GAMING HUB

A MINI PROJECT REPORT

Submitted for the partial fulfillment of the requirement for the degree of

BACHELOR OF SCIENCE
IN
COMPUTER SCIENCE

Submitted by

MOHAMED MUSHRAF F A
2213181058111

Under the Guidance of

Dr. K. TAJUDIN, M.S., M.Phil., Ph.D., M.Sc.,

P.G. DEPARTMENT OF COMPUTER SCIENCE

THE NEW COLLEGE
(Autonomous)

CHENNAI – 600 014

MARCH – 2025

YA ALLAH
INCREASE MY KNOWLEDGE

BONAFIDE CERTIFICATE

This is to certify that the mini project work entitled "**GAMING HUB"** is a Bonafide record of the project work done by **MOHAMED MUSHRAF F A** with Register Number **2213181058111** is partial fulfilment for the award of Bachelor of Science in Computer Science during the academic year **2024 - 2025**.

Dr. K. TAJUDIN
Project Guide

Dr. P. Hakkim Divan Mydeen
Head of the Department

Submitted for the project Viva Voice Examination in THE NEW COLLEGE, Chennai

held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

Date:

Place:

ACKNOWLEDGEMENT

1. First of all, I thank the Almighty for blessing me with his abundance grace in completing my project successfully.

2. I express my sincere gratitude to the Principal, Dr. M. Asrar Sheriff M.Sc., M.Phil., Ph.D., for permitting me to do the project with fullest spirit

3. I am pleased to acknowledge my great thanks to Dr. P. Hakkim Divan Mydeen M.Sc.,M.Phil.,Ph.D., Head of the Department of Computer Science, for his useful guidance and encouragement to complete this project.

4. My sincere thanks to my guide Dr. K. Tajudin M.S., M.Phil., Ph.D., M.Sc., Assistant Professor, Department of Computer Science, for his valuable guidance and encouragement for finishing this project successfully.

5. I thank all the Staff Members of Computer Science for their co- operation in completing this project.

6. I acknowledge my heartfelt thanks to my parents for their encouragement, social and economic support for completing this project successfully.

MOHAMED MUSHRAF F A

TABLE OF CONTENT

INTRODUCTION

This project is **consisting** of two python-based 2D games, shooting game and car game Developed using the pygame library, integrated with a SQL server database for player data Management. The shooting game is a space-themed shooter where the player controls a spaceship to Destroy incoming enemies, featuring dynamic scoring, sound effects, and a replay system. The car Game is a lane-based racing game where the layer avoids incoming vehicles, with increasing difficulty and collision detection. Both games utilize a database (GameDB) to store and retrieve player information, including username and password, high scores, enabling features like player authentication, high score tracking, and demonstrate core game development concepts such as sprite management, collision detection, and event handling, while showcasing database integration for data persistence and user management.

MODULE:

- user

USER MODULE:

LOGIC PAGE:

- Players must **log in** before starting the game. The system ensures that each player has a unique username
- User must enter a **valid username** that exists in the database to log in. If an incorrect username is entered, the system **denies access**.
- After logging in, players are **directed to the game selection screen**.
- After login, the **player's username** is displayed on the selection screen. This ensures that each player knows they are logged into the correct account

SINGUP PAGE:

- During **signup**, the system ensures that **usernames are unique**. If a username already exists, the user is prompted to **choose a different one**.

GAME SELCTION PAGE:

- The screen is designed to be **simple and easy to navigate**. Players can quickly select their desired game without confusion.
- Players can **choose from different games**, such as:
    - **Car Racing Game**
    - **Shooting Game**

- After selecting a game, the screen **seamlessly transitions** to the chosen game. If the player exits a game, they return to the **selection screen** instead of restarting the application

SHOOTING GAME PAGE:

- The player **controls a shooter** (a spaceship, soldier, or any other character). **Arrow keys or specific buttons** are used to move the character. The **shooting action** is triggered using a key (e.g., spacebar or mouse click).
- The player must **shoot and destroy** them to gain points. Enemies **appear at random locations** on the screen. Their movement speed and behavior increase as the game progresses.
- The **score increases** each time the player shoots an enemy. The **high score is saved** to track the player's best performance. Scores are displayed on the screen during the game.

- If an enemy reaches the player's position or **collides with them**, the game ends. If the player's bullets hit an enemy, it **gets destroyed**.
- The game includes **animated backgrounds** (e.g., space, battlefield). **Sound effects** for shooting, explosions, and enemy movements enhance the experience. Background music may be added for immersion.
- When exiting the game, the screen smoothly **transitions back to the game selection screen**. If the player sets a new high score, it is **saved in the database**.

CAR GAME PAGE:

- The player controls a **car that moves on a multi-lane road**. **Arrow keys (Left/Right)** are used to change lanes. The car remains on the road, avoiding obstacles and other vehicles.
- Random **enemy vehicles spawn** on the road. The player's goal is to **avoid crashing into other vehicles**. The difficulty increases as speed and the number of enemy vehicles rise
- The player earns **points** for avoiding obstacles and surviving longer. Score increases as the game progresses. The **high score is saved** and displayed on the screen.
- The game features **moving lane markers** to create a sense of speed. The background consists of **roads, lane dividers, and edge markers**. Vehicles have smooth movement animations.
- If the player's car **collides** with another vehicle, the game ends. A **crash animation** is displayed upon collision. The game prompts the player to **restart** or **return to the selection screen**.

ABSTRACTION

The user interacts with the system through a seamless interface that provides access to multiple games and functionalities. The system requires the user to register through a signup process, creating a unique identity with a username and password, ensuring secure access. Upon successful login, the user is directed to a game selection screen, where they can choose between the Shooting Game and the Car Game. The user's progress, such as high scores, is tracked and stored in the database, enabling personalized experiences. In the Shooting Game, the user controls a spaceship to eliminate enemies, while in the Car Game, the user navigates a car through traffic, avoiding collisions. Each game offers increasing difficulty levels, testing the user's skills and reflexes. Real-time feedback is provided through score updates and game-over prompts. The system handles user inputs like arrow keys for movement and key presses for restarting the game after a loss. Additionally, the user's performance is recorded, allowing high scores to be retrieved and updated automatically. The interface is designed to be intuitive, making the gaming experience smooth and engaging. The user benefits from a personalized profile, game progress tracking, and score management. Overall, the system offers a dynamic gaming experience where the user can seamlessly switch between games, challenge themselves, and improve their performance over time.

Characteristics of User:

- **Unique Identity:** Each user has a unique username for personalized game tracking.
- **Secure Access:** The user's credentials are protected with a login system.
- **Game Selection:** Ability to choose between different games from a central selection screen

- **Progress Tracking:** High scores and achievements are stored in the database.
- **Input Control:** Controls the game using arrow keys and other key inputs.
- **Real-Time Feedback:** Receives instant score updates and game-over messages.
- **Challenge-Oriented:** Competes with personal high scores to improve performance.
- **Personalized Experience:** Tailored game sessions based on the user's login profile.
- **Multi-Game Access:** Can access and switch between different games.
- **Skill Development:** Enhances reflexes and coordination through game challenges.
- **Game Restart Options:** Ability to restart the game instantly after a loss.

SYSTEM ANALYSIS

Existing System:

The existing system was manual, requiring users to manage game access progress without automation. There was no dedicated login/signup process to track individual users, making personalized game experiences impossible.

Game selection had to be done manually, with no unified interface to switch between games. Each game ran separately, lacking integration with user data and game history. Tracking scores and performance required manual recording, which was prone to errors. There was no security mechanism to protect user data or game records, leaving the system vulnerable to unauthorized access and data loss.

Demerits of System:

- No personalized experience due to the absence of user login/signup 9functionality.
- Game selection was disorganized, with no central interface to manage multiple games.
- Players couldn’t save scores or track performance across different game sessions.
- Lack of security measures led to unauthorized access and data manipulation.
- Manual handling of game data increased the chances of errors and data loss.
- Integration between different games was non-existent, requiring separate execution.
- Time-consuming process to switch between games and maintain progress.

Proposed System:

The proposed system is a fully automated gaming platform integrating login, Game selection, and two engaging games: Shooting Game and Car Game Developed with Pygame, and SSMS, the system ensures a best game play in Smooth and secure user experience. The login/signup feature allows users to Create personalized accounts, enabling data tracking and personalized game sessions. The game selection screen acts as a central hub where users can choose between the Shooting Game and Car Game seamlessly. Each game is connected to the system, ensuring smooth transitions and secure data handling. User performance is recorded and displayed instantly, enhancing the overall gaming experience. Security measures protect user data from unauthorized access, making the system reliable and user-friendly.

Merits of Proposed System:

- Personalized gaming experience through user login/signup functionality.
- Centralized game selection screen for smooth navigation between games.
- Secure data handling ensures user data protection.
- Automated tracking of game scores and user performance.
- Reduced human effort by integrating multiple games into one system.
- Enhanced user experience with instant access to games and performance records.
- Cost-effective solution with minimal maintenance compared to manual systems.

SYSTEM CONFIGURATIONS

HARDWARE REQUIREMENT:

- Minimum 8 GB RAM.
- Minimum 120 GB free space on the Hard Disk
- Minimum processor i3 to i7 (i5 recommended).
- Standard LCD/LED Monitor.

