# Investigating importance of mutation in genetic algorithms in a specialist agent scenario

Evolutionary Computing, Group 41
Task 1: Specialist Agent

October 1, 2021

Brian Dmyszewicz
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
2714646@student.vu.nl

Venkat Mohit Sornapudi
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
2721697@student.vu.nl

Don Mani
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
2693434@student.vu.nl

Azhar Shaikh
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
2701842@student.vu.nl

# 1 INTRODUCTION

In the recent years, the field of Artificial Intelligence has seen a rapid expansion with various approaches and algorithms being utilized in seemingly every conceivable domain. Evolutionary Computing[2] is an important branch in this ever-growing discipline. Inspired by the phenomenon of evolution, it aims to replicate the real life processes such as genetic recombination and mutation on artificial populations using various optimization-focused Evolutionary Algorithms in order to find the best solution to a given problem. Evolutionary Algorithms (EAs) have shown promising advancements in beating video games[5].

As a playground to test our EAs we use a framework called EvoMan[1]. Using the EvoMan framework we can run MegaMan II, a simple two-dimensional video game, and benchmark the EAs performance. The objective of the player in the game is to attack the enemy and defeat it by decreasing its health to 0 while avoiding enemy attacks to preserve the player's health. In the framework, the actions of the player are controlled by an Artificial Neural Network[4], which is to be trained using an EA. The goal of the EA is to produce the most optimal weights for the Neural Network, improved upon over several generations through evolutionary processes.

The two main evolutionary methods simulated by any EA are recombination (or more specifically chromosomal crossover) and mutation. Crossover occurs when 2 chromosomes (i.e. 2 strings of genes) split into multiple sections and recombine into 2 new chromosomes. Mutation, on the other hand is a stochastic process that changes the allele (variant) of the gene. In order for the EA to train the Neural Networks to defeat enemies in the EvoMan framework, different variations of those processes can be implemented.

***The research question.*** that we investigate is whether a genetic algorithm can produce optimal results without mutation in a specialist agent scenario. For this purpose, we created two flavours of the same genetic algorithm - one with, and one without mutation. The structure of this report following the introduction consists of the methods, results and conclusions sections where we explain the used algorithm and our implementation of it, present and describe obtained results and conclude on our findings.

# 2 METHODS

## 2.1 EvoMan

The Evoman framework[1] includes 8 static enemies to choose from, each one providing a different difficulty level. Various parameters can be changed throughout the framework such as the mode of the game which decides on player and enemy control methods. For the Individual Evolution mode, where the Neural Network-controlled player (agent) fights against a static enemy, one or more training objectives can be selected to train a specialist or generalist agent respectively. For the Neural Network to receive inputs, the agent has access to 20 sensors that measure its distance to the projectiles, the enemy and it's own orientation.

## 2.2 Algorithm

For the purpose of answering our research question, we use two variations of this algorithm, one with mutation called EA1, and one without mutation called EA2. Below we explain the general steps involved in the Evolutionary Algorithm.

(1) Initialization: generating the initial population consisting of individuals. Each individual is characterized by a genotype - weights of the neural network (parameters), and phenotype - the resulting neural network that outputs individual's actions. This step is only performed once, the following steps apply to every generation.
(2) Evaluation: evaluating every individual with a fitness function and assigning them a fitness score.
(3) Individual Selection: selecting best individuals to recombine to make better individual(s)
(4) Recombination: pairing a number of individuals together and exchanging parts of their genotype
(5) Mutation: randomly selecting different alleles for certain genes of selected individuals, this step is skipped for EA2
(6) Repeat the evolutionary process (3-6) till last generation
(7) Evaluate the last generation and get the best individual from all generations

## 2.3 Implementation and Motivations

In order to construct an EA, we used the tools provided by the DEAP[3] framework (Distributed Evolutionary Algorithms in Python). The exact implementation of the aforementioned algorithm including selected parameters, exact evolution operators and the motivation behind design decisions are as follows:

***Population:*** We generate the initial population consisting of 20 individuals, each of them is assigned random initial weights (parameters) using random uniform function. Those weights are used by a Neural Network to control the player. The low population size of 20 is motivated by our goal to test EAs in limited time budget and computational resource scenarios and an assumption that a high population size would make the effect of mutation negligible since enough variation would already exist in the initial genotype to reach optimal fitness scores.

