Piecemeal Evolution of a First Person Shooter Level

Antonios Liapis

Institute of Digital Games, University of Malta
antonios.liapis@um.edu.mt,
WWW home page: http://antoniosliapis.com/

Abstract. This paper describes an iterative process for generating multi-story shooter game levels by means of interlocking rooms evolved individually. The process is highly controllable by a human designer who can specify the entrances to this room as well as its size, its distribution of game objects and its architectural patterns. The small size of each room allows for computationally fast evaluations of several level qualities, but these rooms can be combined into a much larger shooter game level. Each room has two floors and is generated iteratively, with two stages of evolution and two stages of constructive post-processing. Experiments in generating an arena-based level for two teams spawning in different rooms demonstrate that the placement and allocation of entrances on each floor have a strong effect on the patterns of the final level.

Keywords: Procedural Content Generation, Level Design, First Person Shooter, Constrained Optimization, Iterative Refining

1 Introduction

While procedural content generation (PCG) has a history of almost four decades in the game industry, it remains a niche —almost mystical— topic for many game development teams. Especially for the PCG research community, communication and collaboration with industry experts and game designers has been sparse. Highlights of academic PCG taking the commercial route usually have the researcher lead the game development cycle; notable examples include *Galactic Arms Race* [1], *Petalz* [2], *Sure Footing* [3], *Unexplored* by Joris Dormans (Ludomotion 2017) and *Darwin's Demons* [4].

Admittedly, a factor for the skepticism of the game industry towards PCG research is the often poor communication between the academic and the game developer community. However, a more tangible factor is that PCG research often builds monolithic generative systems for a specific game; most often, such games are either simplifications of outdated games, such as *Super Mario Bros* (Nintendo 1985), or games designed explicitly for the research problem at hand, such as GVGAI level generation [5]. Even when the generated artefacts could be —under circumstances or after post-processing— applicable to a broader genre of games, developers are skeptical because they do not see a way to control the

generative processes [6]. Being able to both understand (e.g. through visual inspection) and control the outcomes of the generative process is an important reason why commercial generators are often based on *templates*. Indicatively, dungeons in *Diablo III* (Blizzard 2012) consist of large hand-authored 'jigsaw pieces'[1] stitched together by a simple generator that chooses which piece attaches to which, then populates each piece with monsters and treasure. A similar process is followed by *Spelunky* (Mossmouth 2012), where every level is split into 16 'segments' with randomized connections; the layout of each 'segment' is selected from a set of templates and is then randomized somewhat and populated with treasures and monsters [7]. In contrast, academic PCG rarely focuses on modularity for two reasons: (a) often complicated algorithms are included which can not be compartmentalized; (b) in search-based PCG [8] or constraint-based PCG [9], content must usually be in its final form in order to assess its quality (in order to improve it or to test all constraints, respectively).

Providing different degrees of control to human designers has also been a topic of research within PCG, of course. Generators such as those by Doran and Parberry [10] give a multitude of parameters (many of them not entirely intuitive) for the user to tweak. The shader generators of Howlett, Colton and Browne [11] allow the user to specify the target color which evolution will try to match. The editor for *Refraction* allows the designer to specify how each generated level must be solved [12]. *Danesh* [6] allows users to view the generative space and explore more intuitively how the different parameters impact the quality of the results. Generators working alongside designers, e.g. in mixed-initiative design [13], also offer a large degree of control to the designer but are unable to create new content for players during runtime. The academic PCG approach closest to the commercially popular template-based generation is based on grammars [14], where one generator first creates the mission graph —a high-level representation of the final level— and then replaces the high-level nodes of that graph with level architecture which in most cases is based on human-made 'jigsaw pieces'. As an example, in the commercial game *Dwarf Quest* (Wild Card 2013) the mission can be generated from editable grammars describing the allowed sequence of generative commands [15] or evolved towards a designer-chosen fitness function [16]; then the mission is converted into a *Dwarf Quest* level consisting of premade rooms mapped to the description of the node (e.g. a boss room will have a specific room layout, monsters and treasures within it).

Inspired by template-based commercial generators, this paper presents a generative algorithm for first person shooter (FPS) games which allows many degrees of control and outputs *rooms* which can then be combined like jigsaw pieces as desired by game designers (or while the game is played) to create the full map. The generator creates rooms with two floors because many level patterns popular in FPS games, such as a gallery or a sniping position [17], require the presence of areas above and below them. The generative process is iterative, first evolving a draft of the ground floor's architecture, then creating a draft of

[1] Interior jigsaw pieces is a term used by a Blizzard employee on the Blizzard forums under the nickname Bashiok.

the top floor's architecture which is then evolved along with the placement of game objects. A two-step evolutionary process has been shown to be beneficial when evolving levels with two floors [18, 19]. Both evolutionary steps use a constrained optimization algorithm to ensure that the final content is playable; the criteria for what is playable, the room's size and the game items within it are controlled by the designer before evolution begins. More importantly, the designer specifies the number and placement of entrances to the room, which cannot be changed by the evolutionary process. Where entrances are placed affects how different rooms connect to each other, but also affects the patterns favored by the generated content as entrances are considered in the fitness functions of the evolutionary iterations; this will be evaluated in experiments of this paper.

