# Snake Game using Python

A project report submitted in partial fulfilment for the Award

Of

**BACHELOR OF TECHNOLOGY**

IN

Computer Science & Engineering

By

**Varshith Reddy Surakanti**

SECTION-K23CH

Roll no: 43

Reg.No: 12301407

TO

**LOVELY PROFESSIONAL UNIVERSITY, PUNJAB**

**INDIA**

# Acknowledgement

The satisfaction that accompanies the successful completion of this project would be in complete without the mention of the people who made it possible, without whose constant guidance and encouragement would have made efforts go in vain. I consider myself privileged to express gratitude and respect towards all those who guided us through the completion of this project.

I convey thanks to my project guide **Mr. Aman Kumar Sir** of **Computer Science and Engineering** Department for providing encouragement, constant support and guidance which was of a great help to complete this project successfully.

Last but not the least, we wish to thank our parents for financing our studies in this college as well as for constantly encouraging us to learn engineering. Their personal sacrifice in providing this opportunity to learn engineering is gratefully acknowledged.

# Abstract

This project focuses on developing a classic Snake game using Python and the pygame library. The Snake game is a popular arcade game where the player controls a growing snake to consume food items while avoiding collisions with the walls and the snake's own body. The primary objective is to achieve the highest possible score by maneuvering the snake efficiently.
Our game is designed with an intuitive and visually appealing interface that enhances the user experience. It features a checkered background, smooth snake movements, and responsive controls. The game includes sound effects for food consumption and game over scenarios, adding to the engaging gameplay experience.
The project is built on modular code principles, ensuring ease of maintenance and scalability.
Future iterations may include advanced features such as different game levels, powerups, and more complex obstacles to increase the game's challenge and excitement.
This project demonstrates the practical application of Python programming and game development principles, offering an enjoyable and educational experience for players and developers alike.

## Abbreviations

- GUI: Graphical User Interface

# Contents

1. Introduction

2 .Objective of the Project

3. Application Tools

4 .Requirement Specification

5. Project Design

6. Flow Chart

7. GUI of the Project

8. Project Implementation

9. Testing and Validation

10. Conclusion

11. References

# 1. Introduction

In this section, introduce the project and provide a brief overview of
what the Snake game is. Mention the programming language used and
the motivation behind creating the game.
**Example:** The Snake game is a classic arcade game developed using Python's turtle module. This project involves creating a simple yet engaging game where the player controls a snake that grows in length as it consumes food, while avoiding collisions with the walls and itself. The motivation behind this project is to demonstrate the use of Python .For game development and to provide a fun and interactive way to learn programming concepts.

# 2. Objective of Project

Outline the main goals of the project. What are you aiming to achieve by creating this game?
**Example:** The primary objective of this project is to develop an interactive and user-friendly Snake game using Python. The project aims to:

- Implement fundamental game development concepts.
- Enhance programming skills through practical application.
- Provide a fun and educational tool for learning Python.
- Create an engaging user interface with intuitive controls

# 3. Application Tools

**Programming Language:**

- Python: The primary programming language used for developing the Snake Game due to its simplicity and readability.

**IDEs (Integrated Development Environments):**

- PyCharm: A popular Python IDE known for its powerful code editing features and integrated tools.
- Jupyter Notebook: An open-source web application that allows you to create and share documents containing live code, equations, visualizations, and narrative text.
- Visual Studio Code: A versatile code editor that supports Python development with extensions and integrated tools.

**Libraries/Packages:**

- Turtle: A Python library used for creating the game's graphics and animations.
- Tkinter: Python's standard GUI (Graphical User Interface) package used for creating the game menus and user interface.
- Random: A Python library used for generating random positions for the fruits.

- Time: A Python library used for handling time delays and the game's pause functionality.

**Version Control:**

- Git: A version control system that helps track changes in the code, collaborate with others, and maintain the project's history.

**Other Tools:**

- Any additional tools or applications that enhance the project, such as:
    - Documentation tools like Sphinx for generating project documentation.
    - Testing frameworks like unittest for verifying the correctness of the code.

# 4. Requirements

List the software and hardware requirements needed to develop and run the game. Include any external libraries or tools used.

**Example:**

**Software Requirements:**

- Python 3.6 or higher
- turtle module (included in Python Standard Library)
- random module (included in Python Standard Library)
- time module (included in Python Standard Library)

**Hardware Requirements:**

- A computer with at least 2 GB of RAM
- A monitor with a resolution of 1024x768 or higher
- Keyboard for user input

# 5. Project Design

The Snake Game project is structured into several main components and functions, each responsible for specific tasks. This modular design helps in organizing the code, making it easier to understand, maintain, and extend.

**Main Components:**

1. **Initialization**:
    - **Global Variables**: Initializes variables such as snake, fruit, old_fruit, scoring, screen, score, delay, difficulty, paused, and pause_message.
    - **Tkinter Window**: Sets up the main menu and difficulty selection interfaces.
2. **Game Setup**:
    - **Start Game Function**: Initiates the game and shows the difficulty selection screen.
    - **Show Difficulty Screen Function**: Displays a new Tkinter window for selecting the difficulty level.
    - **Set Difficulty Function**: Adjusts the game delay based on the selected difficulty and starts the game with the chosen difficulty.
3. **Game Initialization**:
    - **Start Game with Difficulty Function**: Sets up the game screen, initializes the snake, fruit, scoring display, and pause message.
    - **Create Border Function**: Draws the game boundary using Turtle graphics.

4. **Game Mechanics**:

   - **Snake Movement Functions**: Controls the snake's movement using functions such as snake_go_up(), snake_go_down(), snake_go_left(), and snake_go_right().
   - **Toggle Pause Function**: Toggles the game's pause state and displays/hides the "Paused" message.

5. **Collision Detection and Game Logic**:

   - **Game Loop Function**: The main game loop that continuously updates the game state, checks for collisions, and updates the screen.
   - **Game Over Function**: Clears the screen and displays the game-over message with the final score.

**Functions and Their Interactions:**

1. **Global Variables**:
   - Define and initialize the main game elements and state variables used throughout the project.

2. **Tkinter Window Setup**:
   - start_game(): Hides the main menu and calls show_difficulty_screen().
   - show_difficulty_screen(): Creates a new window for difficulty selection and provides buttons for Easy, Medium, and Hard levels.
   - set_difficulty(level, window): Adjusts the game delay based on the selected difficulty and calls start_game_with_difficulty().

3. **Game Screen Initialization**:
   - start_game_with_difficulty(): Initializes the Turtle screen, sets up the snake and fruit, creates the game border, and initializes the scoring display and pause message.
   - create_border(): Draws the game border using Turtle graphics.

4. **Snake Movement**:
   - snake_go_up(), snake_go_down(), snake_go_left(), snake_go_right(): Update the snake's direction based on user input.
   - snake_move(): Moves the snake in the current direction and updates its position.

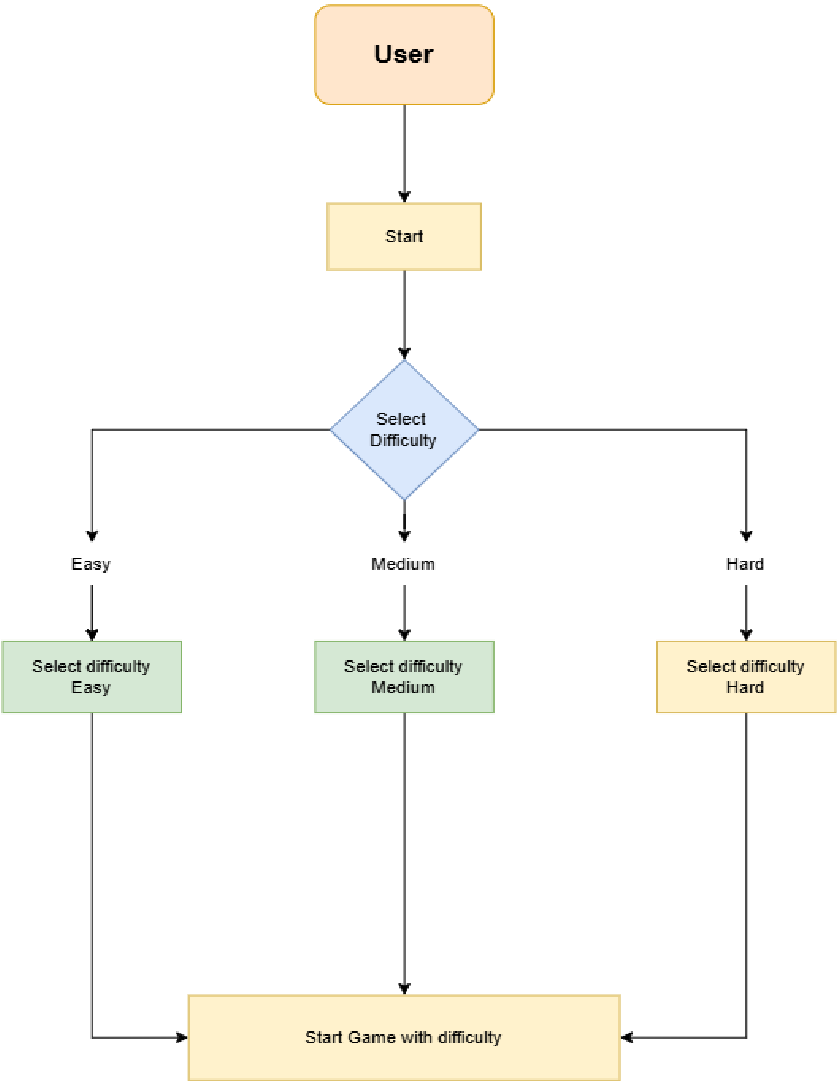
5. **Pause Functionality**:
   - toggle_pause(): Toggles the game's paused state and updates the pause message display.

6. **Game Loop and Logic**:
   - game_loop(): The main game loop that updates the screen, checks for collisions, and updates the score and snake segments.
   - game_over(score): Displays the game-over message with the final score.

