Snake Game Report

A Project by:
Syalbia Noor Rahmah (5025221067), Areta Athayayumna Arwaa (5025221068), Iffa Amalia Sabrina (5025221077)

Made to complete the Quiz 2 for class:
***EF234405: Design and Analysis Algorithms*,**

On behalf of:
Ir. M.M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil., IPM (Sr. Pro. Eng.)

Table of Contents

Language Used

Python Programming Language

Source Code

https://github.com/aleahfaa/EF234405-Design-and-Analysis-Algorithms-Quiz-2

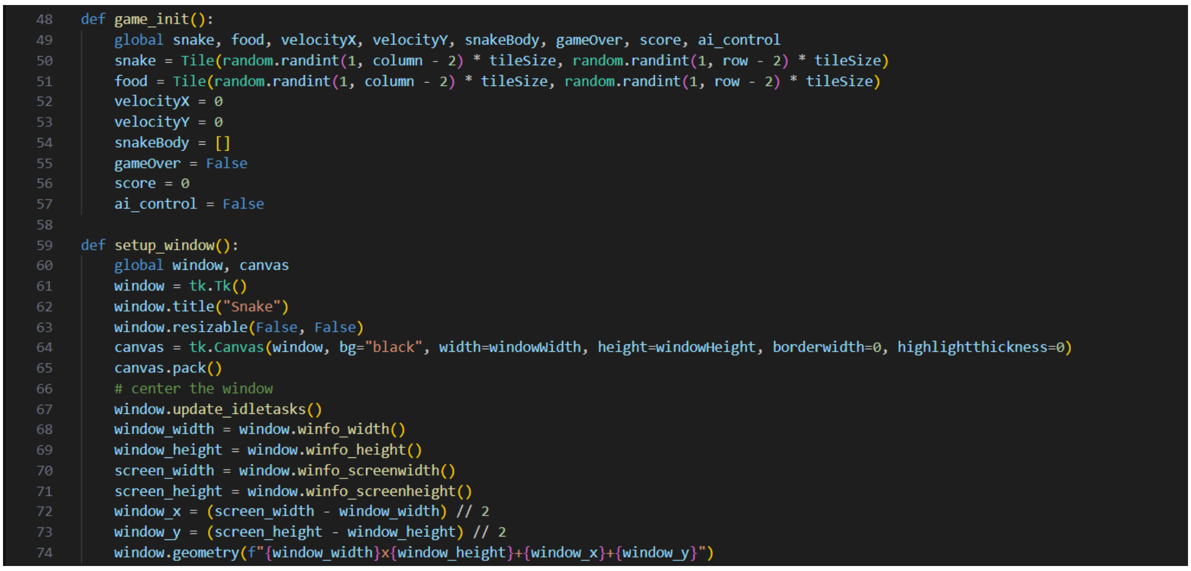
Description

The classic Snake game using Python and the Tkinter library for GUI. The game consists of a snake that moves around a grid, consuming food to grow longer. The player controls the snake's direction using the arrow keys. The game keeps track of the player's score and stores the highest score in a The game includes several algorithms and concepts:

1. A* Algorithm for AI Control
   The A* algorithm is used to control the snake's movement when ai_control is enabled. It finds the shortest path from the snake's current position to the food, considering obstacles (snake body and borders).
2. Collision Detection
   The program checks for collisions with walls, snake body segments, and food. This involves simple coordinate comparison and boundary checks.
3. Scorekeeping and Database Interaction
   The program tracks and displays the player's score and stores the highest score in a SQLite database. This involves basic database CRUD operations. SQLite database.

Implementation

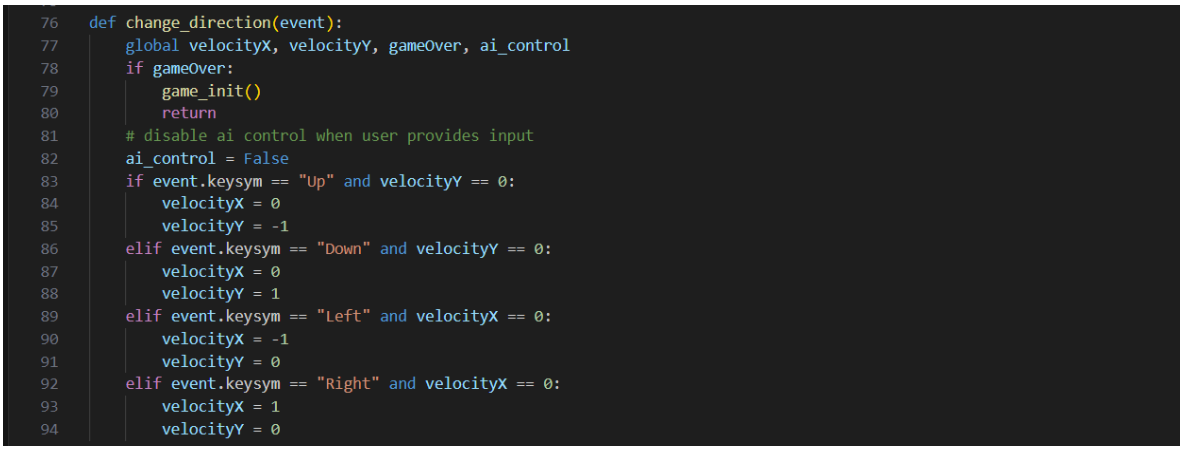
1. Setup and Initialization:



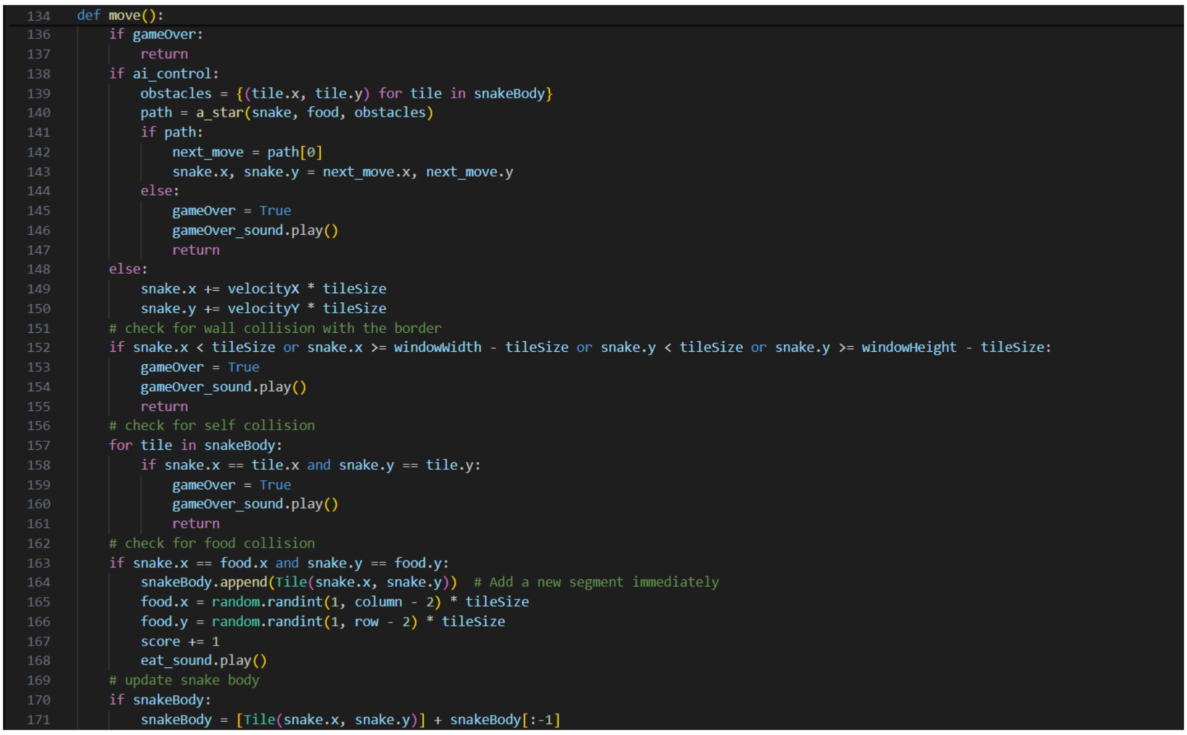
```python
def game_init():
    global snake, food, velocityX, velocityY, snakeBody, gameOver, score, ai_control
    snake = Tile(random.randint(1, column - 2) * tileSize, random.randint(1, row - 2) * tileSize)
    food = Tile(random.randint(1, column - 2) * tileSize, random.randint(1, row - 2) * tileSize)
    velocityX = 0
    velocityY = 0
    snakeBody = []
    gameOver = False
    score = 0
    ai_control = False

def setup_window():
    global window, canvas
    window = tk.Tk()
    window.title("Snake")
    window.resizable(False, False)
    canvas = tk.Canvas(window, bg="black", width=windowWidth, height=windowHeight, borderwidth=0, highlightthickness=0)
    canvas.pack()
    # center the window
    window.update_idletasks()
    window_width = window.winfo_width()
    window_height = window.winfo_height()
    screen_width = window.winfo_screenwidth()
    screen_height = window.winfo_screenheight()
    window_x = (screen_width - window_width) // 2
    window_y = (screen_height - window_height) // 2
    window.geometry(f"{window_width}x{window_height}+{window_x}+{window_y}")
```

   - The game initializes the SQLite database with a table for storing scores.
   - `game_init()` initializes the snake's position, food's position, velocity, snake's body, game status, and score.
   - `setup_window()` creates and configures the Tkinter window and canvas.

