Application of Genetic Algorithms for Game Balancing

Zach Young, Sushil Louis
Department of Computer Science and Engineering
University of Nevada, Reno
Reno, NV United States
zachidk@gmail.com, sushil@cse.unr.edu

***Abstract*— In this work the problem of game balance will be explored by applying a genetic algorithm to a tower defense style game. Traditionally games must be balanced through extensive human testing which this work aims to replace. By using a simple representation binary encoded strings referring to a type of tower and each location in the chromosome referring to a possible tower location in the game, the search space gets exponentially big. This makes approaches such as an exhaustive search impractical. The operators used are single point crossover with bit flip mutation and elitism. In this application fitness is determined by minimizing the total cost of towers while maintaining a health greater than zero. The expected results are that the GA will be able to solve for tower orientations that can be used to determine game balance through tower balance, level design, and resources required for a level. The simulation will be run and visualized using unity. Additional work can be done by wrapping this GA in another GA that solves for optimal game balance parameters.**

***Index Terms*—Genetic Algorithm**

I. Introduction

Game balance is one of the more difficult tasks when developing a game. It is a task that is not necessarily straight forward for deciding what is and isn't balanced and is even harder to test the current balance. The typical means of testing the balance of a game is to have many individuals play the game which takes an extremely large number of man hours which is very costly. In general even after all the testing, games today are often unbalanced and requiring balance updates after it has been released to the public. Finding a means of automating a games balance so that no one strategy is better than the others is thus highly beneficial.

To solve the problem of game balance a simple Genetic Algorithm (GA) is used. This is based on the original work by Holland [1]. This paper aims to test the viability of a GA for the purposes of balancing a game. To validate the results produced by the GA a simple hill climber is compared against. This paper specifically will look at game balance as it relates to a Tower Defense style of game. Tower Defense is a game in which you must defend a home base from numerous attackers through placing defensive structure or towers along the path of attack. You are given a limited number of resources to build a range of towers with different costs as well as advantages and disadvantages. For example one tower might be better against aerial attackers than another. There are many different types of balance approaches you can take in designing a tower defense game but this paper will be looking at three in particular. First uniqueness of towers in that the solution to a given level utilizes a range of tower types rather than just a single type of tower for each tower used. Secondly resources required for a given level in that how many towers and their associated costs are needed to complete a given level. If a GA is not able to find any solution to the level then it is likely too difficult. Lastly map design is evaluated by looking at optimal tower placement locations in the GA solutions. The GA for this problem is represented as a binary string of length 2 times the amount of tower locations for 3 tower types. For crossover the operator of choice is uniform cross over and for mutation the operator will be swap mutation. Further information is presented in future sections.

The results found aren't particularly conducive to being able to determine the balance of the tower defense game proposed but they show promising initial results that could later be used to effectively determine the balance of the game. In particular while a hill climber is able to currently produce a higher fitness in a shorter period of time the GA has benefits that exceed the pitfalls of the hill climber.

The organization of the paper is as follows. In Section II similar work is mentioned and compared to the work proposed here. Section III gives a brief overview of the project. Section IV details the results found and discussion. Section V concludes the paper and discusses future work.

II. Prior Work

A genetic algorithm is applied in a tower defense manner in the work presented in [2]. In their work they look at the effectiveness of a GA compared to a Particle Swarm Optimization in their own tower defense simulation. This work only uses a single type of tower and attempts to solve for optimal tower positions for minimizing the amount of attackers that get to the base. They found that the two algorithms produced similar results but what this work doesn't address is balancing for the tower defense game. The work presented in this paper also solves for optimal tower placement but it offers a more in depth tower defense simulation that can be used to determine balance factors.

A paper that also implements the use of a genetic algorithm for tower defense games is [3]. In their work they use a GA to tune the weights of two different neural networks, specifically

Feed-forward (FFNN) and Elman Recurrent (ERNN). One issue with this paper is it mainly compares the FFNN to the ERNN rather than looking at information about game balance. Another issue is that neural networks are generally hard to understand so it would be hard to get automated balance information from them such as what will be done later in this paper.