Unlike many other search-based PCG projects in academia, the generative processes in this paper are iterative, modular and parameterizable. While artificial evolution is the core generative method, it takes place in iterations interspersed with post-processing steps which can significantly affect the look and feel of the final level. Moreover, the evolutionary steps themselves integrate several constraints which are not strictly based on playability but rather on the look and feel of the level. These additions were made at the request of game designers and with specific gameplay concerns in mind, during the collaboration with Luiz Kruel and his game development team[2]. The room-based generation which allows for modular re-combination of different pieces (generated or human-authored) was found preferable to level designers than a monolithic level generation algorithm. Controlling the room size and entrance placement was similarly vital in order to integrate procedural generation to the team's design process. Dead-end removal in post-processing steps was deemed essential because dead-ends disrupt the run-and-gun FPS gameplay by forcing players to become "trapped" and vulnerable to an ambush. Finally, constraints on open air tiles and walls in each evolutionary step allows designers further control over the patterns they would prefer in the generated level. This parameterized design space gives sufficient control to designers in order to control the look and play patterns that are produced by the generator.

2 Background Work on Map Sketches

The notion of a *map sketch* originates from *Sentient Sketchbook* [20], and encompasses all low-resolution representations of game levels which contain the minimal number of tile types needed to describe the intended gameplay. For example, in real-time strategy games the core necessary tile types are impassable terrain, players' bases, resource-rich locations around the map, and empty passable tiles. For the shooter genre, the core tile types could be the players' (or teams') spawn locations, weapon pickups, and possibly also health pickups if the game does not feature regenerating health [21]. Since the low-resolution sketch

[2] A talk by Luiz Kruel at the 2017 Game Development Conference (GDC) highlighted the generative pipeline followed in this paper, focusing on the transformation of the top-level views of this approach into 3D levels.

is composed of a few tiles of even fewer types, it can be evolved using a direct encoding [8] with genetic operators changing each tile individually, e.g. changing from one tile type to another or swapping adjacent tiles.

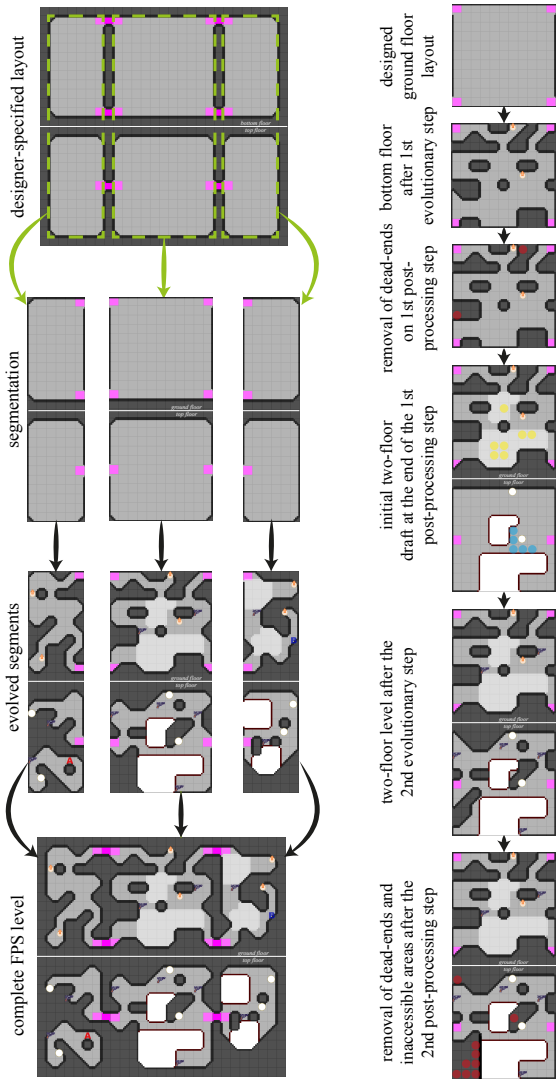
Map sketches are general level descriptors as they can be translated (or represented) at a post-processing *detailing* step into a high-resolution game level in 2D or 3D [21]. A number of general evaluations of game level patterns [21] have been designed with map sketches in mind, focusing on the patterns of *exploration*, *balance* (or symmetry) and *strategic resource control* introduced by Bjork and Holopainen [22]. These evaluations are straightforward to compute via pathfinding (for resource control) and flood fill (for exploration) algorithms, and are lightweight in the small map sizes of sketches. This paper iteratively evolves FPS rooms towards improving exploration and safe area evaluations: the former captures the effort taken to reach one or more tiles starting from another tile, while the latter counts the number of tiles (of any type) which are much closer to one tile than to all other tiles of the same type. Balance for exploration and safe area assesses whether some tiles are easier to find than others or some tiles have far more safe tiles around them than others, respectively. Details of the calculation of these fitness dimensions are provided by [21].

3 Methodology

The levels generated by the algorithm are composed of two floors, one placed on top of the other. Players can move from the ground floor to the top floor through *jump pads*, which lead to *jump landings* on the top floor right above them. Players can move from the top floor to the ground floor through the same jump landing holes, but also through open air tiles which do not have a floor and allow players to jump down to adjacent tiles on the ground floor. Since the levels are intended for team-based deathmatch gameplay, tiles where each team starts from are included (spawn points). Moreover, the level contains weapon pickup tiles which allow players passing through them to gain more powerful weapons (and replenish their ammo). Both floors can contain wall tiles which block line of sight and movement through them. As with all map sketches, these few tile types (in addition to entrance tiles which have no gameplay effect but are vital for connecting rooms together) constitute the minimal components[3] that facilitate FPS gameplay in levels with more than one floor.

