EAST WEST UNIVERSITY

Assignment– 01

Submitted By -

Name : Tamanna Sultana Tinne

ID : 2020-2-60-057

Course: CSE-366(Artificial Intelligence)

Section: 03

Submitted To -

Dr. Mohammad Rifat Ahmmad Rashid

Assistant Professor
Department of Computer Science & Engineering

Title: Enhanced Dynamic Robot Movement Simulation.

Introduction:

Global robot simulation involves utilizing computer-generated models to imitate the actions and features of tangible robots. This technological approach facilitates the examination, validation, and enhancement of robotic systems within a digital setting prior to their implementation in the actual world. The importance of robot simulation is apparent across diverse industries and applications.

In this Assignment the focus is to use a simple robocar simulation using Python object-oriented programming (OOP) concept where I tried to create a simulation of a car detecting object and avoided the obstacles to and move accordingly to reach its destination.

Objective:

The main objective of this project to focus on the oop concept of python.
And by using different class object our goal is to create a robocar simulation using Enhanced algorithms where it should detect the obstacles using its sensors and by avoiding it. The robocar must work
based on how I trained it using algorithms and move accordingly by avoiding the objects or, any kind of obstacles.

Methodology:

Since the main focus was just using python object-oriented class programming I used -

Tools:

1. Python version 3.12
2. Jupyter notebook
3. Visual Studio (Python environment for test run)

Libraries:

1. Numpy
2. Matplotlib
3. Pygame
4. JsAnimation
5. math

Descriptive Analysis:

In python environment we need to use different libraries for 2D arrays, graphical representation, and calculation So, I used Numpy, Pygame and math libraries. Besides for showing the cars movement graph I used the matplotlib to show In which the direction the car will move if the car meets any kind of obstacles and for reaching its final destination the path will show accordingly.

I used different kind of variables for showing height and width of the game window car movement path with a dotted visual effect and for rotation, car speed and censoring the obstacles as well as for battery usage according to the car's distances accordingly.

I used two different classes like Car_class and Obstacle_class to identify and run the loop for car movement and obstacle detection. Besides, I used Euclidean distance algorithm for car movement and rotation. For, rotating the car I set left and right arrow key for choosing its path as the per users wantwit. If the car runs out its battery usage limit it will stop, and the simulation will end. But if the car reaches its destination the simulation window will show a message "Bravo!".

Implementation:

The Implementation of the code snippets looks like -

```
import pygame
import math
import matplotlib.pyplot as pl
```

Imported the necessary libraries.

```
# Initialize Pygame
pygame.init()
```

Initializing the pygame library.

```
# Constants
WIDTH, HEIGHT = 1200, 800
FPS = 60
CELL_SIZE = 40
SPEED = 3
ROTATION_SPEED = 5
SENSOR_LENGTH = 80  # Increased sensor length
```

These constants define the width and height of the game window, frames per second, cell size, speed of the car, rotation speed, and the sensor length used for obstacle detection.

```python
# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
```

These constants define RGB color values for white, black, green, and red.

```python
BATTERY_WIDTH = 20
BATTERY_HEIGHT = 150
BATTERY_POSITION = (WIDTH - 50, HEIGHT // 2 - BATTERY_HEIGHT // 2)
```

Constants related to the car's energy indicator, specifying width, height, and position on the screen.