Another paper that is similar in nature is that proposed in [4]. In this work they apply a GA for game balancing purposes to a capture-the-flag style game. They argue that ”balanced games are more fun and provide a more interesting strategy space for players to explore.” This provides evidence that the work presented in this paper is useful. Furthermore they use coevolution to balance the game so that one strategy doesn’t dominate another which is part of what is being evaluated as it applies to a tower defense style game presented in this work. In future work coevolution is something that will be looked at as a means of level design where towers and attackers are coevolved.

III. Methodology

A. Game Balance

For game balance purposes the balance is split into three areas. These three categories are optimal tower placement P_{max}, tower balance B_t for an indiviudual tower t, and required resources R. Where

$$P(Loc) = \frac{1}{\#of Runs} \sum_{i=1}^{\#of Runs} TowerCost_i(Loc), \tag{1}$$

$$P_{max} = max_{location}(P), \tag{2}$$

$$B_t = \frac{\text{\# of towers of type t}}{\text{\# of expected towers}}, \tag{3}$$

$$R = \sum_{Loc=1}^{\#of Locations} TowerCost(Loc). \tag{4}$$

The optimal tower placement and tower balance metrics were chosen because they are easy to validate and the required resources can be validated by comparing to a hill climber. The game world was designed in a manner where the optimal tower locations are known. The tower parameters can be manipulated to be particularly weak or strong to evaluate if it is able to find weak or strong towers.

B. Simulation

The Tower Defense game was simulated using the Unity Game Engine [5]. Specifically by using Unity’s Tower Defense Template [6]. This template provides a base Tower Defense Game with all game logic already developed so that the GA implementation could be focused on. The simulation was ran using an increased speed to make evaluations faster. However, this increased speed as well as some non-deterministic actions in the game result in noise where a solution that works for one evaluation might fail for another. Using this template a level was developed in a manner where there are clear optimal tower locations. Fig. 1 shows this level design. Locations 6,15,18, and 29 would be the optimal locations due to being on inside corners since they have the largest amount of tiles they can reach. Alternatively locations 9,12,21, and 26 are sub-optimal due to being on outside corners with minimal amount of tiles it can reach. Locations 1, 2, 33, and 34 suffer similarly.

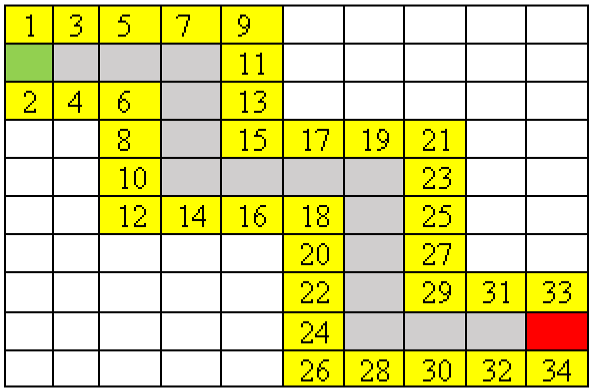
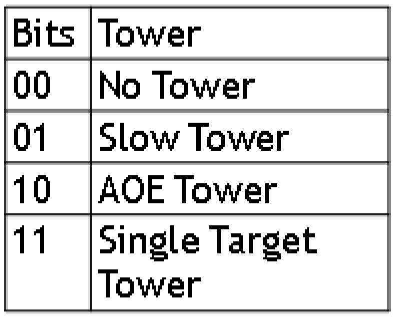


Fig. 1. This figure outlines the level design for the tower defense game. The green tile represents the creeps starting position and the red tile represents the home base with the gray squares being the path the creeps take. The yellow tiles are the locations of the towers. The chromosome for the GA is structured in that position 1 which is the top left tile is the first two bits while 34 which is the bottom right tile is the last two bits.



