A General-Purpose Expressive Algorithm for Room-Based Environments

Konstantinos Sfikas
Institute of Digital Games
Msida, Malta
konstantinos.sfikas@um.edu.mt

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

Georgios N. Yannakakis
Institute of Digital Games
Msida, Malta
georgios.yannakakis@um.edu.mt

ABSTRACT

This paper presents a generative architecture for general-purpose room layouts that can be treated as geometric definitions of dungeons, mansions, shooter levels and more. The motivation behind this work is to provide a design tool for virtual environments that combines aspects of *controllability*, *expressivity* and *generality*. Towards that end, a two-tier level representation is realized, with a graph-based *design specification* constraining and guiding the generated geometries, facilitated by constrained evolutionary search. Expressivity is secured through quality-diversity search which can provide the designer with a broad variety of level layouts to choose from. Finally, the generator is general-purpose as it can produce layouts based on different types of static grid structures or as free-form, curved structures through an adaptive Voronoi diagram that is evolved along with the level itself. The method is tested on a variety of design specifications and grid types, and results show that even with complex design constraints or malleable grids the algorithm can produce a broad variety of levels.

CCS CONCEPTS

• **Applied computing → Computer games**; • **Computing methodologies → Generative and developmental approaches**; • **Human-centered computing →** *Visualization techniques.*

KEYWORDS

level generation, controllability, evolutionary algorithms, quality diversity search, constrained optimization

ACM Reference Format:
Konstantinos Sfikas, Antonios Liapis, and Georgios N. Yannakakis. 2022. A General-Purpose Expressive Algorithm for Room-Based Environments. In *FDG '22: Proceedings of the 17th International Conference on the Foundations of Digital Games (FDG '22), September 5–8, 2022, Athens, Greece.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3555858.3563262

1 INTRODUCTION

This paper addresses the problem of generating room-based environments that can apply to a broad range of games including dungeon crawlers, horror games, first-person or top-down shooters and many more. We especially focus on addressing how procedural content generation (PCG) can help level designers to clearly define their design goals and then efficiently explore the space of possible designs, thus gaining a deeper understanding of the problem at hand and being able to make well-informed decisions.

Consequently, our proposed methodology combines aspects of controllability, expressivity and generality. Controllability refers to two aspects. First, the designer's ability to make specific requests in regard to the generated levels' typology and have the algorithm generate solutions that respect them. Second, the designer's ability to select the dimensions of diversity that should be explored. Expressivity refers to the algorithm's ability to generate diverse solutions, i.e. ones that are novel from past or concurrent solutions [28]. Finally, generality refers to the flexibility of the content representation that makes it applicable to many different scenarios.

We incorporate controllability primarily by having the designer provide an abstract, topological definition of their desired outcome, the *Design Specification* (DS). The DS describes the number and size of rooms and their connections in an intuitive, visual way. Given a DS, the algorithm will generate a set of *Design Implementations*, i.e. concrete, geometric designs that describe the actual boundaries of rooms and the placement of doors that connect them, while respecting the constraints posed by the DS. This distinction between the topological and geometric characteristics of designed spaces has been proposed and utilized, in various forms, in the fields of PCG [5, 12, 34] and evolutionary architectural design [3, 24, 29, 41], both of which have been an inspiration for this approach.

A secondary aspect of controllability is the designer's ability to select a set of Behavioral Characterizations (BCs) which define the search-space and characterize the algorithm's expressive range. In this study we examine a specific pair of BCs, however the algorithm's implementation offers more options that the designer may choose from. The available set of BCs can also be easily expanded, thanks to the modular implementation of our framework.

Expressivity is another central aspect of our approach. It refers to the algorithm's ability to generate a highly diverse set of solutions, per run, i.e. to exhibit a large Expressive Range, as defined in [32]. Importantly, this aspect cannot be addressed in isolation, but must align with both aspects of controllability: (a) designer-imposed and other constraints and (b) designer-selected BCs. Constraint-solving in itself can be efficiently addressed through declarative programming [4, 27, 31], yet such approaches generate a single solution per run and there is no intuitive way of guaranteeing diversity among generated solutions across runs. In order to address expressivity and constraint-solving as one problem, we follow the broad paradigm of PCG driven by Quality-Diversity (PCG-QD) [7] and implement a constraint-solving illumination algorithm [26], inspired by similar work [1, 8, 14, 20]. In combining controllability

FDG '22, September 5–8, 2022, Athens, Greece
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9795-7/22/09…$15.00
https://doi.org/10.1145/3555858.3563262

with constrained expressivity, our proposed methodology becomes a powerful tool that can illuminate the feasible and infeasible parts of the solution space, thus providing the designer with a broad understanding of the problem at hand.

Finally, generality is served in two different ways: First, our algorithm must be able to solve and explore a variety of problems, as long as their DS is of reasonable topological complexity. Apart from the topological perspective, however, the generality of our method also refers to its large geometric flexibility, which is mainly due to the use of a mutable Voronoi diagram [6] as a substrate for generated geometries. Furthermore, the underlying Voronoi diagram can be set to a structured and immutable arrangement, such as a square or hex grid, thus serving some of the commonly requested stylistic constraints of a broad range of games.

2 RELATED WORK

Procedural Content Generation in games is a lively research field [18]. We focus below on approaches that are controllable on a high-level in a way that constrains the low-level topological properties that are handled by the generator. These approaches were a major inspiration for the proposed work.

