BSc Computer Science

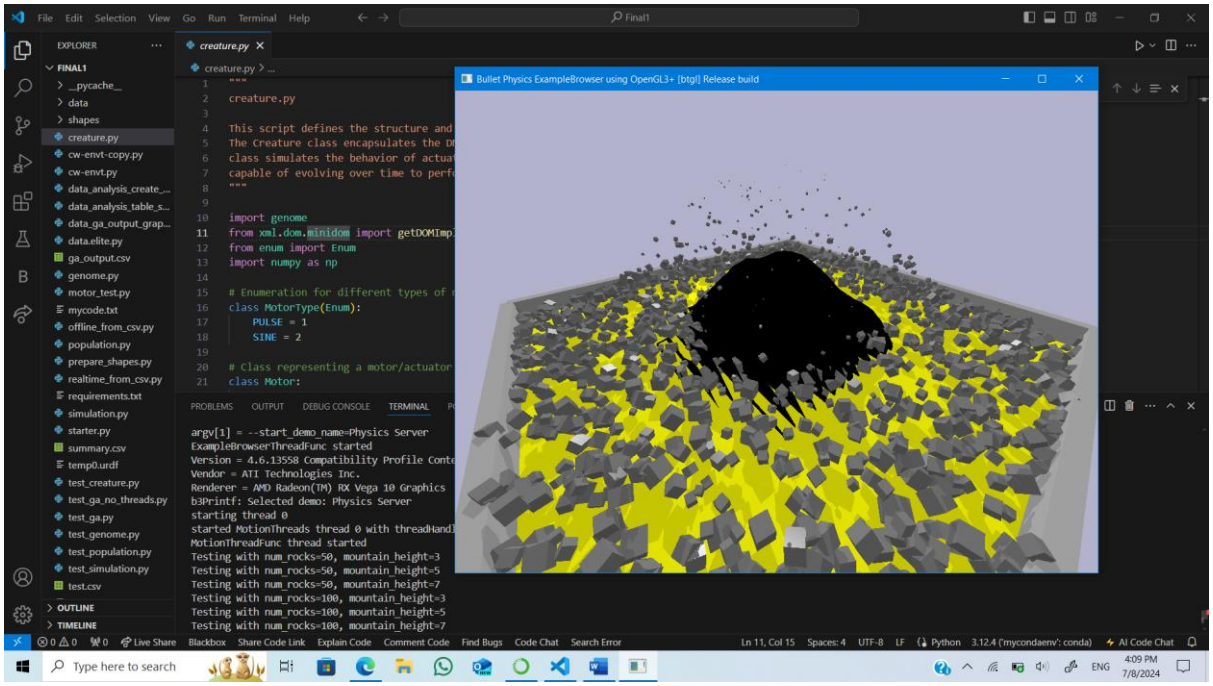
CM3020 Artificial Intelligence Mid-term Coursework

Part B

In this project, the objective is to adapt an existing evolutionary algorithm to enable creatures to evolve and accomplish a new challenge: climbing a mountain. This task not only necessitates a shift in the creatures' goals but also their integration into a freshly designed environment tailored specifically for this purpose. This report outlines the implementation steps undertaken, the experiments conducted, and the findings related to optimizing the evolutionary process within this framework.

Sandbox Arena and Mountain Environment

The sandbox arena and mountain environment were adapted from pre-existing codebases to provide a realistic simulation setting. This environment incorporates diverse terrain and obstacles designed to simulate the challenges associated with mountain climbing, thereby encouraging the evolution of adaptive behaviors among the creatures. Within this arena, rocks of varying sizes are randomly positioned to enhance environmental diversity, with their heights adjusted according to a Gaussian distribution that defines the mountain's contours.



Overview of Genetic Algorithm

Creatures within the simulation are represented as genomes, where each genome encodes structural and behavioral attributes through sequences of genes. These genes determine critical aspects such as joint types, lengths, and control mechanisms, which are essential for the creatures to navigate and scale the

