# Evolving Missions to Create Game Spaces

Daniel Karavolos
Institute of Digital Games
University of Malta
e-mail: daniel.karavolos@um.edu.mt

Antonios Liapis
Institute of Digital Games
University of Malta
e-mail: antonios.liapis@um.edu.mt

Georgios N. Yannakakis
Institute of Digital Games
University of Malta
e-mail: georgios.yannakakis@um.edu.mt

***Abstract*—This paper describes a search-based generative method which creates game levels by evolving the intended sequence of player actions rather than their spatial layout. The proposed approach evolves graphs where nodes representing player actions are linked to form one or more ways in which a mission can be completed. Initially simple graphs containing the mission's starting and ending nodes are evolved via mutation operators which expand and prune the graph topology. Evolution is guided by several objective functions which capture game design patterns such as exploration or balance; experiments in this paper explore how these objective functions and their combinations affect the quality and diversity of the evolved mission graphs.**

## I. Introduction

Procedural content generation (PCG) in games has received considerable academic interest in the last decade, exploring different ways to represent, generate and evaluate game content such as rulesets, card decks, puzzles, weapons, terrain, etc. Among the most prominent generative techniques being explored are search-based techniques [1] which often use artificial evolution to explore a vast search space guided by an objective function, constraint-based techniques [2] which carefully define the space of viable solutions, and generative grammars [3] which define the creation and expansion rules of an artifact and can gradually increase its level of detail.

The vast majority of PCG research focuses on game level generation, following the trends of the game industry where PCG primarily creates game spaces such as the dungeons of *Diablo* (Blizzard 1996), the gameworlds of *Civilization V* (Firaxis 2010) or the mansions of *Daylight* (Zombie Studios 2014). While commercial games primarily use constructive generative techniques [1], academic interest in PCG has moved beyond this narrow focus and has tested a broad variety of techniques, representations, and types of game levels which can be generated. Most often, such generators create the level's layout and then evaluate its spatial characteristics such as its navigable regions [4] or functional characteristics derived from e.g. playtraces of artificial agents running through it [5]. In the case of [6], the generator creates a tile-based layout of a dungeon for a role-playing game adventure module, which is then used to derive a room connectivity graph for placing encounters to follow the progression of a player from the dungeon's entrance. However, an inverse generative process is also possible, where the structure of the player experience (with all its possible variations and branches) is generated first and is used to derive the spatial structure of the game level.

This paper presents a search-based approach for generating levels through an indirect representation, evaluating and evolving the player's sequence of possible actions rather than the explicit sequence of rooms they have to visit. While the level geometry and the action sequence are linked (i.e. the latter constrains the former), the action sequence is a more concise representation as it does not contain trivial information such as empty rooms or walls. Moreover, the action sequences are represented as a graph of nodes while game levels tend to be represented as some form of bit array [7]; this allows the design of genetic operators (for adding, removing, or connecting nodes) which have a better locality and result in non-trivial yet non-destructive changes to the phenotype. Finally, parsing the graph directly allows for fast and simple evaluations of the decision density of a player traversing a level from start to finish. The paper focuses on the generation of mission graphs for the dungeon crawl game *Dwarf Quest* (Wild Card Games 2013), with nodes representing the start and end of the mission, puzzles, rewards and combat sections. Results show that many different types of mission graphs can be generated, from simple, short playthroughs to complex structures with multiple paths to the goal. The *Dwarf Quest* levels created from these mission graphs similarly range from straightforward and short to maze-like and grueling.

## II. Related Work

Procedural content generation has been used in the game industry, and primarily for the generation of game levels, since the 1980s with games such as *Rogue* (Toy and Wichman 1980) and *Elite* (Acornsoft 1984). Level generation has only increased in scale and commercial appeal in recent years with games such as *Minecraft* (Mojang 2011) and *No Man's Sky* (Hello Games 2016) embracing it as a major selling point. Academic interest in level generation is similarly extensive, with levels for first person shooters [4], puzzle games [2], side-scrolling platformers [8], strategy games [9] and many other game genres being generated using a diverse set of techniques.

Particularly relevant to the current work are search-based and grammar-based techniques for generating levels. The family of search-based PCG [1] methods attempt to gradually improve a level by applying local changes; most often, artificial evolution is used and the local changes take the form of mutation of tiles in a grid-based map or recombination of the layouts of two parents to create offspring that combine the features of both parents. In search-based PCG, it is common

to select the most promising parents to create the next batch of results (generation) based on a quantifiable objective function which evaluates how appropriate a game level is: examples include the length of its paths [9], the combat duration between artificial agents [4] or the distribution of its treasures [5].

As their name suggests, grammar-based techniques take advantage of generative grammars, which represent a set of rewrite rules which transform expressions. Although originally designed to analyze and classify language phrases, grammars can be used to transform any expression. For level generation, grammars have been described and used extensively by Dormans [10] while the most well-known commercial application of grammar-based level generation is *Spelunky* (Mossmouth 2008) [11]. At its core, a generative grammar is a set of rules which can be iteratively applied to increase the complexity of an expression (e.g. a game level). Such rules can frame the problem (e.g. dungeon $\rightarrow$ obstacle + treasure) or can be recursive (e.g. obstacle $\rightarrow$ monster + obstacle). If multiple rules can be applied to the expression, one is chosen randomly.

