Report on implementing techniques described in ”Computational Intelligence and Tower Defence Games” by Julian Togelius, Elvis Alistar, and Robert Pieter van Leeuwen

Timur Yakupov, t.yakupov@innopolis.ru

I. Introduction

”Computational Intelligence and Tower Defence Games” is a paper written by by Julian Togelius, Elvis Alistar, and Robert Pieter van Leeuwen where described how Tower Defence (just TD from here) games can provide an important test-bed for the often under-represented casual games research area. There also described a prototype TD game based on experience driven procedural content generation, some elements of which was implemented and tested.

II. Necessary materials

Decision to use a game engine was made due to the lack of experience in game development. Unity3D was chosen to work with for its simplicity and its documentation which is easy to understand. Another purpose for using Unity3D is that there are a lot of guides available in the net.

III. Methods

There are two things that are generated described in this paper: towers and waves of enemies. The descriptions of methods below are almost completely taken from the paper.

A. Tower Evolution

The towers are evolved using a genetic algorithm, and each individual uses chromosomes with 11 genes, 3 for the basic Range, Speed and Damage properties common and necessary for all towers, and 8 for the optional properties that add bonuses to the towers.

Each gene contains a float value between 0 and 1, which is translated into the specified range.

Formulas for defining any parameter of a tower has the form:
geneValue * (maxRangeValue - minRangeValue) + minRangeValue (example for range)

When the game starts the algorithm creates a population of 16 individuals. Two of these individuals have their genes initialized with completely random values, while the other 14 individuals have their genes initialized with values between 0 and 0.2. When the player sells a tower or destroys one in the tower selection grid, a new tower needs to be generated. The new tower is generated as the first child from the one point crossover between two random individuals in the population. Every 15th tower that the game generates will be based on an individual with completely random values in its genes.

B. Creep Evolution

Each creep wave is defined by the number of creeps, speed of the creeps, hit points, armor and spawn delay. When evolving a wave, the chromosome contains four genes in the range 01; these are used to calculate the creep wave in combination with a number of points. The number of points act as a strength parameter for the creep wave, and can be used to adapt the difficulty of a wave without changing its composition.

The wave is calculated from the genes as follows:

- Amount of creeps in a wave: (gene 0) * 20 + 1
- Points per creep: (Total number of points) / (Amount of creeps + 2)
- Total weighing factor: (armor gene) + (speed gene) + (hitpoints gene)
- Speed: (speed gene)/(total weighing factor) * (Total number of points) * (Speed modifier)
- Armor: (armor gene)/(total weighing factor) * (points per creep) * (Armor modifier)
- Hitpoints: (hitpoints gene)/(total weighing factor) * (points per creep) * (hitpoints modifier)

The fitness calculation of a creep wave is simulation based: the wave is played out against the players previous tower configuration (on the previous level) and the fitness value is calculated as the number of creeps that made it through to the end of the path, or the distance the creeps managed to travel along the path if no creeps made it to the end.

A good wave configuration is in this context one where a smaller number of points is needed to break through the defences and reach fitness 1 (one creep gets to the end of the path). The number of points needed is calculated through playing the wave starting with 2000 points, and if it does not reach at least fitness 1 increasing the fitness with 5 percents cumulatively until that fitness is reached.

IV. Simplifications

Since implementation of whole prototype game requires more experience in game development and much more time, only some elements were implemented:

- tower evolution with fixed number of tower population
- creep wave points increment until reaching fitness 1
- saving waves configurations and points in order to evolve a resulting wave from them

- evolving a resulting wave

So, this project is only visualising how the wave which is to be given to the player to deal with next round is evolving.