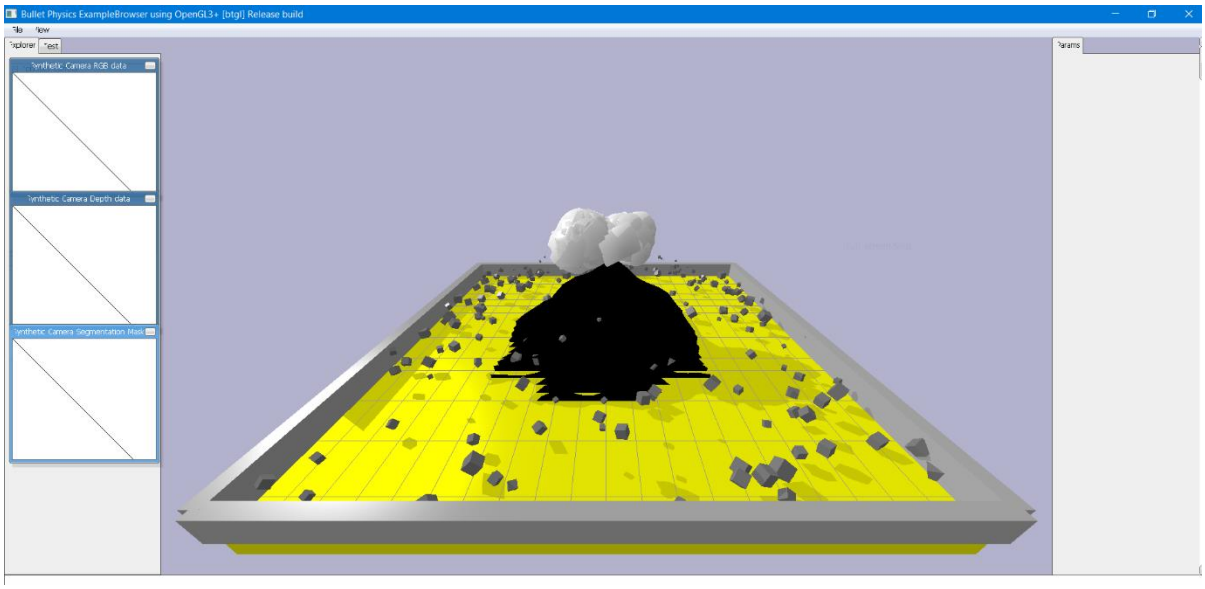
mountainous terrain. Specifications for different gene types, such as link shape, length, radius, mass, joint types, origins, and control parameters, are defined through the get_gene_spec method.

The genetic algorithm framework includes methods for genetic operations like crossover (crossover), point mutation (point_mutate), shrink mutation (shrink_mutate), and grow mutation (grow_mutate). These operations dynamically modify the creatures' genomes, influencing their physical structure and behavior within the PyBullet simulation environment. As genomes evolve across generations through mutation and crossover, creatures exhibit varying movement patterns and interaction capabilities, facilitating adaptive responses to their environment.

Fitness Function and Creature Behaviour in Simulation

Central to the evolutionary process is the fitness function, which plays a pivotal role in evaluating and guiding the evolution of creatures. In the simulation, creatures engage in various actions and behaviors, including movement and motor control using defined motors for each link. These motors, characterized by different types (such as pulse or sine wave) and control parameters (such as amplitude and frequency), dictate how the creatures interact with their environment.

Position tracking is integral to the simulation, enabling continuous updates of the creature's position as it maneuvers through the environment. These positional data points are leveraged to calculate metrics such as total distance traveled and vertical distance climbed, which are essential for fitness evaluation.



Fitness Evaluation

Fitness evaluation employs a fitness-proportional selection technique, commonly known as roulette wheel selection. This method probabilistically selects creatures as parents for the next generation based

on their fitness levels, thereby favoring those with higher performance. To ensure the preservation of successful traits, the approach incorporates elitism by reserving a segment of the top-performing creatures from each generation. This strategy promotes a gradual evolution towards more adept and higher fitness solutions over successive generations.

Fitness metrics evaluated include total vertical distance traveled, horizontal distance covered, and efficiency in movement. These metrics collectively contribute to the fitness score assigned to each creature, guiding the genetic algorithm's selection process. Creatures that exhibit superior vertical climbing ability and efficient navigation across the terrain are assigned higher fitness scores, thereby influencing their likelihood of passing on their genetic material through crossover and mutation.

Landscapes Generated

Beyond individual shapes, the script also facilitates the creation of entire landscapes. The generate_gaussian_pyramid4(filename, size, resolution, sigma, height, noise_scale, noise_factor) function exemplifies this capability by generating pyramid-shaped landscapes. It utilizes a combination of Gaussian distribution for the top surface and Perlin noise for surface variation, resulting in a structured terrain with triangular faces that form the pyramid shape.

Similarly, the generate_noisy_landscape(filename, size, resolution, sigma, height, noise_scale, noise_factor) function constructs noisy landscapes using Perlin noise. This method introduces realistic variations across the landscape surface, ensuring natural aesthetics through triangulated faces that define the landscape's shape.

Moreover, the script provides a means to combine different shapes into integrated landscapes. The combine_shapes(filename, pyramid_size, terrain_size, resolution) function merges a central pyramid shape with surrounding terrain. By connecting vertices with triangular faces, this function creates cohesive landscapes that blend geometric shapes with natural terrain features.

The Python script prepare_shapes.py effectively generates various 3D shapes and landscapes using mathematical functions such as Gaussian distribution and Perlin noise. These shapes are exported to OBJ files, making them suitable for visualization and further use in 3D modeling and simulation projects. Each function within the script serves a specific purpose, from creating basic geometric shapes like pyramids to generating complex terrains with realistic features like rocky mountains and varied landscapes.

Findings and Data Analysis

Please note that all the files below are my original code. They facilitate data analysis, including the creation of summary CSVs, tables, graphs, and more:

- **data_analysis_create_summary_csv.py**
- **data_analysis_table_summary.py**
- **data_ga_output_graph.py**
- **data_elite.py**

The cw-envt_copy.py script includes custom terrain and seamlessly integrates the genetic algorithm into the simulation environment.

