Evolutionary Computing
Task I - specialist agent

Standard Assignment - Group 68

Alexandru Dochian
2776000
adrian.dochian667@protonmail.com

Alexandros Tsiamas
2739450
alexandros.tsiamas98@gmail.com

Kalliopi Liopyraki
2739548
popiliopiraki98@gmail.com

1 INTRODUCTION

This paper presents the performance of two different evolutionary algorithms which had their genomes used as the weights of a Neural Network. The neural network output is used to generate the actions of a player in the evoman game. [**?**]. In this video game, there are 8 different enemies and our goal is to select 3 of them and apply each evolutionary algorithm to compare the results. We are interested in obtaining a good score according to the simulation fitness which is based on our player's energy, the enemy's energy and time lapsed. The algorithms were tested with 10 runs and each run includes 26 generations with 20 individuals each. That is how we can obtain more accurate results, judging the effectiveness upon the average of the simulation fitness score.

2 METHODS

2.1 Evoman Wrapper

In order to run the experiments and train our special agent, we created a wrapper over the Evoman. This process includes different functions to proceed with the experiment, such as train, test, run game, get the best individual overview, compute fitness and create generation folders to save the results as `.json` files.

2.2 Configuration

Some parameters need to be specified for each run. These parameters must be the same for all experiments, thus for both evolutionary algorithms. Tables 1, 2 and 3 present the configuration used for all experiments that we ran.

Hidden layers	10
Number of Sensors	20

Table 1: ANN config

playermode	"ai"
enemymode	"static"
level	2
speed	"fastest"
logs	"off"
player controller	None

Table 2: Enivornment config

number of genomes	*
number of parents	10
number of offsprings	10
population size	20
max generations	25
mutation chance	0.5
distribution inferior threshold	-1
distribution superior threshold	1

Table 3: EA config

* Number of genomes = (number of sensors +1) $*$ hidden layers + (hidden layers +1) $*5$

2.3 Evolutionary Algorithms

Both of our algorithms use three functions: parent selection, crossover and mutation. We tried to have completely different functions for each one.

2.3.1 Evolutionary Algorithm 1. The first evolutionary algorithm uses roulette wheel selection for the parents. It assigns a probability of selection P_i to each individual i based on the fitness value [3]. Then it returns a random choice of the individuals based on the corresponding probabilities that we compute.

$$P_i = \frac{Fitness_i}{\sum_i^n Fitness_i}$$

Then, the algorithm uses a simple crossover of 1 point, computing it by dividing the number of parents by 2. For each offspring, we select 2 parents randomly and apply the crossover. Thus, the child becomes a combination of parent 1 and parent 2 by using the genomes of the first one till the crossover point and the genomes of the second one from the crossover point till the end.

The last part is the mutation. As we described above there is a mutation chance equal to 50%. For each offspring and each genome of it, we generate a random number between 0 and 1. If this number is more than 0.50, which is the mutation chance, then we generate a normal number between -1 and 1 to add to this genome. We have to note that there are some limits (distribution superior threshold and distribution inferior threshold) which are always satisfied by using if statements. Thus, for instance, if the genome after the mutation is more than the distribution superior threshold, then we set it equal to that.

2.3.2 Evolutionary Algorithm 2. The second evolutionary algorithm uses tournament selection [1] for the parents. Our population size is 20 so we randomly choose 10 numbers (each one representing the corresponding generation) and create new lists "new_population" and "new_fitness" containing only the chosen population. We sort these lists according to fitness and then we are interested in the first 5 of the list. We assign equal probabilities(0.20%) of selection to these 5 and we return a random choice of them.

The crossover part in the second evolutionary algorithm is similar to the first one, but now we have 2 crossover points. The first one is equal to the number of parents divided by 3, and the second one divided by 2. Then we have to select randomly 3 parents and follow exactly what we did before. The child will become a combination of the parents, by using the genomes of the first parent till the first crossover point, the genomes of the second one from the first crossover point till the second and the genomes of the third one from the second crossover point till the end.

The mutation is an extension of the one in the first evolutionary algorithm. We follow the same steps for each offspring and each

genome of it, but when we find 2 genomes, that change after mutation, then we swap them in the chromosome and continue with the next offspring. This is called Swap Mutation, but our function is a bit different than the common one, since it also increases/decreases the genome but also swaps them

3 RESULTS AND DISCUSSION

Figures 1,4 and 7 show the average of max and means of fitnesses over the 25 generations. For the first algorithm, we observe that the average of the maximum fitness values is almost constant at 90 for enemies 2 and 5, while for enemy 8 is reduced to 80. For the second enemy, the average of the means prices ranges from 45 to 50, which indicates the effectiveness of the first algorithm. However, the difference with algorithm 2 in terms of efficiency is quite large with an average of means fitness scores at 20-30 for the enemies 2 and 5 and an average of max around 20 for the second enemy and 40 for the fifth, which indicates that although there are games in which the player beats these enemies, they are not enough for algorithm 2 to be reliable.

