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

Bharat Ponnaluri

June 1, 2010

1 Abstract

Currently, the AI's for many video games have algorithms that are composed of combinations of constants and heuristics which are created by programmers. This approach can generate AI that is somewhat intelligent. However, this approach leads to an AI that is poor at pattern recognition and easy to exploit. In chess, computers such as Deep Blue have compensated for this by doing a brute force search and having a complex heuristic evaluation function that evaluates things such as material balance, king safety, and piece mobility. It is especially difficult to generate good AI when there are large numbers of possibilities in due to the complexity of a game. In Civilization 4, the AI often attacks another player by suiciding a large army against a fortified city, which is a result of the current approach to AI design. In Rome Total War, the AI often attacks by simply marching its army straight towards you, which can be easily exploited. Due to the complexity of these games and the randomness of combat, a brute force approach for the AI heuristics does not work. Also, it is difficult to find a good combination of heuristics and constants. Using a genetic algorithm to optimize the AI could potentially work better since the AI will be trained to correctly evaluate potential actions.

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

2 Introduction

Currently, heuristics are important to video game AI. Heuristics work by approximating optimal solutions using combinations of constants and evaluation functions based on the game environment. They are helpful because using a brute force calculuation to find an optimal solution takes too much time. In chess, a brute force approach to finding the best does not work, because the chess games usually take several dozen moves to finish with the search time increasing exponentially by a factor of 16 or more each each move. In order to compensate for this, chess computers have heuristics which calculuate things such as material balance and number of squares controlled.

In order for heuristic evaluation functions to give effective results, the function needs to have an optimal combination of constants and functions. This becomes increasingly difficult as the number of usable functions and constants increases. Genetic algorithms will make it easier to optimize heuristics that involve a large number of evaluation functions and constants. In chess a heuristic needs to compute the value of things such as material balance and the quality of the pawn structure accurately. Also, it needs to effectively weight these factors in a wide variety of positions,which is difficult.

The AI will also be programmed to take advantage of human intelligence to make interesting and capable opponents. For example, the AI will take advantage of patterns in other players' behavior. If one player plays defensively and vigorously counterattacks an opponent that attacked it first or is agressive, the AI will recognize this and wait until that player has been weakened from another conflict. If another player is aggressive, the AI will wait until that player has overextended itself before attacking. 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 Background

Despite the fact that genetic algorithms have not seen widespread use in computer science and to develop AI, they have untapped potential. Researchers have proven that genetic algorithms can solve problems which would be difficult to solve using traditional methods such as manually creating heuristic evaluation functions. A genetic algorithm works well even when a person does not know a solution to a problem or fully understand how to arrive at a solution.

In one example, AI programmer Buckland demonstrates the use of genetic algorithms to create mathematical equations using the numbers 1 to 9 and the operators -,+,*,/ to obtain a an equation for 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 to solve usign a brute force algorithm.However, a brute force algorithm will have trouble finding expressions for prime numbers. The genetic algorithm saves expressions that are close to the target number, which allows the algorithm to quickly converge to a certain number. I tried to replicate Buckland's work 3rd quarter as a starting point for my genetic algorithm. Buckland also demonstrates the use of a genetic algorithm to try to place the largest possible disk in an area full of randomly placed circles without having the disk intersect the circles. Although a human can find the approximate solution for a combination of disks, finding the exact solution is difficult.. However, a genetic algorithm is able to solve it relatively easily. Researcher Ehlis sucessfully created a genetic algorithm to play the snake game when writing a heuristic manually would be difificult.

Julstrom and Raidl created their own heuristic evaluation functions but optimized the constants that the functions were multiplies by to find the shortest degree-constrained spanning tree. A degree-constrained spanning tree is a tree structure that contains all the vertices on a graph as nodes where each node has a limited number of connections. The shortest possible degree-constrained spanning tree occurs when the total distance of all the connections between nodes is minimized. The conclusion is that genetic algorithms are effective tools for optimizing the results produced by heuristics. As a result of their success, Julstrom and Raidl encourage the use of genetic algorithms for other combinatorial problems.