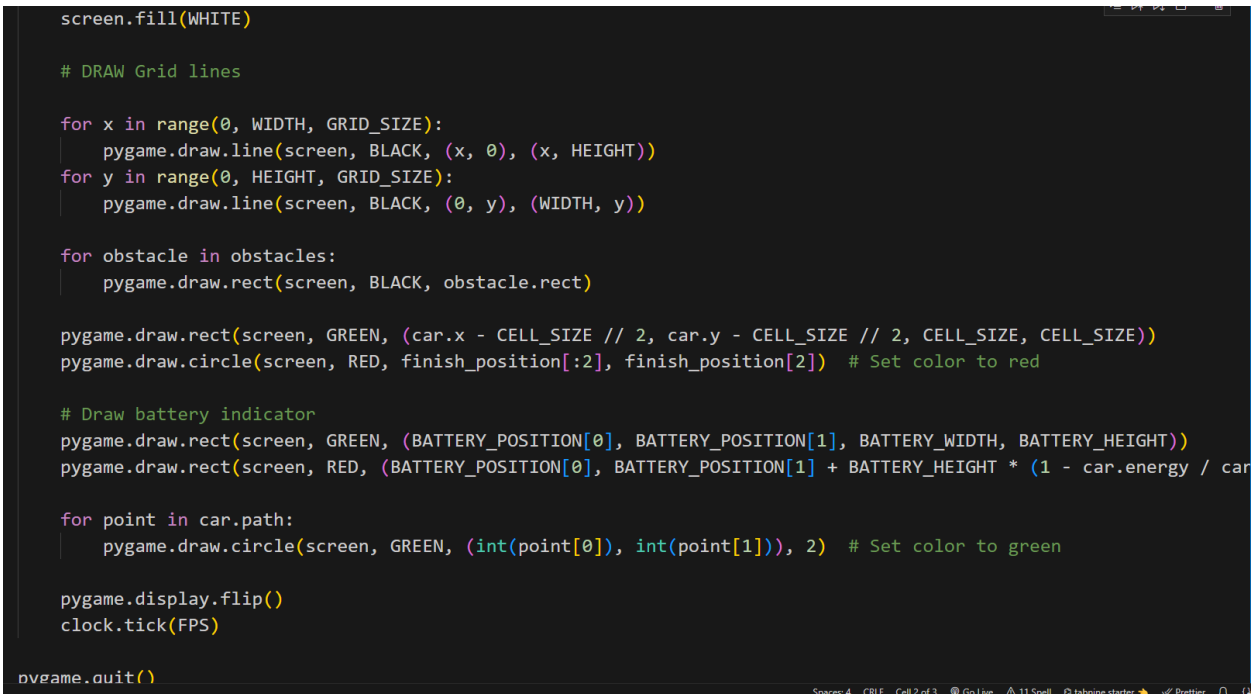
```python
class Car:
    def __init__(self, x, y, angle=0, max_energy=1000, energy_consumption_rate=1, recharge_rate=2):
        self.x = x
        self.y = y
        self.angle = angle
        self.path = []
        self.stuck_counter = 0
        self.max_energy = max_energy
        self.energy = max_energy  # Initial energy level
        self.energy_consumption_rate = energy_consumption_rate
        self.recharge_rate = recharge_rate

    def move(self):
        if self.energy > 0:
            self.x += SPEED * math.cos(math.radians(self.angle))
            self.y += SPEED * math.sin(math.radians(self.angle))
            self.path.append((self.x, self.y))
            self.energy -= self.energy_consumption_rate
        else:
            print("Out of energy! Charging...")
            self.energy += self.recharge_rate
```

```
    def rotate_left(self):
        self.angle += ROTATION_SPEED

    def rotate_right(self):
        self.angle -= ROTATION_SPEED

    def check_sensor(self, obstacles, angle_offset):
        sensor_x = self.x + SENSOR_LENGTH * math.cos(math.radians(self.angle + angle_offset))
        sensor_y = self.y + SENSOR_LENGTH * math.sin(math.radians(self.angle + angle_offset))

        for obstacle in obstacles:
            if obstacle.rect.collidepoint(sensor_x, sensor_y):
                return True  # Obstacle detected by the sensor

        return False  # No obstacle detected by the sensor
```

Defines a class Car with methods to move, rotate, and check sensors. The car has attributes like position, angle, energy level, and various parameters related to energy management.

```
class Obstacle:
    def __init__(self, x, y, width, height):
        self.rect = pygame.Rect(x, y, width, height)


screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Robo-Car with Obstacle Detection")

clock = pygame.time.Clock()


car = Car(50, 50)
finish_position = (1100, 650, 30)


obstacles = [
    Obstacle(0, 0, WIDTH, 20),
    Obstacle(0, 0, 20, HEIGHT),
    Obstacle(WIDTH - 20, 0, 20, HEIGHT),
    Obstacle(0, HEIGHT - 20, WIDTH, 20),
    Obstacle(200, 200, 20, 400),
    Obstacle(200, 200, 420, 20),  # Adjusted position and size for L-shape
    Obstacle(600, 400, 20, 400),
    Obstacle(900, -100, 20, 400),
    Obstacle(600, 400, 150, 20),
```