The dual representation for game levels (as a mission and as a space) was first introduced in [3] and expanded in [10], where the mission graph was created via a graph grammar while the architecture was built from shape grammars which rewrite mission nodes into rooms of various sizes. The paradigm was applied to the game *Dwarf Quest* in [12], where both mission graph and layout was created through grammars: the layout solver places rooms on a 2D grid based on the mission graph, obeying requirements on planarity and orthogonality and applying pre-processing steps as needed to repair non-conforming missions. In [13], the generation of missions and spaces in *Dwarf Quest* was enhanced through a human-computer interface that allowed a human designer to interject in (or replace) the generative grammars with her own intuitions. The tool allowed the designer to create missions in varying levels of detail, e.g. authoring a rough sketch of a mission and allowing the generative grammars to expand on that sketch automatically (or with some human curation).

## III. Methodology

This paper uses search-based techniques to evolve a mission graph representing the player's possible action sequences, which is then used to create a level architecture for *Dwarf Quest* (Wild Card Games 2013). The representation of the mission graph and the types of nodes it can contain is described in III-A, the details of the evolutionary approach and its mutation operators in III-B, the objectives which drive evolution in III-C, and finally the methods for converting the evolved mission graphs into game levels in III-D.

### A. Mission Representation

The evolved artifacts consist of mission graphs represented as a list of nodes and edges. The nodes represent abstract player actions, such as solving a puzzle. This abstract action will later be transformed into a specific action by a grammar, which is then transformed by a layout solver into one or more rooms where gameplay will take place. A more detailed description of this process is provided in [13]. There are 14 types of nodes described in Table I, split into four categories: *fight*, *puzzle*, *reward*, and *neutral*. Fight nodes involve active opposition from monsters, puzzle nodes involve passive opposition (e.g. locked doors), while reward nodes have no opposition but provide power-ups for future fights[1]. Neutral nodes are the *start* node, where the player is initially placed, and the *end* node where the player completes the level; the goal of the mission is to traverse the graph starting from the start node and reaching the end node. For evolution, each node is stored as an integer acting as the identifier of its node type.

Fig. 1. In-game screenshot of *Dwarf Quest* (Wild Card Games 2013), showing a fight node with doorways to two adjacent rooms.

TABLE I
List of possible node types split into categories.

| Neutral | Reward | Fight | Puzzle |
|---|---|---|---|
| Start | RandomItem | Enemy | DoorPuzzle |
| End | HealthPotion | Boss | BridgePuzzle |
| | BattleCard | | ChestPuzzle |
| | Treasure | | FloorTrap |
| | Altar | | |

Edges connect two nodes, and are represented by three parameters: the index of the starting node, the index of the ending node, and a flag on whether the edge is directed. For example, `edge(0,1,false)` represents a bidirectional edge between element 0 and element 1 in the node list. Since the corridors in Dwarf Quest are bidirectional the current work ignores the third parameter, but this representation supports other game modes involving e.g. one-way portals.

### B. Mission Evolution

The generative approach followed in this paper evolves an initial population of individuals in order to maximize a fitness function consisting of one or more objectives (covered in III-C). The initial population consists of identical individuals representing the simplest possible mission: a start node, an end node and an edge between them. The following generations increase the topology of these initial individuals, and after the first generation the selection process favors individuals with a higher fitness. The algorithm uses an elitism of 10%,

[1]Among the reward nodes, battle cards act as one-time powerups while altars function like a shop in which potions and battle cards may be purchased.

TABLE II
LIST OF MUTATION OPERATORS.

| Name | Description |
|---|---|
| Insert Node | A randomly chosen edge is split and a random node is inserted between the edge's start and end nodes, connecting the inserted node via two edges to the initial start and end nodes. This creates longer action sequences. |
| Add Node | As the insert node operator except the chosen edge is not deleted, providing multiple paths between its start and end nodes (directly or indirectly via the new node). |
| Change Node | A randomly chosen non-neutral node changes into a random other non-neutral node type. |
| Delete Node | A randomly chosen non-neutral node is deleted with the following constraints: if the node has one edge, both the node and its edge is deleted; if the node has two edges, an edge is added linking the nodes connected to the deleted node; nodes with 3 or more edges are not deleted as it would be too destructive. |
| Add Edge | Two randomly chosen nodes are connected with a bidirectional edge. This can create duplicate edges, except when the individual only contains a start and an end node (in which case this mutation can not be applied). |
| Delete Edge | A randomly chosen edge is deleted, unless it is a node's last edge. |

making copies of the fittest parents in the next generation; the remaining individuals in the next generation are mutations of parents chosen via fitness-proportionate roulette wheel selection. The same parent can be selected multiple times, thus generating multiple mutated offspring. Evolution is carried out via mutation alone, and each offspring is a copy of its parent to which multiple mutation operators can be applied based on a probability. Several mutation operators are designed in order to change the topology of the mission graph while obeying constraints to avoid undesirable results. The mutation operators are summarized in Table II. Mutation operators are not allowed to place more than one boss node and more than one altar node per level; other node types are chosen in those cases. The mutation probabilities are based on preliminary testing and favor adding nodes and edges over deleting them, as the latter is more disruptive in most fitness landscapes.

### C. Mission Objectives

There are several desirable patterns that evolved mission graphs should exhibit. Inspired in part by the general design patterns of [14] and their mathematical formulations in [15], five fitness dimensions are designed to drive evolution (alone or combined into a weighted sum). Steps have been taken to convert all the metrics into a [0,1] value range, with high scores representing more desirable content. Designer intuition was applied to specify the desirable value ranges of several of these metrics (e.g. a desired shortest path of 5 to 10 nodes).