2. Game Mechanics:



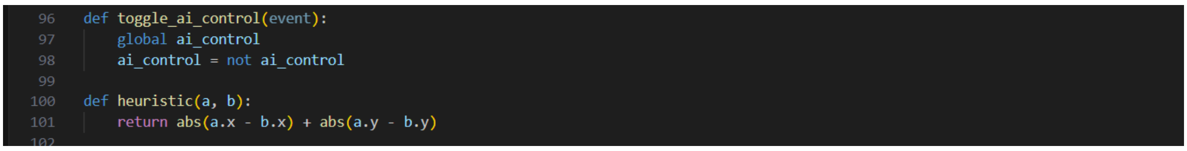
```python
def change_direction(event):
    global velocityX, velocityY, gameOver, ai_control
    if gameOver:
        game_init()
        return
    # disable ai control when user provides input
    ai_control = False
    if event.keysym == "Up" and velocityY == 0:
        velocityX = 0
        velocityY = -1
    elif event.keysym == "Down" and velocityY == 0:
        velocityX = 0
        velocityY = 1
    elif event.keysym == "Left" and velocityX == 0:
        velocityX = -1
        velocityY = 0
    elif event.keysym == "Right" and velocityX == 0:
        velocityX = 1
        velocityY = 0
```



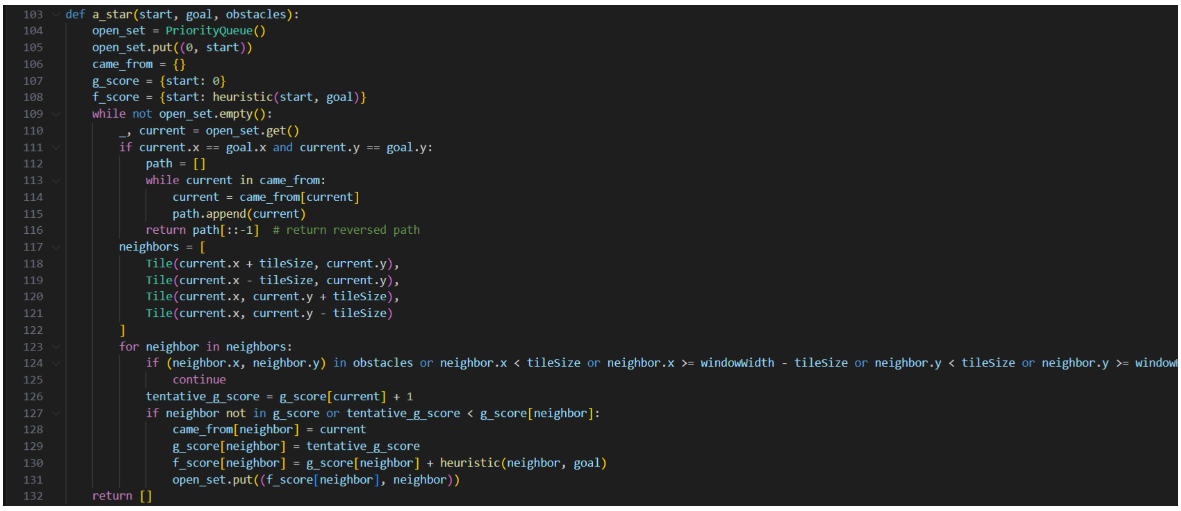
```python
def move():
    if gameOver:
        return
    if ai_control:
        obstacles = {(tile.x, tile.y) for tile in snakeBody}
        path = a_star(snake, food, obstacles)
        if path:
            next_move = path[0]
            snake.x, snake.y = next_move.x, next_move.y
        else:
            gameOver = True
            gameOver_sound.play()
            return
    else:
        snake.x += velocityX * tileSize
        snake.y += velocityY * tileSize
    # check for wall collision with the border
    if snake.x < tileSize or snake.x >= windowWidth - tileSize or snake.y < tileSize or snake.y >= windowHeight - tileSize:
        gameOver = True
        gameOver_sound.play()
        return
    # check for self collision
    for tile in snakeBody:
        if snake.x == tile.x and snake.y == tile.y:
            gameOver = True
            gameOver_sound.play()
            return
    # check for food collision
    if snake.x == food.x and snake.y == food.y:
        snakeBody.append(Tile(snake.x, snake.y))  # Add a new segment immediately
        food.x = random.randint(1, column - 2) * tileSize
        food.y = random.randint(1, row - 2) * tileSize
        score += 1
        eat_sound.play()
    # update snake body
    if snakeBody:
        snakeBody = [Tile(snake.x, snake.y)] + snakeBody[:-1]
```

