Mission Evolution for Dungeon Levels

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: antonis.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.**

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, and generative grammars [2] which define the creation and expansion rules of an artifact and can gradually increase its level of detail.

The dual representation for game levels (as a mission and as a space) was first introduced in [2] and expanded in [3], 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 [4], 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 [5], 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 letting the generative grammars to expand on that sketch automatically (or with some human curation).

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 they do 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 [6]; 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.

II. Methodology

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 into one or more rooms in which the action will take place by a layout solver. A more detailed description of this process is provided in [5]. There are 14 types of nodes, 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 or trapped rooms), while reward nodes have no opposition but provide power-ups for future fights 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 mission 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. 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. 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 evolves an initial population of individuals in order to maximize a fitness function consisting of one or more objectives. 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%, 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:

- Insert a node on an existing edge
- Add a node and an edge between two nodes
- Change a node's type
- Delete a node [1]
- Add an edge between two nodes
- Delete an edge between two nodes

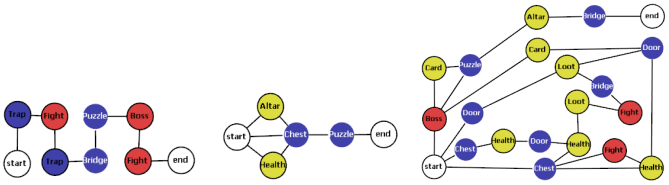
Mutation operators do not affect neutral nodes, since this affects the feasibility of the individual. Also, mutation operators are not allowed to place more than one boss node and more than one altar node per level; other node types are chosen in those cases. The mutation probabilities are based on preliminary 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 [7] and their mathematical formulations in [8], 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 (f_p).* The number of nodes along the shortest path between start and end nodes is normalized by a bell curve to give optimal scores to paths with 5 to 10 nodes.
- *Exploration (f_e).* Inspired by [8], 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. This metric is normalized to give optimal scores to exploration covering three times as many nodes as the shortest path.
- *Variation (f_v).* The percentage of edges that connect nodes of different categories, excluding start and end nodes.

[1] A 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.



(a) f_p (b) f_b (c) All

Fig. 1. Mission Graphs of the fittest individuals based on the single fitness functions.

- *Dispersed rewards (f_s).* Based on [8], the function 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).
- *Balanced rewards (f_b).* Based on [8], the function evaluates whether every reward has an equal number of safe nodes around it as every other reward.

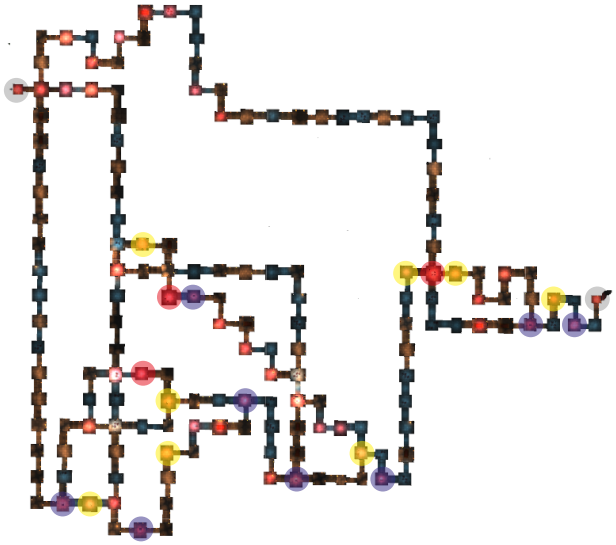
D. From Mission Graphs to Levels

In order to create the game's final levels, evolved mission graphs are refined via the mixed-initiative grammar-based system of [5], 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 [4], which is in turn constrained by the map options of the *Dwarf Quest* game. Due to the constraints of these systems, several post-processing steps must be applied on the evolved mission graphs. For example, non-neutral nodes with only one connection are omitted, since *Dwarf Quest* rooms must have at least two corridors. Also, 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.

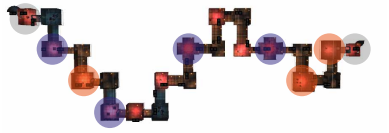
III. Results

Figure 2 illustrates level architectures for *Dwarf Quest* based on the evolved mission graphs of Figures 1 and **??**. The actual rooms which contain nodes in the mission graph are shown in circles of different colors. The level in Fig. 2a is created from the mission graph of Fig. 1c, 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. 1c, 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.

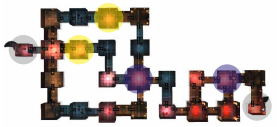
It should be noted that simpler mission graphs with less branching, such as the graph evolved for f_p in Fig. 1a, result in far fewer empty rooms as the level is essentially a single path from start node to end node (see Fig. 2b). Similarly the small yet branching mission evolved for f_b in Fig. 1b creates a similarly simple level (see Fig. 2c) which contains several empty rooms without being exaggerated. The layout solver



(a) Level layout for all objectives



(b) Level layout for f_p



(c) Level layout for f_b

Fig. 2. Level layouts created from missions of Fig. 1c, 1a and 1b. Rooms included in the mission graph are highlighted as circles of different colors. Red, yellow, and blue circles indicate fights, rewards, and puzzles. 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.

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 II-D. By adjusting the layout solver to place graph nodes closer to one another, many of the issues of extraneous rooms could be avoided.

IV. 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 [7].

V. 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).

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] 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.

[3] 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.

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

[5] 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.

[6] 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).

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

[8] 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.