SOFTWARE REQUIREMENT:

- Code Editor: Visual Studio Code.
- Operating System: Minimum Windows 8 and above.
- Front-End: Pygame.
- Database: Microsoft SQL Server with SQL Server Management Studio (SSMS).

SYSTEM DESIGN

The most creative and challenging phase of system development is System Design. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility study. The design process goes through logical and physical stages of development.

Authentication, and high score retrieval. The system aims to manage user data effectively while enhancing the gaming experience by providing personalized data handling. The first step was determining the data flow between the Python program and the SQL Server database. The input data from users, such as username and password, needed a structured storage mechanism, leading to the creation of the “playerdata” table in the database. The operational phases involve program construction, testing, and ensuring smooth communication between the front end and the database.

System design applies techniques and principles to define a structure that supports secure data handling. It offers a solution to efficiently register users, authenticate credentials, and track high scores. This phase provides clarity on data flow, system architecture, and procedural design to ensure smooth integration and functionality.

OUTPUT DESIGN:

The output design focuses on delivering relevant feedback to users after each action, such as registration, login, and high score retrieval. The system displays messages like "User registered successfully!" and "Login successful!".

about the outcome of their actions. In the case of retrieving high scores, the system returns the stored value, enhancing personalization. Proper feedback ensures users understand the system's responses, making it user-friendly and improving engagement.

INPUT DESIGN:

The input design handles user-provided data, such as usernames and passwords, ensuring they are securely stored in the "playerdata" table. The system accepts inputs through prompts and processes them before storing or verifying against the database. Input validation is applied to prevent invalid entries, ensuring data integrity.

The input design plays a vital role in ensuring that data entered by users is captured accurately and efficiently. In this system, user inputs are collected through interactive forms, where users provide details such as usernames and passwords during registration and login. Each input field is validated to ensure the correctness of data, preventing invalid entries. The system prompts users with appropriate messages in case of errors, guiding them to enter the required information correctly. This structured input design not only simplifies data entry but also enhances user experience by minimizing errors and ensuring smooth navigation.

LOGICAL DESIGN:

The logical design maps out the relationships between various components of the system — the Python code, SQL Server database, and the flow of user data. Each user record consists of a username, password, and car game high score, forming a structured entity in the database.

SOFTWARE DESCRIPTION

PYGAME:

Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries built on top of the Simple Direct Media Layer (SDL). Pygame is highly popular for developing simple 2D games and interactive applications.

CHARACTERISTICS OF PYGAME:

- **Simplicity**: Pygame is easy to learn and use, making it ideal for beginners.
- **Cross-Platform:** Supports multiple operating systems like Windows, macOS, and Linux.
- **Modular Structure:** Provides various modules for handling graphics, sounds, input events, and more.
- **Event-Driven:** Uses an event-driven model to handle user inputs and game events efficiently.
- **Community Support:** Has a large community with plenty of tutorials and resources.

ADVANTAGES OF PYGAME:

- Open-source and free to use.
- Open-source and free to use.
- Simplifies game development with built-in support for handling graphics, sounds, and input events.
- Lightweight and runs efficiently on low-end hardware.

USES OF PYGAME:

- Developing 2D games like platformers, puzzle games, and shooters.
- Creating simulations and visualizations for educational purposes.
- Building interactive prototypes for game ideas and concepts.

PYODBC:

Pyodbc is a Python library that allows you to connect to databases using Open Database Connectivity (ODBC) drivers. It's commonly used to interact with SQL Server, but it can also connect to other databases like MySQL, PostgreSQL, and Oracle, provided the appropriate ODBC drivers are installed.

KEY FEATURES OF PYODBC:

- Supports SQL queries and stored procedures.
- Works with multiple databases through ODBC drivers.
- Provides connection pooling for efficient resource use.
- Offers straightforward handling of database cursors to execute queries and fetch results.

SQL SERVER MANAGEMENT STUDIO (SSMS): (DATABASE)

SQL Server Management Studio (SSMS) is a software application used for configuring, managing, and administering all components within Microsoft SQL Server. It combines a rich set of graphical tools and script editors that enhance the productivity of database developers and administrators.

ADVANTAGES OD SSMS:

- Provides a user-friendly interface for managing SQL databases.
- Supports query execution and debugging, making it easier to work with SQL scripts.
- Offers tools for performance monitoring and database tuning.
- Integrates seamlessly with Microsoft Azure for cloud-based database management.

USES OF SSMS:

- Writing and executing SQL queries to interact with databases.
- Managing database objects such as tables, views, stored procedures, and triggers.
- Performing backup and restoration operations on databases.
- Monitoring and optimizing database performance.

SYSTEM IMPLEMENTATION & TESTING

WHITE BOX TESTING:

White Box testing is a test case design method that uses the control structure of the procedural design and derives the test cases. In this method, tests are made with knowledge about the internal workings of the system to ensure that each component operates according to the specification. Logical paths are tested to ensure all conditions, loops, and functions are executed as intended.

White Box Testing would involve analyzing the flow of functions like “Connect to dB, register user, login user, and get high score” to ensure proper handling of database connections, query execution, error handling, and connection closure. Each internal function is tested to ensure that:

BLACK BOX TESTING:

Black Box Testing focuses on testing the functional requirements of the system without considering its internal code structure. It evaluates whether the software behaves as expected when given specific inputs and checks the accuracy of the outputs.

Black box test involves:

- Testing user registration by providing various usernames and passwords to ensure proper insertion into the database.
- Testing the login function by entering correct and incorrect credentials, ensuring that valid users log in successfully while invalid attempts display appropriate error messages.