Defines a class `Obstacle` with a single attribute `rect` representing the rectangular shape of the obstacle.



```
    screen.fill(WHITE)

    # DRAW Grid lines

    for x in range(0, WIDTH, GRID_SIZE):
        pygame.draw.line(screen, BLACK, (x, 0), (x, HEIGHT))
    for y in range(0, HEIGHT, GRID_SIZE):
        pygame.draw.line(screen, BLACK, (0, y), (WIDTH, y))

    for obstacle in obstacles:
        pygame.draw.rect(screen, BLACK, obstacle.rect)

    pygame.draw.rect(screen, GREEN, (car.x - CELL_SIZE // 2, car.y - CELL_SIZE // 2, CELL_SIZE, CELL_SIZE))
    pygame.draw.circle(screen, RED, finish_position[:2], finish_position[2])  # Set color to red

    # Draw battery indicator
    pygame.draw.rect(screen, GREEN, (BATTERY_POSITION[0], BATTERY_POSITION[1], BATTERY_WIDTH, BATTERY_HEIGHT))
    pygame.draw.rect(screen, RED, (BATTERY_POSITION[0], BATTERY_POSITION[1] + BATTERY_HEIGHT * (1 - car.energy / car

    for point in car.path:
        pygame.draw.circle(screen, GREEN, (int(point[0]), int(point[1])), 2)  # Set color to green

    pygame.display.flip()
    clock.tick(FPS)

pygame.quit()
```

For drawing grid lines and battery indicator.

```
# Main loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        car.rotate_left()
    if keys[pygame.K_RIGHT]:
        car.rotate_right()

    # Check sensors
    front_sensor = car.check_sensor(obstacles, 0)
    left_sensor = car.check_sensor(obstacles, 45)
    right_sensor = car.check_sensor(obstacles, -45)

    # Move the car
    car.move()
```

Main loop for car movement.

```
    if car.energy <= 0:
        print("Out of energy! Charging...")
        pygame.font.init()
        font = pygame.font.SysFont('Arial', 50)
        text = font.render('Out of Energy!', True, RED)  # Set color to red
        screen.blit(text, (WIDTH // 2 - 150, HEIGHT // 2 - 20))
        pygame.display.flip()
        pygame.time.wait(2000)  # Wait for 2 seconds
        car.energy += car.recharge_rate


    # Check if the car is close to the finish line
    distance_to_finish = math.hypot(car.x - finish_position[0], car.y - finish_position[1])
    if distance_to_finish < finish_position[2]:
        print("Bravo! You reached your destination.")
        pygame.font.init()
        font = pygame.font.SysFont('Arial', 120)
        text = font.render('Bravo!', True, RED)  # Set color to red
        screen.blit(text, (WIDTH // 2 - 70, HEIGHT // 2 - 20))
        pygame.display.flip()
        pygame.time.wait(2000)  # Wait for 2 seconds
        running = False
```

Conditions for checking the car's energy level and is it reach to the finish line or, not.

```
    if front_sensor or left_sensor or right_sensor:
        # If any sensor detects an obstacle, stop, change direction, and reduce speed
        SPEED = 1
        car.rotate_left()

        # Check if the car is stuck (rotating in place for a certain period)
        car.stuck_counter += 1
        if car.stuck_counter > 50:
            print("The car is stuck!")
            pygame.font.init()
            font = pygame.font.SysFont('Arial', 50)
            text = font.render('Stuck!', True, RED)  # Set color to red
            screen.blit(text, (WIDTH // 2 - 70, HEIGHT // 2 - 20))
            pygame.display.flip()
            pygame.time.wait(2000)  # Wait for 2 seconds
            running = False
```

