

SNAKE GAME

A PROJECT REPORT

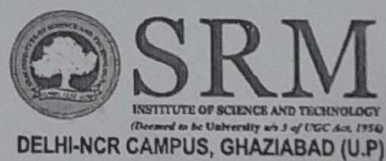
Submitted by

NISARG [RA2311003030593]
AKANSHA [RA2211003030587]
RAGINI [RA2211003030546]
DAKSH [RA2211003030595]
TANISHKA [RA2311003030589]

Under the guidance of

Ms. Neetu Bansla

(Assistant Professor, Department of Computer Science & Engineering)



SRM INSTITUTE OF SCIENCE & TECHNOLOGY, NCR CAMPUS

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

*Certified to be the bonafide record of work done by Nisarg,Ragini,Daksh,Akansha & Tansihka of 3th semester 2nd year **B.TECH** degree course in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, NCR Campus** of Department of Computer Science & Engineering in Database Management Systems Lab, during the academic year 2023-2024.*

SIGNATURE

Ms. Neetu Bansla
Assistant Professor
Computer Science & Engineering

Dr. Avneesh Vashistha
Head of the Department
Computer Science & Engineering

Acknowledgement

I would like to express my special thanks of gratitude to my teacher/mentor, Ms Neetu Bansal , for their invaluable guidance and support throughout the project. I am also grateful to my classmates and friends for their assistance in developing this project. Without their help and encouragement the completion of this project would not have been possible.

Introduction

The Snake Game is one of the most popular and well-known classic games, originating in the late 1970s and gaining popularity through mobile phones and computers. The game requires the player to guide a snake to eat food and avoid obstacles like walls or its own body. As the snake eats, it grows longer, making the game progressively more challenging.

This project involves creating a basic version of the Snake Game using Python's pygame library, which allows for the creation of 2D games and multimedia applications. The development of this project has been undertaken to learn the fundamentals of game programming, including the game loop, event handling, and collision detection.

Abstract

This project focuses on the development of a classic Snake Game using Python programming language. The game is designed to provide a fun and interactive experience for the user, where the player controls a snake that grows in length as it consumes food items. The game ends when the snake collides with the walls or its own body. The primary objective of the project is to enhance understanding of game development concepts and Python programming skills through the implementation of a simple yet effective graphical game.

Objectives

- To design and develop a simple Snake Game using Python.
-
- To implement game loops and event handling in Python.
-
- To enhance problem-solving and programming skills.
-
- To create a user-friendly interface for the game.
-
- To understand the basics of 2D game development.

Hardware Requirements

Processor: Intel Core i3 or equivalent.

RAM: 4 GB (minimum).

Storage: 1 GB of available space.

Display: Standard monitor or laptop screen with 1024x768 resolution.

Software Requirements

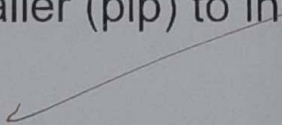
Operating System: Windows/Linux/macOS.

Programming Language: Python 3.x.

Libraries: pygame (for game development).

Code Editor: Visual Studio Code, PyCharm, or any other Python IDE.

Python Package Installer (pip) to install required libraries.



Advantages

Educational Value: Helps in learning programming and game development concepts.

Simple to Implement: The game is relatively simple and easy to develop for beginners.

Interactive: Provides a fun and engaging experience for users of all ages.

Cross-Platform: The game can run on different operating systems like Windows, Linux, and macOS.

Disadvantages

Limited Functionality: Being a simple game, it lacks advanced features or modern graphics.

Repetitive Gameplay: The game can become monotonous over time due to its simplicity.

Collision Bugs: If not implemented correctly, there may be bugs in collision detection.

No Save Feature: Once the game ends, the player's progress is lost.

Procedure

Step 1: Install Python and Pygame

Ensure Python 3.x is installed on your system.

Install the pygame library using the command:

```
pip install pygame
```

Step 2: Set Up the Game Window

Import pygame and initialize it.