ER DIAGRAM

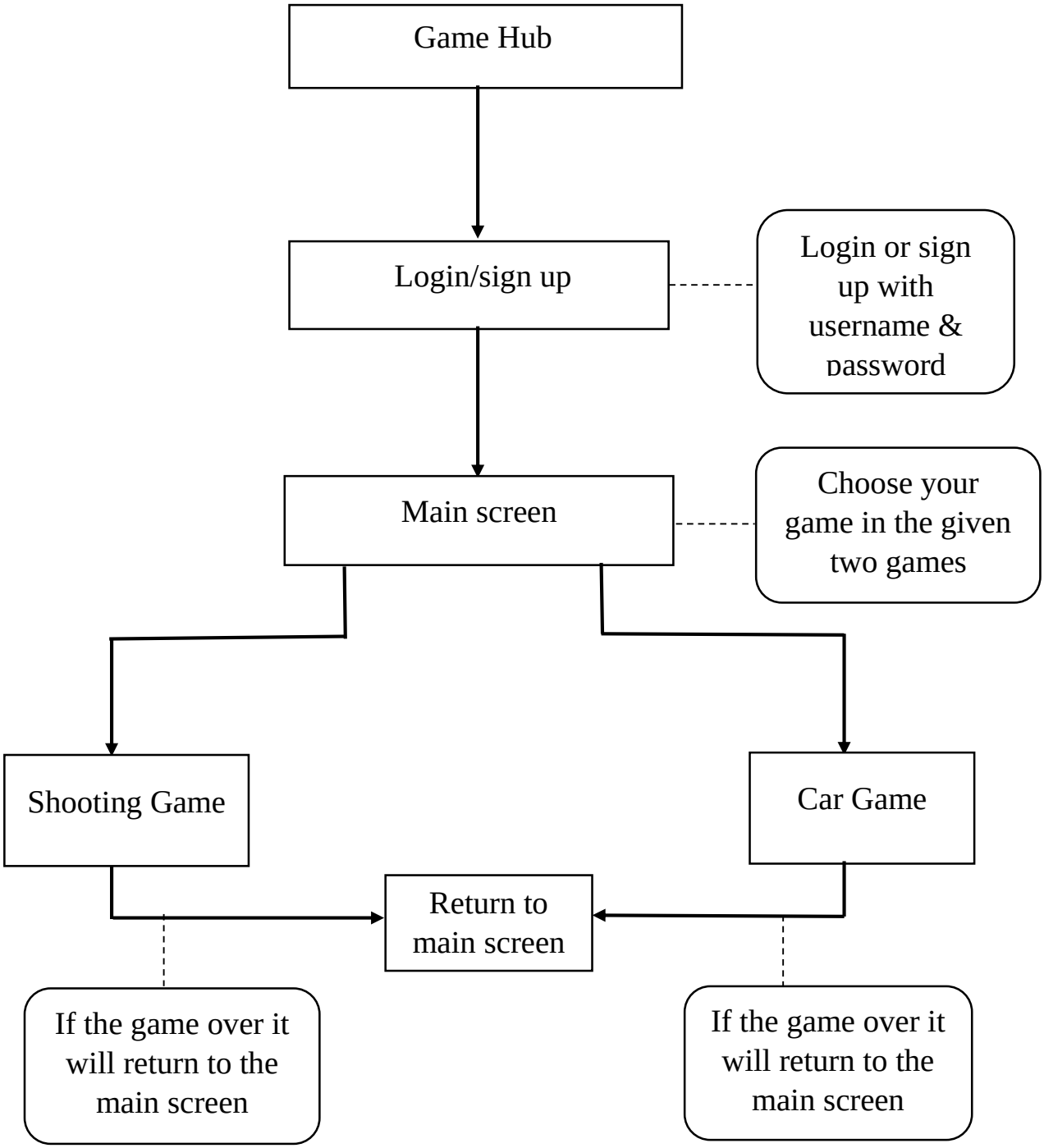
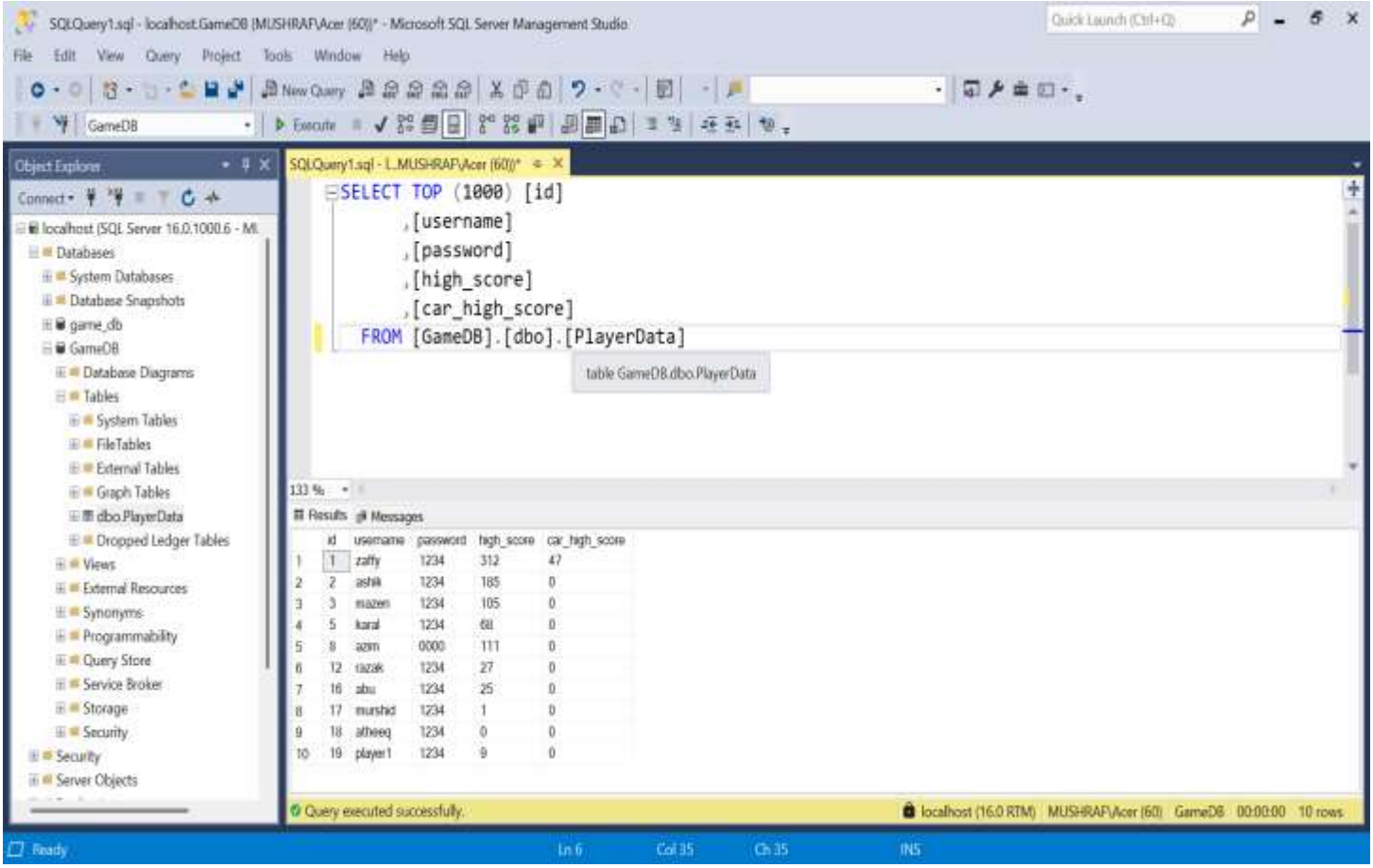

Game Hub
Login/sign up
Login or sign up with username & password
Main screen
Choose your game in the given two games
Shooting Game
Car Game
Return to main screen
If the game over it will return to the main screen
If the game over it will return to the main screen

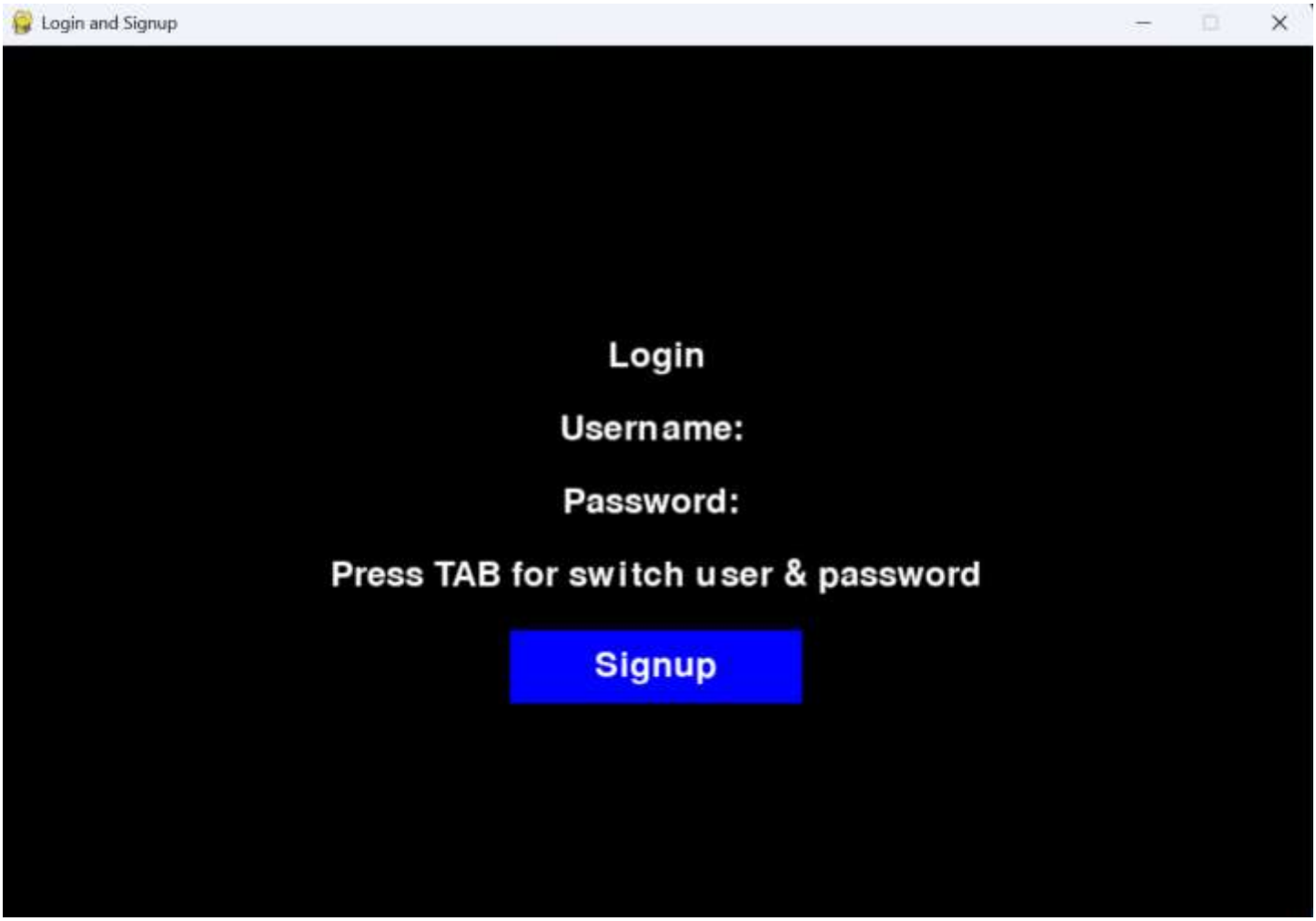
TABLE STRUCTURE

DATABASE TABLE:

