwards), 8-neighbor segments (Fig. 3d) represent areas which are fully surrounded by passable segments, etc. While connection or wall tiles can not be changed as they are "frozen", empty tiles along the segments' edges can be mutated into wall tiles or any other tile, or recombined.

Defining constraints and goals for each type of dungeon segment greatly depends on its type, defined by the dungeon sketch. All segments have constraints that all game objects (monsters, treasures, exits) are connected via passable paths; all segments also have constraints on the number of monsters, treasure, or exit tiles they can have. Another constraint for segments with one or more monsters is that those monsters block paths to treasure and exit tiles of this segment, i.e. there is no path between any of those tiles and any connection tile if monster tiles are treated as impassable. If such a path exists, the number of connected game objects is added to the distance function which the infeasible population of the FI-2pop GA attempts to minimize. For the fitness function of playable segments, the main axes are (a) the area distribution of treasures and monsters (see Fig. 2b), as $f_{area}(\{m,t\})$ and its balance $b_{area}(\{m,t\})$;



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 3: Frozen tiles when initializing the representation of segments with different connectivity constraints based on the layout of the dungeon sketch. Black tiles are impassable walls and magenta tiles are connections: only those tiles are frozen and can not be changed, while empty tiles on the segments' edges can be changed and mutated normally.

Dimension	Segment type E	S	X	C	c	R	r
$f_{area}(\{m,t\})$	0	0	0	-1	1	-1	1
$b_{area}(\{m,t\})$	0	1	1	-1	1	1	1
$f_{saf}(t,m)$	0	0	0	0	-1	0	1
$b_{saf}(t,m)$	0	1	0	1	1	0	1
$f_{exp}(\{n,x\})$	1	1	1	1	1	1	1
$b_{exp}(\{n,x\})$	0	1	1	1	1	1	1

Table 3: Fitness weights for the different segment types specified by the dungeon sketch (see Table 1).

(b) the safety of treasures with respect to monsters, as $f_{saf}(t,m)$ and its balance $b_{saf}(t,m)$, (c) the exploration between connection or exit tiles (see Fig. 2c–2d), thus ensuring that there is no straight line from one connection (or exit) to the next, as $f_{exp}(\{n,x\})$ and its balance $b_{exp}(\{n,x\})$. These are combined as a weighted sum differently from one segment type to the next as shown in Table 3. Segments are evolved with the same genetic process and parameters as the dungeon sketch (see Section 3.1); mutation has a 5% chance of changing each empty tile into a monster, treasure or wall (and vice versa).

4 EXPERIMENTS

In order to evaluate a full dungeon generated by the process of Fig. 1, it is necessary to first evaluate how its constituent parts (the dungeon segments) fare in the evolutionary process. Section 4.1 explores how different objectives and different embryogenies for dungeon segments affect optimization and the appearance of the final results. Section 4.2 demonstrates how the segments can be combined based on the directions of an evolved dungeon sketch.

4.1 Results of Dungeon Segment Evolution

Segments differentiate themselves based on their connectivity requirements with adjacent segments and their feasible and infeasible fitness functions which depend on the tile of the dungeon sketch. Fully exploring all possible connectivity patterns is not possible in this paper, so an indicative sample of the possible segment layouts will be tested: a dead-end with one connection (see Fig. 3a), a straight corridor with 2 connections (see Fig. 3b), a corner of an open-space area with 3 connections (see Fig. 3f) and finally a fully open area with 8 connections to all adjacent tiles (see Fig. 3d). Each

Type	Connections 1	2	3	8
E	1.00±0.00	1.00±0.00	0.99±0.00	0.98±0.00
S	2.95±0.02	3.55±0.14	2.96±0.08	2.88±0.14
X	3.56±0.06	3.39±0.15	3.06±0.09	3.34±0.21
R	1.91±0.00	2.84±0.01	2.41±0.02	2.56±0.01
r	3.79±0.04	4.51±0.10	3.42±0.09	3.70±0.26
C	0.38±0.01	1.32±0.01	1.03±0.03	1.08±0.01
c	3.23±0.03	3.79±0.07	3.26±0.04	3.26±0.09

Table 4: Fitness scores of the best segment for different segment types. Results are averaged from feasible runs among 100 runs, along with the 95% confidence interval.

