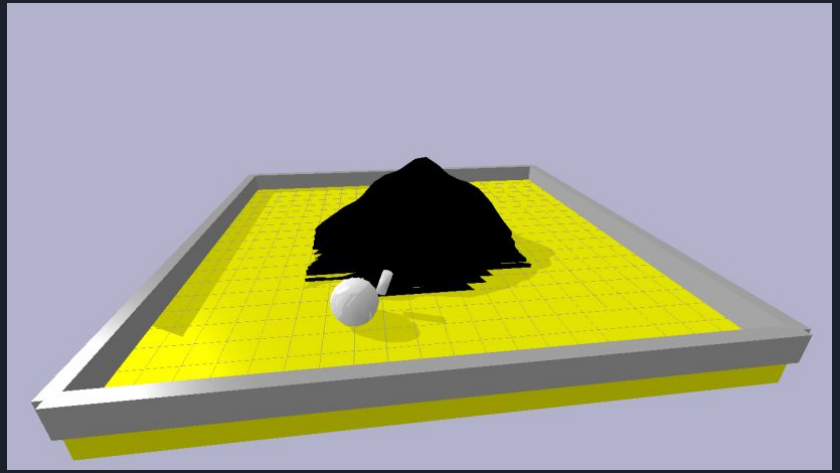
CM3020
AI coursework

Project Overview

- Integrate a genetic algorithm into a new environment where creatures had to climb a mountain

- Use PyBullet for the simulation environment



Genetic Algorithm Integration

Mathematical Formula for Fitness Function

The fitness function can be represented mathematically as:

$$Fitness = -\frac{min}{t\epsilon[0,T]}\left(\sqrt{(x_t - x_{peak})^2 + (y_t - y_{peak})^2 + (z_t - z_{peak})^2}\right)$$

where:

- (x_t, y_t, z_t) is the position of the creature at time t.
- $(x_{peak}, y_{peak}, z_{peak})$ is the position of the mountain peak.
- T is the total duration of the simulation.

Mathematical Formula for fitness function

```python
def run_simulation(agent, creature_instance, total_time=5, wait_time=1.0/240):
    urdf_path = 'test.urdf'
    with open(urdf_path, 'w') as f:
        f.write(creature_instance.to_xml())

    initial_position = [0, -8, 2]
    rob1 = p.loadURDF(urdf_path, initial_position)

    creature_instance.start_position, _ = p.getBasePositionAndOrientation(rob1)

    initial_state = convert_position_to_state(p.getBasePositionAndOrientation(rob1)[0])
    elapsed_time = 0
    step = 0
    jump_interval = 5

    min_distance_to_peak = float('inf')  # Initialize the minimum distance to peak

    while elapsed_time < total_time:
        action = agent.get_action(initial_state)

        if step % jump_interval == 0:
            motor_outputs = creature_instance.jump()
        elif step % 10 < 5:  # Rotate right arm for 5 steps
            motor_outputs = creature_instance.rotate_right_arm(step)
        else:  # Rotate left arm for the next 5 steps
            motor_outputs = creature_instance.rotate_left_arm(step)

        update_motors(rob1, step, motor_outputs, creature_instance.physicsClientId)

        p.stepSimulation()

        next_state = convert_position_to_state(p.getBasePositionAndOrientation(rob1)[0])
        reward = compute_reward(next_state)  # Define how reward is computed
        agent.update_q_table(initial_state, action, reward, next_state)

        initial_state = next_state

        current_position, _ = p.getBasePositionAndOrientation(rob1)
        distance_to_peak = np.linalg.norm(np.array(current_position) - np.array([0, 0, 10]))
        if distance_to_peak < min_distance_to_peak:
            min_distance_to_peak = distance_to_peak

        time.sleep(wait_time)
        elapsed_time += wait_time
        step += 1

    return -min_distance_to_peak
```

