TJHSST Computer Systems Lab Senior Research Project Design of a Strategy Game with a Humanlike AI Opitmized by a Genetic Algorithm. 2009-2010

Bharat Ponnaluri

January 24, 2010

1 Abstract

Currently the AI in many strategy games makes decisions by plugging in data from the game's environment into a combination of heuristic evaluation functions and constants. It is difficult to get an effective combination of heuristic evaluation functions and constants that allow the AI to make intelligent decisions. I plan to design a real-time strategy with an intelligent AI. To do that, I will have a genetic algorithm create combinations of heuristic evaluation functions and constants that allow the AI to make intelligent decisions. The AI will also be made intelligent by replicating human behavior and taking advantage of it by responding to certain behavioral patterns. For example, if an AI recognizes that one player has a tendency to retreat troops, the AI will take advantage of that by attacking that player. My genetic algorithmn will work by starting off with a population of random combinations of constants and heuristic evaluation functions, and evolve them into intelligent AI's using the principles of natural selection and evolution.

Keywords: Genetic Algorithm, Real-Time strategy, heuristic evaluation function,pattern recognition

2 Introduction

Currently, heuristics are important to video game AI. Heuristics are good at approximating optimal solutions with combinations of constants and evaluation functions based on the game environment. Herusitics are helpful because computers have difficulty using brute force calculations to arrive at an optimal solution. For example, in chess, the ability to look forward more than a few moves is a useful skill. A pure brute force approach does not work, because the chess games usually take several dozen moves to finish with the search time increasing exponentially generally by a factor of 16 or more each time. Looking past a certain number of moves is too time consuming for a computer to do. Effective chess AI must prune their search tree by using techniques such as as alpha-beta pruning and then using heuristics to evaluate a position after a certain search depth.

The same idea applies to other strategy games, since computers do not have the ability to look ahead too far using a brute force approach. In order for heuristic evaluation functions to give effective results, the function needs to have an optimal combination of constants and functions. With more than several constants and possible combinations of functions, trying to optimize the combination of functions and constants becomes increasingly difficult. Genetic algorithms will make it easier to optimize heuristics that involve a large number of evaluation functions and constants. For example, there are many factors that need to be considered in chess such as material imbalances, pawn structure, king safety, space, and initiative. The heuristic then needs to weight these factors optimally, which is difficult.

Also, my games AI will take advantage of human intelligence to make interesting and capable opponents. The AI will have the ability to take advantage of patterns in other players' behavior. For example, one player may play defensively and vigorously counterattack an opponent that attacked it first. The AI will recognize this and wait until that player is exhausted from a conflict with another opponent before attacking it. If another player is aggressive, the AI will wait until that player has overextended itself. The AI's will conduct diplomacy, so they will consider whether an opponent is trustworthy or not before making a deal. Since this game involves taking over cities that each give a fixed income, one player will eventually threaten to win if not stopped. The AI is designed to recognize this and try to stop the player. The AI will also recognize if another player has a tendency to stop the stronger player, and will sit back and take advantage of the situation.

3 Scope of Study

The objective will be to create an AI that is intelligent enough to beat someone who is a hardcore fan of my game without using cheats or overly using the fact that it has a faster reaction time than a human. It should be able to beat a skilled human player by making intelligent decisions and being able to exploit the behavior of human players to its advantage. At the same time, I should easily be able to modify the AI so that beginners can win against it. The algorithm should be somewhat generic, so it will be possible for me to extend this algorithm to a different game. In this paper, I will discuss genetic algorithms and why they are useful. Then I will talk about a real time strategy game I am designing to test my genetic algorithm.

4 Background

Genetic algorithms have not been used significantly in computer science and have not really been used in games. However, genetic algorithms have been used to solve several problems which would be difficult to solve using a brute force approach

In one tutorial, the author discusses the use of genetic algorithms for determining a mathematical equations using the numbers 1 to 9 and the operators -,+,*,/ to obtain a an equation that is a certain length that would produce a certain number. If the number is a number that has two factors less than ten such as 24, 42, 32, etc, the problem is relatively easy for a computer to solve. However, a number such as 83 is a prime number and it would be difficult for the computer to determine using a brute force approach. The fitness function is simply 1/(targetNumber-number that equation generates). As a result, suboptimal combinations of numbers and operators can quickly be eliminated, and the algorithm works from combinations that are close to the answer, which makes the algorithm quickly converge on a good answer. This could be a good idea for my game because instead of using the numbers 1 through 9, the operators would operate on the constants and heuristic evaluation functions.