The distinction between topological and geometric aspects of a level has been recognized and utilized in PCG in various ways. Dormans [5] makes the separation between *missions* (directed graphs that represent gameplay routes) and *spaces* (geometric layouts that align with missions). In that work, Dormans proposes a two-stage generation of levels, first generating missions via graph grammars and then spaces via shape grammars. Beyond shape grammars, more recent work has explored how to generate the low-level geometries through constraint solvers [11] and evolution [17]. Representing the high-level view of the level as a graph is common practice in PCG, indicatively optimizing the graphs via artificial evolution [13], human-in-the-loop interfaces [12] or constraint solvers [34]. In other work, level segments were split based on a grid layout in case of equal-sized rooms as e.g. in *Legends of Zelda* (Nintendo, 1986), and the high-level progression between segments was learned from past examples [36] or optimized based on high-level path-based priorities [17]. Finally, *Sonancia* implemented a very different way of controlling the geometric level generation through the intended progression of tension between rooms from entrance to exit, which could be designer-provided [21] or evolved [22]. Interestingly, similar approaches can be found outside the context of games, in the field of evolutionary architectural design. In those cases, the topological aspect does not represent a mission, but rather the connectivity graph of a house's rooms. In an early example, Charman [3] proposed a constraint-solving system that generates architectural layouts while solving room connectivity constraints. Expanding on Charman's work, among others, Medjdoub and Yannou [24] propose the use of a *Design Specification* which includes a set of Dimensional Constraints (surface, dimensions, orientation of rooms) and a set of Topological Constraints (such as adjacency between rooms). Their layout generation method first solves the topological constraints and then optimizes the solution to best satisfy the geometric constraints. Our proposed methodology has been largely inspired by the concepts introduced there. Other relevant and more recent examples can be found in [23, 25, 29, 39–41]

3 METHODOLOGY

The paper introduces a hierarchical way of representing levels (described in Sec. 3.2), and leverages a constrained quality-diversity (QD) algorithm in order to produce feasible, fit and diverse level geometries. The constrained QD algorithm described in Sec. 3.1 relies on an initial population of random layouts (as described in Sec. 3.3) which are then mutated (as described in Sec. 3.4) to improve the quality and diversity of the population as discussed in Sec. 3.5.

3.1 Algorithm

Our proposed algorithm, *Feasible-Infeasible Multidimensional Archives of Phenotypic Elites* (*FI-MAP-Elites*) is a hybrid of the *MAP-Elites* [26] and *FI-2Pop GA* [15] algorithms. Its main goal is to combine the illumination capabilities of the former with the constraint-solving capabilities of the latter. It is strongly inspired by the *Constrained MAP-Elites* [14] algorithm but operates in a slightly different way which can be beneficial for reasons explained later on. The following paragraphs briefly describe all relevant algorithms and conclude with an in-depth description of FI-MAP-Elites.

MAP-Elites is an illumination algorithm which explores all areas of a behavioral space with a selection pressure towards locally higher fitness [26]. Its operation is based on a discretization of the behavioral space into cells of equal size, each of which can store a single individual whose Behavioral Characterizations (BCs) lie within that cell. In every step (evaluation), the algorithm randomly selects an individual from the archive, mutates it and evaluates the offspring's fitness and BCs. It then places the offspring back in the archive at its corresponding coordinates: if that cell is already occupied the fitter individual of the two survives in that cell. By repeating this operation, the algorithm's archive of solutions is gradually expanded (coverage) and optimized (fitness).

The Feasible-Infeasible two-population genetic algorithm (FI-2Pop GA) [15] handles constrained optimization problems by retaining two populations. The first one contains only feasible individuals and its selection pressure is the objective. The second one contains only infeasible individuals and its selection pressure is a feasibility score, calculated as the percentage of satisfied constraints. Individuals are selected and evolved from both populations, with offspring placed on the corresponding population based on their feasibility. The FI-2Pop GA has been extensively used for level generation [19, 35] as it is inherently a constrained optimization problem.

FI-MAP-Elites discretizes the behavioral space, similar to MAP-Elites, but retains two archives of elites (one for feasible and one for infeasible elites). Cells of each archive store a single individual. Importantly, the survival criterion for the feasible archive is the individual's fitness, while for the infeasible archive it is its feasibility score, similar to FI-2Pop GA. In each iteration, *FI-MAP-Elites* performs the following steps (shown in Fig. 1). First, an individual is selected from a random cell of either the feasible or the infeasible archive, alternating between the two when both archives are non-empty. An offspring of the selected individual is produced via mutation. If the offspring is feasible, it is evaluated (fitness & BCs) and placed in the feasible archive, otherwise it is evaluated (feasibility score & BCs) and placed in the infeasible archive.

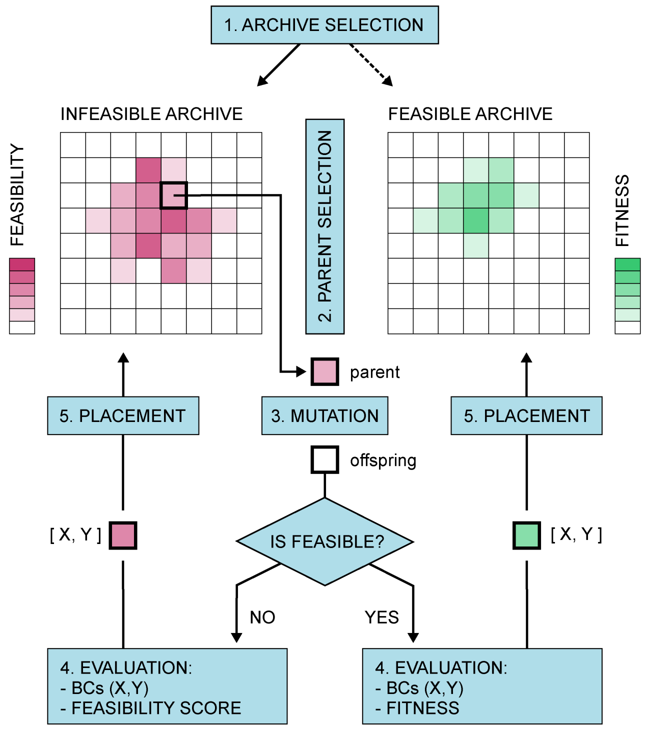
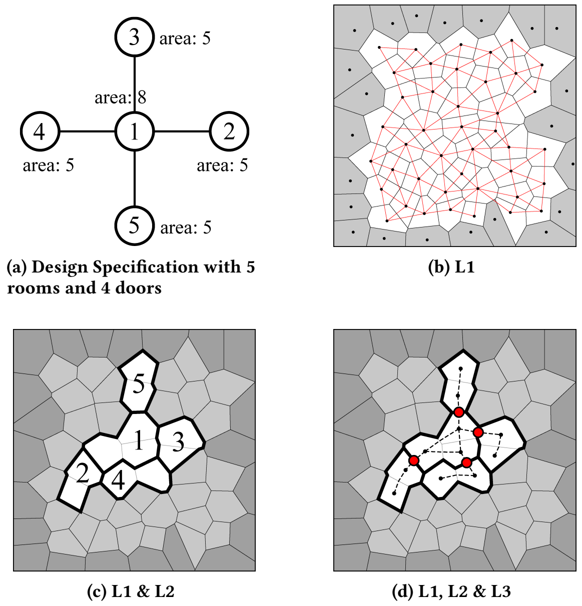


Figure 1: A single iteration of the *FI-MAP-Elites* algorithm.