code snippet where fitness score is calculated in the `run_simulation` function

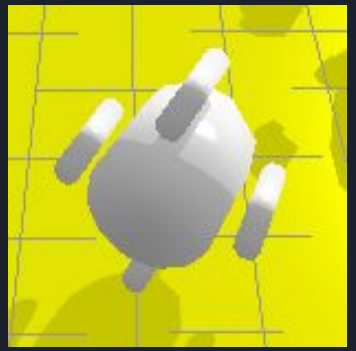
Genetic Algorithm Execution

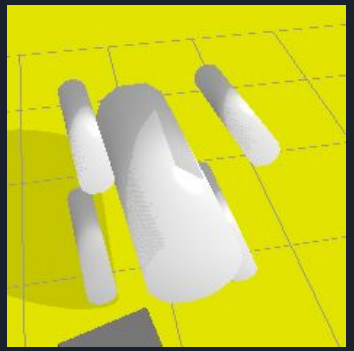
- The algorithm starts with generation 0 to 9
- Five creatures (creature 0 to creature 4) are simulated and their fitness scores are computed.
- Lower fitness scores represent better performance (closer to the mountain peak)
- The algorithm selects the best-performing creatures to breed
- Genetic mutations are applied to the selected genes

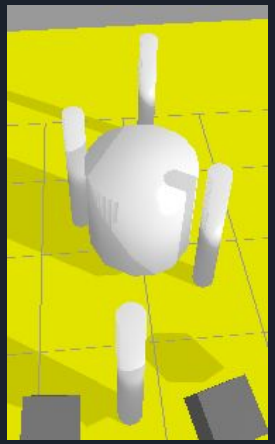
```
Starting Genetic Algorithm...
Generation 0
Running simulation for creature 0 in generation 0
Fitness for creature 0: -15.688571515191812
Running simulation for creature 1 in generation 0
Fitness for creature 1: -10.571794246198385
Running simulation for creature 2 in generation 0
Fitness for creature 2: -16.69004717163262
Running simulation for creature 3 in generation 0
Fitness for creature 3: -14.79648626352541
Running simulation for creature 4 in generation 0
Fitness for creature 4: -11.286785194891522
Selection and breeding
Point mutated gene index 13 from 0.14970907396985522 to 0.45552885952071787
Point mutated gene index 4 from 0.1214183231632261 to 0.6657930348545646
Point mutated gene index 14 from 0.7522579283508803 to 0.0742266374420929
Point mutated gene index 12 from 0.41085763089577976 to 0.5213720170568061
Point mutated gene index 1 from 0.9888834289322305 to 0.11300566572454385
Point mutated gene index 8 from 0.6582197655211208 to 0.04242016686441896
Point mutated gene index 13 from 0.4352886713449685 to 0.3151479365356864
Point mutated gene index 14 from 0.9220504864723641 to 0.8937589252130607
Point mutated gene index 7 from 0.722894288645818 to 0.03605651127792031
Point mutated gene index 9 from 0.4010068086935683 to 0.0982146035186996
Subtree mutation: replaced subtree at index 2 with new subtree of size 3
Point mutated gene index 0 from 0.4072414414416766 to 0.28133466008928243
Subtree mutation: replaced subtree at index 8 with new subtree of size 1
Point mutated gene index 2 from 0.8603102270606567 to 0.6924510847198733
Point mutated gene index 5 from 0.8109872656064242 to 0.09891952688698769
Point mutated gene index 14 from 0.6603888952688473 to 0.7191979000752128
Point mutated gene index 3 from 0.2971905642537862 to 0.7945512293612591
Subtree mutation: replaced subtree at index 4 with new subtree of size 3
Point mutated gene index 6 from 0.015899002560897713 to 0.8799933690863107
Point mutated gene index 1 from 0.24154200822935723 to 0.578985342557612
Point mutated gene index 1 from 0.6757490164868065 to 0.19592962700293992
Point mutated gene index 15 from 0.41825070949649523 to 0.10842747468727376
Point mutated gene index 13 from 0.7925305502768347 to 0.34352705032696595
Point mutated gene index 11 from 0.8304567258928722 to 0.7874174745411924
Subtree mutation: replaced subtree at index 0 with new subtree of size 2
Best fitness in generation 0: -10.571794246198385
```

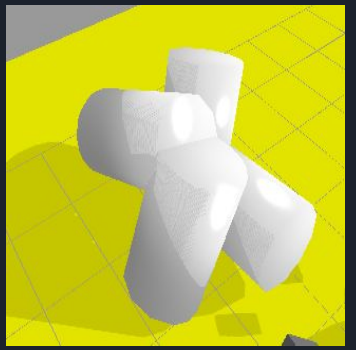
Creature Evolution

Genetic mutations introduce variability, aiding the evolution of more capable creatures over generations.









