INTERIM REPORT STUDENT MANAGEMENT SYSTEM

Dissertation submitted in fulfilment of the requirements for the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

– DATA SCIENCE WITH MACHINE LEARNING

By

Aromal Anil

Registration No: 12306910

Section: K23CH

Roll No: 29

15/11/2024

Supervisor

Aman Kumar

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

October 2024

ACKNOWLEDGEMENT

I at this moment declare that the research work reported in the dissertation/dissertation proposal entitled “CAR GAME USING PYGAMES” in partial fulfilment of the requirement for the award of Degree for Master of Technology in Computer Science and Engineering at Lovely Professional University, Phagwara, Punjab is an authentic work carried out under the supervision of my research supervisor Mr. Aman Sharma. I have not submitted this work elsewhere for any degree or diploma.

I understand that the work presented herewith directly complies with Lovely Professional University’s Policy on plagiarism, intellectual property rights, and the highest standards of moral and ethical conduct. Therefore, to the best of my knowledge, the content of this dissertation represents an authentic and honest research effort conducted, in its entirety, by me. I am fully responsible for the contents of my dissertation work.

AROMAL ANIL
REG.NO - 12306910

Table of Contents

INTRODUCTION

1.1 Project Overview

The Car Racing Game is a Python-based application designed to provide an engaging and interactive gaming experience. Built using the **Pygame** library, this game allows users to control a car, navigate a dynamic track, and avoid obstacles to score points. The project demonstrates the capabilities of Pygame developing visually appealing, interactive, and responsive 2D games.

The game operates with a simple yet exciting interface, where players steer their car through an ever-changing environment. Key functionalities include controlling the car's movement, dynamically generated obstacles, a scoring system that rewards progress, and game-over mechanics when the car collides with an obstacle. The gameplay is enhanced by smooth animations, sound effects, and an intuitive user interface.

1.2 Purpose and Significance

The purpose of this project is to demonstrate how a game can be created using Python and the Pygame library, providing a hands-on experience in game development. By creating a fully functional car racing game, this project highlights the fundamental principles of game loops, sprite management, event handling, and collision detection in Pygame.

The significance of the Car Racing Game lies in its ability to combine entertainment with programming concepts. It showcases how Python can be used to develop a real-world application that is both educational and enjoyable. Additionally, the game can be expanded in future iterations, with added features such as power-ups, multiplayer functionality, or more complex track designs, making it a versatile project for learning and development.

1.3 Game Details

To provide a comprehensive and enjoyable gaming experience, this project includes the following features and mechanics:

1. **Game Elements**:
   - **Player Car**: The player's car is controlled using keyboard inputs (e.g., arrow keys).
   - **Obstacles**: Randomly generated obstacles that the player must avoid to continue the gam.
   - **Track Environment**: A scrolling background that creates the illusion of movement.
   - **Score System**: Points awarded based on the distance traveled or time survived.
2. **Game Loop and Mechanics**:
   - **User Interaction**: Players control the car's movement using key events.
   - **Collision Detection**: Detects collisions between the player’s car and obstacles, triggering game-over state.
   - **Dynamic Difficulty**: The speed of the game increases over time, adding to the challenge.
3. **Pygame Integration**:
   - **Graphics**: Smooth animations and sprite handling for a polished visual experience.
   - **Sound Effects**: Background music and sound effects to enhance gameplay.
   - **Scalability**: The code is modular and can be extended with new features like additional
   - level or cars.

By focusing on the fundamentals of game development and leveraging Pygame's features, the Car .

Game provides an excellent platform for learning and experimentation in the field of interactive programming.