- *Shortest Path*. The number of nodes along the shortest path between start and end nodes ($d_{s,e}$) is normalized by eq. (1) to give optimal scores to paths with 5 to 10 nodes.

$$f_p = \min\left\{(1+e^{3-d_{s,e}})^{-1}, 1-(1+e^{13-d_{s,e}})^{-1}\right\} \quad (1)$$

- *Exploration*. Inspired by [15], this function uses flood fill from the start node to evaluate how much the player will need to explore the level before reaching the end node. Eq. (2) normalizes this metric to give optimal scores to exploration covering three times as many nodes as the shortest path.

$$f_e = 1 - \tfrac{1}{3}|F_{s,e} - d_{s,e}| \quad (2)$$

where $F_{s,e}$ is the number of nodes covered by a flood fill algorithm starting from the start node and stopping once the end node is covered.

- *Variation*. The percentage of edges that connect nodes of different categories, excluding start and end nodes.

$$f_v = \frac{E_d}{E} \quad (3)$$

where $E_d$ is the number of edges connecting non-neutral nodes of different categories (e.g. a fight node and a reward node), and $E$ is the total number of edges connecting non-neutral nodes.

- *Dispersed rewards*. Based on [15], eq. (4) evaluates the number of nodes considered safe to rewards (i.e. nodes which are much closer to one reward node versus all other reward nodes).

$$f_a = \frac{1}{N}\sum_{i=1}^{R} A_i \quad (4)$$

where $N$ and $R$ is the number of nodes and reward nodes in the mission, respectively, and $A_i$ the number of nodes with a safety score for reward $i$ above a threshold of 0.35. Details of how safety is calculated can be found in [15].

- *Balanced rewards*. Based on [15], eq. (5) evaluates whether every reward has an equal number of safe nodes around it as every other reward.

$$f_b = 1 - \frac{1}{R(R-1)}\sum_{i=1}^{R}\sum_{\substack{j=1\\ j\neq i}}^{R}\frac{|A_i - A_j|}{max\{A_i, A_j\}} \quad (5)$$

### D. From Mission Graphs to Levels

In order to create the game's final levels, evolved mission graphs are interpreted by the layout solver described in [12], which is in turn constrained by the map options of the *Dwarf Quest* game. Due to these constraints, three post-processing steps must be applied on the evolved mission graphs:

1) The room with the player's spawn point (start node) has only one corridor. If the start node has more than one edge, we create an empty node linked to the start node and move the start node's edges to the empty one.
2) If there are three nodes that are all pair-wise connected, the layout solver cannot decide which of the rooms to place first. To solve this, we insert an empty node between one of the edges.
3) *Dwarf Quest* rooms must have at least two corridors: non-neutral nodes with only one edge are omitted.

Furthermore, the layout solver considers the edges between nodes as directional edges, even though they are not implemented as such in *Dwarf Quest*, and uses them to determine the relative positions of the rooms. To achieve that, a flooding

TABLE III
MEAN FITNESS SCORES (AND THEIR STANDARD DEVIATION) OF THE FITTEST INDIVIDUAL AFTER 100 GENERATIONS.

| Fitness | $f_p$ | $f_e$ | $f_v$ | $f_s$ | $f_b$ |
|---|---|---|---|---|---|
| Single Objective | 0.99 (0.00) | 0.67 (0.21) | 1.00 (0.00) | 0.68 (0.18) | 0.99 (0.02) |
| $f_p$+$f_e$ | 0.90 (0.05) | 0.84 (0.10) | – | – | – |
| $f_e$+$f_v$ | – | 0.70 (0.25) | 0.99 (0.03) | – | – |
| $f_v$+$f_a$ | – | – | 1.00 (0.00) | 0.67 (0.08) | – |
| $f_p$+$f_e$+$f_v$ | 0.87 (0.09) | 0.64 (0.12) | 0.98 (0.03) | – | – |
| $f_p$+$f_e$+$f_a$ | 0.91 (0.07) | 0.83 (0.12) | – | 0.73 (0.03) | – |
| All | 0.89 (0.08) | 0.67 (0.17) | 0.95 (0.04) | 0.67 (0.03) | 0.71 (0.02) |

algorithm turn the bidirectional edges of the mission graph into directed ones, based on each node's distance to the start. If the result has nodes with only incoming or outgoing edges, an edge is chosen (based on the distance of its linked node to the end node) and its direction is flipped.

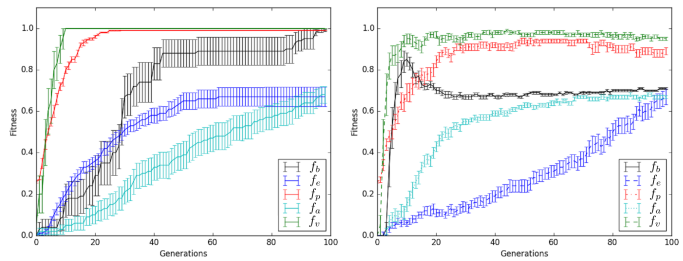
## IV. RESULTS

The experiments in this paper assess the performance of evolution on mission graphs optimizing each objective individually, optimizing all objectives simultaneously, and a few sample combinations of objectives. Each experiment included 20 independent runs of 100 generations, on a population of 100 individuals. The reported values are averaged from these 20 runs, and the standard deviation is displayed in parentheses or error bars (in tables and figures respectively). Statistical significance tests are performed via two-tailed Student's $t$-tests (assuming unequal variance) with a significance threshold of 5%. Since post-processing only contributes to the interpretation of the mission and not to the mission itself, the results below are based on the graphs before post-processing.

