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

April 6, 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. I plan to design a real-time strategy with an intelligent AI that has a combination of heuristic evaluation functions and constants optimzed by a genetic algorithm. 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. Currently, I am designing a simpler version of the genetic algorithm to make sure my genetic algorothm is working correctly. It will become the basis of my new genetic algorithm.

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

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 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 and the genetic algorithm I am working on.

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 generating an expression using the numbers 1 to 9 and the operators -,+,*,/ to 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 expressions can quickly be eliminated, allowing the algorithm to work from combinations that are close to the answer. This makes the algorithm quickly create an expression for a certain number. 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.

Another problem involves a bunch of large disks in a bounded area, and trying to place the largest possible disk within the bounded area. Finding the exact solution using a brute force method or estimations is difficult and not always efficient. However, a genetic algorithm is able to solve this problem relatively easily.

Genetic algorithms have been used to generate strategies for wargames when the creators of the genetic algorithm did not have a precise idea of the most effective strategy. Two researchers used a genetic algorithm to dtermine an effective strategy for a a naval blockadge game that would always be played differently. They also emphasized the fact that genetic algorithms could generate several effective strategies, which would correspond with the different AI personailities that I want to create for my game.

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.

A group of researchers used of genetic algorithms to generate a formula to predict the concentration of carbon-dioxide in the arteries, and was sucessfull. 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 steamrolled 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.

Lancaster's Square Law is a key concept to my game which will be the basis of a heuristic. This gives the relative casualties of two armies based on their relative strength at the end of my 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.

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.

Instead of moving quicly to use the genetic algorithm for my AI, I will spend more time optimizing my genetic algorithm to make sure that there is no stagnation in the fitness values of the population. I will also make sure that my algorithm will output a graph of important statistics so that I can analyze the sucess of my genetic algorithm and look for problems such as stagnation or an inability to consistently produce an effective result. I will be easier to fix fundamental problems with my genetic algorithm now rather than later. Afterwards, I will modify my genetic algorithm so it takes input-output pairs based on common functions such as quadratic polynomials and generates a formual for the outputs based on the inputs.

Unlike I initially planned, I will not have my genetic algoritm test out the fitness values of the AI's by having them play against each other.Having the AI's play against each other will take too much time. This is fine if the algorithm is only running once. However, there will inevitably be a problem, which means that my genetic algorithm will have to run several times.

Instead, I will create a series of scenarios and precisely define the owners of each city, and each player's troop deployment. I will then pick a player and define actions that would be good for it and actions that would be bad for it. The AI's heuristic combinations will be tested on my scenarios to see what actions they will take. If their heuristic suggests an action that I consider bad, they will have their fitness lowered. If the heuristic suggests an action that I consider good, their fitness will increase. I will generally create scenarios with obvious good and bad actions that an AI will take to reduce any misconceptions I may have about the strategy for my game.

Currently, I have created a genetic algorithm that generates an expression for a target number using random double values between 0 and 10, functions on these random numbers, and several mathematical operators. I have also implemented a dynamic mutation rate of the population based on a heuristic evalution of how much the fitness values of the population is stagnating. While I was analyzing the effectiveness of my genetic algorithm, I realized that I was frequenly analyzing the trend of the fitness values in textual form or by converting the data into a graph manually. As a result, I decided that I needed my program to automatically graph important parameters such as the total fitness of the population, the average fitness of the population, and the standard devation of the fitness values.

5.2 Timeline

- April:
  - Week 1: Modify genetic algorithm so that it consmosome'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.

    istently works without crashing. I will also make sure that the standard devation of the fitness values does not stagnate at 0 after the beginning generations.
  - Week 2: Modify genetic algorithm so that it can create formulas for input output pairs corresponding to some function.
  - Week 3: Brainstorm scenarios that I can use to test the genetic algorithm for my games AI and save the scenarios as data that is readable by my genetic algorithm.
- May

  - Week 1: Create the genetic algorithm that will be used to optimze my AI(only the scenarios, AI heuristics, and genetic algorithm will be needed)
  - Week 2: Run the genetic algorithm. While the genetic algorithm is running, I will create a AI class for my game that runs a heuristic created by the genetic algorithm.
  - Week 3: Play against the AI and analyze its flaws. Afterwards, modify genetic algorithm and run it again. While it is running, create several new scenarios for the AI to play out to see if it has improved.
  - Week 4: Play against AI against, analyze its flaws. Afterwards, modify and run genetic algorithm again