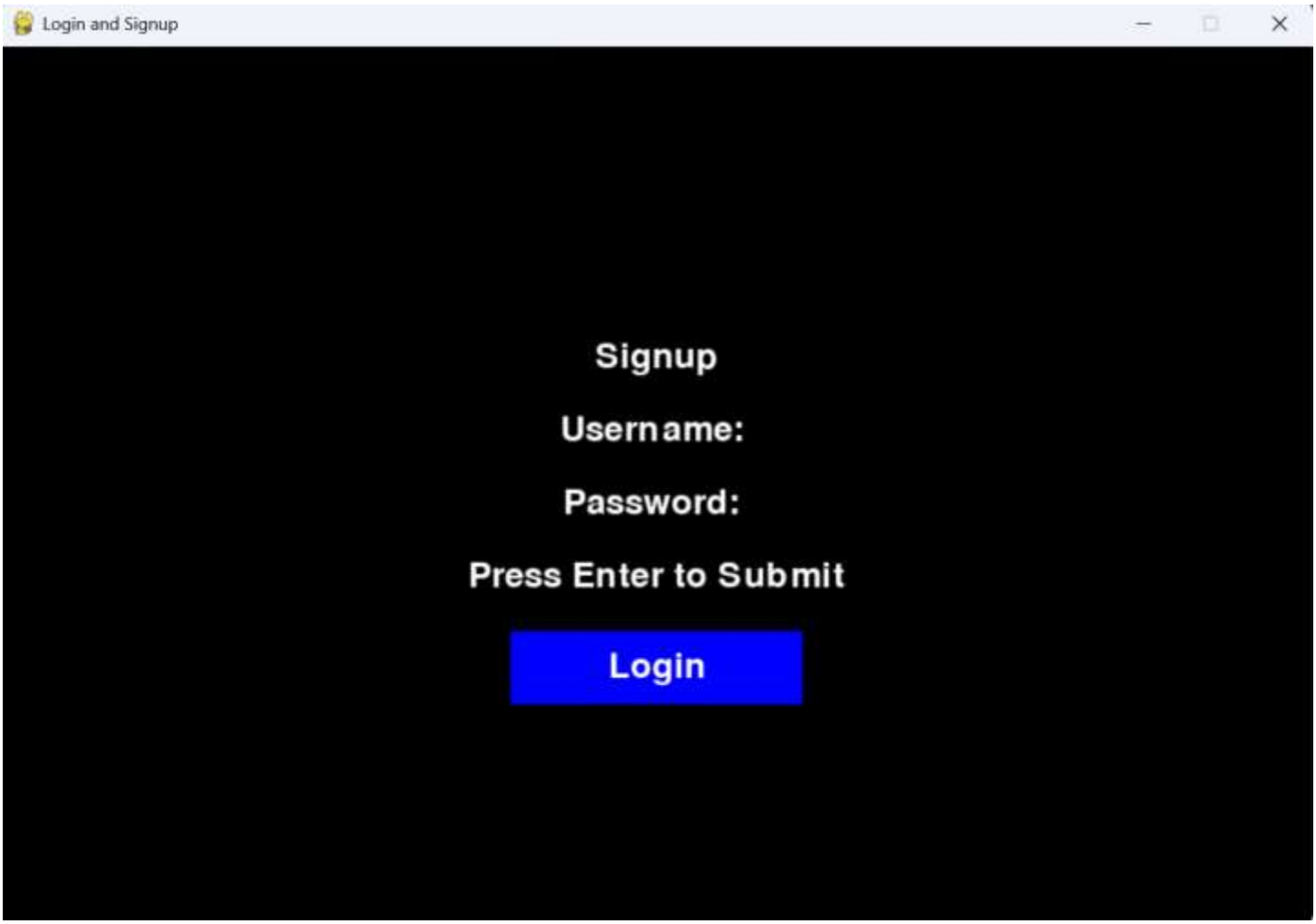


SYETEM LAYOUT

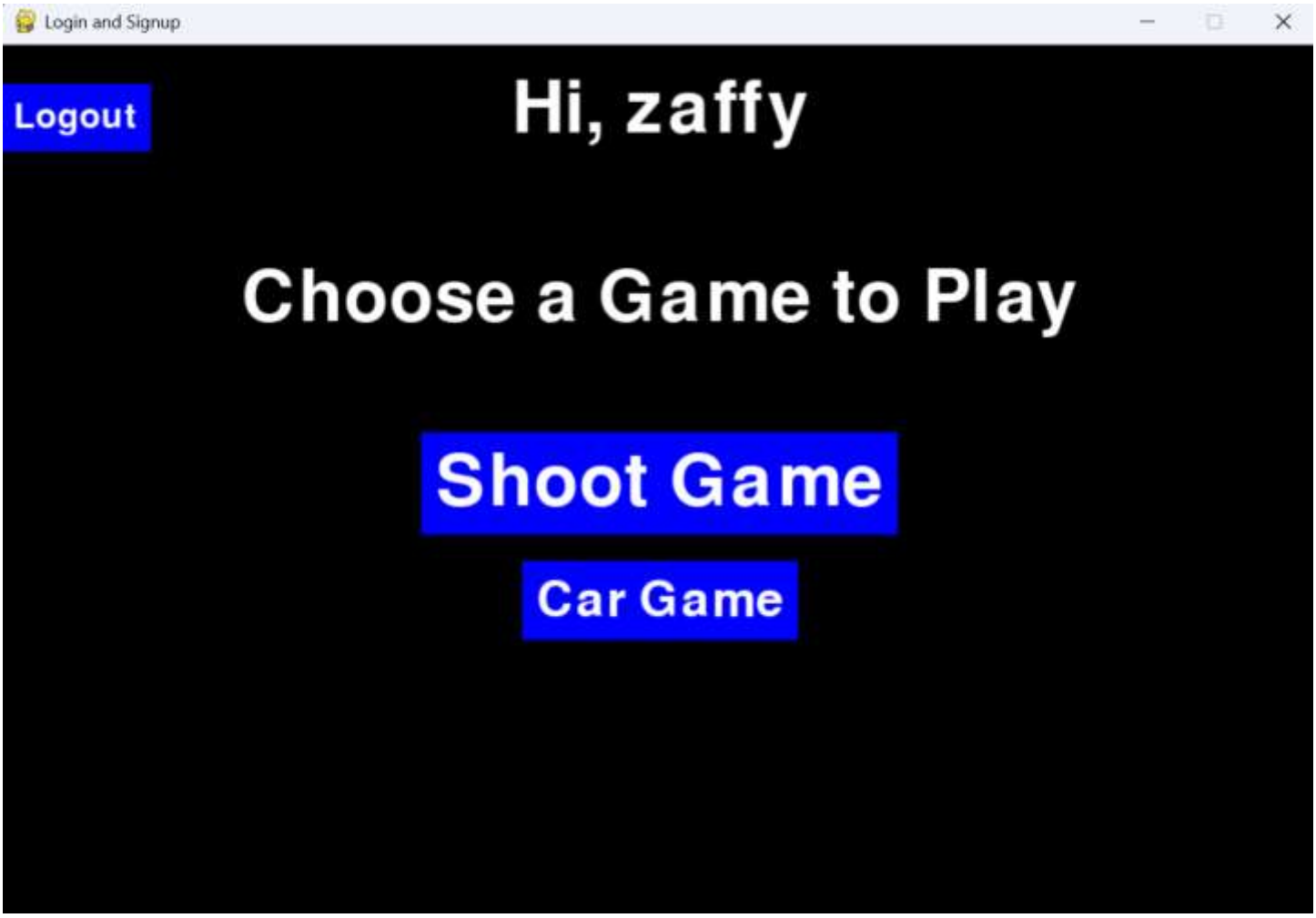
LOGIN SCREEN:



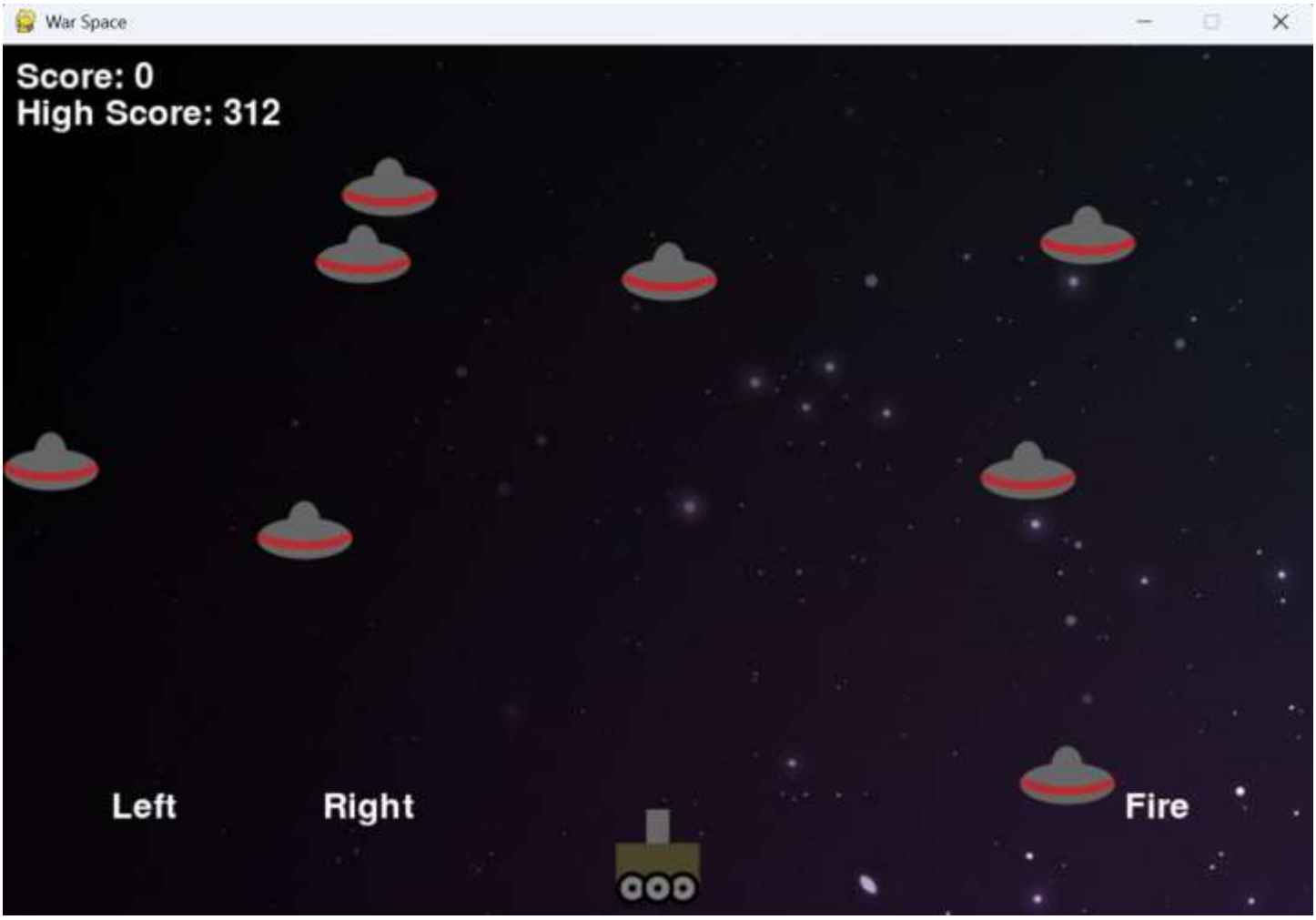
SIGNUP SCREEN:



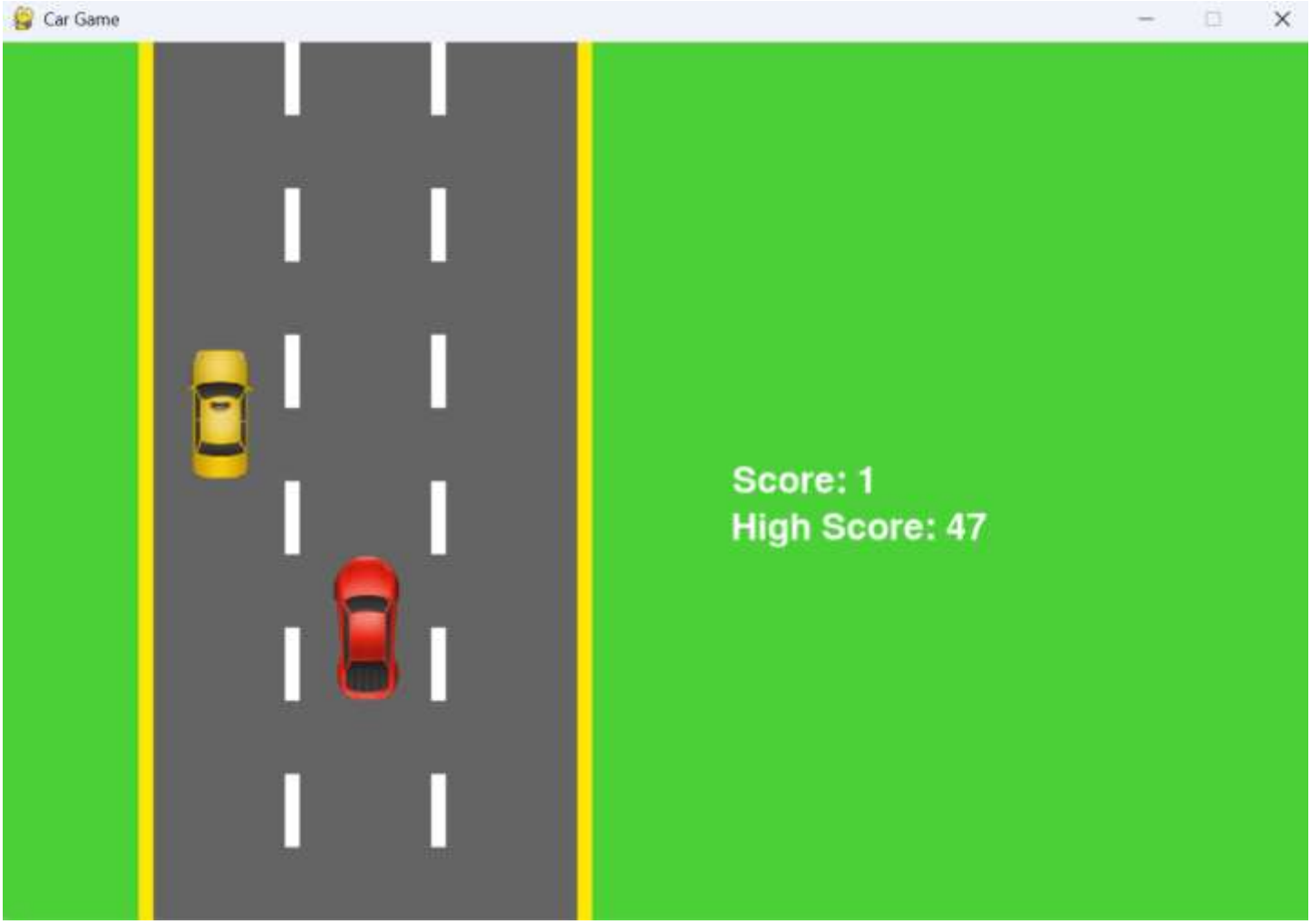
MAIN SCREEN:


Login and Signup
Logout
Hi, zaffy
Choose a Game to Play
Shoot Game
Car Game

SHOOTING GAME SCREEN:



CAR GAME SCREEN:



SOURCE CODE

LOGIN.PY:

```
import pygame
import pyodbc
import time

# Initialize Pygame
pygame.init()

# Screen dimensions
SCREEN_WIDTH = 900
SCREEN_HEIGHT = 600

# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
BLUE = (0, 0, 255)

# Game settings
FPS = 60

# Font settings
font = pygame.font.Font(None, 36)

# Database connection using Windows Authentication
def connect_to_db():
    try:
        conn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};'
                              'SERVER=localhost;'
                              'DATABASE=GameDB;'
```

```
                                        'Trusted_Connection=yes;')
        return conn
    except Exception as e:
        print(f"Error connecting to database: {e}")
        return None

# Save player data (username, password, high_score) to the database
def save_player_data(username, password, high_score):
    conn = connect_to_db()
    cursor = conn.cursor()

    # Check if the username already exists
    cursor.execute("SELECT * FROM PlayerData WHERE username = ?", username)
    existing_player = cursor.fetchone()

    if existing_player:
        cursor.close()
        conn.close()
        return False  # Username already exists
    else:
        # Insert new player data
        cursor.execute("""
            INSERT INTO PlayerData (username, password, high_score)
            VALUES (?, ?, ?)
        """, username, password, high_score)

    conn.commit()
    cursor.close()
    conn.close()
    return True  # Successfully saved new player data

# Retrieve player data for login (verify username and password)
def authenticate_user(username, password):
    conn = connect_to_db()
    cursor = conn.cursor()
```

```
    cursor.execute("SELECT high_score FROM PlayerData WHERE username = ?
AND password = ?", username, password)
    result = cursor.fetchone()
    cursor.close()
    conn.close()

    if result:
        global high_score
        high_score = result[0]  # Fetch the high score
        return True  # Successful login
    return False  # Failed login

# Function to display a message on the screen
def show_message(message, color, y_offset=0):
    text = font.render(message, True, color)
    screen.blit(text, (SCREEN_WIDTH // 2 - text.get_width() // 2, SCREEN_HEIGHT //
2 + y_offset))

# Create the Pygame window
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
pygame.display.set_caption("Login and Signup")

# Function to create a button on the screen
def create_button(message, x, y, width, height, color, text_color):
    pygame.draw.rect(screen, color, pygame.Rect(x, y, width, height))
    text = font.render(message, True, text_color)
    screen.blit(text, (x + width // 2 - text.get_width() // 2, y + height //
2 - text.get_height() // 2))

# Function to check if a button is clicked
def is_button_clicked(x, y, width, height, mouse_pos):
    return x < mouse_pos[0] < x + width and y < mouse_pos[1] < y + height
```

```
# Function to handle switching between login and signup screens
def game_loop():
    logged_in = False
    signup = False
    username = ""
    password = ""
    high_score = 0  # Temporary high score for the session
    done = False
    clock = pygame.time.Clock()

    # Variables to track focus (use tab to switch focus)
    focus_field = "username"  # Start with the username field

    # Message tracking variables
    message = ""  # Variable to store the current message to display
    message_color = WHITE  # Default color of message
    last_message_time = 0  # Time when the message was shown
    message_duration = 3000  # Duration for the message to be displayed
    (in    milliseconds)

    while not done:
        screen.fill(BLACK)
        current_time = pygame.time.get_ticks()  # Get the current time in
milliseconds

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                done = True
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_BACKSPACE:
                    if focus_field == "username" and username:
                        username = username[:-1]
                    elif focus_field == "password" and password:
                        password = password[:-1]
                elif event.key == pygame.K_RETURN:  # Submit on Enter key
```

```
            if signup:

                # Signup process

                if username and password:
                    # Attempt to save player data
                    if save_player_data(username, password, high_score):
                        message = "Account Created! Switching to login..."
                        message_color = GREEN
                        last_message_time = current_time  # Track when the message
                        was shown
                        pygame.display.flip()
                        time.sleep(1)  # Wait for 1 second before switching to login
                        signup = False  # Switch to the login screen after
successful signup
                        username = ""  # Clear fields for login
                        password = ""  # Clear fields for login
                    else:
                        message = "Username already exists! Please choose a
                        different one."
                        message_color = RED
                        last_message_time = current_time
                        time.sleep(1)
                else:
                    message = "Please fill in both fields"
                    message_color = RED
                    last_message_time = current_time
                    time.sleep(1)
            else:
                # Login process
                if username and password:
                    if authenticate_user(username, password):
                        logged_in = True
                        message = "Login Successful!"
                        message_color = GREEN
                        last_message_time = current_time
```

```
                    time.sleep(2)
                    done = True
                else:

                    message = "Invalid credentials. Please try again."
                    message_color = RED
                    last_message_time = current_time

                    time.sleep(1)
            else:
                message = "Please enter both username and password"
                message_color = RED
                last_message_time = current_time
                time.sleep(1)

        elif event.key == pygame.K_TAB:  # Switch between username and
        password using Tab
            if focus_field == "username":
                focus_field = "password"
            else:
                focus_field = "username"
        else:
            # Handle regular typing
            if focus_field == "username" and len(username) < 20 and
event.unicode.isalnum():
                username += event.unicode
            elif focus_field == "password" and len(password) < 20:
                password += event.unicode

    mouse_pos = pygame.mouse.get_pos()

    # Display the login/signup screen
    if not signup:
        show_message("Login", WHITE, -100)
        show_message(f"Username: {username}", WHITE, -50)
        show_message(f"Password: {'*' * len(password)}", WHITE, 0)
```

```
            if not username and not password:
                show_message("Press TAB for switch user & password", WHITE, 50)

            # Create a "Signup" button on the login screen

            create_button("Signup", 350, 400, 200, 50, BLUE, WHITE)
            if is_button_clicked(350, 400, 200, 50, mouse_pos) and event.type ==
pygame.MOUSEBUTTONDOWN:
                signup = True  # Switch to the signup screen
                username = ""  # Clear fields for new data

                password = ""

            # Show error message for login only if it was set recently
            if current_time - last_message_time <= message_duration and message
!= "":
                show_message(message, message_color, 170)

        else:
            show_message("Signup", WHITE, -100)
            show_message(f"Username: {username}", WHITE, -50)
            show_message(f"Password: {'*' * len(password)}", WHITE, 0)
            show_message("Press Enter to Submit", WHITE, 50)

            # Create a "Login" button on the signup screen
            create_button("Login", 350, 400, 200, 50, BLUE, WHITE)
            if is_button_clicked(350, 400, 200, 50, mouse_pos) and event.type ==
pygame.MOUSEBUTTONDOWN:
                signup = False  # Switch to the login screen
                username = ""  # Clear fields for new data
                password = ""

            # Show error message for signup only if it was set recently
            if current_time - last_message_time <= message_duration and message
 != "":
                show_message(message, message_color, 170)
```

```
            pygame.display.flip()
            clock.tick(FPS)

        from play_screen import play_screen

        play_screen(username, high_score)
    game_loop()
```

