Team Project Summary

Team Members

- Shahzaib Waseem
- Kamran Ali

Task Division : Task 1 was performed by Shahzaib Waseem, Task 2 was performed by Kamran Ali)

GitHub link : https://github.com/kamrankhowaja/Evolutionary_Robotics

Task 1: Implement Braitenberg's Vehicles

This task involves setting up the environment for the robot and light interactions, adding sensors and motor control for the robot's behavior.

- Sub-Task 1.1: Implement Torus Space
    - Create a 2D torus space where objects re-enter from the opposite edge.
- Sub-Task 1.2: Define Vehicle Properties
    - Model the robot with position, heading, and speed.
- Sub-Task 1.3: Light Intensity Field
    - Implement a light source with intensity decreasing linearly with distance.
- Sub-Task 1.4: Update Robot's Movement
    - Update robot's position and heading each time step, with limits on speed and turning.
- Sub-Task 1.5: Set Up Light Sensors
    - Create a 2D torus space where objects re-enter from the opposite edge.
- Sub-Task 1.6: Calculate Sensor Intensity
    - Model the robot with position, heading, and speed.
- Sub-Task 1.7: Implement Differential Drive
    - Implement a light source with intensity decreasing linearly with distance.
- Sub-Task 1.8: Implement Aggression and Fear
    - Update robot's position and heading each time step, with limits on speed and turning.
- Sub-Task 1.9: Update Heading
    - Change robot's heading based on wheel velocity difference.

Task 2: Proximity sensors and control by rules

This task is an extension to task 1.

- Sub-Task 1.1: Implement Bounded Space
    - Modify the space to be bounded, creating walls around the edges of the square.
    - Add walls within the square as obstacles.
- Sub-Task 1.2: Implement Proximity Sensors

    - Replace the light sensors with three proximity sensors:
        - Front-left sensor: Pointing to the front-left of the robot.
        - Front-right sensor: Pointing to the front-right of the robot.
        - Front sensor: Pointing straight ahead.
- Sub-Task 1.3: Define Sensor Properties and Distance Calculation
    - Set the sensor range to be 15% of the square's length.
    - Use a ray tracing to determine the distance to the nearest obstacle for each sensor.
    - If the sensor detects an obstacle within its range, so sensor updates the current length of ray
    - If no obstacle is detected within range, the state remains forward.
- Sub-Task 1.4: Handle Obstacle Detection
    - Ensure that the proximity sensors return values based on the distance to walls or obstacles.
- Sub-Task 1.5: Define Navigation Control Rules
    - Implement simple control logic using FSM rules
        - If an obstacle is detected in the left sensor, turn the robot to the right.
        - If an obstacle is detected in the right sensor, turn the robot to the left.
        - If an obstacle is detected straight ahead, turn around or adjust the robot's heading to avoid the collision.
- Sub-Task 1.6: Update Robot’s Heading
    - Change the robot’s heading based on the proximity sensor readings and defined rules:
        - Turn left or right based on obstacles detected by the left or right sensors.
        - If the front sensor detects an obstacle, it rotates randomly.

Platform Used

- The platform used was p5.js, an online open-source platform for JavaScript coding.