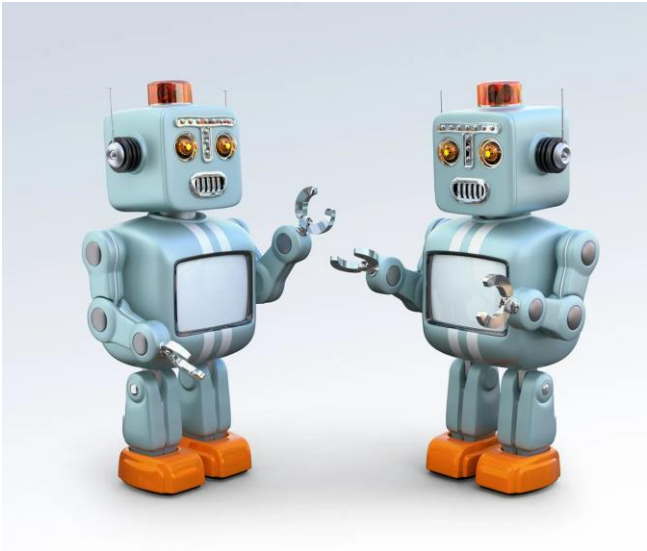


ARITIFICIAL INTELLIGENCE PROJECT

RIME 2023

ABSTRACT

Debugging and Addition of Features in Python Based Snake Game

AQIB HABIB MEMON

Reg: 450062

Contents

1 Task A

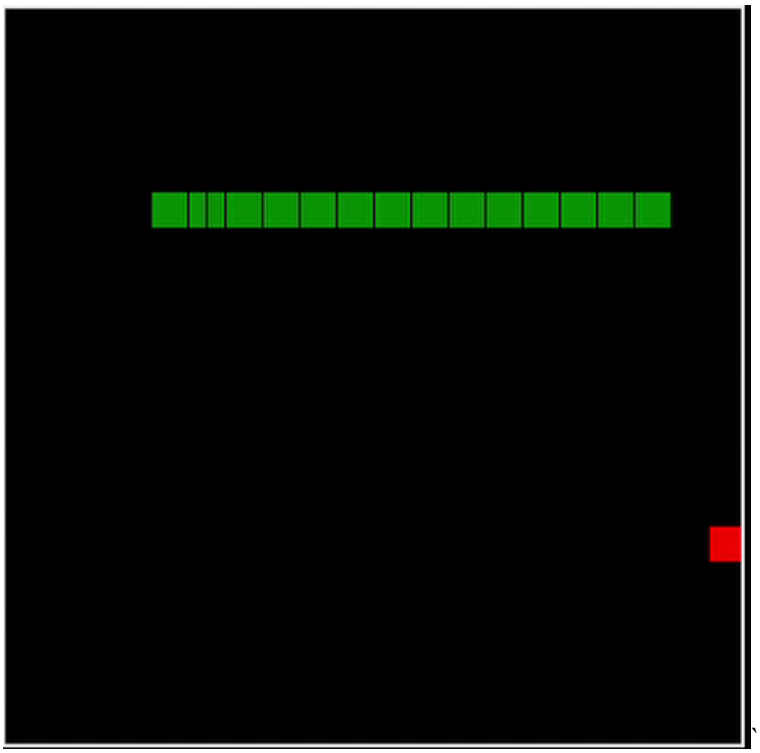


Figure 1 Tail of the Snake Not Synchronized

This bug is fixed by adding a few lines into the move_snake function.

```
def move_snake(self):
    head = self.snake[0]
    if self.direction == "Right":
        new_head = (head[0] + 20, head[1])
    elif self.direction == "Left":
        new_head = (head[0] - 20, head[1])
    elif self.direction == "Up":
        new_head = (head[0], head[1] - 20)
    elif self.direction == "Down":
        new_head = (head[0], head[1] + 20)
self.snake.insert(0, new_head)
```