PLAY_SCREEN.PY:

```
import pygame
import random
import time
import pyodbc


# Initialize Pygame
pygame.init()

# You can change the size (36) as needed
font = pygame.font.Font(None, 36)

# Declaration
username = "" # Initialize username globally

# Screen dimensions
SCREEN_WIDTH = 900
SCREEN_HEIGHT = 600

# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
BLUE = (0, 0, 255)


# Database connection using Windows Authentication
def connect_to_db():
```

```
    try:

        conn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};'
                              'SERVER=localhost;'

                              'DATABASE=GameDB;'
                              'Trusted_Connection=yes;')

        return conn
    except Exception as e:
        print(f"Error connecting to database: {e}")
        return None

# Button class
class Button:
    def __init__(self, text, pos, font, bg_color=BLUE, text_color=WHITE):
        self.x, self.y = pos
        self.font = pygame.font.Font(None, font)
        self.bg_color = bg_color
        self.text_color = text_color
        self.change_text(text)

    def change_text(self, text):
        self.text = self.font.render(text, True, self.text_color)
        self.size = self.text.get_size()
        self.surface = pygame.Surface((self.size[0] + 20, self.size[1] + 20))
        self.surface.fill(self.bg_color)
        self.surface.blit(self.text, (10, 10))
        self.rect = self.surface.get_rect(center=(self.x, self.y))

    def show(self, screen):
        screen.blit(self.surface, self.rect.topleft)

    def click(self, event):
        x, y = pygame.mouse.get_pos()
        if event.type == pygame.MOUSEBUTTONDOWN:
            if pygame.mouse.get_pressed()[0]:
```

```
            if self.rect.collidepoint(x, y):
                return True
        return False



# Create the Pygame window
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))

# Create the "Play" button and welcome message
play_button = Button("Shoot Game", (SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2),
font=74)

# Logout button at the top left
logout_button = Button("Logout", (50, 50), font=36)

# Function to handle logout action
def logout():
    global username, game_active, game_over
    # Reset game state and user session
    username = ""
    game_active = False
    game_over = False
    # Switch to login screen
    if username:
        from login import game_loop
        game_loop()


def play_screen(username, high_score):

    # Fetch the high score from the database when the game ends and the
    player returns to the main menu
    if username:
        # Reload the high score from the database
        conn = connect_to_db()
        cursor = conn.cursor()
        cursor.execute("SELECT high_score FROM PlayerData WHERE username = ?"
```

```
,  username)
        result = cursor.fetchone()
        if result:
            high_score = result[0]  # Set the high score from the database

        cursor.close()
        conn.close()

    done = False

    while not done:
        screen.fill(BLACK)
        font = pygame.font.Font(None, 74)

        # Display player name at the top (just below the top of the screen)
        if username:  # Check if username is not empty
            player_name_text = font.render(f"Hi, {username}", True, WHITE)
            screen.blit(player_name_text, (SCREEN_WIDTH //
2 - player_name_text.get_width() // 2, 20))  # Position at top

        # Display game message
        text = font.render("Choose a Game to Play", True, WHITE)
        screen.blit(text, (SCREEN_WIDTH // 2 - text.get_width() // 2, SCREEN_HEIGHT
// 2 - text.get_height() - 100))

        # Create buttons for game selection
        car_game_button = Button("Car Game", (SCREEN_WIDTH // 2,
SCREEN_HEIGHT // 2 + 80), font=50)

        car_game_button.show(screen)
        play_button.show(screen)
        logout_button.show(screen)  # Display the logout button

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                done = True
            elif event.type == pygame.MOUSEBUTTONDOWN:
```

```
            if play_button.click(event):
                from shoot_game import start1_game
                start1_game(username, high_score)  # Start the game when
 Play is clicked
                done = True  # Exit play screen and start game loop

            elif car_game_button.click(event):
                from car_game import start_car_game  # Import start_car_game from
the car game
                start_car_game(username, high_score)  # Start the car game with
the current username
                done = True

            elif logout_button.click(event):  # Handle logout click
                game_active = False
                game_over = False
                logged_in = False  # Logout the current player
                username = ""
                password = ""
                from login import game_loop
                game_loop()  # Call the game loop to show the login screen again

        pygame.display.flip()
```

