KONGU ENGINEERING COLLEGE

(Autonomous)

PERUNDURAI ERODE – 638 060

Snake Game

A Micro Project Report

submitted by

Bharathi T

23ITR015

Chiba Roshan A

23ITR019

Elakya R

23ITR045

PYTHON PROGRAMMING AND FRAMEWORKS (22ITT32)

DEPARTMENT OF INFORMATION TECHNOLOGY

Description:

The Enhanced Snake Game project revamps the classic Snake Game by incorporating modern features like user authentication and high score tracking, aiming to enrich the gaming experience through personalization and replayability. Developed with Python and the Pygame library, this project maintains the original game's simple mechanics while adding user-focused functionalities. Players can create unique profiles, log in securely, and track their high scores, encouraging continued engagement with the game. Upon starting the game, players are welcomed with a login screen that enables registration or authentication, creating a distinct identity for each user. Successfully authenticated players proceed to the main gameplay, where they control a snake that grows with each food item consumed. The core challenge remains intact, as the player must avoid collisions with the walls and the snake's own body, with the difficulty level naturally increasing as the snake's length extends. Each player's high score is tracked and displayed during gameplay, allowing them to monitor and improve upon their previous best performance. This scoring system is supported by a local database that securely stores user credentials and high scores, adding a sense of accomplishment and motivation to the gameplay. The graphical interface and input handling are managed via Pygame, providing a smooth, interactive experience. This project serves as a demonstration of core programming skills, covering user authentication, database management, and graphical rendering in real-time, all within an entertaining context. Through this blend of classic gameplay and modern features, the Enhanced Snake Game offers a fun yet structured approach to learning foundational software development concepts. The end result is an engaging game that combines nostalgia with a personalized, challenging environment, encouraging users to improve their scores with each session.

CODING:

1.PYGAME:

```
import pygame
import random
import time
import sqlite3
pygame.init()
white = (255, 255, 255)
bright_red = (255, 0, 0)
green = (0, 255, 0)
blue = (0, 100, 255)
black = (0, 0, 0)
yellow = (255, 255, 102)
gray = (169, 169, 169)
dis_width = 700
dis_height = 500
dis = pygame.display.set_mode((dis_width, dis_height))
pygame.display.set_caption('Enhanced Snake Game')
background_image = pygame.image.load('gamestart.jpg')
background_image = pygame.transform.scale(background_image, (dis_width, dis_height))
game_background = pygame.image.load('gamebg.jpg')
game_background = pygame.transform.scale(game_background, (dis_width, dis_height))
clock = pygame.time.Clock()
font_style = pygame.font.SysFont("bahnschrift", 25)
score_font = pygame.font.SysFont("comicsansms", 35)
input_font = pygame.font.SysFont("bahnschrift", 30)
snake_block = 10
error_message = ''
error_timer = 0
def create_database():
    conn = sqlite3.connect("snake_game.db")
    cursor = conn.cursor()
    cursor.execute("""CREATE TABLE IF NOT EXISTS users (
        username TEXT PRIMARY KEY,
        password TEXT,
        high_score INTEGER DEFAULT 0)""")
    conn.commit()
    conn.close()
def register_user(username, password):
    conn = sqlite3.connect("snake_game.db")
    cursor = conn.cursor()
    cursor.execute("INSERT OR IGNORE INTO users (username, password) VALUES (?, ?)", (username, password))
```

```
    conn.commit()
    conn.close()
def validate_user(username, password):
    conn = sqlite3.connect("snake_game.db")
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?", (username, password))
    result = cursor.fetchone()
    conn.close()
    return result is not None
def get_high_score(username):
    conn = sqlite3.connect("snake_game.db")
    cursor = conn.cursor()
    cursor.execute("SELECT high_score FROM users WHERE username = ?", (username,))
    result = cursor.fetchone()
    conn.close()
    return result[0] if result else 0
def update_high_score(username, new_score):
    conn = sqlite3.connect("snake_game.db")
    cursor = conn.cursor()
    cursor.execute("UPDATE users SET high_score = ? WHERE username = ? AND high_score < ?", (new_score,
username, new_score))
    conn.commit()
    conn.close()
def display_score(score, high_score):
    score_text = score_font.render(f"Your Score: {score}  |  High Score: {high_score}", True, bright_red)
    dis.blit(score_text, [0, 0])
    pygame.draw.line(dis, black, (0, 50), (dis_width, 50), 4)
def draw_snake(snake_block, snake_list):
    for x in snake_list:
        pygame.draw.rect(dis, bright_red, [x[0], x[1], snake_block, snake_block])
def message(msg, color):
    mesg = font_style.render(msg, True, color)
    dis.blit(mesg, [dis_width / 6, dis_height / 3])
def button(msg, x, y, w, h, ic, ac, action=None):
    mouse = pygame.mouse.get_pos()
    click = pygame.mouse.get_pressed()
    if x + w > mouse[0] > x and y + h > mouse[1] > y:
        pygame.draw.rect(dis, ac, (x, y, w, h))
        if click[0] == 1 and action is not None:
            action()
    else:
        pygame.draw.rect(dis, ic, (x, y, w, h))
    text_surface = input_font.render(msg, True, black)
    dis.blit(text_surface, (x + (w - text_surface.get_width()) // 2, y + (h - text_surface.get_height()) // 2))
```

```
def welcome_screen():
    while True:
        dis.blit(background_image, (0, 0))
        button("Play", 250, 250, 200, 50, green, blue, login_screen)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()
        pygame.display.update()
        clock.tick(30)
def login_screen():
    global error_message, error_timer
    username = ''
    password = ''
    active_username = False
    active_password = False
    input_box_username = pygame.Rect(250, 200, 400, 50)
    input_box_password = pygame.Rect(250, 280, 400, 50)
    while True:
        dis.blit(background_image, (0, 0))
        username_label = input_font.render("Username:", True, yellow)
        password_label = input_font.render("Password:", True, yellow)
        dis.blit(username_label, (50, 210))
        dis.blit(password_label, (50, 290))
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if input_box_username.collidepoint(event.pos):
                    active_username = True
                    active_password = False
                elif input_box_password.collidepoint(event.pos):
                    active_password = True
                    active_username = False
                else:
                    active_username = False
                    active_password = False
            if event.type == pygame.KEYDOWN:
                if active_username:
                    if event.key == pygame.K_BACKSPACE:
                        username = username[:-1]
                    else:
                        username += event.unicode
```

```
            if active_password:
                if event.key == pygame.K_BACKSPACE:
                    password = password[:-1]
                else:
                    password += event.unicode
        pygame.draw.rect(dis, white if active_username else gray, input_box_username)
        pygame.draw.rect(dis, white if active_password else gray, input_box_password)
        dis.blit(input_font.render(username, True, black), (input_box_username.x + 5, input_box_username.y + 5))
        dis.blit(input_font.render(password, True, black), (input_box_password.x + 5, input_box_password.y + 5))
        button("Login", 250, 350, 180, 50, green, blue, lambda: attempt_login(username, password))
        button("Register", 440, 350, 180, 50, green, blue, lambda: attempt_register(username, password))
        if error_message:
            error_surface = font_style.render(error_message, True, yellow)
            dis.blit(error_surface, (dis_width / 6, 180))
            if time.time() - error_timer > 2:
                error_message = ''
        pygame.display.update()
        clock.tick(30)
def attempt_login(username, password):
    global error_message, error_timer
    if username and password:
        if validate_user(username, password):
            game_loop(username, 15)
            error_message = ''
        else:
            error_message = "Invalid login details."
            error_timer = time.time()
    else:
        error_message = "Username and password cannot be empty."
        error_timer = time.time()
def attempt_register(username, password):
    global error_message, error_timer
    if username and password:
        register_user(username, password)
        error_message = "Registration successful!"
    else:
        error_message = "Username and password cannot be empty."
    error_timer = time.time()
def game_loop(player_name, snake_speed):
    game_over = False
    game_close = False
    x1, y1 = dis_width / 2, dis_height / 2
    x1_change, y1_change = 0, 0
    snake_list = []
```

```
length_of_snake = 1
foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) * 10.0
foody = round(random.randrange(35, dis_height - snake_block) / 10.0) * 10.0
high_score = get_high_score(player_name)
direction = 'STOP'
while not game_over:
    while game_close:
        dis.blit(game_background, (0, 0))
        message("You Lost! Press Q-Quit or C-Play Again", bright_red)
        display_score(length_of_snake - 1, high_score)
        pygame.display.update()
        for event in pygame.event.get():
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_q:
                    if length_of_snake - 1 > high_score:
                        update_high_score(player_name, length_of_snake - 1)
                    game_over = True
                    game_close = False
                elif event.key == pygame.K_c:
                    if length_of_snake - 1 > high_score:
                        update_high_score(player_name, length_of_snake - 1)
                    game_loop(player_name, snake_speed)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            game_over = True
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT and direction != 'RIGHT':
                x1_change, y1_change = -snake_block, 0
                direction = 'LEFT'
            elif event.key == pygame.K_RIGHT and direction != 'LEFT':
                x1_change, y1_change = snake_block, 0
                direction = 'RIGHT'
            elif event.key == pygame.K_UP and direction != 'DOWN':
                x1_change, y1_change = 0, -snake_block
                direction = 'UP'
            elif event.key == pygame.K_DOWN and direction != 'UP':
                x1_change, y1_change = 0, snake_block
                direction = 'DOWN'
    if x1 >= dis_width or x1 < 0 or y1 >= dis_height or y1 < 35:
        game_close = True
    x1 += x1_change
    y1 += y1_change
    dis.blit(game_background, (0, 0))
    pygame.draw.rect(dis, yellow, [foodx, foody, snake_block, snake_block])
```

```
        snake_head = [x1, y1]
        snake_list.append(snake_head)
        if len(snake_list) > length_of_snake:
            del snake_list[0]
        for block in snake_list[:-1]:
            if block == snake_head:
                game_close = True
        draw_snake(snake_block, snake_list)
        display_score(length_of_snake - 1, high_score)
        pygame.draw.line(dis, black, (0, 50), (dis_width, 50), 4)
        pygame.display.update()
        if x1 == foodx and y1 == foody:
            foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) * 10.0
            foody = round(random.randrange(35, dis_height - snake_block) / 10.0) * 10.0
            length_of_snake += 1
        clock.tick(snake_speed)
    pygame.quit()
    quit()
create_database()
welcome_screen()
```