- `change_direction(event)` handles direction changes based on user input (arrow keys).
- `move()` updates the snake's position, checks for collisions with the wall, self-collisions, and food consumption. Updates the snake's body and score when food is eaten.

3. AI Control and A Algorithm*:



```python
def toggle_ai_control(event):
    global ai_control
    ai_control = not ai_control

def heuristic(a, b):
    return abs(a.x - b.x) + abs(a.y - b.y)
```

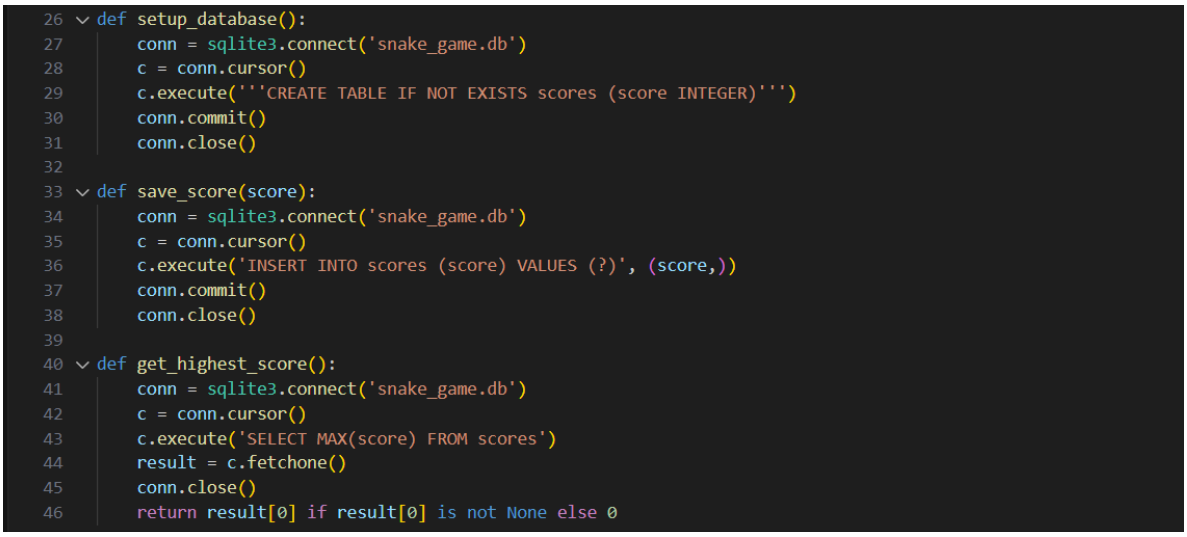


```python
def a_star(start, goal, obstacles):
    open_set = PriorityQueue()
    open_set.put((0, start))
    came_from = {}
    g_score = {start: 0}
    f_score = {start: heuristic(start, goal)}
    while not open_set.empty():
        _, current = open_set.get()
        if current.x == goal.x and current.y == goal.y:
            path = []
            while current in came_from:
                current = came_from[current]
                path.append(current)
            return path[::-1]  # return reversed path
        neighbors = [
            Tile(current.x + tileSize, current.y),
            Tile(current.x - tileSize, current.y),
            Tile(current.x, current.y + tileSize),
            Tile(current.x, current.y - tileSize)
        ]
        for neighbor in neighbors:
            if (neighbor.x, neighbor.y) in obstacles or neighbor.x < tileSize or neighbor.x >= windowWidth - tileSize or neighbor.y < tileSize or neighbor.y >= windowH
                continue
            tentative_g_score = g_score[current] + 1
            if neighbor not in g_score or tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = g_score[neighbor] + heuristic(neighbor, goal)
                open_set.put((f_score[neighbor], neighbor))
    return []
```

- `toggle_ai_control(event)` Toggles AI control on/off.

- `heuristic(a, b)` Calculates the heuristic using Manhattan distance between two points.
- `a_star(start, goal, obstacles)` Implements the A* algorithm to find the shortest path from the snake's current position to the food, avoiding obstacles.