(a) Design Specification with 5 rooms and 4 doors
(b) L1
(c) L1 & L2
(d) L1, L2 & L3

Figure 2: A Design Specification and the three layers of information (L1, L2, L3) of a feasible Design Implementation.

As mentioned earlier, FI-MAP-Elites is heavily inspired by the *Constrained MAP-Elites* [14] algorithm, which allows two populations of individuals (a feasible and an infeasible one) in every cell of the archive. In its presented configuration, FI-MAP-Elites is almost equivalent to a version of Constrained MAP-Elites where the maximum population size (per population, per cell) was set to 1: 1 feasible and 1 infeasible individual. However, FI-MAP-Elites offers extended functionality as (a) it allows the use of different BCs and/or different feature map resolutions between the feasible and infeasible archives and (b) it can use specialized selection methods [30] to increase the algorithm's overall performance. Nevertheless, these algorithmic options are not examined in this paper, as the main focus lies on the specific design problem's particularities.

3.2 State Representation

The levels generated by our approach form different regions that we identify as "rooms"; however, this could include any partition with hard boundaries and one way of accessing it, such as elevated terrain in *Starcraft* (Blizzard, 1998) or islands with bridges in *Heroes of Might and Magic II* (3DO, 1996). Different rooms are separated by walls, and two adjacent rooms can be connected with a door. Importantly, the generator produces the level geometry (currently as a top-down 2D layout) that adheres to a high-level specification of the level. This dual representation, from the high-level Design Specification to the geometric Design Implementation is described below.

3.2.1 Design Specification. A Design Specification (DS) is an abstract description of the layout that is to be generated. It contains an undirected graph whose vertices represent a set of rooms and whose edges represent a direct connection between a pair of rooms through a door. Each vertex (room) is also assigned with an area that this room should occupy. A DS is a critical part of the design constraints and is utilized in various parts of our algorithmic approach, including the initialization, mutation and evaluation methods. An example of a Design Specification is shown in Fig. 2a.

3.2.2 Design Implementation. A Design Implementation (DI) is a geometry that matches the provided DS, produced by the generator. It includes the specific geometric boundaries of each room, as well as the exact placement of doors that connect them. A generated DI can be feasible or infeasible, based on whether it satisfies the constraints posed by the DS and a few more, described in Sec. 3.5.1.

In our proposed algorithmic approach, a DI consists of three hierarchically dependent layers of information (shown in Fig. 2):

Layer 1 (L1). L1 describes the Voronoi tessellation of a 2D rectangle, whose cells become the building blocks for the shape of rooms and levels (as shown in Fig. 2b). Square and Hex grids are simply special cases of the Voronoi diagram, where points have been prearranged accordingly (see Fig. 4). The L1 genotype consists only of the points' coordinates, while the calculated L1 phenotype includes the generated Voronoi diagram and the resulting adjacency between active cells. Cells that touch the bounding rectangle are treated as inactive and omitted from the phenotype, so as to eliminate geometric biases towards orthogonal boundaries. The L1 genotype can be mutated by changing coordinates of existing points, resulting in a different Voronoi diagram. Adding or removing points has been disabled in this study, for the sake of simplicity.

Layer 2 (L2). L2 describes the rooms' placement on the 2D plane and their exact shape (as shown in Fig. 2c). The L2 genotype is a dictionary that assigns the use of specific rooms of the DS to specific cells of the Voronoi diagram. The L2 phenotype consists of the rooms' boundaries. Those are calculated by merging the regions of cells that belong to the same room.

Layer 3 (L3). L3 describes the walls and doors, as well as the resulting connectivity graph of the interior space (as shown in Fig. 2d). The L3 genotype represents the doors' locations as a pair of neighboring cells. The exact coordinates of each door are part of the L3 phenotype and lie in the mid-point of the line-segment shared between the two cells. The L3 phenotype also includes a computed connectivity graph that accounts for boundaries (walls) between rooms and between a room and the exterior, and doors that allow connections between adjacent rooms.

3.3 Random Initialization

Random initialization is a critical part of the algorithm's operation, as it generates the initial population which will be extended and optimized through mutation operators described in Section 3.4. Our random initialization process consists of a sequence of semi-stochastic operations that attempt to generate a feasible DI based on the DS, as described in the following steps:

Tessellation Definition (L1): First, a set of points that lie within the specified bounding rectangle is assigned, and the resulting tessellation and connectivity graph is calculated. The points' coordinates can be random, (forming a generic Voronoi diagram), or structured to form a square or hex grid as shown in Fig. 4.

Room Placement (L2): The following steps are repeated until all rooms have been placed on the map. First, the algorithm selects a room in the DS that is not yet in the DI, prioritizing rooms with more connections. Then the room is assigned to a single cell that is near already placed rooms that it connects to (according to the DS). If no such cell exists (e.g. for the first room), a free cell will be selected at random. Starting from this first cell, the room's area is expanded using free adjacent cells chosen randomly, until the room's area falls within the specified error margin.

Door Placement (L3): Finally, the algorithm finds all potential locations for door placement per connection in the DS and places a door at one of them, at random. Proper locations include the lines of the shared boundary between connected rooms, whose length is at least 0.5 units. This threshold is adjustable depending on the size of the door meshes and the player's avatar.

3.4 Mutation

Mutation occurs in two stages (see Fig. 3): First, the *destruction* method randomly alters parts of the level without taking constraints into account. Then, the *repair* method attempts to bring the level back to a feasible state. The repair operations are semi-stochastic and do not guarantee feasibility, but increase the chances for it.

3.4.1 Destruction: The destruction method selects between 1 and 3 operations from the following list (chosen at random) and applies them on the selected DI, generating an altered, usually infeasible, offspring. The destruction operators target specific DI layers, noted in parentheses. Note that L1 destruction operations are only applied on Voronoi grids.