The principle behind the entire generative pipeline presented in this paper is iterative design, summarized in Figure 1. As shown in Figure 1a, each level is split into rooms which are evolved separately, one at a time. When evolving each room, the iterative process becomes obvious (see Figure 1b): note that each step for individual room generation is automated and does not require designer intervention. First, evolution produces a draft of the ground floor, laying out impassable tiles, jump pads and entrances. The fittest ground floor draft is used

[3] Unlike [21], health pickups are not included following discussions with game designers who did not consider them necessary; health pickups they easily be placed in a constructive fashion post-generation or omitted due to re-generating health.



(a) Overall generative pipeline (b) Step-by-step room generation

Fig. 1. Overall generative pipeline for partitioning and combining rooms (Fig. 1a); the designer specifies how the rooms are sized and where their entrances (in magenta) are placed, and the iterative generation results in two-floor rooms which can be combined into the final shooter level. Each room is generated through a step-wise process shown in Fig. 1b, where an evolved ground floor which after post-processing generates a first draft of the top floor; then both floors are evolved and finalized through a second post-processing step.

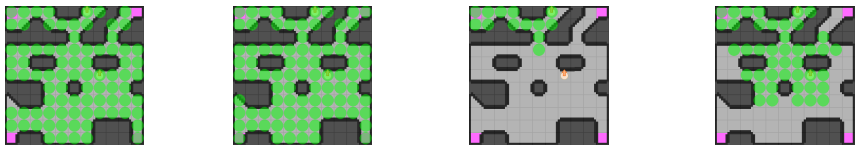
as a basis for creating the two-floor level; initially, the only tiles of this draft top floor are open air tiles, jump landings and empty tiles (which are specified by the patterns of the floor under them) and the entrances (which are specified by the user). After this process, all tiles specified by the above two-step drafting process are "frozen": evolution can not change pre-existing walls, jump pads/landings or open air tiles during subsequent stages, although it can change empty tiles normally. The second stage of evolution populates both floors with weapons and spawn points and may also add walls on either floor. The fittest two-floor level is then post-processed to remove unused corridors and jump pads/landings. Once this process is complete for each individual room, the final rooms are connected together as desired by the designer to form the final FPS game level.

3.1 Evolving the ground floor

The first stage of generation is the evolution of a first draft of the ground floor, the structure of which will form a draft for the top floor. In this stage only the ground floor is evolved: the genotype stores only half of all tiles in the room. Evolution starts from an initial population containing the designer-specified number of jump pads and a number of wall tiles equal to 10% of all tiles randomly placed in the room, as well as the entrances placed where the designer has requested them; entrance tiles will not be changed or moved by evolution. Evolution uses mutation alone to (a) transform an unoccupied passable tile into a wall or vice versa (25% chance), or (b) swap any tile with an adjacent one (5% chance). Following the literature [21], between 5% to 20% of all tiles can be mutated in this way per individual. In order to ensure that evolving levels satisfy certain playability constraints, evolution in carried out in two separate populations using the feasible-infeasible two-population genetic algorithm (FI-2pop GA) paradigm [23]. Each population contains only individuals which satisfy all playability constraints (the *feasible* population) or individuals which fail them (the *infeasible* population); offspring of feasible individuals may be infeasible in which case they migrate to the infeasible population, and vice versa. Both populations evolve to maximize or minimize a fitness value, using fitness-proportionate roulette-wheel selection and minimal elitism: the best individual in each population is copied unchanged to the population of the next generation.

The feasible population evolves to maximize a sum of three equally weighted fitnesses formulated in [21]: (a) exploration from any entrance or jump pad to any other entrance or jump pad, (b) balance of exploration from any entrance to any entrance, (c) balance of exploration from any entrance to any jump pad. On the one hand, exploration rewards all entrances and jump pads which are far away, but the two balance metrics distinguish between entrances (which may be closer together due to designer specifications) and jump pads; the two jump pads should be equally hard to reach from all entrances, but that does not need to be comparable to the exploration effort between two entrances. The calculation of exploration is shown graphically in Fig. 2.

Meanwhile, the infeasible population contains individuals which fail one or more of these constraints: (a) all entrances and jump pads must be connected



(a) Exploration from TLE to BRE. (b) Exploration from BRE to TLE. (c) Exploration from TLE to top JP. (d) Exploration from top JP to TLE.

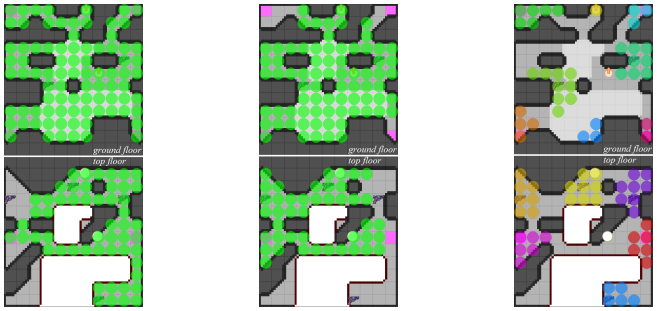
Fig. 2. Fitness calculation of the first evolutionary step: exploration between entrances (top left as TLE and bottom right as BRE). Exploration effort from TLE to BRE is almost as high as from BRE to TLE (i.e. two fewer tiles covered). On the other hand, from TLE the nearest jump pad (JP) is easily found, while starting from that JP the exploration required to find TLE is much higher, and the two efforts are not balanced.

