Automatic level Generation for Tower Defense Games

Yu Du[1], Jian Li[1], Xiao Hou[2], Hengtong Lu[1], Simon Cheng Liu[2], Xianghao Guo[2], Kehan Yang[1], Qinting Tang[1]
[1]College of Computer Science and Technology, Beijing University of Posts and Telecommunications, Beijing, China
[2]LevelupAI, Beijing, China
Email: duyu@bupt.edu.cn

***Abstract*—We created a new method to automatically generate levels for a commercial-grade tower defense game *Kingdom Rush: Frontiers* (KRF) by means of Procedural Content Generation(PCG). Our research focuses on path generation and monster sequence generation. Firstly, we designed a mathematical representation of the game's path using geometric rules, and then we used search algorithms to develop an algorithm, which can generate new paths that are similar to the paths in original games. We implemented monster sequence generation with genetic algorithm in which we designed a representation of genes that reused some human-designed elements. In addition, we also designed a fitness function to make sure the generated level has moderate difficulty and playability. Finally, we automatically generated new levels for KRF with the combine of path generation and monster sequence generation. We used a turing test to show that our PCG levels are hard to be distinguished from the original levels.**

***Index Terms*—Procedural Content Generation, games, evolutionary algorithm, search, automatic design**

I. INTRODUCTION

The tower defense game refers to a strategy game that defends one's base camp by building defensive towers on the map to attack and block the enemy's progress.

In the tower defense game, a level consists of three elements, the map path, and the monster sequence, and the defense tower. These three elements all affect the difficulty of the game, which determines the playability of a level. In the traditional production process of a tower defense game, the generation of a tower defense game level will experience background planning, path design, monster sequence design, playability testing, and art realization in the game industry. The complete process needs a large number of participants and the close coordination between multiple departments, this often leads to poor efficiency.

Therefore, we have come up with the idea of automatically generate tower defense game levels by computer algorithms. This includes generation of paths and monster sequence for tower defense game. In addition, the new paths should conform to the necessary features required for the paths in the general tower defense game. For example, there are terrains such as intersections or detours in the paths, so that there will be areas where firepower is concentrated. At the same time, the difficulty of the game level should be moderate, and it needs a rich variety of monsters, so that the newly designed levels can match the original levels in their playability.

The automatic level generation method shown in this paper is based on a classic tower defense game - Kingdom Rush Frontiers(KRF). We developed a KRF game simulator that can run a complete tower defense game with an level profile. The simulator can visually display the levels we generated, which is the basis of our research.

We proposed a method for automatic generation of tower defense game levels. Specifically, our research focuses on two parts: path generation and monster sequence generation. In terms of path generation, we use the generation and connection of multiple arcs to form a path. We used a search algorithm to generate new paths that were very similar to the paths in the original game. In terms of monster sequence generation, we use genetic algorithms to creatively combine human-designed elements with computers to automatically generate monster sequences that make the levels playable.

The rest of the paper is structured as follows. In Section II we provide an introduction to Procedural Content Generation and briefly describe the basic game mechanics of KRF. In Section III we describe the entire process of level generation in detail, which is divided into path generation algorithm and monster sequence generation algorithm. In Section IV we evaluate the automatic generated levels from several aspects. Section V gives a conclusion to our research and discusses for future work.

II. BACKGROUND

A. Basic Game Mechanics of KRF

This section outlines the basic rules of Kingdom Rush: KRF is a stand-alone game. Each level of the game has a pre-set path. After the game starts, there will be monsters constantly invading along the path. Players need to strategically build and upgrade defensive towers to defend against monster intrusions. In the game, players can also release some special skills, such as summoning reinforcements at a certain point. There are four types of defensive towers in the game, each with its own unique features, and players need to spend gold to build and upgrade these towers. At the beginning of the game, the player will receive a certain amount of initial gold. When a monster dies, the player will receive a corresponding bounty. But when a monster successfully escapes, the player will lose some life. When the player's life is reduced zero, the game fails. And if

978-1-5386-6243-4/19/$31.00 ©2019 IEEE

the players life is not zero after the game is over, the game wins.