- **Points Offset (L1):** Moves all points in a single random direction by a distance $D = r{\cdot}W$, where r is a random number within $[0, 0.25]$ and W is the bounding rectangle's width. Points falling outside the bounding rectangle are symmetrically reinserted. This operation was found to aid constraint satisfaction, as it frees DIs from the grid's boundaries.
- **Points Noise (L1):** Randomly selects a small percentage of points and moves them towards different directions and distances. Points that fall outside the bounding rectangle are symmetrically reinserted.
- **Room Deletion (L2):** Eliminates a single room from the DI, by clearing all cells that belong to it.
- **Unsafe Room Expansion (L2):** Expands a room's area using all of its surrounding cells, whether they be occupied or not.
- **Safe Room Expansion (L2):** Expands a room's area, using all of its unoccupied surrounding cells.
- **Room Erosion (L2):** Reduces a room's area by iteratively removing all cells whose removal does not break its coherence (connectivity between this room's cells) and connectivity (adjacency with connected rooms).
- **Door Deletion (L3):** Removes either 5% or 50% (selected with equal probability) of placed doors. At least one door will be removed, in either case.

3.4.2 Repair: The repair method attempts to bring a DI back to a feasible state by sequentially applying all operations of the following list (denoting the DI layer on which they operate).

- **Missing Rooms Repair (L2):** Places any missing rooms back in the DI, using a single cell that is adjacent to their prescribed neighbors, if possible.
- **Room Coherence Repair (L2):** If any room consists of more than one (disconnected) sub-regions, this method will keep one of them at random and remove the rest.
- **Room Connectivity Repair (L2):** If any room connection in the DS is not implemented in the DI (i.e. no cells of connected rooms are adjacent to each other), this method will attempt to find the shortest path between the rooms and expand them along the path until they become adjacent.
- **Room Area Repair (L2):** If any room's area is very different than that prescribed in the DS, this method will increase or reduce its area by either expanding to adjacent cells, or safely removing cells without breaking its coherence and connectivity, until its area error is below 0.4 (details on the area error E_A and how it is calculated are in Sec. 3.5.1).
- **Doors Repair (L3):** This method deletes any misplaced or redundant doors and then randomly places any missing ones.

3.4.3 Mutation Example: Figure 3 shows an example mutation of a simplified DI. Starting from a feasible parent (Fig. 3 left), the destruction method applies two operations: (D1) a small percentage of points are randomly moved, resulting in an area change for rooms 1 and 5 and a disconnect for room 2, (D2) room 3 is deleted. Starting from the destroyed offspring (Fig. 3 middle), the repair method applies the following operations: (R1) room 3 is placed back

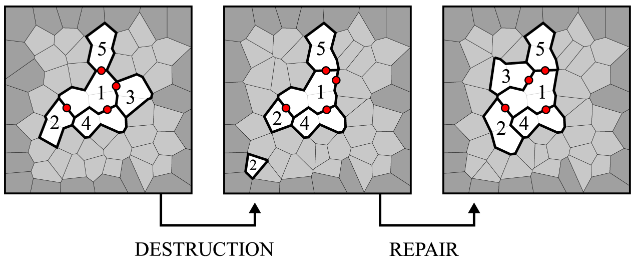


Figure 3: Destruction and repair stages of mutation.

in the level, occupying a single cell adjacent to room 1, (R2) the coherence of room 2 is repaired by deleting the isolated cell, (R3) rooms 2 and 3 are expanded by one cell each, reaching an acceptable area, (R4) the misplaced door that used to connect rooms 1 and 3 is deleted and a new door is added, properly connecting rooms 1 and 3. After all these operations have been applied, the result is a new feasible individual (Fig. 3 right).

3.5 Evaluation

A central part of any evolutionary algorithm is how content is evaluated. In the case of constrained Quality-Diversity search, content evaluation covers a multitude of aspects: the constraints that need to be satisfied (see Sec. 3.5.1) and the feasibility score used to push infeasible individuals towards feasibility (see Sec. 3.5.2), the fitness function for determining quality of feasible individuals (see Sec. 3.5.3), and the Behavioral Characterizations for determining the dimensions of diversity explored by the algorithm (see Sec. 3.5.4).

3.5.1 Constraints: A DI is feasible if all the following constraints are satisfied: (1) all cells in the tessellation are connected, (2) at least 50% of cells are active (not on the borders, as shown in Fig. 4), (3) all prescribed rooms exist in the DI, i.e. at least one cell is assigned to each of them, (4) all rooms are coherent, i.e. all cells assigned to each room must be connected, (5) every connection between rooms specified in the DS must exist as adjacent cells of these rooms in the DI. (6) every room's area-error (E_A) must be smaller than 0.4. For room i its $E_A(i)$ is calculated via Eq. (1), where $A(i)$ is the room's area in the DI and $A_P(i)$ is the room's prescribed area in the DS, (7) doors in the DI precisely satisfy the connectivity described in the DS, and must be on walls spanning at least 0.5 units, (8) a room's internal pathways between walls should be at least 0.5 units wide.

$$E_A(i) = 1 - \frac{min(A(i), A_P(i))}{max(A(i), A_P(i))} \tag{1}$$

3.5.2 Feasibility Score. If any of the constraints of the DI is not satisfied, the DI is infeasible and its feasibility score is calculated. The feasibility score is a way of calculating the proximity of a DI to being feasible, and is used to determine elites in the infeasible population. In FI-2Pop [15], this proximity is calculated as the percentage of satisfied constraints. Given the complexity of the problem at hand, however, and based on preliminary tests, we further provide a gradient for each constraint based on partial satisfaction in order to assist the search for feasible DIs. The following list describes the conversion of the boolean constraints (Sec. 3.5.1) to fine-grained scores that capture the degree of satisfaction for each constraint. The feasibility score is calculated as the average of all those scores. Note that if a constraint is satisfied, its score is 1.

