Humza Tahir: htahir@college.harvard.edu
Raahil Sha: raahilsha@college.harvard.edu

Llama Farm

Demo Video

Overview

Using a genetic algorithm and finite state machines, we implemented a world of llamas. Given an initial farm environment and an initial gene pool, llamas compete with each other for resources, with some llamas dying every generation, either through murder or starvation. The llamas that survive each generation pass on their genes to the next generation of llamas (with mutations and meiosis along the way), over time creating a population of llamas fit for survival in their given environment.

Planning

We brainstormed the project over breakfast in Annenberg after watching a YouTube video about stylish and violent llamas. Inspired by this, we discussed various genes (aggression, style, hunger drive, laziness, etc.) that we could use as well as various environmental factors that llamas would encounter (food, land, hazard, etc.). We sat down one day and produced a complete project specification. We made very few changes to our additional specification (namely, annotating what we actually ended up accomplishing, as well as accounting for the addition of the Parameter class and the Tile class). Our final project specification can be found at this link.

Algorithm Implementation

In our genetic algorithm, we strove to simulate mammalian genetic reproduction very closely. To start, we first created a basic gene that could be represented as a series of zeros and ones. Each gene contains two separate strands that are independent from each other (unlike in normal DNA). After modeling DNA, we created a meiosis algorithm that performed crossing over of the two strands of DNA, as well as causing mutations in the genome (small random changes to genes). A mutation was programmed in as a random bit flip of one point of the genome. The full specification of our genetic algorithm can be read within our final project specification.

The next main algorithm of our project was the Finite State Machine. Each llama in the world starts out in the idle state, but chooses which state to transition to using probabilities defined by the llama's genome. For example, a highly lazy llama may not leave the idle state for a long time. A diagram of the Finite State Machine can be found at this link on our GitHub. Behaviors were programmed in for each of the states within the Finite State Machine.

Alongside our main algorithms, we had to create a visualization for the simulation that would be taking place. To do this, we used the Java EE Swing API, which allows us to create windows that can be viewed as an actual application. We implemented a double-buffering algorithm when drawing the window to avoid flickering when the frame rate is inconsistent. We also used basic multithreading to run our Application.

Roadmaps and Milestones

Quite surprisingly, all of our checkpoints were completed almost exactly as planned. For the functionality checkpoint, we had finished implementing all of the four tasks we set for ourselves (Llama.java, GameWorld.java, Tile.java, GamePanel.java complete). The week after that, we

finished implementing all of the rest of the classes our project needed to function. During that week, we added a new class called Parameters.java, with static variables for every parameter that the user may want to change before running the simulation. We also implemented one of our advanced extensions the weekend after that (Changing environment within a simulation). The last week, we finished all of our bugfixing, then completed our other advanced extension (Method to track genes over the course of the simulation). We did this using R, a programming language mainly used for statistical analysis. We had our main Java program output log files in CSV format of the progress of the simulation, then analyzed these CSVs in R using an automated script to generate charts. Using the remainder of our time, we ran various experiments on our farm to analyze what llamas would do over the course of a simulation given an environment.

Contributions

Humza worked on the primary research behind the project, project design, as well as the documentation writeup. Raahil worked primarily with statistical analysis, bug fixing, and most of the Java Application (Java EE Swing) coding. To program the rest of the Java classes, both of us sat together and worked to build the classes at the same time.

Development Experience

Since we used Eclipse as an IDE, we were able to bypass most of the headaches commonly involved with Java development, as Eclipse set everything up for us, such as the classpath and compile directories. The object-orientated side of Java was very useful in our projects, as we could easily create complicated structures such as a World with an arraylist of Llamas, each of which has an array of genomes within it.

We had minimal difficulties from a development standpoint, as our project didn't specifically call for us to use functional programming at any point. Debugging our code was also relatively simple using Eclipse's built-in Java debugger, and our main code signatures didn't particularly change from the time we handed in our functional specification and our completed product.

The Sad Stuff

To make our project as awesome as possible, we unfortunately had to make a few concessions. Our original plan was to design graphics for each llama. Ultimately, we realized that we should increase the functionality of our algorithms rather than the visual appeal of our ultimate GUI. We had a plan to have a second Window that would interact with the main simulation window to allow the user to modify the simulation in the middle. In the end, we realized that this was just something aesthetic and implemented an analog to such a window using the command line.

