CM3020 Midterm Part B Report

In this project, the goal is to adapt the existing evolutionary algorithm so that the creatures can evolve to complete a new and challenging task: climbing a mountain. This task involves not only a change in the creatures' objective but also their integration into a newly created environment specifically designed for this purpose.

The sandbox arena and the mountain are adapted from the existing codebase.
I have tried several methods to further develop creatures that progressively learn and adapt to efficiently navigate and eventually master the complex task of mountain climbing.

This involves adjusting the fitness function to reward behaviours that contribute to successful climbing and possibly altering the creatures' genetic representation to better suit the new task.

Through iterative testing and evolution, I aim to witness the emergence of creatures that demonstrate the remarkable ability to climb the mountain, showcasing the poIr of genetic algorithms in evolving solutions to complex problems.

Overview of the Genetic Algorithm

Genomic Representation: Creatures in the simulation are encoded as genomes, which are essentially sequences of genes that determine their structural and behavioural attributes. These genes define various aspects such as the type and length of joints and the mechanisms by which they are controlled.

Mutation Mechanisms:

1. **Point Mutation**: This process involves the random alteration of specific genes within a genome. It occurs according to a predefined mutation rate and alters the gene's characteristics to a certain degree.

2. **Shrink Mutation**: This mutation type probabilistically removes genes from the creature's genome, effectively simplifying its structure.

3. **Grow Mutation**: Conversely, this mutation adds new genes to the genome at certain probabilities, thus increasing the creature's complexity.

4. **Crossover Mechanism**: This is a genetic operation that combines the genetic information of two parent creatures to create offspring. It's a way to mix and match parent traits in hopes of producing superior progeny that inherit the best characteristics from both parents

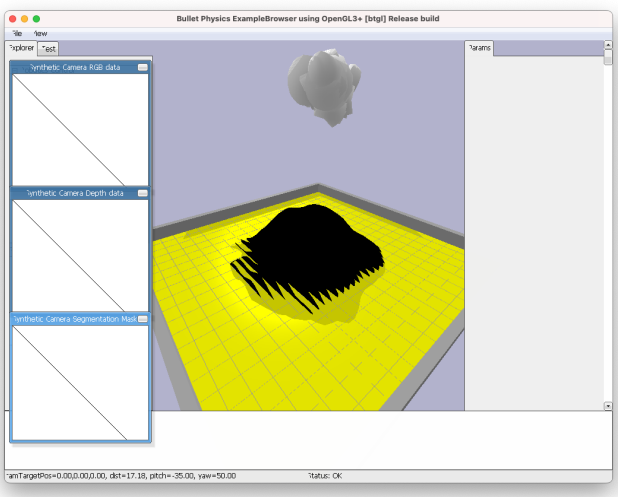
Overview of the Fitness Evaluation:

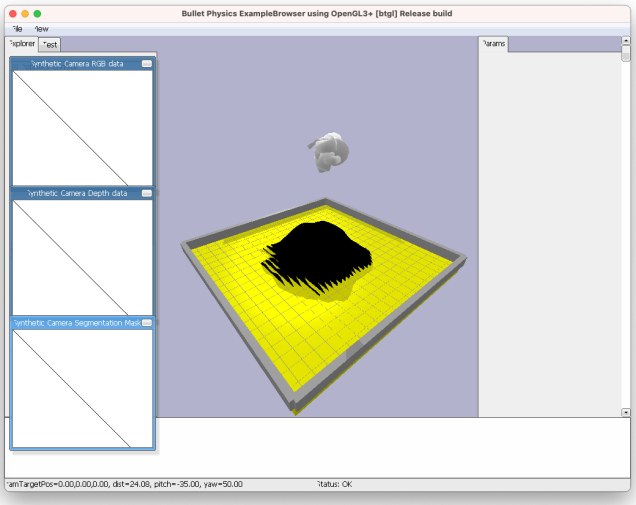
1. **Fitness Evaluation**: The fitness function is crucial as it assesses each creature's efficacy in the simulation, primarily focusing on metrics like the distance travelled among other locomotion indicators.