Set up the game window with a specified height and width using `pygame.display.set_mode()`.

Step 3: Create the Snake and Food

Represent the snake using a list of coordinates that will grow as the snake eats food.

Randomly generate the food's position on the grid.

Step 4: Implement Movement and Controls

Define movement functions for the snake using keyboard input (up, down, left, right).

Ensure that the snake moves continuously, adjusting its position on the game board.

Step 5: Detect Collisions

Implement logic to detect if the snake collides with the walls of the game window or with its own body.

End the game if any of these collisions occur.

Step 6: Score System

Keep track of the score based on how many food items the snake eats.

Display the score on the screen using `pygame.font`.

Step 7: Game Over and Restart

Display a "Game Over" message when the game ends.

Provide an option for the user to restart the game.

Step 8: Testing and Debugging

Test the game for bugs and ensure smooth gameplay.

Make any necessary adjustments to improve performance.

1. Importing Libraries:

```
```python
import pygame
import sys
import random
import sqlite3
```
```

- ``pygame``: This library provides functions and classes for writing games and multimedia applications in Python. It handles things like drawing graphics, handling user input, and more.
- ``sys``: This module provides access to some variables used or maintained by the interpreter and to functions that interact with the interpreter.
- ``random``: This module implements pseudo-random number generators for various distributions.
- ``sqlite3``: This module provides a simple way to work with SQLite databases.

2. Initializing Pygame:

```
```python
pygame.init()
```
```

This line initializes the Pygame library. It needs to be called before any other Pygame functions are used.

3. Constants:

```
```python
WIDTH, HEIGHT = 400, 400
GRID_SIZE = 20
GRID_WIDTH = WIDTH // GRID_SIZE
GRID_HEIGHT = HEIGHT // GRID_SIZE
SNAKE_SPEED = 7
SNAKE_INITIAL_SPEED = 1
SPEED_INCREMENT = 0.1
SCORE_FONT = pygame.font.Font(None, 36)
NAME_FONT = pygame.font.Font(None, 24)
```
```

- These are constants used throughout the code to define properties of the game, like screen dimensions, grid size, snake speed, fonts, etc.

4. Colors:

```
```python
WHITE = (255, 255, 255)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
```
```

- These are color values defined using RGB format.

5. Initializing the Screen:

```
```python
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Snake Game")
```
```

- This creates a window with the specified dimensions and sets the title to "Snake Game".

6. User Input:

```
```python
user_name = input("Enter your name: ")
```
```

- This line prompts the user to enter their name and stores it in the variable `user_name`.

7. Database Initialization:

```
```python
conn = sqlite3.connect("high_scores.db")
cursor = conn.cursor()
```
```

- This establishes a connection to a SQLite database named `high_scores.db` and creates a cursor object to interact with the database.

...ing a table (if not exists):

```
```python
cursor.execute("""CREATE TABLE IF NOT EXISTS high_scores (
 id INTEGER PRIMARY KEY,
 player_name TEXT,
 score INTEGER
)""")
```
```

- This SQL command creates a table named `high_scores` with three columns: `id`, `player_name`, and `score`. The `IF NOT EXISTS` clause ensures that the table is only created if it doesn't already exist.

9. Committing Changes and Closing Database Connection:

```
```python
conn.commit()
conn.close()
```
```

- This commits any changes made to the database and closes the connection.

10. Initializing Snake, Food, Score, and Game Over Flags:

```
```python
snake = [(5, 5)]
snake_direction = (1, 0)
food = (random.randint(0, GRID_WIDTH - 1), random.randint(0, GRID_HEIGHT - 1))
score = 0
game_over = False
```
```

- `snake`: A list containing the initial position of the snake.
- `snake_direction`: A tuple representing the direction of the snake (initially moving to the right).
- `food`: A tuple representing the initial position of the food (randomly generated).
- `score`: An integer representing the player's score (starts at 0).
- `game_over`: A boolean flag to indicate if the game is over.