SHOOT_GAME.PY:

```
import pygame
import random
import time
import pyodbc

#def start_game(username, high_score):
# Initialize Pygame
pygame.init()

# Declaration
```

```
username = ''
high_score = ''

# Screen dimensions
SCREEN_WIDTH = 900

SCREEN_HEIGHT = 600

# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
BLUE = (0, 0, 255)

# Game settings
FPS = 60
PLAYER_SPEED = 5
ENEMY_SPEED = 2

# Database connection using Windows Authentication
def connect_to_db():
    try:
        conn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};'
                              'SERVER=localhost;'
                              'DATABASE=GameDB;'
                              'Trusted_Connection=yes;')
        return conn
    except Exception as e:
        print(f"Error connecting to database: {e}")
        return None

# Save player data (username, password, high_score) to the database
def save_player_data(username, password, high_score):
    conn = connect_to_db()
    cursor = conn.cursor()
```

```
    # Check if the username already exists
    cursor.execute("SELECT * FROM PlayerData WHERE username = ?", username)
    existing_player = cursor.fetchone()

    if existing_player:

        cursor.close()
        conn.close()
        return False  # Username already exists
    else:
        # Insert new player data
        cursor.execute("""
            INSERT INTO PlayerData (username, password, high_score)
            VALUES (?, ?, ?)
        """, username, password, high_score)

    conn.commit()
    cursor.close()
    conn.close()
    return True  # Successfully saved new player data

# Retrieve player data for login (verify username and password)
def authenticate_user(username, password):
    conn = connect_to_db()
    cursor = conn.cursor()
    cursor.execute("SELECT high_score FROM PlayerData WHERE username = ?
AND password = ?", username, password)
    result = cursor.fetchone()
    cursor.close()
    conn.close()

    if result:
        global high_score
        high_score = result[0]  # Fetch the high score
        return True  # Successful login
    return False  # Failed login
```

```
# Update high score in the database
def update_high_score(username, high_score):
    conn = connect_to_db()
    cursor = conn.cursor()


    # Check the current high score in the database for the given username


    cursor.execute("SELECT high_score FROM PlayerData WHERE username = ?"
, username)
    result = cursor.fetchone()

    if result:
        current_high_score = result[0]
        if score > current_high_score:
            # Update the high score if the current one is higher


            cursor.execute("UPDATE PlayerData SET high_score = ? WHERE username
=?", high_score, username)
            conn.commit()

    cursor.close()
    conn.close()

# Load assets
BACKGROUND_IMAGE = pygame.image.load("assets/background3.jpg")
PLAYER_IMAGE = pygame.image.load("assets/player.png")
ENEMY_IMAGE = pygame.image.load("assets/enemy.png")
BULLET_IMAGE = pygame.image.load("assets/bullet1.png")
EXPLOSION_IMAGE = pygame.image.load("assets/explosion.png")
MUSIC = pygame.mixer.Sound("assets/music1.wav")
BULLET_SOUND = pygame.mixer.Sound("assets/bullet1.wav")
EXPLOSION_SOUND = pygame.mixer.Sound("assets/explosion.wav")

# Game variables
```

```
score = 0
done = False
clock = pygame.time.Clock()
game_active = True  # Start the game directly
game_over = False

replay_cost = 100  # Initial replay cost

# Screen setup

screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))

pygame.display.set_caption("War Space")

# Player class
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = PLAYER_IMAGE
        self.rect = self.image.get_rect()
        self.rect.centerx = SCREEN_WIDTH // 2
        self.rect.bottom = SCREEN_HEIGHT – 10

        self.speed_x = 0

    def update(self):
        self.speed_x = 0
        keys = pygame.key.get_pressed()
        if keys[pygame.K_LEFT] or keys[pygame.K_a]:
            self.speed_x = -PLAYER_SPEED
        if keys[pygame.K_RIGHT] or keys[pygame.K_d]:
            self.speed_x = PLAYER_SPEED
        self.rect.x += self.speed_x
        if self.rect.right > SCREEN_WIDTH:
            self.rect.right = SCREEN_WIDTH
        if self.rect.left < 0:
            self.rect.left = 0
```

```
    def shoot(self):
        bullet = Bullet(self.rect.centerx, self.rect.top)
        all_sprites.add(bullet)
        bullets.add(bullet)

        BULLET_SOUND.play()

# Enemy class
class Enemy(pygame.sprite.Sprite):

    def __init__(self):
        super().__init__()
        self.image = ENEMY_IMAGE
        self.rect = self.image.get_rect()
        self.rect.x = random.randint(0, SCREEN_WIDTH - self.rect.width)
        self.rect.y = random.randint(-100, -40)
        self.speed_y = ENEMY_SPEED + random.uniform(-1, 1)

    def update(self):
        self.rect.y += self.speed_y
        if self.rect.top > SCREEN_HEIGHT:
            self.rect.y = random.randint(-100, -40)
            self.rect.x = random.randint(0, SCREEN_WIDTH - self.rect.width)
            global score
            score -= 1

# Bullet class
class Bullet(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.image = BULLET_IMAGE
        self.rect = self.image.get_rect()
        self.rect.centerx = x
        self.rect.bottom = y
        self.speed_y = -10
```

```
    def update(self):
        self.rect.y += self.speed_y
        if self.rect.bottom < 0:
            self.kill()


# Explosion class
class Explosion(pygame.sprite.Sprite):
    def __init__(self, center):
        super().__init__()
        self.image = EXPLOSION_IMAGE

        self.rect = self.image.get_rect()
        self.rect.center = center
        self.last_update = pygame.time.get_ticks()
        self.frame_rate = 50

    def update(self):
        now = pygame.time.get_ticks()
        if now - self.last_update > self.frame_rate:
            self.kill()

# Button class
class Button:
 def __init__(self, text, pos, font, bg_color=BLUE, text_color=WHITE):
     self.x, self.y = pos
     self.font = pygame.font.Font(None, font)
     self.bg_color = bg_color
     self.text_color = text_color
     self.change_text(text)
 def change_text(self, text):
     self.text = self.font.render(text, True, self.text_color)
     self.size = self.text.get_size()
     self.surface = pygame.Surface((self.size[0] + 20, self.size[1] + 20))
     self.surface.fill(self.bg_color)
     self.surface.blit(self.text, (10, 10))
     self.rect = self.surface.get_rect(center=(self.x, self.y))
```

```
 def show(self, screen):
    screen.blit(self.surface, self.rect.topleft)
 def click(self, event):
    if event.type == pygame.MOUSEBUTTONDOWN:
       if self.rect.collidepoint(event.pos):  # Only check position, no get_pressed()

          return True
    return False

# Sprite groups
all_sprites = pygame.sprite.Group()
enemies = pygame.sprite.Group()
bullets = pygame.sprite.Group()

def start_game():
 global all_sprites, high_score, enemies, bullets, player, score, game_over
, game_active, replay_cost, username
 print(f"Welcome, {username}!")
 print(f"Your current high score is: {high_score}")
 all_sprites = pygame.sprite.Group()
 enemies = pygame.sprite.Group()
 bullets = pygame.sprite.Group()
 player = Player()
 all_sprites.add(player)

 for i in range(8):
    enemy = Enemy()
    all_sprites.add(enemy)
    enemies.add(enemy)

 score = 0
 replay_cost = 100
 game_over = False
 game_active = True

# Play background music
MUSIC.play(-1)
```

```
# Add button areas
left_button = pygame.Rect(50, SCREEN_HEIGHT - 100, 100, 50)
right_button = pygame.Rect(200, SCREEN_HEIGHT - 100, 100, 50)
fire_button = pygame.Rect(SCREEN_WIDTH - 150, SCREEN_HEIGHT - 100, 100, 50)

# Draw buttons function (updated for transparent buttons)
def draw_buttons():
    font = pygame.font.Font(None, 36)
    left_text = font.render("Left", True, WHITE)
    right_text = font.render("Right", True, WHITE)
    fire_text = font.render("Fire", True, WHITE)

    # Render text directly onto the screen at the button locations
    screen.blit(left_text, (left_button.x + 25, left_button.y + 10))
    screen.blit(right_text, (right_button.x + 20, right_button.y + 10))
    screen.blit(fire_text, (fire_button.x + 20, fire_button.y + 10))

# Speed increment for button-based movement
BUTTON_MOVEMENT_SPEED = 35  # Increased speed for button clicks

def start1_game(username, high_score):
 global done, game_over, score, replay_cost
 # Fetch the high score from the database when the game ends and the
player returns to the main menu
 if username:  # Check if username is valid (after login)
        # Reload the high score from the database
        conn = connect_to_db()
        cursor = conn.cursor()
        cursor.execute("SELECT high_score FROM PlayerData WHERE username = ?"
, username)
        result = cursor.fetchone()
        if result:
            high_score = result[0]  # Set the high score from the database
        cursor.close()
        conn.close()
```

```
 # Start the game directly
 start_game()

# Game loop (modified)
 while not done:

    screen.fill(BLACK)
    screen.blit(BACKGROUND_IMAGE, (0, 0))

    if game_over:
        if score >= replay_cost:
            restart_button = Button("Restart", (SCREEN_WIDTH // 4, SCREEN_HEIGHT
// 2), font=50)

            quit_button = Button("Quit", (3 * SCREEN_WIDTH // 4,
SCREEN_HEIGHT // 2)
, font=50)
            restart_button.show(screen)
            quit_button.show(screen)
            font = pygame.font.Font(None, 36)
            text = font.render(f"Use {replay_cost} score points to replay?",
True, WHITE)
            screen.blit(text, (SCREEN_WIDTH // 2 - text.get_width() // 2,
SCREEN_HEIGHT // 2 - 100))
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    done = True
                elif event.type == pygame.MOUSEBUTTONDOWN:
                    if restart_button.click(event):
                        score -= replay_cost
                        replay_cost += 100
                        start_game()
                    elif quit_button.click(event):
                        done = True
        else:
            font = pygame.font.Font(None, 36)
            text = font.render("Not enough score for replay. Exiting in 5 seconds.",
```

```
True, RED)
            screen.blit(text, (SCREEN_WIDTH // 2 - text.get_width() // 2,
SCREEN_HEIGHT // 2 - 100))
            pygame.display.flip()
            time.sleep(5)
            from play_screen import play_screen
            play_screen(username, high_score)

            # Transition back to home screen
            game_active = False
            game_over = False
        else:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    done = True
                elif event.type == pygame.KEYDOWN:

                    if event.key == pygame.K_SPACE:
                        player.shoot()
                elif event.type == pygame.MOUSEBUTTONDOWN:
                    if left_button.collidepoint(event.pos):
                        player.rect.x -= BUTTON_MOVEMENT_SPEED
                    elif right_button.collidepoint(event.pos):
                        player.rect.x += BUTTON_MOVEMENT_SPEED
                    elif fire_button.collidepoint(event.pos):
                        player.shoot()

            all_sprites.update()
            hits = pygame.sprite.groupcollide(enemies, bullets, True, True)
            for hit in hits:
                score += 1
                explosion = Explosion(hit.rect.center)
                all_sprites.add(explosion)
                EXPLOSION_SOUND.play()
                enemy = Enemy()
                all_sprites.add(enemy)
```

```
            enemies.add(enemy)

        hits = pygame.sprite.spritecollide(player, enemies, True)
        if hits:
            game_over = True

        # When game over happens, save high score to the database
        if game_over:

            update_high_score(username, score)

        all_sprites.draw(screen)
        font = pygame.font.Font(None, 36)
        text = font.render("Score: " + str(score), True, WHITE)
        screen.blit(text, (10, 10))
        high_score_text = font.render(f"High Score: {high_score}", True, WHITE)
        screen.blit(high_score_text, (10, 35))


        draw_buttons()  # Draw buttons

    pygame.display.flip()
    clock.tick(FPS)
```