via passable paths, (b) jump pads are not placed under entrance tiles on the top floor, (c) entrances or jump pads are at least two tiles away from any entrance or jump pad, (d) the number of walls does not exceed 30% of the floor's tiles and (e) the number of tiles that have all their neighbors passable does not exceed 10% of the floor's tiles. Some of these constraints are self-evident; all entrances and jump pads must be connected so that a player can use them but also so that the exploration metric can be calculated. Other constraints are based on designer feedback, so that maps are not overwhelmed by walls and maze-like corridors but also not leading to overwhelming open air sections in the top floor (discussed in Section 3.2). The infeasible population evolves to minimize an infeasible fitness function proportionate to the distance to feasibility, which is a sum of features that violate constraints (e.g. the number of disconnected entrances and jump pads or the number of game elements closer than two tiles to each other).

3.2 Creating the top floor from the ground floor

Once the first evolutionary step is complete, the fittest individual in the final population becomes the basis for the first draft of the top floor. Before this happens, however, a constructive algorithm removes corridors leading nowhere (i.e. dead-ends); this level pattern was found undesirable for the FPS genre as players could easily be ambushed, or must backtrack their steps which is not desirable for the run-and-gun aesthetic of such games. This step iteratively replaces unoccupied dead-ends with a wall (red circles in Fig. 1b). A *dead-end* is a tile with only one connection (on the navigation mesh) to adjacent passable tiles. Dead-ends with jump pads or entrances are not removed as they still allow movement to the top floor or to the next room respectively.

Once dead-ends are removed, the top floor is initialized with entrances (as specified by the designer), jump landings (on the same locations as their jump pad counterparts on the ground floor), passable platforms and open air tiles. Open air tiles represent areas of the top floor which have no solid flooring, allowing the player to aim at opponents on the ground floor or jump down to it.



(a) Exploration effort from ground floor.

(b) Exploration effort from top floor.

(c) Safe areas around weapons & entrances.

Fig. 3. Exploration effort from the bottom right (BRE) entrance on the ground floor to the left (LE) entrance of the top floor (Fig. 3a) which requires use of jump pads. Compare with the exploration effort from LE which uses the open air sections to jump down and quickly find BRE (Fig. 3b). The other fitness when evolving both floors is the number of safe tiles to an entrance, weapon, or spawnpoint (Fig. 3c). Two weapons on the top floor have safe tiles on both floors: the weapon with blue safe tiles (due to open-air tiles); the weapon with yellow safe tiles (due to the nearby jump landing).

In order to maintain some structural integrity with the ground floor, only large areas in the ground floor have open air sections above them. This is calculated based on the navigation mesh by finding tiles on the ground floor which are fully connected, i.e. all their neighbors are passable tiles. If there is a fully connected tile in the ground floor (yellow circles in Fig. 1b), then the top floor tile above it and its immediate neighbors become open air tiles with one exception: if any open-air tile inserted in this way is adjacent to an entrance or jump landing, then it is replaced by a passable platform instead (such exempt tiles appear as cyan circles in Fig. 1b). This exception ensures that jump landings and entrances always have a platform around them for the player to land on or survey the level.

All rooms with two floors have a special navigation mesh which provides connections across floors. Two-way connections are created between jump pads and jump landing tiles, which is the only way that a path from the ground floor to the top floor can be computed. The system also creates one-way connections between platforms on the top floor and passable tiles on the ground floor which are over open air tiles adjacent to these platforms. The navigation mesh assumes that players can not jump more than a tile's worth of distance, and thus can not jump over an open air tile to another platform on the top floor.

3.3 Evolving both floors

Once the layout of the top floor is created, both floors go through another evolutionary step which adds walls, weapon pickups and the teams' spawn points in the room. Using the same FI-2pop GA and parameters (including the mutation

operators and probabilities), evolution starts by randomly allocating in both floors the weapons and spawn points. The number of weapons and spawn points are the same in all evolving levels, and are specified by the designer. Unlike the previous evolutionary stage, no additional wall tiles are placed in the initial population but are added via mutation (which changes passable tiles to walls). This mutation can add walls on either floor in theory; however, most of the level structure of the ground floor is "frozen", so walls seem to be added almost exclusively to the top floor. Elements evolved (and post-processed) from the previous steps are "frozen", which allows for the room to be refined in steps, adding new elements (e.g. weapons) but respecting designs finalized in past steps.