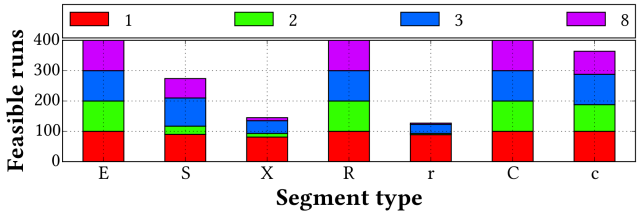


Figure 4: Runs (out of 400 in total) with at least one feasible result. The runs are split by segments with 1 or 2 connections, a corner (3), or entirely open (8).

of the segment types of Table 1 are evolved using these four layouts, for 100 evolutionary runs each. Each evolutionary run uses a total population of 50 individuals (both feasible and infeasible) and runs for 20 generations. The number of generations is purposefully low as the generator should create multiple segments for a single dungeon sketch so speedy evolution is a priority.

4.1.1 Performance Optimization. Since the different segment types have different fitnesses, it makes sense to compare the fitness scores for the same segment type but with different embryogenies. Table 4 shows the fitness scores of all segment types and embryogenies, averaged from those runs that resulted in feasible segments among 100 runs. Comparing only within the same segment type, we observe that in most cases the 2-connection corridor reaches the highest fitness from other embryogenies (significantly so in R, r, C, c, S via the Student's t-test at 5% significance level and the Bonferroni correction), followed by either 8-connection or 3-connection segments. Based also on the discussion of sample results in Section 4.1.3, a reason for this high fitness is the exploration dimension, $f_{exp}(\{n, x\})$; it is easier to create a winding path between two connections, especially when they are on opposite sides of the segment (as in Fig. 3b), than among 8 nearby connections (as in Fig. 3d).

4.1.2 Constraint Satisfaction. One of the most important differences when using different embryogenies and objectives was their ability to handle constraints: Fig. 4 shows the number of runs among 100 per embryogeny examined which result in at least one feasible result. In general, segment types which did not have both monsters and treasures (E, C, R) were quick to find feasible individuals; however, for other segment types the constraint that monsters must block paths to treasure resulted in many runs not having even

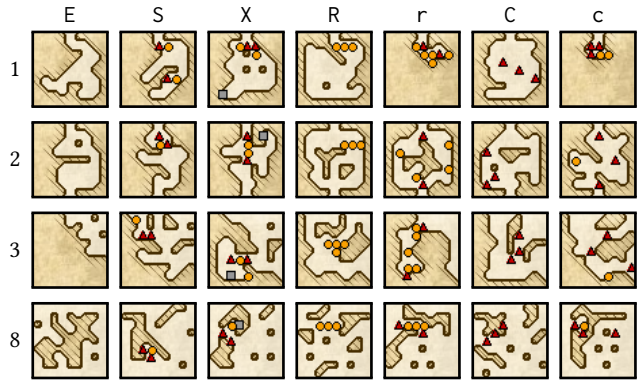


Figure 5: Fittest evolved dungeon segments with 1 or 2 connections, a corner or entirely open (see Fig. 3).

one feasible result within 20 generations. Especially problematic were segments with many treasures and few monsters (r) or with an exit which should also be blocked (X). Moreover, 2-connection corridors were more challenged in satisfying the connectivity and blocking constraints than other embryogenies since the two connections are far away from each other (and could be easily accidentally blocked off via mutation), while the blocking monsters requirement can only be satisfied if the wall tiles form niches with a monster in front; these constraints dictate a rather specific level pattern (see Figure 5) and limit the feasible search space that can be explored.