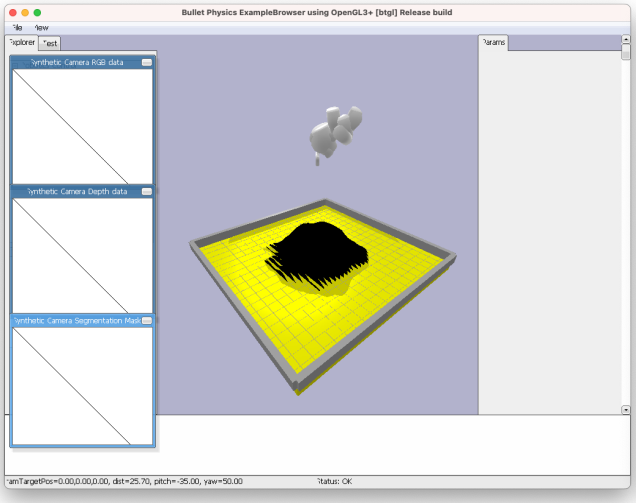
2. **Selection Strategy**: The process employs a fitness-proportional selection technique, often referred to as roulette wheel selection. This method probabilistically selects creatures to become parents of the next generation based on their fitness levels, favouring those with higher performance.

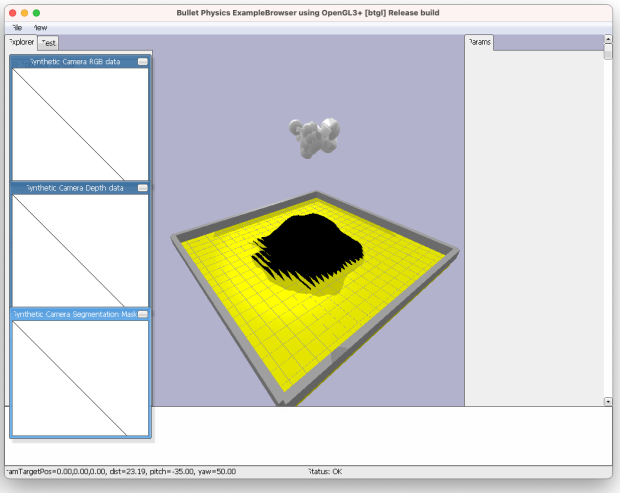
3. **Preservation of Elite**: The approach incorporates elitism by reserving a segment of the top-performing creatures from the existing population. This ensures their traits are carried over to subsequent generations, thereby fostering a gradual evolution towards more adept and higher fitness solutions.

Screenshots:









Findings

The aim of this research is to investigate how varying the settings of the genetic algorithm impacts the evolutionary process. The algorithm's performance is examined and documented across different setups of parameters, mutation rates, and population sizes. For each set of parameters, multiple iterations are executed, and the evolution's pace is gauged by tracking the rise in the population's average fitness across generations. The outcomes from each set of parameters are systematically recorded into CSV files and illustrated using pandas and matplotlib, providing a clear visual representation of how each parameter influences the rate of evolution. The conclusions drawn from these experiments are intended to identify the most effective configuration strategy for enhancing the genetic algorithm's rate of evolution.

Experiments

To investigate the impact of mutation rates on the experiment, I assess the average fitness (measured as the distance travelled) in relation to the number of generations (time elapsed). I vary the mutation rates using three types: point mutation, shrink mutation, and grow mutation, ranging from 0.01 to 0.25. If the experiment involves a mutation type not specified, I default to a mutation rate of 0.25.

To ensure fairness, I maintain a population size of 10 individuals with 3 genes each, and the program will utilise 4 threads. I continue measuring fitness until the population of genomes has evolved over 100 generations.