# 6. Flow Chart :

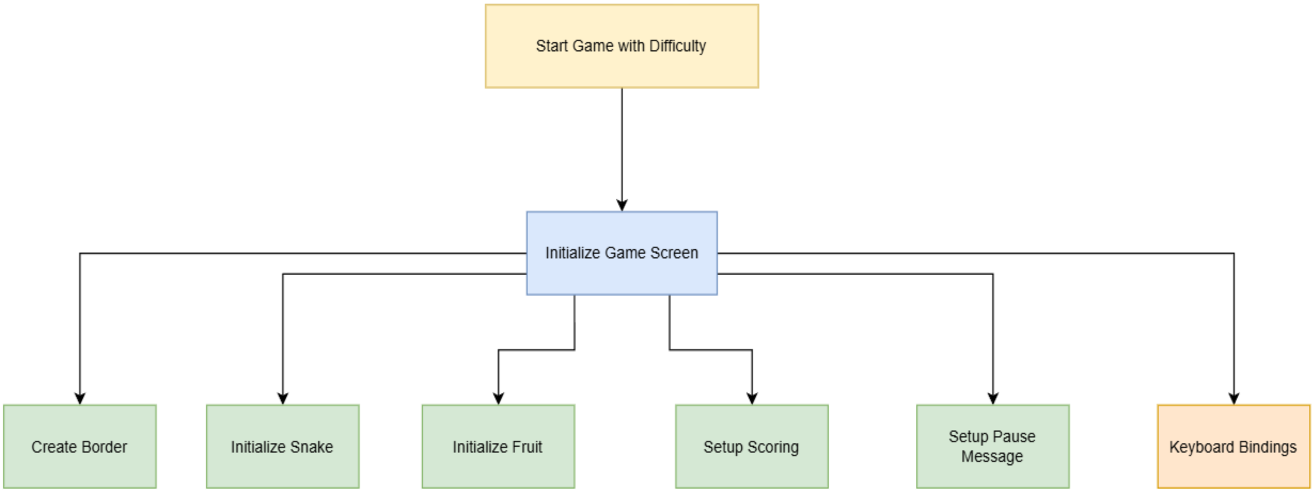
## 1. Starting Game & Selecting Difficulty:

Varshith Reddy

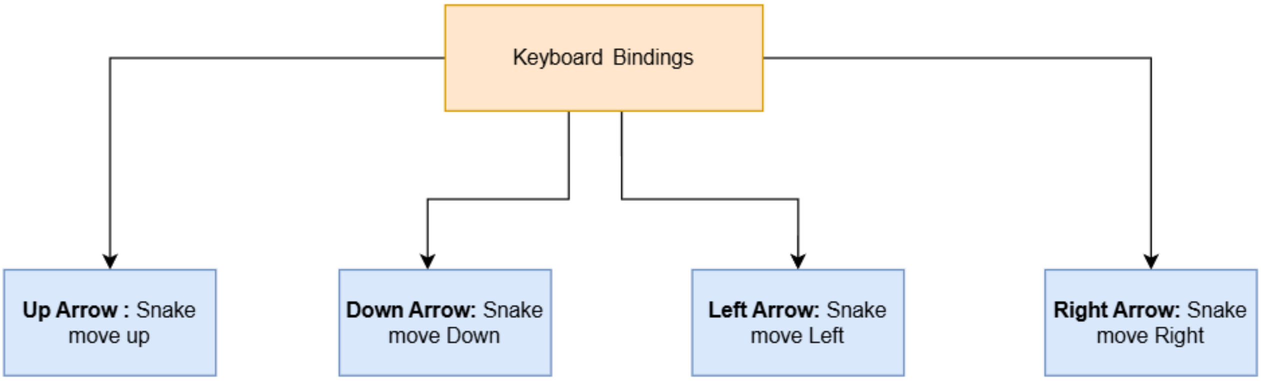


## 2. Initializing Game Screen according to Selected Difficulty:

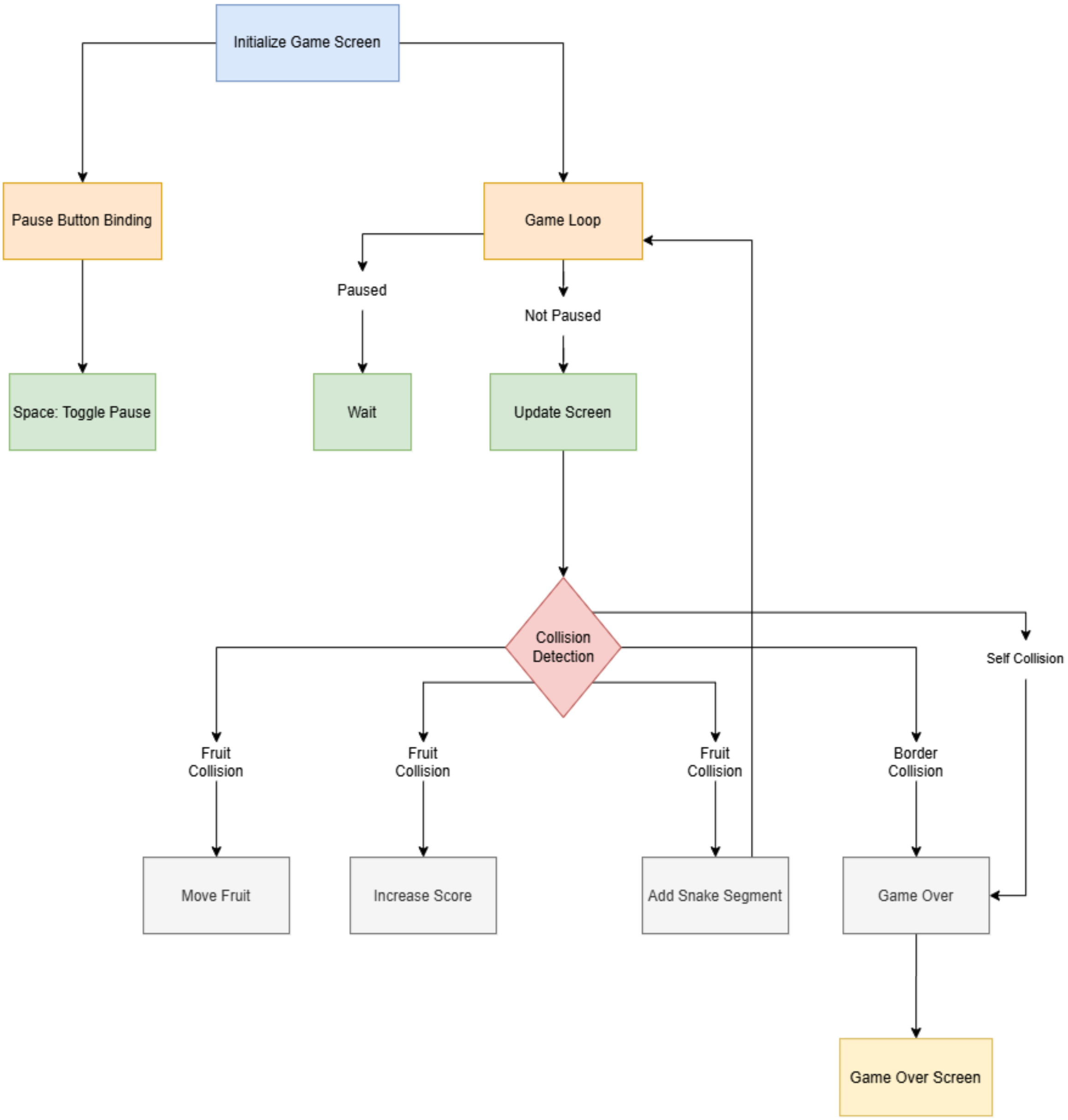
Varshith reddy



## 3. Snake Moves according to user input direction:


Keyboard Bindings
Up Arrow : Snake move up
Down Arrow: Snake move Down
Left Arrow: Snake move Left
Right Arrow: Snake move Right

## 4. Game loop , Pause button , Collision Detection and Game over screen:



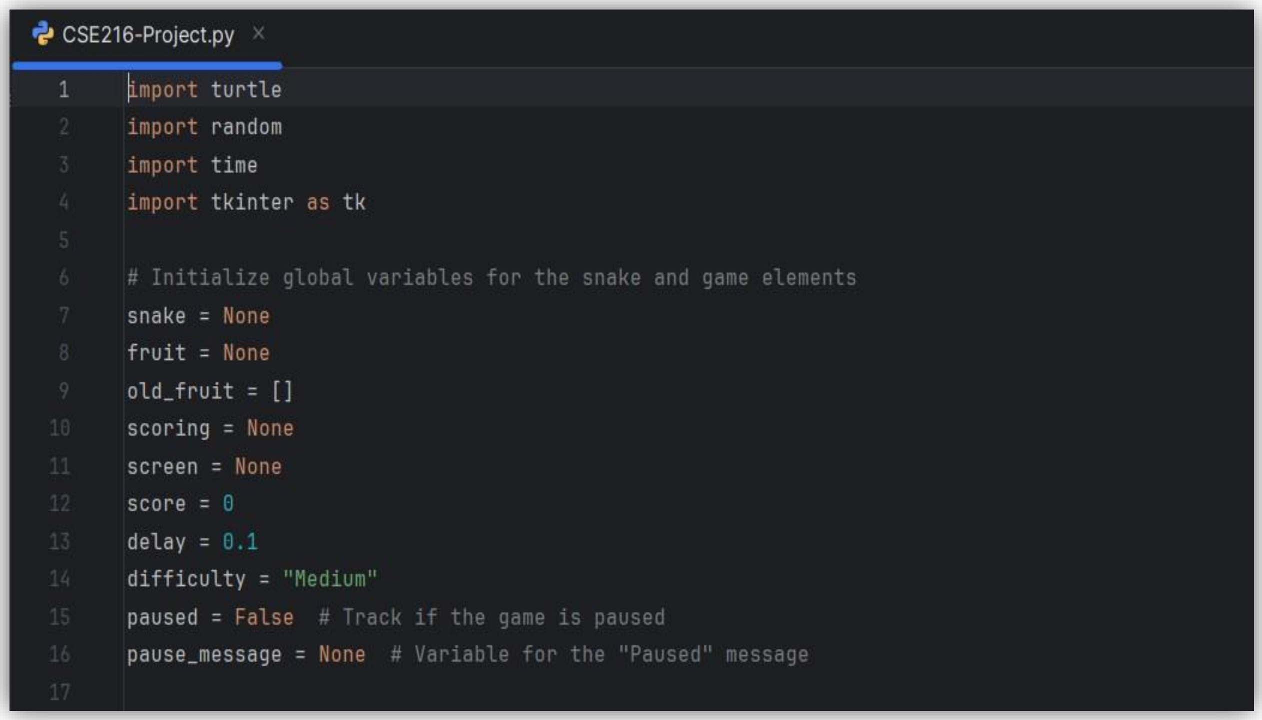
## 7. GUI of the project :

Describe the graphical user interface of the game. Include information about the layout, design elements, and controls.