- June
  - Week 1:Implement code so that the AI will observe the behaviour of other AI's and take advantage of its observations.
  - 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

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

5.7 Genetic Algorithm

5.8 Data Graph

Legend for graph

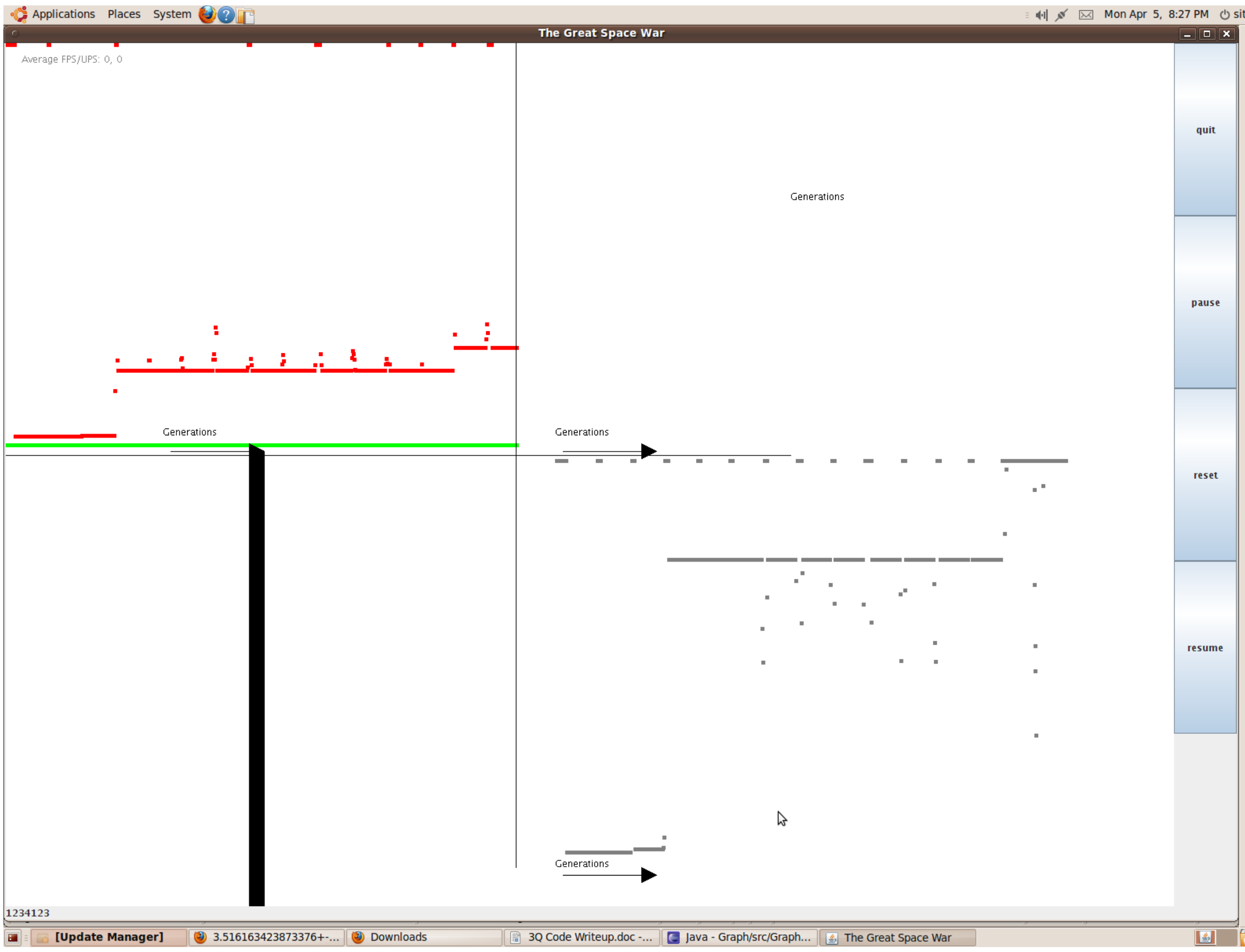
- Red-Highest fitness

- Gray-Average Fitness

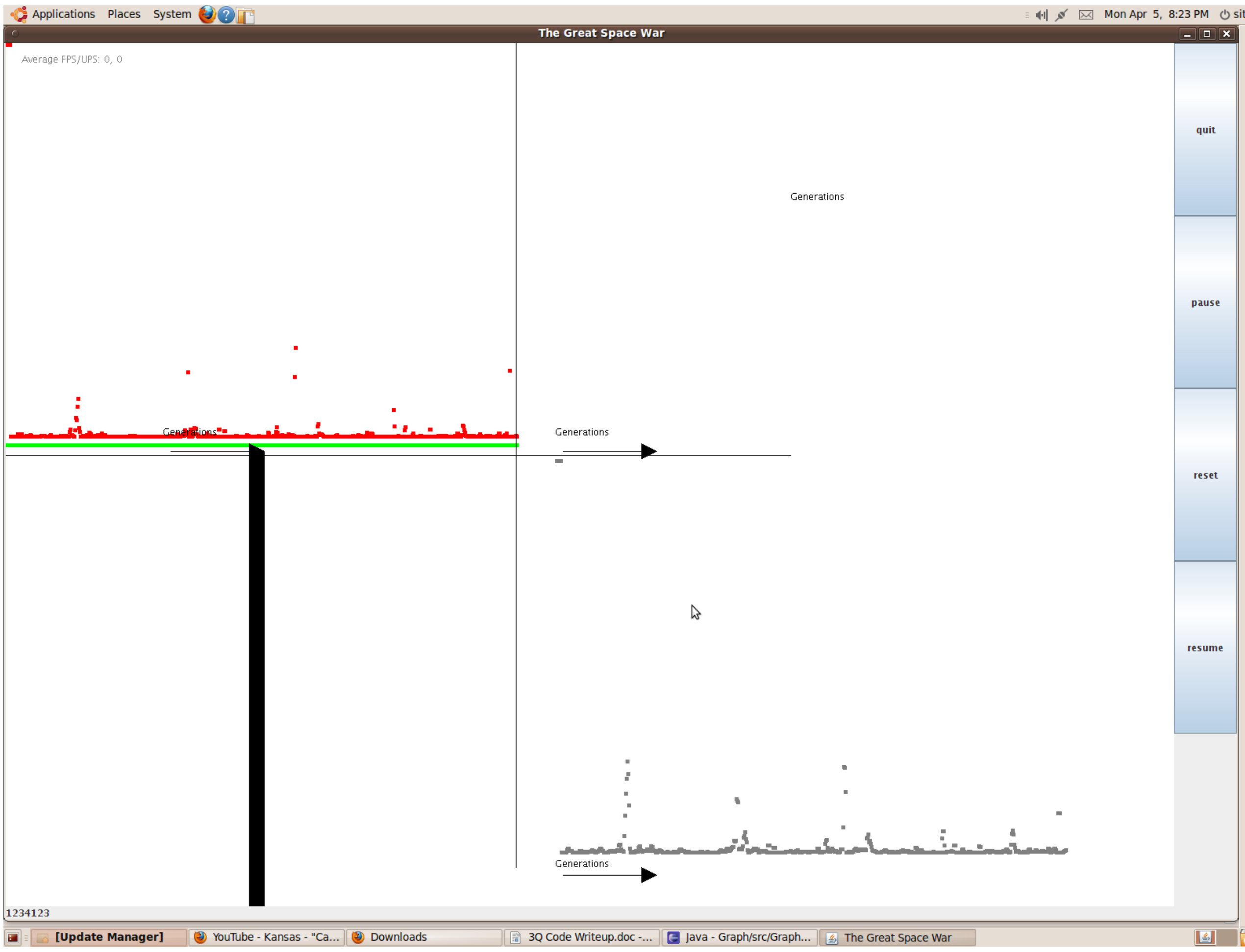
- Green-Standard Deviation

Values to the right are later values. The higher the graph, the higher the value. The line for the standard deviation is in the middle due to a bug that makes the standard devation converge to 0.

Dynamic Mutation



Static Mutation



5.9 Sample expressions with static mutation

TargetNumber=150

3.516163423873376 + (((-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)

5.10 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.

5.11 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.

