2012 IEEE International Games Innovation Conference

Achieving Desirable Gameplay Objectives by Niched Evolution of Game Parameters

Scott Watson, Andrew Vardy, Wolfgang Banzhaf
Department of Computer Science
Memorial University of Newfoundland
St John's. NL. A1B 3X5. Canada
saw104@mun.ca, av@mun.ca, banzhaf@mun.ca

***Abstract*—Setting parameters for video games is important and often difficult. Poor choices can lead to games being too easy or too difficult, too short or too long, too linear or too open-ended. In many cases there is no alternative to making value judgments on what these parameter values should be. This paper explores whether evolutionary computation can be successfully applied to finding values that facilitate desired gameplay objectives in a Tower Defense game. Initial experiments suggest that a two-tier evolutionary method can create strategies to successfully play such games and configurations of game parameters that facilitate gameplay objectives desired by the developer.**

***Index Terms*—game design, evolutionary algorithms, evolutionary computation, multi-objective optimization, niching**

I. Introduction

The gameplay experience a game offers its players is critical to its success. Many factors influence gameplay but it is often very difficult to predict all the effects of changing a single parameter or identify the changes that need to be made to achieve a desired outcome. This makes it extremely difficult to achieve the gameplay experience the developer seeks through manual selection of game parameter values. This paper explores whether evolutionary computation can be used to effectively aid the selection of parameters in tower defense games. In order to do this, gameplay objectives are identified and player behavior is modeled.

Tower Defense games are a type of strategy game in which the player has the task of preventing mobile enemy units, usually arriving in a fixed number of discrete waves, from reaching an end point by placing stationary 'towers' that fire projectiles at the enemy units. If a certain number of enemy units reach the end point the player loses the game; this is sometimes presented as having a fixed number of 'lives'. The player earns points for number of enemy units destroyed, money left, number of 'lives' left and so on. In some cases the enemy units follow a fixed path, in others the player can control the path they take by building towers that block the way. Usually there will be several types of tower with different values for firing rate, damage, range and cost. Many variations exist and this description should not be considered representative of all such instances.

Although desirable properties of gameplay can vary from developer to developer and project to project, some widely desirable properties can be identified in a tower defense game. In order to succeed, the player should be forced to make meaningful choices about which towers to build, where to build them and when to build them. There should be multiple viable strategies to succeed to increase replay value and encourage experimentation by the player.

II. Problem Specification

A. Game Specification

A simple tower defense game was created for the purpose of carrying out these experiments. The player is assigned a number of 'lives' and an amount of money to start the game with. The playing area is a 20x20 grid. Enemy units follow a fixed path which is two grid squares wide, they enter at a designated starting point and exit at a designated end point. Any units that reach the end point re-enter the map at the start point, the player loses a life each time this happens. Every time an enemy unit is destroyed the player's money and score is increased.

Towers cannot be built on the path. There are three types of tower - red, green and blue - and the same three types of enemy unit. Red towers have a damage bonus when targeting red units and a damage penalty when targeting green units, green towers have a damage bonus when targeting green units and a damage penalty when targeting blue units and blue towers have a damage bonus when targeting blue units and a damage penalty when targeting red units. Tower types in other games are often determined by the game theme, for instance fireball towers or ice-throwing towers but fundamentally a tower type can be described by the properties of its weapon and, in some cases, a list of modifiers for enemy units. One common example of a modifier is that often certain towers will do no damage at all to enemy units that are designed as flying. Each game round features one type of enemy unit and they rotate between red, green and blue. There are twelve rounds in total. Towers can only be built between rounds.

B. Gameplay Objectives

1) Maximize Number of Viable Strategies

It is desirable to give the player several ways to play the game to increase replay value and encourage experimentation. It is important to note that it is the number of optimal or near-optimal strategies that is desirable, not the total number of strategies. If a great many strategies exist but one is far superior to any others, there is little incentive to use anything else. It is also important to note that this objective must be balanced by competing objectives that impose restraints on how a strategy can be viable. If placing a single tower of any type in any position on the grid leads to a successful outcome then the criteria for maximal viable strategies is met but the triviality of each one makes the overall effect undesirable.