II. Objectives and Scope of the Project

2.1 Project Objectives

The primary objective of this project is to design and implement a **Car Racing Game** that provides an engaging and immersive gaming experience using Python and Pygame. The game focuses on interactive gameplay, dynamic challenges, and a user-friendly interface. Specific objectives include:

1. **Develop an Interactive Car Racing Game**:
   - Create a game that allows players to control a car, navigate through obstacles, and achieve high scores.
   - Incorporate responsive controls to ensure smooth and precise gameplay.
2. **Implement Dynamic and Challenging Gameplay**:
   - Design dynamically generated obstacles to make the game unpredictable and engaging.
   - Introduce increasing difficulty by adjusting the speed and frequency of obstacles as the game progresses.
3. **Provide a Robust Game Loop and Features**:
   - Implement a continuous game loop to handle events, update the game state, and render graphics.
   - Add collision detection to end the game when the car collides with obstacles.
   - Include a scoring mechanism to track player performance.
4. **Highlight the Use of Pygame for Game Development**:
   - Showcase how Pygame can be used to handle graphics, animations, sound effects, and game logic.
   - Provide an example of how Python can be leveraged to create real-time applications like games.
5. **Design a Scalable and Extendable Game Architecture**:
   - Develop modular code that allows for future enhancements, such as adding power-ups, new car designs, or multiplayer modes.

- Build a flexible system for integrating new features like different tracks, levels, or leaderboard functionality.

2.2 Project Scope

The scope of this project is centered on the design, development, and demonstration of a **Car Racing Game** using Pygame, focusing on the following aspects:

1. **Core Game Mechanics**:
    - Develop a car racing game where the player controls a car to avoid obstacles and earn points.
    - Implement basic game functionalities, including car movement, obstacle generation, collision detection, and scoring.
2. **Dynamic Gameplay and Challenges**:
    - Introduce random obstacle generation for variability and excitement.
    - Increase game difficulty over time by speeding up the gameplay and increasing obstacle frequency.
3. **Graphics and Sound Integration**:
    - Use Pygame's graphics capabilities to render smooth animations and visually appealing sprites.
    - Incorporate sound effects and background music to enhance the gaming experience.
4. **Scalability and Future Enhancements**:
    - Design the codebase to be modular, allowing for the addition of new features like different levels, power-ups, or multiplayer modes.
    - Plan for the integration of a leaderboard system to track player scores globally or locally.
5. **Limitations**:
    - The current implementation is focused on a single-player experience with basic

mechanics.

- Advanced features, such as a variety of tracks, story modes, or multiplayer support, are beyond the scope of this version.
- Persistent data storage for scores or progress (e.g., using databases or files) is not

  included in this version.

- Security measures for score manipulation or hacking prevention are not implemented.

III. Application Tools

3.1 Software Applications

The development of the **Car Racing Game** project utilizes several software tools to ensure a streamlined and efficient development environment. These tools are as follows:

1. **Python (Version 3.x)**:
   - Python is the core language used for implementing the game. It is versatile, beginner-friendly, and supports a wide range of libraries, including Pygame, which is specifically designed for game development. Python's simplicity allows for rapid prototyping and maintenance, making it an excellent choice for this project.
2. **Pygame Library**:
   - Pygame is used to handle the game's graphics, animations, event handling, and sound effects. It provides an intuitive framework for building 2D games, enabling developers create engaging and interactive gameplay experiences with minimal complexity.
3. **Integrated Development Environment (IDE) - Visual Studio Code (VS Code)**:
   - VS Code is used for writing, debugging, and testing Python code. It offers features like syntax highlighting, integrated debugging tools, version control support, and extensions enhance productivity during development.
4. **Git and GitHub**:
   - Git is utilized for version control, while GitHub is used as a platform for project

collaboration and code management. Git allows developers to track changes in the codebase, while GitHub facilitates sharing, collaboration, and synchronization of the project across multiple contributors.

5. **Graphics and Sound Tools**:
    - **GIMP/Photoshop**: These tools can be used to design or edit game sprites and backgrounds.
    - **Audacity/FreeSound**: Audio editing software like Audacity or online resources like FreeSound can be utilized for creating and integrating sound effects and background music.

3.2 Programming Languages of the Project

The project is primarily implemented in Python due to its extensive libraries and ease of use in game development.

1. **Python**:
    - Python is the core language for implementing all game mechanics, including the game loop, collision detection, scoring system, and obstacle generation.
    - The Pygame library, integrated with Python, provides functionality for rendering graphics, handling events, and playing sounds, making it an ideal choice for this project.
2. **Future Expansion (Optional)**:
    - While the current implementation focuses on core game functionality, future iterations could integrate additional programming tools or frameworks, such as:
        - **SQLite**: To store high scores or player data persistently.
        - **Networking Libraries**: For multiplayer functionality, if added in future versions.