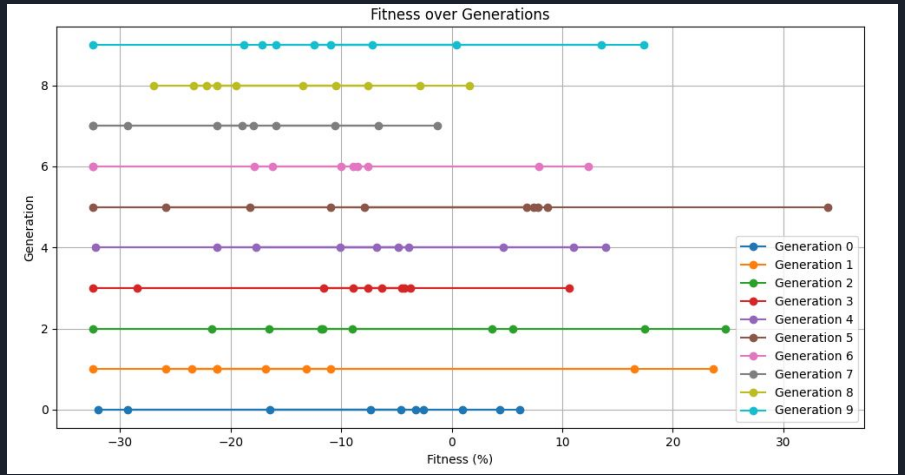
Q-Learning Integration

```python
import numpy as np
import random

class QLearningAgent:
    def __init__(self, action_space, state_space, alpha=0.1, gamma=0.9, epsilon=0.1):
        self.action_space = action_space
        self.state_space = state_space
        self.alpha = alpha  # Learning rate
        self.gamma = gamma  # Discount factor
        self.epsilon = epsilon  # Exploration rate
        self.q_table = np.zeros(state_space + (action_space,))

    def get_action(self, state):
        if random.uniform(0, 1) < self.epsilon:
            return random.randint(0, self.action_space - 1)  # Explore
        else:
            return np.argmax(self.q_table[state])  # Exploit

    def update_q_table(self, state, action, reward, next_state):
        best_next_action = np.argmax(self.q_table[next_state])
        td_target = reward + self.gamma * self.q_table[next_state][best_next_action]
        td_error = td_target - self.q_table[state][action]
        self.q_table[state][action] += self.alpha * td_error
```

- Q-Learning is a reinforcement learning algorithm used to find the optimal action-selection policy.
- Alpha (α): Learning rate, controls how much new information overrides the old
- Gamma (γ): Discount factor, determines the importance of future rewards.
- Epsilon (ϵ): Exploration rate, probability of choosing a random action.
- Enhances the creature's ability to learn optimal movement strategies and improves performance by learning from interactions with the environment.

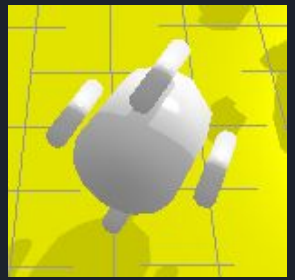
Summary of Fitness over Generations

- The graph illustrates the improvement in fitness scores of creatures across ten generations.
- Initially, the fitness scores are widely spread and predominantly negative, indicating poor performance.
- The scores become more positive and clustered as generations progress.



Inspiration for Creature Design

The design of the creatures was inspired by off-road vehicles tires like Jeeps and Outbacks, which are known for their ability to navigate rocky terrains.



Thank you!