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+7.28051566
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

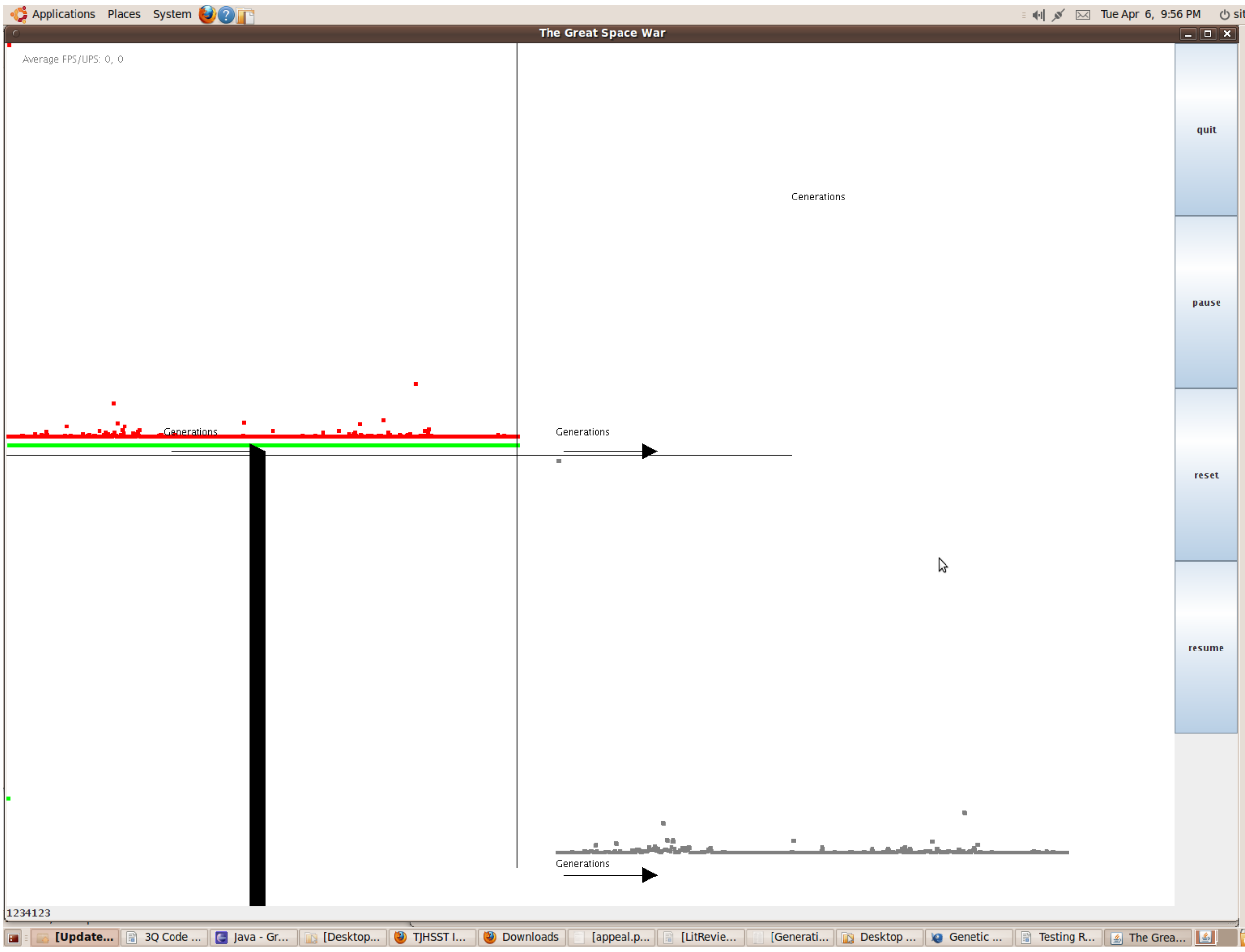
3.685033589723196 - 10.387789333905408 + 0.4213130030207199 - 1.193960461842504 + 0.559596282903128 - 0.012400962661350889 + (0.8529266847944381 * 9.223912879229427) + 8.750641737810112 - 8.683919775648373 = 1.00583551 Target number is 1

