NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

Genetic Programming on the classic Snake game

Computational Intelligence for Optimization

Fábio Lopes (20200597)

Felipe Costa (20201041)

Jorge Pereira (20201085)

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

1. METHODOLOGY

1.1. SNAKE GAME

1.1.1. Snake Movement

A critical aspect of this project was the logic behind the movement of the snake. There were two approaches possible: the board as the "centre of the universe" or, the snake head as the "centre of the universe". This work took the approach of using the snake head to coordinate movements because there are a few key benefits made possible in the genetic programming.

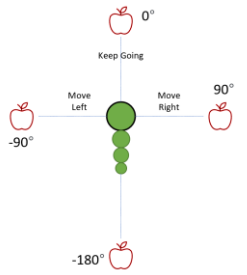
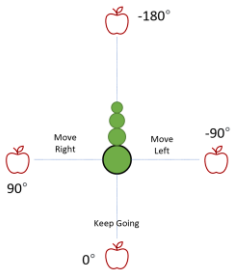
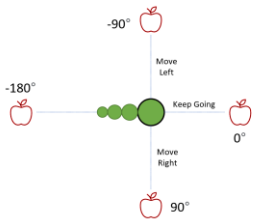


Figure 1 - Snake Vertical Movements





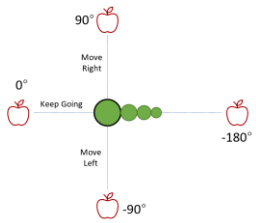


Figure 2 - Snake Horizontal Movements

First, since the angle between the food and the snake needs to consider the direction of movement, the learning process will become simpler by not having to interpret both the angle and the direction.

Second, as can be seen in Figure 1 and Figure 2, the angle to the food becomes consistent with the location of the snake, irrespective of the direction of movement. When fruit is to the front of the snake it will have a 0° angle, to the back will be -180°, to the left will be -90° and to the right 90°. This allows for the prediction of the movement to be simplified and just consider "continue movement", "turn 90° to the left" and "turn 90° to the right".

By contrast, if the angle to the fruit was related to the board, the learning process would need to consider both the angle and the direction of movement and, the prediction of movements would need to be made on the 4 directions of movement, which would introduce a great degree of variability in the inputs and reduce the ability of the snake to learn.

1.1.2. Predicting Movement

To predict movement, a Fully Connected Neural Network was designed to handle the inputs received from the current state of the game and to predict the movement by defining 3 output nodes. Table 1 and

Table *2* contain the characteristics of both inputs and outputs, respectively.

Table 1 - Inputs of the Neural Network

Input	***Possible Values***	***Description***
Normalized angle to the food	[-1,1]	Angle to the food computed in degrees and divided by 360.
Normalized Distance to the food	[0,1]	Euclidean Distance to the food divided by the corner-to-corner distance of the board.
Possible CONTINUE movement	0 or 1	Indicates if a CONTINUE movement will result in a collision with the board or the snake.

Possible LEFT movement	0 or 1	Indicates if a LEFT movement will result in a collision with the board or the snake.
Possible Right movement	0 or 1	Indicates if a RIGHT movement will result in a collision with the board or the snake.
Last movement is CONTINUE	0 or 1	Indicates if the last movement was CONTINUE.
Last movement is LEFT	0 or 1	Indicates if the last movement was LEFT.
Last movement is RIGHT	0 or 1	Indicates if the last movement was RIGHT.

Table 2 - Outputs of the Neural Network

Output	***Possible Values***	***Description***
CONTINUE	[0,1]	Indicates the probability of choosing the CONTINUE movement.
LEFT	[0,1]	Indicates the probability of choosing the LEFT movement.
RIGHT	[0,1]	Indicates the probability of choosing the RIGHT movement.

