Procedural Content Generation of Level Layouts for Hotline Miami

Joseph Alexander Brown, *Member, IEEE*
Artificial Intelligence
in Games Development Lab
Innopolis University, Innopolis
Russia 420500
j.brown@innopolis.ru

Bulat Lutfullin
Artificial Intelligence
in Games Development Lab
Innopolis University, Innopolis
Russia 420500
b.lutfullin@innopolis.ru

Pavel Oreshin
Artificial Intelligence
in Games Development Lab
Innopolis University, Innopolis
Russia 420500
p.oreshin@innopolis.ru

***Abstract*—*Hotline Miami* is a series of 80s themed arcade shoot-em-up games. In early 2016, a beta level editor was provided for the second release in the series which is utilized in this paper as a basis for a Procedural Content Generation (PCG) method for level development. We provide an evolutionary algorithm for the placement of rooms and examine a number of statistical measures for the outcome of our representation and it is classified as part of an existing PCG taxonomy. The level generator demonstrates control of the features in the developed levels based upon a set of representative fitness functions. This is a step towards a full level creator, and is implemented in a commercial game and is not just a technical demo.**

I. Introduction

The *Hotline Miami* series of games [1][2] involves a set of psychopathic killers who wear various animal masks during their attacks. It stylistically takes inspirations from movies like *Drive* and other neo-noir films with a 1980s bright neon visuals and synthesizer europop soundtrack. The storyline has been compared to the movies of David Lynch. The game is a top-down view of what looks like an architecture diagram of the building where a player has perfect knowledge of the space. Game play is fast paced, meeting with the idea of a psychopathic murder's spree, and the player is killed with a single hit by an enemy; the player upon death is immediately re-spawned into the level at the start in order to play through the level again. This makes the design of such levels decidedly different from the majority of game genres looked at for implementing Procedural Content Generators (PCG) for level design, see [3].

Hotline Miami levels are best examined like top down dungeon crawlers and non-digital games such as *Dungeons & Dragons* [4], [5], [6], [7], [8]. Action based Role Playing Games (ARPGS) such as *Diablo* are close to the feel of *Hotline Miami*, however, the level size is much larger in *Diablo*, and considerations of line of sight are not as pressing in a hack and slash world with a character able to take a number of hits, compared to the immediate death and re-spawn mechanic of *Hotline Miami*. See Fig 1 for examples of the levels.

Many representations were introduced that allow the designer control over the type of mazes that are generated, e.g., caverns, rooms, and etc. Ashlock et. al.'s studies use a checkpoint-based genetic algorithm that optimizes the fitness function toward creating mazes with user-defined characteristics[6], [7]. McGuinness and Ashlock [5] showed that the resulting mazes generated in the previous work could be tiled in order to create larger, more complex mazes. This technique could be extended to include other generation types, e.g terrain, in order to create large maps. Valtchanov and Brown [8] use a genetic programming approach to create *Diablo* style levels via the placement of pre-generated tiles. The tile method follows a number of current industry practices of the creation of random levels with game elements. The fitness functions used include a placement of 'event rooms' which could contain boss battles or treasure rooms. Aslam et al. [9] uses Genetic Algorithms as a placement method for title in a serious game, and demonstrate that they out preform humans on relatively simple placement tasks compared with an objective measure of fitness.

There have also been a number of PCG generations for the design of the internals of buildings, mostly for architectural design problems such as those in [10], [11], [12]. Though their evaluations are based on optimizations of floor plans for realistic buildings and look to architectural requirements as their constraints. These methods primarily assume the external structures are fixed and are akin to layout and packing problems which have additional restrictions on the flow between the named areas, or restrictions based on use of the room, e.g. placing bathroom beside the master bedroom.

However, in all of these methods little integration to real games has been presented. In this paper looking at *Hotline Miami* it was necessary to reverse engineer the level files in order to place the developed levels into a commercially playable game, rather than acting as a simple technical demo. We present a Genetic Algorithm (GA) approach to the layout for levels using the reverse engineered level files, and examine a set of fitness functions for their suitability to create level layouts which meet with the constraints of player movement and with creating a connected tight space for players to act within.