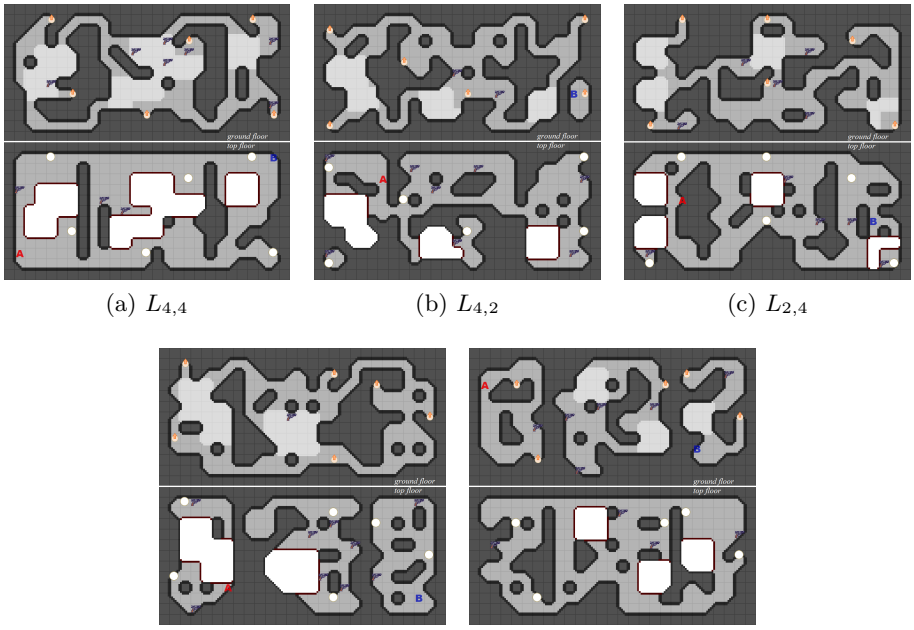
Evolution is carried out in a feasible and an infeasible population as described in Section 3.1. The feasible population attempts to maximize a sum of these equally weighted fitnesses: (a) exploration from any entrance, spawn point or jump pad/landing and (b) the balance thereof, (c) the number of safe tiles around weapons, spawn points and entrances and (d) the balance thereof. Any passable tile is safe for e.g. a weapon if the tile's distance to this weapon is half or less of the distance to the next closest weapon, spawn point or entrance. Formulations and parameters of these metrics are found in [21]. The constraints for feasible individuals are: (a) all entrances, jump pads/landings, weapons and spawn points must be reachable via a passable path (including jumps through open air tiles), (b) the number of total walls does not exceed 30% of the total room's tiles.

3.4 Post-processing to create the final room

Once evolution of two-floor rooms is complete, the fittest feasible individual is chosen as the final result. Several post-processing steps are applied before it is presented to the designer. Dead-ends are again iteratively filled with wall tiles unless they are occupied by entrances, jump pads/landings, weapons or spawn points; the process continues until no more walls can be placed. After this step, jump pads or landings which are surrounded only by wall tiles or open air tiles are removed along with their corresponding tile on the other floor. This step is necessary in case the jump pad leads nowhere, especially after dead-end removal. After this step, inaccessible areas on both floors are filled with wall tiles; this is achieved via flood-fill algorithms originating from the jump pad tiles (for the ground floor) or the jump landings (for the top floor). All tiles removed in this fashion are shown as red circles in Fig. 1b.

4 Experiments

Five example layouts for FPS levels will be tested in this paper, to evaluate the impact that entrance placement has on the evolved rooms. For the sake of simplicity, three rooms are evolved per level: a large square *arena* of 11 by 11 tiles, and two rooms of 6 by 11 tiles acting as *bases* —each hosting a team's spawn point. The three rooms are horizontally aligned and are simply joined together



(a) $L_{4,4}$ (b) $L_{4,2}$ (c) $L_{2,4}$

(d) $L_{4,0}$ (e) $L_{0,4}$

Fig. 4. Sample evolved levels for different layouts.

side-by-side in the end, as in Fig. 1a. Experiments in this paper test five configurations of entrances for each of these rooms (entrances are mirrored between adjacent rooms). The arena can have two entrances to each base either on the top or on the ground floor, four entrances (two per floor), and combinations of three entrances. In all cases, each base has one spawn point tile and two weapon tiles while the arena has six weapon tiles; all rooms have two jump pads and two landing tiles. From this point forward, the different layouts are identified as $L_{i,j}$ where i and j is the number of entrances on the ground floor and on the top floor respectively: $L_{4,2}$ has four entrances on the ground floor and two entrances on the top floor (see Fig. 1a). In the visualization of Fig. 4, spawn points are shown as A and B in the left and right base respectively; open air tiles are shown as white and passable tiles and platforms in light gray.

In this experiment, 100 evolutionary runs are used to create the total of 15 rooms (3 per layout). Each evolutionary step is performed on a population of 100 individuals evolving for 20 generations. Indicative results of evolved levels for the different layouts are shown in Fig. 4. Each of the sample levels has a fair number of open-air tiles, although not always in every room (e.g. in the right base of Fig. 4d or the left base of Fig. 4e). Except for Fig. 4a, the top floor has passages that are almost as winding as the ground floor. Except for Fig. 4c, it seems that both spawn points are often found on the floor with the fewest

entrances. The general patterns of evolved levels for different layouts in terms of tile placement and winding passageways will be explored in Sections 4.1 and 4.2. Meanwhile, it is interesting to note some properties of the specific levels of Fig. 4. Walls and open air tiles tend to create 'islands' on the top floor which can not be accessed from other parts of the top floor without traversing the ground floor and ascending again through a jump pad. Examples include parts similar to sniper positions which allow players to aim at opponents on the floor below via adjacent open air tiles (e.g. left base in Fig. 4e, or left and right base in Fig. 4c). In other cases, such as the left base in Fig. 4b, these islands are surrounded by walls and act more as a fortified position for a defending player: opponents can only attack through the singular jump pad; defending the spot is easy but leaving it again is almost impossible. This pattern of a locked-in 'island' accessible only via jump pad can also be found on the ground floor, for instance in the right base of Fig. 4b. Another interesting pattern is found in Fig. 4a, where open air tiles in the central arena partition the floor into two equally-sized platforms (only accessible via a narrow passage surrounded by walls on the left of the arena); in this case, the ground floor is a shortcut via the jump pads near each platform.