2.GAME.PY:

```
import pygame
import random
from data_db import get_high_score, update_high_score
pygame.init()
white = (255, 255, 255)
bright_red = (255, 0, 0)
green = (0, 255, 0)
blue = (0, 100, 255)
black = (0, 0, 0)
yellow = (255, 255, 102)
gray = (169, 169, 169)
dis_width = 700
dis_height = 500
dis = pygame.display.set_mode((dis_width, dis_height))
pygame.display.set_caption('Snake Game')
game_background = pygame.image.load('gamebg.jpg')
game_background = pygame.transform.scale(game_background, (dis_width, dis_height))
font_style = pygame.font.SysFont("bahnschrift", 25)
score_font = pygame.font.SysFont("comicsansms", 35)
input_font = pygame.font.SysFont("bahnschrift", 30)
```

```
clock = pygame.time.Clock()
snake_block = 10
def message(msg, color):
    mesg = font_style.render(msg, True, color)
    dis.blit(mesg, [dis_width / 6, dis_height / 3])
def display_score(score, high_score):
    score_text = score_font.render(f"Your Score: {score}  |  High Score: {high_score}", True, bright_red)
    dis.blit(score_text, [0, 0])
    pygame.draw.line(dis, black, (0, 50), (dis_width, 50), 4)
def draw_snake(snake_block, snake_list):
    for x in snake_list:
        pygame.draw.rect(dis, bright_red, [x[0], x[1], snake_block, snake_block])
def game_loop(player_name, snake_speed):
    game_over = False
    game_close = False
    x1 = dis_width / 2
    y1 = dis_height / 2
    x1_change = 0
    y1_change = 0
    snake_list = []
    length_of_snake = 1
    foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) * 10.0
    foody = round(random.randrange(50, dis_height - snake_block) / 10.0) * 10.0
    high_score = get_high_score(player_name)
    while not game_over:
        while game_close:
            dis.blit(game_background, (0, 0))
            message("You Lost! Press Q-Quit or C-Play Again", bright_red)
            display_score(length_of_snake - 1, high_score)
            pygame.display.update()
            for event in pygame.event.get():
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        if length_of_snake - 1 > high_score:
                            update_high_score(player_name, length_of_snake - 1)
                        game_over = True
                        game_close = False
                    elif event.key == pygame.K_c:
                        if length_of_snake - 1 > high_score:
                            update_high_score(player_name, length_of_snake - 1)
                        game_loop(player_name, snake_speed)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                game_over = True
```

```
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LEFT and x1_change == 0:
                    x1_change = -snake_block
                    y1_change = 0
                elif event.key == pygame.K_RIGHT and x1_change == 0:
                    x1_change = snake_block
                    y1_change = 0
                elif event.key == pygame.K_UP and y1_change == 0:
                    y1_change = -snake_block
                    x1_change = 0
                elif event.key == pygame.K_DOWN and y1_change == 0:
                    y1_change = snake_block
                    x1_change = 0
        if x1 >= dis_width or x1 < 0 or y1 >= dis_height or y1 < 50:
            game_close = True
        x1 += x1_change
        y1 += y1_change
        dis.blit(game_background, (0, 0))
        pygame.draw.rect(dis, yellow, [foodx, foody, snake_block, snake_block])
        snake_head = [x1, y1]
        snake_list.append(snake_head)
        if len(snake_list) > length_of_snake:
            del snake_list[0]
        for block in snake_list[:-1]:
            if block == snake_head:
                game_close = True
        draw_snake(snake_block, snake_list)
        display_score(length_of_snake - 1, high_score)
        pygame.display.update()
        if x1 == foodx and y1 == foody:
            foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) * 10.0
            foody = round(random.randrange(50, dis_height - snake_block) / 10.0) * 10.0
            length_of_snake += 1
        clock.tick(snake_speed)
    pygame.quit()
    quit()
```