Code Block 1 Current Code in the Game

```
def move_snake(self):
    head = self.snake[0]
    if self.direction == "Right":
        new_head = (head[0] + 20, head[1])
    elif self.direction == "Left":
        new_head = (head[0] - 20, head[1])
    elif self.direction == "Up":
```

```
            new_head = (head[0], head[1] - 20)
        elif self.direction == "Down":
            new_head = (head[0], head[1] + 20)
        self.snake.insert(0, new_head)
        self.food_position = self.canvas.coords(self.food)
                if new_head == self.food_position:
        # Handle collision with food, grow snake
            self.score += 1
            self.master.title(f"Snake Game - Score: {self.score}")
            self.food_position = self.create_food()  # Generate new food
        else:
            self.snake.pop()  # Remove the tail to maintain snake's length
```

Code Block 2 New Code

The highlighted code firstly acquires the coordinates of the food in the canvas and if the coordinates of the snake head is equal to food's then the score is added and length increases, otherwise in all conditions the after the movement the snake tail is popped out form snake list.

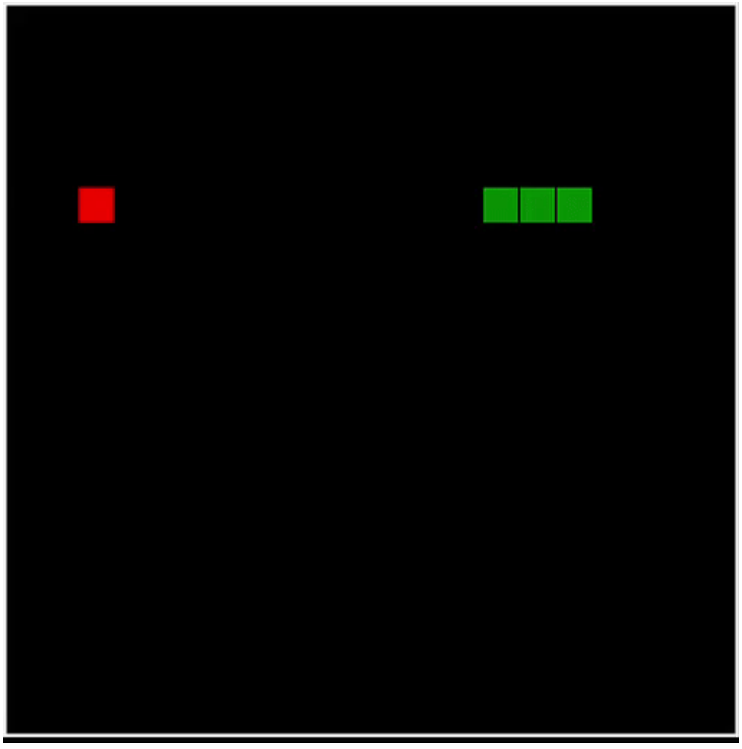


Figure 2 After Fixing Move_Snake Function

2 Task B

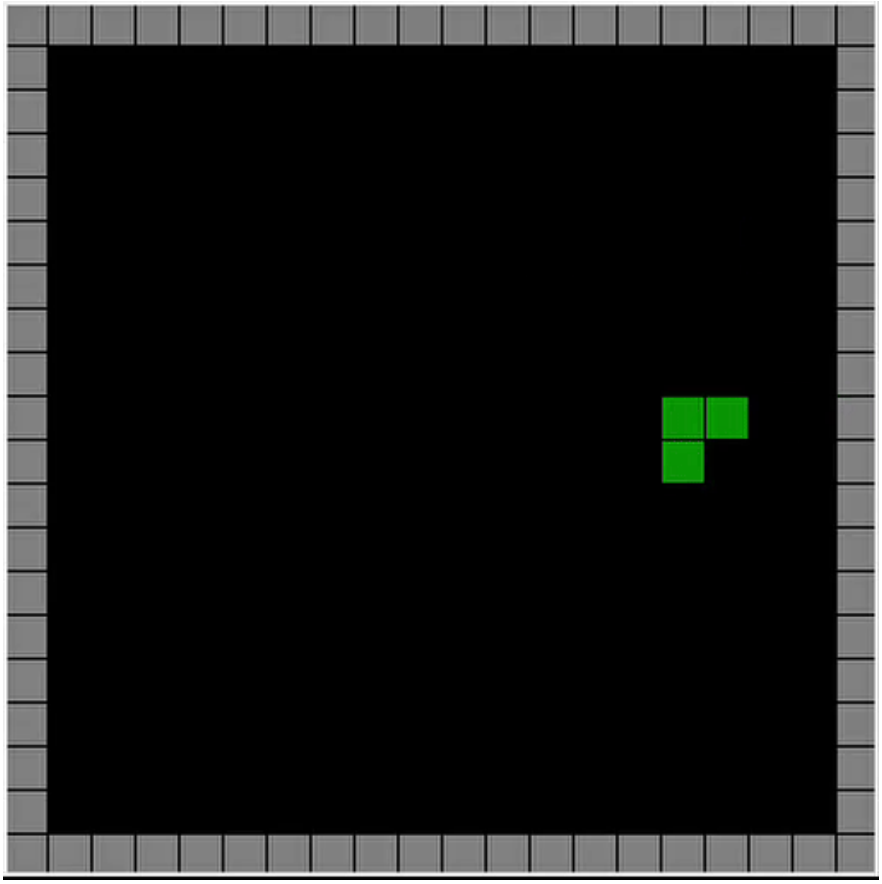
2.1 Game Boundary and Collision Detection

Currently, the snake game has not outer boundaries in the canvas. The outer boundaries were added into the canvas with grey squares.

For that a new function was added into the code as described below

```
def create_boundaries(self):
    # Create boundaries on the canvas