The AIs that I am creating have personality traits, which are numbers. The numbers represent behaviors such as an AIs tendency to spend money, attack other players, or take revenge on other people. Certain combinations work well and some do not. For example, an aggressive AI which saves a lot of money is not going to be very effective. The goal of personaility traits is to provide diversity in how the AI's behave.

A group of researchers sucessfully used of genetic algorithms to generate a formula to predict the concentration of carbon-dioxide in the arteries. The research shows how genetic algorithms are good at generating accurate predictions and formulas. My current algorithm involves the AI's making decisions solely based on short-term goals and some heuristics. As a result, my AI often attacks, only to get steamrolled by another AI who had more troops.

A downside to using genetic algorithms are the presence of evolutionary stable strategies. Because of them, my algorithm could generate unintelligent AI's. 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. I plan to mitigate this problem by making the genetic algorithm test the effectiveness of the AI's by having them evaluate scenarios.

I eventually plan to apply the priniciples of game theory to make the AI's more human-like and intelligent. It will be easy to make the AI's understand game theory because of its simplicity.

Another key concept that could be applied to a heuristic is Lancaster's Square Law. The law gives the relative casualties of two armies based on their relative strength at the end of a battle. This will be something that it helpful for the AI to know, and it would help the AI gain an advantage over a human player without cheating because combat between armies of dots is a central part of my game.

One of the reasons why humans can defeat AI's in a strategy game is because of their skill in pattern recognition. After playing a game repeatedly, a human player will find patterns or groups of similar situations which is something AI has trouble doing. Researchers at the University of Illinois have created a method of automated pattern creation and pattern recognition trained by a genetic algorithm. The research show how genetic algorithm can make computers recognize patterns, which will be something helpful in my game.

3.1 Game Theory Examples

The following table is a representaton of the game chicken. 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 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)

4 Development

4.1 DesignCriteria

My goal is to create AI that can defeat a skilled player who uses a variety of tactics a majority of the time. The AI should also be able to mimic human emotions to make the AI more enjoyable to play against. To test out the intelligence of the AI I create, I will play against the AI and try a wide variety of tactics. Players of all skill levels should enjoy playing against my AI. While I want my genetic algorithm to create an AI that fulfills the above criteria, I want it to be easily applied to other strategy games.

I decided to adopt a model of Iterative Development by incrementally adding complexity to my game. Once I created the real time strategy game that I was using for this project, I started working on a basic version of the AI which randomly attacked. Afterwards, I implemented some heuristic evaluation functions so that the AI could attack and build troops based on opponent strength in surrounding cities. This AI was able to play a competitive game, but the main reason for this was that it could expand faster than me in the beginning of the game because it did not have to make multiple mouse clicks.

Afterwards, I decided to fix the large number of bugs and speed issues before moving on to creating a genetic algorithm. I fixed many of the bugs such as the improper displays of troop count and the inability to tell how many troops were at a certain location. After unsuccessful attempts at improving the speed, I decided to move on and create a second iteration of a heuristic-based AI. I ran into problems and I had to completely rewrite large portions of my code to implement the new heuristics. The development of the AI was slowed by the fact that I was doing my coding on a laptop with Windows Vista which failed twice, which led to significant delays.

In the beginning of February, I decided to move on and start coding a genetic algorithm. My goal was to have the algorithm create an expression composed of heuristics that would make the AI act in an intelligent manner. I decided to initially create a genetic algorithm to generate formulas for a target number based on constants and mathematical operators. Afterwards, I made a genetic algorithm generate formulas for input-output pairs that were based off of polynomial functions. The genetic algorithm was able to generate accurate formulas for numbers close to 0 but was unable to do so for larger numbers because it could only recognize linear functions.

My plan was to eventually apply this genetic algorithm to optimize the heuristics for my AI. The inputs were supposed to be the heuristic evaluation functions and the output was a number that represented what the AI should do.

Instead of having the AI's play against each other, I decided that creating a series of scenarios was better. I knew that my genetic algorithm would have bugs and I did not have enough computing power to have the AI's play against each other in a genetic algorithm and have the genetic algorithm finish in a reasonable amount of time.