The remainder of this paper is organized as follows: Section II examines the suitability of Genetic Algorithms as a search based procedural content generation method for levels of

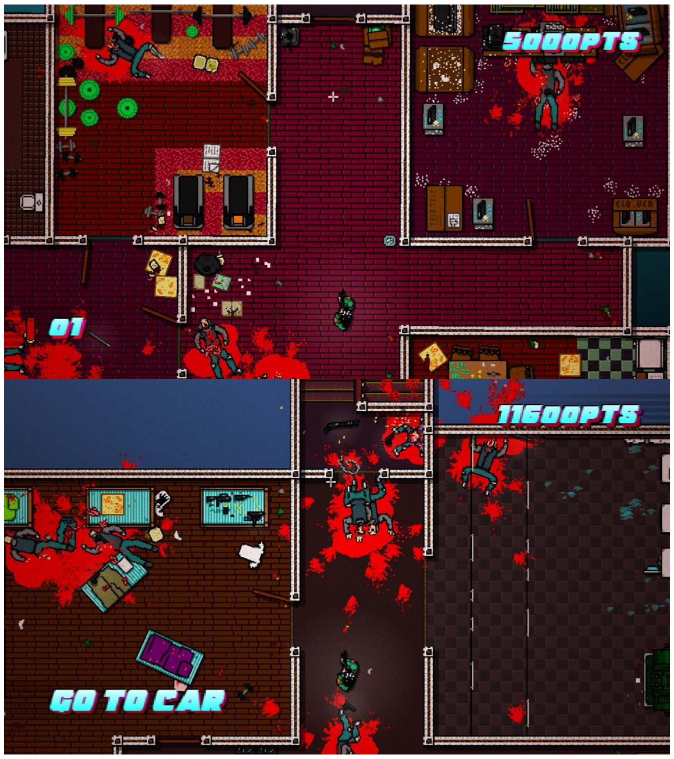


Fig. 1: Level of *Hotline Miami 2: Wrong Number* [2]. Note the connectivity via doorways and the tight room placement.

Hotline Miami paying special attention to the issues of the representation of the levels and the transformation of chromosomes into working content meeting with the constraints of the game and the reverse engineering process. Section III examines the generated levels for a variety of fitness evaluation functions, demonstrating the ability for a designer to control the generative method for their purposes. Finally, Section IV and examines the additional work necessary in order to utilize these layouts within the game.

II. Methods

Evolutionary Computation (EA) uses principles from Darwinian evolutionary theory, specifically a fitness biased selection method, in order to solve problems in diverse field such as optimization and planning, see [13] for a general overview. We are interested in its ability to work as a design tool, which allows for a diversity of solutions, as well as optimization against a set of requirements. Genetic Algorithms (GA) are EAs developed first by Holland [14] which have populations of candidate solutions to the problem called *chromosomes*. Chromosomes are tested for their ability to solve a problem via a *fitness evaluation* which produces a fitness score. The fitness score informs a fitness biased *selection* which decides on the chromosomes which undergo *variation operators*. GAs use the notion of a genetic breeding between pairs of chromosomes, a *crossover*, and small changes to a single chromosome, *mutation*, as variation operators.

A. Representation and Translation

1) Representation: The representation used in this study is similar to the *indirect positive* method used by Ashlock et. al [6] which defines rooms using an integer representation of the areas they take up. We do not utilize a mapping function nor is placement relative to previous rooms such as in [8], but instead expressly place rooms into the space.

Rooms are defined as a five-tuple (x, y, l, w, t) where:

- x is the starting x-coordinate of a room
- y is the starting y-coordinate of a room
- l is the length of the room
- w is the width of the room
- t is the placement type of a room, if it places on-top of (T) or under (U) current rooms.

Chromosomes are sequences of rooms which will be placed in order. The first room is placed into the space. In order to ensure connectivity, a new room is only placed into the layout when it overlaps with a currently existing room. This allows for the algorithm to create levels with a number of rooms less or equal to the length of the chromosome. Doorways are important within *Hotline Miami* as they block line of sight for enemies, and doorways can be used to attack enemies by knocking them down, which stuns them for a *coup de grâce* by the player. The potential doorways are defined by adjacent borders of rooms. Doorways are placed at the random location in each adjacent wall. A T type room, the wall of the newly placed room defines the potential doors, a U type room defines the walls of the previously placed rooms as being the locations of potential doors.

Figure 3 displays the generated level rendered by the in game development render for the chromosome in Table 2.

2) Translation: The translation step changes the chromosome into a full working level for the game. A level editor for *Hotline Miami 2: Wrong Number* was released in beta, and via an exploratory process the level files were reverse engineered.