Mutation	Mutation Rate	Gene Count	Population Size	Thread Used	Generation
Point Mutate	0.01 - 0.25	3	10	4	100
Shrink Mutate	0.01 - 0.25	3	10	4	100
Grow Mutate	0.01 - 0.25	3	10	4	100

Experiment Setup (Population)

In the population size experiment, I evaluate average fitness (distance traveled) relative to the number of generations (time elapsed). The population size vary from 2 to 10, while maintaining a consistent mutation rate of 0.25 across all mutation types, a gene count of 3, and the use of 4 threads. The experiment concludes once the population of genomes has evolved over 100 generations.

Population size	Shrink_mutate	Grow_mutate	Thread Used	Generation
2-10	0.25	0.25	4	100

Evaluation Results (generated from Jupyter notebook):

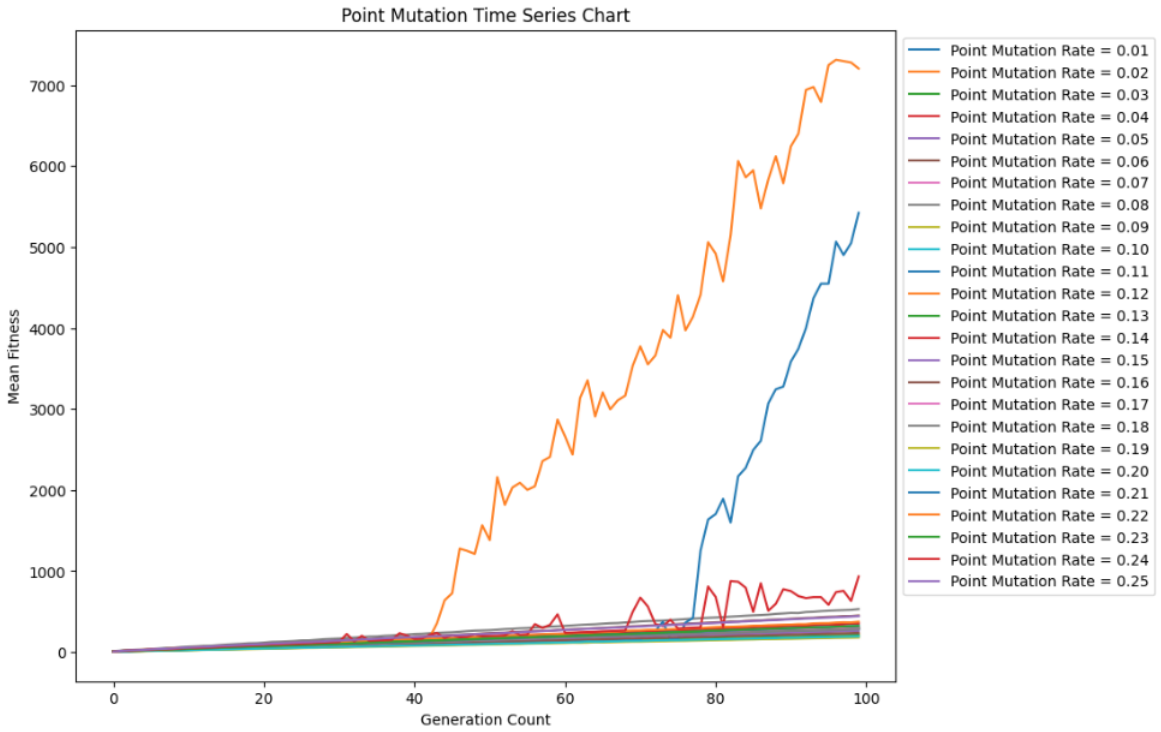
```
import pandas as pd
import matplotlib.pyplot as plt
```

```
files=["pm"+ str(i+1) +".csv" for i in range(25)]
dfs=[]
for file in files:
    df = pd.read_csv("../mutationdata/point_mutation/"+file,
                     header=None)
    dfs.append(df)
```

```
plt.figure(figsize=(10, 8))

for i, df in enumerate(dfs):
    plt.plot(df.iloc[:, 0], df.iloc[:, 1],
             label=f"Point Mutation Rate = {float(i+1)/100:.2f}")

plt.legend(bbox_to_anchor=(1, 1), loc='upper left')
plt.xlabel("Generation Count")
plt.ylabel("Mean Fitness")
plt.title("Point Mutation Time Series Chart")
plt.show()
plt.show()
```