### A. Optimization Performance

Table III displays the average scores in several fitness dimensions of the fittest evolved individuals after 100 generations. Results are derived from optimization runs targeting a single objective (in the single objective row), all objectives and a sample of the possible combinations of objectives. In case of multiple objectives, the overall fittest individuals are considered (according to the summed fitness dimensions' scores). Observing Table III, it is surprising that missions evolved towards $f_e$ and $f_a$ individually have a high deviation and low scores while they often reach higher scores when combined with other objectives (significantly higher for $f_p$+$f_e$ and $f_p + f_e + f_a$). Other objectives exhibit a less surprising behavior, reaching high scores when evolution targets them individually. Among the objectives, $f_v$ manages to achieve near-optimal values in all runs and in all combinations of objectives. This may point to the fact that this objective tends to dominate others during multi-objective evolution, although it is equally likely that its fitness score formulation in eq. (3) can reward optimal values to a broad range of mission graphs.

It should be noted that the efficiency of the GA was tested against a *baseline* which ran 20 evolutionary runs with the same parameters, but rewarding all individuals with a constant fitness score (i.e. random selection). The final maximum scores




(a) Single objective evolution for each of the objective functions.

(b) Evolution towards a linear combination of all objectives.

Fig. 2. Progression of the best individuals' fitness scores during evolution, for the fitness functions in Table III and all fitness functions combined. The error bars show the standard error.

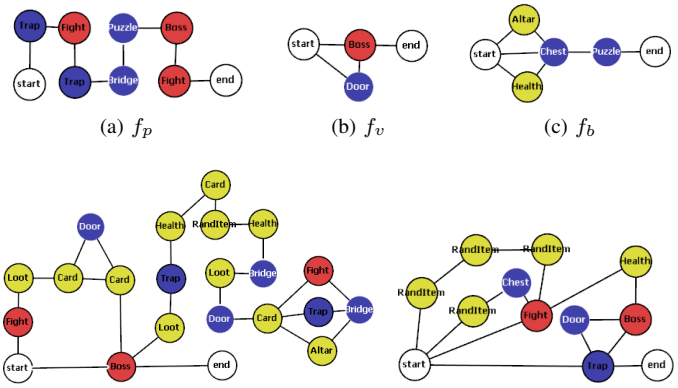
of individual fitnesses in the baseline was significantly lower than the respective single-objective optimization runs; while $f_v$ was relatively close, fitness scores in $f_a$ and $f_e$ were 18 times and 6 times those of the baseline respectively. Comparing the best individuals for all objectives (summed) between the baseline and the optimization run targeting it, similar differences were found, with optimization runs creating individuals with 2.8 times the fitness scores of the baseline.

Figure 2a shows the optimization behavior of each fitness dimension when used as a single objective. It is obvious that $f_p$ and $f_v$ are quick to optimize, reaching optimal scores in the first 10 to 20 generations; by comparison, $f_b$ reaches optimal scores much more slowly, with a high standard deviation in most generations (shown as error bars) indicating an unpredictable optimization behavior. On the other hand, $f_e$ and $f_a$ reach lower scores (as evidenced by Table III) and improve much slower than the other objectives: $f_a$ in particular seems to be the slowest to reach even sub-optimal scores.

Figure 2b shows how the scores in individual fitness dimensions fluctuate in the overall fittest individual when evolution targets the sum of all five objectives. Comparing Fig. 2b with Fig. 2a, the differences are surprising. While $f_v$ and $f_p$ unsurprisingly reach optimal scores quickly and remain high throughout evolution, $f_b$ also increases quickly (reaching far higher scores than when evolving individually) and then drops, stabilizing at lower final scores than in Fig. 2a. The optimization behavior of $f_s$ and $f_e$ is similarly affected: while they reach similar final scores as in Fig. 2a, $f_a$ optimizes faster when combined with other objectives and $f_e$ optimizes far slower. This is likely due to the way that $f_e$ is computed:

TABLE IV
MEAN AND STANDARD DEVIATION OF THE DUNGEON METRICS FOR A SAMPLE OF THE FITNESS FUNCTIONS.

| Fitness | Graph Size | Shortest Path | Branching Factor | Fights Ratio | Puzzles Ratio | Rewards Ratio |
|---|---|---|---|---|---|---|
| $f_p$ | 9.25 (0.43) | 9.00 (0.00) | 1.85 (0.12) | 0.14 (0.13) | 0.39 (0.18) | 0.47 (0.19) |
| $f_e$ | 8.60 (2.67) | 2.75 (0.43) | 2.75 (0.21) | 0.20 (0.17) | 0.45 (0.19) | 0.35 (0.24) |
| $f_v$ | 4.10 (0.30) | 2.90 (0.70) | 1.99 (0.30) | 0.23 (0.25) | 0.38 (0.20) | 0.39 (0.20) |
| $f_a$ | 12.15 (4.40) | 6.55 (3.47) | 2.24 (0.40) | 0.05 (0.07) | 0.21 (0.10) | 0.69 (0.21) |
| $f_b$ | 5.90 (1.81) | 3.10 (0.94) | 2.38 (0.36) | 0.09 (0.13) | 0.32 (0.18) | 0.59 (0.13) |
| $f_p$+$f_e$ | 22.35 (1.88) | 6.36 (0.65) | 2.68 (0.15) | 0.14 (0.06) | 0.50 (0.09) | 0.36 (0.12) |
| $f_v$+$f_e$ | 8.75 (3.86) | 2.55 (0.50) | 2.87 (0.23) | 0.21 (0.15) | 0.37 (0.13) | 0.42 (0.19) |
| $f_v$+$f_a$ | 12.76 (3.11) | 6.03 (3.05) | 2.35 (0.16) | 0.19 (0.08) | 0.35 (0.08) | 0.45 (0.10) |
| $f_p$+$f_e$+$f_v$ | 17.85 (2.35) | 6.15 (0.85) | 2.57 (0.13) | 0.21 (0.05) | 0.40 (0.04) | 0.39 (0.06) |
| $f_p$+$f_e$+$f_a$ | 22.75 (1.61) | 6.55 (0.74) | 2.58 (0.11) | 0.10 (0.06) | 0.35 (0.10) | 0.55 (0.10) |
| All | 19.00 (0.45) | 6.35 (0.08) | 2.56 (0.02) | 0.18 (0.02) | 0.39 (0.00) | 0.43 (0.02) |