A Hotline Miami level is described by a set of plain text files, described bellow, with each line encoding a single value.

- *level.hlm*
  'Hotline Miami level' file. Holds meta information about the level such as name of the level, name of the author, and size of the level.
  The level of Hotline Miami is organized in square tiles each having the size of $16px$ by $16px$. Maximum allowed size of the level is $1088px$ by $768px$.
- *level.ver* 'version' file. Contains a single integer, which represents version of the editor. At the moment of writing the value was '2'.
- *level0.obj* 'objects' file. Holds information about the player character, player's car, doors, enemies and decoration sprites. Each entity is described by its object ID, coordinates (x, y), sprite ID, rotation in degrees of the sprite, behaviour type for AIs(static, patrol, idle), cut scene flag to show if it is a persistent object during cut scenes.

(8,11,6,4,U)	(4,14,7,6,O)	(5,9,11,8,O)	(9,7,9,7,U)	(4,14,4,3,O)	(5,5,7,5,O)

Fig. 2: Sample Chromosome

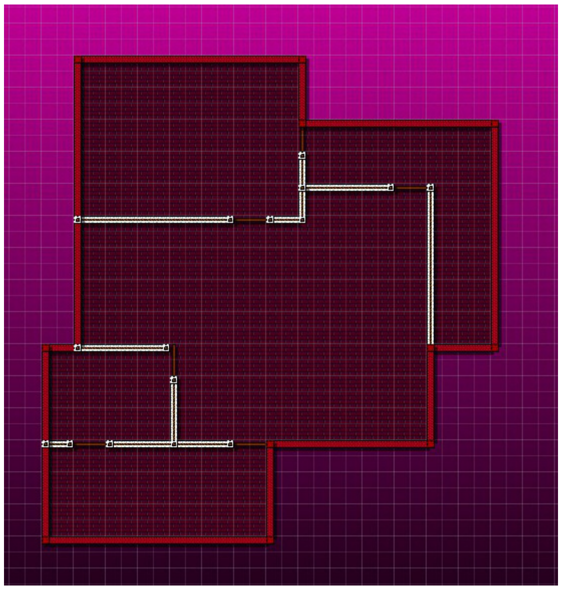


Fig. 3: Internal rendering of the generated level.

- *level0.tls* ‘tiles’ file. Describes the floor tiles. Each tile is a square of size $16px$ x $16px$.

- *level0.wll* ‘wall’ file. Contains descriptions of wall segments. Each wall is assumed to be 2 tiles or $32px$ wide.

- *level0.play* ‘play’ file. This file contains all game entities. On level launch, every entity from another files is transferred here. Level editor works without this file, but game will not launch.

This was a major undertaking to decode the files to allow for a new level to be input into the system and demonstrates that developers who have explicitly provided level editors can prevent ease of use of their systems for the modding and academic communities.

B. The Algorithm

1) Internal Representation: Walls in Hotline Miami span two tiles, so to be properly translated into Hotline level format we used ’Wall Segment’ class which holds block of 2x2 tiles. For each segment there is corresponding number describing for which room this segment belongs.

2) Level Building: The core of the algorithm is the level building step, which takes a chromosome and subsequently adds each rectangle to the level. During that process a room is created and assigned a list of segments, that belong to the room. If overlapping rooms splits another room into two sections, then they will not be placed. After all chromosomes

```
1: intialize segments with -1
2: for each chromosome in gene do
3:    if chromosome is first OR chromosome is overlap
      existing then
4:       fill corresponding segment with this chromosome
5:    end if
6: end for
7: for each adjacent room do
8:    place door at random position in wall
9: end for
```

Fig. 4: Pseudocode for level building

have been added the algorithm places a door between adjacent rooms.

C. Variation Operations

The evolution progresses for 100 generations, rounds of fitness evaluation followed by crossover/mutation, with a randomly initialized population subject to the following operations:

1) Crossover: The crossover utilized is a uniform order method with a probability of 0.5 for taking a room from each of the parents. See Fig. 5 for an example.

2) Mutation: The mutation operation selects a random chromosome and creates new, completely random chromosome instead of selected one. See Fig. 6 for an example.

D. Selection

The population in this study is set to 20 chromosome levels of size 10, which are evaluated via the fitness function. The crossover of the two fittest individuals is copied over the entire population, and the children are then subjected to the application of a mutation.