***Evaluation:*** We run the EvoMan simulation on the existing population in order to evaluate each individual. The simulation uses Individual Evolution mode, where the player is controller by an Artificial Neural Network (with 10 neural nodes) trained by the EA and the static enemy follows behaviour rules defined by the game. In order to reach more meaningful results, random enemy initialization in EvoMan has been enabled. The fitness function used to evaluate individuals by assigning them a fitness score is as follows: $f = 0.9 * (100 - e) + 0.1 * p - log$, t representing the duration of the fight, p and e being the respective values of player's and enemy's remaining hit points. For sake of budget optimization, we do not reevaluate individuals whose properties were not altered during either crossover or mutation.

***Selection:*** For exploitation, we utilize DEAP's selTournament function that implements tournament selection operation, the parameters passed are: population, number of individuals to select, number of participating individuals and fitness score of individuals. The number of individuals to select is linked to the initial number of individuals to preserve the population size, it is therefore set to 20. Number of participating individuals in tournament was set to

3 based on initial testing and it seemed appropriate for our population size. With those parameters, the function randomly picks 3 individuals and saves the one with the highest fitness score, this is repeated 20 times.

***Crossover:*** For this step we pair odd and even-numbered individuals together and use DEAP's Uniform Crossover function. The function switches weights within the pairs with a defined probability per each weight. In our implementation, that probability has been set to 30% as a result of initial testing. For the purpose of fine tuning, we also decided to only apply this function to 60% of the population. We decided to use uniform crossover as opposed to other crossover methods because of its simplicity and under the assumption that it would represent performance of other methods well enough for us to investigate the research question.

***Mutation:*** *(when applies).* We use DEAP's Gaussian Mutation function to randomize 30% of an individual's properties. We only apply this process to 20% of the population in order to avoid overly drastic changes in fitness values which could potentially slow down the process of evolution. The standard deviation has been set to 1 and the mean to 0 in order to avoid generating too many extreme values which we speculated could be disadvantageous when dealing with relatively small population size. We used the Gaussian mutation method due to it's simplicity. Furthermore, since the genotypes are real-valued floating point numbers, it made sense to use Gaussian Mutation. As needed to answer the research question, the second version of our algorithm omits this step entirely by changing the mutation rate from 20% to 0%. Table 1 below lists all the parameter values used.

**Table 1: Parameter values**

| Parameter | Values |
|---|---|
| Representation | Real-valued Vectors |
| Population Size | 20 |
| Number of Generations | 15 |
| Selection | Tournament (tournsize=3) |
| Crossover | Uniform (indpb=0.3) |
| Crossover Probability | 0.6 |
| Mutation | Gaussian ($\sigma = 1$, $\mu = 0$, indpb=0.3) |
| Mutation Probability | 0.2 for EA1, 0 for EA2 |

## 2.4 Experiment Design

We randomly chose 3 of the 8 enemies of the Evoman framework for the purpose of our experiment, they are Airman (enemy 2), Metalman (enemy 5), and Quickman (enemy 8). We used both versions of our EA - with and without mutation and ran it 10 times against every selected enemy. We obtain the means of each generation's maximum fitness scores, average fitness scores and standard deviations over the 10 completed runs for each EA and every enemy.

To study the average behaviour of the best individuals from all runs, using each algorithm, they were tested 5 times against the same enemy that they were trained to fight. Their average individual gain was obtained by subtracting the remaining enemy life from remaining player life and calculating the average over 5 runs.