4.1.3 Level Patterns of Evolved Segments. Figure 5 shows the overall fittest segments in the final population for the different segment types and embryogenies, after post-processing to remove areas not accessible from connection tiles (those were always empty tiles) and smoothen wall sections to create curvier forms. Walls are drawn in a darker color with diagonal hatching, treasures as orange circles, monsters as red triangles and exits as gray squares.

Fig. 5 highlights the differences between segment types: exit tiles are only present in X segment types, game-specific tiles are absent in empty segments, high challenge segments have no treasures and high reward segments have no monsters. Moreover, sparse reward segments (r) have more treasures than other segments (except R) while sparse challenge (c) segments have more monsters (except C). Constraints described in Section 3.2 are satisfied: all connection tiles are connected and all treasures are guarded by one or more monsters which are usually placed on choke points formed by neighboring wall segments: this is especially obvious in the 2-connection r segment where the monsters are placed at both entrances of the segment, adjacent to the connection tiles. Exit tiles are always guarded as well: in the 1-connection X segment, for example, heroes coming through this exit must fight the monsters at the segment's edge in order to explore the rest of the dungeon.

In terms of the patterns specified by the feasible fitness dimensions, results are less coherent. All segments of Fig. 5 have labyrinthine corridors as most of them are optimal on exploration, $f_{exp}(\{n, x\})$. The pattern that monsters and treasures are dispersed, $f_{area}(\{m, t\})$, is prominent in some cases of r and c segments,

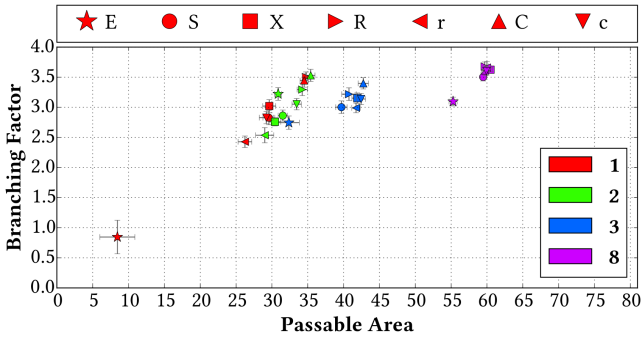
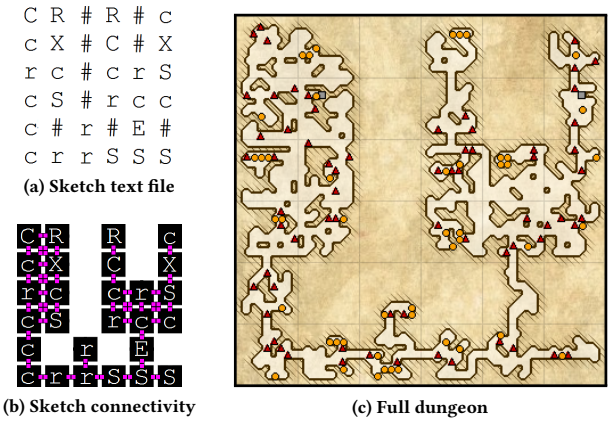


Figure 6: Scatter plot of the accessible passable areas in the segments, and their tiles' average branching factor, grouped by segment type or connections. Data is collected from 100 runs, and error bars show the 95% confidence intervals.



(a) Sketch text file

(b) Sketch connectivity

(c) Full dungeon

Figure 7: Sample evolved sketch and the high-resolution map evolved based on it.

which explicitly reward it; when not targeted as an objective, monsters are often placed side by side (and next to treasures) to block adjacent choke points, e.g. in most X segments. When the $f_{area}(\{m,t\})$ objective receives a negative weight, results consistently place treasures or monsters adjacent to each other (especially in R but also in C segments). Finally, the resource control requirement of $f_{saf}(t,m)$ can be gleaned in r segments where treasures are adjacent to monsters, but it is highly inconsistent since monsters are near each other (resulting in lower safety). Due to the highly constrained search space (especially for 2-connection segments) and the multiple combined objectives which are likely easier to optimize than $f_{saf}(t,m)$, it seems that the latter remains sub-optimal in most cases.

