Exploring Genetic Algorithms to Solve Simple Games

Team Snek:

Abhishek Hanchate, Rylan Hunter, George Mikhaeil, Austin White

Texas A&M University

I. INTRODUCTION

A. Genetic Algorithm

A genetic algorithm is a metaheuristic inspired by Charles Darwin's theory of natural selection. It belongs to a larger class of evolutionary algorithms. The fittest individuals are selected for reproduction in order to produce offspring of the next generation. The process of natural selection starts with the selection of the fittest individuals from a population which then produce offspring which inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than them and have a better chance of survival. This process of selection based on fitness keeps on iterating and at the end, a generation with the most adapt or fittest individuals will be found. A similar notion is applicable to search algorithms where we consider a set of solutions and try to find out the best ones out of them.

B. Benefits of Genetic Algorithm

Genetic Algorithms are mostly used in situations where one can rely on generating high-quality suboptimal solutions to optimization and search problems.

C. Pygame

Originally written by Pete Shinners, Pygame is a cross-platform project of several Python modules designed for building video games. It consists of computer graphics and sound libraries which are made compatible to run with the Python programming language.

II. FLAPPY BIRD

A. Game description

Flappy Bird is a side scrolling mobile game, the objective is to direct a flying bird continuously through a series of pipes. If the user touches a pipe or the ground, they lose. The bird "flaps" upwards each time the user taps the screen.

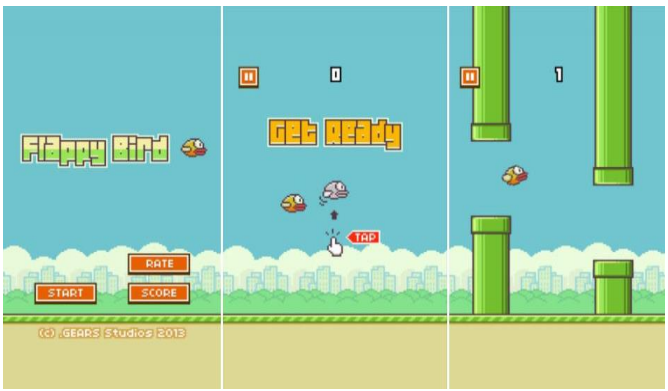

Fig. Flappy Bird Game

B. Why use a Genetic Algorithm for the problem?

After learning and implementing a genetic algorithm in the game Smart-Rockets, we decided to implement a similar type of genetic algorithm into a real game. Flappy Bird was an easy choice as it was fairly simple and only

had a few parameters to input into a predictive function. The goal of the genetic algorithm was to learn from previous mistakes and attempt to "flap" a bird as far as possible without hitting any obstacles.

C. Parameter descriptions

To create the genetic algorithm, we needed to build a neural network that would predict the "DNA" traits needed as well as define a "fitness" function to apply a weight to the most successful generation. The DNA traits for the inputs to the neural network were defined as a series of horizontal and vertical distances away from the pipes. These inputs were as follows:

- Horizontal distance from left of the bird to the right of the pipe
- Horizontal distance from right of the bird to the left of the pipe
- Vertical distance from the top of the bird to bottom of the top pipe
- Vertical distance from the bottom of the bird to top of the bottom pipe
- Vertical distance from the bottom of the bird to the ground
- Vertical velocity of the bird
- Bias node (constant input of 1)

We determined that all of these inputs into a simple neural network would be sufficient in determining whether or not the bird was needed to "flap" to pass the pipe. The neural network chosen had these 7 inputs to 1 single output without any hidden layers. The fitness algorithm used was determined using the score of the current bird and the number of frames alive.

$$fitness = (score)^2 + \frac{(Number\ of\ Frames\ Alive)}{20}$$

This ensured that a bird with a longer lifespan would have a significantly higher fitness and we applied this fitness weight to the next generation after a mutation and crossover.