B. Procedural Content Generation

Procedural Content Generation, or PCG, is a method for creating game content using limited or indirect user input [1]. In other words, PCG refers to computer software that can create game content by itself, or computer software that creates game content with human designers.

Procedural content generation dates back to the early 1980s. One of the early examples in this area is the space trading game *Elite*, it provided a huge environment with eight galaxies, each containing 256 stars. But the entire universe needed only few kilobytes to be store, because the content of each galaxy was generated by procedural approach. *Diablo* [2] is an action role-playing videogame that used procedural generation to generate maps, as well as the type, number and placement of monsters. In the game *Spore*[3], players can design new creatures and these personalized creatures are animated by procedural methods. *Civilization IV* [4] is a turn-based strategy game that can provide a unique gaming experience by generating random maps. *Minecraft* [5] is a popular game where PCG technology is widely used to generate the entire world and its content. *Borderlands* generated three million weapons using a single parameter vector [6]

There are many approaches for procedural content generation. In recent years, Generative-and-test approaches have been intensively investigated in academic PCG research in which evolutionary algorithm or some other random search/optimization algorithm is typically used to search for content of a desired quality. This research area has been recently named Search-Based Procedural Content Generation [8].

Togelius et al.combined procedural content generation principles with an evolutionary algorithm to evolve racing tracks of a simple two-dimensional racing game [7]. This game has been used in a series of experiments to study how to best use evolutionary algorithms to create agents that could play the game well.

C. Evolutionary search algorithms

Evolutionary algorithm is a random search algorithm that is inspired by Darwinian evolution through natural selection. The core idea is to maintain a group of individuals, evaluate them in each generation, and select the most suitable (highest rated) individuals to breed, while the most inappropriate individuals are removed. Therefore, one evolution can be divided into a selection and reproduction phase. In a search-based PCG implementation, a generation of game units may be subject to selection of an evaluation function that ranks them by their performance. Then the selected individuals were mixed (recombined or cross-over) or replicated by small random changes (mutation). Generally, the next generation's fitness is better than the previous generation. Even if the randomly generated first generation performed poorly, this algorithm is still effective because there are always some individuals that are better than others.

III. METHOD

In this chapter, we will describe the automatic generation method of tower defense game levels. The entire method consists of two parts, the path generation and the monster sequence generation.

A. Road generation

The path of the tower defense game KRF we studied has many of its own characteristics. Firstly, compared to the simple combination of straight lines, the KRF path contains more arcs, which makes the KRF path look more natural. Though such irregular path is more playable, it brought us great challenges to automatic design.

Fig. 1. Paths in original levels

To solve this problem, we first proposed a mathematical representation of the path, and we use "smooth" connections of a number of arcs and lines to represent a path. The smooth connection means that each arc is generated along the direction of the end of the previous path. To ensure that the arc is smoothly connected, when generating the next arc, we make sure that the center of curvature of this arc is on a line that is perpendicular to the tangent of the end of the previous path. Based on this mathematical representation, we turn the path generation problem into a geometric problem.

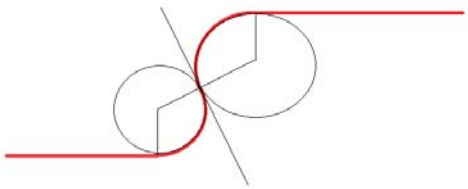

Fig. 2. The geometric representation of the path. The red line is a path generated, this path consist of two lines and two arcs.

We use a search-based algorithm to generate a path, and the algorithm contains three parts, they are exploration, pruning and evaluation.

- *Exploration*. The process of exploration selects curves and lines of arbitrary curvature and arbitrary length. We analyzed several sets of parameters through experiments

and selected the optimal set of parameters for the path generation. Some of the experimental results will be shown in the Section IV.

- *Pruning*. The pruning strategy is a filter to cut the branches in exploration that do not meet the requirements, thereby reducing the number of explorations and improving the efficiency of path generation. For example, there is no closed loop in the tower defense game path. Therefore, in each exploration, when a new path is detected to generate a closed loop, the exploration fails. In addition, we also stipulate that if a legal path cannot be generated after a certain number of attempts, the entire generated path is deleted, and a new path generation is started. Doing so avoids the convergence of the search algorithm to an infeasible situation.
- *Evaluation*. The evaluation strategy is a method that evaluates the rationality of a path after it has been generated to determine whether the path is available. We cut the map into 9 areas, if the number of areas that covered by the path is greater than 6, the currently generated path is considered qualified, otherwise the currently generated path is deleted and re-generated.