# 3 RESULTS AND DISCUSSION

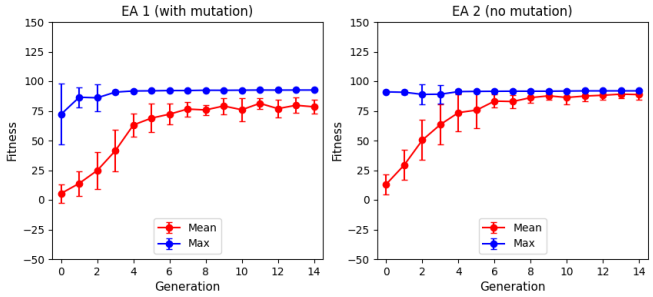


**Figure 1: Line plots for enemy 2 (Airman)**

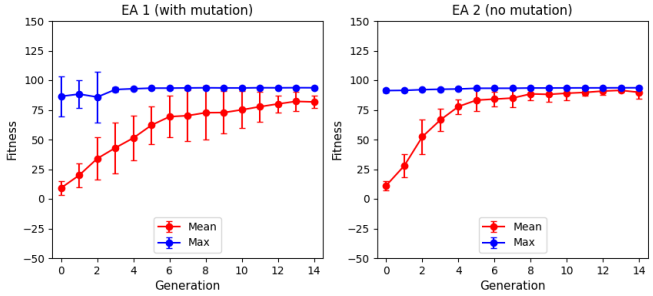


**Figure 2: Line plots for enemy 5 (Metalman)**

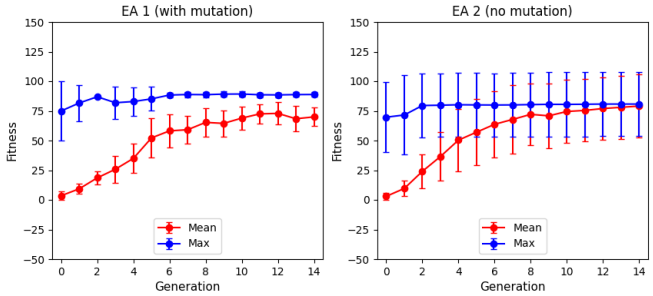


**Figure 3: Line plots for enemy 8 (Quickman)**

The line plots portray the 10 run average of maximum and mean fitness values of the population for each generation. For every generation, the graphs also portray the standard deviation. In all three line plots, we observe that, for EA2 (no mutation), the mean fitness of the population converges to the max fitness (best individual) quickly and plateaus to a local maximum. In contrast, in case of EA1, due to the randomness associated with mutation, some individuals remain different from rest of the population across all generations

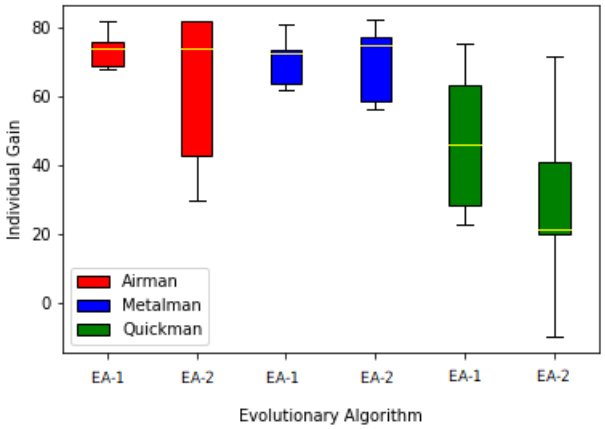


**Figure 4: Box plots**

and therefore, the mean fitness can never converge to the maximum fitness. In case of the two enemies - Airman and Metalman, both evolutionary algorithms seem to perform very similarly in their search for an individual with the highest fitness score. In case of Quickman, EA1 manages to produce an individual with a better fitness score than EA2. We speculate that this small discrepancy has to do with the enemy difficulty, which we assume to be greater for Quickman.