4.1 Comparing level structures

Table 1 shows the number of tiles in each level (i.e. the three rooms joined together) and how they are distributed between the two floors. Results are collected from 100 independent evolutionary runs per layout. Obviously the number of weapons and spawn points is always the same in every layout (10 and 2 respectively); the number of entrances depends on the layout but does not differ among runs using the same layout. Since the last post-processing step removes jump pads/landings leading nowhere, the number of jump pads is on average less than what was specified by the designer (6), but not by much. In 75% of runs (across all layouts) the number of jump pads was indeed 6; this number is not affected by layout much. The layout does affect the number of wall tiles and open air tiles: despite constraints, there is sufficient leeway for important differences. There are certain patterns worth noting: layouts with few entrances on the ground floor ($L_{2,4}$, $L_{0,4}$) have fewer open air tiles than other layouts. Layouts with more entrances on the ground floor ($L_{4,2}$, $L_{4,0}$) have more wall tiles than other layouts. Interestingly, $L_{4,0}$ has far more wall tiles than all others but also far more open air tiles, likely due to the second post-processing step filling unreachable areas (due to lack of entrances) with walls on the top floor.

A much clearer picture is gleaned from the ratios of tiles per floor. Table 1 summarizes this, as the ratio of tiles of one type on the ground floor over the total number of tiles of that type. Due to the five layouts chosen, the entrance ratio on the ground floor is unique for each layout (bold in Table 1). Using this as the primary characteristic to compare effects of the layout, we observe statistically significant Pearson correlations ($p < 0.05$) with all other tile ratios shown in Table 1. Specifically, there are very strong negative correlations between entrance ratio and wall ratio ($\rho = -0.941$), weapon ratio ($\rho = -0.959$), spawn point ratio ($\rho = -0.906$) and game element ratio, i.e. weapons and spawn points

Table 1. Average number of tiles, tile ratios and metrics for the different tested layouts. Results are averaged from 100 generated levels, along with the 95% confidence interval. The game elements metric encompasses weapon and spawn point tiles.

Layouts	$L_{4,4}$	$L_{4,2}$	$L_{2,4}$	$L_{4,0}$	$L_{0,4}$
Wall tiles	326±3	349±2	334±3	364±3	335±3
Open air tiles	41±2	41±2	36±2	47±3	30±3
Jump pad tiles	5.8±0.1	5.7±0.1	5.8±0.1	5.8±0.1	5.8±0.1
Ratio of tiles on the ground floor					
Entrances	**0.50**	**0.67**	**0.34**	**1.00**	**0.00**
Walls	0.48±0.00	0.45±0.00	0.49±0.00	0.43±0.01	0.50±0.00
Weapons	0.48±0.03	0.39±0.03	0.55±0.02	0.38±0.03	0.64±0.02
Spawn Points	0.15±0.05	0.11±0.04	0.36±0.07	0.08±0.04	0.76±0.06
Game Elements	0.43±0.02	0.34±0.03	0.51±0.02	0.33±0.03	0.66±0.02
Level patterns and fitnesses					
Spawn distance	31.8±0.9	34.8±1.0	35.4±1.3	43.4±1.4	39.4±1.7
Spawn exploration	0.92±0.011	0.94±0.008	0.93±0.01	0.94±0.008	0.91±0.014
Spawn explor. balance	0.95±0.009	0.94±0.010	0.93±0.01	0.95±0.008	0.93±0.013
Weapon area	0.33±0.010	0.35±0.012	0.36±0.012	0.38±0.016	0.41±0.012
Weapon area balance	0.57±0.013	0.58±0.014	0.58±0.012	0.56±0.017	0.59±0.012

combined ($\rho = -0.953$). Finally, there is a significantly positive correlation between entrance ratio and total open air tiles in the level ($\rho = 0.988$); a strong correlation with the number of total wall tiles ($\rho = 0.755$) is however not significant due the small set of 5 layouts tested. Even with the few layouts tested in this paper, there are clear trends which are common-sensical given the fitnesses used for feasible individuals in both evolutionary stages. For the first stage, when there are few or no entrances on the ground floor, exploration focuses only on placing jump pads as far apart as possible. This is easier than trying to improve exploration of e.g. 6 tiles (4 entrances and 2 jump pads in the arena), so the room does not need many walls to create winding passageways. In the second stage, exploration favors spawn points as far away from jump pads/landings and entrances as possible; tile safety favors weapons far away from entrances and spawn points. When entrances are only on the top floor ($L_{0,4}$) then obviously the most distant spots for weapons or spawn points are on the ground floor. Since weapons must also be far away from each other, on the other hand, it is often the case that a few weapons are on the floor with more entrances while most weapons are on the floor with fewer entrances.

4.2 Comparing level patterns

While ultimately a playtest with human players must test the playability of each level, some gameplay qualities can be estimated based on the distance and exploration effort between the two teams' spawn points as well as how spaced apart weapons are. Distance between spawn points is calculated on the shortest path; exploration between spawn points, safe areas around weapons (and their

balance) are calculated through the same formulas as for evolution (except no other tiles are considered), and the whole level is assessed rather than each room.