If we were to tell future students the one thing we learned from this project, it would be to completely finish all algorithms and backend-related code before focusing on creating a polished front-end/GUI. Whereas a flawed interface may be easily rectified, errors in algorithms are much more likely to lead the project to failure. Thus, we would have probably focused more on getting the Finite State Machine and Genetic algorithms out of the way if we were to create a project similar to this one in the future.

One other small change that we would make to make our program better is to have used a Boolean array for the genome, as opposed to the strings that we are using now. Using a

Humza Tahir: htahir@college.harvard.edu
Raahil Sha: raahilsha@college.harvard.edu

Boolean array would have most likely saved a lot of RAM that is used to hold each Llama's genome.

If Only We Had More Time

There are two primary tasks that we would complete if we had more time to work on the project. The first is to implement a graphical menu to allow the user to change world parameters before starting the simulation and to change these parameters in the middle of the simulation. Currently, we do this by making the user manually edit the Parameters.java file and building from source to change parameters before the simulation starts. To change parameters during the middle of the simulation, we simply prompt the users to enter numbers within the command line.

The second (a little bit more exciting) task is to use a genetic algorithm for our plant tiles. As stated in our project specification, we would do the following. Instead of having food simply appear each generation based on environment parameters, the plants that make up the food species would decide how to multiply based on their genetic programming. This would be affected by the environment, as well as the plants' interactions with the llamas.

Other Advice

GitHub is amazing as a version control system. We used GitHub to successfully revert our meiosis code back to a working function after we had accidentally messed it up.

Keep logs of all data and experiments done. In our project, we print out a CSV data file representing gene expression over time, as well as a TXT file that contains information on the parameters being passed to the function as well as experimenter notes on the experiment being performed.

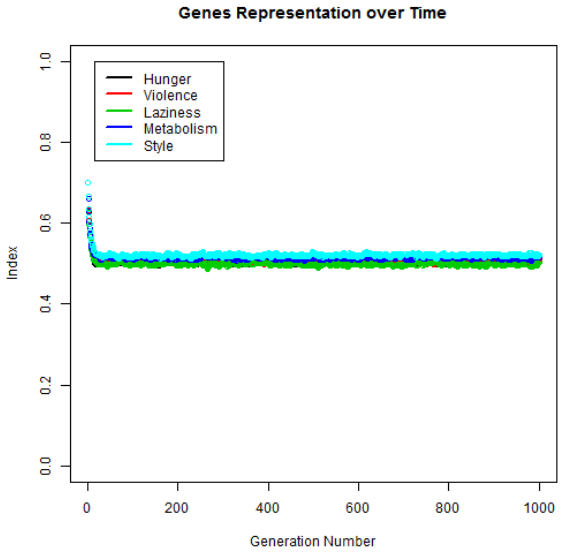
Humza Tahir: htahir@college.harvard.edu
Raahil Sha: raahilsha@college.harvard.edu

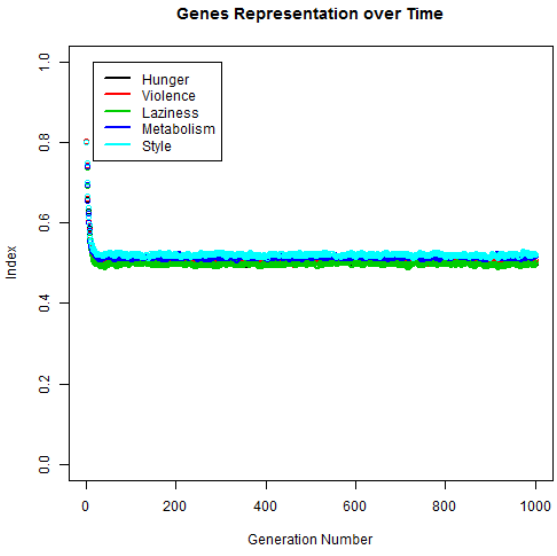
Experiments!