E. Fitness Evaluations

In order to demonstrate the method to the show a control on the level development, a number of fitness evaluations

Parent 1

(12,54,25,17,O)	(20,34,10,14,U)	(20,2,11,17,U)	(54,56,78,12,O)	(20,34,10,14,U)

Parent 2

(12,23,34,5,U)	(1,23,34,21,O)	(3,12,2,5,O)	(20,32,8,55,U)	(4,2,42,3,U)

Child

(12,54,25,17,O)	(1,23,34,21,O)	(20,2,11,17,U)	(20,32,8,55,U)	(20,34,10,14,U)

Fig. 5: Example uniform order crossover with probability of 0.5 of the parents which was randomly selected to occur after the third room

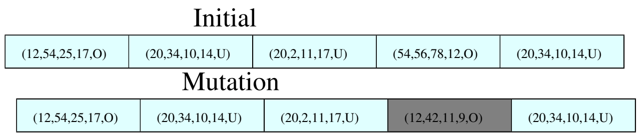


Fig. 6: Mutation in the chromosome at position four, labeled in dark gray.

were proposed in order to discover a setting which will create levels with properties conducive to the game. At first, fitness was set to encourage maximum amount (10) of rooms. But in this case, rooms were disconnected from each other. To fix that, a new constraint was introduced: only overlapped or connected rooms are participated in fitness calculation. With this conditions other fitness was implemented:

1) Maximize Rooms: This evaluator increases fitness value for each connected room.

$$Fitness = N_{room} \tag{1}$$

This provides levels with larger amounts of rooms that are connected.

2) Maximize Area : This evaluator increases fitness value for each tile inside building. One tile is a unit of area.

$$Fitness = area \tag{2}$$

This provides levels with very large rooms and many rooms.

3) Minimize Area: This evaluator decreases fitness value for each tile. But it has maximizing number of rooms in priority, because otherwise there will be one small room in the output of generator.

$$Fitness = 1000 * N_{room} - area \tag{3}$$

Attempts to provide small rooms and also cause rooms to not be utilized in the chromosome.

F. Graph Based Fitnesses

It is assumed that the reader has some familiarity with graph theory, see [15]. A *Simple Graph* $G(V, E)$ is a non-empty set V of vertexes or nodes and a set E of unordered pairs of elements of V, called edges. Two distinct nodes, v_1 and v_2, are said to be *neighbours* if $(v_1, v_2) \in E$. The number of edges in E which contain a vertex is called the *degree* of the vertex. The *diameter* of a graph is the largest number of edges in any shortest path between any two of the vertices.

1) Maximize Degree: For given chromosome, a graph is constructed, where every room defines a vertex, and each adjacent wall defines connection between two rooms. The fitness evaluator returns cardinality of edge set.

$$Fitness = |E| \tag{4}$$

This fitness function should create levels with many connections between rooms.

2) Maximize Diameter: For given graph, this evaluator calculates the diameter and returns the sum of rooms quantity and diameter.

$$Fitness = 1000 * N_{room} + diameter \tag{5}$$

where N_{room} is the number of rooms.

This fitness function should encourage long paths between any pair of rooms.

3) Minimize Diameter: For given graph, this evaluator calculates the diameter and returns the difference of rooms quantity and diameter.

$$Fitness = 1000 * N_{room} - diameter \tag{6}$$

This fitness function should encourage short paths between any pair of rooms.

G. Fitnesses with Corridor Penalty

1) Corridor Penalty: In all the aforementioned fitness functions narrow rooms in which a character could not move within could result in the building. This fitness function penalizes the gene for each tile in a narrow corridor and for one tile rooms.

$$Fitness = \frac{N_{room}}{(1 + N_{narrow}) * 10^{N_{tiny}}} \tag{7}$$

where N_{narrow} is the number of narrow rooms (with width or height equals to 1 tile) and N_{tiny} is the number of one tiled rooms.

2) Complex Fitness: This evaluator was developed to create interesting and realistic buildings. This fitness function aims to maximize the number of rooms, the diameter of graph, and keep the average degree close to 2. It also penalizes each tile in a narrow corridor and one tile rooms.

$$Fitness = \frac{expDegree * N_{room} * \log(diameter)}{\log(e + N_{narrow}) * 10^{N_{tiny}}} \tag{8}$$

where $expDegree = e^{-(avgDegree-2)^2}$.

