COSC343: Assignment 2 report

Yashna Shetty (2901410)
September 27, 2022

1 Introduction

This python program implements a Genetic Algorithm in order to train a population of snakes. For this assignment, the task involved implementing my_agent.py. This program begins by assigning 3 randomly generated chromosomes, with values between -10 and 10, to each Snake object and letting them run a certain number of times. After this, an evaluation of fitness function is run to evaluate the results of the old population.

2 Implementation

My approach involved the use of Roulette Wheel Selection in order to decide, in an organised-chaotic fashion, on the parents to be used for crossover.

Roulette Wheel Selection employs the use of a normalised fitness array. To do this, the fitness values, as returned by the fitness function, are summed up - via np.sum() function - to reflect the fitness of the population.

The fitness function used calculated the result by taking the maximum size that each snake grew up to and adding the quotient of the total number of turns it was alive and the total population.

$$f = maxSize + \frac{turnsAlive}{N}$$

The original fitness values are then sorted into ascending order, as well as the array of the old_population. To normalise the sorted array, a for loop is used to divide the current value by the population fitness. At the end of this process, we are left with a sorted, normalised fitness array. Where nF is the normalised fitness.

$$nF = \frac{f}{totalf}$$

Pseudo code to utilise np.random.choice in order to implement Roulette Wheel Selection can be found at https://bit.ly/3dP0FnN.

Upon picking the two parent chromosomes, I then implemented Single-Point Crossover. Parent chromosomes 1, 2, and 3, are passed into the parentCrossover function 3 times. This function takes 2 chromosomes, 1 from each parent, and implements single-point crossover. A random integer between 1 and the length of the chromosome is generated to be the crossover point. Everything up to this crossover point in parent 1's chromosome will be concatenated with everything after this crossover point in parent 2's chromosome, to make the child chromosome. There is a chance of mutation at 5%.