11. Game Loop:

```
```python
while not game_over:
```

- This is the main game loop. It continues running as long as `game\_over` is `False`.

## 12. Handling Events:

```
```python
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        game_over = True
    elif event.type == pygame.KEYDOWN:
        if not game_started and event.key == pygame.K_RETURN:
            game_started = True
        elif event.key == pygame.K_UP and snake_direction != (0, 1):
            snake_direction = (0, -1)
        elif event.key == pygame.K_DOWN and snake_direction != (0, -1):
            snake_direction = (0, 1)
        elif event.key == pygame.K_LEFT and snake_direction != (1, 0):
            snake_direction = (-1, 0)
        elif event.key == pygame.K_RIGHT and snake_direction != (-1, 0):
            snake_direction = (1, 0)
```

- This loop iterates through all the events (like keyboard inputs, mouse clicks, etc.) that have occurred since the last frame.
- If the event is of type `pygame.QUIT`, it means the user closed the window, so `game_over` is set to `True`.
- If the event is of type `pygame.KEYDOWN`, it means a key was pressed.
 - If `game_started` is `False` and the Enter key is pressed, `game_started` is set to `True`.
- If arrow keys are pressed, the snake's direction is updated based on the input, but certain conditions ensure the snake can't turn back on itself.

13. Moving the Snake:

```
```python
if game_started:
 # Move the snake
 x, y = snake[0]
 new_head = (x + snake_direction[0], y + snake_direction[1])
```
```

- If the game has started (`game_started` is `True`), the code calculates the new position for the snake's head based on the current direction.

14. Checking for Collisions:

```
```python
if new_head == food:
 snake.append(food)
 food = generate_food() # Generate new food location
 score += 1
 SNAKE_SPEED += SPEED_INCREMENT # Increase the snake speed
else:
 # Check for game over (updated condition)
 if (
 new_head[0] < 0
 or new_head[0] >= GRID_WIDTH
 or new_head[1] < 0
 or new_head[1] >= GRID_HEIGHT
 or new_head in snake[1:]
):
 game_over = True
 else:
 snake.pop()
```
```

- If the new head position matches the food position, the snake has eaten the food. The food is added to the snake, a new food position is generated, the score is increased, and the snake's speed is incremented.

- Otherwise, it checks for conditions that would result in the game ending. These include hitting the walls or colliding with itself.

15. Updating Snake Position:

```
```python
snake.insert(0, new_head)
```
```

- This line inserts the new head position at the beginning of the snake list.

16. Drawing the Game:

```
```python
screen.fill(WHITE)
for segment in snake:
 pygame.draw.rect(screen, GREEN, (segment[0] * GRID_SIZE, segment[1] * GRID_SIZE, GRID_SIZE, GRID_SIZE))
pygame.draw.rect(screen, RED, (food[0] * GRID_SIZE, food[1] * GRID_SIZE, GRID_SIZE, GRID_SIZE))
display_info()
pygame.display.update()
```
```

- This code draws the game elements on the screen. It clears the screen, then draws the snake segments and the food.
- `display_info()` is called to display the player's name and score.

17. Controlling Game Speed:

```
```python
pygame.time.Clock().tick(SNAKE_SPEED)
```
```

- This controls the game's frame rate. It ensures that the game runs at a consistent speed, regardless of the computer's processing power.

18. Saving the Score to the Database:

```
```python
conn = sqlite3.connect("high_scores.db")
cursor = conn.cursor()
cursor.execute("INSERT INTO high_scores (player_name, score) VALUES (?, ?)", (user_name, score))
conn.commit()
conn.close()
```
```

