National University
Of Computer and Emerging Sciences

AI Project

Submitted by:

Section Q:

Taimoor Anwar i21-1232

Abdul Rehman i21-1133

Ghulam Mohiuddin i21-130

Submitted to:

Ma’am Shahela Saif

Project Report: Autonomous Driving Simulation

Project Overview

This project involves creating a simulation for an autonomous vehicle navigating through a course. The primary goal is to develop a model that can autonomously drive a car around a track, avoiding obstacles like potholes and adhering to traffic signals, using neural networks to process inputs and make decisions.

Technologies Used

Python programming language was used for developing the simulation with the Pygame library for rendering the graphical user interface and handling game dynamics. The NEAT (NeuroEvolution of Augmenting Topologies) algorithm was utilized to evolve and optimize neural networks for decision-making in the simulation.

Code Analysis

The simulation sets up a graphical interface where a car navigates a track with defined checkpoints and obstacles. The car's movement is controlled by a neural network that receives input from sensors (radars) detecting the proximity to the track edges and obstacles. Traffic lights and potholes are simulated as environmental challenges that the car must respond to.

Key functionalities include initializing the Pygame window, loading assets, defining the car's behavior such as movement, rotation, and collision detection, and utilizing a NEAT-configured neural network to evolve the car's driving strategy over generations. Fitness scores are assigned based on the car's performance, such as the number of checkpoints crossed without collisions.

Results

The expected result of this simulation is that over generations, the neural networks evolve to improve the car's ability to navigate the course with fewer collisions and higher efficiency. The simulation outputs data on each car's fitness and the overall generation's performance, which can be used to track the evolutionary progress of the driving strategies.

Introduction

In the simulation project for autonomous driving, calculating the fitness of cars is crucial for evaluating their performance and effectiveness in navigating a track autonomously. Fitness calculation is intended to measure how well a car adheres to the desired behaviors, such as speed, avoiding obstacles, and following traffic rules.

Methodologies Attempted

Several methodologies were attempted to calculate the fitness of the cars accurately, each with unique challenges:

1. **Velocity-Based Fitness**: Initially, fitness was measured based on whether the car was moving (velocity > 0). This method proved problematic as some cars began moving in circles or drifting, which technically counted as movement but did not contribute positively towards the actual goal of navigating the track efficiently.

2. **Survival Time Fitness**: Another approach was to measure fitness based on how long a car could survive without colliding. While this method rewarded endurance, it failed to account for cars that would stop moving altogether, thus surviving by inactivity rather than skillful navigation.

3. **Checkpoint-Based Fitness**: The final method introduced checkpoints throughout the track. Cars gain fitness points for each checkpoint collected. This method aligns more closely with the objective of navigating the entire track, as it requires moving towards specific targets and not just avoiding crashes.

Challenges Encountered

Each method brought its own set of challenges. The velocity-based approach did not effectively discriminate between productive movement and unproductive behaviors like drifting. The survival time method did not motivate cars to keep moving, allowing for exploitation by remaining stationary. These methods illustrated the complexity of defining fitness in a dynamic and interactive environment such as autonomous driving simulations.

Optimal Solution

The checkpoint-based fitness calculation was ultimately adopted as it best matched the project's goals. This method incentivizes purposeful navigation, encourages exploration of the track, and directly measures progression towards completing the track, making it the most suitable metric for this simulation.

Data Analysis and Visualization

Project Overview

This segment of the project focuses on analyzing the performance data of the neural network-driven autonomous vehicles from the simulation. The data includes generations, average fitness values, and population sizes, which are crucial for evaluating the efficiency of the neural network configurations over time.

Technologies Used

Python is used as the primary programming language, leveraging the pandas library for data manipulation and the matplotlib library for visualizing the data. This combination allows for effective analysis and graphical representation of the simulation results.

Code Analysis

The code reads performance data from a text file, extracts and stores relevant metrics into lists, and computes overall statistics such as total average fitness and population size. A key feature of the analysis is the calculation of a performance ratio, which provides insight into the overall efficiency of the genetic algorithm across all generations.

Visualization is performed using a bar chart, which not only displays the average fitness per generation but also includes the overall performance ratio as a separate bar for comparative analysis. This visual representation helps in quickly assessing the evolutionary progress of the simulation.

Results

The expected result is a clear visual representation of the average fitness values across generations, complemented by a performance ratio that indicates the efficiency of the simulation. This visualization serves as a critical tool for stakeholders to gauge the effectiveness of the neural network algorithms over time and make informed decisions about future simulations or adjustments in the neural network configurations.

Testing Stages:

Config.txt file contain every variable that can effect the performace so we will change some variable and plot result.