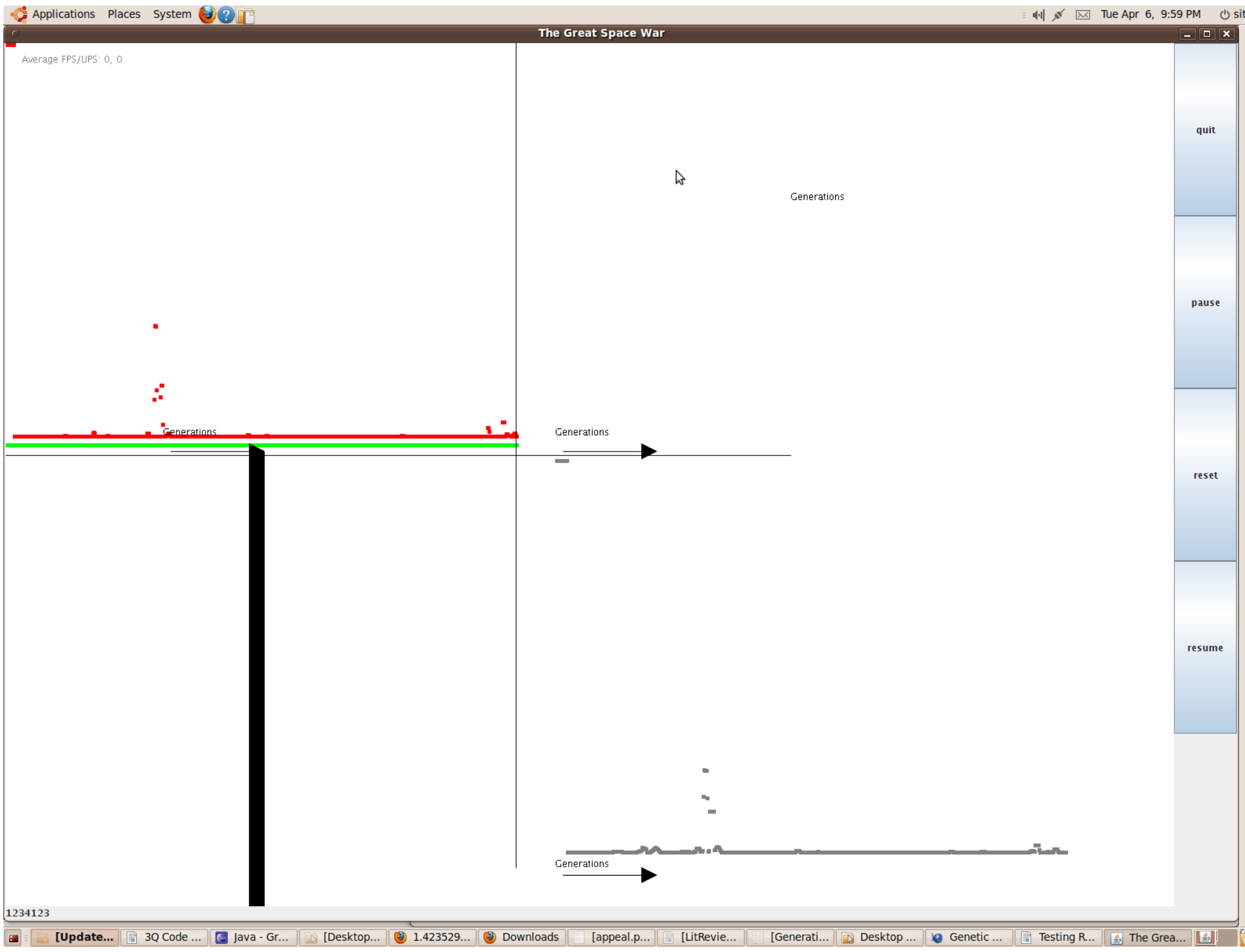
7.3 Genetic Algorithm with dynamic mutation rate and graph of data

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


Applications
Places
System
Tue Apr 6, 9:56 PM
The Great Space War
Average FPS/UPS: 0, 0
Generations
Generations
Generations
Generations
quit
pause
reset
resume
1234123
[Update...
3Q Code ...
Java - Gr...
[Desktop...
TJHSST I...
Downloads
[appeal.p...
[LitRevie...
[Generati...
Desktop ...
Genetic ...
Testing R...
The Grea...