This qualitative analysis can be substantiated via several level metrics: this paper focuses on the number of accessible tiles to any connection (measuring how "empty" segments are) and the average branching factor of those accessible tiles (measuring how "winding" pathways are). Figure 6 shows a scatter plot of those two metrics on each segment type and embryogeny examined: values are averaged from 100 evolutionary runs and the error bars show the 95% confidence interval on either axis. Unsurprisingly, the open-space segments with 8 connections have more accessible tiles, and the number of accessible tiles drops proportionately with 3, 2 and 1 connections. Since open-space segments have 8 connections and no "frozen" impassable edge tiles, they require more empty space to ensure that everything is connected. On the other hand, empty segments always have fewer accessible tiles as they do not need to connect any tiles except connections. Another interesting finding is that for less than 8 connections, r tiles have a low branching factor as they need to create more crannies for the many rewards to be stashed in, and choke points for monsters to guard.

4.2 Sample Full Dungeon

In order to assess how the multiple components (sketch and segment evolution) work together, Figure 7 shows a sample dungeon evolved via this two-step process. The dungeon is generated based on a high-level dungeon sketch of 6 by 6 tiles (see Fig. 7a), which evolved for 20 generations using a population of 50 individuals (including feasible and infeasible individuals). Fig. 7a shows the best dungeon sketch from 100 runs, which is refined into the full dungeon of Fig. 7c by evolving custom segments using the same parameters as in Section 4.1: evolution of each segment is re-run until a feasible one is found. Finally, post-processing removes empty tiles that are not connected to any exit tiles in the final dungeon, and the walls are smoothened. The symbols of the final dungeon are the same as those described in Section 4.1.3.

The patterns of the dungeon sketch in Fig. 7a reflect the objectives of Eq. (1). The exploration fitness (f_{exp}) is nearly optimal, as the two exits are on opposites sides of the dungeon, connected via a winding corridor acting as a chokepoint. The resource control fitness (f_{saf}) is also high, as each high challenge segment is adjacent to a high reward segment (and these pairs of segments are far away from each other as they are on opposite sides of the dungeon). On the other hand, the safe areas around challenge (C, c), reward (R, r) and exit segments are few, since many of these segments are adjacent to each other: therefore, the safe area fitness (f_{area}) is fairly low. Its optimization seems to have been dominated by the other, easier to optimize fitnesses combined in the weighted sum of eq. (1). This is a consistent finding in all 100 runs, as on average the best evolved dungeon sketch per run had f_{area} scores as significantly lowest, with f_{saf} also being significantly lower than the remaining fitnesses; however, balance dimensions (b_{saf}, b_{area}, b_{exp}) and exploration (f_{exp}) were nearly optimal in all runs.

5 DISCUSSION

Experiments in this paper primarily demonstrated how different connectivity requirements of dungeon segments, which in turn are transformed into custom embryogenies for evolution, affect the performance of the constrained optimization algorithm. Layouts such as straight corridors with 2 connections are more difficult to evolve segments for, due to the constraint that monsters block paths from connections to treasures; this obviously depends on the number of monsters and treasures in each segment. In general, the combination of different objectives — with different weights — and the different connectivity requirements can greatly affect the performance of the optimizer and the appearance of resulting segments, as discussed in Section 4.1.3. On the one hand, this demonstrates

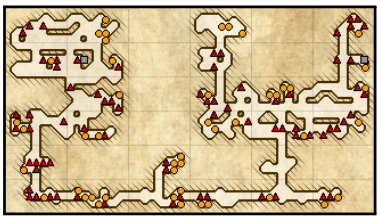


Figure 8: Dungeon created from the sketch of Fig. 7a, with segments of 9 by 5 tiles.