The performance of the network was of critical importance since for each movement of the snake, the inputs need to be feed forwarded to the network to obtain a prediction and this will be required for thousands of iterations across all generations and children per generation. After testing, the architecture which resulted in a balance between performance and quality of decisions was to use 2 hidden layers, the first with 9 nodes and the second with 15, which can be seen in **Error! Reference source not found.**. The activation function used in the hidden layers is the Hyperbolic Tangent, while in the output layer, the softmax was chosen to allow the final classification to represent the probability of each class being chosen. After the prediction was obtained with the feed forward algorithm, a translation is performed between the CONTINUE, LEFT and RIGHT movements to the board movements: UP, DOWN, LEFT, RIGHT.

1.2. GENETIC PROGRAMMING

1.2.1. Fitness Function

The chosen Fitness function for this work is:

$$Fitness = Score * \ \ 100000 + Rewards * 1000 - Penalties * 1000000$$

Table 3 - Fitness Function Parameters

Metric	***Possible Values***	***Description***
Score	[0,∞[	Indicates the maximum score obtained.
Rewards	[0,∞[	Indicates the number of movements performed by the Snake which placed it closer to the fruit.
Penalties	0 or 1	Indicates if the snake took several steps without eating.

The goal of this fitness function is to reward the score obtained and to encourage the snake to always get closer to the fruit by favoring movements which provide rewards, while penalizing snakes which spend a lot of movements without eating.

This fitness function was the result of several iterations. At first, only the score and number of steps performed were considered but as expected, it led to poor convergence and overall worst results. One of the main concerns was that snakes which randomly walked through the board were getting chosen because of the number of steps taken even without eating any fruit.

In the next iteration, the penalty system was introduced. This allowed for the elimination of snakes which would spend a determined number of steps without eating but, presented a similar problem where the learning process would select snakes where the number of steps would be high while having lower scores.

At last, the rewards system was introduced. The goal of it is to converge to 0 rewards the snakes which randomly move back and forth to the location of the fruit. This allows the learning to privilege snakes which always choose a movement that places it closer to the fruit.

1.2.2. Selection Methods

The selection methods chosen and implemented in this work were: Tournament Selection (with a 10% of children Tournament Size), Fitness Proportionate Selection and Rank Selection. All the methods were implemented to work with both Maximizing and Minimizing the Fitness Function.

During the testing phase, the FPS and Rank Selection were performing worse (in fitness value) and as such, an elitism approach was implemented for these 2 selection operators where, in each generation, only the top 50% of children (in fitness value) were considered. This increased the overall performance of these operators.

1.2.3. Crossover Operators

The Crossover operators chosen for this work were: Uniform, Whole Arithmetic, One Point and Two Point. All the genes in the Childs are of decimal values so, the crossover operators were chosen to be compatible with this paradigm.

For the Whole Arithmetic the parameter α was set to 0.5, effectively computing the average of each gene in both parents.

1.2.4. Mutation Operators

For this work, the Mutation Operators chosen and implemented were:

- Random Value, where for each selected gene, the value is replaced by a sample from a uniform distribution with bounds [-1,1].
- Normal, where for each selected gene, a value is sampled from a normal distribution with mean 0 and standard deviation 0.1 and added to the gene. The goal of it is to prevent huge shifts in the gene value.
- Inversion, where a subset of genes is selected and reversed in order. For each selected gene, it plus the 4 next genes are reversed in order.

1.2.5. Genetic Algorithm

The genetic algorithm in this work starts by creating the first generation by randomly sampling the gene values from a uniform distribution with range [-1,1]. For each child in the generation, the Snake game is played and the fitness for the game result is computed. During the number of generations predefined in the code, the following loop is performed:

1. Perform the selected Selection Algorithm.
    a. Sample 2 parents from the current generation.
    b. With 80% probability, perform the Crossover Algorithm. Else, choose the best of the 2 parents.

c. With 30% probability, perform the Mutation Algorithm. Each gene has a 3% probability of being mutated.

2. Play the Snake game for each new child.

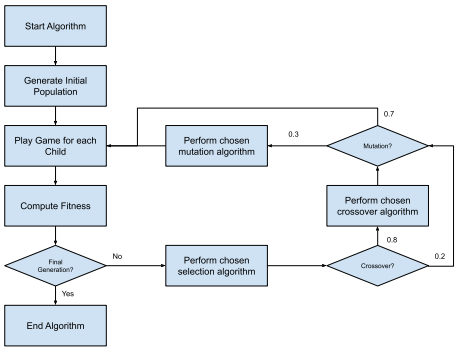


Figure 3 - Diagram of the Genetic Algorithm in this work

2. RESULTS AND DISCUSSION

2.1. EXPERIMENTS

To test all the genetic operators, the algorithm was executed with all possible combinations of operators possible with the following common parameters: 100 children per generation, 50 total generations to execute, maximize fitness, 80% crossover probability and 30% mutation probability. The metrics that are being evaluated are: The fit of the last generation, the maximum fit achieved in learning, the maximum score in the last generation and the maximum score achieved in learning. Due to performance reasons, a single run was performed for each combination of operators where the only control step performed was setting the seed of the random operations the same between runs.

Table 4 - Results of the combination of Genetic Operators sorted by the fit at last generation

Crossover Strategy	Mutation Strategy	Selection Strategy	Last Fit	Max Fit	Last Score	Max Score
Uniform	Random Value	Tournament	9014k	10247k	67	78
Uniform	Normal Distribution	Tournament	7617k	10767k	56	83
Uniform	Inversion	Tournament	7123k	9622k	54	72
Single Point	Random Value	Tournament	6977k	10024k	51	75
Single Point	Normal Distribution	Tournament	6905k	10130k	52	77
Single Point	Inversion	Tournament	6622k	10320k	51	76
Two Point	Normal Distribution	Tournament	5941k	8609k	44	65
Two Point	Inversion	Tournament	5734k	8372k	44	63
Whole Arithmetic	Random Value	Tournament	4020k	5295k	31	41
Whole Arithmetic	Normal Distribution	Tournament	3892k	5626k	30	44
Whole Arithmetic	Inversion	Tournament	3877000	6682k	30	50
Whole Arithmetic	Random Value	FPS	3709k	4355k	28	33
Two Point	Random Value	Tournament	3028k	8438k	23	63
Two Point	Random Value	FPS	2779k	6653k	21	52
Whole Arithmetic	Normal Distribution	FPS	1151k	1311k	10	10
Two Point	Random Value	Rank	1055k	1799k	8	14
Single Point	Inversion	FPS	7590k	1425k	6	11
Whole Arithmetic	Inversion	FPS	7590k	1390k	6	11

Crossover Strategy	Mutation Strategy	Selection Strategy	Last Fit	Max Fit	Last Score	Max Score
Whole Arithmetic	Normal Distribution	Rank	699k	1302k	5	11
Uniform	Normal Distribution	Rank	541k	4758k	4	37
Uniform	Normal Distribution	FPS	530k	2348k	4	18
Uniform	Random Value	Rank	509k	6978k	4	54
Two Point	Normal Distribution	FPS	463k	1558k	3	12
Two Point	Inversion	Rank	456k	962k	4	8
Uniform	Inversion	FPS	396k	1311k	3	10
Two Point	Normal Distribution	Rank	377k	3093k	3	23
Single Point	Normal Distribution	FPS	373k	3740k	3	28
Single Point	Random Value	Rank	268k	1621k	2	13
Whole Arithmetic	Random Value	Rank	252k	1026k	2	8
Uniform	Random Value	FPS	227k	989k	2	8
Single Point	Inversion	Rank	224k	968k	2	8
Uniform	Inversion	Rank	218k	1350k	2	10
Single Point	Normal Distribution	Rank	129k	800k	3	6
Whole Arithmetic	Inversion	Rank	123k	2219k	2	17
Two Point	Inversion	FPS	119k	612k	1	7
Single Point	Random Value	FPS	26k	613k	1	7

As per *Table 4*, the Tournament Selection consistently achieved the highest metrics both in fitness value and in scores. Within the Tournament Selection, there is a clear pattern favoring the use of a Uniform Crossover strategy, while there's no clear advantage in choosing a particular Mutation Strategy.