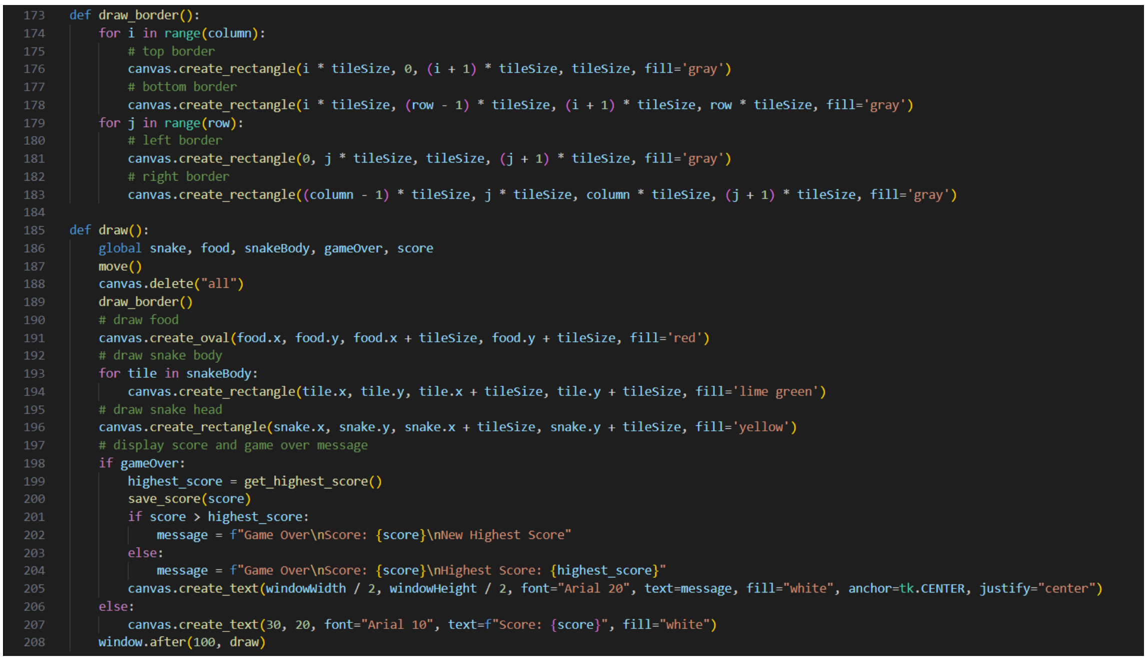
4. Database Interaction:



```
def setup_database():
    conn = sqlite3.connect('snake_game.db')
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS scores (score INTEGER)''')
    conn.commit()
    conn.close()

def save_score(score):
    conn = sqlite3.connect('snake_game.db')
    c = conn.cursor()
    c.execute('INSERT INTO scores (score) VALUES (?)', (score,))
    conn.commit()
    conn.close()

def get_highest_score():
    conn = sqlite3.connect('snake_game.db')
    c = conn.cursor()
    c.execute('SELECT MAX(score) FROM scores')
    result = c.fetchone()
    conn.close()
    return result[0] if result[0] is not None else 0
```

- `setup_database()`, `save_score(score)`, `get_highest_score()` functions for initializing, saving, and retrieving scores from the database.

5. Game Rendering:



```
def draw_border():
    for i in range(column):
        # top border
        canvas.create_rectangle(i * tileSize, 0, (i + 1) * tileSize, tileSize, fill='gray')
        # bottom border
        canvas.create_rectangle(i * tileSize, (row - 1) * tileSize, (i + 1) * tileSize, row * tileSize, fill='gray')
    for j in range(row):
        # left border
        canvas.create_rectangle(0, j * tileSize, tileSize, (j + 1) * tileSize, fill='gray')
        # right border
        canvas.create_rectangle((column - 1) * tileSize, j * tileSize, column * tileSize, (j + 1) * tileSize, fill='gray')

def draw():
    global snake, food, snakeBody, gameOver, score
    move()
    canvas.delete("all")
    draw_border()
    # draw food
    canvas.create_oval(food.x, food.y, food.x + tileSize, food.y + tileSize, fill='red')
    # draw snake body
    for tile in snakeBody:
        canvas.create_rectangle(tile.x, tile.y, tile.x + tileSize, tile.y + tileSize, fill='lime green')
    # draw snake head
    canvas.create_rectangle(snake.x, snake.y, snake.x + tileSize, snake.y + tileSize, fill='yellow')
    # display score and game over message
    if gameOver:
        highest_score = get_highest_score()
        save_score(score)
        if score > highest_score:
            message = f"Game Over\nScore: {score}\nNew Highest Score"
        else:
            message = f"Game Over\nScore: {score}\nHighest Score: {highest_score}"
        canvas.create_text(windowWidth / 2, windowHeight / 2, font="Arial 20", text=message, fill="white", anchor=tk.CENTER, justify="center")
    else:
        canvas.create_text(30, 20, font="Arial 10", text=f"Score: {score}", fill="white")
    window.after(100, draw)
```

- `draw_border` Draws the game borders.
- `draw()` handles the drawing of the game elements on the canvas, such as the snake, food, and border.