(a) $f_p$ (b) $f_v$ (c) $f_b$ (d) $f_a$ (e) $f_e$

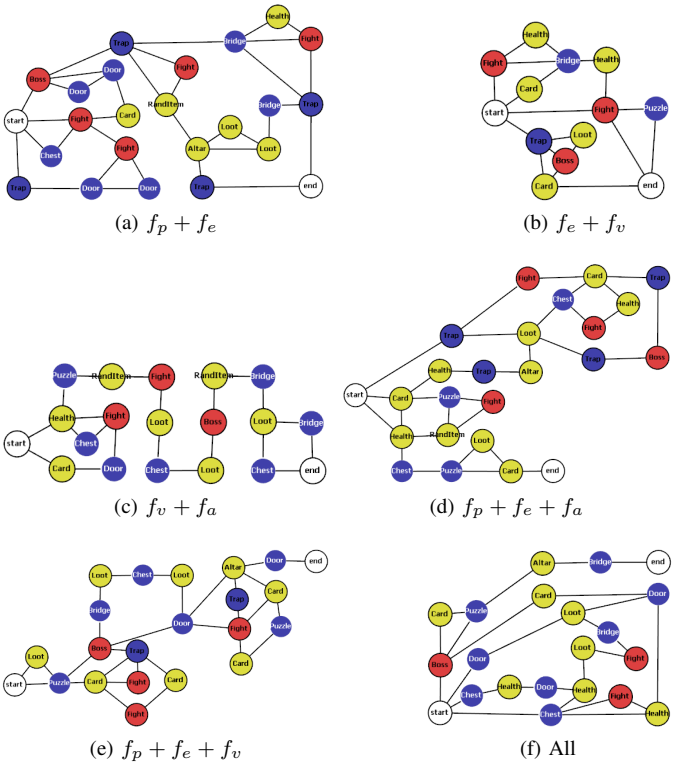
Fig. 3. Mission graphs of the fittest individuals evolved on single objectives.

since it depends on $d_{s,e}$ for its normalization procedure, when $d_{s,e}$ is quickly optimized due to the $f_p$ dimension then the number of nodes which must be added before $f_e$ reaches even sub-optimal score increases — requiring more mutations and thus more generations.

### B. Quality of Final Mission Graphs

While observing the progress of optimization in Section IV-A from a purely quantitative perspective provides insights on the fitness design, it is perhaps more worthwhile to observe the final mission graphs from the perspective of their potential in-game use. Towards that effect, this section evaluates the fittest final mission graphs (according to different objective functions) in terms of their size, shortest path length, branching factor and composition. Such metrics, which are shared by all mission graphs regardless of the objective function used to evolve or evaluate them, allow for a better comparison between the patterns favored by the different objectives.

Table IV contains the metrics' scores of the fittest individuals evolved towards different objectives; the level heuristics chosen evaluate the structure of the graph (e.g. its size and branching factor) and the composition of its nodes (i.e. how many of them belong to the reward, fight, or puzzle category). We observe that the ratio of puzzles, fights and rewards tends to fluctuate significantly (based on the standard deviation) between individuals, even when they are optimized towards the



(a) $f_p + f_e$ (b) $f_e + f_v$ (c) $f_v + f_a$ (d) $f_p + f_e + f_a$ (e) $f_p + f_e + f_v$ (f) All

Fig. 4. Mission graphs of the fittest individuals evolved on multiple objectives.

same objective. This should not be surprising considering the fact that when adding new nodes or changing existing ones, the node type is picked randomly. Moreover, the objectives $f_p$ and $f_e$ do not differentiate between node types. In graphs evolved towards $f_p$ or $f_e$ or their combination, the number of puzzle and reward nodes is roughly equal, with fight nodes being roughy half the number of each other node category. In the case of $f_v$, the fitness of eq. (3) rewards changes in type between adjacent nodes, although this does not seem to affect the number of fight nodes in a significant way; therefore, variation likely alternates between reward and puzzle nodes rather than adding more fight nodes. Finally, since $f_a$ and $f_b$ specifically focus on reward nodes when evaluating their safety or balance, they create mission graphs with far more rewards than any other type. However, when combining $f_v$ with $f_a$ or

$f_b$ (e.g. for $f_v + f_a$ or all objectives), the number of rewards stays close to that of puzzles due to the variation requirement.