(1) **Connected Graph Score (L1):** we assign $s = 1/N_i$ where N_i is the number of islands (sub-graphs) of the tessellation.
(2) **Active Cells Score (L1):** if active cells are less than 50% of total cells, then we assign $s = r/0.5$, where r is the ratio of active cells over all cells.
(3) **Rooms Existence Score (L2):** calculated as the ratio of existing rooms over the total number of rooms in the DS.
(4) **Rooms Coherence Score (L2):** calculated as the ratio of coherent rooms over the total number of rooms in the DS.
(5) **Connections Existence Score (L2):** calculated as the ratio of satisfied room adjacencies in the DI over the total number of connections in the DS.
(6) **Rooms Area Score (L2):** calculated as the average *area score* of all rooms. For room i, its area score $S_A(i)$ is 0 if the room does not exist, 1 if $E_A(i) > 0.4$ and $(1-E_A(i))/0.6$ otherwise; $E_A(i)$ is described in Eq. (1).
(7) **Doors Score (L3):** calculated as the ratio of current accurate doors over the number of prescribed doors.
(8) **Pathways Score (L3):** calculated as the ratio of current pathways over 0.5 units wide, over the total pathways.

3.5.3 Fitness function: If all constraints of the DI are satisfied, we calculate its feasible fitness which represents the precision of the rooms' areas. It is calculated as the average area precision of all rooms. For room i its area precision is $P_A(i) = 1 - E_A(i)$, where $E_A(i)$ is the room's area error from Eq. (1).

3.5.4 Behavioral characterization: We use two Behavioral Characterizations (BCs) as measures of diversity among the generated DIs. This results in a two-dimensional feature-map for the feasible population (and another feature-map for the infeasible), which is useful for visualizing the archive of maps to a designer. Both BCs are based on the notion of *compactness* (C), a unit-less measure that expresses the relation between a shape's perimeter and its area [16]. The formula for compactness [16] is shown in Eq. (2), but each BC uses this formula with different variables.

$$C = \frac{2\pi A}{\Pi^2} \tag{2}$$

The BCs are as follows:

Plan Compactness. (C_p) expresses the compactness of a level as a whole, accounting its external borders (walls). C_p is calculated via Eq. (2), setting A as the total area of all rooms and Π as the outer perimeter of the layout, disregarding borders between rooms.

Average Room Compactness. (C_r) expresses the average compactness of each room. A room's compactness is calculated via Eq. (2), by setting A as the total area of the room in the DI and Π as the room's perimeter considering borders between rooms, doors, and the outer perimeter. Note that if a room (or the entire layout) is missing from the DI, its compactness is 0; thus we can still calculate these BCs for infeasible individuals.

4 EXPERIMENT PROTOCOL

In order to evaluate our general-purpose generative algorithm, we test it in a broad variety of design specifications including symmetrical specifications that may be useful for multi-player games

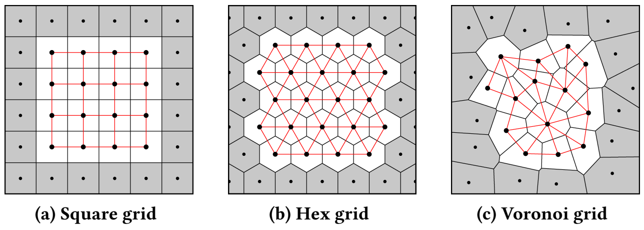


Figure 4: Three indicative spatial tessellations: Square grid, Hex grid and Voronoi grid. In all three cases, cells that touch the border are omitted as inactive (gray), while adjacency between active cells is marked with thin red lines.

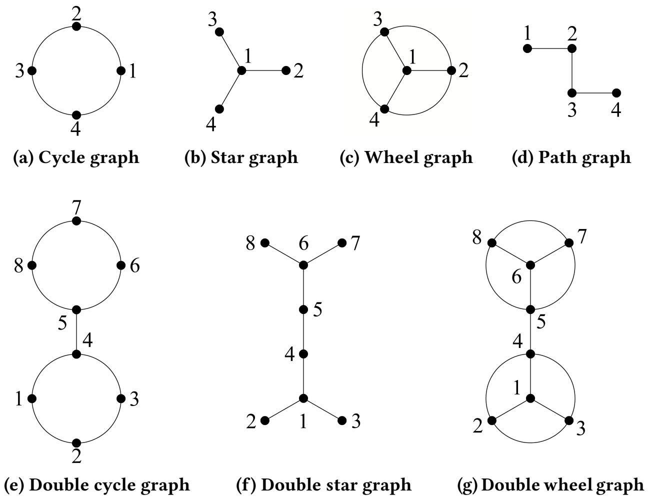


Figure 5: Indicative examples of the examined graph structures, at order of 4 for Fig. 5a-5d and order of 8 for Fig. 5e-5g.

with two players, such as *Starcraft*, or two teams, such as *Counter-Strike* (Valve, 2000). Moreover, we test three different types of grid structures in order to show the versatility of the algorithm but also to assess how the grid structure (and its flexibility) impacts the algorithm's performance.

We test three types of spatial tessellation, placing the points within a boundary rectangle of 16×16 units; cells touching the border are disregarded as inactive (see Sec. 3.2). The *Square grid* is generated by arranging a set of 16×16 (256) points along an orthogonal grid. The *Hex grid* tessellation is generated by arranging 256 points along a triangular grid. For both Square and Hex grids, the tessellation results in 196 active cells (square or hexagon) of approximately 1 square unit area each. The *Voronoi grid* is generated by randomly placing 256 points within a boundary rectangle of 16×16 units. The number of active cells, as well as their exact shape and size, are in this case dynamic and depend on the arrangement of points within the boundary. Contrary to the static Square and Hex grids, the Voronoi grid is mutable (L1 mutation operations) and its structure is co-evolved with the rest of the level.