Bits	Tower
00	No Tower
01	Slow Tower
10	AOE Tower
11	Single Target Tower

Fig. 2. Decoded values of the chromosome.

For the purposes of this application 3 tower types were used. Tower type 1 had a cost of 6 and was a utility based tower that didn’t directly damage the creeps but instead provided a movement speed slow. Tower type 2 was an area of effect damaging tower with a cost of 12. Tower type 3 was a single target damaging tower with a cost of 15. These towers and costs were default for the template but after initial trials the type 3 tower was noticeably weaker so it’s damage was increased. The creeps were designed with the intention of being not particularly beneficial to a single target or area of effect tower dominant setup.

C. Implementation

A simple implementation of a GA was used for this application. The GA used single point crossover with flip mutation on a binary encoded chromosome. The encoded chromosome can be decoded as in Fig 2 Every two bits corresponded to a type of tower with 00 representing no tower. Due to the number of tower locations being 34 and two bits required for each tower the length of the chromosome used is 68. The population size was 20 over the course of 10 generations corresponding to 200 evaluations. Fitness was determined by minimizing the cost of the towers but no fitness was awarded if the health remaining was 0. A form of elitism was used to insure the best individual survived but it had to be modified to ensure that the best individual is not lost due to noise. Similarly the simple hill climber was implemented using the same chromosome with random bit flipping. The hill climber used the same fitness function and was limited to the same amount of 200 evaluations.

IV. Results and Discussion

To evaluate the GA 5 runs were conducted with averages taken for the results of the 5 runs. The hill climber preformed better on average but this can be attributed to a few factors that will be discussed.

First the Fitness will be evaluated. In Fig 5 it can be seen that the hill climber is able to achieve a higher fitness than the GA faster. This can be attributed to multiple factors. First is that the GA could have more optimally tuned parameters or more optimal operators. A CHC approach to the application likely would have resulted in better results than using elitism for a single parent. Another issue is the way to simulation works. Due to the noise the GA suffers much more than the hill climber. The children produced should have higher fitness if the noise didn't cause a given child to fail. In the hill climber it doesn't matter if it fails because it can easily produce the same or similar increase in fitness not much later. This causes the GA to converge to "safe" solutions that succeed more while the hill climber produces more volatile solutions that are more prone to failure. The fitness is calculated as $Fitness = 500 - Resources$ the resources are then $Resources = 500 - Fitness$. The resources required from equation [4] is 190 for the GA while the hill climber is 166. The resources required is then determined to be closer to 190 than 166 due to the safe solutions the GA produces. Furthermore despite the hill climber producing better results after 200 evaluations in this case, it is likely to produce solutions that are local optima. In Fig 5 it can be seen that you can't get from the first state to the second using a hill climber. For these reasons the GA is more suited to the task of finding the number of resources required and thus although it's fitness is lower the solutions produced are better.

For the optimal tower locations neither the GA or hill climber was successful in accurately finding the optimal locations. As noted above the inside corners are optimal and outside corners are suboptimal but in Fig 4 it can be seen that neither definitively express the optimal locations in the solutions found. If the GA was allowed to run longer it would likely produce better results for this. The hill climber however would not necessarily find these after more iterations due to converging to local optima as shown previously. Further if more runs were conducted the randomness would be reduced producing a more accurate distribution.