Regarding the topology of the missions, from Table IV it is obvious that $f_e$ and $f_a$ create larger mission graphs (graph size) although that does not ensure that the end node is far from the start node. Meanwhile, the fittest mission graphs for $f_p$ always have a shortest path between start and end node equal to 9 in all runs; this is not surprising as this fitness directly rewards mission graphs with 5 to 10 nodes and the highest value of eq. (1) is when $d_{s,e}$ is around 9. Additionally, optimal graphs for $f_p$ have only slightly larger graph size than shortest path length: all nodes of the mission graph are on the shortest path as evidenced by the low branching factor. Missions evolved towards $f_e$ have the highest branching factor as $f_e$ directly rewards a much larger flood filled area around the start node than the shortest path length to the end node. Mission graphs for $f_v$ and $f_b$ can reach optimal values without reaching a large graph size; this explains why in Fig. 2a these fitness dimensions are optimized so quickly as a few mutations which add nodes to the mission can yield optimal scores. However, these same fitness dimensions when combined with others ($f_a$, $f_p$ or $f_e$) can create large graphs which still have high scores in that dimension (e.g. in $f_p + f_e + f_v$). Finally, combining all fitness dimensions seems to create levels with the best traits of each objective: large graphs, with long paths from start to end node (although not as long as when $f_p$ is optimized alone) and a high branching factor. It should be noted that when optimizing both $f_e$ and $f_p$ (e.g. when combining all fitness dimensions), the graph size is larger than when $f_e$ is optimized by itself since $f_p$ rewards longer paths, pushing $f_e$ to add more nodes to the mission graph in order to increase the covered area between start and end node up to triple the length of the shortest path.

Figures 3 and 4 show the fittest mission graphs for each of the objectives when optimized alone or in combination, respectively. These graphs support the conclusions from Table IV: the graph for $f_p$ has no side-passages outside the single path to the end node, the graph for $f_e$ is large but only one node separates start and end node, the graph for $f_v$ and $f_a$ are very small while the graph for $f_a$ mostly contains reward nodes. It is worthwhile to investigate why $f_v$ and $f_a$ are optimal despite their small size: the graph for $f_v$ has only two non-neutral nodes, which are different and thus assign an optimal $f_v$ score according to eq. (3). Indeed, having more than three non-neutral nodes (granted that there are three such categories) would be more difficult to optimize due to random node assignment, causing $f_v$ to actively favor smaller graphs. On the other hand, the graph for $f_b$ has two rewards placed symmetrically to all other nodes: due to the reward nodes' connections, all nodes are actually unsafe (i.e. equally close) to both rewards and thus the mission graph is "balanced" in terms of safe areas around rewards, with the caveat that there are no such safe areas for either reward.

Observing Figure 4, we observe that all graphs are much larger and complex when optimizing multiple objectives. The paths from start to end node also seem more 'interesting' from the perspective of progression between node types ($f_e + f_v$ is an exception, as the hero can reach the exit node by crossing one fight node). Of particular note is the graph for $f_a + f_v$, where the path to the end node (which lacks many side-passages) consists of shifts between reward nodes and fight or puzzle nodes, shaping a gameplay that oscillates between tension and relaxation. When all objectives are optimized in Fig. 4f, an interesting pattern emerges: there is extensive branching in the first steps between start and end node, so if the hero takes the right choice at the start then they can reach the exit quickly and without much decision-making later (no branching paths near the end node) or much challenge (one fight along that path). However, if the hero takes the wrong initial decision they can get lost in mazelike side-passages which can make them go in circles back to the start node.

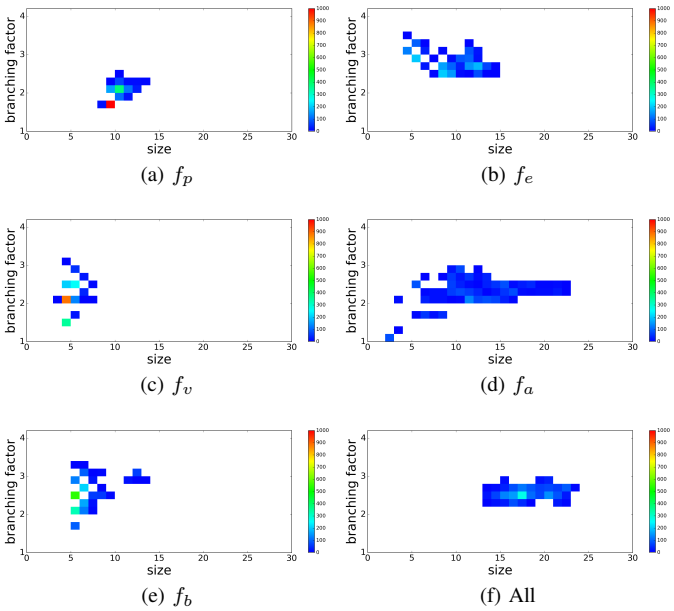



Fig. 5. Graph size versus average branching factor of all final populations evolved for different objectives.

### C. Expressivity Analysis

While observing the fittest mission graphs in Section IV-B provides vital insight into the patterns favored by these objectives, only the one fittest individual per run is assessed. On the other hand, the expressive range [16] of the generator can assess the variety of possible results when optimizing different objectives. The two dimensions explored in this paper are the graph size and branching factor: both of these metrics are not directly targeted by the objectives, as suggested by [16], and are indicative of the actions the hero has to make and the decisions they have to take respectively.

Figure 5 shows heatmaps of the branching factor and graph size values of the final populations of all runs, i.e. a total of 2000 individuals per objective. We observe that the $f_p$, $f_v$ and $f_b$ have the most consistent results, with little spread and most individuals centered in specific areas of this expressivity space.