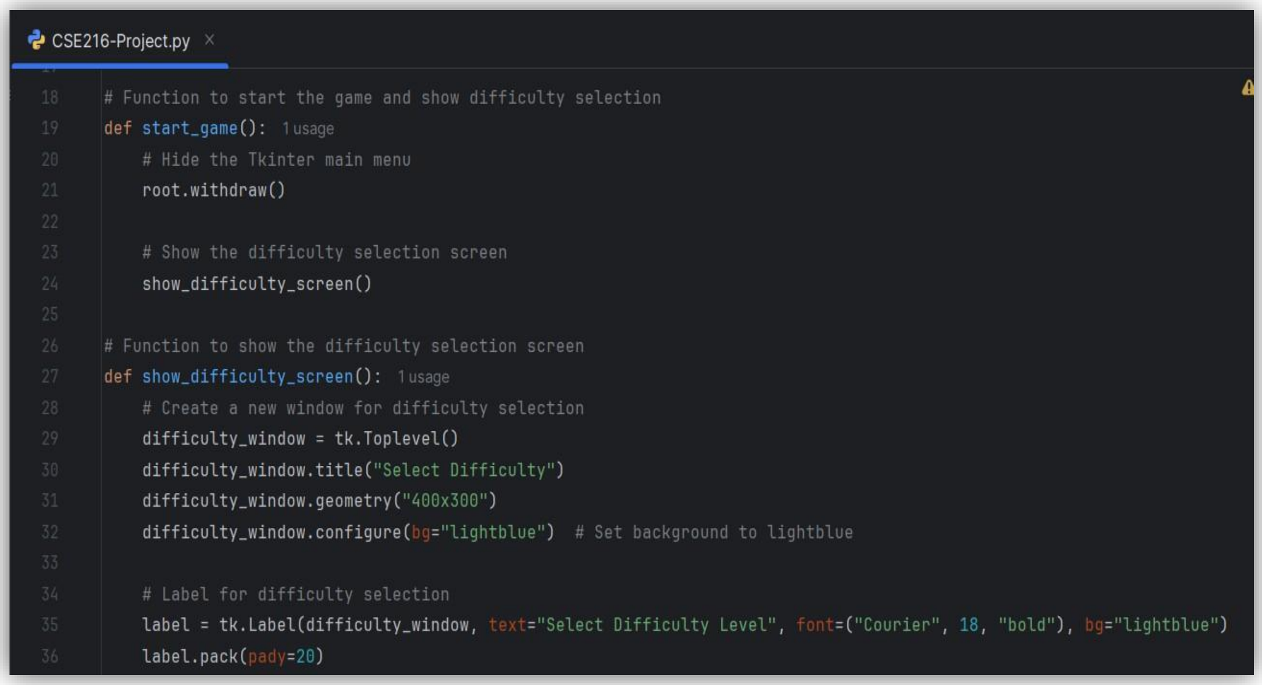
**Example:** The graphical user interface of the Snake game includes the following elements:

- **Game Screen:** The main playing area where the snake moves and consumes food.
- **Score Display:** A text element at the top of the screen displaying the current score.
- **Start Button:** A button to initiate the game.
- **Game Over Screen:** A message displayed when the game ends, with an option to restart the game.

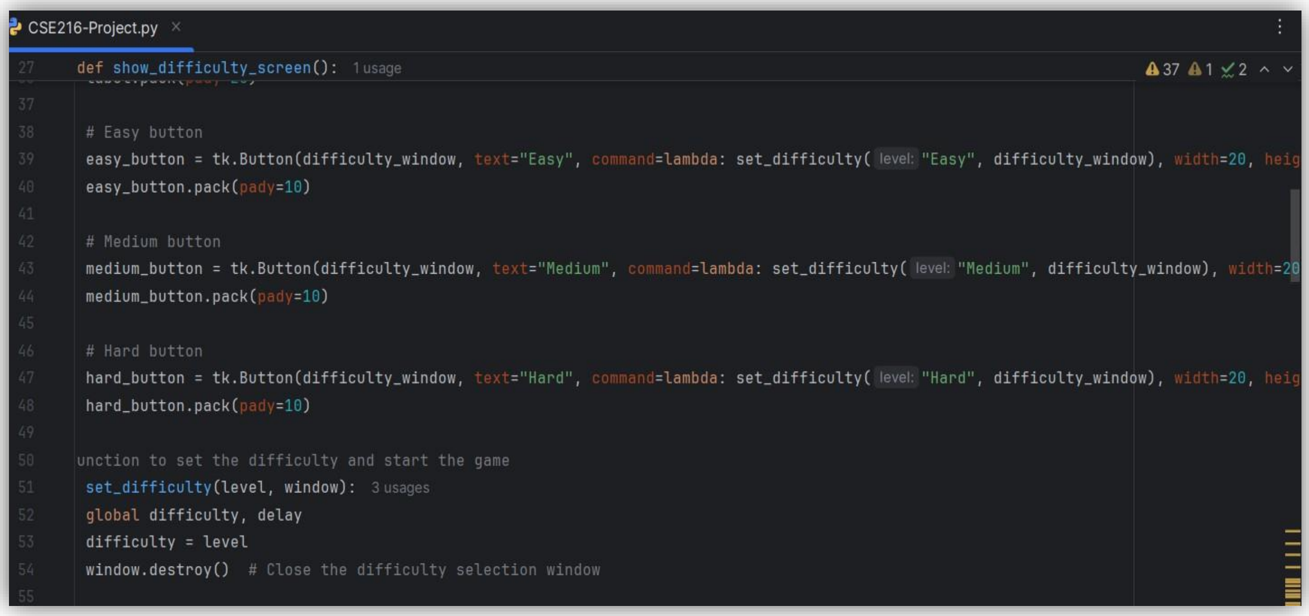
## 8. Project Implementation



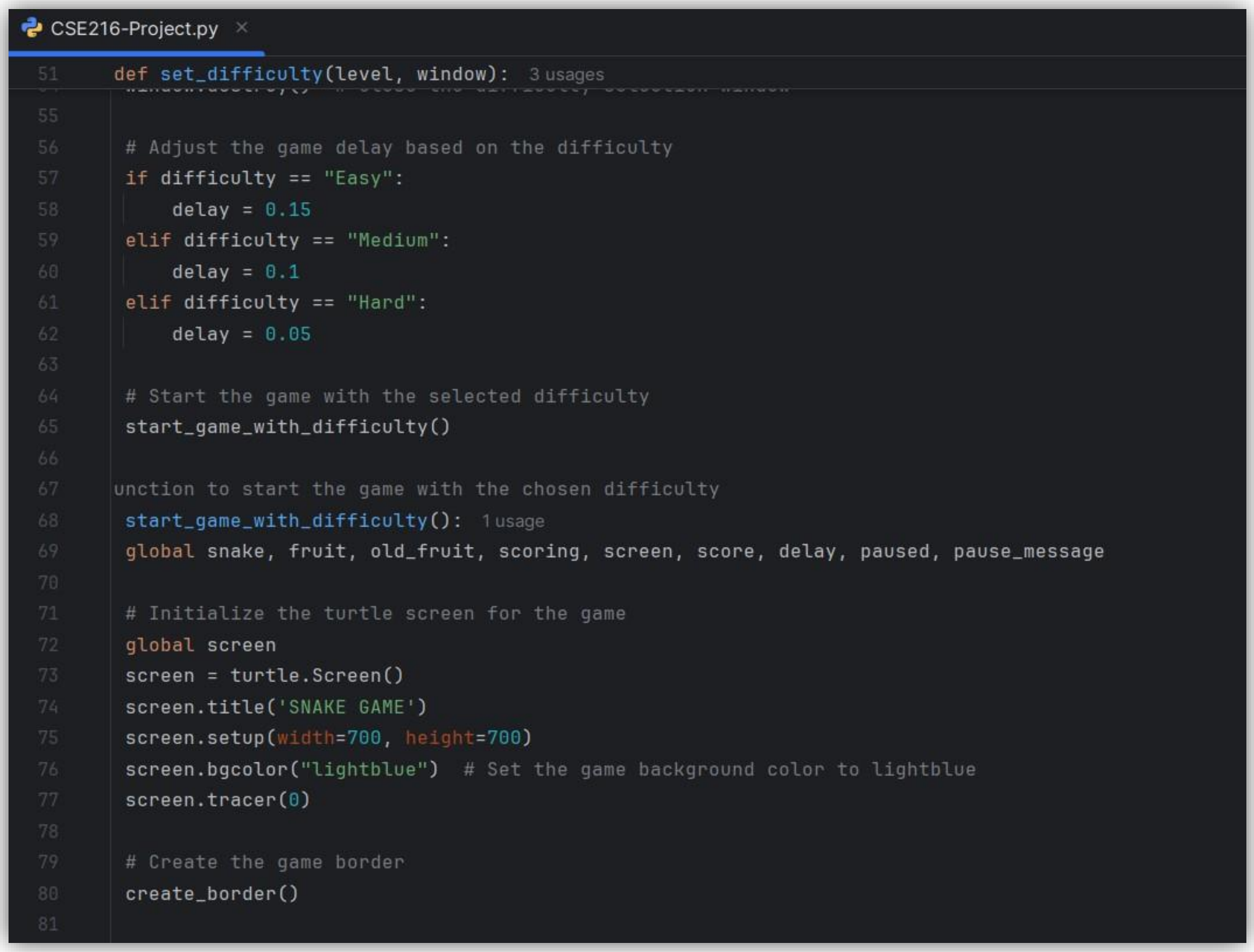
```python
import turtle
import random
import time
import tkinter as tk

# Initialize global variables for the snake and game elements
snake = None
fruit = None
old_fruit = []
scoring = None
screen = None
score = 0
delay = 0.1
difficulty = "Medium"
paused = False  # Track if the game is paused
pause_message = None  # Variable for the "Paused" message

```



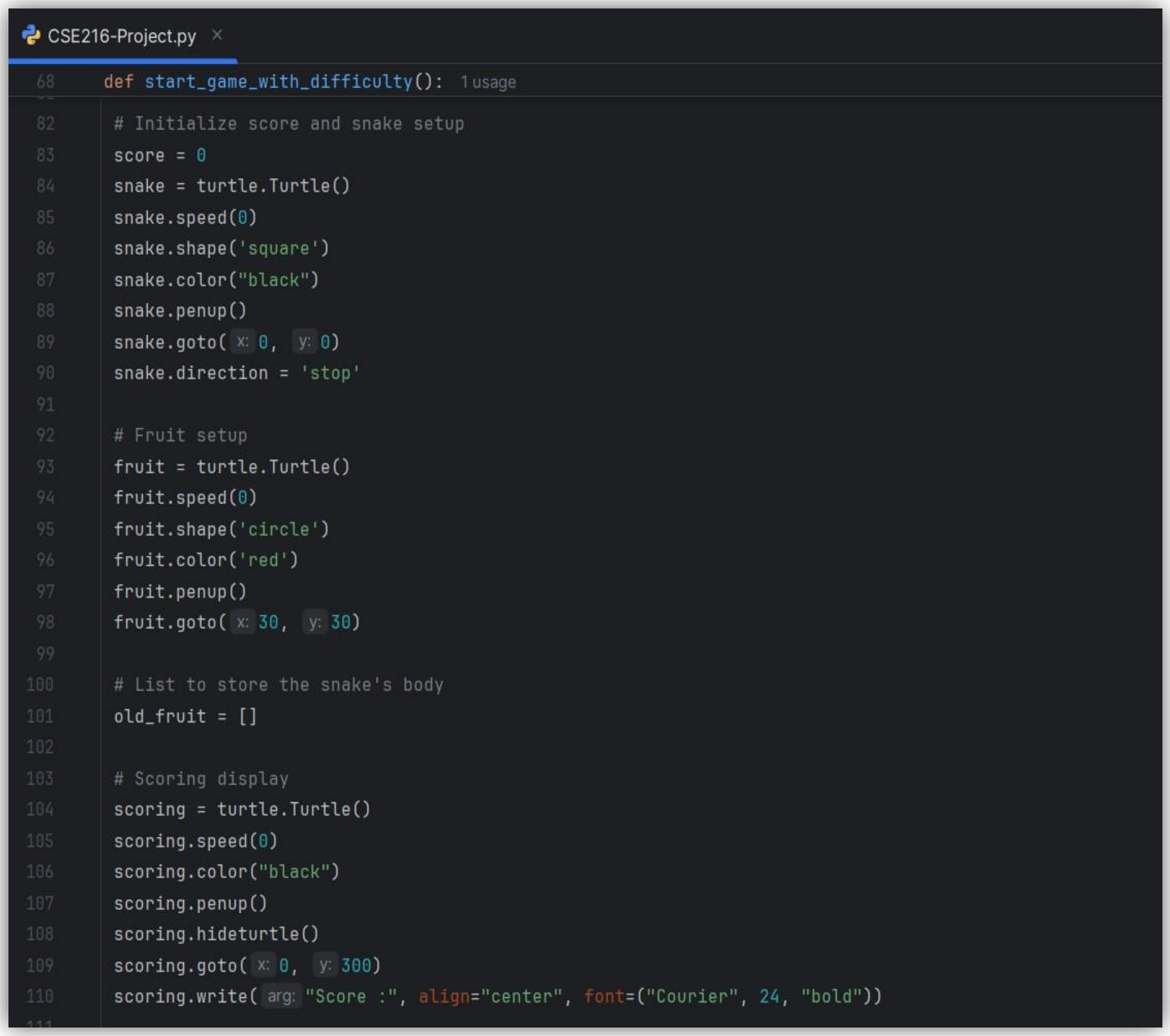
```
# Function to start the game and show difficulty selection
def start_game():  1 usage
    # Hide the Tkinter main menu
    root.withdraw()

    # Show the difficulty selection screen
    show_difficulty_screen()

# Function to show the difficulty selection screen
def show_difficulty_screen():  1 usage
    # Create a new window for difficulty selection
    difficulty_window = tk.Toplevel()
    difficulty_window.title("Select Difficulty")
    difficulty_window.geometry("400x300")
    difficulty_window.configure(bg="lightblue")  # Set background to lightblue

    # Label for difficulty selection
    label = tk.Label(difficulty_window, text="Select Difficulty Level", font=("Courier", 18, "bold"), bg="lightblue")
    label.pack(pady=20)
```