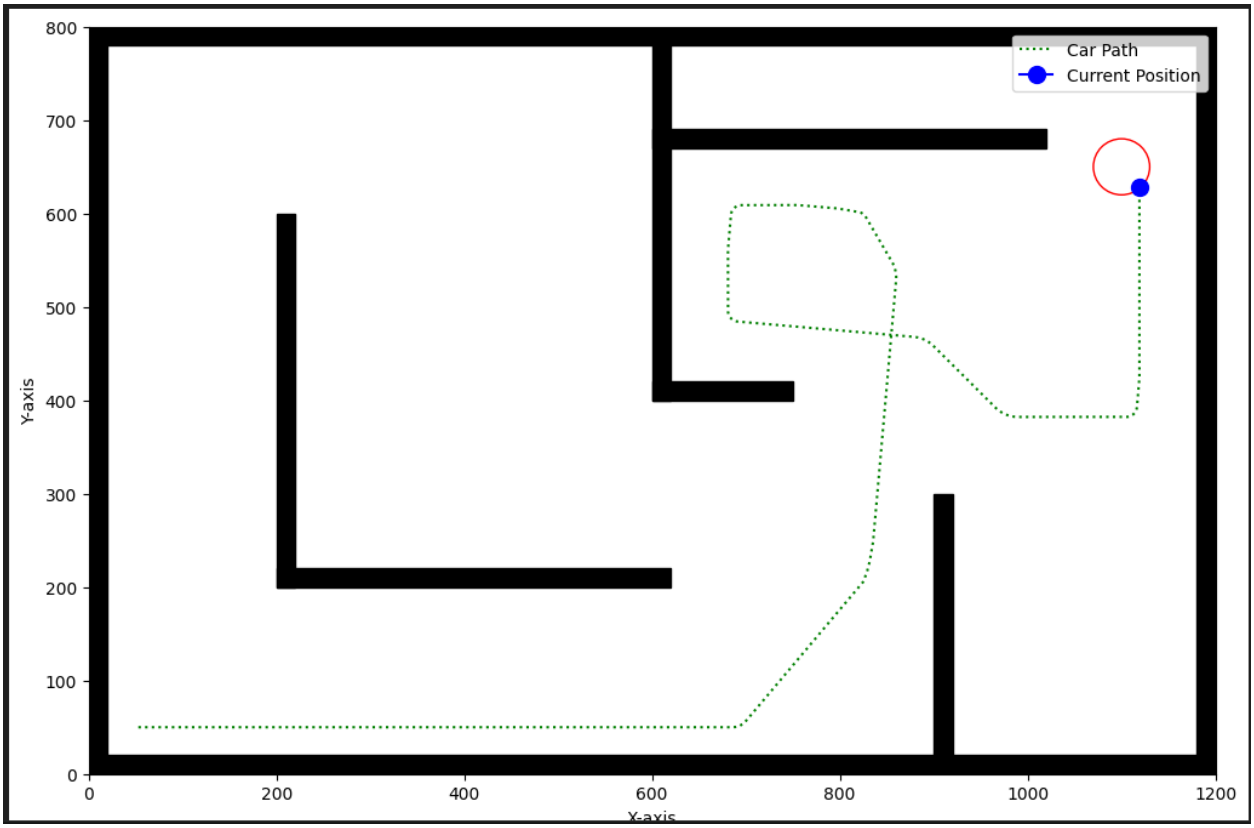
Adjustment of the sensor reading for detecting obstacles or, not.