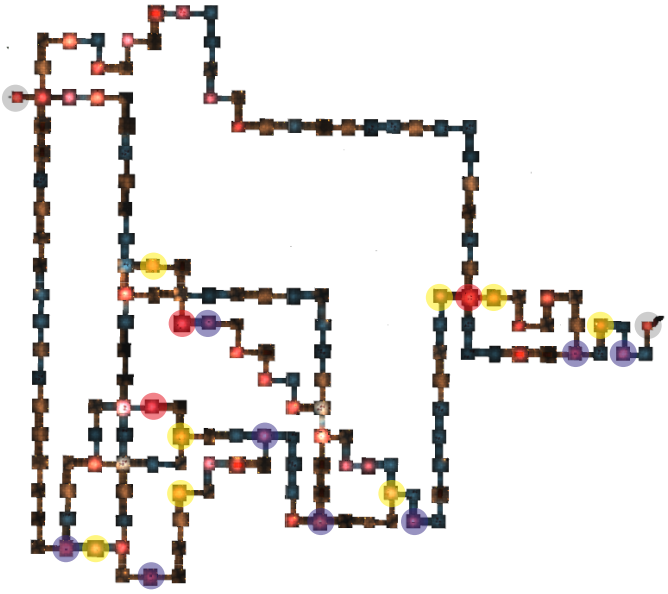
The vast majority of graphs evolved for $f_p$ have a branching factor of less than 2 and a size of 9 nodes, although when the branching factor increases the graph size also increases (since the shortest path is likely still 9 nodes, the extra branching paths add to the graph size). Most graphs evolved for $f_v$ are very small (4 or 5 nodes) and no mission graphs have more than 6 nodes; a similar expressivity is exhibited by $f_b$ although the branching factor is higher. In contrast, graphs evolved for $f_e$ or $f_a$ exhibit more expressivity, being able to create very small mission graphs (e.g. with only start and end nodes in the case of $f_a$ as shown by the bottom-left corner of its heatmap) but tending towards larger mission graphs. Graphs evolved towards $f_e$ tend towards more branching paths than those evolved via $f_a$, which tend towards larger graphs. Finally, when combining all objectives, the expressivity of the results is interesting as it is not similar to that of any individual fitness dimension. Evolved graphs of Fig. 5f are larger with average branching factors, and the values are less dispersed on either metrics than for most of the dimensions. This points to an interesting consensus reached by the — sometimes conflicting — fitness dimensions being optimized.

### D. Example Level

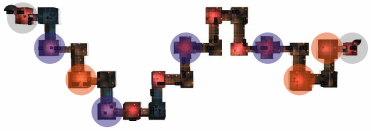
Since the player will experience the evolved mission graphs as a spatial layout of the dungeon of *Dwarf Quest*, it is worthwhile to investigate how such a level architecture would be. The evolved mission graphs are post-processed and then refined via the mixed-initiative grammar-based system of [13], which creates a larger and more detailed mission graph. This refined mission graph is converted into *Dwarf Quest* levels by the layout solver described in [12].

Figure 6 illustrates level architectures for *Dwarf Quest* based on the evolved mission graphs of Figures 3 and 4. The actual rooms which contain nodes in the mission graph are shown in circles of different colors. The level in Fig. 6a is created from the mission graph of Fig. 4f, which was evolved to maximize all objectives. It is immediately obvious that most rooms in the final level layout are empty and in many cases form long corridors to connect the nodes. This is due to the high branching factor of the graph in Fig. 4f, which forces the layout solver to connect areas far away spatially to their adjacent nodes in the mission graph. In contrast, the central part of the dungeon has fewer empty rooms, with only a couple of rooms between each pair of mission graph nodes.

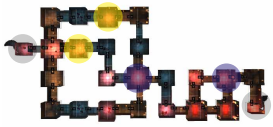
It should be noted that simpler mission graphs with less branching, such as the graph evolved for $f_p$ in Fig. 3a, result in far fewer empty rooms as the level is essentially a single path from start node to end node (see Fig. 6b). Similarly the small yet branching mission evolved for $f_b$ in Fig. 3c creates a similarly simple level (see Fig. 6c) which contains several empty rooms without being exaggerated. The layout solver used for these conversions seems less suited for creating levels with high branching factors or complex topologies, which is also evidenced by the need for the post-processing steps described in Section III-D. By adjusting the layout solver to place graph nodes closer to one another, many of the issues of extraneous rooms could be avoided.



(a) Level layout for all objectives



(b) Level layout for $f_p$



(c) Level layout for $f_b$

Fig. 6. Level layouts created from missions of Fig. 4f, 3a and 3c. Rooms included in the mission graph are highlighted as circles of different colors. Red, yellow, and blue circles indicate fights, rewards, and puzzles respectively. Gray circles are the start node (left-most) and end node (right-most). In the above illustrations, bright rooms were necessary to place this mission into space, while dark rooms were added as part of the variation process.

## V. Discussion

The results in this paper highlighted the strengths and weaknesses of search-based mission generation, as well as the patterns favored by different objectives of Section III-C. Overall, evolving towards a single objective tends to result in one-dimensional graphs which e.g. have no branching (and thus require no decision-making from the player) or have very trivial level traversals with a couple of non-neutral nodes. Meanwhile, aggregating the scores of multiple fitnesses into a simple sum results in more interesting mission graphs with emergent features such as a larger size or pacing between challenge and relaxation. Observing the way each fitness dimension is optimized when aggregating all objectives hints at the fact that some of the objectives are conflicting and thus a multi-objective optimization approach [17] would probably enhance the quality of the results. However, even with the admittedly naive aggregated approach the outcomes are useful: optimizing all objectives simultaneously creates the most interesting missions with long paths to the end, multiple side-passages and a variety of fight, reward and puzzle nodes.