Output Analysis and Evaluation

Initial Output

1. Window Initialization:
   - A game window is created using tkinter with a size of 625x625 pixels (based on 25 rows and 25 columns, each 25 pixels in size).
   - The window is centered on the screen.
2. Game Elements:
   - The game initializes the snake's position randomly within the grid, avoiding the borders
   - The food position is also randomly initialized within the grid, avoiding the borders.

Gameplay

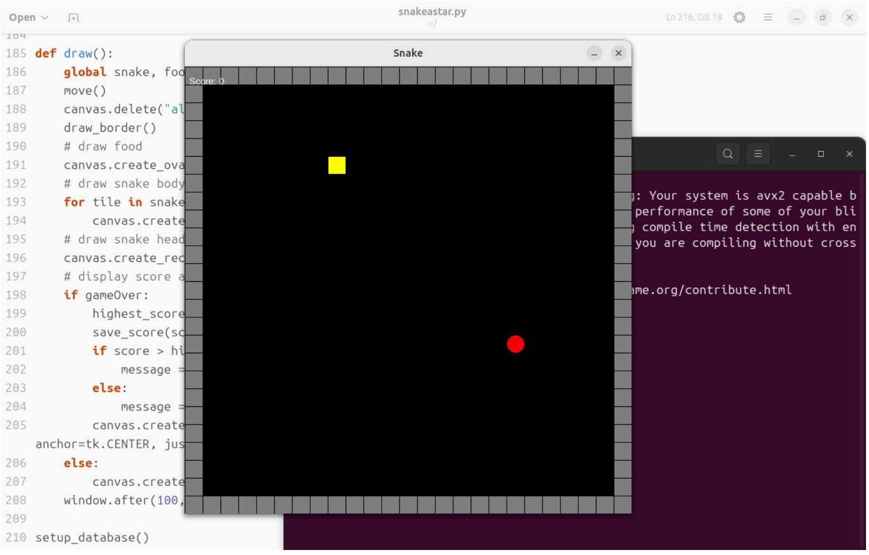
1. Drawing the Initial State:
    - The game border is drawn as gray rectangles around the edges of the grid.
    - The food is drawn as a red circle at its initial position.
    - The snake is drawn as a yellow square at its initial position.
    - The score is displayed at the top center of the window.
    - The highest score from previous games is also displayed at the top left of the window.
2. Snake Movement:
    - The snake starts stationary until a key press is detected.
    - Arrow keys change the direction of the snake's movement (Up, Down, Left, Right), ensuring the snake doesn't move in the opposite direction directly.
    - The snake's body follows the head's movement, growing when the snake eats food.
    - The game checks for collisions with the borders and the snake's own body.
3. AI Control:
    - Pressing the space bar toggles AI control on or off.
    - When AI control is enabled, the snake follows the shortest path to the food using the A* algorithm.
4. Food Consumption:
    - When the snake's head reaches the food position, the snake's body grows by one tile, and a new food position is generated randomly.
    - The score increments by one each time the snake eats food.
    - A sound effect (eat.wav) is played each time the snake eats food.
5. Game Over:
    - If the snake collides with the border or its own body, the game ends.
    - A "Game Over" message is displayed in the center of the window.
    - A game-over sound effect (die.wav) is played.
    - The current score is saved to the database.
    - The highest score is updated if the current score exceeds the previous high score.

Continuous Update

1. Game Loop:
    - The draw function is called repeatedly every 100 milliseconds, updating the game state and redrawing all elements.
    - The game continues running, allowing for continuous play until the player either loses or closes the window.