Table 1 shows the average fitness scores and metrics of 100 generated levels per layout. The high value for spawn point exploration (and its balance) indicates that players of both teams will explore most of the level to find the opponents' spawn point, regardless of layout. Exploration for $L_{4,0}$ and $L_{4,2}$ has almost identical scores, which are significantly higher than those of $L_{0,4}$ and $L_{4,4}$. After a few minutes of playtime, however, players will have identified the other team's spawning area so the more pertinent metric is the distance between spawn points: the lowest distance is for $L_{4,4}$ and the highest for $L_{4,0}$ (both findings are statistically significant). The other layout without entrances on one floor ($L_{0,4}$) has significantly higher spawn point distances than all other layouts except $L_{4,0}$. There are therefore some similarities between the patterns shown from spawn point exploration and their distance: the few entrances of $L_{4,0}$ lead to longer paths that are more complex to follow, while the many entrances of $L_{4,4}$ provide shortcuts to opposing players and makes spawn points easier to find. While neither distance nor exploration effort of spawn points is explicitly targeted by evolution (since no room has more than one spawn point), when rooms are combined these desirable patterns emerge as each room rewards exploration between entrances (and one spawn point, in the case of base rooms).

On the other hand, roughly 40% of tiles in the levels are much closer to one weapon than to others but not all weapons have similar number of safe tiles around them. This admittedly points to a sub-optimal level pattern, as some weapon locations may be more easily reachable and thus preferred to others. It is not surprising however that any algorithm struggles to balance the placement of so many weapon pickups. Moreover, the rooms are not equal: the arena has double the size and triple the number of weapons of each base. Weapons in bases have far more safe tiles than those in arenas, which explains the imbalance of this metric. Here, splitting the level into rooms is detrimental as the algorithm has no way of knowing how other rooms place weapons. There is no easy way of alleviating that fact except by having weapons proportionate to each room's area, yet this would remove much of the benefit of ad-hoc controllability of each room by the designer. Regarding differences between layouts, the only significant difference worth noting is for $L_{4,4}$ which has the lowest weapon area score than all other layouts, likely due to the fact that the arena has to optimize the safe areas around 8 entrances and 6 weapons versus fewer entrances of other layouts.

5 Discussion

The modular way in which the level is built out of components offers substantial control to the designer. While experiments focused on a small subset of possible level layouts, there are endless possibilities for how custom-defined rooms can be combined together. For example, the current rooms can be combined to form four-floor shooter levels, by placing $L_{4,4}$ bases on either side of a stacked set of $L_{0,4}$ (top) and $L_{4,0}$ (bottom) arena. The current two-floor rooms can also

be combined with single-floor rooms evolved in a variant of the second evolutionary and post-processing step, to create two-story levels with an internal courtyard, for instance. Finally, the designer can combine evolved rooms with human-authored ones, such as a manually designed corridor or gallery to connect two arenas. The ability to specify rooms of any size, any connectivity and however many game objects within it, as well as the ability to combine rooms in many ways with other generated or human-authored rooms affords an exceptionally large design space for both designer and algorithm to explore. While the connectivity between rooms, and the properties of the rooms themselves, are currently controlled by human designers, an extension of the current system could allow another generator to produce the high-level layouts (i.e. room sizes and how they connect to each other) in a similar fashion as [24].

While the different entrance setups can affect the patterns that emerge, all tested layouts seem to achieve high-quality results with diverse pathways connecting one team's spawn area to the other and weapons which generally are not clumped together. Further experimenting with layouts, room sizes, and the many controllable parameters of the generator (maximum ratio of walls or open air tiles, number of weapons and spawn points) could allow designers to find the perfect setup for parts of a shooter level (e.g. a "bunker" or a "sniper's nest" room). The relatively large number of parameters in the generative system are admittedly kept constant for the most part in the experiments of this paper; the values chosen however came from experimentation and designer feedback by Luiz Kruel. The system proved quite sensitive to constraints on minimum wall ratio and maximum open air ratio, which affected discovery of any feasible results.

The work presented here is intended for use in a commercial game, leading to specific design decisions, post-processing steps, fitness functions and constraints. While this system outputs 2D maps, converting them into 3D is relatively easy with *Houdini*, an engine developed by SideFX; the 3D output of the two-floor levels of this paper for use in the *Unreal Engine 4* (Epic Games 2014) was shown in the GDC 2017 talk by Luiz Kruel. The generative system is a specialized version of the general generative framework of [21], since the included game objects and navigational patterns are both biased by and biasing the design options for such a game. For instance, the tile-based representation complicates the addition of larger structures which span multiple tiles (e.g. staircases): jump pads/landings are convenient for the generator but may limit designers. On the other hand, the team-based gameplay of the intended game guides the sparse use of spawn points and the fitness functions that assess them, which favor 'hiding' them away from entrances to prevent enemy camping. In a free-for-all deathmatch game, the generator would likely benefit from more spawn points which could be evaluated differently: see [18, 25] for examples. Finally, dead-end removal was requested by FPS level designers: the post-processing steps provided an easy fix to this problem, but at the cost of the resulting room being at times much different than what is being evolved (and evaluated). A different representation, for instance based on rooms as in [26], could make removing deadends and placing larger structures more straightforward.