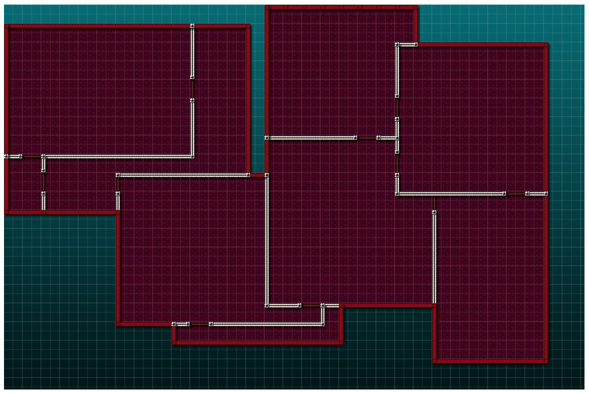
The goal of the evaluation of all these fitness functions is to demonstrate that the fitness function can provide a control to the designer on generation to create a room with the required characteristics.

III. Generated Level Examples

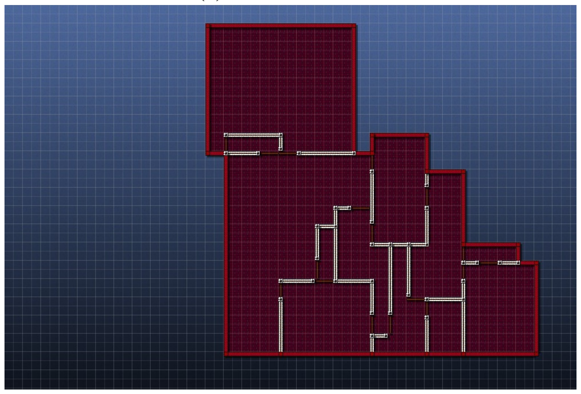
In order to show this control to be effective, a number of levels were generated of each type and compared statistically to demonstrate the effect of the control from the fitness functions above. The evaluated feature set includes the: Number of rooms placed in the final level, the total area taken by the map, the minimum room area, the maximum room area, the number of one tile corridors, the diameter of the graph formed by the connections between doors and the average degree of the graph formed by the connected rooms. The one tile corridors value is important, as a player character cannot move though such corridors, and should be minimized. The fitnesses expressed in Section II-E are

TABLE I: Mean and 95% Confidence internals for each of the fitness evaluation types with 30 runs of the GA. Note that $0E0$ represents a machine zero.

Fitness Measure	Rooms	Area	Min Room Area	Max Room Area	One Tile Corridors	Diameter	Degree
Maximize Rooms	10.00 $\pm 0E0$	293.40 $\pm$14.94	5.17 $\pm$1.45	71.80 $\pm$5.45	15.47 $\pm$3.05	4.03 $\pm$0.27	3.08 $\pm$0.14
Maximize Area	9.33 0.30	433.63 $\pm$7.36	8.87 $\pm$2.10	99.73 $\pm$9.55	12.27 $\pm$2.79	3.73 $\pm$0.26	3.05 $\pm$0.14
Minimize Area	10.00 $\pm 0E0$	227.93 $\pm$10.67	4.19 $\pm$0.91	55.19 $\pm$4.36	16.00 $\pm$2.60	3.90 $\pm$0.20	3.18 $\pm$0.11
Maximize Degree	9.77 $\pm$0.16	291.93 $\pm$14.46	4.23 $\pm$1.13	76.13 $\pm$5.69	15.5 $\pm$2.28	2.97 $\pm$0.15	4.07 $\pm$0.11
Maximize Diameter	10.00 $\pm 0E0$	298.00 $\pm$15.10	5.27 $\pm$1.28	71.93 $\pm$6.23	12.43 $\pm$2.16	5.17 $\pm$0.20	2.84 $\pm$0.11
Minimize Diameter	10.00 $\pm 0E0$	303.37 $\pm$16.46	5.00 $\pm$0.93	72.10 $\pm$6.84	15.50 $\pm$3.00	2.90 $\pm$0.15	3.39 $\pm$0.10
Corridor Penalty	8.67 $\pm$0.30	295.43 $\pm$17.48	10.00 $\pm$1.86	74.10 $\pm$6.46	0.00 $\pm 0E0$	3.77 $\pm$0.25	2.84 $\pm$0.14
Complex	7.70 $\pm$0.26	263.53 $\pm$12.06	11.07 $\pm$1.71	71.77 $\pm$5.40	0.10 $\pm$0.11	4.70 $\pm$0.24	2.08 $\pm$0.07