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

The purpose of this study is to investigate how varying mutation rates and population sizes affect the performance of a genetic algorithm (GA) in evolving populations of simulated creatures. The study explores three types of mutations: point mutation, shrink mutation, and grow mutation, with rates ranging from 0.01 to 0.25.

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
Shrink Mutate	0.01 – 0.25	3	10	4	100

In the population size experiment, evaluated the average fitness (distance travelled) 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

Impact of Mutation Rates:

- **Point Mutation:** Generally showed gradual improvements in fitness over generations. Higher mutation rates (0.2 to 0.25) resulted in more significant variations in fitness.
- **Shrink Mutation:** Produced mixed results, with some instances showing improved fitness stability at lower rates (0.01 to 0.05).
- **Grow Mutation:** Demonstrated increased diversity in fitness outcomes, with higher rates (0.2 to 0.25) leading to more pronounced changes in fitness metrics.

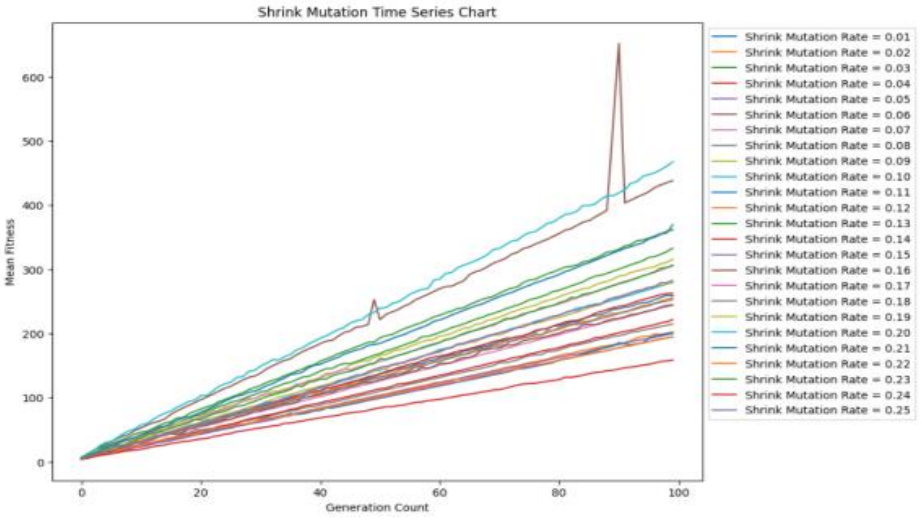
Impact of Population Size:

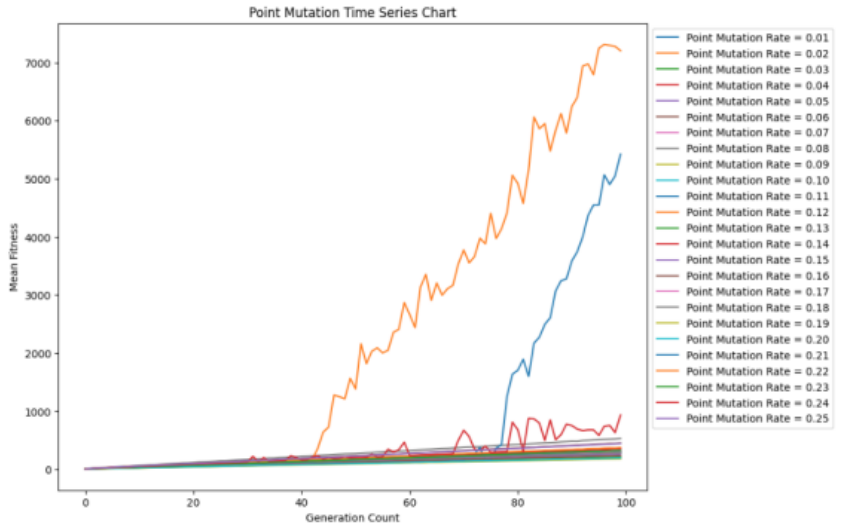
- Generally, larger population sizes (6 to 10 individuals) showed more stable and faster improvements in fitness compared to smaller sizes (2 to 4 individuals).
- Consistent mutation rates (0.25) across different population sizes ensured comparable evolution rates, emphasizing the role of population diversity in optimizing fitness outcomes.

The below graphs and tables illustrate the influence of mutation rates and population sizes on mountain climbing performance. Visual representations depict trends in fitness improvement across generations under different experimental conditions, providing insights into optimal GA configurations for enhancing evolutionary dynamics.

Evaluation Results

Here, the following plots were generated using Jupiter Notebook.





In conclusion, the project effectively integrates an evolutionary algorithm within a simulated mountain climbing scenario, demonstrating its capability to evolve adaptive creatures through genetic operations and fitness evaluation. The adaptation of the sandbox arena and mountain environment provides a realistic simulation platform for testing evolutionary strategies. The findings from experiments on mutation rates and population sizes highlight key parameters influencing the genetic algorithm's evolutionary dynamics. This iterative process of refinement contributes to the development of robust creatures capable of navigating complex terrains, thus showcasing the efficacy and potential of genetic algorithms in simulated environments.