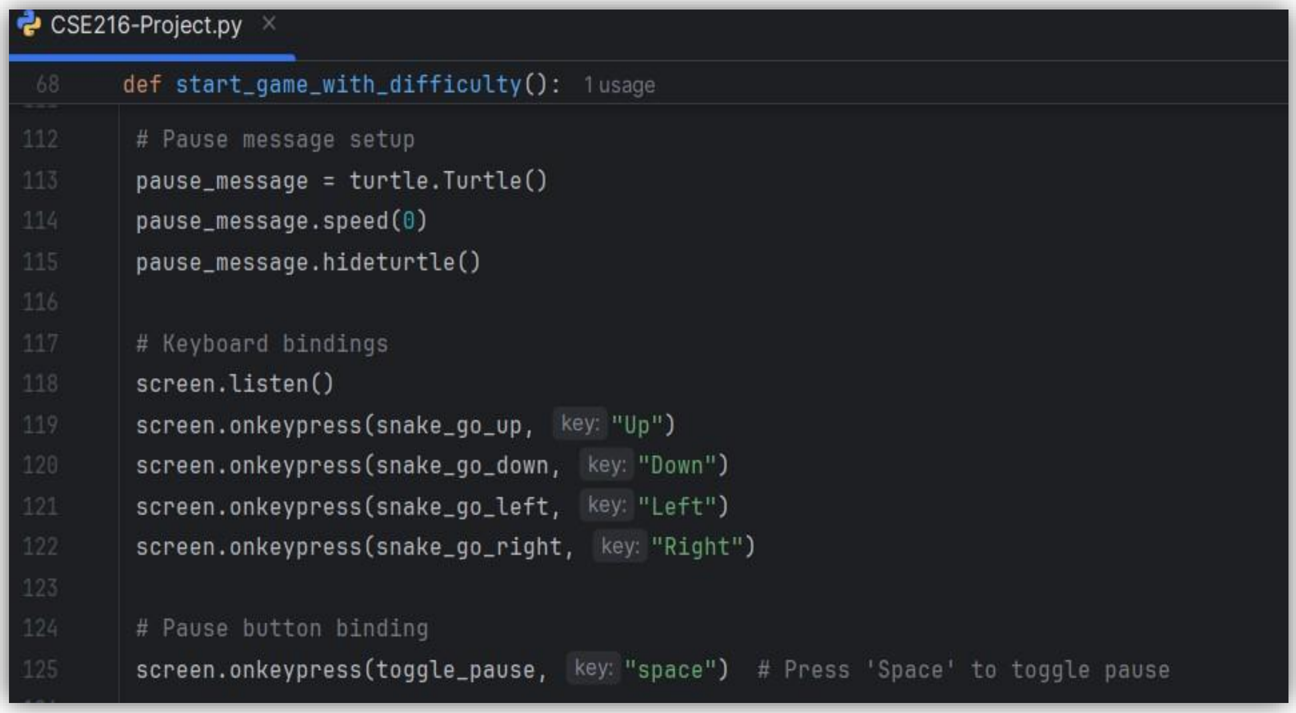
```
def show_difficulty_screen():  1 usage

    # Easy button
    easy_button = tk.Button(difficulty_window, text="Easy", command=lambda: set_difficulty( level: "Easy", difficulty_window), width=20, heig
    easy_button.pack(pady=10)

    # Medium button
    medium_button = tk.Button(difficulty_window, text="Medium", command=lambda: set_difficulty( level: "Medium", difficulty_window), width=20
    medium_button.pack(pady=10)

    # Hard button
    hard_button = tk.Button(difficulty_window, text="Hard", command=lambda: set_difficulty( level: "Hard", difficulty_window), width=20, heig
    hard_button.pack(pady=10)

unction to set the difficulty and start the game
    set_difficulty(level, window):  3 usages
    global difficulty, delay
    difficulty = level
    window.destroy()  # Close the difficulty selection window
```



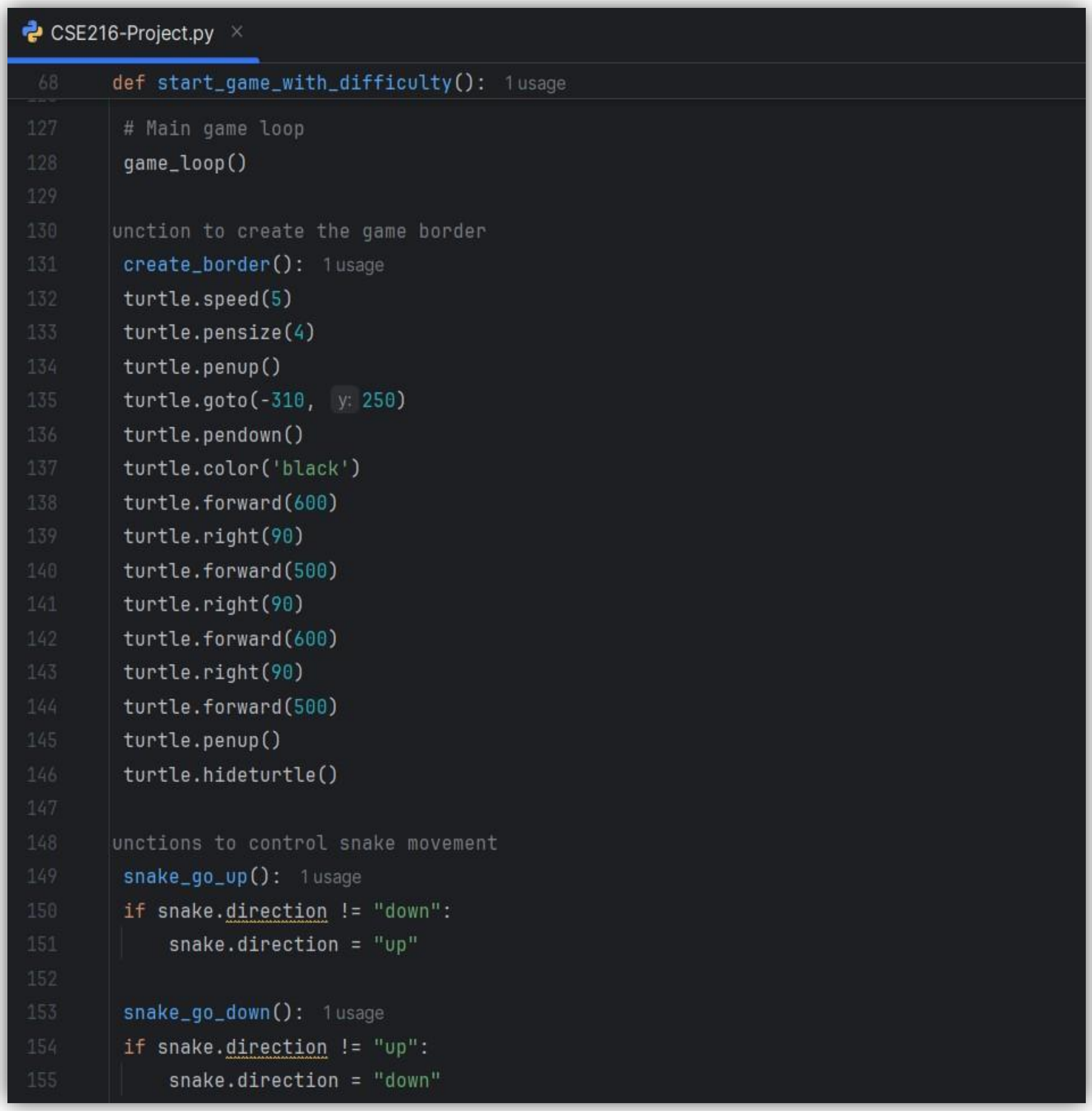
```
def set_difficulty(level, window):  3 usages

 # Adjust the game delay based on the difficulty
 if difficulty == "Easy":
     delay = 0.15
 elif difficulty == "Medium":
     delay = 0.1
 elif difficulty == "Hard":
     delay = 0.05

 # Start the game with the selected difficulty
 start_game_with_difficulty()

unction to start the game with the chosen difficulty
 start_game_with_difficulty():  1 usage
 global snake, fruit, old_fruit, scoring, screen, score, delay, paused, pause_message

 # Initialize the turtle screen for the game
 global screen
 screen = turtle.Screen()
 screen.title('SNAKE GAME')
 screen.setup(width=700, height=700)
 screen.bgcolor("lightblue")  # Set the game background color to lightblue
 screen.tracer(0)

 # Create the game border
 create_border()

```



```python
def start_game_with_difficulty():  1 usage
    # Initialize score and snake setup
    score = 0
    snake = turtle.Turtle()
    snake.speed(0)
    snake.shape('square')
    snake.color("black")
    snake.penup()
    snake.goto( x: 0,  y: 0)
    snake.direction = 'stop'

    # Fruit setup
    fruit = turtle.Turtle()
    fruit.speed(0)
    fruit.shape('circle')
    fruit.color('red')
    fruit.penup()
    fruit.goto( x: 30,  y: 30)

    # List to store the snake's body
    old_fruit = []

    # Scoring display
    scoring = turtle.Turtle()
    scoring.speed(0)
    scoring.color("black")
    scoring.penup()
    scoring.hideturtle()
    scoring.goto( x: 0,  y: 300)
    scoring.write( arg: "Score :", align="center", font=("Courier", 24, "bold"))
```