For the initial 3iterations of my project where I created the RTS and the two iterations of my AI, I carefully wrote down an outline of what I was going to code. For the genetic algorithm, I just spend a while thinking about how I was going to write the code, made a quick plan, and then started coding.

4.2 Genetic Algorithms

A genetic algorithm works by randomly creating a sets of data representing potential solutions that are represented in chromsomes. It applys the principles of evolution and natural selection to produce a solution to a problem. During each iteration, an evaluatio function is run on each chromsome to see how well it corresponds with a solution. Afterwards, based on how close a chormosome is to a solution the chromosome receives a fitness score.

Then the chromosomes with the lower scores are eliminated. The remaining chromosomes randomly mutate and swap data between themselves. enetic algorithms have a tendency to have their chromosomes converge on locally optimal places. For my algorithm this could 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 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.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.

4.3 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

4.4 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

4.5 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

4.6 Classes

- FormualaGenerator: Driver program that initializes the visual display of the average
- GraphPanel: Displays the graph of the important data and runs the genetic algorithm.
- Chromosome: Represents the chromosomes/expressions.

4.7 Methods

- createRandomExpression: Randomly creates a combination of double values between 1 and 10 plus functions on the numbers and operators

- mutate: Iterates through each expression and randomly mutates a part based on the mutation rate.
- calculateFitness: Evaulates expression using order of operations. The fitness value is the absolute value of (targetNumber) divided by the difference between the expression value and the target number.
- removeFailures: Finds all the chromosomes with a fitness that is below average and removes them.I plan to modify this method so that a few of the unfit chromosomes are saved.
- crossover: Randomly picks two chromosomes to mate data and saves a child. Once there are 20 children, the parents are deleted and the population becomes the children.
- mutations: Generates a mutation rate based on the standard deviation of the fitness values and then randomly calls the mutate() method to attempt mutations. This method will be modified
- drawGraph: Draws a graph of the average fitness of the population, the best fitness of the population, and the standard devation. These values are saved for every generation, allowing the graph to be completetly redrawn for every iteration.

4.8 Sample expressions with static mutation

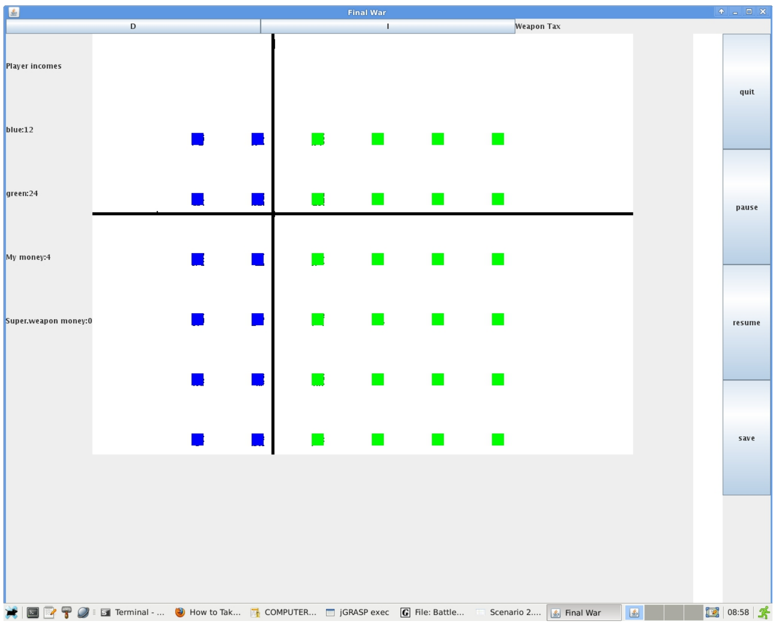
TargetNumber=150

3.5161634238733376 + (((-0.12546679293851723) / 5.888617462304062) / 8.599315382204576) + ((2.11143617460612 * (-0.9941215166113628)) / 0.7683425092250671) + (6.573680892927516 / 0.04670622769266575) + 7.754730242410334 = 149.281806(Good