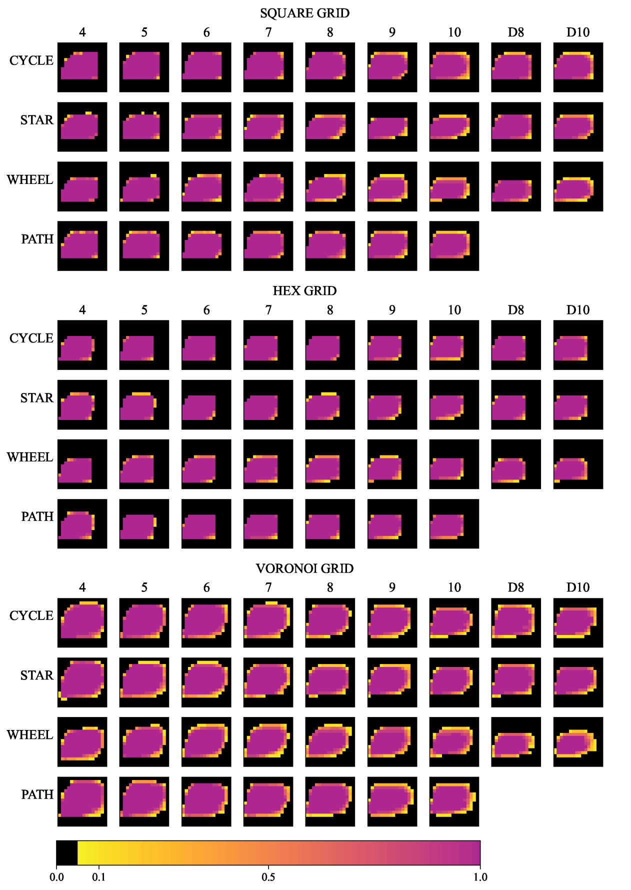


Figure 6: Expressive Range Analysis: Density plots (average coverage across 10 runs), after 2^{19} evaluations, grouped by grid type. The axes in each density plot are C_p (x) and C_r (y).

For each of the three tessellations, we use a set of Design Specifications of varying complexity, based on seven graph typologies (showcased in Fig. 5) at certain ranges of order. The *Cycle*, *Star*, *Wheel* and *Path* graphs are examined in the order-range of 4 to 10, while the *Double Cycle*, *Double Star* and *Double Wheel* graphs are examined in the orders of 8 and 10 vertices, resulting to a total of 34 different graphs. Every DS specifies the desired area per room (A_p) in the same way as $A_p = 4 + D_v$ units, where D_v is the degree of the room's vertex (i.e. the number of edges connecting to it).

Based on the various DS graph structures, we have 34 experiments per grid type (total of 102 experiments). For each experiment, we run 10 independent runs of the FI-MAP-Elites algorithm with an archive-size of 16x16 cells for 2^{19} (524, 288) evaluations and report the aggregated results from these 10 runs.

The source code for all experiments, which can be used to reproduce or expand on the results, can be found at https://github.com/konsfik/FI-MAP-Elites .

5 RESULTS

Figure 6 shows the average coverage of the feasible archive (at a resolution of 16×16 cells) for all 102 experiments across DS and

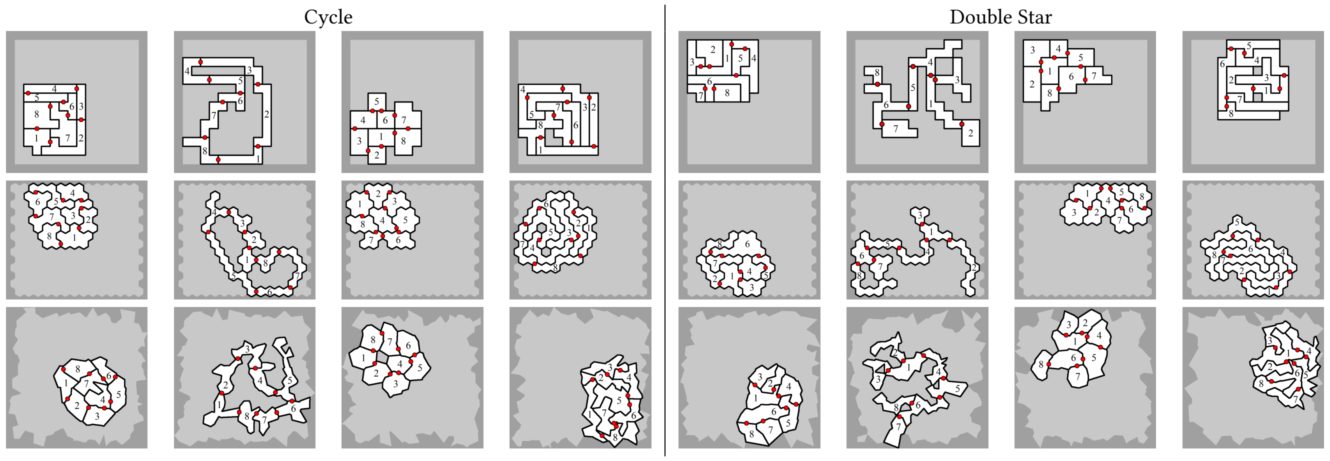


Table 1: Sample levels for two design specifications of order 8, using Square (top), Hex (middle) and Voronoi (bottom) grids. Samples were selected among those with the highest and lowest C_p values and the highest and lowest C_r values, in that order.

grid types. This visualization is also valuable for expressive range analysis [32] since the PCG-QD protocol used here dynamically produces these as part of its standard operation. We measure *coverage* as the ratio of occupied cells in the feasible feature map (as we are only interested in valid level layouts) over the total number of cells (256 in this paper).

We observe that the Square, Hex and Voronoi grids exhibit an average coverage of 38%, 37% and 53% respectively. An interpretation of this large difference between the two static grids and the mutable one is that the Square and Hex grids have most probably reached the limits of their solution space which is naturally smaller than that of the Voronoi grid, due to the geometric constraints that their fixed grids impose on the generated solutions. A quick glance at Table **??** can offer an intuitive understanding of those constraints, as well as the larger flexibility of the mutable Voronoi grid at representing both circular shapes (high compactness) and elongated or branching ones (low compactness).

Another indication that for Square and Hex grids the entire space of possible layouts (in terms of these BCs) has been discovered is that most independent runs cover the same cells in Fig. 6. In comparison, for the Voronoi grid the discovered cells vary more (especially in the borders of the possibility space) from run to run. Observing the progress of coverage over the number of evaluations, we generally notice that for both Hex and Square grids coverage marginally increases after approximately 250, 000 evaluations, while for the Voronoi grid coverage continues to increase up to the end of the evolutionary run (524, 288 evaluations). While coverage keeps increasing for the Voronoi grid, it is worth noting that the initial discovery of feasible individuals is much more difficult for this tessellation than for the other two. We observed that on average, the first feasible individual was discovered after 430, 102 and 4370 evaluations for the Square, Hex and Voronoi grid, respectively. Specifically, feasible individuals were found during the random initialization (i.e. the first 100 evaluations) in 75%, 78% and 12% of runs among experiments for the Square, Hex and Voronoi grid, respectively. Both findings support that constraint satisfaction is much easier for the Square and Hex grids than the Voronoi one, despite the latter's eventual superiority in terms of coverage of the feasible space.