6 Conclusion

This paper described a multi-step process for generating rooms of a shooter game which can be specified *a priori* and re-combined *a posteriori* by a human designer. Generated two-floor levels allow for level patterns such as sniping positions and shooting galleries to emerge as a byproduct of multiple evolutionary steps which target the core qualities of each room: exploration effort from different locations of the room and dispersal of game items within it. Experiments tested several room layouts, combining them into a simple arena-based shooter level for two teams, and showed that the placement of entrances affects how game items are allocated on each floor, and thus indirectly which floor will be visited more frequently by human players. However, different layouts do not severely impact the general qualities of the generated levels (e.g. the effort of reaching opponents' bases). As the game is developped, these hypotheses must be tested with human players competing in 3D levels generated from different layouts.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 693150. The generator was the result of a collaboration with Luiz Kruel, who provided valuable feedback on level patterns and designer constraints for using the system.

References

1. Hastings, E.J., Guha, R.K., Stanley, K.O.: Automatic content generation in the galactic arms race video game. IEEE Transactions on Computational Intelligence and AI in Games 1(4) (2009)
2. Risi, S., Lehman, J., D'Ambrosio, D.B., Hall, R., Stanley, K.O.: Combining search-based procedural content generation and social gaming in the petalz video game. In: Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (2012)
3. Dewsbury, N., Nunn, A., Syrett, M., Tatum, J., Thompson, T.: Scalable level generation for 2d platforming games. In: Proceedings of the FDG Workshop on Procedural Content Generation (2016)
4. Soule, T., Robison, B.D., Heck, S., Haynes, T.E., Street, D., Wood, N.: Darwin's demons: Does evolution improve the game? In: Proceedings of the European Conference on the Applications of Evolutionary Computation (2017)
5. Khalifa, A., Perez-Liebana, D., Lucas, S., Togelius, J.: General video game level generation. In: Proceedings of the Genetic and Evolutionary Computation Conference (2016)
6. Cook, M., Gow, J., Colton, S.: Danesh: Helping bridge the gap between procedural generators and their output. In: Proceedings of the FDG workshop on Procedural Content Generation (2016)
7. Shaker, N., Liapis, A., Togelius, J., Lopes, R., Bidarra, R.: Constructive generation methods for dungeons and levels. In: Shaker, N., Togelius, J., Nelson, M.J. (eds.) Procedural Content Generation in Games: A Textbook and an Overview of Current Research, pp. 31–55. Springer (2016)

8. Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C.: Search-based procedural content generation: A taxonomy and survey. IEEE Transactions on Computational Intelligence and AI in Games 3(3) (2011)
9. Smith, A.M., Mateas, M.: Answer set programming for procedural content generation: A design space approach. IEEE Transactions on Computational Intelligence and AI in Games 3(3), 187–200 (2011)
10. Doran, J., Parberry, I.: Controlled procedural terrain generation using software agents. IEEE Transactions on Computational Intelligence and AI in Games 2(2), 111–119 (2010)
11. Howlett, A., Colton, S., Browne, C.: Evolving pixel shaders for the prototype video game subversion. In: Proceedings of AISB'10 (2010)
12. Smith, A.M., Butler, E., Popovic, Z.: Quantifying over play: Constraining undesirable solutions in puzzle design. In: Proceedings of the International Conference on the Foundations of Digital Games (2013)
13. Yannakakis, G.N., Liapis, A., Alexopoulos, C.: Mixed-initiative co-creativity. In: Proceedings of the 9th Conference on the Foundations of Digital Games (2014)
14. Dormans, J., Bakkes, S.C.J.: 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), 216–228 (2011)
15. van der Linden, R., Lopes, R., Bidarra, R.: Designing procedurally generated levels. In: Proceedings of the AIIDE Workshop on Artificial Intelligence in the Game Design Process (2013)
16. Karavolos, D., Liapis, A., Yannakakis, G.N.: Evolving missions to create game spaces. In: Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG) (2016)
17. Hullet, K., Whitehead, J.: Design patterns in fps levels. In: Proceedings of the Foundations of Digital Games Conference (2010)
18. Cachia, W., Liapis, A., Yannakakis, G.N.: Multi-level evolution of shooter levels. In: Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference (2015)
19. Liapis, A., Yannakakis, G.N.: Refining the paradigm of sketching in AI-based level design. In: Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference (2015)
20. Liapis, A., Yannakakis, G.N., Togelius, J.: Sentient sketchbook: Computer-aided game level authoring. In: Proceedings of the 8th Conference on the Foundations of Digital Games. pp. 213–220 (2013)
21. Liapis, A., Yannakakis, G.N., Togelius, J.: Towards a generic method of evaluating game levels. In: Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference (2013)
22. Bjork, S., Holopainen, J.: Patterns in Game Design. Charles River Media (2004)
23. Kimbrough, S.O., Koehler, G.J., Lu, M., Wood, D.H.: 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)
24. Liapis, A.: Multi-segment evolution of dungeon game levels. In: Proceedings of the Genetic and Evolutionary Computation Conference (2017)
25. Cardamone, L., Yannakakis, G.N., Togelius, J., Lanzi, P.L.: Evolving interesting maps for a first person shooter. In: Proceedings of the Applications of evolutionary computation (2011)
26. Lopes, P., Liapis, A., Yannakakis, G.N.: Targeting horror via level and soundscape generation. In: Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference (2015)