2) Maximize Tower Type Diversity

If the player can win the game by building only, or mainly, one type of tower, they are not being forced to make choices about which tower types to build. Therefore it is desirable that viable strategies should require the presence of all types of tower.

3) Low Tower Position Clustering

If the optimal tower positions are all close together, say around a bend in the path or in some central location, there is little requirement for strategic choice of tower positioning. Therefore, the lower the clustering of towers in the population of strategies, the better. This objective is not intended to prevent the natural exploitation of features in the path, such as bends, but to reward configurations where there are choices between exploiting several features.

4) . Uniform Distribution of Towers Built Per Round

If all the towers are built in one round or in only a small number of rounds, it is likely the player will not be engaged during much of the game experience. Therefore the closer to a uniform distribution of towers built across rounds the better.

III. Method

An evolutionary algorithm can be applied to evolve a configuration of game parameters which facilitate the objectives specified in section II-B. In order to measure the objectives for a candidate configuration of parameters, it is necessary to find the strategies that are successful for that configuration and then evaluate their properties. This could be achieved by user testing but this is a slow and expensive process. The approach used in this paper is to use another evolutionary algorithm to evolve strategies to play a given configuration. This method is underpinned by the assumption that evolved strategies will be similar to the strategies that human players will develop.

A. Strategy Representation

In the simple tower defense game used for this experiment, the player's strategy is in choosing which towers to build, where to position them on the map and at what point in time to build them.

In games where the player controls the path of the enemy units by building towers in their way, there is additional pressure on tower positioning to force the enemy units to take a path that is as disadvantageous to them as possible. Further, in games where towers can be destroyed, a tactic termed 'juggling' can be employed where the player alters the path the enemy units follow during the round by opening and closing gaps in the line of towers by destroying existing towers and making new ones. Many tower defense games also allow players to upgrade towers, spending some of their money to increase the range, damage and rate of fire of a tower. In such cases, the decision of whether to upgrade an existing tower or build a new one must also be included in a player's strategy. These features are deliberately omitted from this experiment for the sake of simplicity.

A strategy in this example can be thought of as a list of 12 lists (one list per round) where the *ith* element is a list of towers to be built before the start of the *ith* round. Each list element is a tuple of <x, y, type> where *x* is the x co-ordinate of the tile the tower should be built on, *y* is the y co-ordinate of the tile the tower should be built on and *type* is the type of the tower that should be built.

B. Strategy Evolution

An elitist, steady state evolutionary algorithm is used to evolve strategies. A population size of 40, is evolved for 40 generations. Selection is performed with binary tournament selection using a tournament of size 8.

Recombination is carried out by randomly selecting two candidate strategies from the set of selected parents and performing uniform crossover to select the *ith* element of each parent with probability 0.5.

Five mutation operators are defined; add a tower, remove a tower, change a tower's x co-ordinate, change a tower's y co-ordinate and change a tower's type. These are carried out with mutation rate 1%.

Replacement in the population is carried out using restricted tournament selection, a form of crowding, to increase population diversity. In restricted tournament selection, a candidate solution is compared to the most similar solution of a randomly selected subset of the population and replaces that solution if it has higher fitness [1]. The size of the subset is called the window size.

The fitness of a candidate is determined by executing the game using the strategy represented by that candidate. A penalty is subtracted from the candidate's fitness score if a tower is specified in an invalid location (outside the map or on the path), if more than one tower is specified for the same location or if a tower cannot be built due to lack of money. Given that fitness is provided by the game's scoring formula, we can establish the theoretical upper bound on fitness for any given configuration. In practice this upper bound will be unobtainable since money is one factor in the score and the player will have to spend some of their money to build towers.

C. Configuration Representation

A configuration for the game we have specified consists of *lives*, the number of lives the player starts with, *mone*y, the amount of money the player starts with, a list of 12 round configurations and three tower specifications $<R_c, R_d, R_f, R_r>$, $<G_c, G_d, G_f, G_r>$ and $<B_c, B_d, B_f, B_r>$, specifying the cost, damage, firing rate and range for the red, green and blue tower types respectively.

A round configuration is specified by *speed*, the speed that enemy units move at, *health*, the number of 'hit points' each enemy unit has, *cash_reward*, the cash gained for destroying an enemy unit and *score*, the addition to the score for destroying an enemy unit.