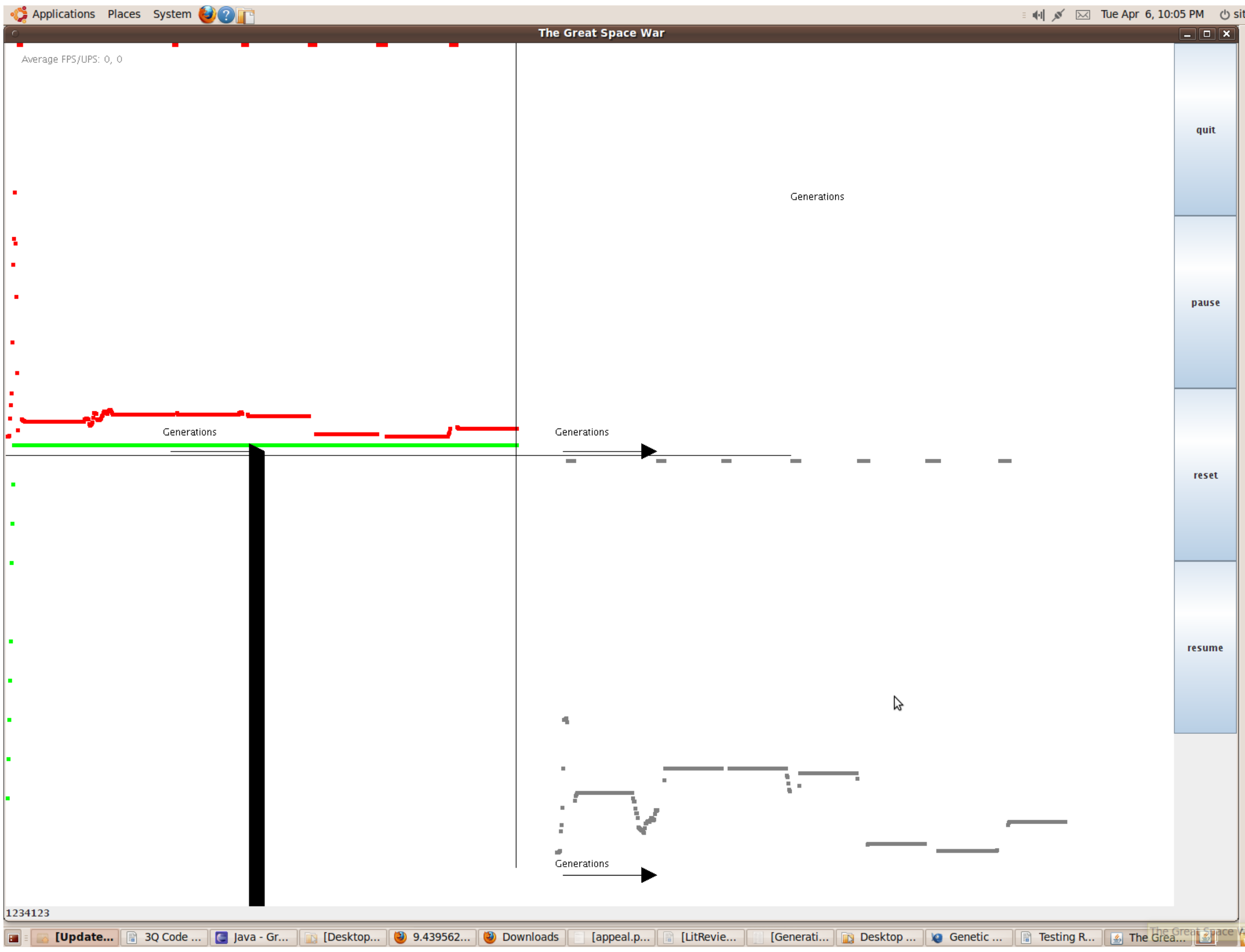


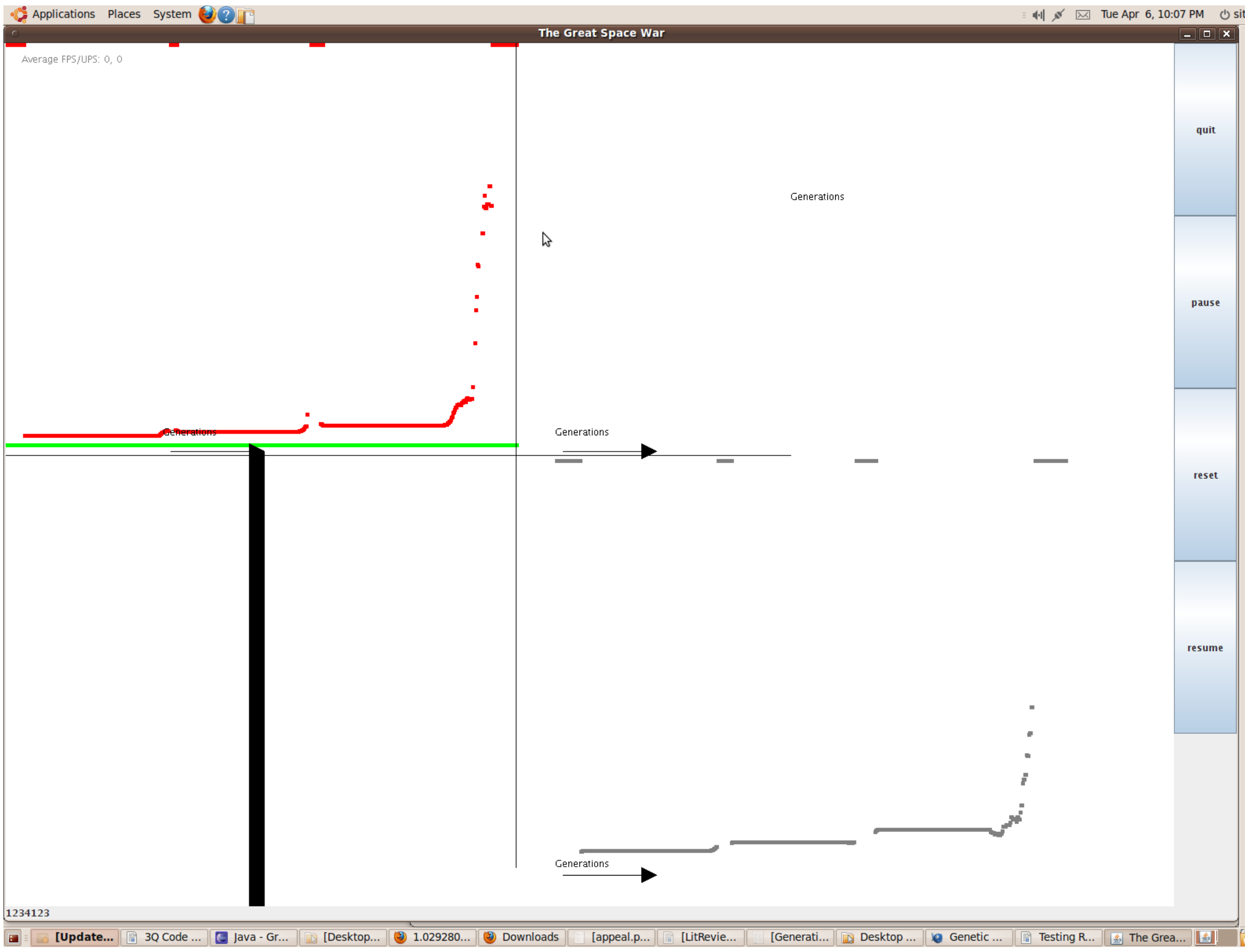
These graphs show that the fitness rates are not stagnating because of the high amount of variability. The reason why the variations do not seem signficant is because the fitness rates randomly spiked up to a very high level in the beginning. Even though it may seem bad that the genetic algorithm moved away from a fit solution, it is possible that the solution may have been a local maximum rather than the most optimal answer.