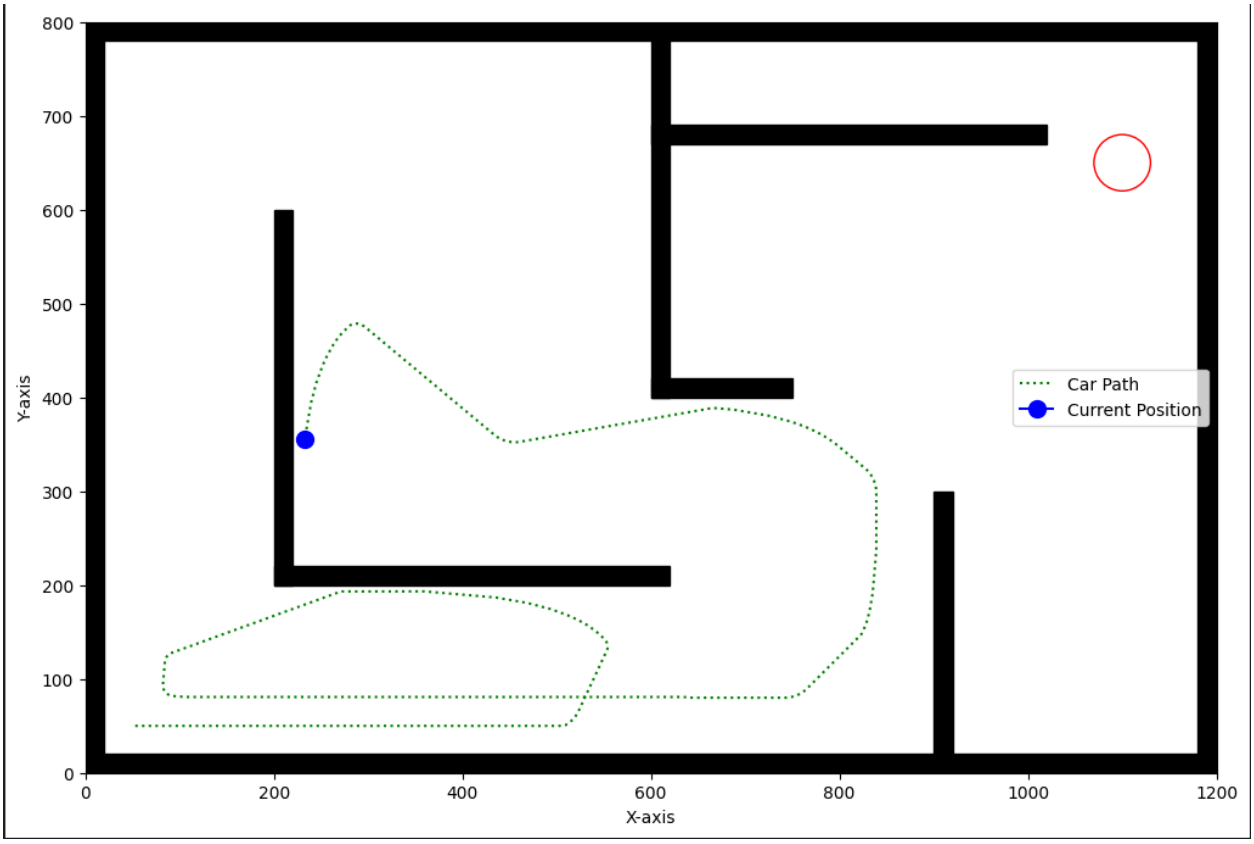
```
fig, ax = pl.subplots(figsize=(12, 8))

# Plot the obstacles
for obstacle in obstacles:
    rect = pl.Rectangle((obstacle.rect.x, obstacle.rect.y), obstacle.rect.width, obstacle.rect.height, color='black'
    ax.add_patch(rect)

# Plot the finish position
finish_circle = pl.Circle((finish_position[0], finish_position[1]), finish_position[2], color='red', fill=False)
ax.add_patch(finish_circle)

# Plot the car's path as a dotted line
path_x, path_y = zip(*car.path)
ax.plot(path_x, path_y, linestyle='dotted', color='green', label='Car Path')

# Plot the current position of the car
ax.plot(car.x, car.y, marker='o', markersize=10, color='blue', label='Current Position')

# Set axis limits and labels
ax.set_xlim(0, WIDTH)
ax.set_ylim(0, HEIGHT)
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
```