D. Configuration Evolution

The evolution of the game configuration is a multi-objective optimization problem. The first step is to evolve a population of strategies for each configuration. From these populations, the values for each objective can be calculated.

1) Objectives

a) Minimize Gap Between Average and Optimal Fitness.

For any given configuration, the optimal fitness can be calculated. By definition, the smaller the gap between this optimal fitness and the average fitness of the evolved strategies, the more confident we can be that the evolved strategies are effective at playing the game.

b) Maximize Population Diversity

The more dissimilar the strategies are, the more distinct ways there are of playing the game. A function is defined that scores the similarity of two candidate strategies, with a value of 1 indicating the two strategies are identical and a score of 0 indicating the strategies have nothing in common.

All pairs of strategies in the population can be compared using this function and the summed average similarity scores give a measure of the population diversity. In a similar manner to the pairwise similarity function, a score of 0 indicates no strategy in the population has any shared elements with any other strategy and a score of 1 indicates all strategies in the population are identical. Smaller values are more desirable.

c) Maximize Tower Type Diversity.

For each strategy, the number of towers of each type is available. A simple function takes the difference between the tower type with the highest proportion of total towers and the tower type with the lowest proportion of total towers to rate tower type diversity from 0 to 1, with 0 indicating an even distribution of tower types and 1 indicating that all towers are of only one type.

The average of this value for all strategies in the population is taken, with smaller values being more desirable.

d) Maximize Tower Position Dispersion

For each strategy, the position of each tower is available. Average separation of towers, with each grid square considered 1 unit length, is used to measure dispersion with larger values indicating the towers are further apart.

The average of this value for all strategies in the population is taken, with larger values being more desirable.

e) Uniform Distribution of Towers Built Per Round

For each strategy, the number of towers built in each round is available. A simple function takes the difference between the round with the highest proportion of total towers and the round with the lowest proportion of total towers to rate the distribution of towers to rounds from 0 to 1, with 0 indicating an even distribution of towers to rounds and 1 indicating that all towers are built in exactly one round.

The average of this value for all strategies in the population is taken, with smaller values being more desirable.

2) Algorithm

The Non-Dominated Sort Genetic Algorithm II (NSGA-II) developed by Deb et al. is used to evolve configurations [2]. NSGA-II works by performing a non-dominated sort such that candidate solutions are sorted into non-domination ranks, where solutions in the first rank are not dominated by any solutions, solutions in the second rank are only dominated by solutions in the first rank, and so on. A crowding metric is calculated using the values for each objective.

Parents are selected to reproduce via tournament selection. Tournaments winners are determined first by rank (lower is better) and ties are broken by crowding metric (lower is better).

Recombination is carried out using uniform crossover, with the twelve round configurations, three tower configurations and starting lives and money values all being considered discrete genes.

Mutation operations are defined for each of the values in the configuration and each has a mutation rate of 1%. Step sizes vary based on the value to be mutated.

The combined current population and its offspring are sorted. The first rank is added to the new population, then subsequent ranks are added until the desired population size is reached. If adding a whole rank would result in a population larger than desired, the least crowded *n* individuals of the rank are added until the desired population size is reached.

IV. RESULTS

A. Strategy Evolution

Given that the game's scoring function places an upper bound on the maximum fitness of a candidate strategy, the number of generations, on average, it takes to converge to average fitness near the upper bound can be tested. This is very important as minimizing the number of generations has a large impact on the performance of configuration evolution. Table I shows the result of various population sizes and numbers of generations being tested against a set of random game configurations. It is non-trivial to determine if the maximum achieved fitness is the maximum possible fitness but given that the subset of strategies in the population that achieved the maximum observed fitness contains variation in structures, it can be said with reasonable confidence that the convergence of fitness to the levels seen is not simply a case of a local optima. From these results, a population of 40, evolved for 40 generations gives the best trade off between performance and computational expense.

TABLE I. POPULATION FITNESS

Population Size	*Generations*	*Average (Max 100)*	*Highest (Max 100)*
20	20	46.73	72.73
30	30	66.41	72.73
40	40	71.72	72.73
50	50	71.72	72.73

Testing various values for the 'window size' tunes the evolution process to evolve as diverse a population of strategies as possible. Table II shows that a window size of 24 is the best trade-off between performance and computational expense.