3.WELCOME.PY:

```
import pygame
import time
from auth import attempt_login, attempt_register
from game import game_loop
from data_db import get_high_score, create_database, register_user
```

```
pygame.init()
white = (255, 255, 255)
bright_red = (255, 0, 0)
green = (0, 255, 0)
blue = (0, 100, 255)
black = (0, 0, 0)
yellow = (255, 255, 102)
gray = (169, 169, 169)
dis_width = 700
dis_height = 500
dis = pygame.display.set_mode((dis_width, dis_height))
pygame.display.set_caption('Enhanced Snake Game')
background_image = pygame.image.load('gamestart.jpg')
background_image = pygame.transform.scale(background_image, (dis_width, dis_height))
game_background = pygame.image.load('gamebg.jpg')
game_background = pygame.transform.scale(game_background, (dis_width, dis_height))
font_style = pygame.font.SysFont("bahnschrift", 25)
score_font = pygame.font.SysFont("comicsansms", 35)
input_font = pygame.font.SysFont("bahnschrift", 30)
clock = pygame.time.Clock()
error_message = ''
error_timer = 0
def render_error_message(msg, color):
    mesg = font_style.render(msg, True, color)
    text_width = mesg.get_width()
    dis.blit(mesg, [dis_width - text_width - 20, 20])
def button(msg, x, y, w, h, ic, ac, action=None):
    mouse = pygame.mouse.get_pos()
    click = pygame.mouse.get_pressed()
    if x + w > mouse[0] > x and y + h > mouse[1] > y:
        pygame.draw.rect(dis, ac, (x, y, w, h))
        if click[0] == 1 and action is not None:
            action()
    else:
        pygame.draw.rect(dis, ic, (x, y, w, h))
    text_surface = input_font.render(msg, True, black)
    dis.blit(text_surface, (x + (w - text_surface.get_width()) // 2, y + (h - text_surface.get_height()) // 2))
def welcome_screen():
    global error_message, error_timer
    while True:
        dis.blit(background_image, (0, 0))
        button("Play", 250, 250, 200, 50, green, blue, login_screen)
        if error_message:
            render_error_message(error_message, bright_red if 'Invalid' in error_message else blue)
```

```
            if time.time() - error_timer > 2:
                error_message = ''
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()
        pygame.display.update()
        clock.tick(30)
def login_screen():
    global error_message, error_timer
    username = ''
    password = ''
    active_username = False
    active_password = False
    input_box_username = pygame.Rect(250, 200, 400, 50)
    input_box_password = pygame.Rect(250, 280, 400, 50)
    while True:
        dis.blit(background_image, (0, 0))
        username_label = input_font.render("Username:", True, yellow)
        password_label = input_font.render("Password:", True, yellow)
        dis.blit(username_label, (50, 210))
        dis.blit(password_label, (50, 290))
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if input_box_username.collidepoint(event.pos):
                    active_username = True
                    active_password = False
                elif input_box_password.collidepoint(event.pos):
                    active_password = True
                    active_username = False
                else:
                    active_username = False
                    active_password = False
            if event.type == pygame.KEYDOWN:
                if active_username:
                    if event.key == pygame.K_BACKSPACE:
                        username = username[:-1]
                    else:
                        username += event.unicode
                if active_password:
                    if event.key == pygame.K_BACKSPACE:
```

```
                password = password[:-1]
            else:
                password += event.unicode
        pygame.draw.rect(dis, white if active_username else gray, input_box_username)
        pygame.draw.rect(dis, white if active_password else gray, input_box_password)
        dis.blit(input_font.render(username, True, black), (input_box_username.x + 5, input_box_username.y + 5))
        dis.blit(input_font.render(password, True, black), (input_box_password.x + 5, input_box_password.y + 5))
        button("Login", 250, 350, 180, 50, green, blue, lambda: attempt_login_action(username, password))
        button("Register", 440, 350, 180, 50, green, blue, lambda: attempt_register_action(username, password))
        if error_message:
            render_error_message(error_message, bright_red if 'Invalid' in error_message else blue)
            if time.time() - error_timer > 2:
                error_message = ''
        pygame.display.update()
        clock.tick(30)
def attempt_login_action(username, password):
    global error_message, error_timer
    if not username or not password:
        error_message = "Username and password cannot be empty!"
        error_timer = time.time()
    elif attempt_login(username, password):
        game_loop(username, 15)
    else:
        error_message = "Invalid login details."
        error_timer = time.time()
def attempt_register_action(username, password):
    global error_message, error_timer
    if not username or not password:
        error_message = "Username and password cannot be empty!"
        error_timer = time.time()
    else:
        register_user(username, password)
        error_message = "Registration successful!"
        error_timer = time.time()
welcome_screen()
```

