Part III: Write-up for Evolution:

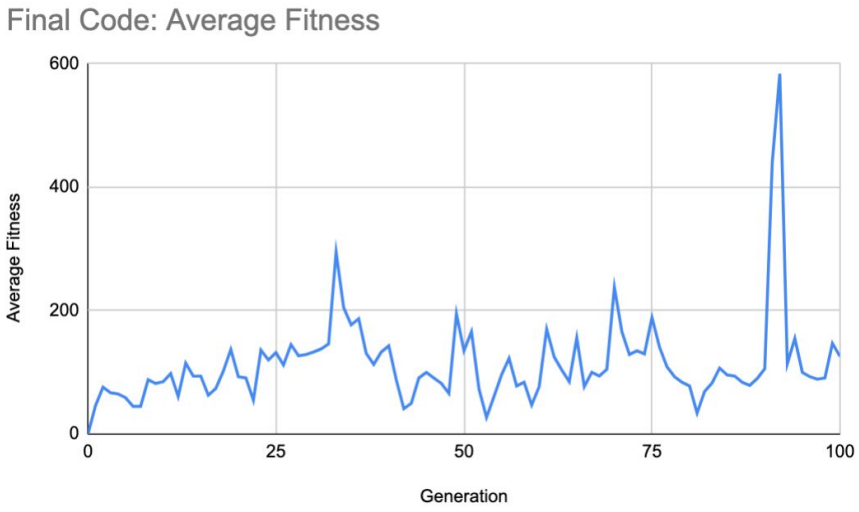
1. The optimal configuration I found for my Evolution's training parameters is as follows:

- Activation Function: sigmoid function
- Mutation rate: 2%
    - Mutating by increasing .15 to syn0 and syn1 values
    - Randomizing if increase leads to value being out of bounds of -1 and 1
- Selection Rate:
    - 25 Birds take the 1st Best Bird's NN
    - 15 Birds take the 2nd Best Bird's NN
    - 10 Birds take the 3rd Best Bird's NN
- Did not normalize (explained more below)

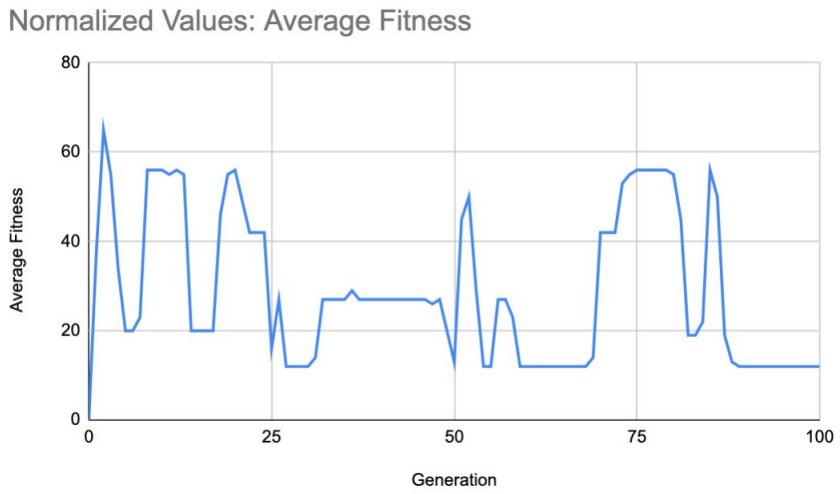
2. My part 1 design is straightforward. I had 6 main classes: App, PO, Game, Bird, Pipe, and Constants each handling their own code. I had certain methods handled by certain classes that it pertained to such as the Bird class holding methods for updating and pipes for generating a specific pipe. This ensures that when the birds need to be updated or pipes need to be generated, the Game class doesn't need to hold all the code. I also had an updateBird method in Game to check whether the game was over, which I then implemented into an updateEvolution method that updated the Population and also checked whether a certain bird was dead.
   I modeled the Population using 2 different types of data structures: ArrayLists and Stacks. I used two ArrayLists, one to store all the alive birds and one to store the best birds and a stack for all the dead birds. A stack was useful because then when I bird died, I could pop the top three birds that died the latest to add to the best birds ArrayLists since those birds were the "best" in the population. I removed birds from the ArrayLists by killing them graphically and logically (kill is a method in Bird).
   I designed the algorithm to train the birds feasible by having a separate NeuralNetwork (NN) class to handle all the ML logic and included "getters" to get the information needed to pass on or manipulate. I also have a Population class to handle all the birds at once to I can call certain methods like mutate and getNN in order to let the birds learn. Finally, both Bird and NN had overloaded constructors in order for the next bird to have the important information of the previous bird (like the weights of syn0 and syn1).
3. The major changes I made to optimize my algorithm were related to the Selection, Mutation, and Activation Functions. I first tried changing the Activation Functions, specifically testing out three types: sigmoid, tanh, and ReLU. What I learned later was that simply changing the functions would not change much because, as Mikey mentioned: "We are not using them to calculate gradients." I have my plotted data for both tanh and ReLU functions but what I learned was that changing the function worsened the learning of my birds, therefore I did not implement either and stuck to the sigmoid function.
   The next optimization I tried was with the selection method. I tried changing the number of birds in the next generation that had the NN of the second-best bird. Instead of the ratio 25:15:10, I switched it to 25:20:5. The convergence of the algorithm changed in that