IV. Project Structure of the Car Racing Game

The Car Racing Game is organized into several components, classes, and functions to ensure a modular, maintainable, and scalable structure.

4.1 Main Components of the Project

1. **Game Entity (Car Class and Obstacles Class)**:
   - **Purpose**: Represents the primary entities in the game, including the player's car and obstacles.
   - **Responsibilities**: Stores attributes like position, speed, dimensions, and appearance for the car and obstacles.
2. **Game Manager (Game Class)**:
   - **Purpose**: Manages the overall game logic, including game states (start, running, game over) and score tracking.
   - **Responsibilities**: Handles collision detection, score updates, and interactions between the car and obstacles.
3. **Graphical and Audio Interface (Pygame)**:
   - **Purpose**: Provides the visual and audio representation of the game.
   - **Responsibilities**: Renders the game environment, displays score, handles animations, and plays sound effects.
4. **Input Handling**:
   - **Purpose**: Captures player input (e.g., arrow keys for movement).
   - **Responsibilities**: Translates player input into actions such as moving the car or restarting the game.

4.2 Classes and Their Functions

1. **Car Class**
   - **Purpose**: Represents the player's car in the game.
   - **Attributes**:
     - `x, y`: Position of the car.
     - `width, height`: Dimensions of the car.

        - `speed`: Movement speed of the car.
        - `color/image`: Visual representation of the car.
    - **Methods**:
        - `__init__(self, x, y, width, height, speed)`: Initializes the car's attributes.
        - `move(self, direction)`: Updates the car's position based on the player's input.
        - `draw(self, screen)`: Renders the car on the game screen.
2. **Obstacle Class**
    - **Purpose**: Represents obstacles on the track.
    - **Attributes**:
        - `x, y`: Position of the obstacle.
        - `width, height`: Dimensions of the obstacle.
        - `speed`: Speed of the obstacle's movement.
    - **Methods**:
        - `__init__(self, x, y, width, height, speed)`: Initializes the obstacle's attributes.
        - `update(self)`: Moves the obstacle downward.
        - `draw(self, screen)`: Renders the obstacle on the game screen.
3. **Game Class**
    - **Purpose**: Manages the game loop and overall functionality.
    - **Attributes**:
        - `car`: Instance of the Car class.
        - `obstacles`: List of Obstacle objects.
        - `score`: Current score.
        - `running`: Boolean indicating the game state.
    - **Methods**:
        - `__init__(self)`: Initializes the game elements.
        - `spawn_obstacle(self)`: Generates new obstacles at random positions.
        - `check_collision(self)`: Detects collisions between the car and obstacles.
        - `update(self)`: Updates the position of all game elements.
        - `draw(self)`: Renders the game on the screen.
        - `reset(self)`: Resets the game after it ends.