((1.3927347303146886 * 0.459609238527513) / 0.009618556327199768) + ((((5.562886669798012 / 0.8976440279752964) * 7.875601052535996) / (-0.9133704835596479)) * (-0.8646457868104498)) - (0.888014311694606 / 5.748443177380818) = 112.59849(Bad)

4.9 Sample Scenarios

Here are several scenarios that I will be using to test the effectiveness of the AI's during my genetic algorithm.In the first scenario, one player has more cities and one player has more troops In the second scenario, each palyer has the same start position. In the third scenario, one player has more cities than the other players combinted



Player incomes

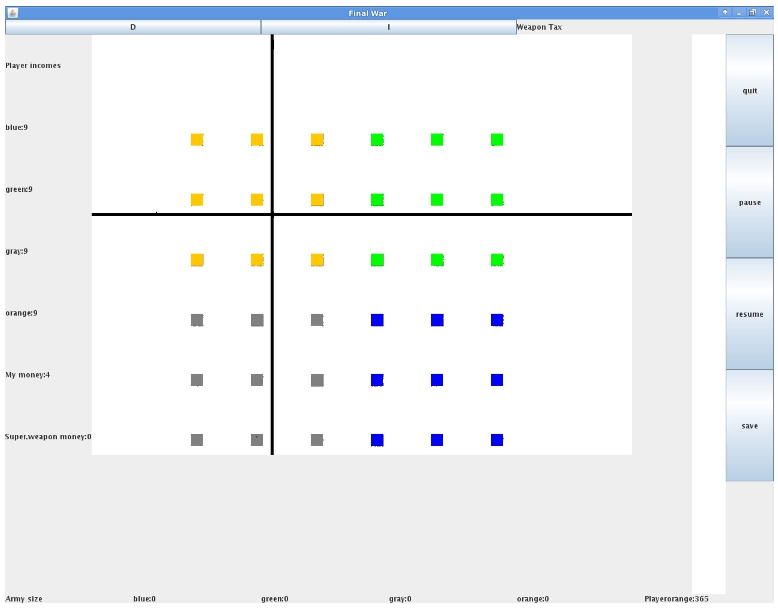
blue:12

green:24

My money:4

Super.weapon money:0

1.jpg



Player incomes

blue:9

green:9

gray:9

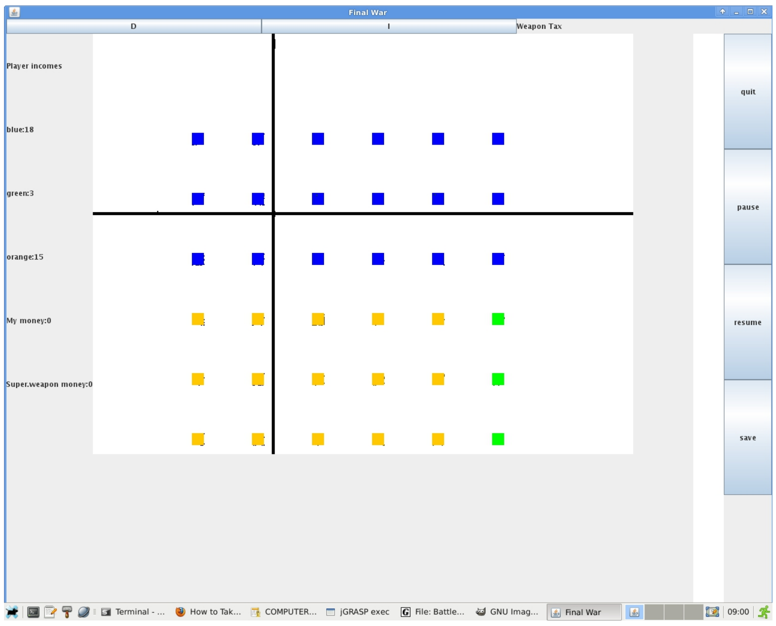
orange:9

My money:4

Super.weapon money:0

Army size blue:0 green:0 gray:0 orange:0 Playerorange:365

2.jpg

3.jpg 

5 Results

6 Discussion

My genetic algorithm works as it generates expressions for numbers somewhat accurately. However, due to a bug in my algorithm which sometimes causes all the chromosomes to be eliminated, the algorithm crashes before it finishes. While I was coding the genetic algorithm, I realized that I should have a solid genetic algorithms without problems such as long periods of stagnation or a popluation with a low standard deviation. This is because I realized that these problems would be easier to deal with when my genetic algorithm was solving a problem that is not as complex as creating a heuristic for my game. To cope with the periods of stagnation, I decided to make the mutation rate dependent on the standard deviation and the average change in the average fitness values of the population. However, the dynamic mutation rate is not working perfectly because the standard deviation of the population approaches 0 and stays there. Also, I created a visual display of the standard deviation, average fitness, and maximum fitness values because I felt that I was spending too much time trying to analyze the output(the data on the visual display) and trying to manually create graphs on the computer. In summary, I have a functioning genetic algorithm, but the population has a low standard deviation and my program ocassionally deletes all the chromsomes, leading to a null pointer exception.

7 Results

7.1 Genetic Algorithm Results Before implementing dynamic mutation rate and graph of data

I 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