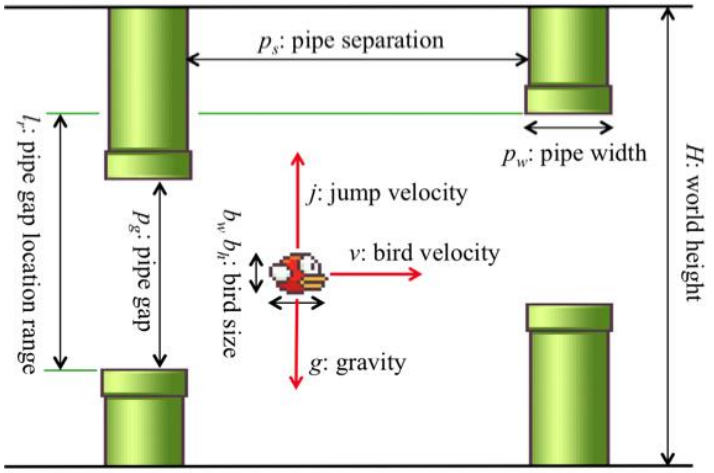

Fig. Flappy Bird DNA Traits

D. Initialization

We utilized an open source model of Flappy Bird from MIT and after restructuring the code to be object oriented, we recreated a similar genetic algorithm as a start point as Smart Rockets. We used population sizes from 20 to 1000 to determine the best and fastest method of a "solution". We also added a mutation rate which would mutate the worst birds using neural network parameters by the weights of the fitness' to change the birds for a future success. When a mutation is triggered, a child has a 10% change to replace its entire weight of the Neural Network and the remaining 90% of the time it would shift it by a random value within the standard gaussian from [-1, 1]. A crossover was finally added, once 2 parents were chosen, the baby bird would have a 50% change for any weight to be selected from parent A or parent B.

The final implementation would first initialize a population with random weights on each input of the neural network, would then play the game until the full population of birds would 'die', calculate the fitness for each bird, and perform a natural selection process in which 2 parents were chosen (based on fitness) and would perform a crossover to create a new child bird whose DNA to the neural network was 50% of both parent A and parent B. Finally, the child would be mutated based as described previously then loop back and rerun the game.

E. Playing around with the parameters

We created different implementations of the algorithms used before we decided on our final method. We learned that the fitness function as well as the neural network inputs had massive impacts on our solution and the speed at which the birds would learn. We started with a different neural network that factored in the distance between the bird to the vertical center of the pipes, the distance of the bird to the horizontal center of the bottom pipe, and the current height of the bird. This initial algorithm also had a constant fitness added to the bird's individual fitness depending on the score of the bird. We found that the parameters inserted were not enough to properly train the model. Only 3 inputs to the neural network (4 if you count a bias of 1), was not enough information to create a reliable neural network to succeed. The constant fitness used also proved to be inefficient until one of the birds somehow 'flapped' through the first set of pipes. Including the 'lifespan' or the frame the bird exists into the fitness helped push the bird past the first set of pipes.