(a) Maximize Area

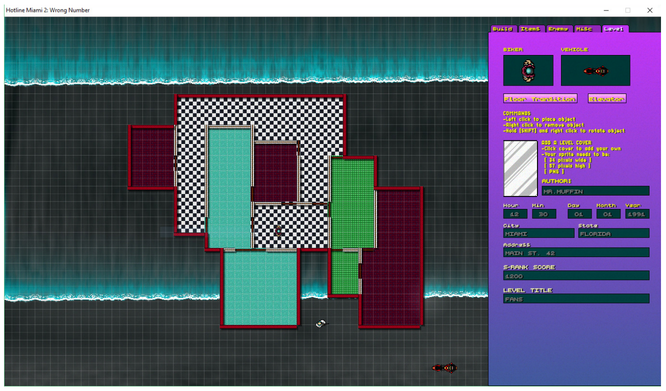


(b) Minimize Area

Fig. 7: Example levels of the Maximization and Minimization of level area

examined on the average of 30 runs of the generator and means and 95% confidence intervals are shown in Table I, in order to show the expression of the levels based on the selected fitness function.

The statistics from the developed levels demonstrate that the



COMMANDS
-Left click to place object
-Right click to remove object
-Hold [SHIFT] and right click to rotate object

ADD A LEVEL COVER
-Click cover to add your own
-Your sprite needs to be:
[34 pixels wide]
[57 pixels high]
[PNG]

AUTHOR: MR.MUFFIN
Hour 12 Min 30 Day 01 Month 01 Year 1991
City MIAMI State FLORIDA
Address MAIN ST. 42
S-RANK SCORE 1200
LEVEL TITLE FANS

Fig. 8: Example level using the complex fitness function in the editor with coloured rooms

generator responds well to control via a fitness function.

Maximize rooms fitness was able to utilize all rooms in all cases, other fitness methods which provided the same outcome were the minimize area and the two diameter controls. Total Area and Min/Max Room Area in maximization of rooms fitness was not that different from the diameter fitness cases. One tile corridors was reduced in the minimization of diameter. Respectively, the Minimize and Maximize Diameter both were significantly different from Maximize Rooms in both diameter and degree. The Maximize Rooms fitness has no controls over area, diameter, or degree, and thus provided a baseline assessment of the generated room layouts.

There is a statistically significant increase/decrease in the room size for the Maximize/Minimize Area while maintaining similar degree and diameter of the graph, as seen in Figure 7. The levels visually demonstrate a common connective look, but the space is compressed or expanded.

Similarly, changes to the min/max diameter of the graph do not significantly change other parameters, however, levels with low diameter have higher degree of about one more. This is expected as reduction of diameter is an outcome higher connectivity between rooms. The GA can successfully see this trade. The maximization of the degree reduced the diameter compared to the Maximization Diameter fitness function, and even was significantly higher from minimization of diameter, accomplished most likely by small trade-offs on the number of rooms utilized.

The Corridor Penalty in its own fitness check removes the one tile rooms completely as part of the fitness. The Complex fitness levels share this removal of tight corridors as they also penalize the smaller rooms. Via removal of these small corridors, there is a significant increase, about five units, except for Maximize Areas were it is only two, in the minimum room area against all fitness functions which do not apply this penalty.

The Complex fitness function provided for levels which are close to some of those seen in the game, such as in Figure 8. The generated examples met with our original requirements of a compact construction of rooms with many connected doors which closely resembles a building layout as seen from above. Interestingly, this fitness function provided for a statistically significantly less rooms than any of the other methods, and was smaller in terms of total area, while keeping the minimum room area relatively high. Degree was kept within a tight distance to exactly two, only deviating upwards by 0.08. The diameter was also the second highest value. Other than the number of rooms, the generated levels were well within our requirements of this fitness function, most likely this is due to the fitness function in a way expecting too much and the room lacking penalization was not high enough to provide selection pressures away from increasing the other values by losing rooms.