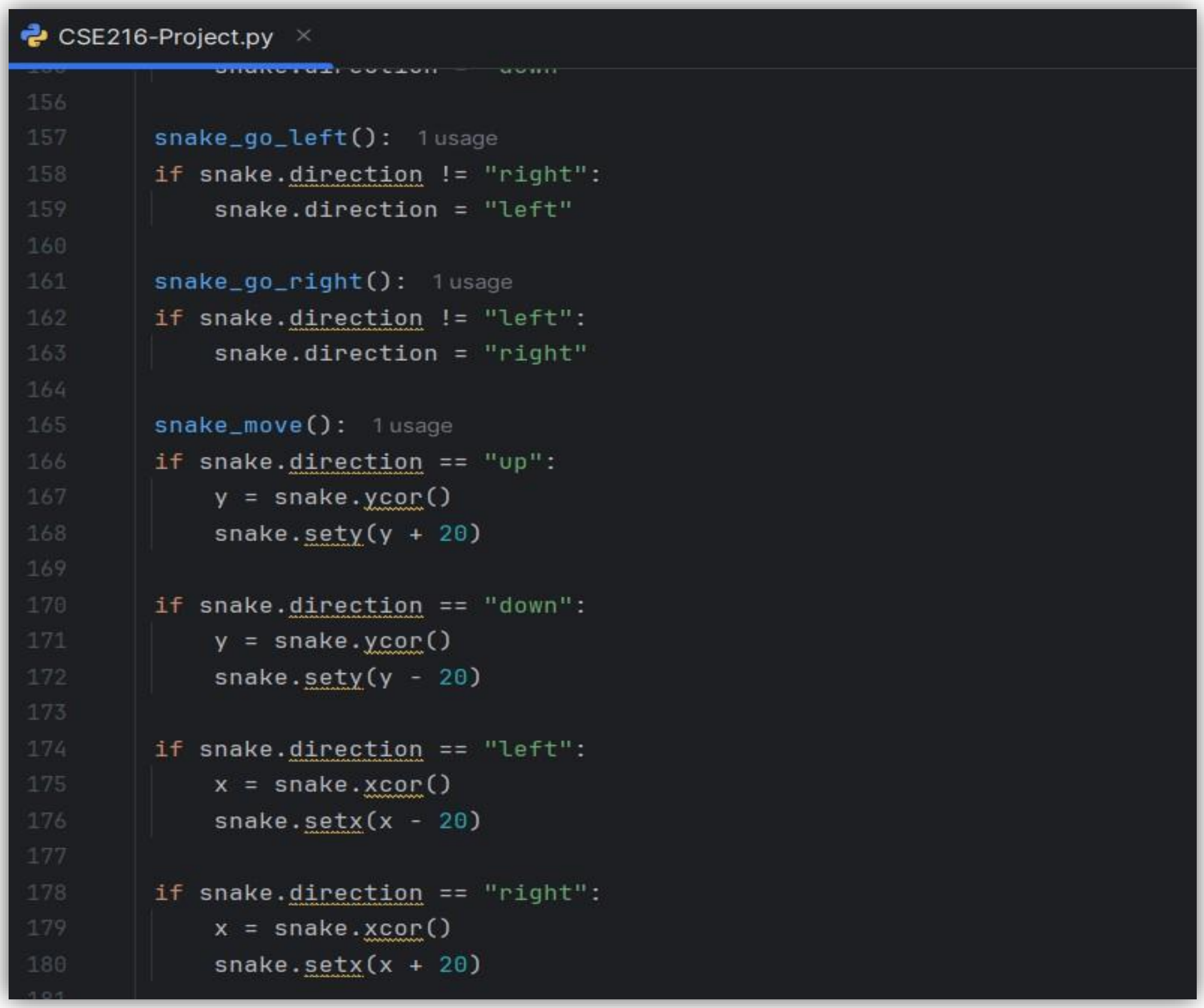


```
def start_game_with_difficulty():  1 usage
    # Pause message setup
    pause_message = turtle.Turtle()
    pause_message.speed(0)
    pause_message.hideturtle()

    # Keyboard bindings
    screen.listen()
    screen.onkeypress(snake_go_up,  key: "Up")
    screen.onkeypress(snake_go_down,  key: "Down")
    screen.onkeypress(snake_go_left,  key: "Left")
    screen.onkeypress(snake_go_right,  key: "Right")

    # Pause button binding
    screen.onkeypress(toggle_pause,  key: "space")  # Press 'Space' to toggle pause
```

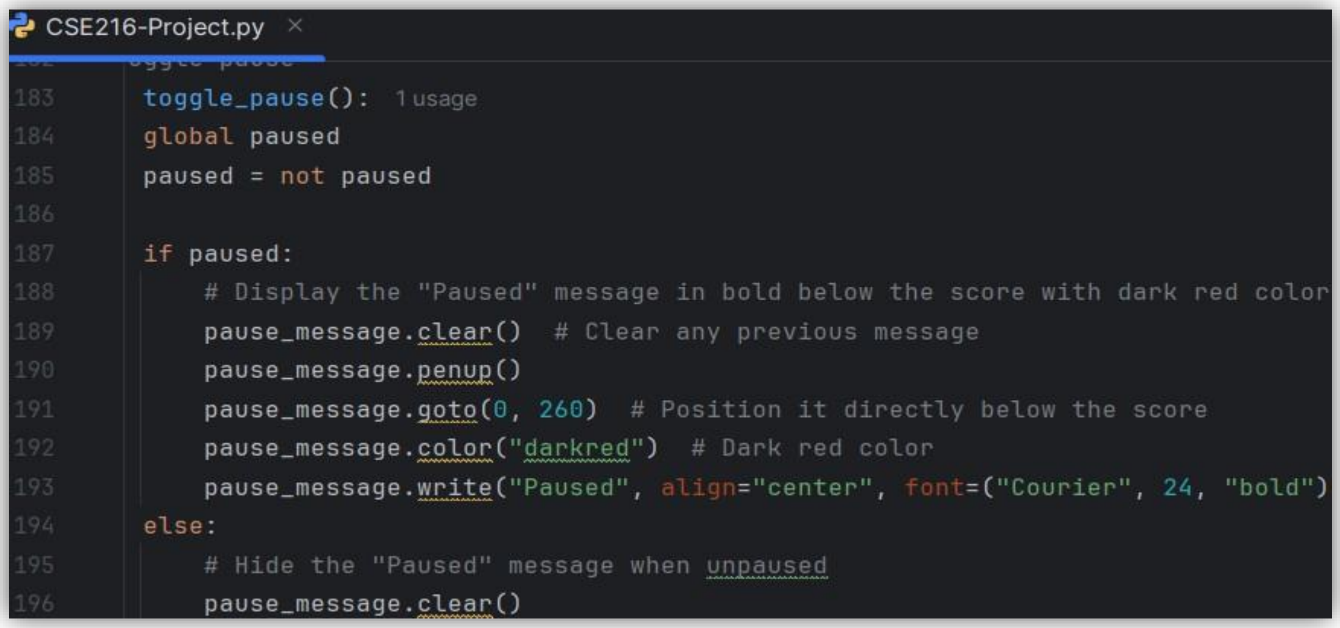


CSE216-Project.py

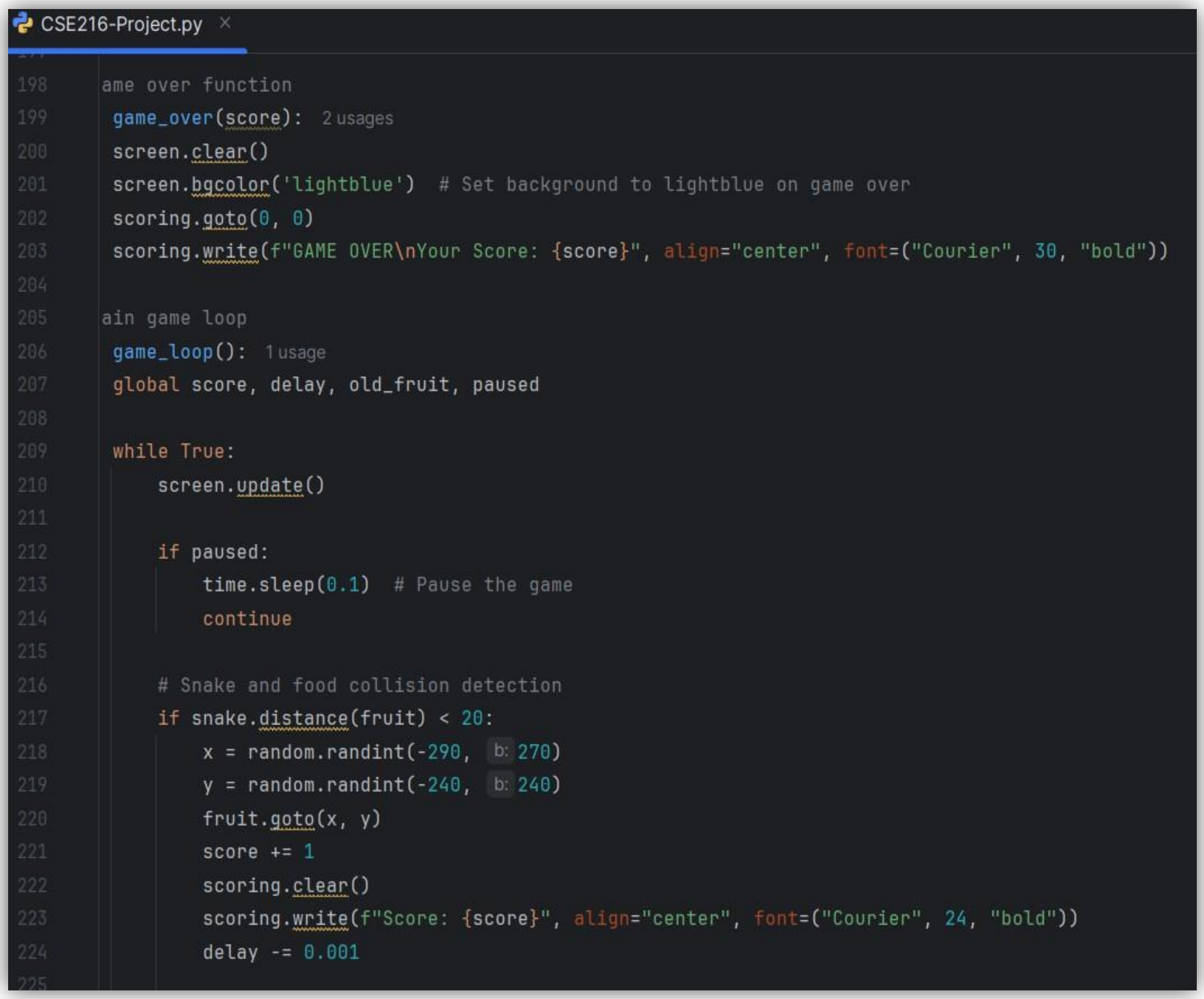
```
def start_game_with_difficulty():  1 usage
 # Main game loop
 game_loop()

unction to create the game border
 create_border():  1 usage
 turtle.speed(5)
 turtle.pensize(4)
 turtle.penup()
 turtle.goto(-310,  y: 250)
 turtle.pendown()
 turtle.color('black')
 turtle.forward(600)
 turtle.right(90)
 turtle.forward(500)
 turtle.right(90)
 turtle.forward(600)
 turtle.right(90)
 turtle.forward(500)
 turtle.penup()
 turtle.hideturtle()

unctions to control snake movement
 snake_go_up():  1 usage
 if snake.direction != "down":
     snake.direction = "up"

 snake_go_down():  1 usage
 if snake.direction != "up":
     snake.direction = "down"
```