7.2 Genetic Algorithm with dynamic mutation rate and graph of data

(((8.59482714158532 / 0.12743034663208305) * 5.9371079519490095) / 0.0536931183605234) + (((8.930089557028241 * 8.813013817165544 * 9.282327628105781) / 0.2880566114056905) * 8.677481119426583 * 8.745399861997422) = 199 914.503 Target number is 200,000

(2.055574986563656 * 4.335029892851425 * 7.568503498458528 * 9.343999739233327) + (((1.6722547919379975 * 10.789077601745419) / 4.919120586930043) * 7.53127380275555 * (-0.5353799409737934) * 8.800371889538296) = 500.039271 Target number is 500

2.307398166817651/7.995576884793135/3.16972562542193041.2330432667289233+7.280515
7.567277299096906/1.4304429261486098/5.517071133787506=100.000328
target number is 100

0.7597134634863661 - (-0.14867283650832108) + ((6.441531305776507 * 8.303861039429357) / 2.147059039097051) + (3.051381052450619 *

5.809939526621756) + 5.061739125061215 + (4.62427875647421 / 3.5263720470114497) = 49.9227595 target number is 50

(4.556576554235145 / 0.49335793475102296) - 8.160516744230366 - 10.120765459284236 + (6.682040081860103 * 4.464895746516452) + ((6.28639375188405 / (-0.6644120380450653)) / 5.4998318459599735) + 5.94270330865704 = 25.0115352 Target number is 25

((0.7453864418363613 * (-0.3654754093120093)) / 3.746496782926417) - 0.08409066384711861 + ((0.2797259446445456 / 2.659944395374488) * 6.904283877318391 * 1.8173949131194018) + (1.8834446207822981 * 2.0375893412829855) = 5.0004396

Target number is 5

3.685033589723196 - 10.387789333905408 + 0.4213130030207199 - 1.193960461842504 + 0.559596282903128 - 0.012400962661350889 + (0.8529266847944381 * 9.223912879229427) + 8.750641737810112 - 8.683919775648373 = 1.00583551 Target number is 1

7.3 Genetic Algorithm with dynamic mutation rate and graph of data

7.4 Static Mutation

(1.4235296195128524 * 110.12989687505203) + (0.3906144184326358 * 0.8595183365764429) - 0.988322027621972 + ((0.2973975627804106 * 0.44242815385986156) / 0.7140064935764429) + 0.7560777186760231 - 7.589630723762884 = 149.471315 TargetNumber=150

((8.427753236689725 * 7.280445046374003) / 0.43150246412251514) + (4.53457629720121 / 1.1165769688920042) + 0.6496880193449707 + 5.857515152576542 - ((8.46398403132354 / (-2.751053561042653)) * 0.02020405432027738) = 152.826203 TargetNumber=150

7.5 Dynamic Mutation

1.0292804250299605 + ((2.7400839829565777 / 1.0605221382369576) * 7.741718241469524 * 7.316070011134583) - ((-0.9821438516501173) / 4.043144676725913) + (0.9986661909539591 * 6.634650893629718 * 0.3645726822807098) = 150.026535

