Adaptive AI to Play Tower Defense Game

Paul A. Rummell Department of Computer Science
University of Victoria
Victoria, British Columbia, Canada
prummell@uvic.ca

***Abstract*—Adaptive algorithms give a way to solve a problem through successive attempts, without needing much information about a solution. The purpose of a tower defense game is to place towers that will kill enemies before they can reach the end of a fixed path. Using adaptive algorithms for a tower defense game allows the AI to not require any information about the types of towers and the layout of the path. This paper discusses a simplified algorithm to this problem (using only one type of tower), and possible extensions to this algorithm that would include multiple tower types and other modifications.**

***Keywords*-adaptive algorithm; tower defense; artificial intelligence;**

I. Introduction

Tower defense games are not usually associated with strong artificial intelligence (AI) as games are usually a human player against a predetermined order of enemies. By replacing the human player with an AI, the strengths and learning styles of both can be compared. Using adaptive algorithms, the AI needs very little information about the game and instead relies on experiential learning to develop its strategy. This prevents the AI from just being a copy of a human player, using pre-programmed methods based off what the programmer thinks is the best strategy.

II. Tower Defense Games

Tower defense games are relatively simple. The goal of the player is to place towers along a fixed path to prevent enemies from reaching the end. Enemies are released in waves (a number of troops released in short succession); the next wave is released when each enemy in the current wave has either been eliminated or has reached the end of the path. The player has several types of towers available, each with various attributes and side-effects. The attribute that affects the AI the most is range, which determines how far a tower can shoot at passing enemies. Thus towers that do not contain any path within their range will not be able to shoot at anything and will be useless. Towers cost money to build, and more money is earned by killing enemy troops.

Tower defense games provide an excellent field to study AI for several reasons. The simplicity of the game allows the AI to be tested on the entire game, while more complex games often require splitting components apart and studying them individually. Tower defense games also are easy to implement, requiring less time setting up the testing environment than other games would. They require strategy to be successful but usually provide a balanced learning curve. [1] Tower defense games also share many characteristics with real time strategy (RTS) games. RTS games are a popular genre, which has been worked on for years, but RTS AI performance is horrible compared to humans. [2] Although RTS games are far more complex, lessons learned on tower defense games can be extended to improve the quality of AI in RTS games.

A. Studied Game

The game that the AI played was a simple tower defense game, and the AI was restricted even further to focus the study. In the game studied, troops were released in waves of 30 at the start, increasing by 5 troops every 5 waves. Also, the player loses the game once 100 troops have made it to the end of the path. The player starts with 100 money, and earns 1 per kill (the amount earned increases by 1 every 5 waves). Each enemy can take an amount of damage equal to the wave number (starting at 1) before it is destroyed. The path that troops follow is shown in Figure 1.

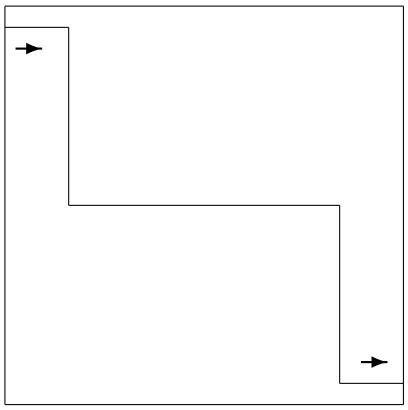

Fig. 1. Troop Path

The restrictions placed on the AI are as follows:

1) Only able to build one type of tower: a tower with medium range, 1 damage every 100 ms, and cost of 10.
2) Not able to upgrade towers.

Both of these restrictions were enforced so that the AI would not need to be concerned with its purchases, it could build a new tower whenever it gained 10 money. Without this restriction, if the AI was waiting to build an expensive tower costing 200 and several troops made it past its defenses that were not as strong as they could be (since it has unspent money), it would adversely affect the performance of the run.

If the AI loses while waiting for this tower, it makes the towers that it currently has built look weaker, when the problems were mostly caused by the waiting.

III. Initial Algorithm

The initial idea was to give each possible position on the map a rank based off how effective a tower is in that position. Ranks are initially set to ∞, and after each run, the value of each position in the run is set to the minimum of its current rank and the number of kills that the run achieved. The AI then builds towers starting with those of maximum rank and working down from those (it can omit any with rank 0 as they are useless). One problem with this is that it can get stuck on its best run and always repeat the exact same order. Another problem is that the best positions may have had their first run with lots of useless towers, which ends up giving them a bad rank because they were the only tower getting kills.

A. Modifications

These problems were solved by some small changes to the algorithm. Another rank was added, the first one becoming the minimum rank and the new one as the maximum rank. Maximum ranks were initialized as 0 and set to the maximum of its current value and the number of kills for each position in a run. Towers were built in descending order of the sum of their maximum and minimum ranks. This allows towers that have had one bad run to still be able to be picked, provided they have had a good run as well. To prevent it from getting stuck on one order, the AI builds 3 random towers before starting to build based on the ranks. This gives a chance for towers that were in a bad run to be in a better run, salvaging their use.

The problem that the algorithm now had was that a couple of useless towers would be in the best run, wasting the AI's money that it could spend on more useful towers. To fix this, at the start of each run, the AI would throw away 3 random towers from the best solution. Unfortunately, this random process was not very effective at eliminating the useless towers. So the AI was changed so it could detect when no progress had been made recently (i.e. last 100 runs). When it detected this, it would, instead of throwing away 3 random towers from the best run, throw away only 1, but it kept track of which ones it had done already so that it would go through all the possibilities once.