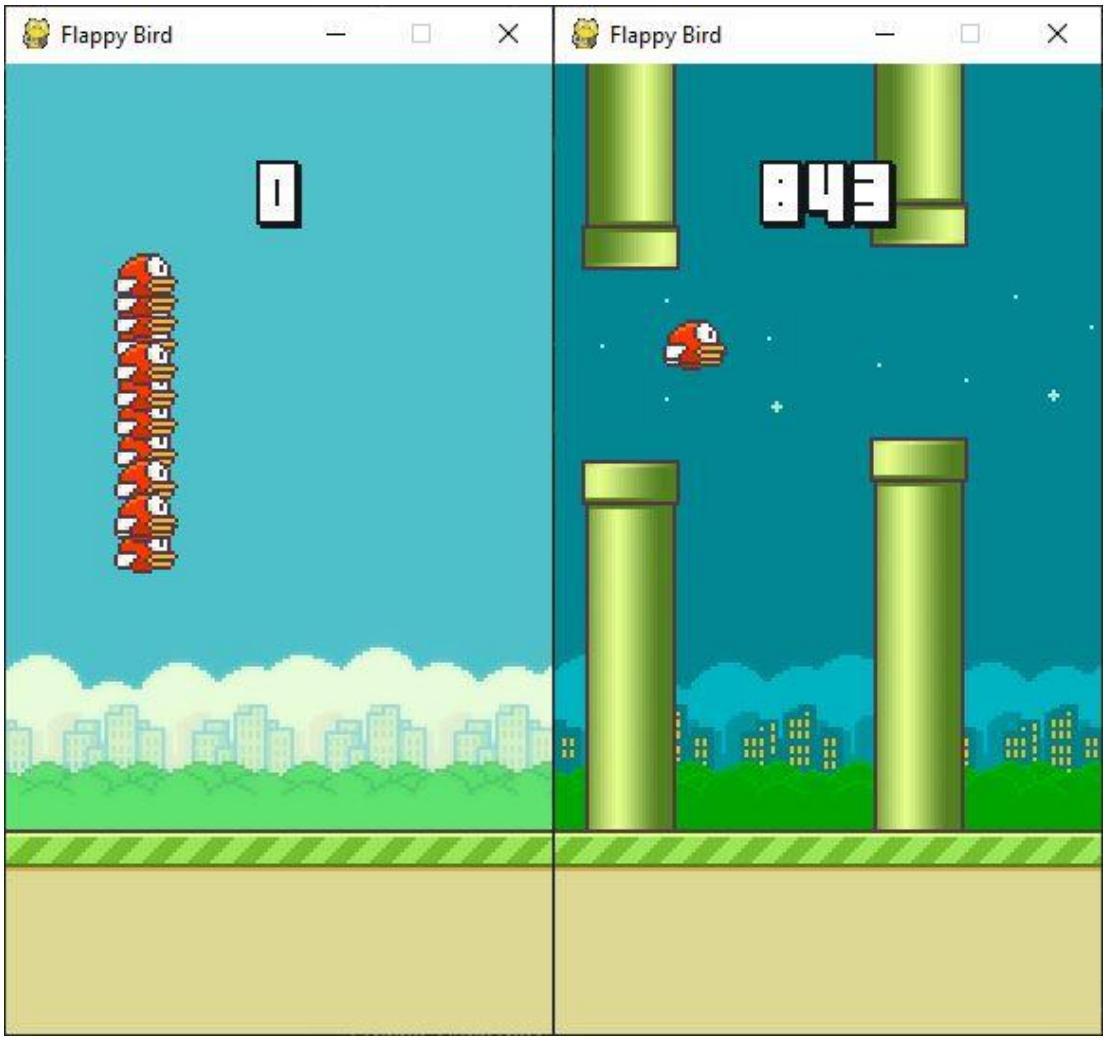

Fig. Flappy Bird Game After Running for 50 Generations

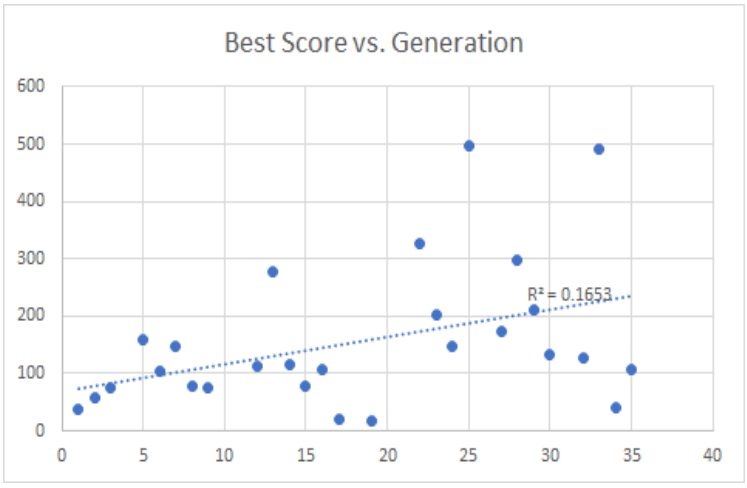

Fig. Best Score vs Generation Plot

F. Limitations

While running a peak algorithm with 1000 birds and a mutation rate of 15%, we found our limitations was more on hardware than software, the program would crash constantly. To get around this problem we only would display 20 birds onto the screen at once when testing, however, until a majority of the population died, we would experience major lagging and almost failure of execution by the program. Outside of the hardware limitation, we also found that using the genetic algorithm may not have been the best method at a solution. No matter how many generations we used, there seemed to be an outlier generation in which the lifespan would hit a score of 42 (generation 34), and then the next generation would score at 931 (generation 35). This may be resolved by adding more parameters to indicate all the pixel lengths of the pipes.

G. Conclusion

Overall, our algorithm proved to be successful in the long run. We were able to implement a genetic algorithm based on the implementation designed for the Smart Rockets program to a game such as Flappy Bird. We were able to utilize our previous knowledge of neural networks and apply it to a new situation of genetic algorithms and learn from real experience. Our genetic algorithm was successful in finding how to stay alive with randomly generating pipes at randomly generated heights. By adjusting the DNA (inputs to the neural network) and the fitness function, as well as our mutation and crossover rates, we found a model to solve the problem initially set out by our team, to find an acceptable and effective AI.

III. REFERENCES

Flappy Bird Inspiration:
Code Bullet, "A.I. Learns to play Flappy Bird," *YouTube* Available: https://www.youtube.com/watch?v=WSW-5m8lRMs.

Flappy Bird base source code for a human-controlled player:
Sourabhv, "sourabhv/FlapPyBird," *GitHub* Available: https://github.com/sourabhv/FlapPyBird.

IV. APPENDIX

Our Code Repository:
https://github.com/InformationScience/FinalProject-teamSnek