4.DATA.DB_PY:

```
import sqlite3
def create_database():
    connection = sqlite3.connect("snake_game.db")
    cursor = connection.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS users (
```

```
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT UNIQUE,
                password TEXT,
                high_score INTEGER DEFAULT 0
            )'''')
    connection.commit()
    connection.close()
def register_user(username, password):
    try:
        connection = sqlite3.connect("snake_game.db")
        cursor = connection.cursor()
        cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, password))
        connection.commit()
    except sqlite3.IntegrityError:
        print("Username already exists.")
    finally:
        connection.close()
def validate_user(username, password):
    connection = sqlite3.connect("snake_game.db")
    cursor = connection.cursor()
    cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?", (username, password))
    user = cursor.fetchone()
    connection.close()
    return user is not None
def get_high_score(username):
    connection = sqlite3.connect("snake_game.db")
    cursor = connection.cursor()
    cursor.execute("SELECT high_score FROM users WHERE username = ?", (username,))
    high_score = cursor.fetchone()
    connection.close()
    return high_score[0] if high_score else 0
def update_high_score(username, new_score):
    connection = sqlite3.connect("snake_game.db")
    cursor = connection.cursor()
  cursor.execute("UPDATE users SET high_score = ? WHERE username = ? AND high_score < ?", (new_score,
username, new_score))
    connection.commit()
    connection.close()
create_database()
```