When assessing the quality of the fittest individuals with respect to their topology and variety of nodes, it is obvious that there are far fewer fight nodes than other types. From a designer's perspective, fights are the most challenging and interesting encounters to be had in the dungeon as they involve the most varied game mechanics (including expending rewards found in the dungeon, such as battle cards). The lack of fight nodes was an artifact of the random node type selection in the different mutation operators: the two types of fight nodes were less often picked than the 4 or 5 node types in the other categories, especially since the boss node could be picked once per level. This could be countered by biasing the choice of fight nodes with a higher probability. More interestingly, designing objectives on the 'quality' of the fight node progression could also enhance the importance of fights in the generated missions. To a degree, the variation ($f_v$) objective achieves that effect, and mission graphs that optimize it (such as in Fig. 4c) alternate between fight or puzzle nodes and reward nodes. However, putting explicit emphasis on fight nodes and e.g. the placement of the boss node towards the end of the mission could improve the current results.

As noted in Section IV-D, the level layouts created from the mission graphs often contain too many empty rooms. The mission generator for the most part creates graphs that adhere to the rules of the level generator, especially after post-processing. Post-processing steps may seem overbearing, such as omitting nodes with one edge: these steps are less destructive than it seems, however, since the mutation operators rarely result in single-edge nodes (none of the examples in Fig. 3 and 4 have non-neutral nodes with one edge). Nonetheless, the resulting spatial structure may be less suited for gameplay than the mission graph suggests. Apart from changes to the level layout solver in order to better handle the branching mission graphs created by some objectives, this limitation can be addressed by evaluating the final level instead of — or in conjunction to — the mission graph. An interesting approach could be to evaluate how much empty space (i.e. non-node rooms) are in the final level layouts of a certain mission graph, applying a penalty to its fitness (calculated as per Section III-C) proportionate to the amount of empty rooms.

## VI. Conclusion

This paper described an approach for generating game levels by evolving their indirect representation (a player's action sequence) rather than their direct representation (room layout). Mission graphs representing the possible paths of the player for reaching the goal (end node) were evolved towards different objectives inspired by general game design patterns such as exploration, balance and safety of resources [14]. Experiments in evolving mission graphs towards different objectives individually and in conjunction showed that while different objectives favor different patterns, combining multiple objectives (or even all objectives) results in more complex and more interesting mission graph structures. These more complex graph structures similarly result in quite complex level layouts, which may increase player fatigue when navigating them. How to address such limitations, and evaluate both the graph structure and the final level layout (i.e. the direct and indirect representation of a game level) is a promising direction for future research.

## VII. Acknowledgment

We would like to thank Dylan Nagel for giving us full access to *Dwarf Quest*'s source code, as well as Rafael Bidarra and Roland van der Linden for the layout solver of *Dwarf Quest*. This work was supported, in part, by the FP7 Marie Curie CIG project AutoGameDesign (project no: 630665) and the Horizon 2020 project CrossCult (project no: 693150).

## References

[1] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, no. 3, pp. 172–186, 2011.

[2] A. M. Smith, E. Butler, and Z. Popovic, "Quantifying over play: Constraining undesirable solutions in puzzle design," in *Proceedings of the International Conference on the Foundations of Digital Games*, 2013.

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

[4] L. Cardamone, G. N. Yannakakis, J. Togelius, and P. L. Lanzi, "Evolving interesting maps for a first person shooter," in *EvoApplications (1)*, 2011, pp. 63–72.

[5] A. Liapis, C. Holmgård, G. N. Yannakakis, and J. Togelius, "Procedural personas as critics for dungeon generation," in *Applications of Evolutionary Computation*. Springer, 2015, vol. 9028, LNCS.

[6] D. Ashlock and C. McGuinness, "Automatic generation of fantasy role-playing modules," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2014.

[7] D. Ashlock, S. Risi, and J. Togelius, "Representations for search-based methods," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015, (In progress).

[8] G. Smith, J. Whitehead, and M. Mateas, "Tanagra: Reactive planning and constraint solving for mixed-initiative level design," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 201–215, 2011.

[9] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, G. N. Yannakakis, and C. Grappiolo, "Controllable procedural map generation via multiobjective evolution," *Genetic Programming and Evolvable Machines*, vol. 14, no. 2, pp. 245–277, 2013.

[10] J. Dormans and S. C. J. Bakkes, "Generating missions and spaces for adaptable play experiences," *IEEE Transactions on Computational Intelligence and AI in Games. Special Issue on Procedural Content Generation*, vol. 3, no. 3, pp. 216–228, 2011.

[11] D. Yu, "The full spelunky on spelunky," 2011, accessed 14 April 2016. [Online]. Available: http://makegames.tumblr.com/post/4061040007/the-full-spelunky-on-spelunky

[12] R. van der Linden, "Designing procedurally generated levels," Master's thesis, TU Delft, 2013.

[13] D. Karavolos, A. Bouwer, and R. Bidarra, "Mixed-initiative design of game levels: Integrating mission and space into level generation," in *Proceedings of the International Conference on the Foundations of Digital Games*, 2015.

[14] S. Björk and J. Holopainen, *Patterns in Game Design*. Charles River Media, 2004.

[15] A. Liapis, G. N. Yannakakis, and J. Togelius, "Towards a generic method of evaluating game levels," *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference*, 2013.

[16] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010.

[17] C. A. C. Coello, "A comprehensive survey of evolutionary-based multiobjective optimization techniques," *Knowledge and Information systems*, vol. 1, no. 3, pp. 269–308, 1999.