In the specific path generation process, we take one path as the main path and the remaining paths as the branches on the main path called sub-path. A complete path generation process includes one primary path generation and several secondary path generations.

The process of generating a path is as follows: First, select a starting point C_0 in the center of the map, and randomly select an initial direction, then randomly add an arc or a straight line l_i to the current path one by one until the boundary is reached. In the process of randomly adding lines, it is detected whether the pruning condition is satisfied and whether it is finished. Puring condition includes closed loop and infeasible situation, for example, if the end point of the last path is in the center of a ring road, the next path will form a loop with great probility, it will waste much time here. If the pruning condition is met, the last added line will be deleted and regenerated from end of the previous path. If the end condition is met, the current path is passed to the evaluation strategy. If it is usable, it will be retained, otherwise we will start to regenerate a new path. The first generated path R acts as the primary path, and several secondary paths will extend from it to form the final mapping M. We can set N_r to decide how many auxiliary paths do we need.

Algorithm 1 Road generation pseudocode

$N_r \leftarrow$ number of roads to draw (1 to 3)
initialize road map M
current junction point C_0 = random point near center of map
while $N_r > 0$ **do**
  $i \leftarrow 0$
  initialize new road R
  while edge of map unreached **do**
    draw random straight line or arc l_i from current junction point C_i (make sure the tangent of l_i 's start point is coincide with that of l_{i-1}'s end point)
    if new line l_i does not intersect existing ones **then**
      $C_{i+1} \leftarrow l_i$'s end point
      add l_i to R
      $i \leftarrow i + 1$
    end if
  end while
  add R to M
  current junction point $C_0 \leftarrow$ random point on existing road
  $N_r \leftarrow N_r - 1$
end while
return M

B. Monster Sequence Generation

In the tower defense game, the monster sequence defines monster's number and type on each path at each time point. The configuration of the monster determines the difficulty and playability of the entire level. A too difficult level will hit the player's interest, and a too simple level will make the player feel boring. Therefore, the design of the monster is a crucial part of the tower defense game.

A good monster design not only needs moderate difficulty, but also require using a variety of monsters to form rich and interesting combinations, so as to continuously attract players. We proposed an evolutionary algorithm that can automatically generate such monster sequence. We first show our genetic representation and how to convert a gene into a complete configuration, then we introduce fitness function and the entire procedure.

1) Genetic representation: There are many modes among monsters in the original KRF game, for example, some monsters are special monsters that can add buff to their surrounding monsters. A buff monster surrounding with several ordinary monsters is just a mode designed by human designers. We extracted all such modes from the original game and used such a cluster of monsters as a basic gene in the genetic algorithm. The method of gene extraction is as follows: we first select a threshold T, if the time interval between two monsters appearing on a path is less than this threshold, they are regarded as monsters of the same gene, otherwise they will be classified into two genes. Here T is a hyperparameter, and the value of T affects the granularity of gene selection. We chose T to be 300 frames. A total of 560 genes was extracted to form the gene pool of the genetic algorithm. Such genetic representation has three advantages. The first is that it can utilize artificially designed elements such as formations and therefore obtain strong scalability. The second is that it made our algorithm independent of a specific tower defense game. The third is that it could accelerate the searching prograss. When we want to generate content of another tower defense game, we only need to replace the genes with the monster modes of that game.

Fig. 3. Monster modes in the game

There are several waves of monsters in a level. We use a fixed-length chromosome to represent a sequence of monsters on a road of a particular wave. Here each gene is a numeric ID representing a particular monster mode. It will be converted into complete representation when needed. A gene with an ID of 0 indicates a null gene, and the remaining genes are effective genes. The introduction of null genes is to enable the genetic algorithm to change the total length of a certain wave of monsters by controlling the number of effective genes in the chromosome.