the birds learned very quickly but immediately forgot what they learned and became dumb again. Then immediately it would learn again and then become dumb. This constant up and down led to worse optimization so it was not included.
Then, I tried optimizing my mutation method. First, I changed the mutation method to mutate 5% instead of 2%, increasing the mutation rate. This caused the birds to stay dumb for a while, but then out of the blue, the birds became very smart and their average fitness went up to the 1000s. But, that was followed by a sharp decrease and the birds stopped learning again. Therefore, the increase in the mutation rate was not implemented. Second, I changed the mutation value. Instead of increasing, I tried decreasing the value by 0.15, which led to good fitness in the beginning, but then slowly decreasing fitness. This was also not implemented.
Finally, after many people mentioned normalizing would help the birds learn, I tried normalizing my input nodes to ensure the values were between 0 and 1. Yet, after I normalized the input nodes, my birds were just as unstable when learning. There was no trend with the average fitness, but there were times when the fitness was constant, but then sharply decreases. Therefore, the nodes were not normalized due to this issue.
Although the final optimal configuration does not learn as best as it could (and in fact, also unlearns as well), I tried a majority of the suggestions for optimization suggested in the Handout. Other methods could have been implemented like incorporating the distance to a pipe when dead to the fitness (fitness is currently found by how long a bird is alive for) or cross-breeding, but due to time and other errors with code (explained in part 5), these other methods of optimization could not be implemented.

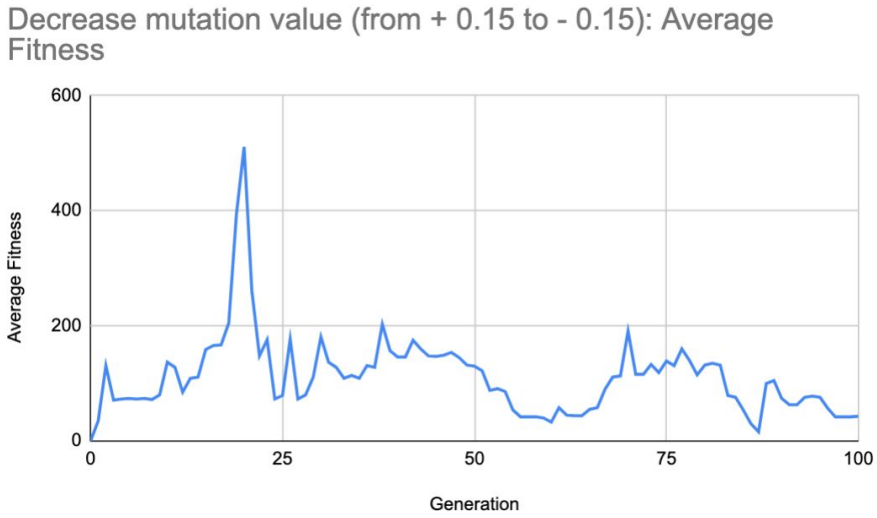
4. Here are the visuals of all the data of different optimizations that were tried from part 3. Each graph shows 100 generations and the average fitness of the birds from each generation. Most of these generations had at least one bird reach the max fitness (4001) at one point, but not all birds could reach it.