    for i in range(0, 400, 20):
        self.canvas.create_rectangle(0, i, 20, i + 20, fill="grey")  # Left boundary
        self.canvas.create_rectangle(i, 0, i + 20, 20, fill="grey")  # Top boundary
        self.canvas.create_rectangle(i, 380, i + 20, 400, fill="grey")  # Bottom boundary
        self.canvas.create_rectangle(380, i, 400, i + 20, fill="grey")  # Right boundary
```

Code Block 3 Boundary Function



Code Block 4 Boundary Editions

This function is called in **__init__(self, master)** function of the snake game.

But there are still no collision detections in the game with snake itself or with boundaries and due to addition of boundaries, sometimes the food is spawned inside the boundary.

To add the collisions detection with the boundaries, a new function is added to check the collision.

```
def check_boundary_collision(self, position):
    # Check collision with grey boundaries
    x, y = position
    return (
        x < 20 or x >= 380 or y < 20 or y >= 380
```

Code Block 5 Boundary Collision Detection Function

This function is called in update function of the code as following:

```
if (
        head[0] < 0
        or head[0] >= 400
        or head[1] < 0
        or head[1] >= 400
        or head in self.snake[1:]
        or self.check_boundary_collision(head)
    ):
        self.game_over()
```

Code Block 6 Boundary Collision Detection

The function “game_over()” is also a customized function which stops the tkinter canvas and reports the final score of the game on window.

```
def game_over(self):
    self.master.title(f"Snake Game - Game Over! Final Score: {len(self.snake)}")
    self.canvas.delete(tk.ALL)
```