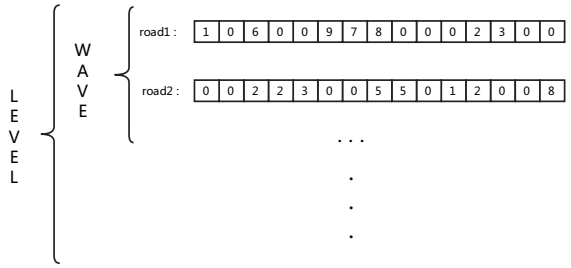

Fig. 4. The genetic representation of a level's monster sequence with max length of 15. Each gene number represent a mode of monsters. A level may have n waves each wave may contains m roads, each road is represented as a sequence of genes.

2) Gene expression: Gene expression is the process that converts a level's genetic representation (shown in Fig 4)into a configuration file. In this game, a road consists of three sub-paths: left, center, and right. When we extract genes from the original levels to construct a gene pool, we keep the monsters' relative distribution on the three sub-roads. When we express a level's gene, we will first find the modes corresponding to that gene, and then write these modes to a new road configuration without changing monsters' relative positions. The relative time of a mode's monsters will be converted into an absolute time in the new level.

3) Fitness function: Fitness function is used to evaluate the generated configurations, and it takes a level's phenotype as input.The phenotype of a level refers to a complete configuration expressed by its genotype. In a particular generation, we select one original level as a $reference$, while call the generated new level a $candidate$. The similarity of the $reference$ and $candidate$ was taken as our fitness function. To construct this fitness function, we used three sets of values. The three sets of values are showed as follows.

1) Monster count, N_c and N_r are lists representing the number of monsters per wave in a candidate level and reference level respectively. N_{ci} and N_{ri} are the number of monsters in the ith wave of the given level.
2) Type numbers of monsters, T_c and T_r, are two list of type numbers of candidate level and reference level. Where T_{ci} and T_{ri} are the types number of monsters that appeared in the ith wave.
3) Total health point, H_c and H_r, represents the total sum of health point for all monsters in a candidate or reference level. H_{ci} and H_{ri} are the total health point of all monsters in the ith wave.

We used six functions to measure the similarity between $candidate$ and $reference$. $f1, f3, f5$ are designed by Manhattan Distance, they make $candidate$ and $reference$ consistent in magnitude. And $f2, f6$ are designed by cosine similarity, they keep $candidate$ and $reference$ consistent in their changing trends. $f4$ try to make $candidate$ has a similar monster type numbers with $reference$.

$$f1 = \sum_{j=1}^{n} |N_{ci} - N_{ri}| \tag{1}$$

$$f2 = (1 - \frac{\sum_{j=1}^{n} N_{ci} N_{ri}}{\sqrt{\sum_{j=1}^{n} N_{ci}^2} \sqrt{\sum_{j=1}^{n} N_{ri}^2}}) \tag{2}$$

$$f3 = \sum_{j=1}^{n} |T_{ci} - T_{ri}| \tag{3}$$

$$f4 = \sum_{j=1}^{n} T_{ci} - \sum_{j=1}^{n} T_{ri} \tag{4}$$

$$f5 = \sum_{j=1}^{n} |H_{ci} - H_{ri}| \tag{5}$$

$$f6 = (1 - \frac{\sum_{j=1}^{n} H_{ci} H_{ri}}{\sqrt{\sum_{j=1}^{n} H_{ci}^2} \sqrt{\sum_{j=1}^{n} H_{ri}^2}}) \tag{6}$$

Overall fitness value is a linear combination of the six functions above. Where θ_j is a weight parameter.

$$fitness(x) = \sum_{j=1}^{6} \theta_j f_j \tag{7}$$

4) Generation: The evolutionary algorithm calculates the optimal fitness F_i for each generation P_i. If F_i has not changed for 10 consecutive generations, or F_i reached a threshold F_e, the algorithm will end. Otherwise, the top 20% individuals will be selected as the parent of the next generation, they will cross over to produce 800 offspring. These 800 descendants form the next generation P_{j+1} with their parents. Beside, we also choose 30% individuals of this generation to mutate, then the fitness function of the P_{j+1} is calculated to be F_{j+1}. Repeat the above procedure until F_{j+1} converges.