One more problem the algorithm had was that once it found a good run, it could not repeat it because it kept trying to build the towers in the wrong order. To fix this, positions were weighted by their order in the run. This means that towers that have been around longer get a higher rank than those built at the end. Only the maximum rank was affected because lowering the minimum rank for later towers causes the last few to get very low rank. The first 10 towers (those around at the start) receive full weight for the run while ones after that receive lower and lower, until the last one built receives almost 0. This ordering helps the AI to repeat the order the towers were built in again.

B. Results and Observations

Results from the initial algorithm and modifications were good but not spectacular. Depending on the variation and the randomly selected first set of runs, the best run was between 138 and 307 kills before it lost. This shows that the AI is learning the game since completely random runs usually got between 0 and 90 kills. Even with the random selection and discarding of a few positions, the algorithm seemed to get stuck at "local optimums." It would find runs that were fairly good, but it never seemed to throw away the worst positions while picking the best new positions and it would do almost exactly as well as before. Even running for several hours it could not seem to beat its record and often it would end up doing considerably worse than its best run.

One observation that was made was that combining runs of the algorithm significantly improved performance. This was done by running the algorithm until it stopped making progress and saving the results, then running the algorithm a second time until it stopped making progress and saving its results. Then by combining the saved runs of both times, it would be able to combine the best runs of both to find one that worked better. The two separate runs also allowed it to identify more positions that were completely useless as there are more chances for it to have picked 10 useless towers (and recognize that they are all useless because it got no kills). As an example, one combination combined a series with a record of 125 kills with one of record 134 to get a combination that worked up to a record of 209 kills.

IV. Variance Algorithm

The variance algorithm was an extension to the initial algorithm and modifications designed to speed up convergence. Rather than always throwing away 3 towers from the best solution and adding 3 random ones, it determines how close it is to a goal and replaces towers based on its current performance. Here the goal was set at 2000 kills and it replaced $\log_2(2000/record)$ towers from the best run with a random selection, where $record$ is the highest number of kills achieved so far. This allows runs which are not very good to vary greatly and runs that are really good to vary only slightly.

A. Modifications

The most important modification to the algorithm was the recognition of towers that did not fire during a run. This meant that the AI would eventually recognize all the useless positions and thus be able to spend money on only useful towers. Before, useless towers could only be recognized if a run contained all useless towers, so it would not recognize them all. This elimination of any non-firing tower needed limitations to avoid throwing away extra towers. The algorithm would not throw away towers built during the last wave because they might have been built right before the AI lost, and thus they never had a chance to fire, so they might not be useless. This modification allowed the AI to actually reach its goal of 2000 kills, but some improvements could still be made.

Another improvement made was determining the build order based off not only a position's value, but also the values

TABLE I
GAMES

Game 1		Game 2	
Run Number	**Kill Record**	**Run Number**	**Kill Record**
2	3	1	1
4	16	3	28
5	39	4	40
11	49	40	51
15	80	22	86
41	87	51	119
53	195	52	175
55	2000	54	176
		56	179
		57	226
		59	2000

of the positions adjacent to it. This allowed the AI to more quickly determine useless positions because it could assume that neighbors of a useless position would most likely be useless too. This sped up the convergence of the algorithm to the goal. Another modification was weighting each run by

$$kills * towersBuilt/(towersBuilt - nonFiringTowers)$$

This meant that towers in runs with lots of useless towers would not be penalized as much by the lack of the run's kills. This change increases the algorithms chances to converge because it decreases the dependence on what the first set of runs are.

B. Results

The recognition of non-firing towers allowed the AI to reach its goal. It would usually hit the goal after 55-60 rounds. With all the modifications, the algorithm would achieve its goal at least 50% of the time while the other times it would still get stuck at a "local optimum." Table I shows two games that the AI played and the records they set for number of kills.

V. Comparison of Human Players and AI

It is difficult to compare the AI with human players because of the strong limitations placed on the AI. Much more research is needed before an accurate comparison can be made. Even so, it is clear that pattern recognition plays an important role in performing well at tower defense games. Without pattern recognition, an AI would have to start from scratch when the position of the path changes, even if it could successfully complete the previous path. If it could infer "rules" from what it learns, then lessons learned would be transferable between layouts. But that is a different field of AI, and not the subject of this research.

VI. Further Research

Given the time constraints of this project, there is much left to study. One such area would be to find ways to speed up the algorithm's convergence to its goal. This could be accomplished by finding more ways for the AI to determine useless towers to eliminate. Another area that could be looked into is now that the algorithm can achieve its goal, is finding a way to achieve it with the fewest number of troops making it through as possible. Removing the AI's restrictions and finding a way past the problems they were intended to solve would be another useful extension of the algorithm. It would give the AI much more flexibility and also provide some useful insights into the effectiveness of various types of towers. Would the AI built all of one type of tower, or a mixture of several types? More research of AI in tower defense games can answer many questions as the simplicity and need for strategy of the game style provide an excellent area to study AI.

Paul A. Rummell is a fourth year student at the University of Victoria, completing a Combined Honours in Math and Computer Science. His interests include video games, music, and puzzles. He spends his free time designing and coding games, some of which are available online at http://squarevortex.ucoz.com/.

References

[1] P. Avery, J. Togelius, E. Alistar and R. P. van Leeuwen "Computational Intelligence and Tower Defense Games," to be presented at IEEE Congress on Evolutionary Computation. Available: http://julian.togelius.com/Avery2011Computational.pdf

[2] M. Buro and T. Furtak, "RTS Games as Test-Bed for Real-Time Research," Invited Paper at the Workshop on Game AI, JCIS, 2003.