Even when a person does not know what the answer is, a genetic algorithm can work effectively. For example, another problem involves a bunch of large disks in a bounded area, and trying to place the largest possible disk within the bounded area. Although a human can find the approximate solution for a combination of disks, finding the exact solution is difficult. A brute force approach would be inefficient, especially if there are a large number of pixels in the area. However, a genetic algorithm is able to solve it relatively easily. Another genetic algorithm involves creating a genetic algorithm to play the snake game. The AI snake is intelligent and is capable of competing with a experienced human. Although programming skill was required to code it, the programmer did not need to be skilled at playing snake.

Another research project involves using a genetic algorithm to optimize the constants for a heuristic evaluation function that is supposed to find the shortest possible degree-constrained spanning tree. A degree-constrained spanning tree is a tree structure that contains all the vertices's on a graph as nodes with the limitation that each node cannot be connected to more than a certain number of nodes. The shortest possible degree-constrained spanning tree occurs when the total distance of all the links between nodes. The conclusion for this project is that genetic algorithms can find shorter degree constrained spanning trees than traditional heuristics, even with heuristics intended to mislead a genetic algorithm. As a result of the success of a genetic algorithm to create a degree constrained spanning tree, the conclusion supports the use of genetic algorithms in other areas for other combinatorial problems, especially constrained ones.

The AIs that I am creating have personality traits, which are numbers. The personality traits are behaviors such as an AIs tendency to spend money, attack other players, or take revenge on other people. Certain combinations work well and some dont. For example, an aggressive AI which saves a lot of money is not going to be very effective.

A group of researchers used of genetic algorithms to generate a formula to predict the concentration of carbon-dioxide in the arteries, and was sucess-full. It shows the usefulness of prediction using genetic algorithms and the usefullness of using the genetic algorithm to generate forumulas.My current algorithm involves the AI's mostly making decisions on short-term goal and some heuristics instead of trying to predict the future. As a result, it may attack a player to gain a short-term income advantage, only to get steam-rolled by another AI who had more troops. Even with the genetic algorithm optimizing my AI, predicting the future would be difficult unless I modify my AI's design.

A potential pitfall with the genetic algorithms are evolutionary stable strategies. Even though they may generate an interesting personality, that personality may lead to very unintelligent behavior. For example, my genetic algorithm may develop a population of turtling AI's which build up troops until they have enough of them to blitz the board and will only otherwise fight to take back lost cities. As a result, their personalities and their heuristic evaluation functions will be similar and different types of AI will be unable to be created by having the turtling AI's share the chromosomes/(heuristic evaluation functions). And randomly mutating or slightly altering the heuristic may not work. Aggressive players will lose too many troops. Even AI's which strike a intelligent balance between attacking and turtling will find themselves losing as even attacking a little bit will lead to a net loss of because the target player will be focusing all its troops on the attacking AI. The main problem with this AI is that it is very boring to play against, since a human player will just be sitting there and buying troops.

Game Theory will be one of the tools that I will apply to make the AI's more human-like. Game Theory is something that the computer's can understand and is something that explains human behavior. The game of chicken is an excellent example of this. In the game, two AI's want control of a particular city. Neither of them wants to back down because the other players will observe that the player who backs down is weak and can be attacked. On the other hand, if neither of them backs down, then both players will end up spending an unnecessary amount of troops for no reason, increasing the relative strength of other players. As a result, the game of Chicken will greatly simplify AI behavior because there are only two possible choices. This game also will make it far easier for me to help determine an AI's conflict behavior and whether my genetic algorithm is working correctly.

4.1 Game Theory Examples

The following table is an example of how game theory can be used. Here two players want to capture a certain city. Neither of them wants to give up because the other players will perceive them as weak-willed and easy to attack. On the other hand, if both of them fight, then they are weakening themselves to the benefit of a third player. The first number in the row describes the relative payoff to player one, the second number describes the payoff to player two, and the third number is the payoff to player three. "t" represents how long the players fight over the city, and the longer they fight, the more the third player benefits.

	.	Fight	Don't Fight
	Fight	(0,0,0)	(-t+1,-t,t*2)
	Don't Fight	(-t,-t+1,0)	(-t,-t,t*2)

The following table represents the Prisoner's Dilemma and how evolutionary stable strategies can lead to suboptimal or other types of unwanted results. For example, a population of defectors is evolutionary stable strategy because any strategy which attempts cooperation will have a negative payoff while the defectors will gain a positive payoff. In my game, this could lead to AI's which never cooperate with each other and conduct diplomacy. This could lead to games being bogged down in stalemates.