Algorithm 2 Monster sequence generation pseudocode

function EVOLVE($candidate, reference$)
  Randomly initialize 1000 individuals as P_1
  for each $i \in P_1$ **do**
    Get fitness F_i
  end for
  Get best fitness of P_1 as F_1
  $j = 1$
  while $F_j \neq F_{j-10}$ **or** $F_j > F_e$ **do**
    $j = j + 1$
    Pick the top 200 individuals as parents M_{j+1}
    Add M_j to P_{j+1}
    for $k = 1, 800$ **do**
      Randomly pick two parents from M_{j+1} and crossover to produce a child C
      Add C to P_{j+1}
    end for
    mutate 30% individuals of P_{j+1} randomly
    Get best fitness of P_j as F_{j+1}
  end while
  $F_{best} = F_j$
  Select the corresponding individual I_{best} of F_{best}
  Express I_{best} to generate a new monster sequence file
end function

IV. Results

A. Road generation

Here we show the influence of different parameters on the path style. The main parameters influencing paths generated are curve ratio, angle range of arcs, radius range of curvature of arcs and number of sub-paths. Where curve ratio means the ratio of curves to the straight lines that was used to generating a new road.

Fig. 5. Paths generated with curve ratio = 0.7, angle $\in [70, 180]$, radius $\in [80, 120]$, sub-paths number = 2

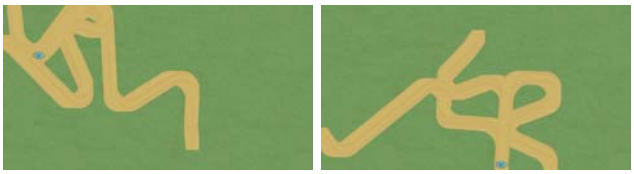

Fig. 6. Paths generated with curve ratio = 0.7, angle $\in [70, 180]$, radius $\in [40, 80]$, sub-paths number = 2

Fig. 7. Paths generated with curve ratio = 0.7, angle range $\in [70, 180]$, radius $\in [120, 200]$, sub-paths number = 2

Fig. 8. Paths generated with curve ratio = 0.7, angle range $\in [30, 120]$, radius $\in [120, 200]$, sub-paths number = 2

Fig. 9. Paths generated with curve ratio = 0.4, angle range $\in [70, 180]$, radius $\in [80, 120]$, sub-paths number = 2

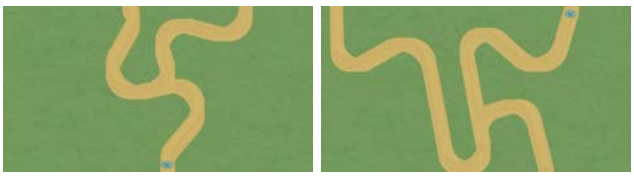

Fig. 10. Paths generated with curve ratio = 0.7, angle $\in [70, 180]$, radius $\in [80, 120]$, sub-paths number = 1

Different combinations of parameters can generate a variety of paths with different styles. By comparison, it can be seen that the paths in Fig. 5 looks better. When the radius range is too small, the generated path looks too compact (Fig 6). And when the radius is too large, the generated curve also becomes too large, which makes it not similar to the original one (Fig 7). When we reduce the range of angles, the paths will lack big turns (Fig 8). When the radius and angle are the same but

the ratio becomes smaller, there will be more straight lines in the path, which makes our path style more blunt (Fig 9). The number of sub-paths determines the complexity of the map, fewer sub-paths tend to generate simple levels (Fig 10).

The algorithm runs on a single 6th-gen i7 processor. After 100 runs, it takes an average of 3 seconds to generate a new map.

B. Monster Generation

In the tower defense game, when other factors are constant, the total blood volume of a wave's all monsters can roughly represent the difficulty of this wave. In order to prove that our algorithm can generate new levels with moderate difficulty, we take an original level(level 8) of the game as a $reference$ of our genetic algorithm's fitness function to generate a new level ,and then we compare the distribution of the new level and the original level on the monster's total blood volume of every wave. As can be seen from the Fig 11, the newly generated level is very close to the original level in difficulty, which is not surprising, because the indicator-total hp itself is part of the fitness function in the genetic algorithm. Enemy wave configuration takes average 76 minutes on a single core.