We don't see particular differences between graph types in the DS at the same order, with the Star connectivity faring slightly worse in all three grid types. Interestingly, the Double Star usually reaches higher coverage than the Star at the same order. We anticipate that the algorithm struggles to make a single central room with 9 or 10 adjacent rooms without elongating it and thus lowering room compactness. Fig. 6 corroborates this, as at high DS orders (9 and 10) in all graph types both lower and upper boundary for C_r (the y axis) are clipped without a similar sacrifice for C_p (the x axis).

To showcase the type of levels generated with the different grid types and BCs, we show some example layouts generated for the Cycle and the Double Star at order 8 in Table **??**. The four levels shown are selected from the edges of the possibility space (with highest or lowest BC values) in order to showcase the diversity of the possible DIs for the same DS. We see that for high C_p all layouts are compact and have no holes or protruding walls; especially for the Voronoi grid the layouts are almost circular. The opposite is true for low C_p, since most rooms are corridors of minimal width and many holes, dead-ends, and protruding edges. Similar corridor-like rooms prevail for low C_r but the layout has far fewer holes and a smaller footprint. Finally, for high C_r we observe compact rooms but as a footprint there are more jutting edges and a less circular layout in the case of Voronoi grids. Levels with high C_r values seem the most intuitive as e.g. dungeon levels (for the Cycle graph) or arena-style deathmatch FPS levels [2] (for the Double Star graph), which have two larger, central, arenas at rooms 1 and 6.

6 DISCUSSION

As indicated in Section 5, our proposed methodology can support all tested specifications, across all types of tessellations, generating diverse sets of feasible solutions that approach the natural limits of the solution space. This is a clear indication that our proposed algorithm exhibits a high degree of expressivity and thus satisfies one of the main goals of this approach. Furthermore, the relation between the problem's complexity and the algorithm's behavior, in terms of computational performance, constraint-satisfaction and diversification follows an expected pattern, where more complex specifications tend to be more difficult (and slow) to solve and diversify. However, as the results showcase, the complexity of the tested specifications has not approached the practical limits of addressable problems, suggesting that the algorithm can be applied to even more complex ones.

This study explored diversity only on simple compactness metrics. While our framework includes more BCs (e.g. based on walls' orthogonality and distances between rooms), finding and quantifying BCs that are meaningful to designers is not an easy task. A possible way forward would be utilizing metrics of spatial evaluation from the field of visibility graph analysis [16]. For example, *Through Vision* [16, 38] can evaluate spatial regions in regard to movement, while *Visual Control* [10, 16, 37] and *Visual Controllability* [16, 37] can evaluate spatial regions in regard to *Control* [10]. Based on such a theory-driven approach, one may find more meaningful modes of diversification for specific types of games, in regards to elements of the levels' usability and player-experience. Such BCs would also be highly relevant to e.g. first-person shooters (FPS) where lines of sight and lines of fire are highly relevant.

Another aspect that can be improved is the specialization of our methodology for the FPS and dungeon crawler genres, or even its expansion to other types of games, like Real Time Strategy (RTS) or even platformers. For example, in the context of FPS games, our proposed methodology can be improved by including more types of openings, such as windows and exterior doors, treating walls as obstacles of a variant height, or even extending along the z-axis with multiple floors and stairs that connect them [2]. In the context of dungeons, the levels can be be extended to include interior elements, such as furniture and other items. Importantly, this extension can be implemented as a discrete, automated step, after the initial geometric diversification of the levels' structure. An inspiring example of this type is *Dungeon Alchemist* [9], a level design tool that generates impressively detailed and feature-rich rooms given their user-specified boundaries. Representing maps for RTS games is also an attainable goal. By excluding the L3 part of the DI, and adding details including bases and resources, each "room" of the DI can be treated as a region of strategic importance in the map. Alternatively, by treating the DI as if it was a side-view instead of a top-down view, and applying minor modifications in the constraints and BCs, our methodology can also generate certain types of platformer levels. In this case the constraints must probably be adapted to take into account gravity-related issues [33].

7 CONCLUSION

This paper introduces a way of representing room-based layouts in a free-form, flexible way that can lend itself well to many types of games set in square, hex, or ad-hoc spatial layouts. Through the use of a constrained MAP-Elites algorithm, a designer can control the generative output through a high-level, visual design specification while also being able to see the full breadth of room configurations laid out in a two-dimensional arrangement based on chosen dimensions to explore. Results indicate that the algorithm can handle a broad variety of design specifications and can create diverse, feasible levels in different grid formats. Future work aims to improve the algorithm's controllability by allowing the designer to control the output during evolution and to extend its applicability to more game genres.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union H2020 Horizon Programme (2014-2020) under grant agreement 952002, project PrismArch: Virtual reality aided design blending cross-disciplinary aspects of architecture in a multi-simulation environment.

REFERENCES

[1] Alberto Alvarez, Steve Dahlskog, Jose Font, and Julian Togelius. 2019. Empowering quality diversity in dungeon design with interactive constrained MAP-Elites. In *Proc. of the IEEE Conf. on Games*.

[2] William Cachia, Antonios Liapis, and Georgios N. Yannakakis. 2015. Multi-level evolution of shooter levels. In *Proc. of the AIIDE Conf.*

[3] Philippe Charman. 1994. A constraint-based approach for the generation of floor plans. In *Proc. of the IEEE Conf. on Tools with Artificial Intelligence*. 555–561.

[4] Kate Compton, Adam Smith, and Michael Mateas. 2012. Anza Island: Novel gameplay using ASP. In *Proc. of the FDG workshop on Procedural Content Generation in Games*.

[5] Joris Dormans. 2010. Adventures in level design: Generating missions and spaces for action adventure games. In *Proc. of the FDG workshop on Procedural Content Generation in Games*.

[6] Steven Fortune. 1995. Voronoi diagrams and Delaunay triangulations. *Computing in Euclidean geometry* (1995), 225–265.

[7] Daniele Gravina, Ahmed Khalifa, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. 2019. Procedural content generation through Quality-Diversity. In *Proc. of the IEEE Conf. on Games*.

[8] Daniele Gravina, Antonios Liapis, and Georgios N.. Yannakakis. 2016. Constrained surprise search for content generation. In *Proc. of the IEEE Conf. on Computational Intelligence and Games*.