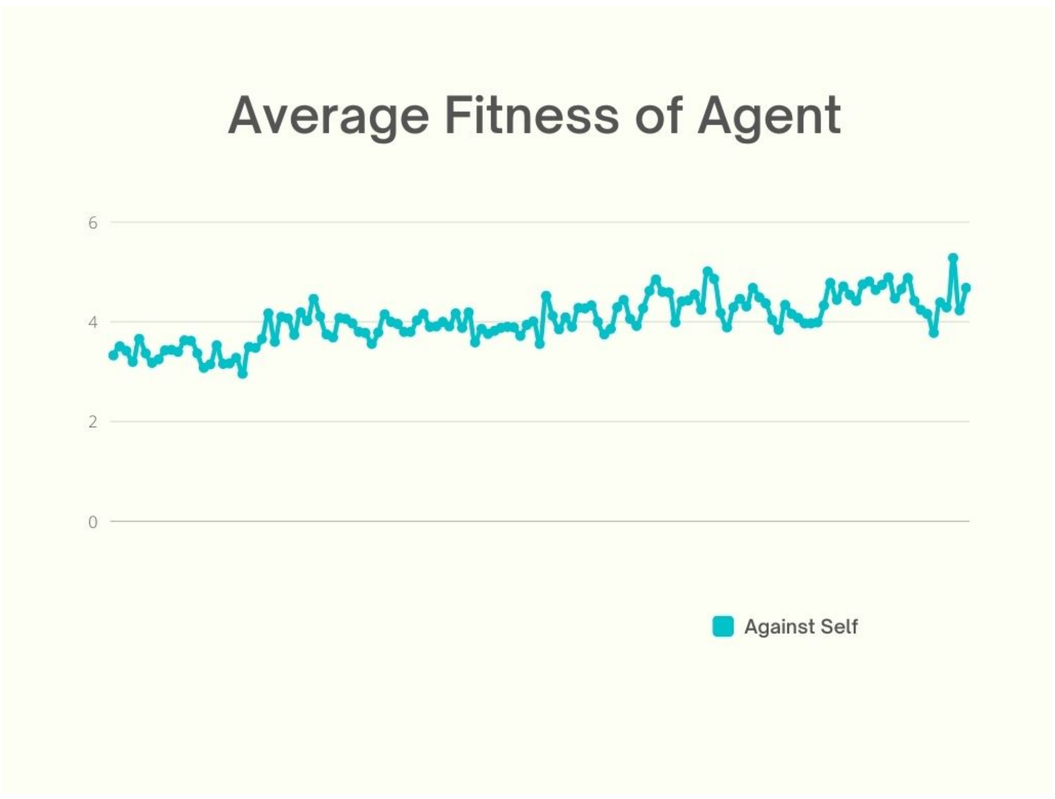

Figure 1: Average fitness of Agent against self

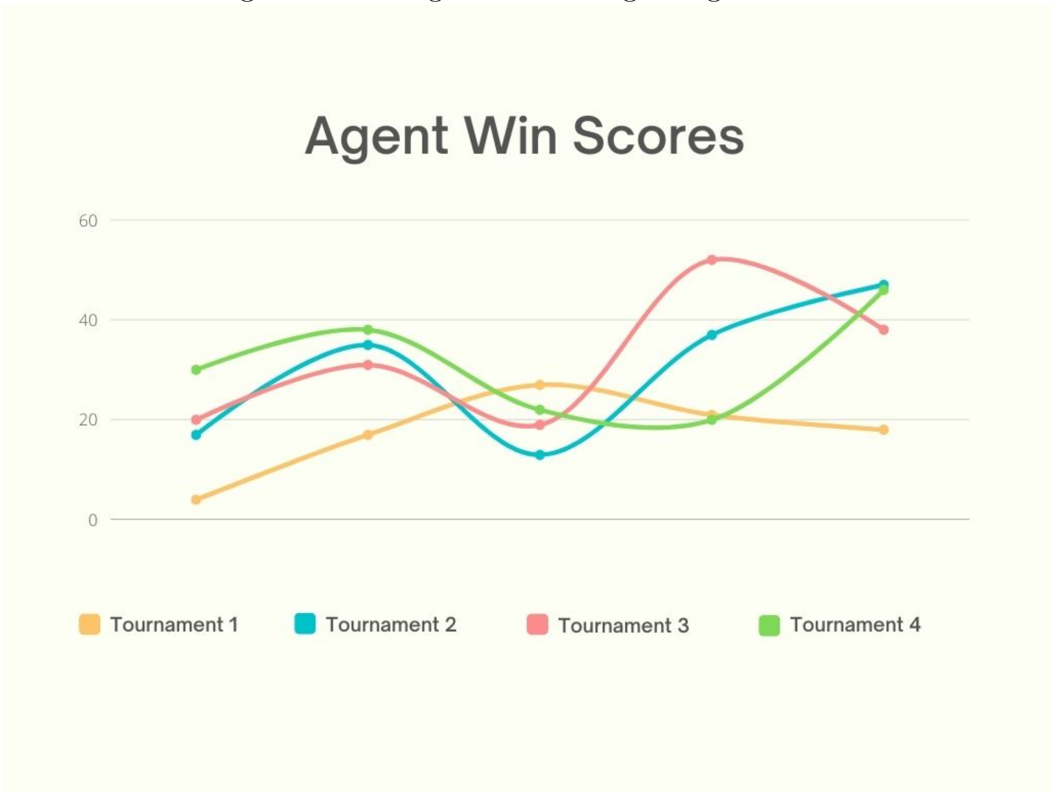

Figure 2: Agent Scores over Tournaments

2.1 Results

Upon completing this implementation of the agent, I found that there was in fact an increase in the average fitness scores of the Agent. Training results, as seen in Figure 1, show this increase in scores. The first ten training sessions averaged a fitness score of 33.8, while the final ten training sessions averaged a fitness score of 50.95.

It is also true that my_agent does not lose a game against the random agent. Figure 2 shows the average of score for Agent increasing during each tournament - one tournament being five games.

3 Explanation

The choice to model the chromosomes was made with the evaluations done by the agent function in mind. In order to directly map percepts to actions, I decided to initialise 3 chromosomes for each snake. I concluded that having different chromosomes map to different actions was a more powerful means of decision making than implementing a threshold value which would be more arbitrarily decided.

Before I began, my decision to maintain the percepts as 3x3 matrices was to keep the state space searches simple. The individuals are able to make more immediate decisions which means the chromosomes can be kept simple, and the program overall is faster as there are less computations to pursue.

The chromosome is a 3x4 matrix of weights, with the last column representing the bias which is still part of the chromosome and still passed on to the child via one of the parents. The choice to implement a matrix was made due the percepts being passed in as a matrix. The functional uses of matrices through Numpy is handled effectively and accurately, therefore I saw no need to change the layout.

The fitness function was not changed as I decided its implementation was sufficient for my cause. The fitness function took into consideration how long the snake was alive to play the game, as well as it's total size by death. This was able to effectively evaluate the individuals weight as my goal was to shift these values onto an interval from 0 to 1.

The agent function was the direct map from percept to action. The three chromosomes were calculated via a perceptron-like model. By summing up all the products of the chromosome values and percept values, we are left with three values, which we are able to use to make a decision about what action the snake must perform. If chromosome one was the largest of the three chromosome sums, the snake would turn left - that is, -1. Chromosome two being the largest meant the agent function mapped to a right turn - that is, 1. And finally, chromosome three being the largest mean the agent function mapped to no change of direction, or forward - that is, 0.

Of course, the more successful chromosomes were held by the fittest snakes. For this reason, I decided that the Roulette Wheel Selection process to picking a parent would be the most reasonable. The fittest individual would have the largest interval - since they would be living the longest - so by using numpy.random.choice, I was able to implement a weighted randomness.

The decision to implement a single-point crossover was made for two reasons. First, the chromosome, when entering the crossover phase, would remain a matrix. The 3x4 layout meant a "single-point" crossover was still able to produce a significant amount of variation in the population due to the information being both horizontally and vertically aligned. Second, with the size of the matrix being a mere 3x4, it would not be reasonable to use a multi-point or uniform crossover approach as too much segmentation of the information could mean that we are not retaining as many strong individuals/chromosomes.

During this crossover process, I also implemented a mutation chance. This mutation is able to keep the population from becoming too genetically similar as the parents crossover, which means behaviour is still capable of adapting should the rules of the game change.

4 Evaluation

The average fitness function of the agent shows that there is definitely an increase in performance with training, exactly the goal of the program. The Agent scores brings an important issue to light, however. It is possible that elitism would have proven to produce a better performing population, and a little less random. It may have been possible to save some of the best performing individuals and pass them on to the next generation. There are certain cases and games where the random behaviour of the Genetic Algorithm components may be performing too heavily. However, this may be bred out by running more tournaments and training, since we are able to see a - random, but - gradual incline in performance.

5 Conclusion

I found this assignment quite interesting! I definitely believe that my program could have performed better if I drew back the level of randomness implemented throughout the program in order for it to work. But due to the stochastic nature of the Genetic Algorithm, I was a little weary to implement more concrete actions in fear that the actions performed by the individuals would lack in diversity, which would be vital for the chromosomes to express should there be a change in rules.

Overall, it was definitely interesting to graph out the data and find genuine changes in the scores and behaviours of the snakes, despite my hesitance towards randomness.