From the Fig 11 we can see that the difficulty does not increase linearly as we may think. There will always be an easy wave of monsters after several waves that have increasing difficulty. This situation is often described as psychological flow [9]. By ensuring that the difficulty of the game matches the level of the player, and by increasing the difficulty at an appropriate rate over time, the game can provide a pleasant experience termed "flow". If the difficulty has been increasing all the time, the player will be exhausted. It is this combination of tension and relaxation that makes the level of a tower defense game attractive.

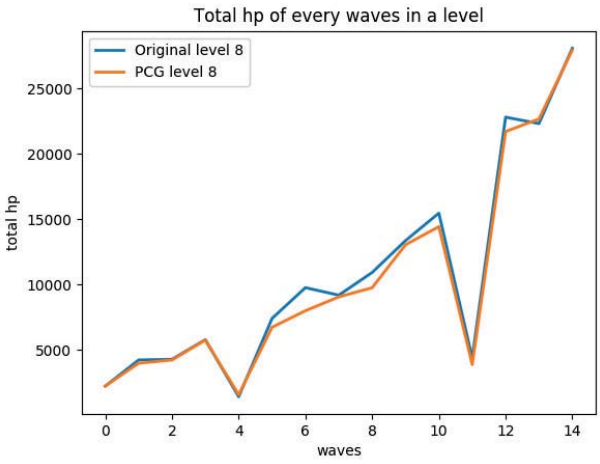

Fig. 11. Comparison of the two levels's total enemy hp in every waves

Previously, we took the data of the original game as a $reference$ for the monster sequence generation algorithm to generate a new level. In this way we generated a level that is similar to an original level. However, our goal is not only to generate a similar level of the original level but also to create a new one. To do so, we can manually design a difficulty curve, and take this curve as the $reference$ of our generation algorithm. Then we will get a new level that matches the difficulty curve we designed.

Figure 12 is the difficulty curve of a level that we designed by ourself, as can be seen, the difficulty of this level satisfies the psychological flow.

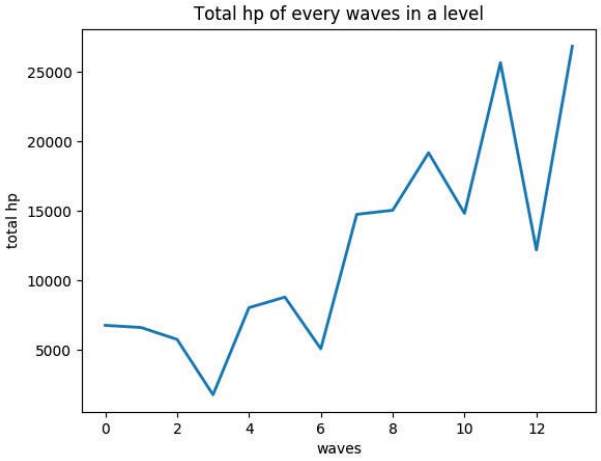

Fig. 12. The difficulty curve of a new level that uses a manually designed $reference$

To further verify the effectiveness of the monster sequence PCG, we designed a simple Turing test. We show six levels to the participants at the same time, three of which are original levels and the other three are PCG-generated. To exclude the effects of the map, all of the levels use the same map path. Participants are required to rate the monster sequence of the video based on the monsters that appear in the video. The scoring criteria range from 1 to 5 (1 = certainly PCG, 2 = may be PCG, 3 = unsure, 4 = may be artificial, 5 = certainly artificial). After scoring all six videos, we will use the following formula to provide participants with a final score.

$$\text{Final Score} = \sum_{i=1}^{6} S_i, \text{ where } S_i = \begin{cases} 6 - \text{rating}_i & \text{if PCG} \\ \text{rating}_i & \text{otherwise} \end{cases}$$