Figure 9: Hand-authored sketch (above) with escalating challenge, and generated dungeon (below).

(a) Distance from the top-left `X` tile (saturated tiles are more distant).

(b) Safety of tiles to the top-left `C` tile (green) and the top-right `C` tile (magenta). Saturated tiles are safer.

Figure 10: Metrics that evaluate the sketch of Fig. 7a can be used to modify the monsters' challenge in each segment based on its distance from the top-left exit in Fig. 10a, or to theme monsters based on the safety of segments to the two `C` segments in Fig. 10b. For instance, the greener the tile in Fig. 10b, the more likely a monster in that segment is a goblin; the more magenta the tile, the more likely it is undead.

that the algorithm as presented here is able to create a broad range of segment patterns, introducing the necessary variety to a dungeon level while still obeying the designer's constraints on playability (i.e. that all game-specific tiles are connected) and challenge (i.e. that all treasures are guarded by monsters). On the other hand, the difficulty in discovering any feasible results can slow down the generative process, as multiple evolutionary trials are needed for corridor and junction segments when generating the dungeon of Fig. 7. This can be addressed at the parameter level (by e.g. increasing the number of generations or the size of the population) or at the game design level (by e.g. increasing the number of monster tiles so that it is easier for them to block paths). More interestingly, however, improvements can look into the genetic operators to ensure, for instance, that a repair function moves monsters into choke points (i.e. tiles with a branching factor of 2) thus increasing the chance that they block paths. Another fundamental improvement can be in the way the infeasible population searches to minimize distance from feasibility. While currently all constraints' distances are added together, a multi-objective approach can be applied to ensure, hierarchically, playability constraints (i.e. paths between connections and all game-specific tiles) and then only if that is the case attempt to ensure that monsters block treasures. This could even be done iteratively, for instance introducing the more challenging blocking monster constraint (which does not strictly affect playability) after the first individual is found which satisfies pure connectivity constraints. A multi-objective approach could also be applied to the feasible population, thus ensuring that not only the exploration objective is optimized (which happened both when evolving segments in Fig. 5 and the sketch in Fig. 7) but also other objectives which were less successful such as the safe areas metric.

The benefits of compartmentalizing the generative process into a high-level dungeon sketch and low-level dungeon segments can be beneficial in many different situations. As the most straightforward application, the same dungeon sketch can be used to generate multiple full dungeons which are to a degree different (e.g. could have different pathways or number of monsters) while still obeying the same high-level patterns of the sketch. Moreover, different dungeons can be created by varying the size of the dungeon segments; Fig. 8 shows a smaller, narrower dungeon evolved from the dungeon sketch of Fig. 7a but with segments of 9 by 5 tiles. More interestingly, any technique can be used to create the low-level dungeon sketch, including other generative algorithms but also human intervention. Since the compact dungeon sketch is stored as a text file (as seen in Fig. 7a), a designer can easily create a custom sketch, which does not need to meet the sketch generator's constraints: an example is shown in Fig. 9 where a custom dungeon is created from a text file with only one exit and an increasing challenge level before the rewards are reaped at the end; the victorious heroes will leave the dungeon from the same exit they entered from. Beyond simple text editing, however, a user interface which allows designers to quickly draft the high-level dungeon sketch can be very beneficial. Via such an interface, the evaluations presented in Section 3.1 can be visualized to provide real-time feedback to the designer while the evolutionary algorithms can be used to create variations of the user's current sketch in a similar fashion to the suggestions in the *Sentient Sketchbook* design tool [11].