IV. Conclusions

As stated above the goal of this project is to fully automate the development of levels for Hotline Miami in two stages: development of the level space and the placement of enemies and power-ups. In this paper we examined only the development of levels and the representation. We have shown a quantitative analysis of the effects of changing the fitness function, in order to allow for different characteristics to be shown in the developed levels. This set of fitness functions, much like those show in [6] and [8], demonstrates an effective control on the features seen in a level under this framework. Controls can be placed on individual features, in this case size and graph measures, to prevent the level generation from demonstrating poor behaviours, i.e. small corridors, or combined into a composite function to provide entire levels meeting with a description of the space.

In future work we aim to include the development of the placement of the enemies into this framework. Further, we have only looked at the development of the level layout in terms of the mechanical properties and have not examined the aesthetics of the rooms — such as placing items to show a room to be a kitchen or a office building.

The system, as we have stated, is currently only looking at the level layout problem an has not examined the placement of enemies, items, and other sprites. While we believe that the levels created can aid a developer in the development of interesting room layouts, we would liked to extend this to placing enemies to meet with a requirement of difficultly in future. This leads to two issues: 1) how to place enemies to meet with skill requirements of the player and 2) how are enemies represented in the game files in order to instance them as part of the generator. The second problem requires some more time attempting to reverse engineer the level files the editor produces, placing various enemies manually via the editor, and examining the file.

Acknowledgments

Thank you to Dennation Games for answering of questions about some of the level storage in the editor files.

References

[1] Dennaton Games, *Hotline Miami*, Devolver Digital, 2012.

[2] Dennaton Games, *Hotline Miami 2: Wrong Number*, Devolver Digital, 2015.

[3] Noor Shaker, Julian Togelius, and Mark J. Nelson, *Procedural Content Generation in Games*, Springer, Sweden, 2016.

[4] Lawrence Johnson, Georgios N. Yannakakis, and Julian Togelius, "Cellular automata for real-time generation of infinite cave levels", in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, New York, NY, USA, 2010, PCGames '10, pp. 10:1–10:4, ACM.

[5] C. McGuinness and D. Ashlock, "Decomposing the level generation problem with tiles", in *2011 IEEE Congress on Evolutionary Computation (CEC)*, 2011, pp. 849–856.

[6] D. Ashlock, C. Lee, and C. McGuinness, "Search-based procedural generation of maze-like levels", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 260–273, 2011.

[7] D. Ashlock, C. Lee, and C. McGuinness, "Simultaneous dual level creation for games", *IEEE Computational Intelligence Magazine*, vol. 6, no. 2, pp. 26–37, 2011.

[8] Valtchan Valtchanov and Joseph Alexander Brown, "Evolving dungeon crawler levels with relative placement", in *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, New York, NY, USA, 2012, C3S2E '12, pp. 27–35, ACM.

[9] Hamna Aslam, Anton Sidorov, Nikita Bogomazov, Fedor Berezyuk, and Joseph Alexander Brown, "Relief camp manager: A serious game using the world health organization's relief camp guidelines", in *Applications of Evolutionary Computation: 20th European Conference, EvoApplications 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings, Part I*, Giovanni Squillero and Kevin Sim, Eds., pp. 407–417. Springer International Publishing, Cham, 2017.

[10] Paul Merrell, Eric Schkufza, and Vladlen Koltun, "Computer-generated residential building layouts", in *ACM SIGGRAPH Asia 2010 papers*, New York, NY, USA, 2010, SIGGRAPH ASIA '10, pp. 181:1–181:12, ACM.

[11] C. Coia and B.J. Ross, "Automatic evolution of conceptual building architectures", in *2011 IEEE Congress on Evolutionary Computation (CEC)*, 2011, pp. 1140–1147.

[12] R. W. J. Flack and B.J. Ross, "Evolution of architectural floor plans", in *Applications of Evolutionary Computation*, Cecilia Chio, Anthony Brabazon, Gianni A. Caro, Rolf Drechsler, Muddassar Farooq, Jörn Grahl, Gary Greenfield, Christian Prins, Juan Romero, Giovanni Squillero, Ernesto Tarantino, Andrea G.B. Tettamanzi, Neil Urquhart, and A.ima Uyar, Eds., vol. 6625 of *Lecture Notes in Computer Science*, pp. 313–322. Springer Berlin Heidelberg, 2011.

[13] Daniel A. Ashlock, *Evolutionary Computation for Modeling and Optimization*, Springer, New York, 2006.

[14] John H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, MA, USA, 1992.

[15] Douglas Brent West et al., *Introduction to graph theory*, vol. 2, Prentice hall Upper Saddle River, 2001.