Output

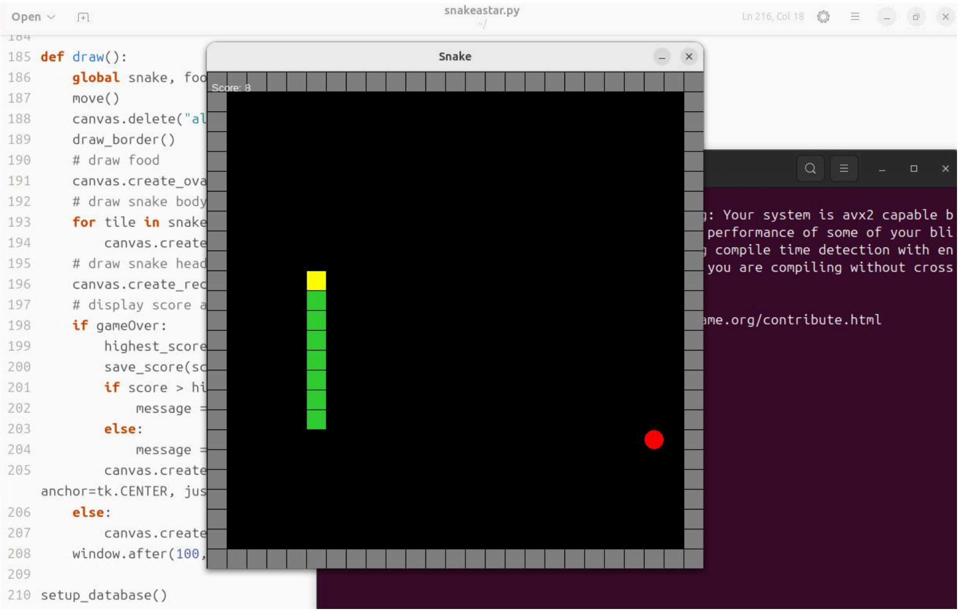
1. Initial State: A black window with gray borders, a yellow square (snake) at a random position, a red circle (food) at a random position, and the score displayed.



```
def draw():
    global snake, foo
    move()
    canvas.delete("al
    draw_border()
    # draw food
    canvas.create_ova
    # draw snake body
    for tile in snake
        canvas.create
    # draw snake head
    canvas.create_rec
    # display score a
    if gameOver:
        highest_score
        save_score(sc
        if score > hi
            message =
        else:
            message =
        canvas.create
anchor=tk.CENTER, jus
    else:
        canvas.create
    window.after(100,

setup_database()
```

```
: Your system is avx2 capable b
 performance of some of your bli
 compile time detection with en
 you are compiling without cross

me.org/contribute.html
```

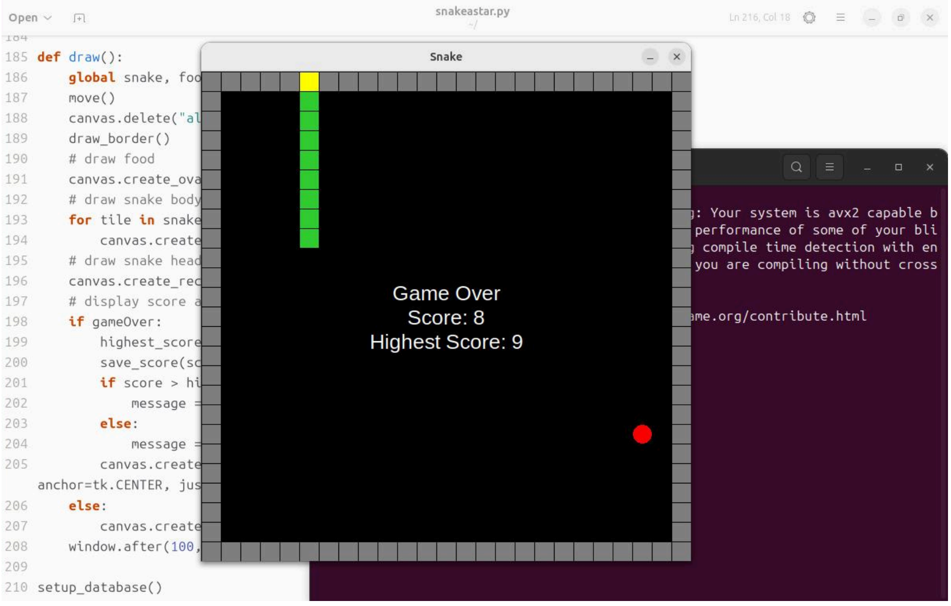
2. After Eating Food: The snake grows, the food moves to a new random position, the score increases, and an eating sound plays.



```
def draw():
    global snake, foo
    move()
    canvas.delete("al
    draw_border()
    # draw food
    canvas.create_ova
    # draw snake body
    for tile in snake
        canvas.create
    # draw snake head
    canvas.create_rec
    # display score a
    if gameOver:
        highest_score
        save_score(sc
        if score > hi
            message =
        else:
            message =
        canvas.create
anchor=tk.CENTER, jus
    else:
        canvas.create
    window.after(100,

setup_database()
```

```
: Your system is avx2 capable b
 performance of some of your bli
 compile time detection with en
 you are compiling without cross

me.org/contribute.html
```