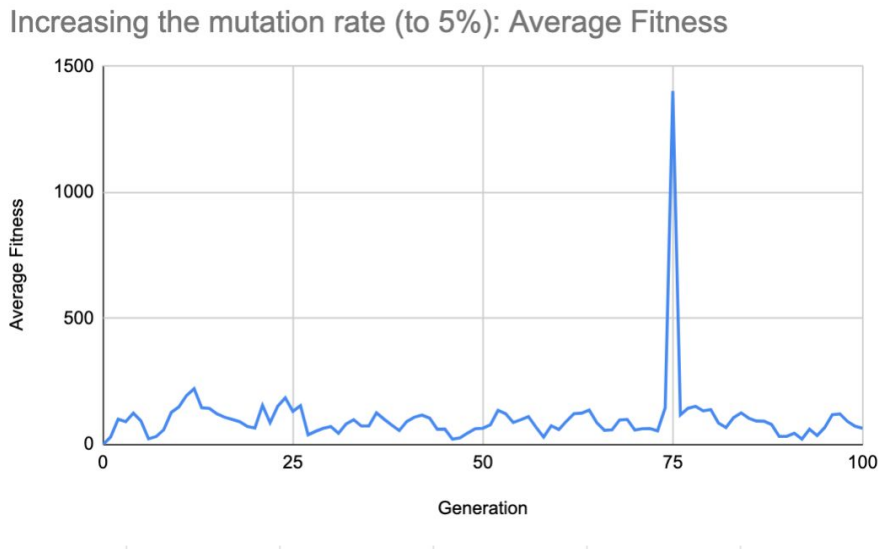
1: Data for the final optimizations (from part 1) - The population increases somewhat steadily, but do not learn as efficiently as possible. This was the most efficient method compared to the others shown below)



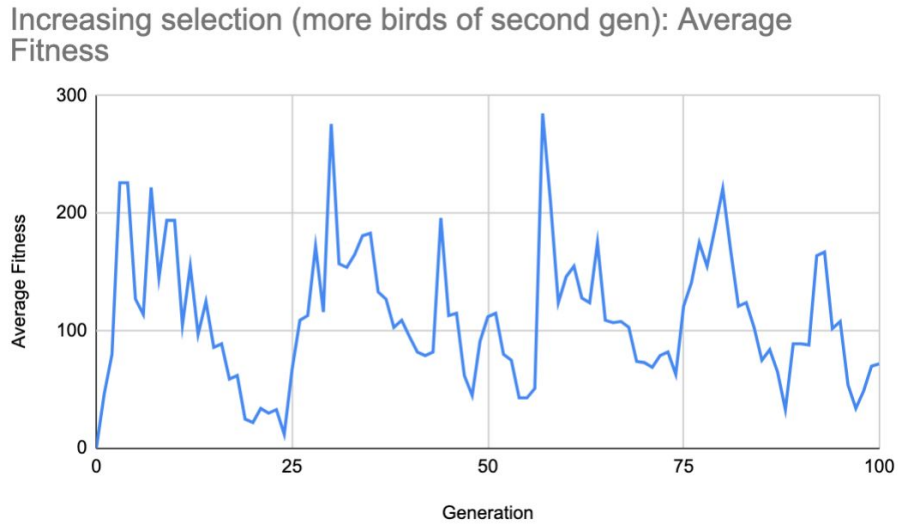
2: Data for normalized values - The population increases and decreases quickly with some plateaus.