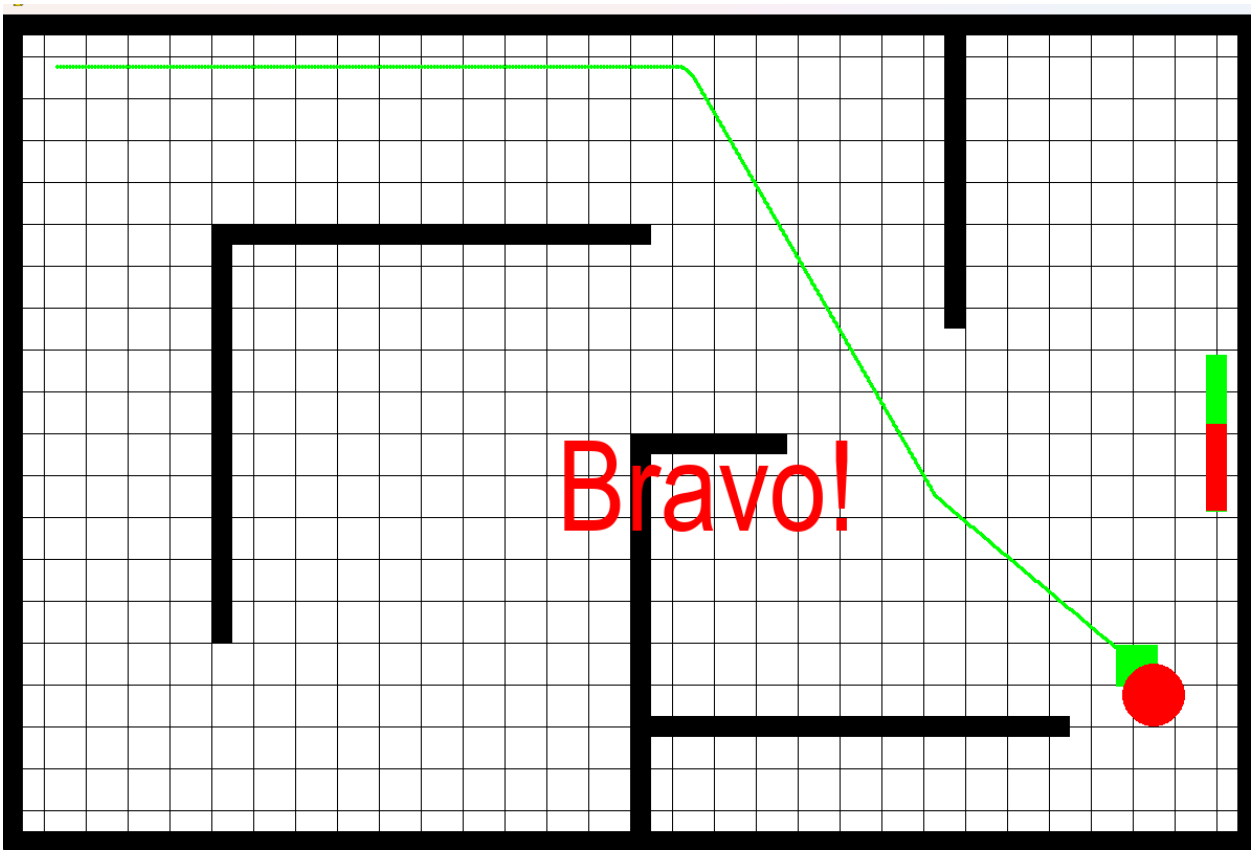
Finally showing the car's path through matplotlib plot show.