TABLE II. POPULATION SIMILARITY

Window Size	*Population Similarity (Max 1)*
4	0.912
8	0.890
12	0.893
16	0.888
20	0.876
24	0.865
28	0.879

Testing shows that a tournament size of 8 is the best trade-off between performance and computational expense as shown in in Table III.

TABLE III. POPULATION FITNESS

Tournament Size	*Average (Max 100)*	*Highest (Max 100)*
4	68.52	72.73
8	70.64	72.73
12	70.79	72.73
16	70.79	72.73

B. Strategy Evolution

Configurations were evolved for 20 generations using a population size of 50. The parameters for the strategy evolution were the same for all configurations, populations of 40 evolved for 40 generations, tournament size of 8 and window size of 24, as described in Section IV-A. Table IV shows the average, median, low and high values for each of the objectives after the final generation. As is typical in multi-objective problems, no single 'best' solution is produced but it can be seen that a wide range of values are produced and measured for all objectives. A solution must be manually selected from the Pareto front. In this case, the author was interested in a low similarity score and picked the configuration with the lowest similarity that scored above the median in the other objectives. In one sense, this is very similar to the standard development approach - a developer must use judgment to trade off one objective against another but using this method the judgment is based on quantified measures of the objective values that a configuration produces and not opinions, or guesses.

TABLE IV. OBJECTIVE VALUES

Objective	*Average*	*Median*	*Low*	*High)*
Fitness	0.74	0.73	0.59	1
Similarity	0.84	0.89	0.39	1
Tower Diversity	0.72	0.8	0.3	0.83
Tower Dispersion	4.17	4.05	2.75	5.63
Tower Distribution	0.26	0.25	0.14	0.42

C. Performance

This method is extremely computationally expensive. The complexity of one iteration of NSGA-II is O(MN^2) where *M* is the number of objectives and *N* is the population size, as described by Deb et al. [2] but this method also requires the execution of one evolutionary algorithm per population member per generation.

Performance is not critical to the success of the primary goal of the method. Even if it takes a full day or a full week to execute, this is not a significant timescale in the context of developing a game, especially when the potential reward is results that are superior to what can be achieved via manual selection. In this experiment, configuration evolution took one hour on a moderately powerful desktop computer.

V. CONCLUSION

This paper has demonstrated that evolutionary computation can be utilized to evolve successful strategies for playing the simple tower defense game created for this experiment. In principle there is no reason why the method used here will not scale effectively to tower defense games with more features.

This experiment also shows that gameplay properties which are extremely difficult to predict via static analysis can be measured through evolutionary computation and that targets within these properties can be approached at a computational cost that is expensive but not inviable.

Another usage for this method is to refine an existing configuration by initializing every member of the population with it. One or several values could be kept constant and the output would guide the developer in predicting the results of minor changes to their existing configuration.

VI. FUTURE WORK

Although this paper has demonstrated evolutionary computation can be effectively employed to help select values for parameters in a tower defense game, there are several areas that future research can expand upon.

The composition of the map has a large impact on the nature of the game. The length, width and shape of the path and the tiles adjacent to it have a direct influence on the nature of the game. In this paper the map is fixed for simplicity and to cut down on computational complexity. In future it may be interesting to evolve the map as part of the game configuration.

Similarly, there are many features of tower defense games that were not included in the game used in this experiment. They too could be incorporated in future experiments.

Because of the high computational cost of this method, it is desirable to investigate optimization of the evolution process. One area that may be worth considering is incorporating existing knowledge into the search procedure, for instance, it is well known that map tiles closer to the path are more valuable so starting strategies could be assigned using weighted probabilities rather than the current uniform distribution.

When the method is mature and the game it is intended for use in is ready for testing, user testing will be carried out to examine the success of the method. This will include testing whether the stated objectives have been achieved (i.e. a configuration identified as requiring a more diverse mixture of towers actually does require it) and testing whether the stated objectives are desirable.

Finally it will be prudent to implement the method using non-evolutionary computation techniques in order to compare and contrast results.

References

[1] G. R. Harik, “Finding multimodal solutions using restricted tournament selection,” in Proceedings of the Sixth International Conference on Genetic Algorithms, 1995, pp. 24–31.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182–197.2002.