4.7341236075335615 + (9.72980002599055 * 9.065626755189918) + 5.210579723186854 + ((10.659058594071906 * 9.275004032434559) / 1.878529370205558) + 0.2664659586129581 - (7.913600216706552 / 7.557771160318405) = 149.998593

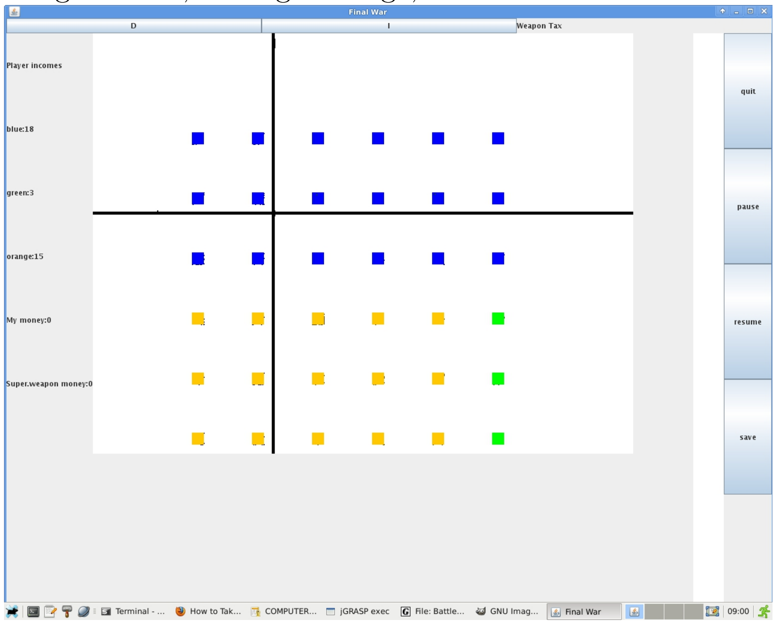
7.6 Genetic Algorithm for Formula Generation

Below are some sample expression for a quadric polynomial with the inputs being multiples of 5 between -5 and 5. The genetic algorithm is unable to recognize anything that is not a linear function.

9*x*1-3*x+10+10*x-3/8*x/5+5*x-3-6*x/2+2*x+6/9*x*3/7*x/10-3*x/9/8*x-7/6*x+2*9

3*x/10*9*x*9/6*x-6-9*x*5*2*x/8/5*x*1/6*x+1-3*x+3/9*x/1/8*x*6+5*x*4+5*x+4-8

Graph of maximum fitness,average fitness, average change, and stan-



Player incomes

blue:18

green:3

orange:15

My money:0

Super.weapon money:0

dard devation over time 3.jpg

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.

7.7 Conclusion

One of the things I learned is that programming soemthing is more difficult than it seems. After sucessfull programming a genetic algorithm to solve the N-queens problem last year, I thought it would be easy to make it generate expressions for a target number. After making expressions for a target number with the genetic algorithm, I thought it would be easy to make the algorithm generate formulas. However, there were are large number of unforseen bugs both times. Also, when generating the formulas, I realized that I needed some additional heuristics in addition to my genetic algorithm.

My project was partially successful as it demonstrated the effectiveness of genetic algorithms in generating expressions. I also had an AI that was slightly more intelligent than the AI that I had at the beginning of the year. However, I was unsucessful in having the genetic algorithm create an AI for the game.

I feel that genetic algorithms are not being used as much as they should even though they have sucessfully solved problems that would be difficult to solve usign traditional heuristics. By creating an AI optimized by a genetic algorithm, I hope to show that genetic algorithms were useful for creating AI for strategy games. I plan to continue working on this project in college.

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

[10] 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

[11] 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

[12] Revello, Timothy.,McCartney, Robert.(2002) *Generating War Game Strategies Using a Genetic Algorithm* Retrieved from http://dynamics.org/ altenber

[13] Wilson, Stewart.(2009) *Coevolution of Pattern Generators and Recognizers* www.illigal.uiuc.edu/