	.	Cooperate	Defect
	Cooperate	(1,1)	(3,-10)
	Defect	(-10,3)	(-5,-5)

5 Development

5.1 DesignCriteria

The AI that I create should be smart enough to defeat a skilled player the majority of the time. Also, it should have the ability to mimic human emotions. However, the main focus will be on making the AI play intelligently in a way that is enjoyable. To test this out, I will set up a game. Also, it should be fun for players of all skill levels to play my AI, which I will test out by getting random people to play against the AI I created. The genetic algorithm that I create should be as generic as possible so that it can be applied to other strategy games.

During third quarter, I will finish debugging the output of my program since implementing the new heuristic evaluations messed it up. Afterwards, I will spend several weeks thoroughly searching for bugs and run a no-graphics version of my game in my genetic algorithm. While the genetic algorithm is running, I will work on improving the speed of my rendering. Also, I will improve the interface of my game and adding add things. I will let my genetic algorithm optimize my heuristic evaluation functions instead of optimizing them myself like I initially planned

5.2 Timeline

- February:
  - Week 1: Make sure improved heuristics are working
  - Week 2: Play game with improved heuristics to make sure they are working
  - Week 3: Finish coding genetic algorithm with no-graphics version of my program
  - Week 4: Start genetic algorithm and improve graphics rendering speed
- March
  - Week 1: Continue improving graphics rendering speed
  - Week 2: Improve program interface
  - Week 3: Add code to create custom maps
  - Week 4: Create interface to allow different AI algorithms to be implemented

- April
  - Week 1:View results of genetic algorithm and improve upon its flaws
  - Week 2: Implement different personalities
  - Week 3 and 4: Implement heuristics that allow AI to observe patterns of other players and take advantage of them
- May: Run 2nd version of my genetic algorithm. While the algorithm is running, I will add new units and make sure the AI knows how to use them
- June: Run genetic algorithm with added units

5.3 Genetic Algorithms

The main advantage of a genetic algorithm is that it is capable of arriving at an optimal solution in a relatively short time without user input, even if there are many constants and function combinations that need to be optimized. A genetic algorithm works by randomly determining a set of parameters and function combinations that are represented in chromosomes. An algorithm is run once for each chromosome based on the data in the chromosome. Afterwards, based on the chromosome's performance during the algorithm, a fitness score is calculated for the chromosome. Then the chromosomes with the lower scores are eliminated. Then the chromosomes randomly mutate and have a small portion of their data randomly modified. Then the surviving chromosomes "mate" and swap data, then the algorithm runs again. Genetic algorithms have a tendency to have their chromosomes converge on locally optimal places. For my algorithm this will be a good thing is long as the local optima are not too far from the optima. This is because I would like a diverse set of AI's because they would have different personalities and make the game more exciting. Here, we propose the use of a genetic algorithm to optimize the heuristic evaluation functions for the AI of a strategy game.

5.4 AI Heuristics

- getIncomePower(Player p): Finds how much income a player has compared with other players
- getEnemyPower(Zone z): Finds how many enemy troops are in a zone

- getPotentialBorders(): Calculated the number of border cities a player will have if they take a certain city
- getAverageProximity(City c): Calculates the average distance of a city from all the player's cities
- getTroopPower(Player p, Location loc): A total of the number of troops around an area weighted by how close they are to loc
- getZonePower(Player p, Location loc): The number of cities around a city weighted by how close they are to loc
- getPotentialNeighbors(Zone z): How many neighoring enemies a player will have if they take a certain zone

5.5 Perecption Heuristics

These will be used by the AI to replicate human behavior by taking advantage of patterns

- HashMap¡Player, Double¿ ThreatFactor: Represents how threatening a player is based on how much they have been fighting the AI
- HashMap¡Player, Double¿ PercievedAssertiveness:Gives a perceived assertiveness value for each player based on how often it stands its ground.
- HashMap¡Player, Double¿ leaderAttackFactor: Percieved likelihood of a player to attack a game leader.
- HashMap¡Player, Double¿ predicability: Percieved likelihood that a player will react predictably

5.6 Personality Traits

- Agressiveness: Determines how likely a player is to attack
- Assertiveness: Determines how likely a player is to stand ground instead of retreat. Retreating too much will make you look like a pushover, and retreating too infrequently will make you loose troops. It should also measure an AI's response to an attack
- Paranoia: How much an AI will reinforce based on the number of enemy troops in nearby cities

- Powerfulness: How likely an AI is to attack a game leader, especially one who is threatening a win. If an AI attacks the game leader too much, others will be more likely to sit back,and reap the spoils from two weakened enemies
- Artillery:How likely is the AI to build artillery. This value goes up as the number of enemy troops goes up
- RevengeFactor:How likely is the AI to attack a player that it has already fought

6 Discussion

Converting the City class to a Zone class and having the Zone class contain references to the locations of AI troops was successful. The coding of the AI heuristics was somewhat simplified and my program ran somewhat faster. However, debugging the implementation of the Zone class took far longer than expected because I ended up accidentally modifying my code. As a result, none of the troops were being displayed. I eventually tracked down the error source and found out that the troops had a movement of 0. I was surprised as I believed that any errors would involve the implementation of the Zone class. As a result, the troops were being moved to undefined locations and being deleted. Changing the troop movement fixed the problem. I resolved to have a more careful plan on how to implement any new code before coding it.