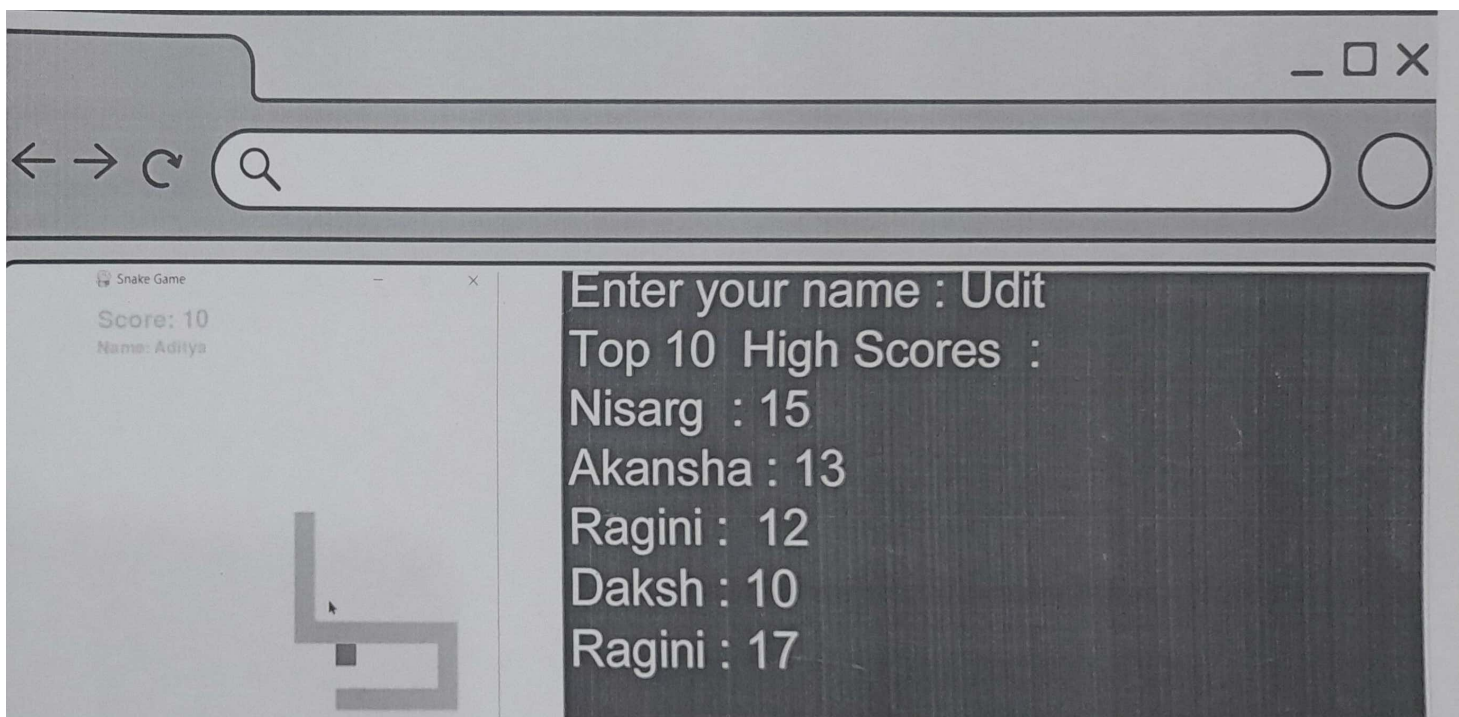
- This code connects to the database, inserts the player's name and score into the `high_scores` table, and then commits the changes before closing the connection.

19. Retrieving and Displaying Top Scores:

```
```python
conn = sqlite3.connect("high_scores.db")
cursor = conn.cursor()
cursor.execute("SELECT player_name, score FROM high_scores ORDER BY score DESC LIMIT 10")

print("Top 10 High Scores:")
for row in cursor.fetchall():
 player_name, high_score = row
 print(f"{player_name}: {high_score}")

conn.close()
```
```



Conclusion

This project provides a simple yet comprehensive example of how to build a Snake Game using Python and the pygame library. Through this project, I learned essential concepts of game development and improved my problem-solving skills. The game demonstrates the importance of understanding loops, event handling, and collision detection in a game environment.

Feel free to modify the report as per your needs!