Participant	S_1	S_2	S_3	S_4	S_5	S_6	Final Score
1	1	5	5	1	1	1	14
2	5	5	1	5	3	1	20
3	1	5	1	5	5	1	18
4	1	1	5	5	1	1	14
5	5	5	1	1	5	5	22
6	5	4	5	5	5	5	29
7	1	1	5	5	5	5	22
8	1	2	2	2	2	3	12
9	1	1	2	4	2	4	14
10	1	1	4	4	1	1	12
11	1	3	1	2	3	2	12
12	5	4	4	5	4	5	27
13	2	5	5	1	1	3	17

TABLE I
SCORES FROM TURING TEST FOR HUMAN-DESIGNED AND PCG MONSTER SEQUENCES

In this experiment, the mean of the participants' score is 17.9, which is close to the expectation of random selection that is 18. This shows that these human participants can't decide if one test level is machine generated level or a level from original game. However, participants 6 and 12, both of whom are experienced KRF players, still found some flows which help them correctly distinguished human-designed and PCG levels. They pointed out that the time interval between each wave of monsters in PCG levels lack change. This is because we simply used a random number to generate that time interval in our current method. We will study the influence of this time interval and try to find a batter method in our future work.

Finally, we combine the path generation algorithm with the monster generation algorithm to achieve automatic generation of tower defense game levels. Generating one level takes on average 76 minutes using a single 6th-gen i7 processor. But it can be accelerated by parallel computing and it takes 20 minutes when using four cores.

V. DISCUSSION AND CONCLUSIONS

In this paper, we proposed a method for automatic generation of tower defense game levels. We first proposed a representation of the path, and then invented a path generation algorithm in the base of this representation. After testing, the algorithm achieved the desired goal and can generate satisfactory path maps. Then we used evolutionary search algorithms combined with manual design to construct a monster sequence generation algorithm, and the new level generated by our algorithm passed our turing test, which proved most people could not accurately distinguish the new level from the original level. Our current tower defense game auto-generation method is able to generate a large number of map paths and monster sequences. Our research has automated the production of levels for tower defense game, greatly improving the production efficiency of this type of game.

However, there is still a long way to go before the current level becomes a mature tower defense game level. Firstly, we do not have a quantitative standard to measure the automatically generated path. We still need to manually identify whether a path is qualified. Besides, when we use the monster sequence data of the original game as a $reference$ of the generation algorithm, we can ensure that the newly generated monster sequence has similar playability to the original game. But when we use our own designed difficulty curve to generate a new level, we can't guarantee that this new level is fully playable. In our future work, we want to develop a method that can measure the generated paths. And we will try to use the reinforcement learning method to develop an automatic verification algorithm for the tower defense game level, and use this to assist the monster sequence generation algorithm to generate a more stable and effective monster sequence.

ACKNOWLEDGMENT

This work was supported by LevelupAI (http://www.levelup.ai/). We thank Li Chaoran, Shen Yiming for many helpful discussions and suggestions on the paper. We also thank LI Yue for setting up the environment of KRF simulator.

REFERENCES

[1] Togelius, Julian, Kastbjerg, et al. What is procedural content generation?: Mario on the borderline[C]// ACM, 2011:1-6.
[2] Blizzard North, 1997, diablo, Blizzard Entertainment, Ubisoft and Electronic Arts.
[3] Maxis: (2008). Spore, Electronic Arts
[4] Firaxis Games: (2005). Civilization IV, 2K Games & Aspyr
[5] Mojang: (2011). Minecraft, Mojang and Microsoft Studios
[6] R. Frushtick, Borderlands Has 3,166,880 Different Weapons, July 2009, http://multiplayerblog.mtv.com/2009/07/28.
[7] Togelius J, Nardi R D, Lucas S M. Towards automatic personalised content creation for racing games[C]// IEEE Symposium on Computational Intelligence and Games. IEEE Computer Society, 2007:252-259.
[8] Togelius J, Yannakakis G N, Stanley K O, et al. Search-Based Procedural Content Generation[C]// Applications of Evolutionary Computation, Evoapplicatons 2010: Evocomplex, Evogames, Evoiasp, Evointelligence, Evonum, and Evostoc, Istanbul, Turkey, April 7-9, 2010, Proceedings. DBLP, 2010:141-150.
[9] M. Csikszentmihalyi, *Flow and the Foundations of Positive Psychology*, Springer, 2014, p. 150.