In this section, we simply have a few interesting results that we found while performing experiments using our llama farm.

Experiment 1: The Problem with Mutation Rate

Having a high mutation rate will always force gene representation to approximately 50% for every single gene, as an average for a high bit flip rate will slowly progress to 0.5.

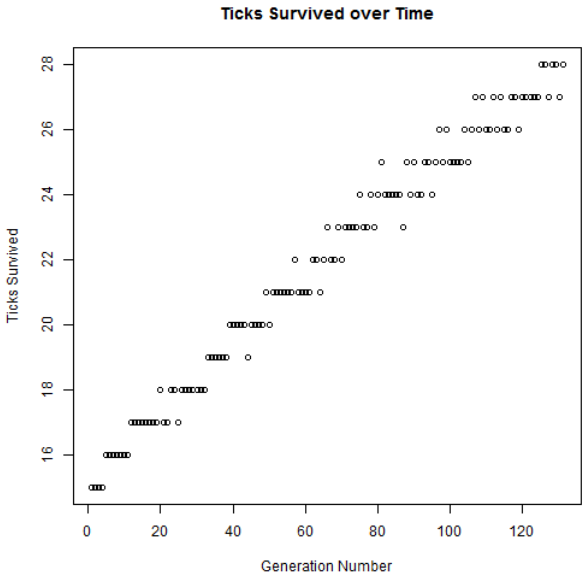


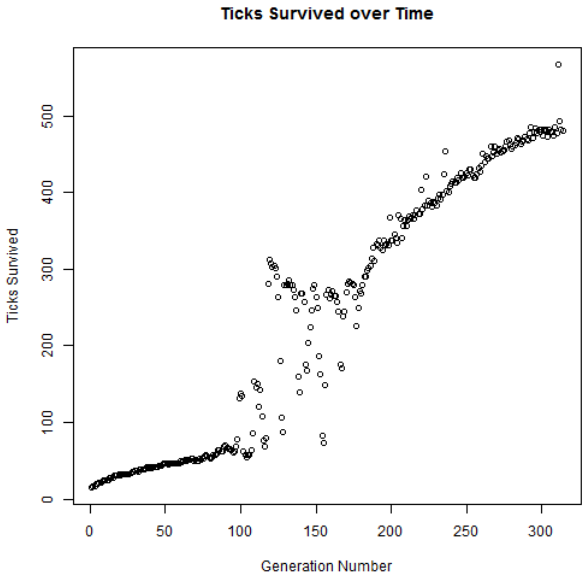


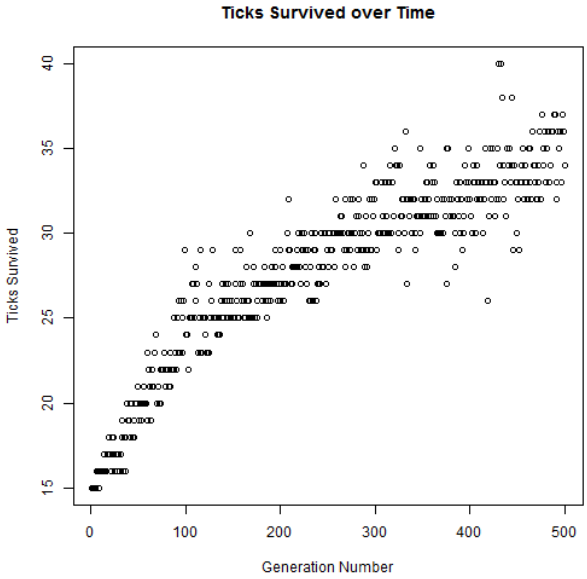
Humza Tahir: htahir@college.harvard.edu
Raahil Sha: raahilsha@college.harvard.edu