This paper focused on the analysis of the currently available segment types, in order to keep objective functions consistent and a small set of tile types (e.g. generic treasures and monsters). However, there are several extensions to the generator which can enrich its outcomes with more or less effort. An obvious extension is the introduction of different types of monsters popular in Role-Playing Games (e.g. goblins, dragonkin, undead, etc.) and treasures (from single-use powerups such as potions to permanent boosts from weapons or armor). Choosing where to place each type of monster and treasure can be based on the dungeon sketch alone in several straightforward ways. The first way is to designate one of the exit tiles as the entrance to the dungeon, and to calculate the distance of each tile in the dungeon from the entrance (see Fig. 10a). Tiles further away from the entrance have monsters of a higher challenge rating (e.g. replacing generic monsters with easy-to-kill goblins near the entrance and powerful dragonkin far away from this entrance) and richer rewards. The second way is to identify "rooms" of highly connected segments (e.g. adjacent

segments sharing 2 or more connections) and theme those with specific monsters and treasure (e.g. the goblin armory). Finally, the safe areas metric (used to calculate f_{area}) to high challenge segments can act as a "boost" to segments' monster difficulty and reward: segments around high-challenge (C) segments will have stronger monsters and richer treasure, but not as strong as those within a C segment itself (see Fig. 10b). Any of these methods for segment variety can also be used to adapt the weight of fitness dimensions for segment evolution (e.g. segments near a C segment put less weight on exploration but double the weight on balance of safe areas). On the other hand, moving the generated dungeons to a playable game can allow audio-visual cues to enhance segment variety: for instance, empty segments can have less ambient light which makes navigation more difficult despite the fact that their exploration score is not much different. A high-challenge segment can be foreshadowed in nearby segments via growls, based on the safe areas metric (see Fig. 10b), while the high-level structure of the dungeon sketch can be presented to the player as a dialog line of a quest-giver NPC, hinting at which areas are safe and which paths should not be taken as they lead nowhere.

6 CONCLUSION

This paper introduced a method for iteratively refining game levels using the concept of sketches at different levels of detail. Firstly, a dungeon sketch is evolved to satisfy connectivity constraints between the two ends (exits) of a dungeon, as well as maximize the exploration effort of a player and the risk/reward options they have to take by placing high reward tiles near high challenge tiles. This dungeon sketch is then refined through a multitude of higher-detail dungeon segments: the dungeon sketch controls the connectivity constraints of each segment, which translate into a custom embryogeny, as well as the objective functions and constraints (e.g. monsters blocking access to treasure) when evolving these segments. Experiments showed that a feasible-infeasible 2-population genetic algorithm is able to evolve segments according to all those criteria, although the different embryogenies and included tiles (e.g. monsters, treasure) resulted in more or less difficult constraint satisfaction tasks. This work can be further extended by adding more types of segments, improving the optimization process with multi-objective approaches on both the feasible and the infeasible population, and by improving the variety of tile types based on metrics or via the aesthetics of a game built around this research.

7 ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 693150.

REFERENCES

[1] Daniel Ashlock and Cameron McGuinness. 2013. Landscape automata for search based procedural content generation. In *Proceedings of the IEEE Computational Intelligence in Games Conference.*

[2] Staffan Björk and Jussi Holopainen. 2004. *Patterns in Game Design.* Charles River Media.

[3] Carlos A. Coello Coello. 2010. Constraint-handling techniques used with evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference.* ACM.

[4] Joris Dormans. 2010. Adventures in level design: generating missions and spaces for action adventure games. In *Proceedings of the FDG Workshop on Procedural Content Generation in Games.*

[5] Joris Dormans and Sander C. J. Bakkes. 2011. Generating missions and spaces for adaptable play experiences. *IEEE Transactions on Computational Intelligence and AI in Games. Special Issue on Procedural Content Generation* 3, 3 (2011), 216–228.

[6] Ken Hartsook, Alexander Zook, Sauvik Das, and Mark O. Riedl. 2011. Toward Supporting Stories with Procedurally Generated Game Worlds. In *Proceedings of the IEEE Conference on Computational Intelligence in Games.*

[7] Daniel Karavolos, Antonios Liapis, and Georgios N. Yannakakis. 2016. Evolving Missions to Create Game Spaces. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG).*

[8] Steven Orla Kimbrough, Gary J. Koehler, Ming Lu, and David Harlan 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.