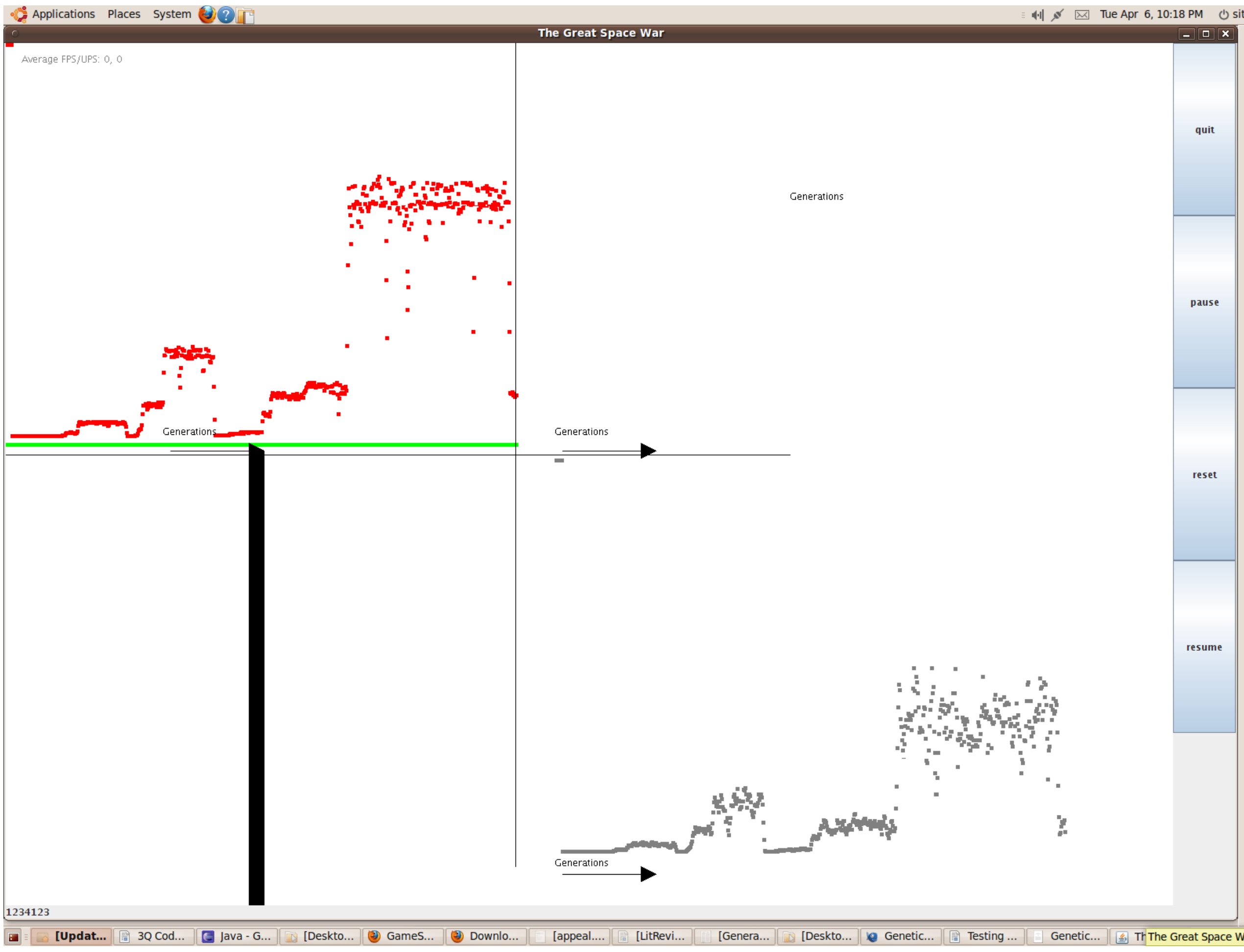
7.5 Dynamic Mutation

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


Applications
Places
System
Tue Apr 6, 10:05 PM
The Great Space War
Average FPS/UPS: 0, 0
Generations
Generations
Generations
Generations
quit
pause
reset
resume
1234123
[Update...
3Q Code ...
Java - Gr...
[Desktop...
9.439562...
Downloads
[appeal.p...
[LitRevie...
[Generati...
Desktop ...
Genetic ...
Testing R...
The Grea...


Applications
Places
System
Tue Apr 6, 10:07 PM
The Great Space War
Average FPS/UPS: 0, 0
Generations
Generations
Generations
Generations
quit
pause
reset
resume
1234123
[Update...
3Q Code ...
Java - Gr...
[Desktop...
1.029280...
Downloads
[appeal.p...
[LitRevie...
[Generati...
Desktop ...
Genetic ...
Testing R...
The Grea...



These graphs also indicate a spike in fitness at the beginning followed by a rapid decline. Although the dynamic mutation allowed the chromosomes to depart from a local maximum, there were periods where the fitness rate remained nearly constant, which shows stangation for the first two graphs. However, the third graph shows that there was a great deal of variability in the fitness rates.

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

[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