A Multi-population Genetic Algorithm for Procedural Generation of Levels for Platform Games.

Lucas Ferreira
Institute of Mathematics and Computer Science
University of Sao Paulo
Sao Carlos, Brazil
lucasnfe@icmc.usp.br

Leonardo Pereira
Institute of Mathematics and Computer Science
University of Sao Paulo
Sao Carlos, Brazil
leonardop@usp.br

Claudio Toledo
Institute of Mathematics and Computer Science
University of Sao Paulo
Sao Carlos, Brazil
claudio@icmc.usp.br

ABSTRACT

This paper presents a multi-population genetic algorithm for procedural generation of levels for platform games such as Super Mario Bros (SMB). The algorithm evolves four elements of the game during its generations: terrain, enemies, coins and blocks. Each element has its own codification, population and fitness function. At the end of the evolution, the best four elements are combined to construct the level. The method has as input a vector of parameters to configure the characteristics of each element. Experiments were made to evaluate the capability of the method in generating interesting levels. Results showed the method can be controlled to generate different types of levels.

Categories and Subject Descriptors

I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems—*Games*; K.8 [**Personal Computing**]: [Games]

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Procedural content generation, games, genetic algorithm, platform, Super Mario Bros

1. INTRODUCTION

Automatic generation of game content has been used for several purposes inside the game industry. One of the main purposes is to use it as a tool to generate a huge amount of content as references for human designers. It can also be used in real time, inside the games, to generate dynamic content for players. Recently, procedural content generation (PCG) has becoming an emerging area of research in game design. Several PCG methods has been proposed to generate different content types like tracks for racing games [1], maps [4], levels [5] and even whole games [2].

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.
ACM 978-1-4503-2881-4/14/07.
http://dx.doi.org/10.1145/2598394.2598489.

A platform game involves guiding an avatar to jump between suspended platforms, over obstacles, or both to advance the game. In the Super Mario Bros (SMB), the main character must jump between platforms and kill enemies to save the princess Peach. There are several enemy types and the platforms are made from blocks, hills and pipes. The game also has collectable items such as coins and powerups.

In this paper we propose a search-based method for level generation in platform games like SMB. The method uses a genetic algorithm to evolve separately four game aspects: ground, blocks, enemies and coins. Each element inside the game has its own purpose, so each of them has its own population, fitness function and genetic operators. To construct a level, the algorithm joins the best individual from each population in the end of the last generation. All the elements are evolved using a *direct* fitness function. It means there is no need to simulate the game during its evaluation process.

The goal of the developed level generator is to design several levels with gameplay characteristics specified by the user. So, it has a parameters vector to configure the characteristics of each game element. We proposed several experiments to evaluate the capacity of the method in generating interesting levels. Results showed it is possible to configure the method for generating levels with different gameplay characteristics.

2. METHOD

The proposed genetic algorithm evolves separately the four game elements, so it keeps four populations during the evolutionary process. Each population has individuals representing an specific type of element. In the end of the evolution, the best individuals of the populations are used to form the level.

2.1 Level Representation

Each level element (ground, blocks, enemies and coins) is represented by an integer vector, where the elements codify a different data type. Thus, a level is a combination of four vectors of this type. In the ground's vector, each element represents a height $0 \leq h \leq 15$ of a ground segment in the level, from the bottom to the top of the screen. The height $h = 0$ means there is a gap in that position. The height limit 15 is given by the height of the game resolution.

The blocks vector has a fixed size given by the level width l_w and each element represents a block type. The index in the vector is the position where the block will be placed in

the level. Since there are only four types of blocks, each element b is inside the interval $0 \leq b \leq 4$. An element $b = 0$ means there is no block in that position. The elements from 1 to 4 represent the empty block, the powerup block, the coin block and the rock block, respectively.

The blocks vector does not have information about the height of the blocks in the level. As a simplification, the blocks are placed over the ground in a height where Mario can hit them. This height h_b is limited by the interval $1 \leq b_{types} \leq 4$. The 4 limit is based on the Mario maximum jump height. It is important to highlight the case the height is equal to 1, because the blocks will only hit if the Mario is in a small form.

The representations for the enemies and the coins are similar to the blocks' representation. Both have a fixed size vector based on the level width. In the enemies' vector each element represents an enemy type. It does not carry any information about the height where the enemies will be placed. Since all the enemies do not fly, all the enemies are placed over the ground. In the coins' vector, each integer represents the height from the ground which the coin will be placed.