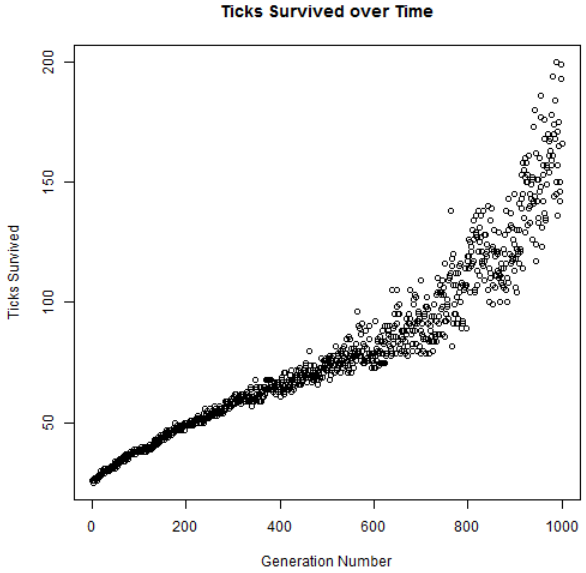
Experiment 2: Llama Fitness over Time

As a simulation progresses, the llamas will become more fit over time, allowing them to survive longer before reaching the threshold to create a new generation of llamas.





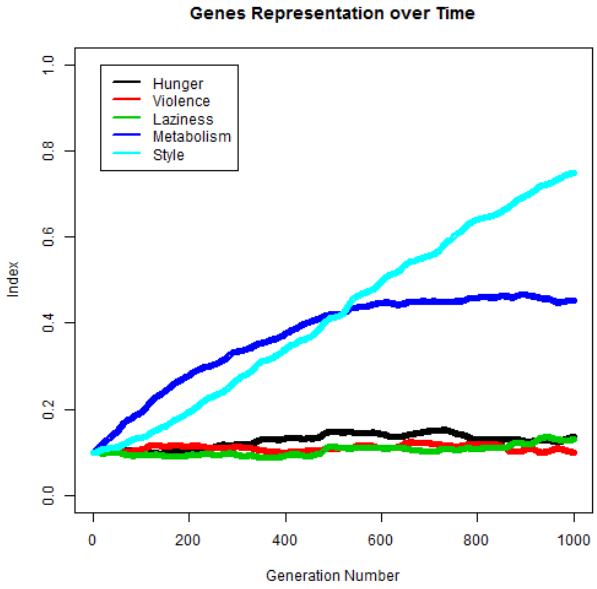


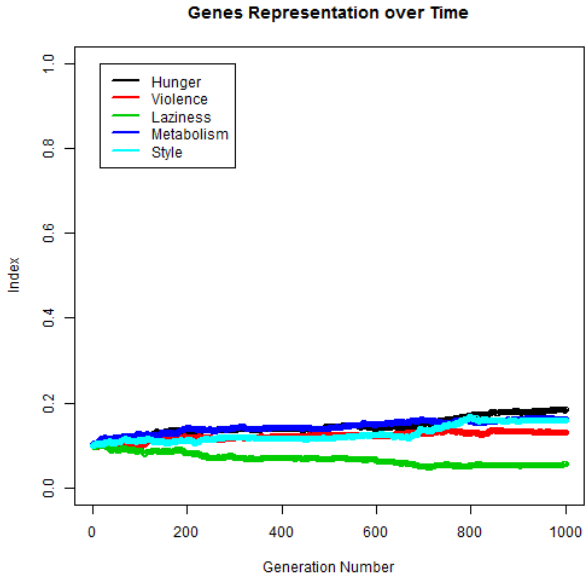


Humza Tahir: htahir@college.harvard.edu
Raahil Sha: raahilsha@college.harvard.edu

Experiment 3: Llama Stylishness

The more llamas there are, the quicker evolutionary pressure will select for stylishly fashionable llamas. The diagram below on the left has 750 llamas. The diagram on the right only was a simulation for 50 llamas. Style increased a lot more quickly when there were more llamas. Metabolism also rose a lot faster, most likely because competition for food resources was higher in the diagram on the left.





Humza Tahir: htahir@college.harvard.edu
Raahil Sha: raahilsha@college.harvard.edu

Experiment 4: A Sudden Drought

In the diagrams below, a drought began to occur at generation 500. The drought then disappeared at generation 750. You can see a significant drop in average lifespan once the drought began. The llamas never actually evolved to overcome the drought conditions, but were able to regain some of their former longevity after the end of the drought. One interesting note is that the gene levels seem to quickly stabilize after the drought occurred.