[9] Antonios Liapis. 2016. Exploring the Visual Styles of Arcade Game Assets. In *Proceedings of Evolutionary and Biologically Inspired Music, Sound, Art and Design (EvoMusArt).* Springer.

[10] Antonios Liapis and Georgios N. Yannakakis. 2015. Refining the Paradigm of Sketching in AI-Based Level Design. In *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference.*

[11] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. 2013. Sentient Sketchbook: Computer-Aided Game Level Authoring. In *Proceedings of the 8th Conference on the Foundations of Digital Games.*

[12] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. 2013. Towards a Generic Method of Evaluating Game Levels. In *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference.*

[13] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. 2015. Constrained Novelty Search: A Study on Game Content Generation. *Evolutionary Computation* 23, 1 (2015), 101–129.

[14] Phil Lopes, Antonios Liapis, and Georgios N. Yannakakis. 2016. Framing Tension for Game Generation. In *Proceedings of the International Conference on Computational Creativity.*

[15] Zbigniew Michalewicz. 1995. Do Not Kill Unfeasible Individuals. In *Proceedings of the Fourth Intelligent Information Systems Workshop.*

[16] Diego Perez, Julian Togelius, Spyridon Samothrakis, Philipp Rohlfshagen, and Simon M. Lucas. 2014. Automated map generation for the physical traveling salesman problem. *IEEE Transactions on Evolutionary Computation* 18, 5 (2014).

[17] Mike Preuss, Antonios Liapis, and Julian Togelius. 2014. Searching for Good and Diverse Game Levels. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG).*

[18] Marco Scirea, Julian Togelius, Peter Eklund, and Sebastian Risi. 2016. Meta-Compose: A Compositional Evolutionary Music Composer. In *Proceedings of Evolutionary and Biologically Inspired Music, Sound, Art and Design (EvoMusArt).* Springer.

[19] Noor Shaker, Antonios Liapis, Julian Togelius, Ricardo Lopes, and Rafael Bidarra. 2016. Constructive generation methods for dungeons and levels. In *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, Noor Shaker, Julian Togelius, and Mark J. Nelson (Eds.). Springer, 31–55.

[20] Noor Shaker, Miguel Nicolau, Georgios N. Yannakakis, Julian Togelius, and Michael O'Neill. 2012. Evolving Levels for Super Mario Bros Using Grammatical Evolution. In *Proceedings of the IEEE Conference on Computational Intelligence and Games.*

[21] Noor Shaker, Julian Togelius, and Mark J. Nelson. 2016. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research.* Springer.

[22] Ruben M. Smelik, Tim Tutenel, Rafael Bidarra, and Bedrich Benes. 2014. A survey on procedural modeling for virtual worlds. *Computer Graphics Forum* 33, 6 (2014), 31–50. http://graphics.tudelft.nl/Publications-new/2014/STBB14 doi: 10.1111/cgf.12276.

[23] Nathan Sorenson and Philippe Pasquier. 2010. Towards a generic framework for automated video game level creation. In *Proceedings of the international conference on Applications of Evolutionary Computation.*

[24] Kenneth O. Stanley and Risto Miikkulainen. 2003. A Taxonomy for Artificial Embryogeny. *Artificial Life* 9, 2 (2003), 93–130.

[25] Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, and Georgios N. Yannakakis. 2010. Multiobjective exploration of the Starcraft map space. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games.*

[26] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. 2011. Search-based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011).

[27] Roland van der Linden, Ricardo Lopes, and Rafael Bidarra. 2014. Procedural generation of dungeons. *IEEE Transactions on Computational Intelligence and AI in Games* 6, 1 (mar 2014), 78–89. http://graphics.tudelft.nl/Publications-new/2014/LLB14 doi: 10.1109/TCIAIG.2013.2290371.