TJHSST Computer Systems Lab Senior Research Project
Design of a Simple Real Time Strategy Game with a genetic AI
2009-2010

Bharat Ponnaluri

October 30, 2009

1 Abstract

Currently, the AI for strategy games involves combinations of heuristics and constants that need to be optimized for heuristic evaluation function to work efficiently. The problem is that there are a large number of constants and combinations of heuristics to optimize, a heuristic evaluation function may return suboptimal functions. For example, chess computers operate by evaluating heuristics based on things such as material, and position, which can be composed of many variables such as the moves available, and where pieces are. I plan to design a simple real-time strategy game and use a genetic algorithm to create the AI. Genetic algorithms can produce constants and combinations of heuristic algorithms that are optimized relatively quickly and accurately. They operate in a manner similar to evolution by eliminating suboptimal combinations of heuristics and constants, and swap the data of the surviving combinations.

Keywords: Genetic Algorithm, Real-Time strategy, heuristic evaluation function, machine learning

2 Introduction

Currently, heuristics are an important part of the artificial intelligence for computer games. Heuristic algorithms are a programming technique for ar-

riving at an approximately optimal solution by deciding on a solution using heuristic evaluation functions. They are helpful because computers do not have sufficient computing power to arrive at an optimal solution in many cases. 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. Basically, this means that the computer would have to increase the number of searched exponentially as the depth increases. As a result, effective chess computers must reduce the size of their search tree using methods such as minimax and alpha-beta pruning and then use heuristic evaluation functions to eliminate suboptimal positions and clearly suboptimal moves. A genetic algorithm can be used to help methods such as minimax reduce the size of a search tree

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 large numbers of function combinations and constants and function without user input

3 Scope of Study

The objetive 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 behaviour 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 inolves 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 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.

5 Development

5.1 DesignCriteria

The AI that I create should have the skill level of someone who is a hardcore fan of my game and has been playing it for a long time and be capable of diplomatic interaction with humans and the ability to mimic human emotions. However, the main focus will be on making the AI play intelligently. 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 AI that I create should have the skill level of someone who is a hardcore fan of my game and has been playing it for a long time and be capable of diplomatic interaction with humans and the ability to mimic human emotions. 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. Also, I hope to make the AI algorithm as generic as possible, so it can be applied to other strategy games.

For the second quarter, I will improve the heuristic evaluation functions for my AI code. For my genetic algorithm during the 3rd quarter, the chromosomes for each AI will be composed of mathematical operators, constants, math functions, and the heurstic evaluation functions I am writing. The heuristic functions will be the basic building blocks of the chromosomes, and a certain AI chromosome may not use a certain chromosome. As a result, during 2nd quarter, I will focus on writing basic heuristic evaluation functions. Then I will put together a combination of those functions for the AI

5.2 Timeline

- 2nd quarter
  - 11-2 to 11-6
  - Rest of November
  - December
  - January

- 3rd quarter
  - February-Write genetic algorithm
  - March-Run genetic algorithm
  - March-While genetic algorithm is running, polish the rest of the game
  - April-Test out genetic algoritm against AI I coded 2nd quarter to see if it is more intelligent
- 4th quarter
  - Find another strategy game and design a genetic AI for it

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 chromosones. 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 suriving chromosomes "mate" and swap data, then the algorithim 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.

6 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