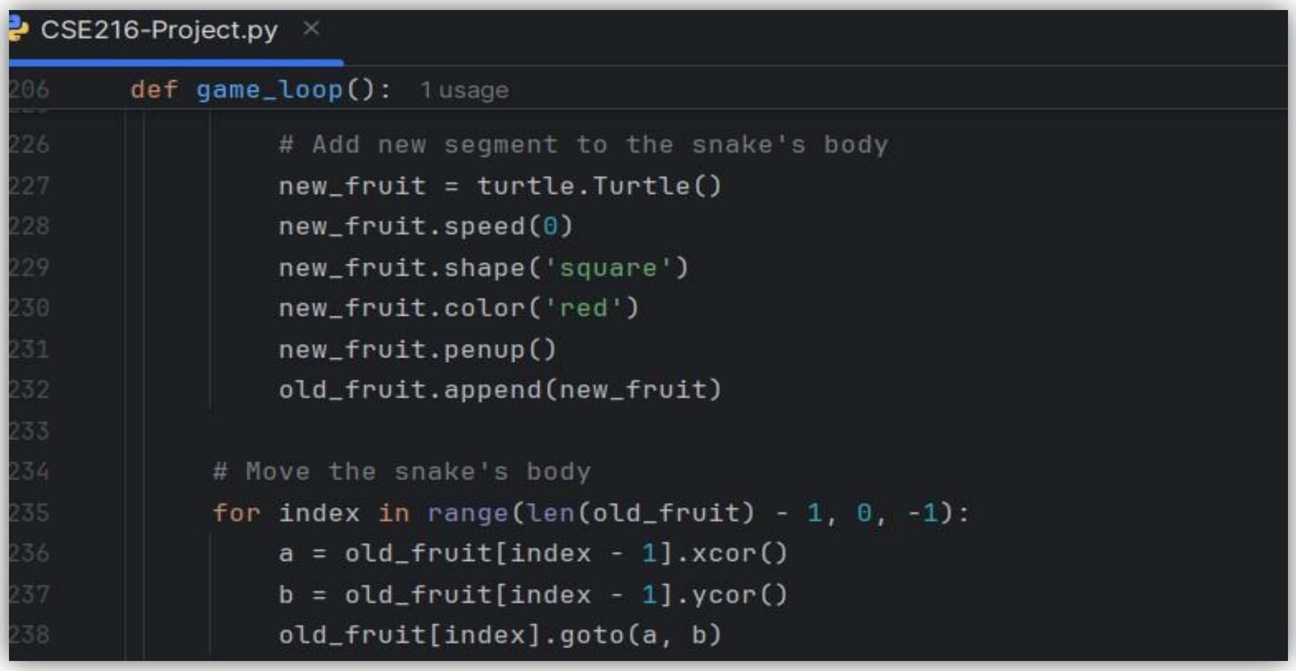
```
    snake.direction = "down"

snake_go_left():  1 usage
if snake.direction != "right":
    snake.direction = "left"

snake_go_right():  1 usage
if snake.direction != "left":
    snake.direction = "right"

snake_move():  1 usage
if snake.direction == "up":
    y = snake.ycor()
    snake.sety(y + 20)

if snake.direction == "down":
    y = snake.ycor()
    snake.sety(y - 20)

if snake.direction == "left":
    x = snake.xcor()
    snake.setx(x - 20)

if snake.direction == "right":
    x = snake.xcor()
    snake.setx(x + 20)
```



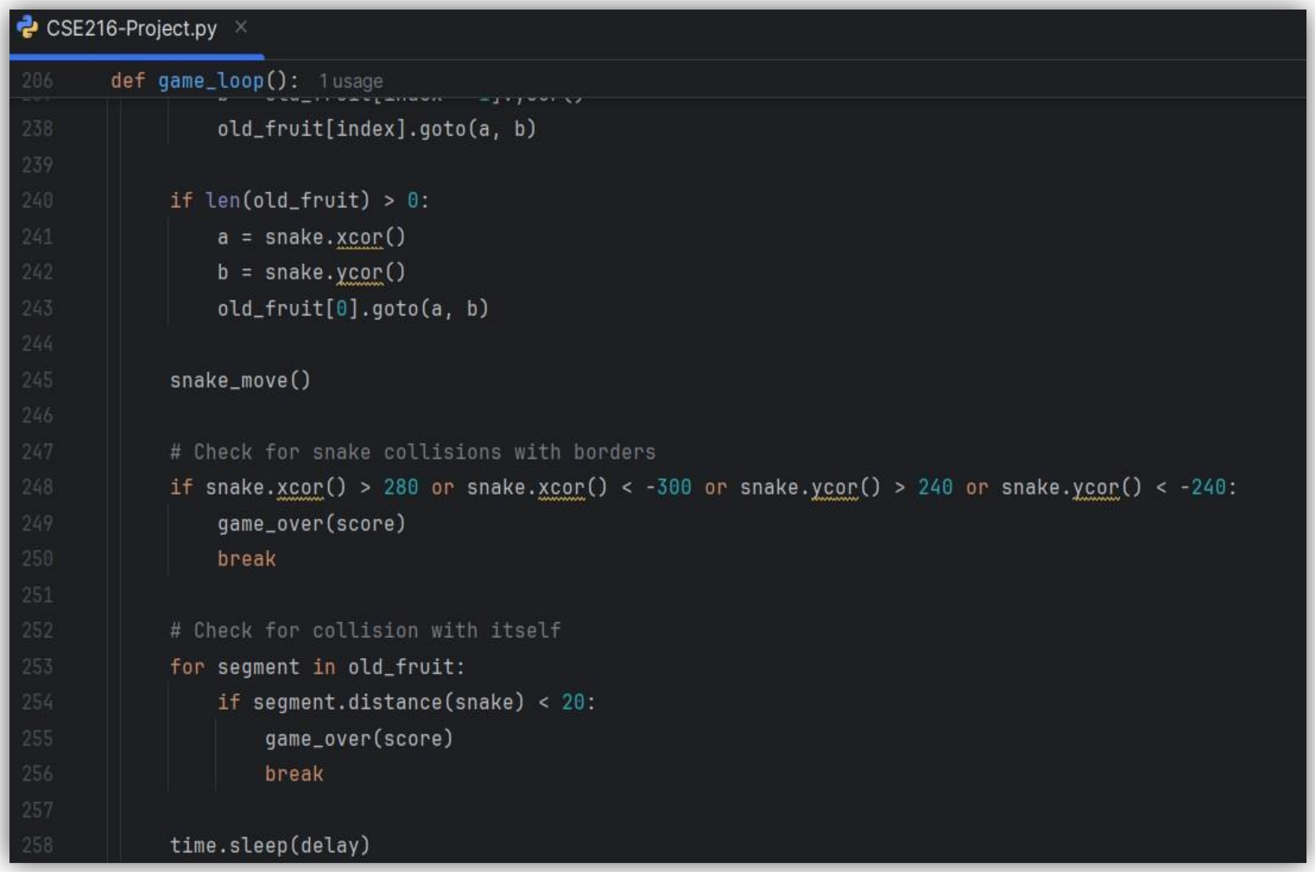
```
toggle_pause():  1 usage
global paused
paused = not paused

if paused:
    # Display the "Paused" message in bold below the score with dark red color
    pause_message.clear()  # Clear any previous message
    pause_message.penup()
    pause_message.goto(0, 260)  # Position it directly below the score
    pause_message.color("darkred")  # Dark red color
    pause_message.write("Paused", align="center", font=("Courier", 24, "bold")
else:
    # Hide the "Paused" message when unpaused
    pause_message.clear()
```



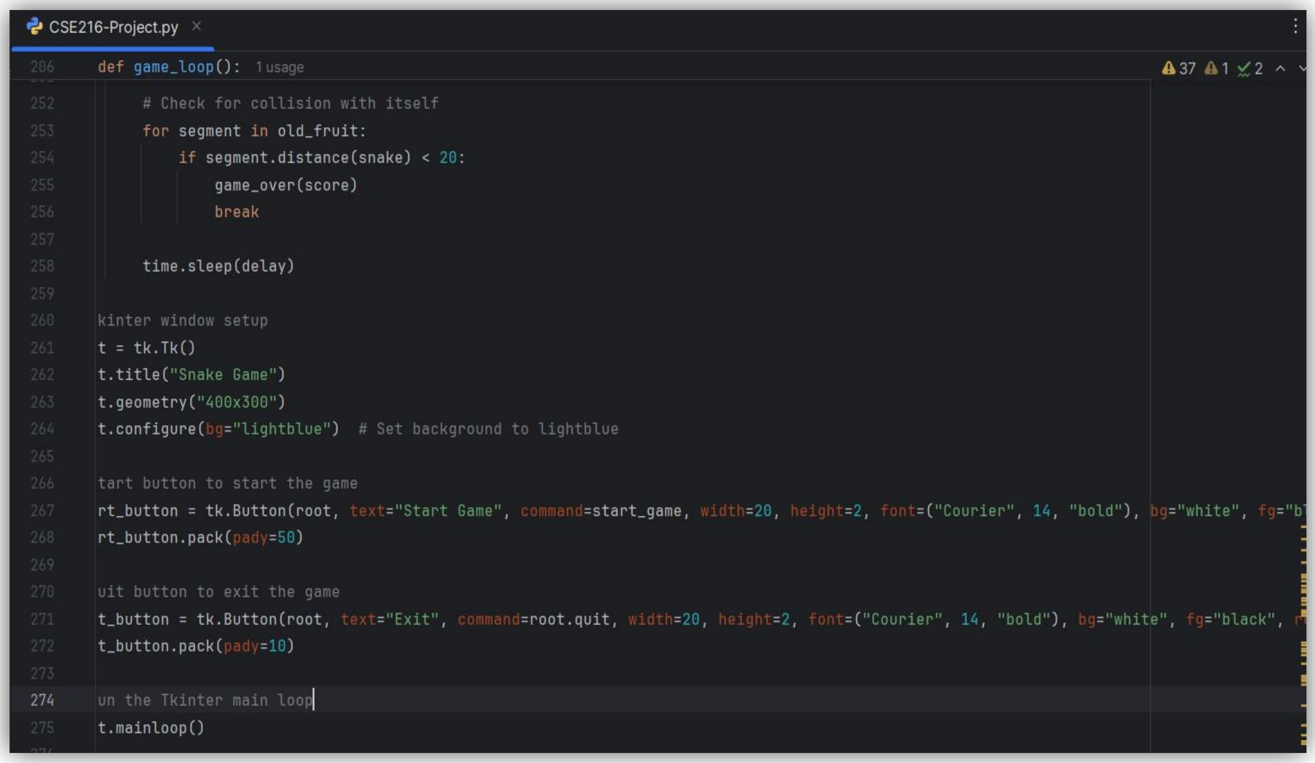
```
ame over function
 game_over(score):  2 usages
 screen.clear()
 screen.bgcolor('lightblue')  # Set background to lightblue on game over
 scoring.goto(0, 0)
 scoring.write(f"GAME OVER\nYour Score: {score}", align="center", font=("Courier", 30, "bold"))

ain game loop
 game_loop():  1 usage
 global score, delay, old_fruit, paused

 while True:
     screen.update()

     if paused:
         time.sleep(0.1)  # Pause the game
         continue

     # Snake and food collision detection
     if snake.distance(fruit) < 20:
         x = random.randint(-290,  b: 270)
         y = random.randint(-240,  b: 240)
         fruit.goto(x, y)
         score += 1
         scoring.clear()
         scoring.write(f"Score: {score}", align="center", font=("Courier", 24, "bold"))
         delay -= 0.001
```



```python
def game_loop():  1 usage
            # Add new segment to the snake's body
            new_fruit = turtle.Turtle()
            new_fruit.speed(0)
            new_fruit.shape('square')
            new_fruit.color('red')
            new_fruit.penup()
            old_fruit.append(new_fruit)

        # Move the snake's body
        for index in range(len(old_fruit) - 1, 0, -1):
            a = old_fruit[index - 1].xcor()
            b = old_fruit[index - 1].ycor()
            old_fruit[index].goto(a, b)
```



```python
def game_loop():  1 usage
            old_fruit[index].goto(a, b)

        if len(old_fruit) > 0:
            a = snake.xcor()
            b = snake.ycor()
            old_fruit[0].goto(a, b)

        snake_move()

        # Check for snake collisions with borders
        if snake.xcor() > 280 or snake.xcor() < -300 or snake.ycor() > 240 or snake.ycor() < -240:
            game_over(score)
            break

        # Check for collision with itself
        for segment in old_fruit:
            if segment.distance(snake) < 20:
                game_over(score)
                break

        time.sleep(delay)
```