I waited until I carefully planned the AI heuristics before implementing them. I implemented the AI heuristics described in the previous section and they printed out sensible values. Also, when I plugged in the values of these heuristics into my findTargets() method, the output was generally between 0 and 2, which means that they have been correctly implemented. However, I implemented the heuristics in a way that made every city owned by the AI to turn white. Also, the AI's never built troops.

In summary, tracking down errors has taken an unexpectedly long time, even after I started planning. As a result, I did not accomplish as much as I had hoped to. However, this does not surprise me because it took me several months last year to fully implement my old AI heuristics..

7 Results

I then decided to test out the current AI algorithm to see how fast it runs. The heuristic algorithm I currently use for the AI will be useful with the improved AI I created. The algorithm depends mostly on the number of cities on the map and the size of the squares used to store the troops. The speed of the algorithm does not mainly count on the number of troop presents. It runs once every .266666 seconds, which means that the speed of this algorithm is a significant issue. Making the AI algorithm run less often and staggering each of the AI methods would give a significant performance boost while not significantly affecting the intelligence of the AI

Number of Troops on Map	Time Taken in Seconds for AI algorithm
9	0.14
1024	0.16
2000	0.2
5000	0.2
6822	0.3

I then tested the graphics and rendering part of my game, which is where I think the speed issue is the most significant. It runs once every 0.0655737 seconds so it runs 15 times per second. I do not want to make the graphics algorithm run less often or the frame rate will be too low. Even with a low number of troops on the map, the graphics algorithm takes too much time, especially since the graphics and the AI algorithms do not run in parallel. As the number of troops approaches 1000, the graphics algorithm starts taking more time than the length of a graphics frame, which is why the frame rate is low and the why the game is unresponsive to user input after a while. Although I could take advantage of running parts of my game in parallel using a dual core processing, concurrent programming makes a program difficult to debug.

Number of Troops on Map	Time Taken in Seconds for graphics, Time(seconds) taken to run
10	0.05,0.75
1000	0.08,1.2
2000	0.15,2.25

In summary, my current code has significant speed issues. The new AI algorithm I am designing will take up a significant amount of computing power. Also, having more efficient code will make a genetic algorithm finish faster and reduce the need for complex networking because I will be able to run multiple instances of my game on the same processor core.

References

[1] Neville, Melvin., Sibley, Anaika.(2000). Developing a Generic genetic algorithm.*ACM,1*, Retrieved from: http://portal.acm.org/

[2] Frayn, Colin.(2005, Aug. 5) *Computer Chess Programming Theory.*Retrieved October 28, 2009 from:http://www.frayn.net/beowulf/theory.html

[3] Buckland, Matt. Genetic Algorithms in Plain English. October 21, 2009, from ai-junkie:http://www.ai-junkie.com/ga/intro/gat1.html

[4] Ehlis, Tobin. (2000, Aug 10)Application of Genetic Programming to the Snake Game.October 21, 2009 from http://www.gamedev.net/reference/articles/article1175.asp

[5] Raidl, GR.,Julstrom, Bryant A. (2000). A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem. ACM,1,Retrived from

[6] Shor, Mike. (2007). Retrieved from http://www.gametheory.net

[7] Sirlin, David. (2006, Apr 24). *Playing to Win: Becoming the Champi- http://chestjournal.chestpubs.org/content/127/2/579.full.htmlon* Retrieved from http://www.sirlin.net/ptw/

[8] Phelps, Selcen.,Koksalan, Murat.(2003). *An Interactive Evolutionary Metaheuristic for Multiobjective Combinatorial Optimization. 49*,1726-38 Retrieved from http://www.jstor.org

[9] Engoren,M.,Plewa, M.,O'Hara,D.,Kline,J.(2005) *Evaluation of Capnography Using a Genetic Algorithm To Predict Paco2. 127* 579-84 Retrieved from http://chestjournal.chestpubs.org/content/127/2/579.full.html