Code Block 7 game_over function

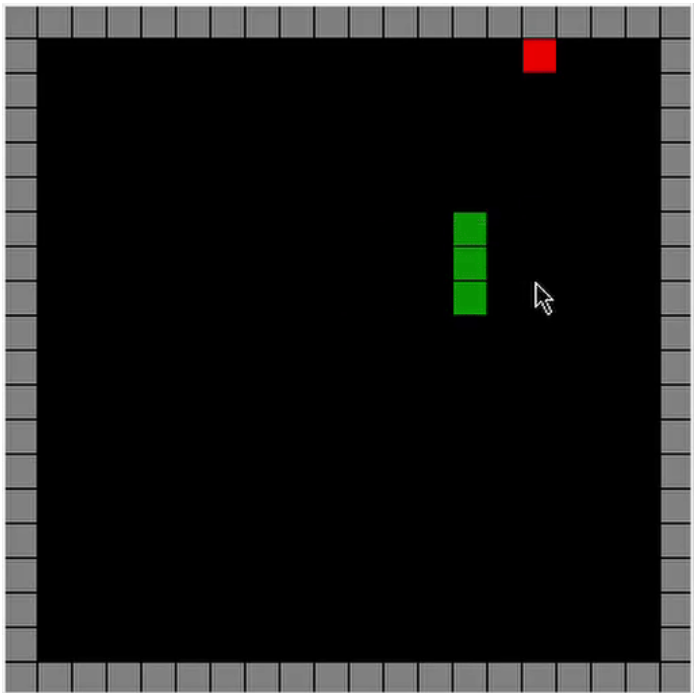


Figure 3 Boundary Collision

2.2 Obstacle Generation

To generate randomly placed obstacles in the game, a rand function is used and every time the update function is called, the probability is set to 3% that the random obstacle will be generated.

```
def create_obstacle(self):
    while True:
        x = random.randint(0, 19) * 20
        y = random.randint(0, 19) * 20
        overlapping = False
        if (
        x < 20 or x >= 380 or y < 20 or y >= 380
    ):
            overlapping = True
        if not overlapping:
            break
    obstacle = self.canvas.create_rectangle(x, y, x + 20, y + 20, fill="blue")
    return obstacle
```

Code Block 8 Random Obstacles Generator

This function utilizes the random number generator function to create obstacles in the canvas based on the generated x, y position. This function also checks whether the generated obstacles is on the boundary or not and returns false in that case and the obstacle is not generated.

The obstacles are generated as blue in the canvas.

```
    if random.random() < 0.03:  # Adjust probability to create obstacles
        obstacle = self.create_obstacle()
        self.obstacles.append(obstacle)
```

Code Block 9 Calling the Obstacle Generator Function

This function is called inside the update function.

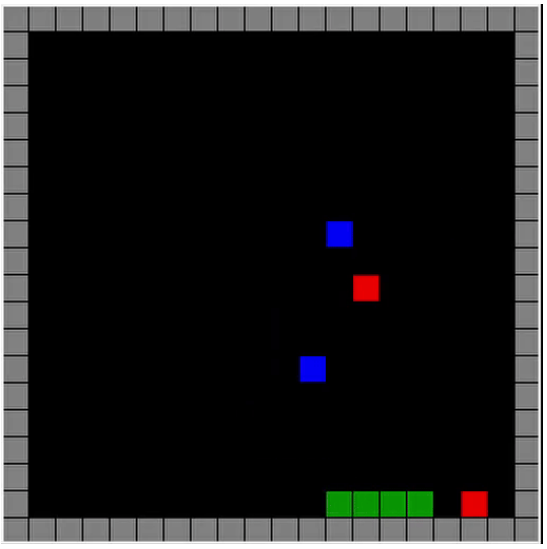


Figure 4 Obstacle Generation

2.3 Collision Detection with Obstacles

Like in the previous section, we need to add a collision detection function or logic with the obstacles.

```
# Check collision with obstacles
for obstacle in self.obstacles:
    obstacle_coords = self.canvas.coords(obstacle)
    if head[0] == obstacle_coords[0] and head[1] == obstacle_coords[1]:
        self.game_over()
```

Code Block 10 Collision Detection

These lines are coded in the update function to check the collision of snake with the generated obstacles.

3 Task C

A simple score board is generated at the top left corner of the canvas after the boundary which calculates the snake length and updates it whenever the snake eats food.

```
self.score_label = tk.Label(self.master, text="Snake Length: 3")  # Initialize score_label
self.score_label.pack()
self.score_display = self.canvas.create_text(25, 22.5, text="Snake Length: 3", fill="white", anchor="nw")
```

Code Block 11 Score Board

The above lines have been added in the **__init__(self, master)** function.

