ASCII Car Racer: A Console-Based Racing Game

Course: CL1004 - Object-Oriented Programming (OOP) Lab
Batch: 24K | **Section:** 2K
Group Members:

- Rayyan Asif (24K-0993)
- Akbar Younsi (24K-0947)
- Ali Hasnain (24K-0696)

Project Overview

This project is an engaging console-based racing game where the player controls a car represented by ASCII characters (|O|) and must avoid obstacles (#) while driving down a scrolling track. The player can move left (A) or right (D) to dodge obstacles, and the game progressively increases in difficulty as the speed and number of obstacles increase.

Objectives

- Develop an interactive ASCII-based racing game using C++.
- Implement keyboard controls to move the car left and right.
- Generate random obstacles for the player to avoid.
- Implement collision detection to determine when the player crashes.
- Track the score based on survival time.
- Explore the possibility of transitioning to a GUI-based version for improved visual representation.

Technologies Used

- **Programming Language:** C++
- **Optional GUI Framework:** SDL2, SFML, or Qt for future graphical enhancements.

Features

- Simple ASCII Graphics – Retro-style racing experience.
- Keyboard Controls – Move left (A) or right (D).
- Randomized Obstacles – Each game session is unique.
- Progressive Difficulty – Speed increases as the score rises.
- Score Tracking System – Display high scores.
- Game Over Condition – Triggered when the car hits an obstacle.
- Multiple Lanes – Adds complexity and challenge.
- Power-ups – Shields or speed boosts for enhanced gameplay.
- Potential GUI Upgrade – Future implementation of graphical elements for an enhanced user experience.

Game Mechanics

- **Track Representation:** Two walls (| |) define the track, with the player's car and obstacles appearing in between.
- **Car Movement:** Player moves left (A) or right (D) to dodge obstacles.
- **Obstacle Behavior:** Obstacles appear randomly and move downward.
- **Collision Detection:** If the car position matches an obstacle, the game ends.
- **Speed Increase:** As the player survives longer, the speed of obstacle movement increases.

Object-Oriented Programming (OOP) Concepts Used

1. Encapsulation (Data Hiding & Modular Code)

- The Car, Obstacle, and Game classes will encapsulate their respective functionalities, ensuring modularity and data protection.

2. Abstraction (Hiding Implementation Details)

- The game logic (movement, collision detection, score tracking) is abstracted inside a Game class.
- Players only interact with simple controls, without needing to understand the internal implementation.

3. Inheritance (Code Reusability)

- If power-ups (e.g., shields, speed boosts) are implemented, they can inherit from a base class Obstacle.

4. Polymorphism (Dynamic Behavior)

- By using virtual functions, different obstacle types (Obstacle, PowerUp) can have their own movement patterns.

5. Composition (Has-A Relationship)

- The Game class has a Car and a list of obstacles, which is a better design choice than inheritance for these relationships.

6. Dynamic Memory Management

- If obstacles are dynamically generated, we will use pointers and dynamic allocation (new / delete or unique_ptr).

7. Event Handling & State Management

- The game must process user inputs (A/D keys) and update game state (score, collisions, speed).
- Finite State Machines (FSMs) can manage menu, gameplay, and game-over states.

Expected Outcome

By the end of this project, we will have a fully functional console-based car racing game featuring:

✅ Real-time player control
✅ Obstacle avoidance mechanics
✅ A score-tracking system
✅ Increasing difficulty for enhanced challenge
✅ Potential transition to a GUI version for a more visually appealing experience.

This project reinforces OOP principles through a structured, modular approach while delivering an engaging gameplay experience.

Conclusion

This ASCII-based car racing game will be an excellent demonstration of Object-Oriented Programming (OOP) in C++, incorporating:

- **Encapsulation** for data security
- **Abstraction** to simplify game logic
- **Inheritance & Polymorphism** for code reusability
- **Composition** for better object relationships
- **Dynamic Memory Management** for efficient object handling
- **Exploration of GUI Implementation** for future graphical enhancements

This project will provide valuable experience in designing and implementing OOP-based applications in C++ with potential scalability into a graphical user interface.