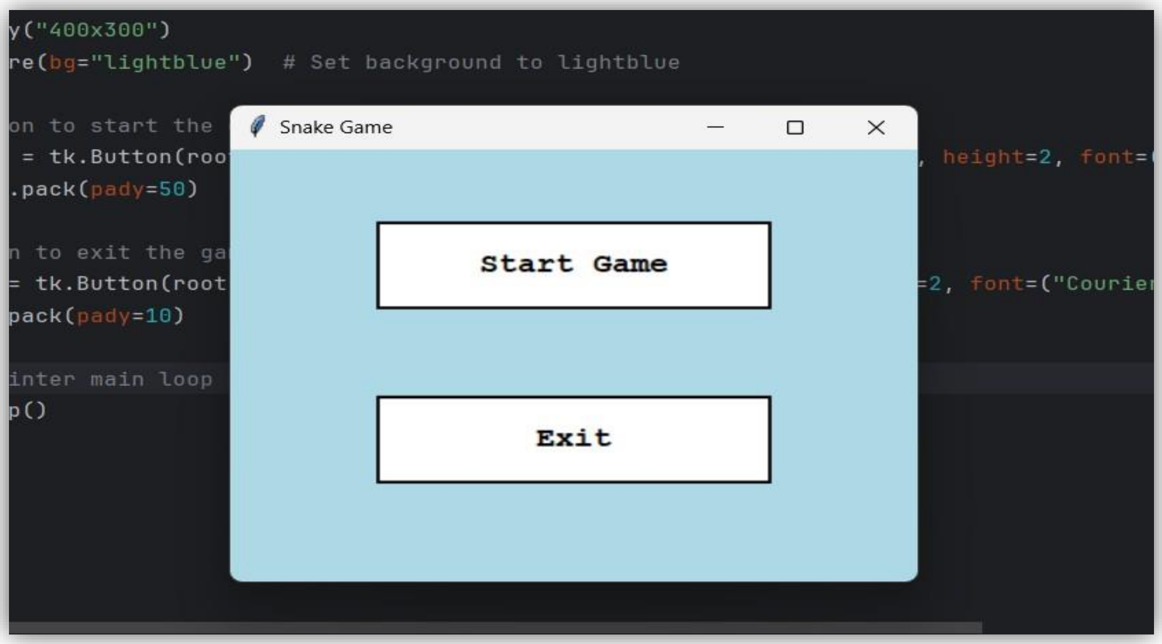
CSE216-Project.py

```
def game_loop():  1 usage
        # Check for collision with itself
        for segment in old_fruit:
            if segment.distance(snake) < 20:
                game_over(score)
                break

        time.sleep(delay)

kinter window setup
t = tk.Tk()
t.title("Snake Game")
t.geometry("400x300")
t.configure(bg="lightblue")  # Set background to lightblue

tart button to start the game
rt_button = tk.Button(root, text="Start Game", command=start_game, width=20, height=2, font=("Courier", 14, "bold"), bg="white", fg="b
rt_button.pack(pady=50)

uit button to exit the game
t_button = tk.Button(root, text="Exit", command=root.quit, width=20, height=2, font=("Courier", 14, "bold"), bg="white", fg="black", r
t_button.pack(pady=10)

un the Tkinter main loop
t.mainloop()
```

## Outputs:-

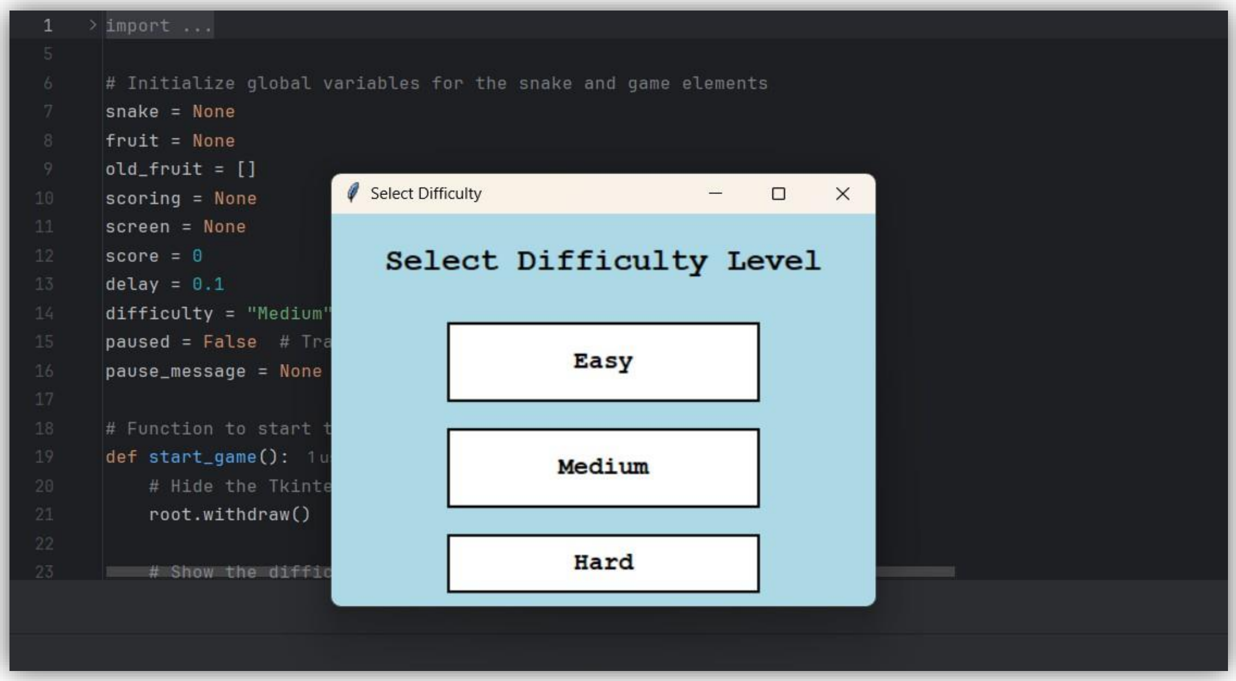
## Snake Game interface:

**1.**



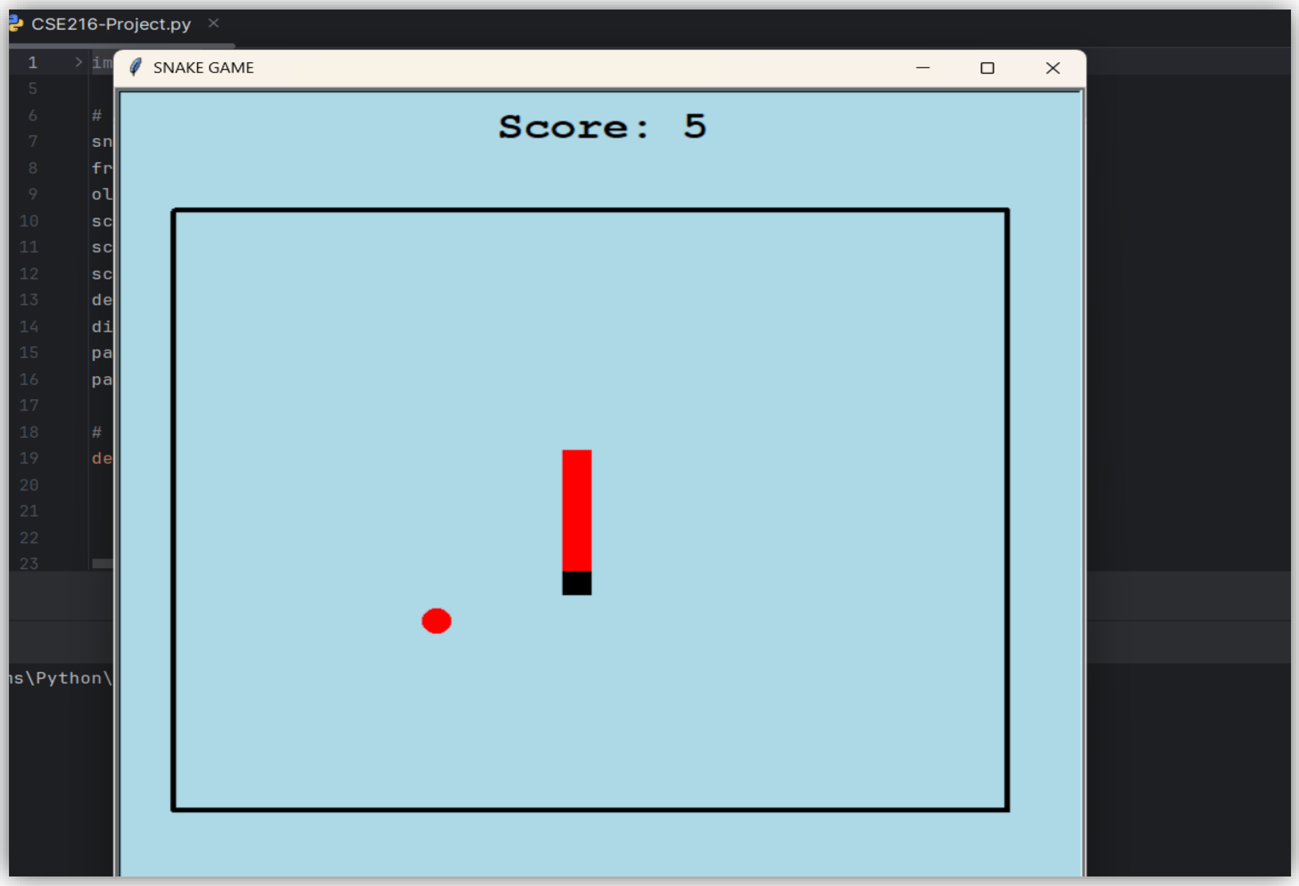
```
y("400x300")
re(bg="lightblue")  # Set background to lightblue

on to start the
 = tk.Button(roo                                  , height=2, font=(
.pack(pady=50)

n to exit the ga
= tk.Button(root                                  =2, font=("Courier
pack(pady=10)

inter main loop
p()
```