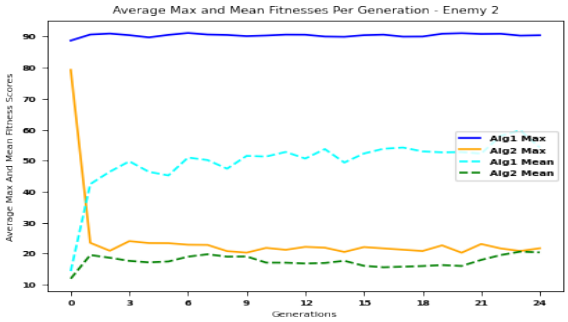

Figure 1: Average of means and maxs fitnesses - enemy 2

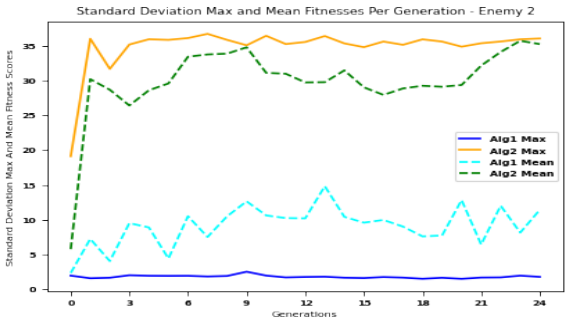

Figure 2: St.deviation of means and maxs fitnesses - enemy 2

Figures 2,5 and 8 show the standard deviation of max and means of fitnesses over the 25 generations. The standard deviation plots of max and mean fitness score for the first algorithm do not exceed 15 for most of the generations and scores generally are in a stable attitude, which means that algorithm 1 has consistently high fitness scores for the 3 enemies, with a slightly higher variation in enemy 8. On the other hand, the standard deviation scores for algorithm 2 show a lot of variation in the values, which shows that the scores are not stable. Figure 8 shows a lot of variation in the values, mostly for the second algorithm for enemy 8.