For the tower balance the towers used were designed in hopes of given a relatively even distribution of each tower. For the GA there were 92 towers used over the course of 5 runs. 39 of type 1, 26 of type 2, and 27 of type 3. In a perfect world if all is balanced perfectly then each tower will have the same tower balance of 1. However type 1 has a tower balance of 1.27, compared to type 2 and type 3's balance of

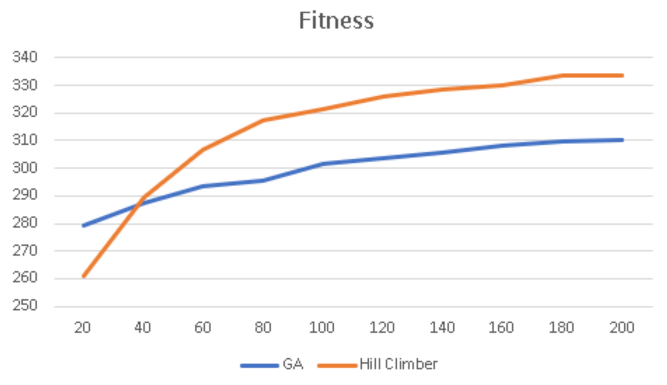


Fig. 3. Fitness of GA vs Hill Climber over 200 evaluations. The Hill Climber converges to a high fitness faster than the GA.

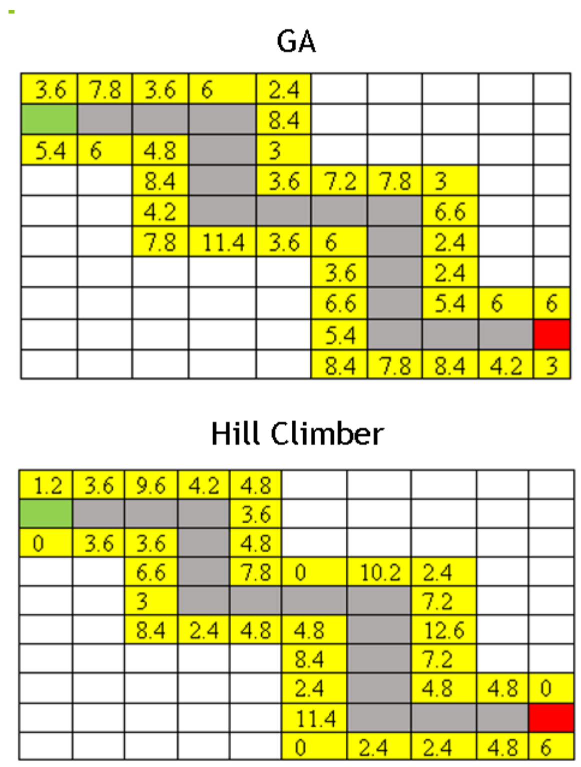


Fig. 4. The P(Loc) values for each position of the GA vs Hill Climber. Note that the inside corners are optimal locations while the outside corners are suboptimal locations. Neither the GA nor Hill Climber particularly finds these locations to be suboptimal in the trials done.

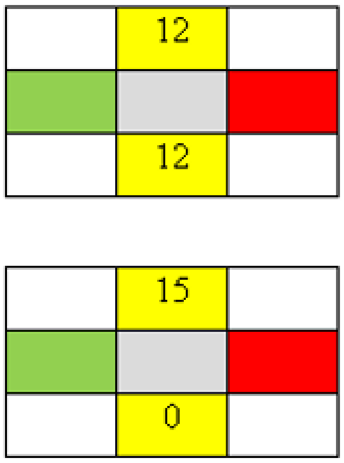


Fig. 5. If the hill climber is in a state such as the first above where two 12 cost towers are able to complete a level then the hill climber will never able to converge to the optimal solution of a single 15 cost tower being able to beat the level.

0.85 and 0.88 respectively. Similarly the hill climber produced a tower balance of 0.87, 1.62, 0.513 for towers 1, 2, and 3 respectively. Neither the GA nor hill climber particularly found such a solution but the results are useful nonetheless. The GA found a nearly even amount of area of effect and single target towers insinuating that those two towers are potentially equal in power relative to their cost. However it found an unreasonably high number of slowing towers of type 1. The GA's inability to remove these excess slowing towers likely led to the decrease in fitness. The hill climber on the other hand found a relatively good amount of towers of type 1 but it's distribution of the area of effect and single target towers was incredibly poor. This can be attributed to the local optima issue mentioned before in Fig 5 rather than the area of effect being vastly superior to the single target tower. If given more time to run or a more optimal configuration of the GA it likely would have determined that the towers produced balance numbers closer to 1.