Individual gain is obtained by subtracting the remaining enemy life from remaining player life. The best individual from each of the 10 runs, fights 5 times against each enemy to get the average individual gain. The distribution of these gains across 10 runs is portrayed in the box plots. Upon observation of the box plot, we find that, on average, both EAs do reasonably well against all enemies. This is most likely due to well tweaked parameters and a sufficient number of generations and individuals. Both EA1 and EA2 seem to have higher individual gains when facing Airman and Metalman than Quickman. The reason for this is attributed to difficulty, like in case of the line plots. For Airman and Metalman, the mean of individual gain is very similar. However, by observing the distance between the whiskers, we can also observe that the standard deviation for EA2 is always greater than that of EA1. This is probably linked to the random enemy initialization factor and could mean that mutation helps evolve more robust individuals with higher adaptability. This could also explain why EA2 produces a significantly lower average individual gain than EA1 when facing Quickman, even to the point where it's (EA2) individual gain values sometimes go below 0, indicating the loss of the player. This shows that the best individual produced by EA1 is more consistent than in case of EA2.

***Comparison to Miras (2016):*** From table 2, we observe that the final scores of our EAs against the chosen enemies (Airman, Metalman, Quickman) are on par with the reported scores of the Genetic Algorithm used in the first experiment of [1]. While [1] ran across 100 generations for population size of 100, we ran across 15 generations for population size of 20. We chose these particular parameters as, after running multiple experiments, we did not see much fitness or gain improvement by increasing them. Moreover, keeping them low has helped us to optimise with a low budget (time and computation).

**Table 2: Player Final Scores compared to Miras(2016)**

| Enemy | GA by Miras | EA1 | EA2 |
|---|---|---|---|
| Airman | 90 | 82 | 82 |
| Metalman | 80 | 85.60 | 86.80 |
| Quickman | 77 | 75.40 | 71.80 |

Interestingly, our EAs also converge to the same strategy (simply jumping and shooting without moving forward or backward) against Airman and Metalman as observed in [1].

## 4 CONCLUSIONS

By observing the strategy both of the EA's developed against Airman and Metalman and considering their higher average individual gains, we can conclude that those enemies provide a lower level of difficulty while Quickman proves more difficult and requires a more advanced strategy to beat while also producing lower individual gains. Based on the experiment's results and our observations regarding them, we also see that the effects of mutation on the fitness value of the best individual across all generations are quite minor against enemies that we concluded to offer a low level of difficulty. In case of the more difficult enemy (Quickman), mutation is necessary to reach an optimal score. This leads us to conclude that mutation ensures the necessary variety of alleles in a population, preventing the fitness score from converging to a suboptimal local maximum, much like it did in case of our mutation-less algorithm. We can also observe that implementing mutation reduces the mean fitness values of a population. This, however, is not very relevant in search of the best possible individual and can be corrected using individual selection. By observing lower standard deviation for an algorithm utilizing mutation and its much higher average individual gain against the hardest enemy, we are also able to conclude that mutation helps produce more robust and consistent individuals, with higher levels of adaptability, as shown through the use of random enemy initialization, though in an otherwise strictly specialist scenario. All this leads us to the conclusion that mutation is necessary for an evolutionary algorithm to produce optimal results in a specialist agent scenario, given sufficient difficulty of the problem. This conclusion holds true for all the previously described methods and parameters. To generalize it, we would need to perform tests on a broader spectrum of problems and models.

## 5 CONTRIBUTIONS

**Azhar** was involved in experiment code development, refactoring of code for ease of running experiments, analysis of resulting data and editorial work. **Brian** was involved in initial algorithm design, data analysis, documentation, report creation and editorial work. **Don** was involved in creation of plotting code for visualizing data in line plots and box plots, running experiments, data analysis and editorial work. **Mohit** was involved in experiment code development, plotting of data, data analysis and editorial work.

## REFERENCES


[1] Karine da Silva Miras de Araújo and F. O. D. França. 2016. Evolving a generalized strategy for an action-platformer video game framework. In *CEC*.

[2] Agoston E Eiben, James E Smith, et al. 2003. *Introduction to evolutionary computing*. Vol. 53. Springer.

[3] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* 13 (jul 2012), 2171–2175.

[4] Frank Rosenblatt. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* 65, 6 (1958), 386.

[5] Julian Togelius, Sergey Karakovskiy, Jan Koutník, and Jurgen Schmidhuber. 2009. Super mario evolution. In *2009 ieee symposium on computational intelligence and games*. IEEE, 156–161.