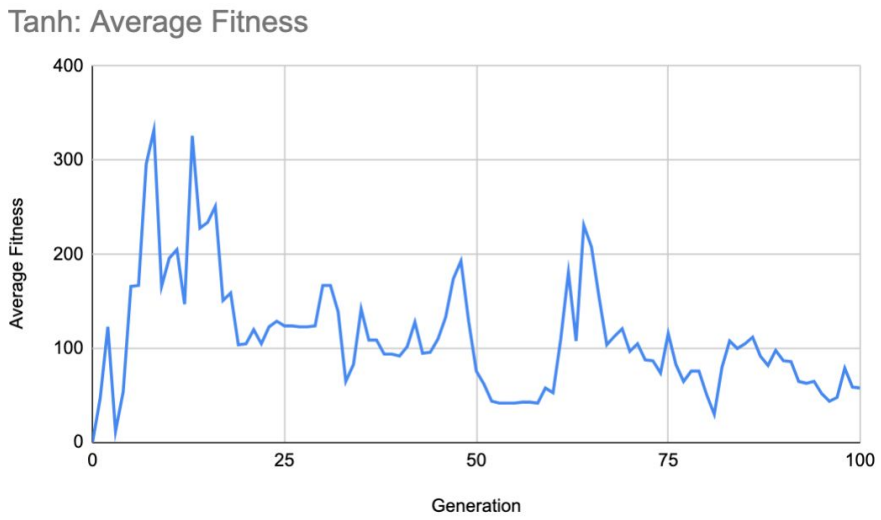
3: Data for decrease mutation value - The population increases before 25 gens but begins a decline with little learning afterward.



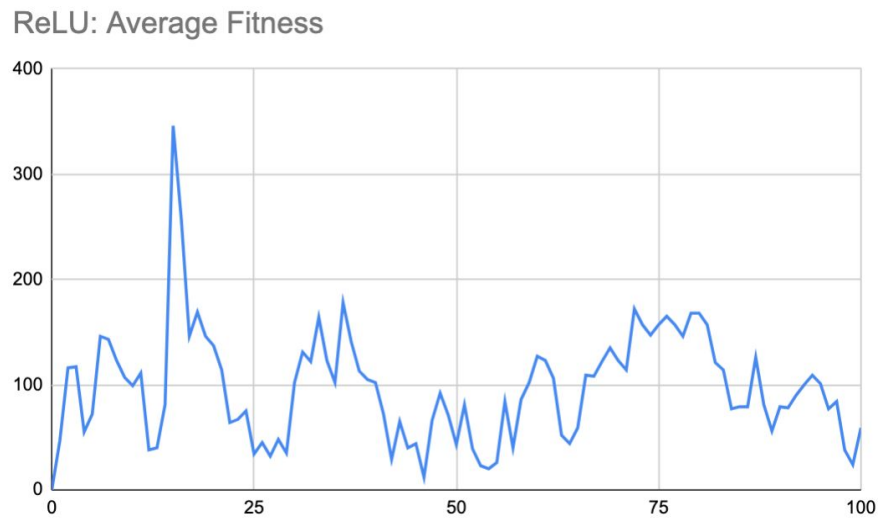
4: Data for increasing mutation rate - The population doesn't do great but then all of a sudden they do amazing (at around the 75th generation), but then they go back to being dumb.



5: Data for increasing selection - The population quickly does good and then decreases steadily, all done in chunks of around 25 generations.



6: Data for tanh - The population learns quickly, but then begins to decrease with some learning. Learning is not constant.



7: Data for ReLU - The population's fitness increases and decreases constantly, but no constant upward fitness.

Containment Diagram: