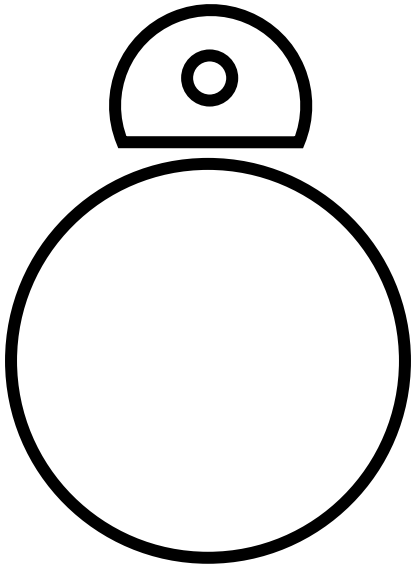
UNIVERSIDADE D COIMBRA

Introdução à Inteligência Artificial

FCTUC – LDM 2019 / 2020



Trabalho Prático Nº3**:**

D31 – The Rise of the Ballz

Feito por:
Pedro Tavares - 2017269091
Paulo Fernandes - 2009129179
Kelvin Clark -2017230108

Index

D31 – The Rise of the Ballz

Table of Content:

Introduction:

Background:

This practical project focuses on evolving the D31 unit into a more intelligent unit over hundreds of simulations. To do this, we are required to use the genetic algorithm to modify and mutate newer units while inheriting older unit genes. The robot unit will have to learn how to play football, both as an attacker and a defender. There are two units, an attacker and a defender. The attacker will attempt to learn how control the ball and score a goal while the defender will learn how to defend his goals and try to not allow any goals enter by hitting the ball away.

Methods:

The implementation of allowing the evolution of the unit is done using the genetic algorithm as well as some controllers specified for the role (defender, attacker). The genetic algorithm needs a few algorithms to be implemented; These algorithms are namely the crossover algorithm (In order to crossover genes to newer generations) and gaussian mutation algorithm (In order to mutate new units at the beginning of each generation). In addition, a tournament selection algorithm (In order to choose the best unit of the generation to pass over the genes) along with two different fitness functions (controllers for the unit to be able to know what to learn) will also be needed for implementation.

Objective:

Our objective in this practical work is to create controllers for the unit so that the ability to know what to learn is achieved, in this case – to play football. These controllers will focus on defending, controlling the ball and playing against one another.

Implementation:

Crossover algorithm: The crossover algorithm is implemented by receiving an individual and a probability, it then chooses a random value and compares it the probability value. If the value is less, then the algorithm chooses a random point along the genotype range to know when to swap genes between the current individual and the received individual. Both individuals will be modified, one with the beginning genes before the cut point, and one with the genes after the cut point.

Gaussian Mutation Algorithm: The mutation algorithm is implemented by receiving a probability value and doing something like the crossover algorithm, if the value of the probability is higher than a random value then the mutation is done. The mutation is done by using an equation to manipulate the current genotype values into a newer set of values with the use of a mean value and a standard deviation value.

Tournament selection Algorithm: The tournament selection algorithm is implemented by receiving a list of individuals and a size value. This algorithm will choose a ‘size’ number of individuals from the list and return the individual with the

best fitness value. We have also implemented a condition in which the same individual is not chosen twice so that there are no repetitions during the selection process.

Fitness function (Attacker): The fitness function for the attacker is implemented by training the unit to control the ball to the adversary goal. This is achieved by mapping two values (the average distance from the unit to the ball and the minimum distance from the ball to the adversary goal) with the fitness value and receiving an inversed value. Once this is done, it then gets multiplied by a value to create a priority list. Another influence on this value is the amount of times the unit hits the ball and the amount of goals that the unit scores, these two numbers are also multiplied by a priority value.

Fitness attack variables and respective weights

To develop the attacker's Fitness function, we considered the following variables with the following weights:

-Average distance to ball: 1000

-Minimum distance from ball to adversary goal: 5000

-Hit the ball: 200

-Goals on adversary goal: 10000

Fitness function (Defender: The fitness function for the defender is implemented by training the unit to defend his goals and not allow the ball to enter the goal. This is achieved by mapping three values (the average distance from the unit to the ball, the maximum distance from the ball to its goal, and the minimum from the unit to its goal) with the fitness value and receiving a positive or negative value depending on the value mapped. Once this is done, it then gets multiplied by a value to create a priority list. Another influence on this value is the amount of times the unit hits the ball and the amount of goals conceded, these two numbers are also multiplied by a priority value.

Fitness defence variables and respective weights

To develop the defender's Fitness function, we considered the following variables with the following weights:

-Average distance to ball: 5000

-Maximum distance from ball to my goal: 5000

-Minimum distance to my goal: 5000

-Hit the ball: 1000

-Goals on adversary goal: -10000

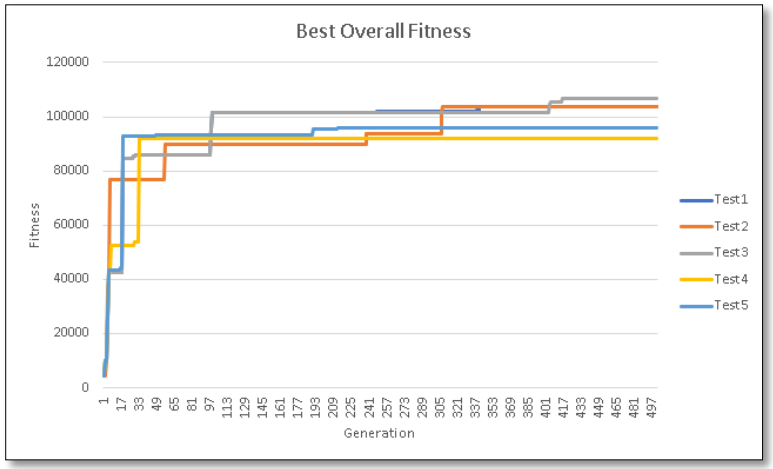
Results and Analysis:

The following tests were made by changing the mutation probability and the crossover probability, in some cases the tournament size and the random seed has also been manipulated. Each combination of parameters has been repeatedly tested 5 times on each map to be able to view any changes in behaviour through the generations. The tests were all done through 500 generations with 20 simulations each. Below are graphical representations of the results (fitness).

Due to the stochastic nature of this experiment, it was necessary to repeat it with the same parameters and with different seeds, to ensure some degree of statistical relevance.

Map 1 – Controlling Ball To Adversary Goal

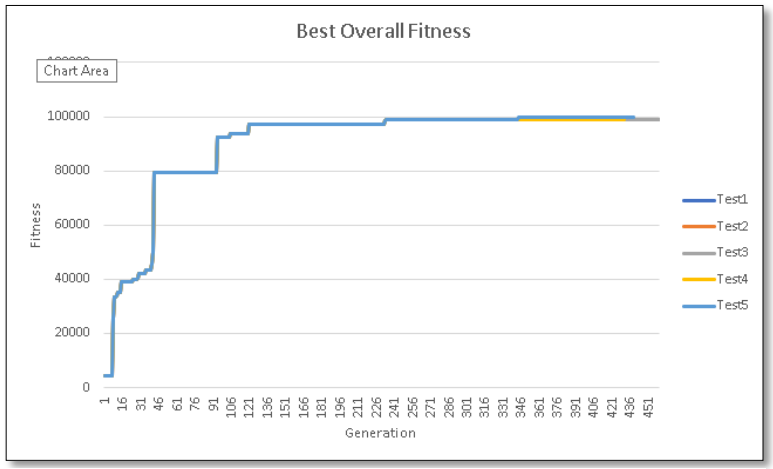
Mutation at 0.75 || Crossover at 0.5 || Tournament size at 2 || Random seed at 50



Even though parameters stay the same, there is a significant difference in each test. All tests show increasing fitness in different ways, therefore the unit is learning in all tests, but at different rates each time.

Map 1 – Controlling Ball To Adversary Goal

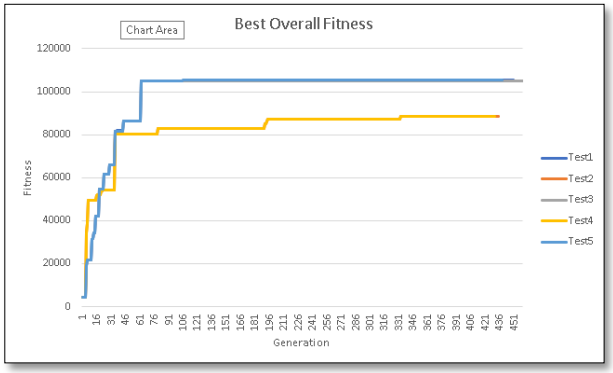
Mutation at 0.25 || Crossover at 0.3 || Tournament size at 2 || Random seed at 50



Each test has returned almost the exact same result. This might have to do with the mutation rate being drastically lower than the last parameter combination. The crossover rate has also decreased which might also factor into these results.

Map 1 – Controlling Ball To Adversary Goal

Mutation at 0.4 || Crossover at 0.8 || Tournament size at 2 || Random seed at 50

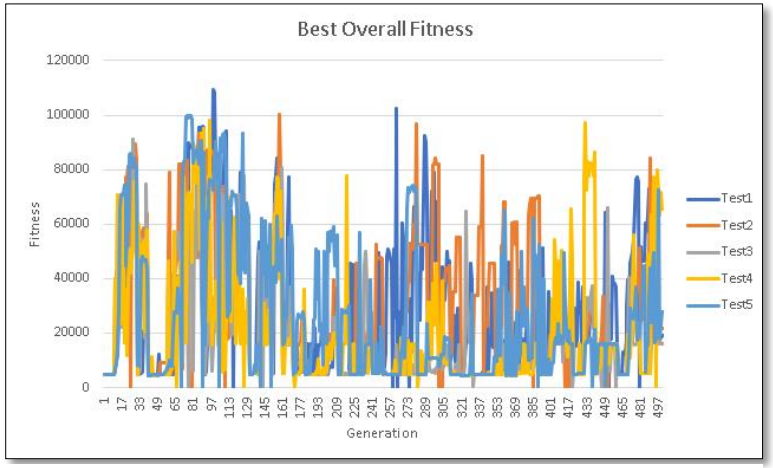


The results seem to be a combination of the previous two combinations. The mutation rate is at a midpoint, therefore providing reason to the idea of the combination. However, the crossover rate is higher, which might result in less gene transferral which would cause units to carry most genes of the previous generation.

Although setting the Mutation/Crossover probabilities at 0,4 and 0,8 achieves a good result in less generations, by setting them at 0.75 and 0,5 we ended up getting a better fitness peak value.

Map 2 – Controlling Ball To Adversary Goal (Random)

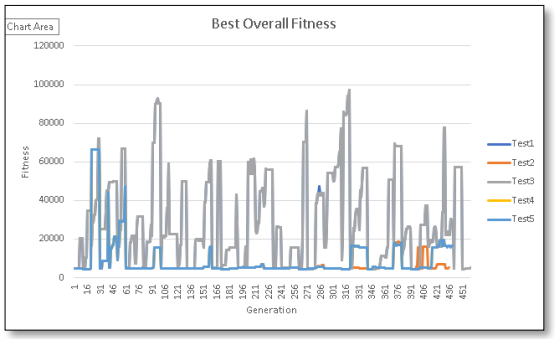
Mutation at 0.5 || Crossover at 0.75 || Tournament size at 3 || Random seed at 50



The best overall fitness changes quite drastically in each generation. The fact that the map condition includes a random ball position at the start of each generation might be the reason. However, the unit does seem to gradually begin to learn to move towards the ball and push it closer to the goal under these conditions.

Map 2 – Controlling Ball to Adversary Goal (Random)

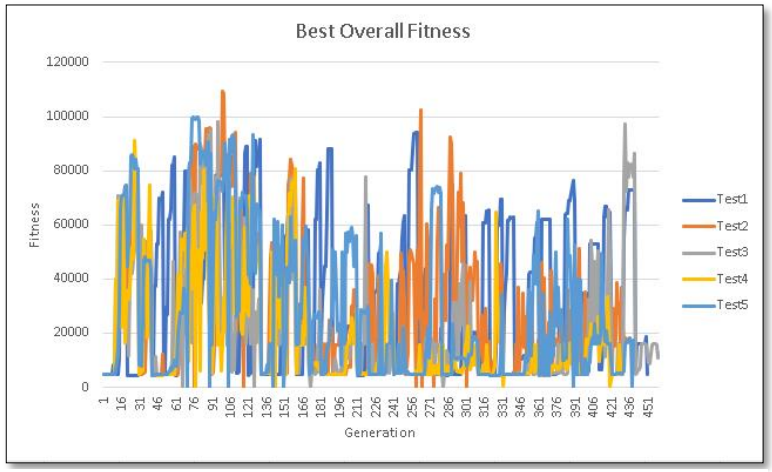
Mutation at 0.6 || Crossover at 0.4 || Tournament size at 2 || Random seed at 369



The random seed and the lower crossover rate has seemed to change the results quite a bit. The results still seem to be difficult to capture due to map situations, but the learning rate seems to have dropped overall, with the exception of 2 tests being almost exactly the same.

Map 2 – Controlling Ball to Adversary Goal (Random)

Mutation at 0.75 || Crossover at 0.5 || Tournament size at 2 || Random seed at 50

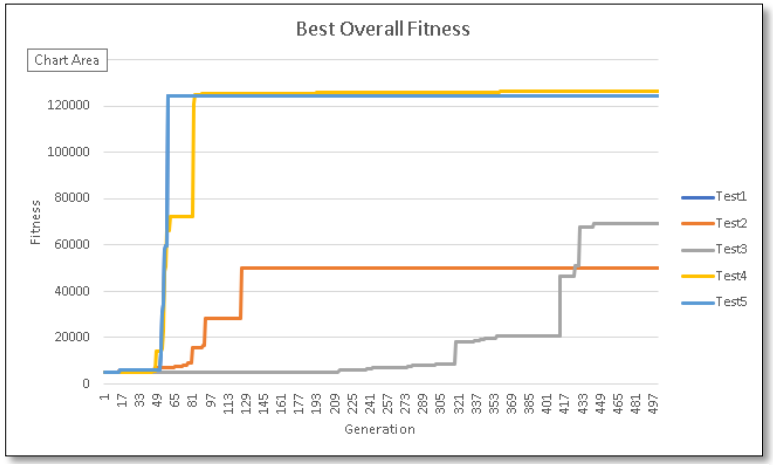


Results are similar to the last parameter combination tests, yet slightly different. Yet again, it seems to be difficult to capture the learning pattern due to different situations during each simulation.

The random seed input value seems to have impacted considerably the results, since the experiment with the 369 seed had a lot worse results than the experiments with the seed as 50 that peaked around the same fitness value, but with the experiment with the Mutation/Crossover pair of 0,5 and 0,75 achieving it a bit sooner.

Map 3 – Defense

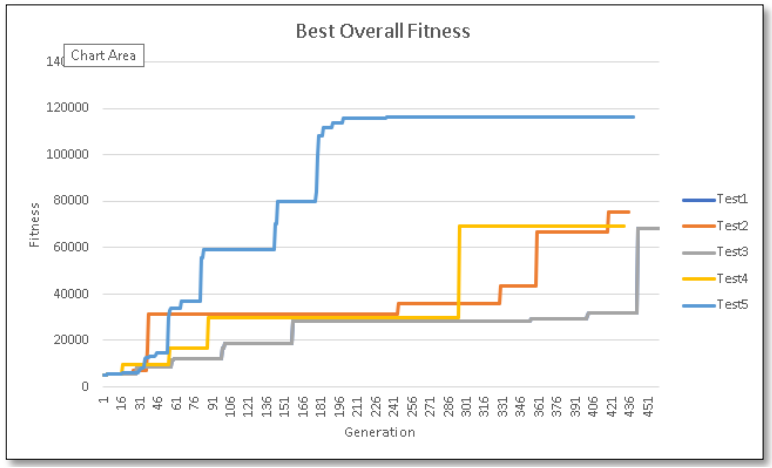
Mutation at 0.35 || Crossover at 0.6 || Tournament size at 3 || Random seed at 50



Similar behavior as the attacker, the unit learns how to defend the ball at different rates each time. Sometimes slower than others.

Map 3 – Defense

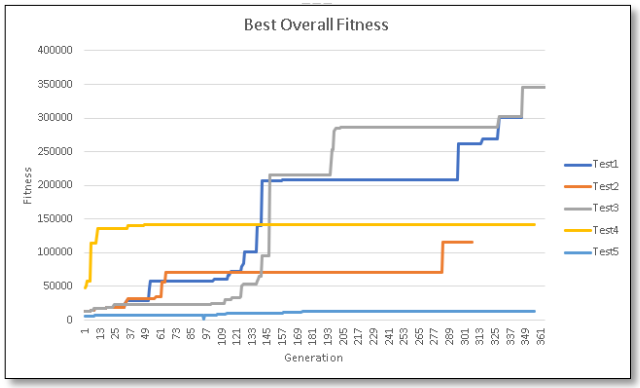
Mutation at 0.75 || Crossover at 0.5 || Tournament size at 2 || Random seed at 50



The significantly higher mutation rate seems to have differentiated each test quite drastically. However, the unit does seem to continue to learn at different rates.

Map 3 – Defense

Mutation at 0.67 || Crossover at 0.78 || Tournament size at 3 || Random seed at



Even though the mutation rate has decreased, the elevated crossover rate seems to differentiate each test. Nevertheless, the unit still learns gradually over time at different rates.

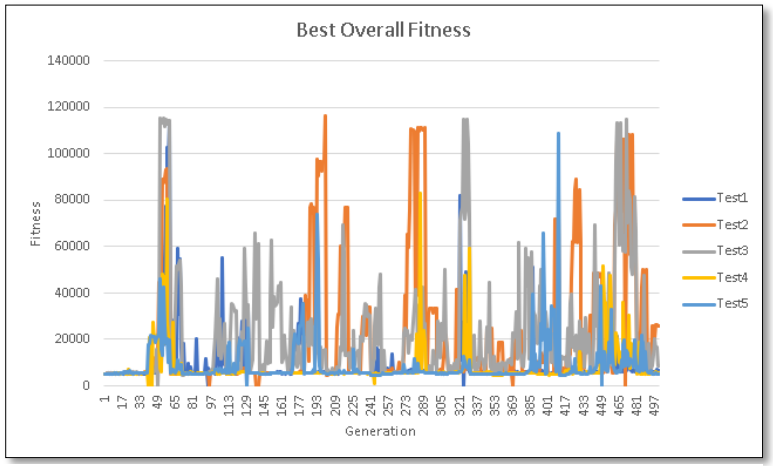
Overall, the defending agent is showing better results, with a higher fitness peak value and showing improvements sooner.

We conducted 2 experiments with the seed at 50 and one with the seed at 777. The experiment with the 777 seed showed a huge improvement in the fitness peak value.

While the Mutation/Crossover pair of 0,67/0,78 and seed 777 shows, by far, the best fitness value, the experiment with the Mutation/Crossover pair of 0,35/0,6 achieved a good result a lot faster.

Map 4 – Defense (Random)

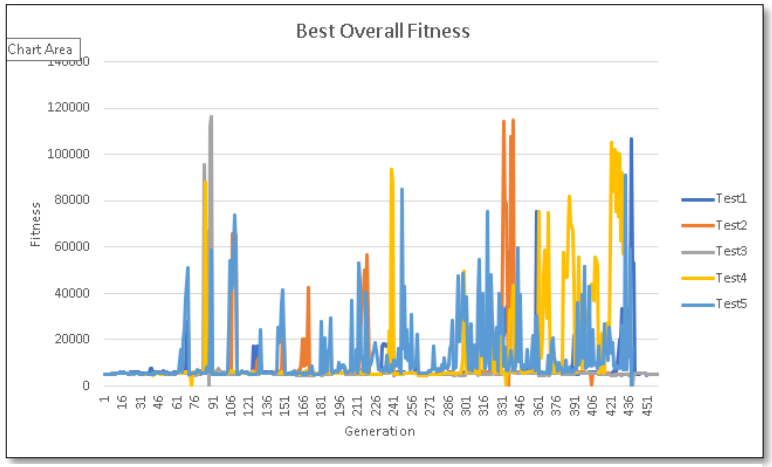
Mutation at 0.75 || Crossover at 0.5 || Tournament size at 2 || Random seed at 50



This map has the ball starting at random positions after each simulation, similar to the 2nd map; Therefore the unit learns with various different fitness levels shown refering to that specific map situation.

Map 4 – Defense (Random)

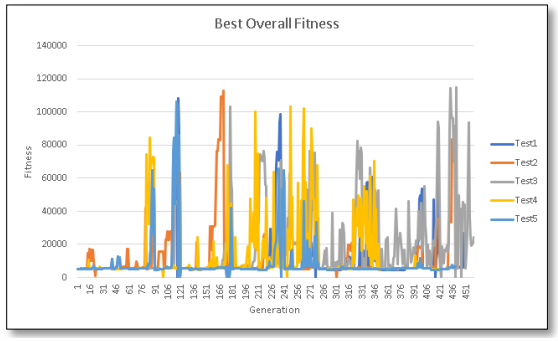
Mutation at 0.4 || Crossover at 0.6 || Tournament size at 3 || Random seed at 50



Due to the mutation level dropping compared to the last parameter combination tests, the results seem to have also dropped. The unit still learns and adapts the fitness to the specific map situation, but at a slower pace.

Map 4 – Defense (Random)

Mutation at 0.25 || Crossover at 0.25 || Tournament size at 3 || Random seed at 50



Similar results as the other tests which is to be expected due to map situations. The unit's behaviour seems like it is learning to defend the ball as it faces the ball and attempts to stop the ball entering its' goal.

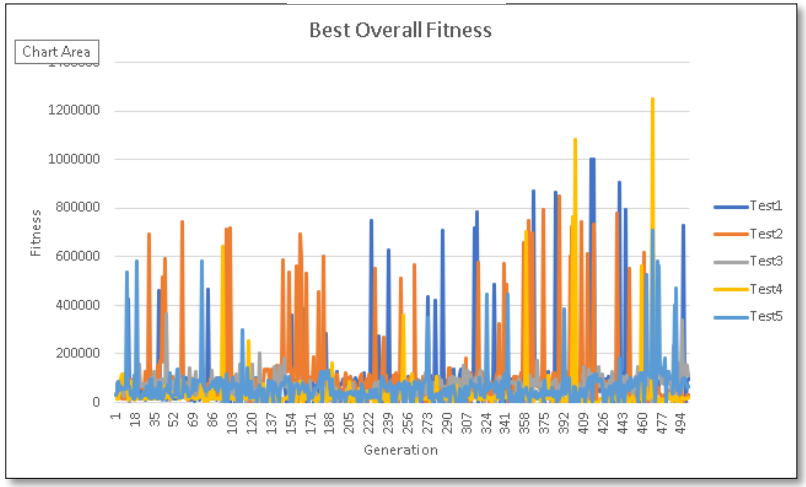
Once again, the random map seems to shake the tests quite a bit, but the robots do have generations with very high fitness, again higher than its correspondent "attacking" map.

The Mutation/Crossover pair of 0,75 and 0,5, shows the best test results. Despite having a fitness peak value similar to the other experiments, this peak was reached very early in the experiment.

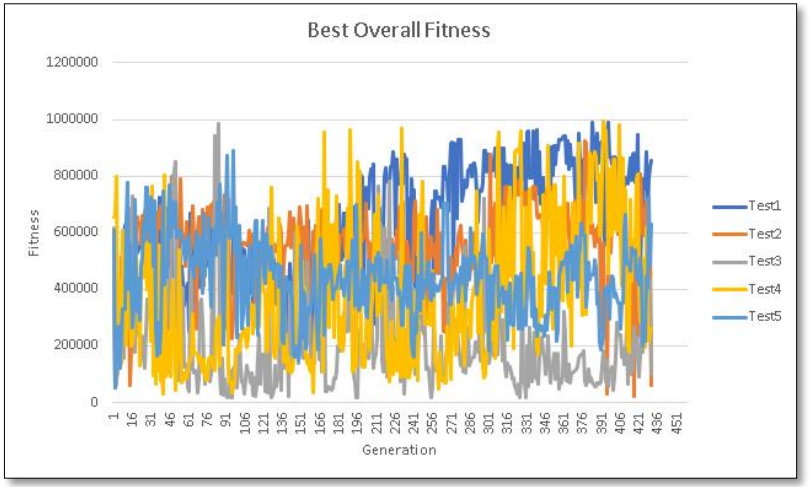
Map 5 – One vs One

Mutation at 0.5 || Crossover at 0.5 || Tournament size at 2 || Random seed at 50

Red (Attacker)

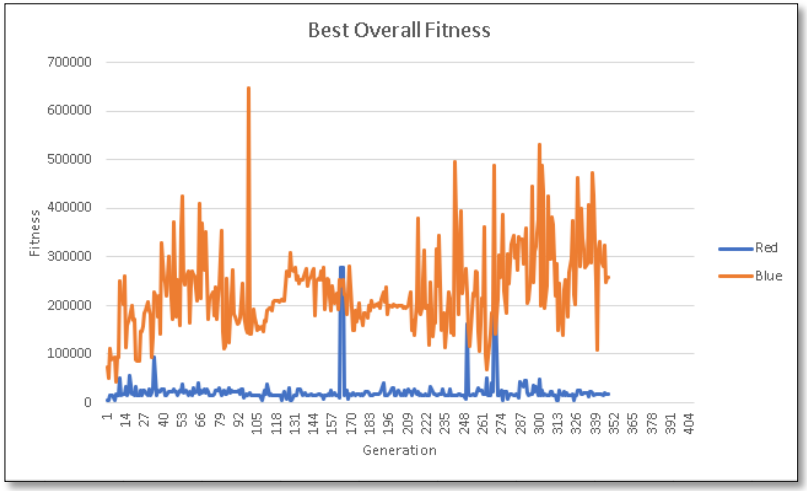


Blue (Defender)



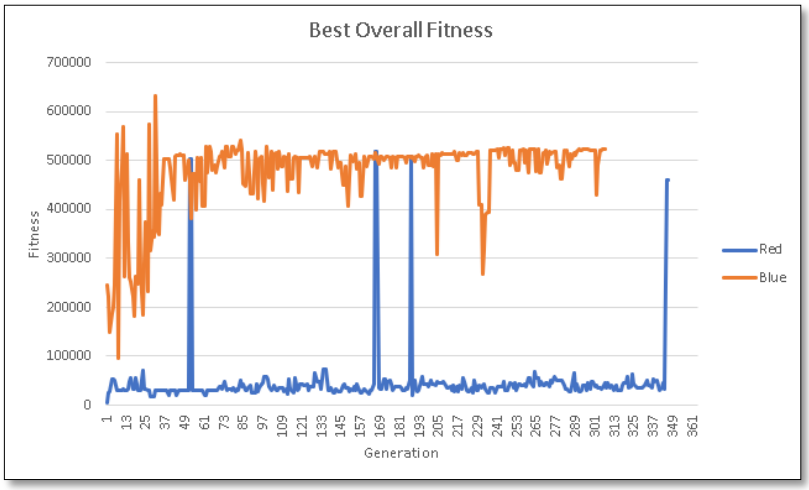
Map 5 – One vs One (1 test)

Mutation at 0.35 || Crossover at 0.65 || Tournament size at 3 || Random seed at 50



Map 5 – One vs One (1 test)

Mutation at 0.65 || Crossover at 0.35 || Tournament size at 3 || Random seed at 50



Overall, in the first experiment (Mutation/Crossover at 0,5/0,5), it can be observed that the attacker has a constantly low fitness while it looks to control the ball to perform an attack. When that attack is successful (or is very close to scoring), its fitness values shoot and, either after a goal reset or a successful defence, it goes back to its constant low.

Meanwhile, the defender has a bigger variation of the fitness values, since they should only really drop when a goal is scored, but the variation happens every time the ball comes close to the goal (fitness value down) or it gets away by a miss (fitness value moderately up) or by a defence (fitness value increasingly up).

In the second and third experiment graphs, we directly compare the attacker and defender in two different scenarios ((Mutation/Crossover at 0,35/0,65 and (Mutation/Crossover at 0,65/0,35).

You can see glimpses of the interaction between the two agents, where the attacker has peaks in fitness values where the defender can't stop its initiative and the opposite is observable too, where it is clear that the defender stopped a goal or completely threw the ball away.

It can also be interpreted that the attacker was grinding the ball while the defender was at the goal or ignoring it.

The attacker also showed to be a lot more active than the defender.

Conclusion:

In conclusion, we have developed a training simulator for attacking and defending agents within the specific football situation. Considering the data and observations, we can assess that we indeed developed attacking and defending agents.

We could also observe that the defending agent, despite being more passive most of the time, due to the variables and weights used (mainly the distance to my goal variable), this also made him more successful.

The attacking agent still had more progress to be made when it came to control the ball, but it was indeed improving and scoring goals. To improve the control of the ball, some adjustments should be made to the weights of the variables.

Despite all of this, we were also successful in creating a confrontation between the two types of agents.