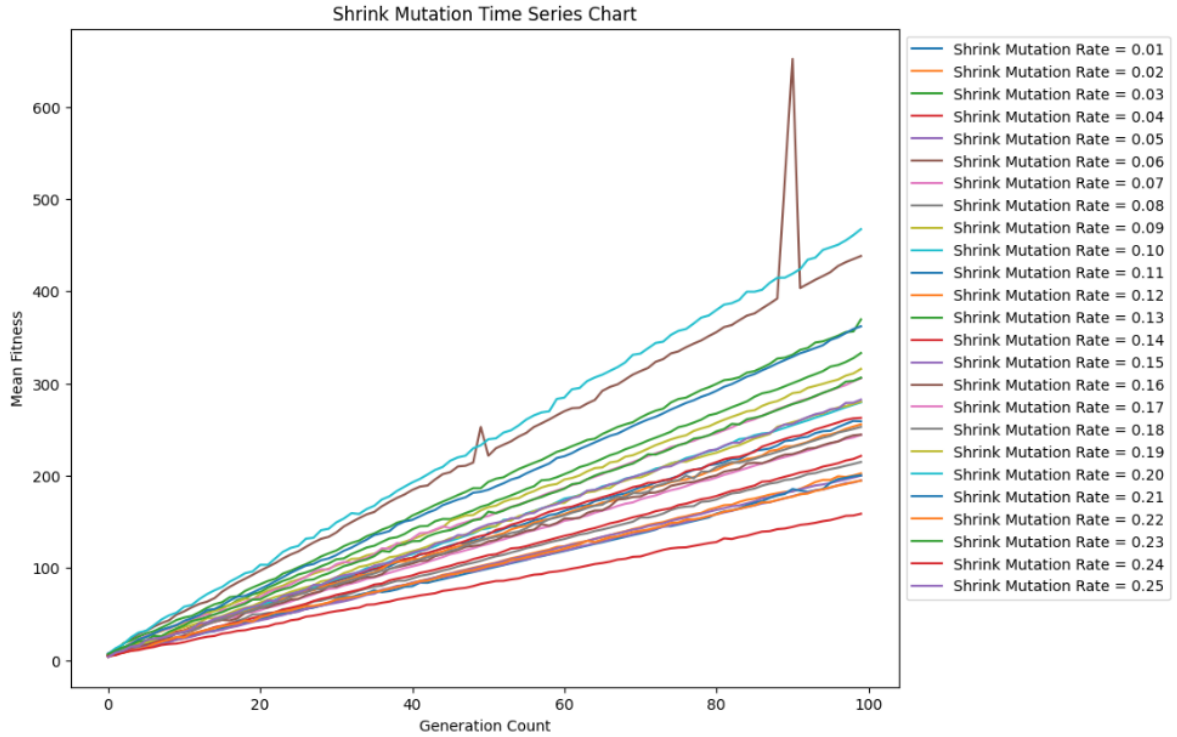


```
files=["sm"+ str(i+1) +".csv" for i in range(25)]
dfs=[]
for file in files:
    df = pd.read_csv("../mutationdata/shrink_mutation/"+file,
                     header=None)
    dfs.append(df)
```

```
plt.figure(figsize=(10, 8))

for i, df in enumerate(dfs):
    plt.plot(df.iloc[:, 0], df.iloc[:, 1],
             label=f"Shrink Mutation Rate = {float(i+1)/100:.2f}")

plt.legend(bbox_to_anchor=(1, 1), loc='upper left')
plt.xlabel("Generation Count")
plt.ylabel("Mean Fitness")
plt.title("Shrink Mutation Time Series Chart")
plt.show()
plt.show()
```