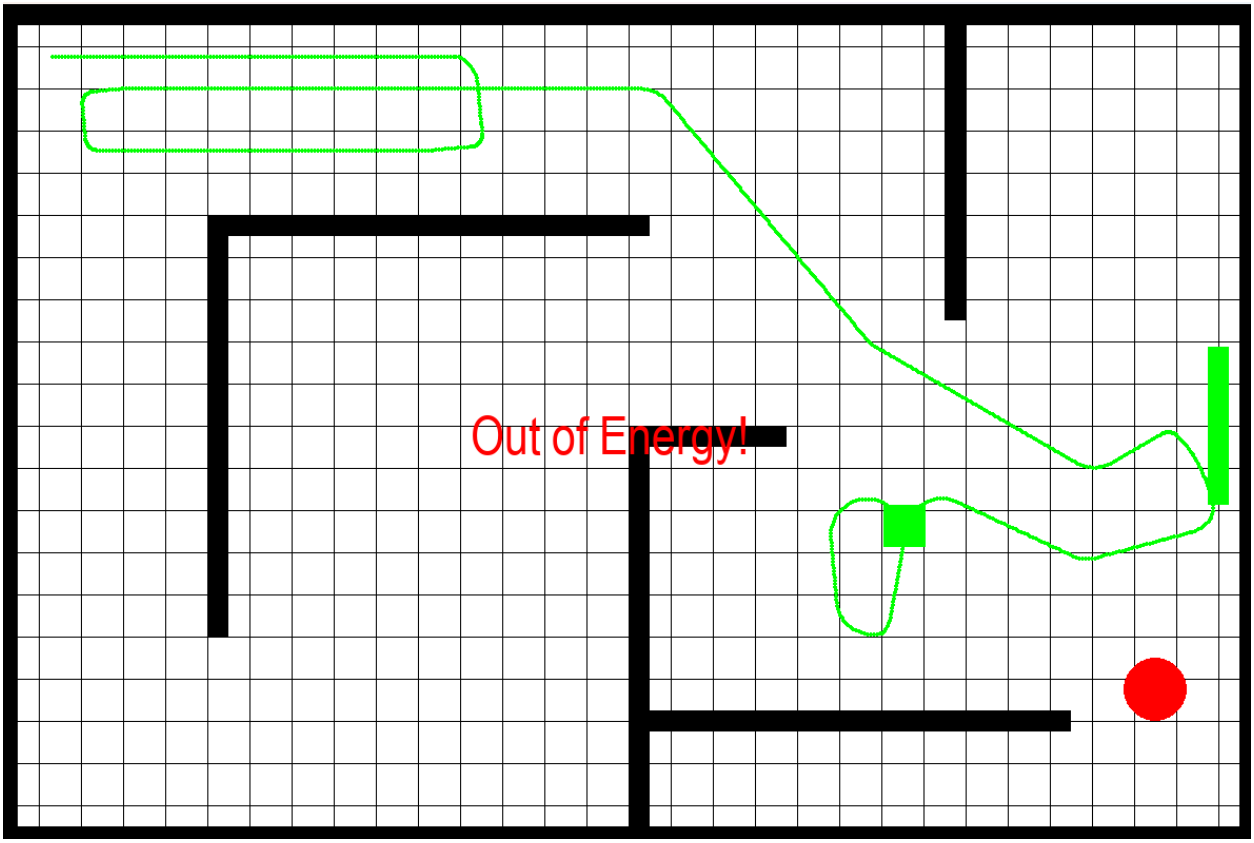
Results:

multiple results based on the car’s movement -




Car Path
Current Position
Y-axis
X-axis
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200


Bravo!


Out of Energy!

Challenges during Implementations:

1. One of the most challenging part of the code was setting up the sensor even still, The sensor logic in the `check_sensor` method may need further validation. It currently returns **True** if any obstacle is detected by any sensor, but it might be useful to differentiate between front, left, and right sensors.

2. The recharge time after running out of energy is hardcoded to `pygame.time.wait(2000)`. This value might need to be adjusted based on the desired simulation behavior. In near future I will fix it properly.

3. The simulation logic, sensor checks, and user input handling are all within the main loop. In near future I will try to be separating these concerns into distinct functions or methods for better code organization and also for the cost efficiency.

Conclusions:

In summary, robot simulation technology holds significant importance globally, serving as a transformative tool in various aspects of robotic system development, testing, and deployment. Its impact is evident in cost reduction, efficiency enhancement, risk mitigation, educational support, facilitation of global collaboration, ethical considerations, and fostering future innovations within the robotics domain.

And this assignment was just a simple simulation that will make us understand the challenges of robotics domain. I tried my best to create a virtual platform using the python OOP concepts to give a hunch how can we make a robot with training to sense the obstacle in its movement with a visual representation.

Links:

Github -
G.drive (simulation video) -