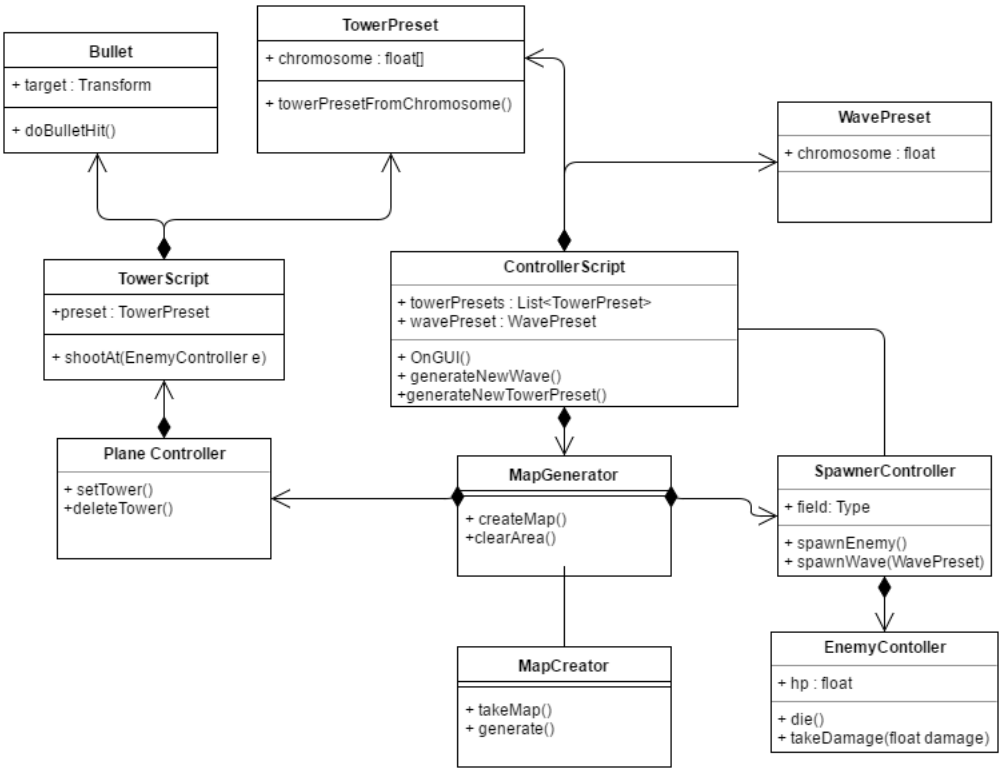
Not all the parameters of towers affects the game, only damage and shooting speed and radius.

It's also not described in the paper how exactly parameters should affect the game, so improvisation takes place. Some parameters were affected by some modifiers. For example, if speed is calculated like it is described in paper, enemies' speed is too fast that player sees it as teleportation.

V. Implementation

Implementation consists of 10 classes:

- MapCreator - responsible for generating a 2d array map, works similar to Look Ahead Digger
- MapGenerator - responsible for making a 3D map from 2d array and keeping all 3D objects
- ControllerScript - main game controller, provides all evolution, scoring and other game behaviour
- Tower preset - responsible for keeping tower's parameters
- Wave preset - responsible for keeping waves' parameters
- EnemyController - responsible for enemy movement and notifying ControllerScript of deaths
- SpawnerController - responsible for spawning enemies
- PlaneController - class for representing plane and its behaviour, used to select, deselect a plane and to set or destroy a tower on it
- TowerScript - responsible for tower behaviour (detecting enemies, shooting)
- Bullet - responsible for bullet behaviour



The project itself looks like it is a (hardly) playable TD game but actually it is just a visualisation of how next wave should be evolved. At start there is a generated map and a set of 16 generated towers. After placing a number of towers (or maybe all) lets consider this map as a previous player's tower configuration.

Each testing wave chromosome is generated randomly and initial points = 2000. Enemies are trying to reach the end and increasing points each time the wave fails to do so. If at least one enemy breaks through to the end, the wave configuration and number of points must be saved and the new testing wave must be generated. In this implementation all data is also saved to a .txt file. After several configurations saved, it is possible to evolve a resulting wave which is actually the wave that should be given to a player next round. This resulting wave is also saved in a .txt file and for now it's the only possible way to see the its configuration, it can not be tested in gameplay.

The final wave can be generated manually by pressing the corresponding button.

Since it is not clear from the paper how resulting wave must be generated, it generates as follows: Taking the average of points in testing waves. Each i-th gene of a resulting wave calculates as maximum of all i-th genes of all testing waves minus random value in range from 0 to average value of i-th gene within all testing waves.

VI. Results

It is hard to say if implementation is done the way it was supposed in paper, but wave reaches the end very quick (within 6-7 iterations of points increment). The final wave generates correctly based on testing waves simulations.

Here is an example of output to .exe file:
Chromosome: 0.42, 0.98, 0.94, 0.82, points: 2100
Chromosome: 0.27, 0.39, 0.23, 0.02, points: 4365.749
Chromosome: 0.74, 0.58, 0.87, 0.90, points: 2000
FinalChromosome: 0.67, 0.50, 0.60, 0.75, points: 2821.916

Project still requires a lot of work and may contain some bugs. It also has primitive 3D models for visualisation and user un-friendly interface, but it is still playable.

Screen shot:

VII. Taxonomy

New content (towers, waves) is generated during game play, so it is **online**. Player can't play without enemies and towers, so it is **necessary**. There are a lot of random elements, so it is **stochastic**. The initial generation requires no parameters, so it is a **random seed**. At every generation of a test wave there is a checking if enemy has reached the end, so it is **generate and test**.

VIII. Conclusion

This project proves that evolutionary algorithms can be successfully used for generating content in TD games. It also was a good experience of using genetic algorithms, procedural generation and game development in general.