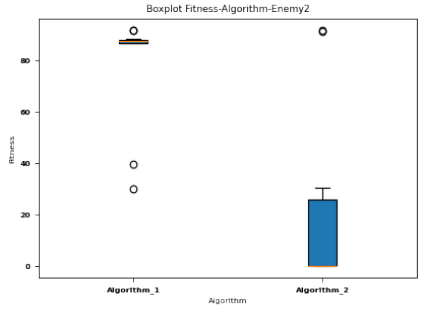

Figure 3: Boxplot fitness score - algorithm - enemy 2

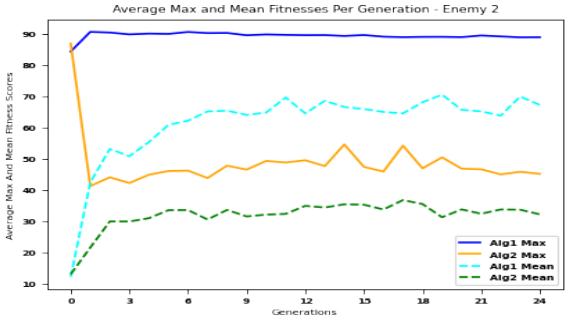

Figure 4: Average of means and maxs fitnesses - enemy 5

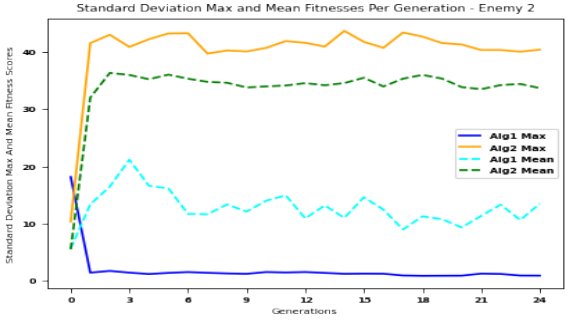

Figure 5: St.deviation of means and maxs fitnesses - enemy 5

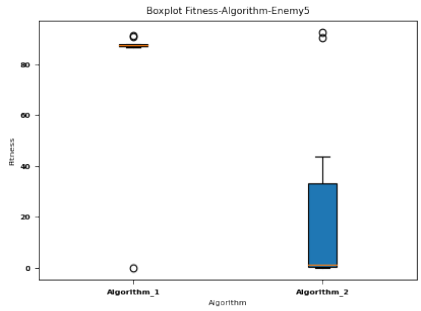

Figure 6: Boxplot fitness score - algorithm - enemy 5

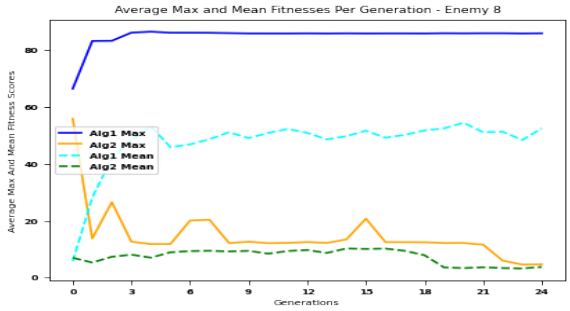

Figure 7: Average of means and maxs fitnesses - enemy 8

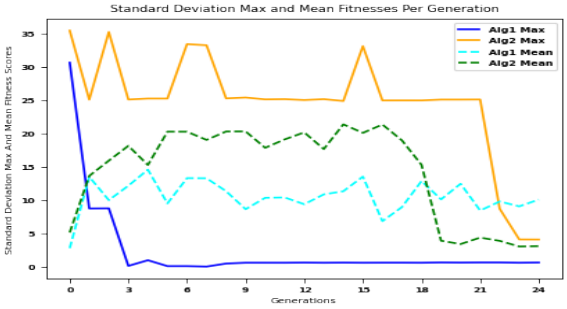

Figure 8: St.deviation of means and maxs fitnesses - enemy 8

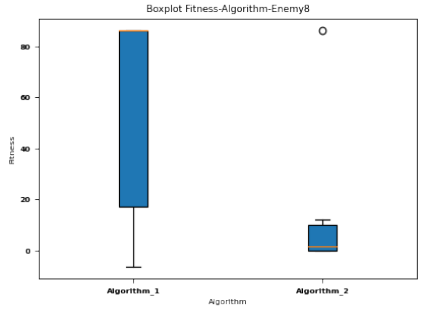

Figure 9: Boxplot fitness score - algorithm - enemy 8

Figures 3,6 and 9 are boxplots, which compare the performance of the best solutions of the 10 experiments for both algorithms, each boxplot for the 3 different enemies(2,5,8). It is clear from Figure 3 that algorithm 1 works perfectly for enemy 2, because we can see that by testing the best solutions in each run, the results are consistently good, which confirms the reliability of the first algorithm for the second enemy. However, the differences in the fitness scores are significant between the groups of best solutions for the second algorithm. We observe that algorithms 1 and 2 have a similar effect on enemy 5 as on enemy 2, with the first algorithm giving slightly better fitness results on the second enemy. Furthermore, applying the first algorithm 10 times was 70% successful, while applying the second was 10% successful for enemy 8. This can also be seen from Figure 9, where the range of best solutions of the 10 experiments is from 20 to 90 for the first algorithm, while for the second one, varies close to 0 except in one case where it is quite high.
Compared to the results from the baseline paper for enemies 2,5 and 8, the results of heuristic experiments are better than both algorithms, because the mean of our final score for the first one is approximately 65 for enemies 2 and 5 and around 20 for the enemy 8, while the heuristic experiments have around 100. Regarding the second algorithm, the final scores are quite a bit lower. For the co-evolutionary experiments, we notice that the first algorithm is better, while the second one is a little worse.

4 CONCLUSIONS

There is an improvement in the special agent and they get trained with both algorithms. However, the first one performs better than the second. Playing against enemies 2, 5 and 8, we obtained a very good simulation fitness score with Evolutionary Algorithm 1. As we can see the Roulette Wheel Selection outperforms Tournament Selection, since the most significant part of the whole process is the parent selection, which is why the first algorithm is better than the second one. The crossover functions are almost the same and the mutation chance is 50% for both cases. Evolutionary Algorithm 1 is considered to be a very good choice in order to train the artificial neural network for the special agent.

Focusing on the second Evolutionary Algorithm, as future work, we could try to decrease the number of contestants in the tournament selection and apply probabilities of selection to the winners according to their rank. That would be a combination of tournament selection and rank-based selection and it might have been good, as long as the results after applying pure linear ranking parent selection on this game were worse.

5 CONTRIBUTION TO PROJECT

- **Alexandros Tsiamas**. Evoluionary Algorithms, Report
- **Kalliopi Liopyraki**. Graphs, Results, Report
- **Alexandru Dochian**. Evoman Wrappaer, Evolutionary Algorithms

5.1 References

1. Blickle, T. (2000). Tournament selection. Evolutionary computation, 1, 181-186.
2. Evoman,
https://karinemirasblog.wordpress.com/portfolio/evoman/
3. Pencheva, T., Atanassov, K., Shannon, A. (2009). Modelling of a roulette wheel selection operator in genetic algorithms using generalized nets. International Journal Bioautomation, 13(4), 257.