**Genetic Algorithms Final Project**

Between the two of us, the following things were implemented into our final project:

* A method, both for strings of text and also images, that generates a certain number of random population members, either a specified number of times(number of generations), or until our random generations were able to output the desired image or string of text.
* An efficient method of determining the fitness of that “population member” (how alike it is to the input target phrase/image) by using logical statements and the total number of elements in the given target phrase or image.
* Implementing some way of choosing two random parents from a mating pool by determining that parents fitness (how alike it is to the target), and giving more fit parents more “tickets” for the mating raffle. Every ticket for the parent is the index number of that parent in the current population of images or strings. A mating factor determined how many tickets a parents would get by multiplying that parents fitness by the input mating factor, where a higher mating factor lead to more tickets for a more fit parent. The more tickets a parent had, the more likely it would be for the index of that parent to be randomly chosen from the mating pool full of the indexes for different members of that population.
* Breeding the two parents or strings that were randomly chosen from the mating pool by creating a vector of random ones and twos that is the same length as the number of elements in the parent, and using a for loop, go through the vector of random ones and twos, and when there’s a one, the ith element of the child is the ith element of the first parent, and when there’s a two, the ith element of the child is the ith element of the second parent, and by going through the entire vector of random ones and twos, you will breed a child from the two parents.
* The DNA mutation is determined by the input chance (in a percentage) that a DNA mutation will occur. If a 1% is entered, then a vector of 100 integers will be created, and a random element from that vector will be chosen. A for loop goes through every element of the child, and for every iteration a random element will be chosen from your mutation vector. If that randomly chosen element is a one, then the ith element of the child gets changed to a random character, either a random upper case letter, lower case letter, or a space. For the images, the random mutation changes the ith pixel of the child image to some other random value in a specified range of values input by the user. There is also another chance that if a pixel is chosen to be randomly, there is a chance that that randomly mutated pixel is mutated to a completely random value instead of something inside the specified mutation range.
* Later on an exponential factor is introduced, which can greatly decrease your run time in returning a relatively or completely fit child. This exponential factor raises each value in the population fitness vector to some power, making the difference between different fitness values much greater, and then normalizing them again. Now that the difference between more and less fit members of the population is much greater, more fit member of the population will get MANY more tickets than the less fit population members, and the more fit members will be much ore likely to be chosen to breed a child.
* Fitness and mutation of images is very similar to the methods used for strings, with the exception of an additional mutation component for the images that gives it the possibility of a completely rando mutation as described above. The fitness of the images also isn’t determined by complete match of values to the parent image, but instead to a match of the values in the parent image plus or minus some amount of tolerance.

This project really taught me how to think creatively in order to maximize efficiency of a script or program. For example, I learned how very time costly nested for loops are when you deal with large amounts of data or large matrices of values. It was really interesting to experiment with different methods of eliminating nested for loops to improve efficiency and run time of the program. This project taught me the fundamental ideas behind how a population of organisms reproduce, mutate, and change over time, and this program allows you to see those drastic changes that would normally occur over vast amounts of time in the real world. In general, the larger the population size, mating factor, and exponential was, the quicker the desired output was achieved. Lowering the chance of genetic mutation and restricting the number of characters the random mutation could incur also helped optimized desired output.

Feedback for Julian:

Things we thought were well explained were the methods used to build the initial population, calculate the fitness of each population, and how to breed two parent organisms. The task that were a little more challenging to carry out was the mating pool. Just the way that it is worded made it a little confusing to me with the whole raffle idea behind the method of creating the mating pool. It really helped with the way you explained it in office hours where the tickets in the raffle were the indexes of the parent, and how a more fit parent would receive more tickets by being multiplied by the mating factor, and how a more fit parent gets more of its indexes added to the mating pool, making it more likely to be randomly chosen to breed a child.