4.3 Interaction Between Components

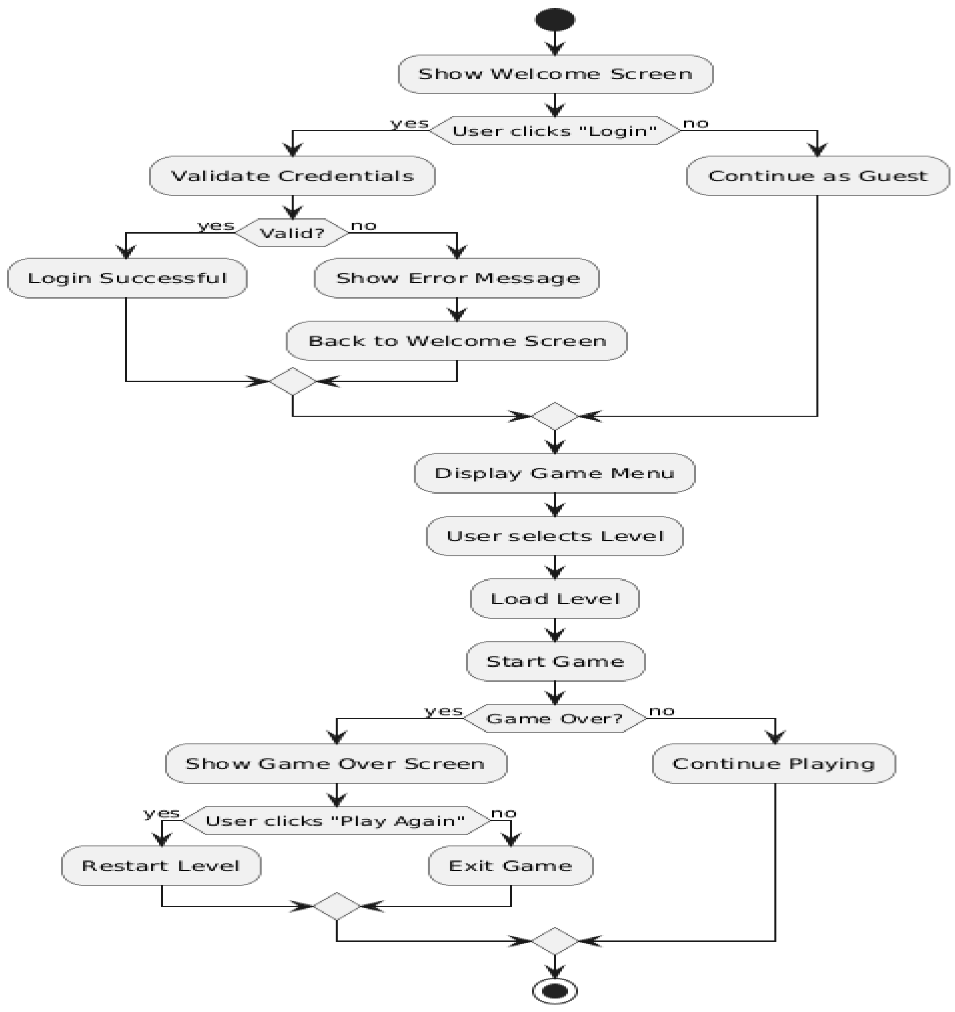
1. **Starting the Game**:
    - The `Game` class initializes the car and obstacle objects.
    - The game enters a loop where the screen is updated, and player input is captured.

2. **Player Input**:
    - Arrow key inputs move the car left or right by calling the `move()` method in the `Car` class.
3. **Obstacle Movement**:
    - Obstacles move downward based on their speed. Once they exit the screen, they are removed, and new obstacles are spawned.
4. **Collision Detection**:
    - The `check_collision()` method in the `Game` class compares the car's position with obstacles to determine if a collision has occurred.
5. **Scoring and Game Over**:
    - Each obstacle avoided increases the score.
    - Upon collision, the game stops, and the player has the option to restart.
6. **Rendering**:
    - The `draw()` methods of the `Car` and `Obstacle` classes render the game entities on the screen, while the `Game` class handles the overall screen updates.

This structure ensures that the Car Racing Game is well-organized and allows for easy future expansion, such as adding new obstacles, difficulty levels, or multiplayer functionality

V. Flowchart or Algorithm of the Project

Below is a simplified flowchart that illustrates the process flow of the priority scheduling system.



Explanation of the Flowchart:

1. **Start:**
   - The flow begins when the user enters student data into the GUI.
2. **Check if Student Exists:**
   - The system checks the records to determine if a student with the provided ID already exists.
3. **Update or Add Student Record:**
   - If the student exists, their record is updated with the new data.
   - If the student does not exist, a new record is created and added to the system.
4. **Success Confirmation:**
   - The system confirms the operation (update or addition) has been successfully completed.
5. **Display Success Message:**
   - A message is displayed in the GUI to inform the user of the successful operation.
6. **Prompt for New Action:**

- The system prompts the user to perform another task, such as adding, updating, searching, or deleting a student.

7. **Decision:**
    - If the user chooses to continue, the flow loops back to the start.
    - If the user chooses to end, the operation terminates.
8. **End:**
    - The flow ends once the user completes their tasks and exits.