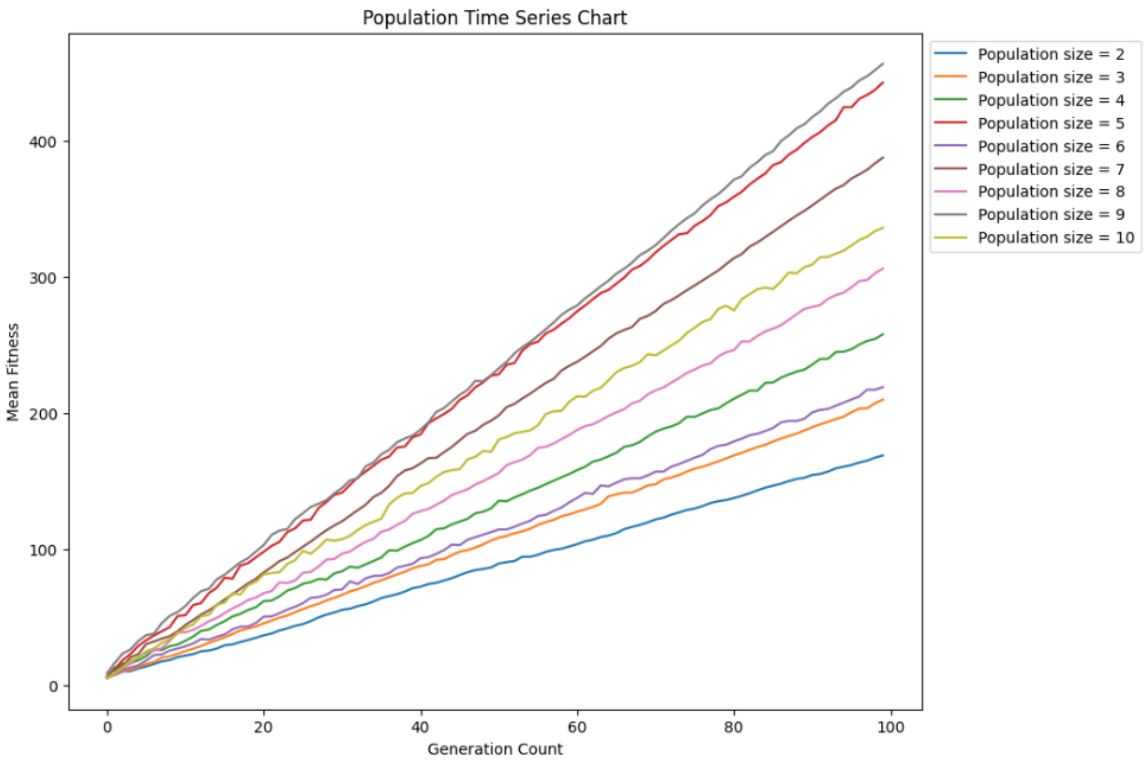



```
files=["gm"+ str(i+1) +".csv" for i in range(25)]
dfs=[]
for file in files:
    df = pd.read_csv("../mutationdata/grow_mutation/"+file,
                     header=None)
    dfs.append(df)
```

```
plt.figure(figsize=(10, 8))

for i, df in enumerate(dfs):
    plt.plot(df.iloc[:, 0], df.iloc[:, 1],
             label=f"Grow Mutation Rate = {float(i+1)/100:.2f}")

plt.legend(bbox_to_anchor=(1, 1), loc='upper left')
plt.xlabel("Generation Count")
plt.ylabel("Mean Fitness")
plt.title("Grow Mutation Time Series Chart")
plt.show()
plt.show()
```



Conclusion

Observations from the Mutation Rate Experiment:
When examining the effects of point mutation, I observed that at low point mutation rates, there was a significant increase in mean fitness. This resulted in creatures capable of covering substantial distances when simulated. It appears that the influence of shrink mutation and grow mutation indirectly contributes to such outcomes, particularly when their rates are high and point mutation is low.

Another noteworthy observation from the mutation rate experiments is that when a specific mutation rate is low, it tends to lead to higher mean fitness.

Observations from the Population Size Experiment:
Analysing the distribution of mean fitness, I also noticed that smaller population sizes tended to result in loIr mean fitness, whereas larger population sizes led to higher mean fitness.