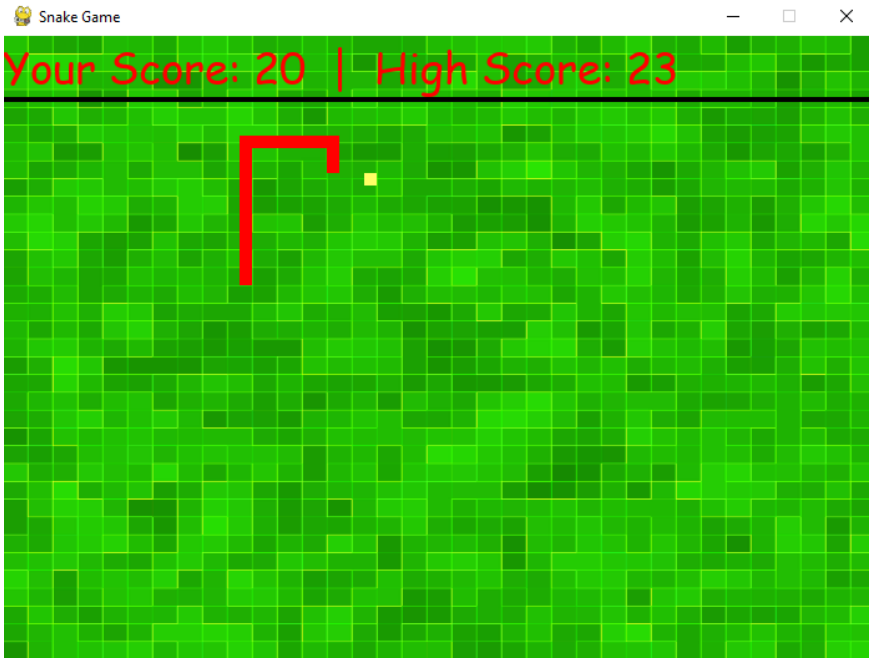
5.AUTH.PY:

```
import sqlite3
```

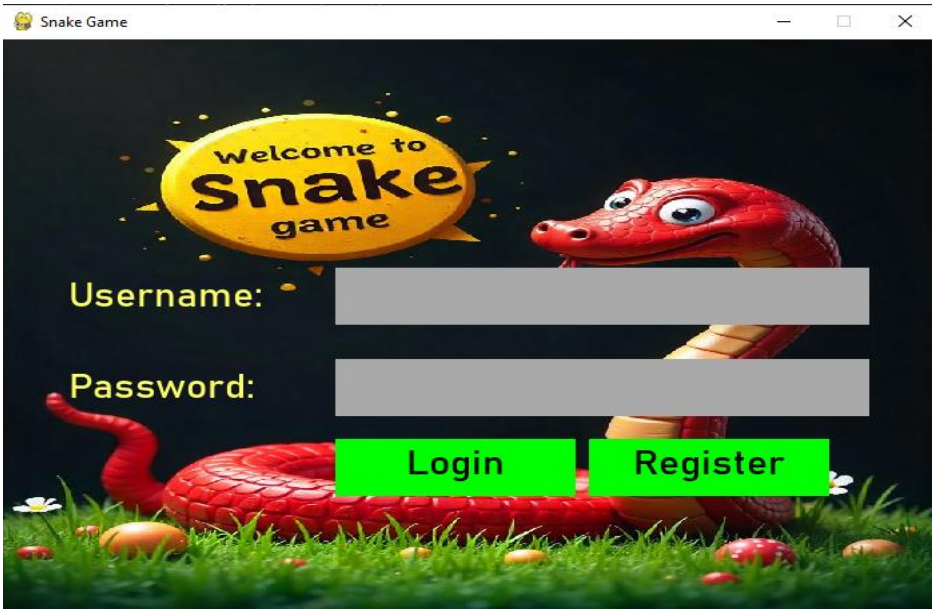
```
def create_connection():
    conn = sqlite3.connect('snake_game.db')
    return conn
def create_table():
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS users (
                      id INTEGER PRIMARY KEY AUTOINCREMENT,
                      username TEXT UNIQUE NOT NULL,
                      password TEXT NOT NULL)''')
    conn.commit()
    conn.close()
def register_user(username, password):
    conn = create_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, password))
        conn.commit()
        return True
    except sqlite3.IntegrityError:
        return False
    finally:
        conn.close()
def validate_user(username, password):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT password FROM users WHERE username = ?", (username,))
    stored_password = cursor.fetchone()
    conn.close()
    if stored_password:
        return stored_password[0] == password
    return False
def attempt_login(username, password):
    return validate_user(username, password)
def attempt_register(username, password):
    return register_user(username, password)
def get_high_score():
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT score FROM highscores ORDER BY score DESC LIMIT 1")
    score = cursor.fetchone()
    conn.close()
    return score[0] if score else 0
def save_high_score(username, score):
```

```
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("INSERT INTO highscores (username, score) VALUES (?, ?)", (username, score))
    conn.commit()
    conn.close()
create_table()
```

OUTPUT:


Snake Game
Your Score: 20 | High Score: 23


Snake Game
Your Score: 20 | High Score: 23
You Lost! Press Q-Quit or C-Play Again


Snake Game
Welcome to
Snake
game
Username:
Password:
Login
Register