Overall due to the limited time to evaluate the success of the GA and hill climber not a lot was able to be accomplished in regards to game balance but both methods show promising results and flaws that could with more work be useful in determining good game balance.

V. Conclusion And Future Work

Overall this paper didn't quite get accomplished everything it set out to accomplish. Due to limited time for evaluations and the length of time that evaluations take despite there only being 200 evaluations done for each run the GA was not able to be properly tuned to then use for game balancing purposes. However the GA did show promise in some regards. When given enough runs it is able to converge to good solutions that the hill climber cannot due to being stuck in local optima

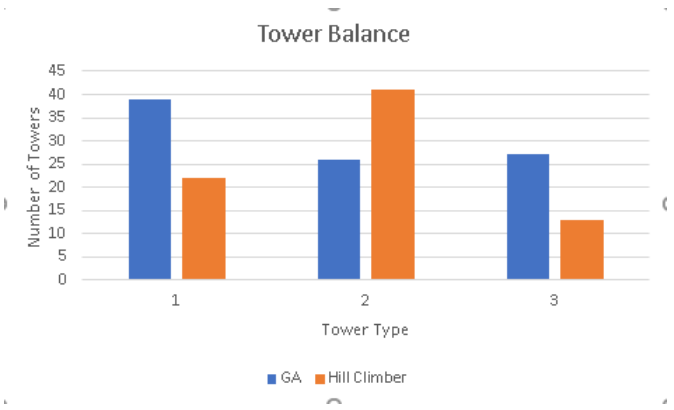


Fig. 6. The number of each type of tower found over 5 runs for the GA vs the Hill Climber. Note that they don't agree with each other so it is difficult to determine which if any is correct.

but not enough data was collected to accurately use. Even if the GA never achieves the same fitness as the hill climber in a small amount of evaluations, if the solutions it eventually produces are better then it is still more optimal to use a GA than endless man hours for game balance. Furthermore the noise was a huge hindrance and much time was spent trying to eliminate the noise. If this was continued in the future an environment that doesn't have noise would be developed. If more time was available more situations would have been able to test how well the GA is at finding artificially created balance through intentionally making things unbalanced for the purpose of discovering if the GA is able to find it.

Additional future work includes using coevolution to evolve a creep wave along with towers to further learn of balance of both creep waves and towers and possibly for automated level design. This was not able to be done in this paper due to the extensive computation time to evaluate two populations against each other. Even further than that is creating a GA wrapper around the whole application that has GA's running inside of it to balance the games parameters such as tower damage and creep health automatically.

References

[1] J. H. Holland, *Adaption in Natural and Artificial Systems*. Cambridge, MA: MIT Press, 1975.

[2] P. Huo, S. C. K. Shiu, H. Wang, and B. Niu, "Application and comparison of particle swarm optimization and genetic algorithm in strategy defense game," in *2009 Fifth International Conference on Natural Computation*, vol. 5, Aug 2009, pp. 387–392.

[3] T. G. Tan, Y. Yong, K. Chin, J. Teo, and R. Alfred, *Automated Evaluation for AI Controllers in Tower Defense Game Using Genetic Algorithm*, 01 2013, vol. 378, pp. 135–146.

[4] R. Leigh, J. Schonfeld, and S. Louis, "Using coevolution to understand and validate game balance in continuous games," 01 2008, pp. 1563–1570.

[5] Unity Technologies, "Unity." [Online]. Available: https://unity3d.com/unity

[6] ——, "Unity Tower Defense Template." [Online]. Available: https://learn.unity.com/project/tower-defense-template