2.2 Fitness Function

The ground is evaluated by the concept of *entropy* [3]. Entropy is a measure of unpredictability of information content, but it is used here to measure the ground unpredictability. The entropy function is applied to ground parts with size $1 \leq g_{part} \leq l_w$. g_{part} is a parameter of the method and it is useful to control how "unpredictable" the ground will be. The idea of separating the ground in parts to calculate the entropy is to avoid having either a flat shape (minimal entropy) in the whole ground or a too uncertain terrain (maximal entropy). The other game components use the *sparseness* [2] concept in their fitness function. An array is sparse if the average distance between two elements is high.

The method uses parameters to control the sparseness and the entropy of the elements. This means the fitness functions are normalized and they are actually calculated by the distance from the fitness of the individual to the desired entropy/sparseness values. The objective of the algorithm is to minimize this distances.

3. EXPERIMENTATION AND RESULTS

We conducted experiments to test the method's ability in designing SMB levels with different gameplay characteristics. To generate different levels we must use different parameters vectors as input for the method. All the parameter of the method are summarized in Table 1.

In our first experiment, we wanted to generate levels similar to initials levels of the original SMB. These levels usually are responsible to teach the player the core mechanic of the game. Thus, it must teach the player how to walk and run, how to jump and teach that he can die, if an enemy touches him. This is achieved with placing the Mario in the beginning of the level over a flat ground (without gaps) with at least one "question" block, the level must have also an enemy coming in the ground from the right of the screen.

The Figure 1 shows the first segment of a level generated by our algorithm. This segment has similar characteristics to initials levels of the original SMB. The values for the parameters used in this experiment are defined in the Table 1.

Parameter	**Definition**	Value
l_w	Level width	200
$0 \leq g_e \leq 1$	Desired ground entropy	0
$1 \leq g_{min} \leq 3$	Ground parts	1
$1 \leq g_{part} \leq l_w$	Entropy parts	200
$0 \leq g_{h_max} \leq 15$	Ground maximum height	2
$0 \leq b_s \leq 1$	Desired blocks sparseness	1
$1 \leq b_{types} \leq 4$	Blocks types	3
$1 \leq b_{part} \leq l_w$	Blocks sparseness parts	10
$0 \leq e_s \leq 1$	Enemies sparseness	0
$0 \leq e_{types} \leq 4$	Enemies types	2
$1 \leq e_{part} \leq l_w$	Enemies sparseness parts	20
$0 \leq c_s \leq 1$	Coins sparseness	0.5
$0 \leq c_{h_max} \leq 10$	Coins maximum height	2
c_{part}	Coins sparseness parts	10

Table 1: All the parameters available in the method.

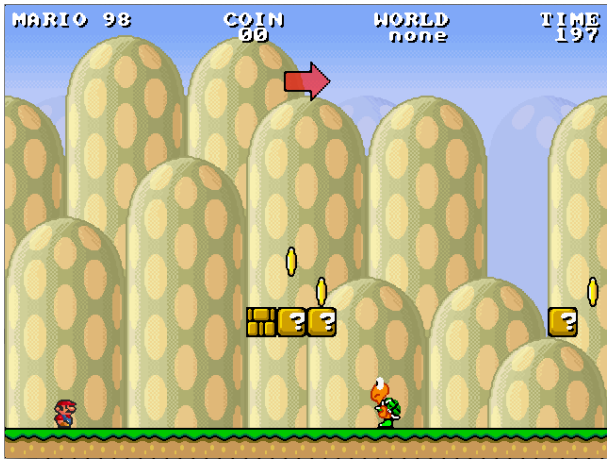

Figure 1: The first segment of a generated level.

4. REFERENCES

[1] L. Cardamone, D. Loiacono, and P. L. Lanzi. Interactive evolution for the procedural generation of tracks in a high-end racing game. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 395–402, New York, NY, USA, 2011. ACM.

[2] M. Cook and S. Colton. Multi-faceted evolution of simple arcade games. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 289–296, 2011.

[3] C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.

[4] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelback, and G. Yannakakis. Multiobjective exploration of the starcraft map space. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 265–272, 2010.

[5] V. Valtchanov and J. A. Brown. Evolving dungeon crawler levels with relative placement. In *ACM International Conference Proceeding Series*, pages 27–35, School of Computer Science, University of Guelph, Guelph, ON N1G 2W1, Canada, June 2012. ACM Press.