To update the scoreboard, the following code has also been added to the update function.

```
if len(self.snake) > 3:
    self.canvas.itemconfig(self.score_display, text=f"Snake Length: {len(self.snake)}")
```

Code Block 12 Updating Score Board

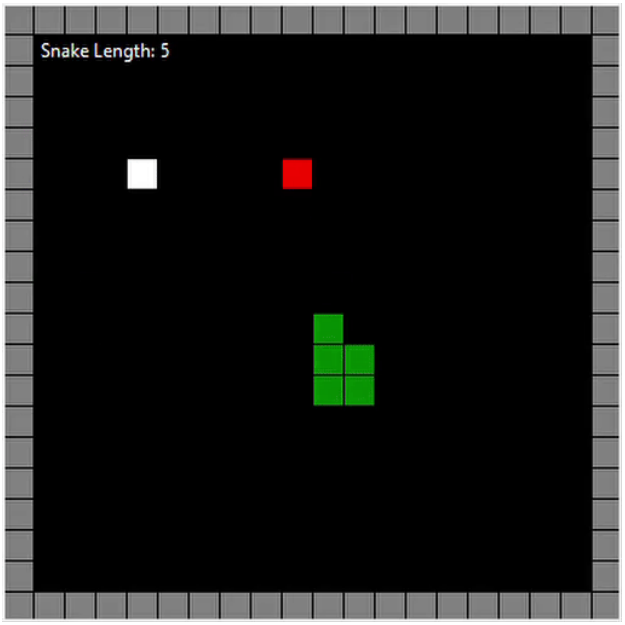


Figure 5 Score Board

4 Task D

In the last part, we have to add a competing snake that will compete with the player in real-time.

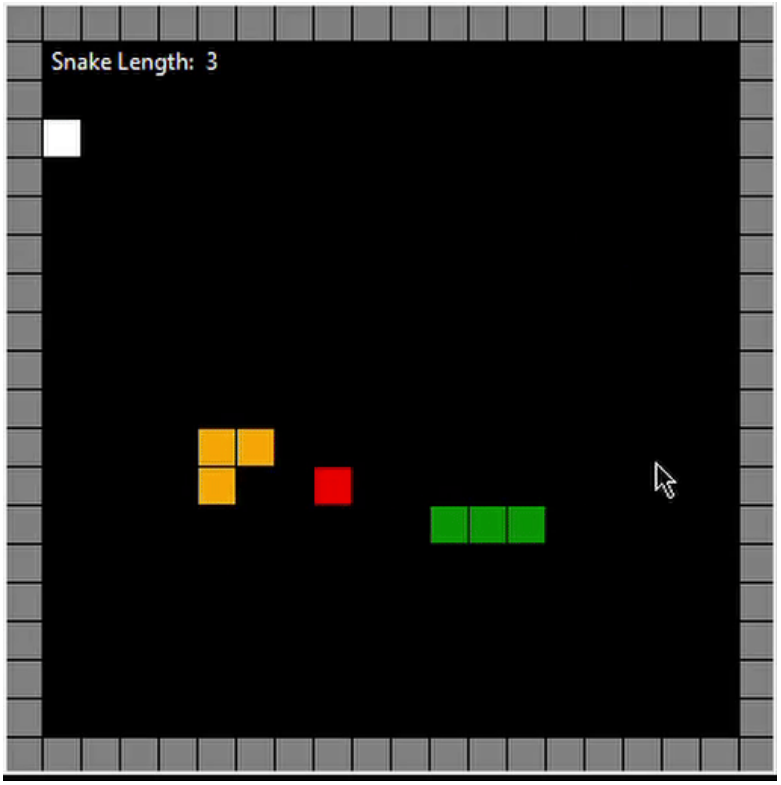


Figure 6 Randomly Moving Snake

For this task, A* search algorithm is to be implemented using Manhattan Heuristic