[9] Wim De Hert and Karel Crombecq. 2022. Dungeon Alchemist: AI-powered dungeon generation. https://www.dungeonalchemist.com/. [Online; accessed 24 May 2022].

[10] Bill Hillier and Julienne Hanson. 1984. *The social logic of space*. Cambridge university press.

[11] Ian Horswill and Leif Foged. 2012. Fast procedural level population with playability constraints. In *Proc. of the AIIDE Conf.*

[12] Daniel Karavolos, Anders J. Bouwer, and Rafael Bidarra. 2015. Mixed-initiative design of game levels: Integrating mission and space into level generation. In *Proc. of the Foundations of Digital Games Conf.*

[13] Daniel Karavolos, Antonios Liapis, and Georgios N. Yannakakis. 2016. Evolving missions to create game spaces. In *Proc. of the IEEE Conf. on Computational Intelligence and Games*.

[14] Ahmed Khalifa, Scott Lee, Andy Nealen, and Julian Togelius. 2018. Talakat: Bullet hell generation through constrained map-elites. In *Proc. of the Genetic and Evolutionary Computation Conf.* 1047–1054.

[15] 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.

[16] Petros Koutsolampros, Kerstin Sailer, Tasos Varoudis, and Rosie Haslem. 2019. Dissecting visibility graph analysis: The metrics and their role in understanding workplace human behaviour. In *Proc. of the Intl. Space Syntax Symposium*.

[17] Antonios Liapis. 2017. Multi-segment evolution of dungeon game levels. In *Proc.of the Genetic and Evolutionary Computation Conf.*

[18] Antonios Liapis. 2020. 10 years of the PCG workshop: Past and future trends. In *Proc. of the FDG workshop on Procedural Content Generation in Games*.

[19] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. 2013. Towards a Generic Method of Evaluating Game Levels. In *Proc. of the AIIDE Conf.*

[20] 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.

[21] Phil Lopes, Antonios Liapis, and Georgios N. Yannakakis. 2015. Targeting horror via level and soundscape generation. In *Proc. of the AIIDE Conf.*

[22] Phil Lopes, Antonios Liapis, and Georgios N. Yannakakis. 2016. Framing tension for game generation. In *Proc. of the Intl. Conf. on Computational Creativity*.

[23] Chongyang Ma, Nicholas Vining, Sylvain Lefebvre, and Alla Sheffer. 2014. Game Level Layout from Design Specification. *Computer Graphics Forum* 33 (05 2014), 95–104.

[24] Benachir Medjdoub and Bernard Yannou. 2000. Separating topology and geometry in space planning. *Computer-aided design* 32, 1 (2000), 39–61.

[25] Paul Merrell, Eric Schkufza, and Vladlen Koltun. 2010. Computer-Generated Residential Building Layouts. *ACM Trans. Graph.* 29, 6, Article 181 (dec 2010), 12 pages.

[26] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909* (2015).

[27] Kyungjin Park, Bradford Mott, Wookhee Min, Eric Wiebe, Kristy Elizabeth Boyer, and James Lester. 2020. Generating game levels to develop computer science competencies in game-based learning environments. In *Proc. of the Intl. Conf. on Artificial Intelligence in Education*. 240–245.

[28] Graeme Ritchie. 2007. Some empirical criteria for attributing creativity to a computer program. *Minds and Machines* 17 (2007), 76–99.

[29] Eugénio Rodrigues, Adélio Rodrigues Gaspar, and Álvaro Gomes. 2013. An evolutionary strategy enhanced with a local search technique for the space allocation problem in architecture, Part 1: Methodology. *Computer-Aided Design* 45, 5 (2013), 887–897.

[30] Konstantinos Sfikas, Antonios Liapis, and Georgios N. Yannakakis. 2021. Monte Carlo Elites: Quality-diversity selection as a multi-armed bandit problem. In *Proc.of the Genetic and Evolutionary Computation Conf.*

[31] Anthony J Smith and Joanna J Bryson. 2014. A logical approach to building dungeons: Answer set programming for hierarchical procedural content generation in roguelike games. In *Proceedings of the 50th Anniversary Convention of the AISB*.

[32] Gillian Smith and Jim Whitehead. 2010. Analyzing the expressive range of a level generator. In *Proc. of the FDG workshop on Procedural Content Generation in Games*.

[33] Gillian Smith, Jim Whitehead, Michael Mateas, Mike Treanor, Jameka March, and Mee Cha. 2011. Launchpad: A Rhythm-Based Level Generator for 2-D Platformers. *IEEE Trans. on Computational Intelligence and AI in Games* 3, 1 (2011).

[34] Thomas Smith, Julian Padget, and Andrew Vidler. 2018. Graph-based generation of action-adventure dungeon levels using Answer Set Programming. In *Proc. of the FDG workshop on Procedural Content Generation in Games*.

[35] Nathan Sorenson, Philippe Pasquier, and Steve R. DiPaola. 2011. A generic approach to challenge modeling for the procedural creation of video game levels. *IEEE Trans. on Computational Intelligence and AI in Games* 3, 3 (2011), 229–244.

[36] Adam James Summerville, Morteza Behrooz, Michael Mateas, and Arnav Jhala. 2015. The learning of Zelda: Data-driven learning of level topology. In *Proc. of the Foundations of Digital Games Conf.*

[37] Alasdair Turner. 2001. Depthmap: a program to perform visibility graph analysis. In *Proc. of the Intl. Space Syntax Symposium*.

[38] Alasdair Turner. 2007. To move through space: Lines of vision and movement. In *Proc. of the Intl. Space Syntax Symposium*.

[39] Tim Tutenel, Ruben M. Smelik, Ricardo Lopes, Klaas Jan de Kraker, and Rafael Bidarra. 2011. Generating consistent buildings: a semantic approach for integrating procedural techniques. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 274–288.

[40] Roland van der Linden, Ricardo Lopes, and Rafael Bidarra. 2013. Designing procedurally generated levels. In *Proceedings of IDPv2 - Workshop on Artificial Intelligence in the Game Design Process, co-located with AIIDE 2013*.

[41] Samuel SY Wong and Keith CC Chan. 2009. EvoArch: An evolutionary algorithm for architectural layout design. *Computer-Aided Design* 41, 9 (2009), 649–667.