By analyzing (in Figure 4 and Figure 5) the minimum, average and maximum fitness and score for the top scorer, one can see that the learning has converged and we see no significant improvement across new generations except for variations due to the randomness of the process (regarding the fruit location).

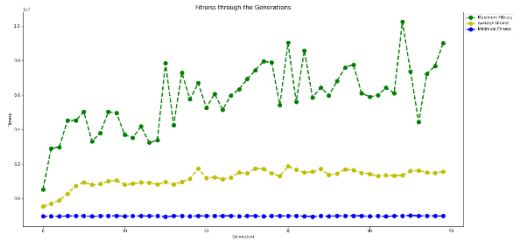


Figure 4 - Fitness across all generations for the Tournament, Uniform and Random Value Operators

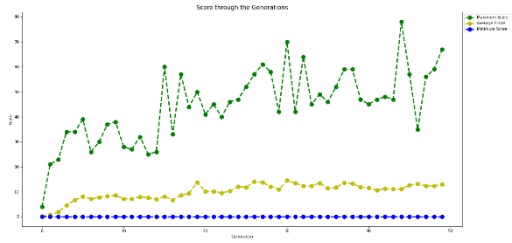


Figure 5 – Score across all generations for the Tournament, Uniform and Random Value Operators

By contrast, if we analyze one of the FPS Section results (in *Figure 6* and *Figure 7*), one can see that the convergence has not been reached yet. Which indicates that the Tournament Selection is a better approach for this problem, since it will converge much faster than the other methods, even with a 50% elitism not applied to the Tournament like it is in the other methods.

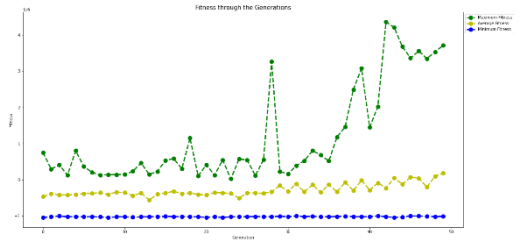


Figure 6 - Fitness across all generations for the FPS, Whole Arithmetic and Random Value Operators

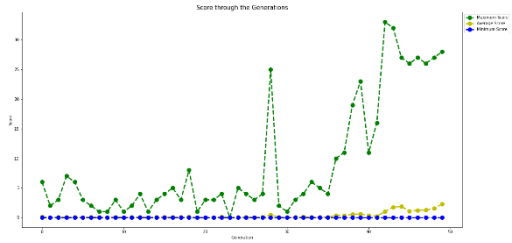


Figure 7 – Score across all generations for the FPS, Whole Arithmetic and Random Value Operators

Another relevant topic to discuss is the lack of knowledge obtained in the learning process for the bottom combinations, which seem to indicate that those operators, in conjunction, are not suited for the problem at hand.

3. CONCLUSIONS

Overall, there is a feeling of achievement throughout the group with respect to this work resulting from the use of genetic programming to solve a problem usually reserved for humans without resorting to any heuristic logic to guide the prediction of movements of the snake. From the start, a lot of challenges appeared related to both performance of the code plus performance of the learning process itself. The solution to both was a full revamp of the code, re-writing most of the code to use the Numpy library instead of Keras and a deeper look at all the genetic operators. This allowed the code to run about 90% faster and with great improvements to the learning process.

For future work, there are a couple of open points which the team would like to see addressed. First, by analyzing the behavior of the snake in the videos produced, one can see that the major culprit of a Game Over in the final generation is usually the snake eating itself. If the snake would receive a penalty for eating itself after a certain number of generations (to allow it to first learn how to eat the fruit and then how to avoid itself), would perhaps bring good results at the cost of slower convergence. Second, there is a feeling that the FPS and Rank Selection were not fulfilling their true potential in this work and, by applying a selection stress parameter which would control the elitism plus other parameters to force the selection of better snakes, better results could be obtained in these selection methods.