3. After Collision: The game displays a "Game Over" message, plays a game-over sound, and saves the current score.



```
Game Over
Score: 8
Highest Score: 9
```

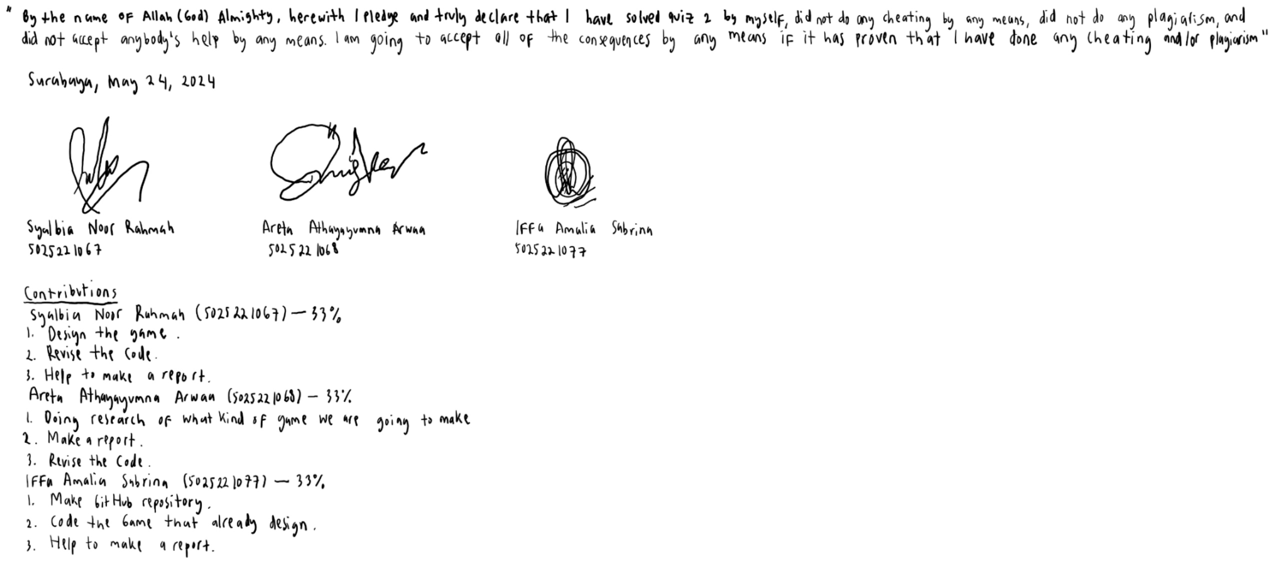
```
def draw():
    global snake, foo
    move()
    canvas.delete("al
    draw_border()
    # draw food
    canvas.create_ova
    # draw snake body
    for tile in snake
        canvas.create
    # draw snake head
    canvas.create_rec
    # display score a
    if gameOver:
        highest_score
        save_score(sc
        if score > hi
            message =
        else:
            message =
        canvas.create
anchor=tk.CENTER, jus
    else:
        canvas.create
    window.after(100,

setup_database()
```

```
: Your system is avx2 capable b
 performance of some of your bli
 compile time detection with en
 you are compiling without cross

me.org/contribute.html
```

Conclusion

The Snake game program effectively combines classic gameplay mechanics with features such as AI pathfinding, persistent high scores, and sound effects. It demonstrates a clear understanding of game design principles, algorithm implementation, and user interface development. This project not only provides an engaging and interactive gaming experience but also serves as an excellent example of integrating various programming concepts and technologies in a cohesive application.

Declaration



" By the name of Allah (God) Almighty, herewith I pledge and truly declare that I have solved quiz 2 by myself, did not do any cheating by any means, did not do any plagiarism, and did not accept anybody's help by any means. I am going to accept all of the consequences by any means if it has proven that I have done any cheating and/or plagiarism"

Surabaya, May 24, 2024

Syalbia Noor Rahmah
5025221067

Areta Athayayumna Arwaa
5025221068

IFFa Amalia Sabrina
5025221077

Contributions

Syalbia Noor Ruhmah (5025221067) — 33%

1. Design the game.
2. Revise the Code.
3. Help to make a report.

Areta Athayayumna Arwan (5025221068) — 33%

1. Doing research of what kind of game we are going to make
2. Make a report.
3. Revise the Code.

IFFa Amalia Sabrina (5025221077) — 33%

1. Make GitHub repository.
2. Code the Game that already design.
3. Help to make a report.