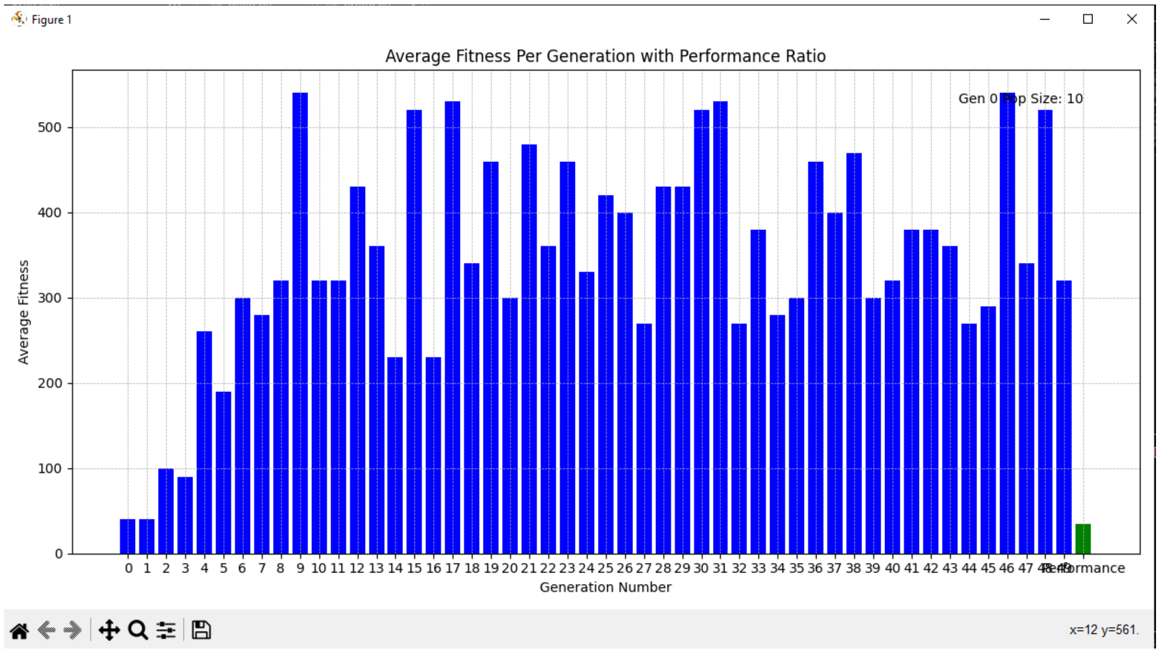
Testing Stage 1:

pop_size = 10

Result Obtain:

Generation executed =50

But no car complete the track



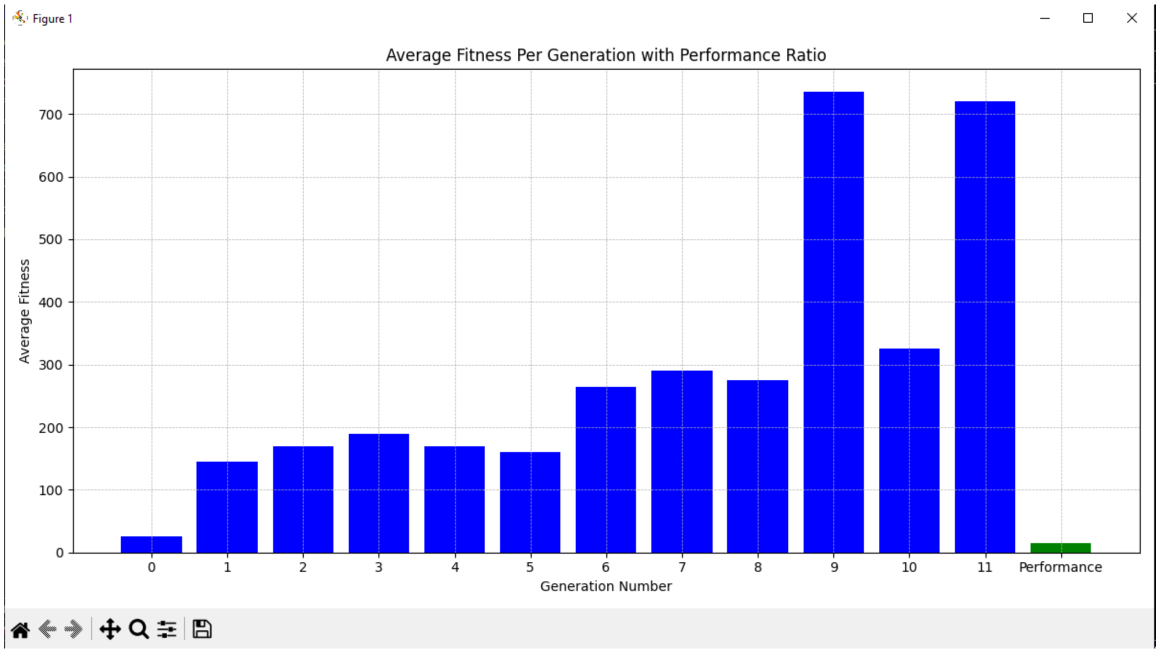
Testing Stage 2:

pop_size = 20

Result Obtain:

Generation executed =11

1 car complete the track



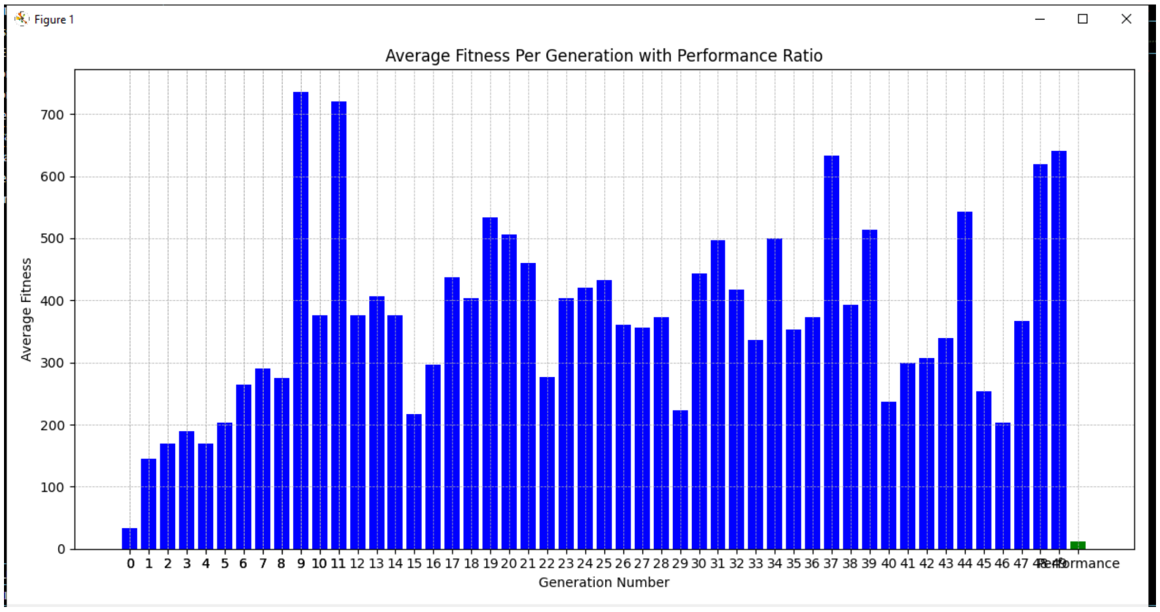
Testing Stage 3:

pop_size = 30

Result Obtain:

Generation executed =50

no car complete the track



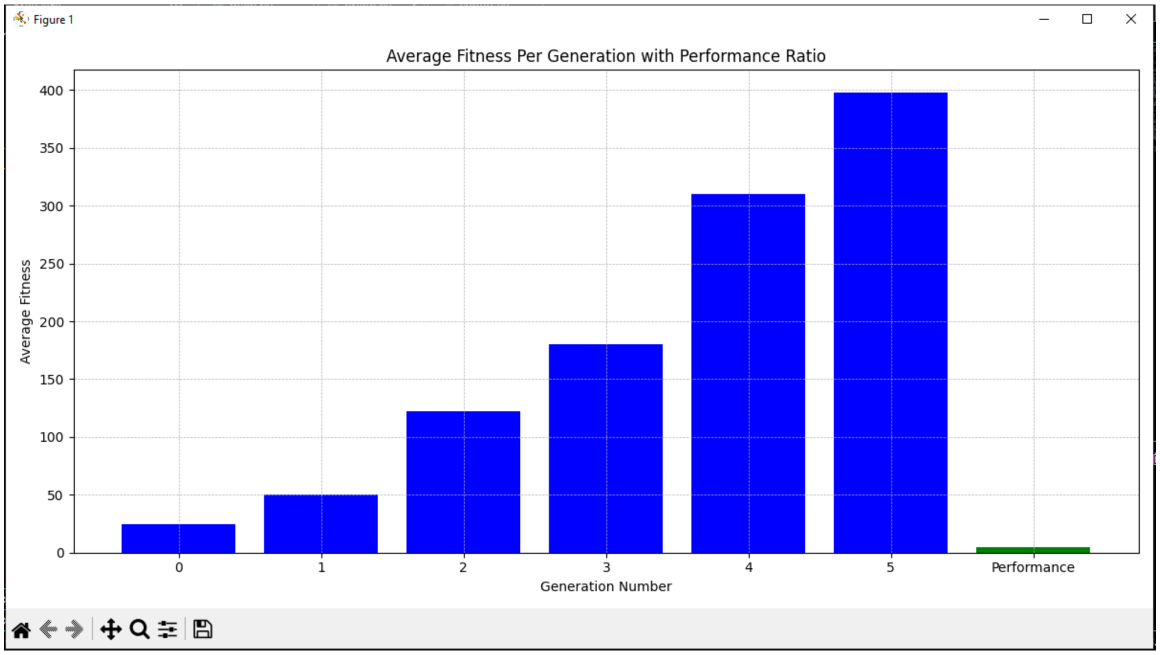
Testing Stage 4:

pop_size = 40

Result Obtain:

Generation executed =5

1 car complete the track



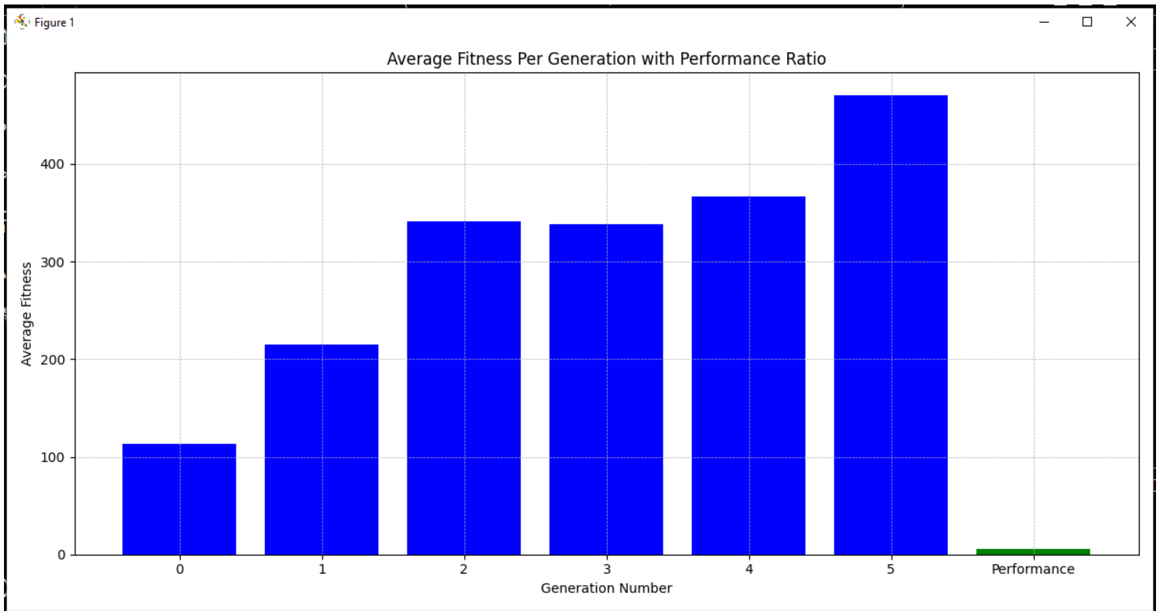
Testing Stage 5:

pop_size = 60

Result Obtain:

Generation executed =5

1 car complete the track



Testing Stage 6:

pop_size = 60

bias_mutate_power inc by 0.1

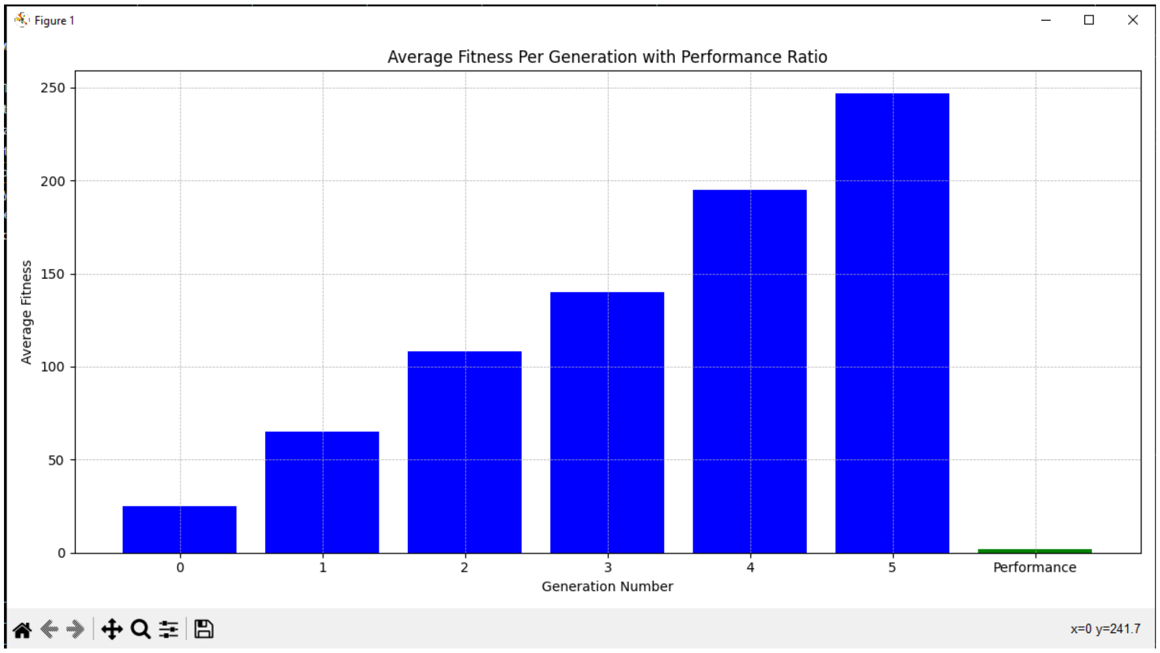
bias_mutate_rate inc by 0.1

bias_replace_rate inc by 0.1

Result Obtain:

Generation executed =5

2 car complete the track



Testing Stage 7:

pop_size = 60

bias_mutate_power inc by 0.2

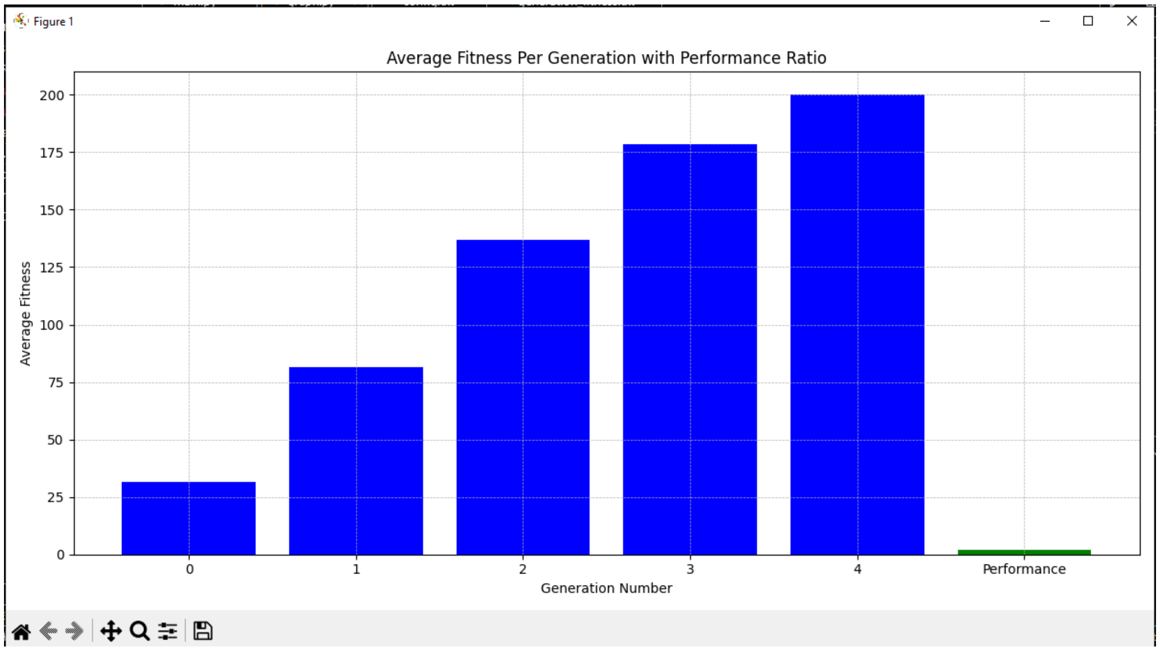
bias_mutate_rate inc by 0.2

bias_replace_rate inc by 0.2

Result Obtain:

Generation executed =4

1 car complete the track



Testing Stage 8:

pop_size = 60

bias_mutate_power inc by 0.3

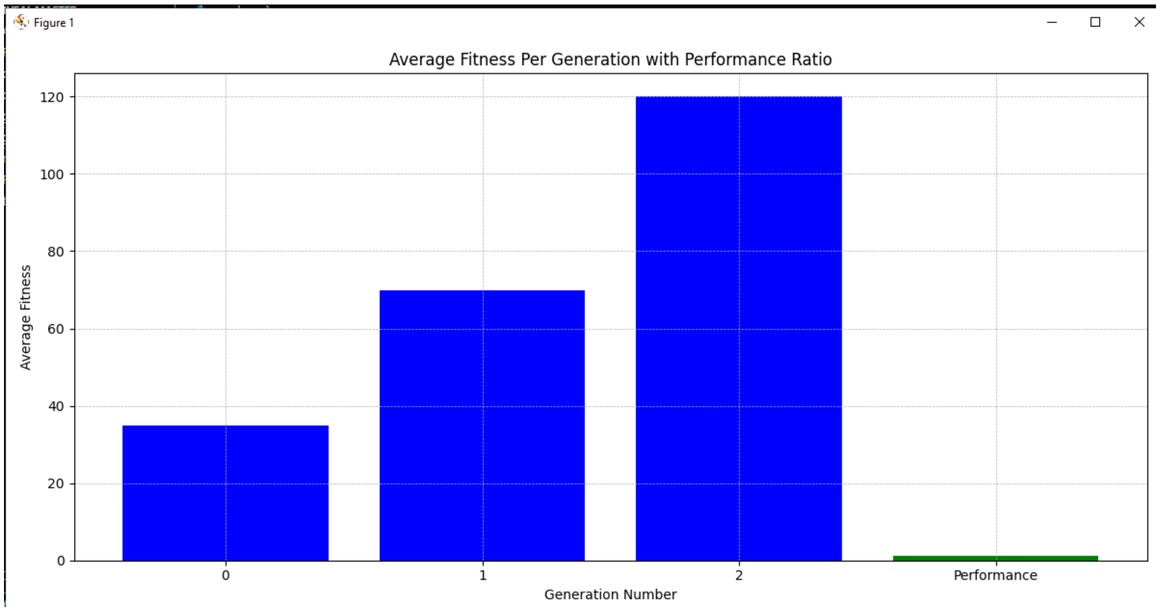
bias_mutate_rate inc by 0.3

bias_replace_rate remain unchanged 0.1

Result Obtain:

Generation executed =2

1 car complete the track



Testing Stage 9:

pop_size = 60

response_mutate_power inc 0.5

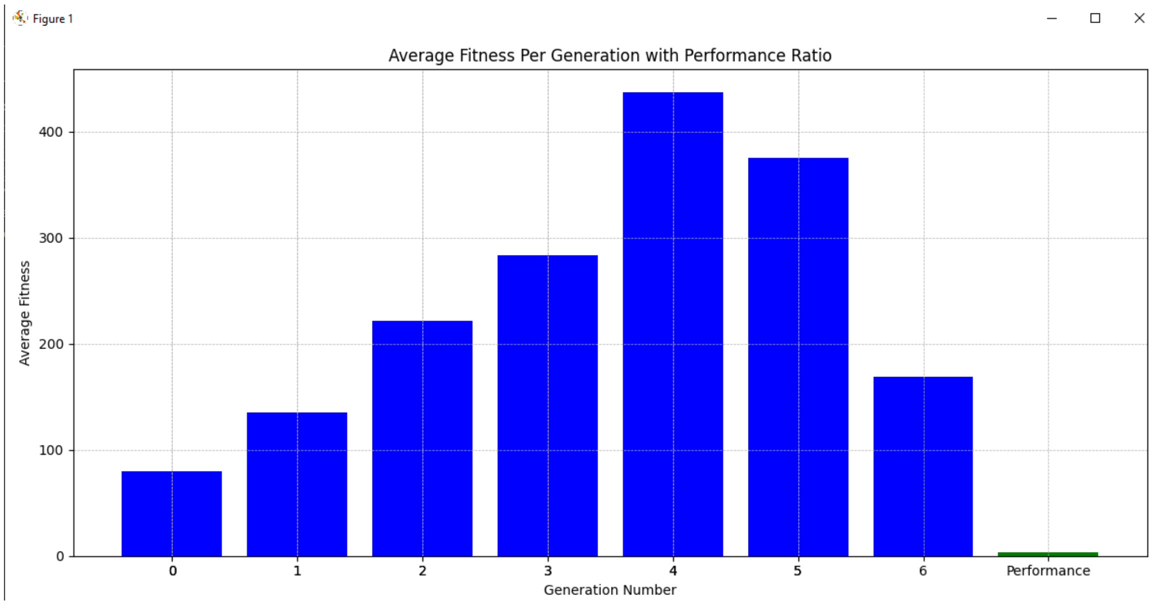
response_mutate_rate inc 0.5

response_replace_rate inc 0.5

Result Obtain:

Generation executed =6

1 car complete the track



Testing Stage 10:

pop_size = 60

weight_mutate_power inc 0.1

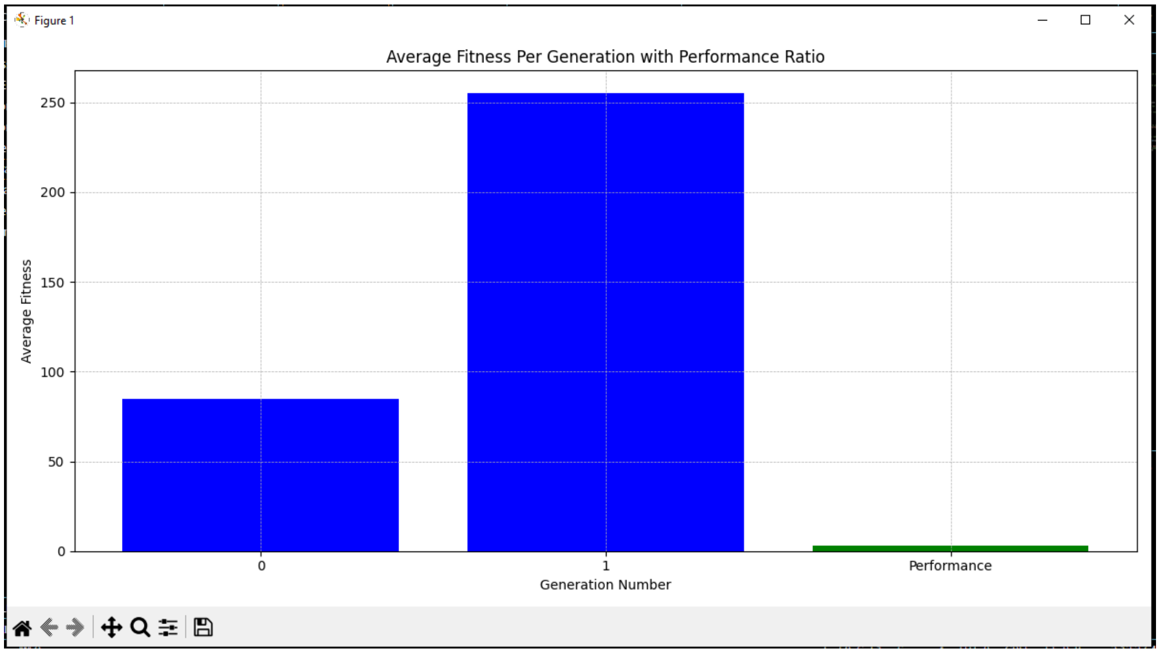
weight_mutate_rate inc 0.1

weight_replace_rate inc 0.1

Result Obtain:

Generation executed =1

1 car complete the track



Testing Stage 11:

pop_size = 60

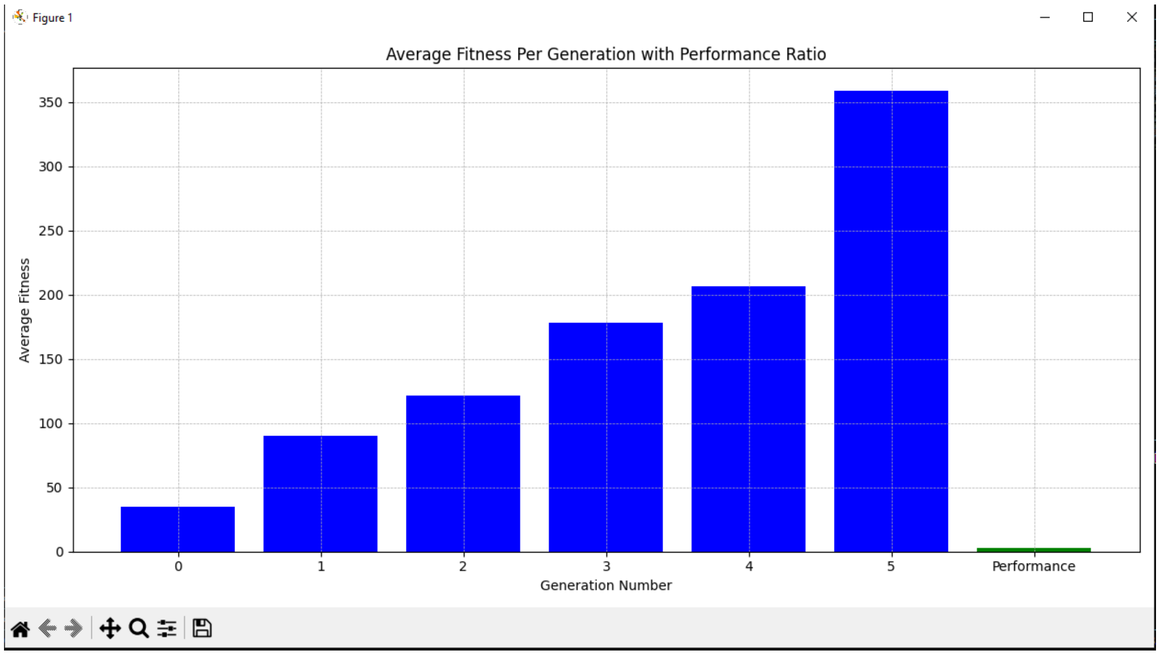
max_stagnation inc by 20

species_elitism inc by 2

Result Obtain:

Generation executed =5

1 car complete the track



Testing Stage 12:

pop_size = 60

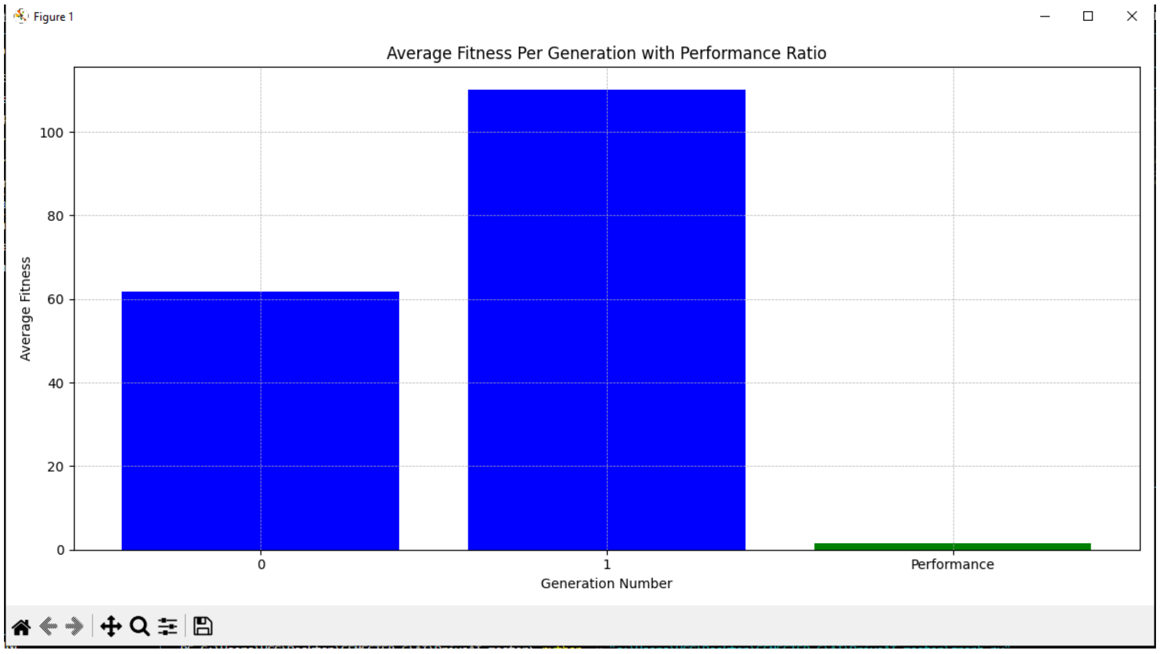
elitism inc by 2

survival_threshold inc by 0.2

Result Obtain:

Generation executed =1

1 car complete the track



Analysis and Conclusions

Overview of Testing Stages:

The simulation's testing stages involved modifying various parameters in the config.txt file to observe their effects on performance. The main criteria for evaluating performance were the number of generations executed and the number of cars that completed the track.

Results from the Testing Stages:

Population Size (pop_size): As pop_size increased, there was a noticeable reduction in the number of generations needed to complete the track, indicating more effective learning with a larger population, but this trend plateaued at a population size of 60.

Mutation Parameters: Incremental increases in mutation parameters such as bias_mutate_power, bias_mutate_rate, and bias_replace_rate led to a reduced number of generations and increased the number of cars completing the track, suggesting that higher mutation rates might be facilitating more effective exploration of the solution space.

Stagnation and Elitism Adjustments: Adjusting parameters related to elitism and survival thresholds, specifically max_stagnation, species_elitism, elitism, and survival_threshold, also resulted in fewer generations executed and maintained or improved track completion, indicating that allowing better performers to survive longer or in greater numbers encourages overall performance improvement.

Detailed Conclusions:

Effect of Population Size: A larger population provides a wider genetic pool which enhances the algorithm's ability to explore different strategies efficiently. However, beyond a certain point (pop_size = 60), further increases do not yield proportional improvements, indicating a possible optimal population size for this specific simulation setup.

Impact of Mutation Rates: Increasing mutation rates generally led to improved performance. This can be attributed to the introduction of new genetic variations, allowing the algorithm to explore new and potentially more effective strategies. The balance between exploration (through mutation) and exploitation (through crossover and selection) is crucial.

Influence of Elitism and Stagnation Settings: Increasing the parameters related to elitism and extending stagnation periods allowed better strategies more time to evolve without being prematurely discarded. This indicates that the survival of the fittest, along with an extended period to prove effectiveness, is beneficial in environments with complex navigational challenges.

Overall Conclusion: Adjusting the config.txt variables shows that the performance of autonomous cars in the simulation can be significantly influenced by genetic algorithm settings. Optimal settings depend on a balance between genetic diversity (achieved through higher mutation rates and larger populations) and focused evolutionary pressure (achieved through selective elitism and extended stagnation periods). The results suggest that carefully tuning these parameters can lead to more efficient learning and better overall performance in terms of quicker generations and more cars completing the track. Future simulations should consider dynamic adjustment of these parameters based on real-time performance metrics to continuously optimize learning outcomes.