**2.**

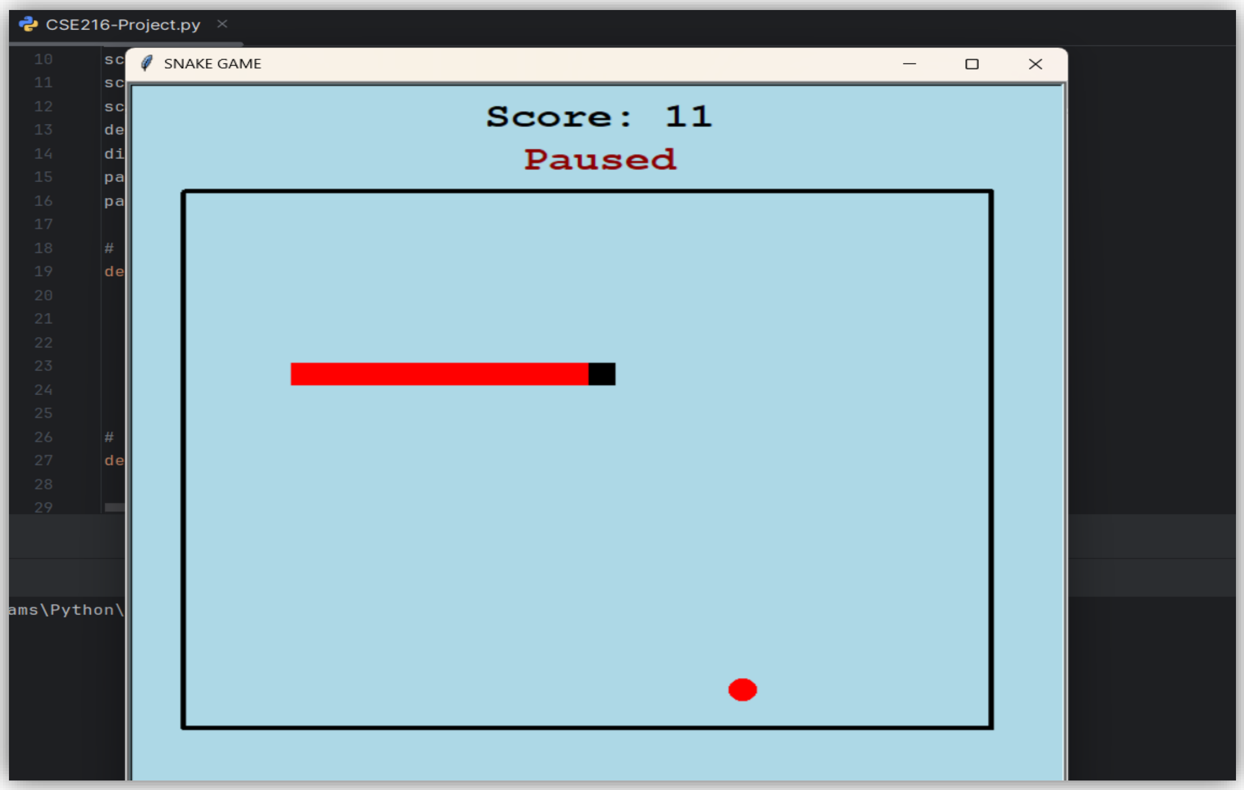


```
import ...

# Initialize global variables for the snake and game elements
snake = None
fruit = None
old_fruit = []
scoring = None
screen = None
score = 0
delay = 0.1
difficulty = "Medium"
paused = False  # Tra
pause_message = None

# Function to start t
def start_game():  1 u
    # Hide the Tkinte
    root.withdraw()

    # Show the diffic
```

**3.**

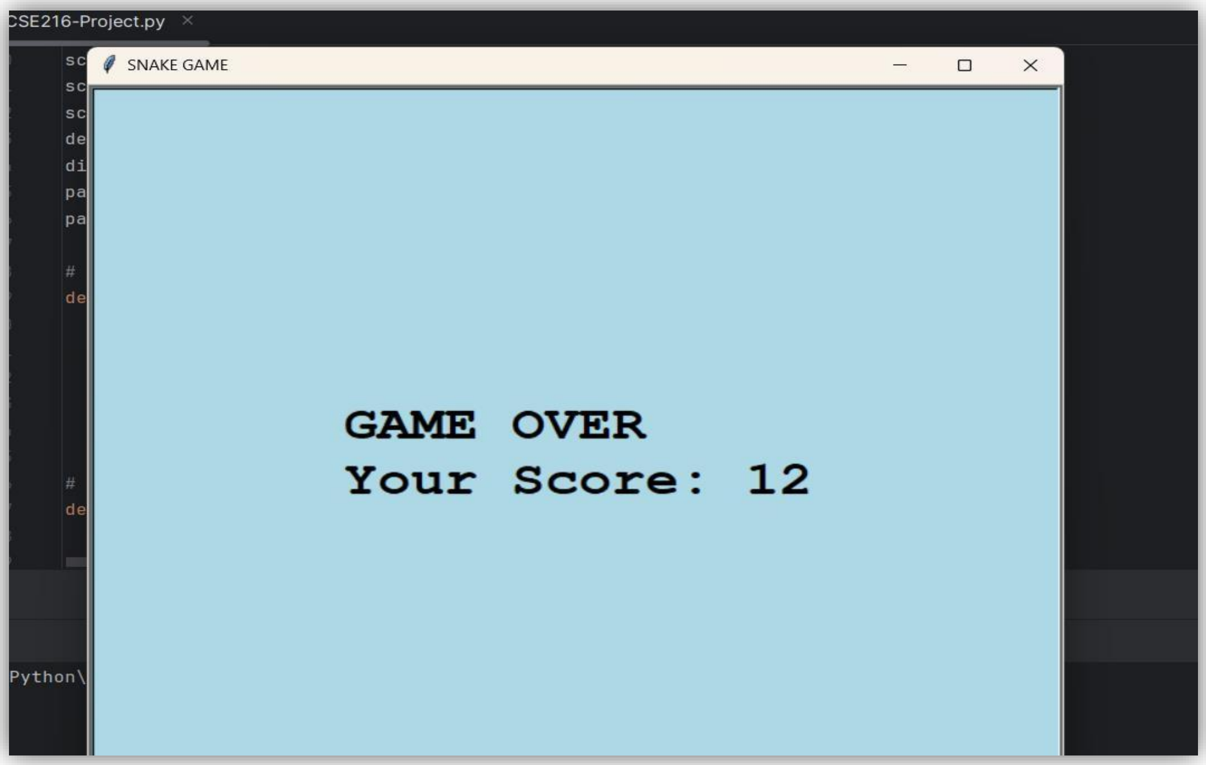


**4.**



```
Score: 11
Paused
```

**5.**



```
GAME OVER
Your Score: 12
```

# 8. Testing and Validation

Testing and validation are crucial to ensuring that the Snake Game operates correctly, providing a smooth and enjoyable user experience. Here's an explanation of the testing methods applied:

## Unit Testing

**Definition**: Unit testing involves testing individual components or functions of the software in isolation to ensure they work correctly.

**Application**:

- **Snake Movement Functions**: Each function responsible for the snake's movement (snake_go_up(), snake_go_down(), snake_go_left(), snake_go_right()) is tested to verify that the snake changes direction appropriately based on user input.
- **Collision Detection**: Functions that handle collision detection are tested to ensure they correctly identify when the snake collides with the fruit, the borders, or itself.
- **Score Update**: The function responsible for updating the score is tested to ensure the score increments correctly when the snake eats a fruit.

**Tools Used**:

- Python's unittest module is used to create and run unit tests for individual functions and components.

**System Testing**

**Definition**: System testing involves testing the entire system as a whole to ensure all components work together correctly.

**Application**:

- **Game Initialization**: The entire game initialization process, including setting up the main menu, difficulty selection, and game screen, is tested to ensure the game starts correctly and all elements are displayed properly.
- **Gameplay**: The complete gameplay is tested, including snake movement, fruit collection, score updating, collision detection, and game-over conditions, to ensure the game runs smoothly without errors.
- **Pause Functionality**: The pause and resume functionality is tested to ensure the game can be paused and resumed correctly without affecting the game state.
- **User Interface**: The user interface components, including buttons and labels, are tested to ensure they function correctly and provide the intended interactions.

**Tools Used**:

- Manual testing is conducted to verify the overall gameplay experience and user interface functionality.
- Automated system tests are written using Python scripts to simulate user interactions and verify the system's response.

**Validation**

**Definition**: Validation involves ensuring that the software meets the specified requirements and provides the intended user experience.

**Application**:

- **Requirements Review**: The game's features and functionalities are reviewed to ensure they meet the project objectives and user requirements.
- **User Feedback**: Feedback is collected from users to validate that the game provides an enjoyable and engaging experience. Any issues or suggestions for improvement are noted and addressed.
- **Performance Testing**: The game's performance is tested to ensure it runs smoothly without

significant lag or delays, providing a responsive gameplay experience.

**Tools Used**:

- User surveys and feedback forms to collect input from players.
- Performance monitoring tools to track the game's responsiveness and resource usage.

# 9. Conclusion :

Summarize the project's achievements and its potential impact. Discuss any future improvements or extensions that could be made.

**Example:** The Snake game project successfully demonstrates the use of Python for creating interactive games. It provides an enjoyable way to learn programming concepts and enhances problem-solving skills. Future improvements could include adding sound effects, different levels of difficulty, and more sophisticated graphics.

# 10. References

List any references or resources you used during the project, such as tutorials, documentation, or books.

**Example:**

- Python Official Documentation: https://docs.python.org/3/
- turtle module documentation: https://docs.python.org/3/library/turtle.html
- Stack Overflow: Various threads and discussions