CAR_GAME.PY:

```
import pygame
from pygame.locals import *
import random
import pyodbc

def start_car_game(username, high_score):
    pygame.init()

    # create the window
    width = 900
```

```
height = 600
screen_size = (width, height)
screen = pygame.display.set_mode(screen_size)
pygame.display.set_caption('Car Game')

# colors
WHITE = (255, 255, 255)
gray = (100, 100, 100)
green = (76, 208, 56)

red = (200, 0, 0)
white = (255, 255, 255)
yellow = (255, 232, 0)

# road and marker sizes
road_width = 300
marker_width = 10
marker_height = 50

# lane coordinates
left_lane = 150
center_lane = 250
right_lane = 350
lanes = [left_lane, center_lane, right_lane]

# road and edge markers
road = (100, 0, road_width, height)
left_edge_marker = (95, 0, marker_width, height)
right_edge_marker = (395, 0, marker_width, height)

# for animating movement of the lane markers
lane_marker_move_y = 0

# player's starting coordinates
player_x = 250
player_y = 400
```

```
# frame settings
clock = pygame.time.Clock()
fps = 120

# game settings
gameover = False
speed = 2
score = 0


# Database connection using Windows Authentication
def connect_to_db():
    try:
        conn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};'
                              'SERVER=localhost;'
                              'DATABASE=GameDB;'
                              'Trusted_Connection=yes;')
        return conn
    except Exception as e:
        print(f"Error connecting to database: {e}")
        return None

# Update high score in the database
def update_car_high_score(username, car_high_score):
    conn = connect_to_db()
    cursor = conn.cursor()

    # Check the current high score in the database for the given username
    cursor.execute("SELECT car_high_score FROM PlayerData WHERE
username = ?", username)
    result = cursor.fetchone()

    if result:
        current_car_high_score = result[0]
        if current_car_high_score is None or car_high_score >
current_car_high_score:
```

```
                # Update the high score if the current one is higher or if there's no
existing high score
                cursor.execute("UPDATE PlayerData SET car_high_score = ?
WHERE username = ?", car_high_score, username)
                conn.commit()

            cursor.close()
            conn.close()

        class Vehicle(pygame.sprite.Sprite):

            def __init__(self, image, x, y):
                pygame.sprite.Sprite.__init__(self)
                image_scale = 45 / image.get_rect().width
                new_width = image.get_rect().width * image_scale
                new_height = image.get_rect().height * image_scale
                self.image = pygame.transform.scale(image, (new_width,
new_height))
                self.rect = self.image.get_rect()
                self.rect.center = [x, y]

        class PlayerVehicle(Vehicle):
            def __init__(self, x, y):
                image = pygame.image.load('images/car.png')
                super().__init__(image, x, y)

        # sprite groups
        player_group = pygame.sprite.Group()
        vehicle_group = pygame.sprite.Group()

        # create the player's car
        player = PlayerVehicle(player_x, player_y)
        player_group.add(player)

        # load the vehicle images
```

```
        image_filenames = ['pickup_truck.png', 'semi_trailer.png', 'taxi.png',
'van.png']
        vehicle_images = []
        for image_filename in image_filenames:
            image = pygame.image.load('images/' + image_filename)
            vehicle_images.append(image)

        # load the crash image
        crash = pygame.image.load('images/crash.png')
        crash_rect = crash.get_rect()

        # Fetch the high score from the database
        if username:  # Check if username is valid (after login)

            conn = connect_to_db()
            cursor = conn.cursor()
            cursor.execute("SELECT car_high_score FROM PlayerData WHERE
username = ?", username)
            result = cursor.fetchone()
            if result:
                car_high_score = result[0]  # Set the high score from the database
            else:
                car_high_score = 0  # Default high score if no record exists
            cursor.close()
            conn.close()
        else:
            car_high_score = 0  # Default high score if no username is provided

        running = True
        while running:
            clock.tick(fps)

            for event in pygame.event.get():
                if event.type == QUIT:
                    running = False

                # move the player's car using the left/right arrow keys
```

```
            if event.type == KEYDOWN:
                if event.key == K_LEFT and player.rect.center[0] > left_lane:
                    player.rect.x -= 100
                elif event.key == K_RIGHT and player.rect.center[0] < right_lane:
                    player.rect.x += 100

                # check if there's a side swipe collision after changing lanes
                for vehicle in vehicle_group:
                    if pygame.sprite.collide_rect(player, vehicle):
                        gameover = True
                        if event.key == K_LEFT:
                            player.rect.left = vehicle.rect.right


                            crash_rect.center = [player.rect.left, (player.rect.center[1] +
vehicle.rect.center[1]) / 2]
                        elif event.key == K_RIGHT:
                            player.rect.right = vehicle.rect.left
                            crash_rect.center = [player.rect.right, (player.rect.center[1]
+ vehicle.rect.center[1]) / 2]

        # draw the grass
        screen.fill(green)

        # draw the road
        pygame.draw.rect(screen, gray, road)

        # draw the edge markers
        pygame.draw.rect(screen, yellow, left_edge_marker)
        pygame.draw.rect(screen, yellow, right_edge_marker)

        # draw the lane markers
        lane_marker_move_y += speed * 2
        if lane_marker_move_y >= marker_height * 2:
            lane_marker_move_y = 0
        for y in range(marker_height * -2, height, marker_height * 2):
```

```
            pygame.draw.rect(screen, white, (left_lane + 45, y +
lane_marker_move_y, marker_width, marker_height))
            pygame.draw.rect(screen, white, (center_lane + 45, y +
lane_marker_move_y, marker_width, marker_height))

        # draw the player's car
        player_group.draw(screen)

        # add a vehicle
        if len(vehicle_group) < 2:
            add_vehicle = True
            for vehicle in vehicle_group:
                if vehicle.rect.top < vehicle.rect.height * 1.5:
                    add_vehicle = False

            if add_vehicle:
                lane = random.choice(lanes)
                image = random.choice(vehicle_images)
                vehicle = Vehicle(image, lane, height / -2)
                vehicle_group.add(vehicle)

        # make the vehicles move
        for vehicle in vehicle_group:
            vehicle.rect.y += speed
            if vehicle.rect.top >= height:
                vehicle.kill()
                score += 1
                if score > 0 and score % 5 == 0:
                    speed += 1

        # draw the vehicles
        vehicle_group.draw(screen)

        # display the score
        font = pygame.font.Font(pygame.font.get_default_font(), 25)
        text = font.render('Score: ' + str(score), True, white)
        text_rect = text.get_rect()
```

```
        text_rect.center = (550, 300)
        screen.blit(text, text_rect)
        car_high_score_text = font.render(f"High Score: {car_high_score}",
True, WHITE)
        screen.blit(car_high_score_text, (500, 320))

        # check if there's a head on collision
        if pygame.sprite.spritecollide(player, vehicle_group, True):
            gameover = True
            crash_rect.center = [player.rect.center[0], player.rect.top]

        # display game over
        if gameover:
            screen.blit(crash, crash_rect)

            pygame.draw.rect(screen, red, (0, 50, width, 100))
            font = pygame.font.Font(pygame.font.get_default_font(), 16)
            text = font.render('Game over. Play again? (Enter Y or N)',
True, white)
            text_rect = text.get_rect()
            text_rect.center = (width / 2, 100)
            screen.blit(text, text_rect)

            # When game over happens, save high score to the database
            update_car_high_score(username, score)

        pygame.display.update()

        # wait for user's input to play again or exit
        while gameover:
            clock.tick(fps)
            for event in pygame.event.get():
                if event.type == QUIT:
                    gameover = False
                    running = False
                if event.type == KEYDOWN:
```

```
                    if event.key == K_y:
                        gameover = False
                        speed = 2
                        score = 0
                        vehicle_group.empty()
                        player.rect.center = [player_x, player_y]
                    elif event.key == K_n:
                        gameover = False
                        running = False

        # After the car game ends, return to the play_screen
        from play_screen import play_screen  # Import play_screen from the shooting game
        play_screen(username, high_score)  # Call play_screen to return to the game selection screen
```

CONCLUSION

The provided code represents a comprehensive game system that integrates user authentication, game selection, and two distinct games: a space shooter game and a car racing game. The system is built using the Pygame library for game development and Pyogbc for database connectivity, ensuring that player data such as usernames, passwords, and high scores are securely stored and retrieved from a SQL Server database.

KEY FEATURES:

User Authentication:

- Players can create new accounts or log in using existing credentials.

Game Selection:

- After logging in, players are presented with a menu to choose between two games: a space shooter game and a car racing game.

Space Shooter Game:

- Players control a spaceship, shooting down enemies to score points.

Car Racing Game:

- Players navigate a car through traffic, avoiding collisions to increase their score

.

BIBLIOGRAPHY

- **Pygame** - Pete Shinners
- **Pyodbc** - Michael Kleehammer.
- **SQL server management** - Microsoft.