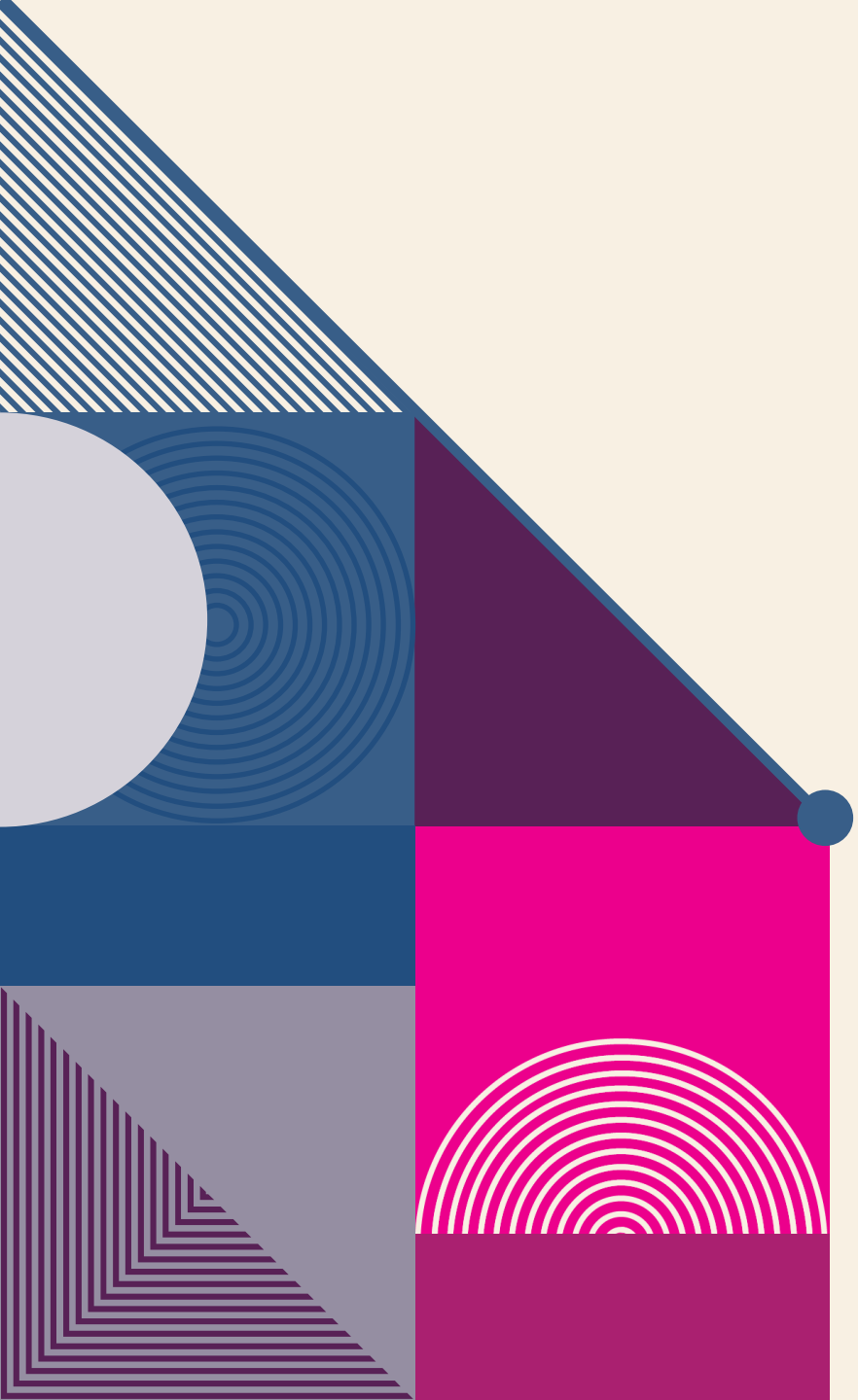
CAR GAME

OOP PROJECT

Riya Bhart (23k-0063)
Syeda Rija Ali (23k-0057)
Rayyan Merchant (23k-0073)



AGENDA

Introduction

Libraries

Classes

Functions

Additional Explanation

Oop Concepts

INTRODUCTION

THIS CODE PRESENTS A SIMPLE YET ENGAGING CAR GAME IMPLEMENTED IN C++, LEVERAGING OBJECT-ORIENTED PROGRAMMING (OOP) PRINCIPLES AND WINDOWS API FUNCTIONS FOR CONSOLE MANIPULATION. THE GAME CHALLENGES PLAYERS TO CONTROL A CAR AND NAVIGATE IT THROUGH A STREAM OF ONCOMING TRAFFIC, REPRESENTED BY ENEMY CARS. AS THE GAME PROGRESSES, THE PLAYER'S SCORE INCREASES FOR EACH SUCCESSFULLY AVOIDED COLLISION, OFFERING MULTIPLE DIFFICULTY LEVELS TO CATER TO VARYING SKILL LEVELS.

LIBRARIES

The libraries used in the project are :

#include <iostream>

#include <windows.h>

#include <time.h>

1. iostream (#include <iostream>):

This library provides basic input/output operations in C++. It includes standard objects like cin (for input) and cout (for output).

2. windows.h (#include <windows.h>):

windows.h provides Windows API functions for console manipulation, including cursor positioning, color settings, and screen clearing. It defines data types like HANDLE and structures like COORD for interacting with console resources.

Functions like SetConsoleCursorPosition(), SetConsoleCursorInfo(), SetConsoleTextAttribute(), and GetStdHandle() are used for console manipulation.

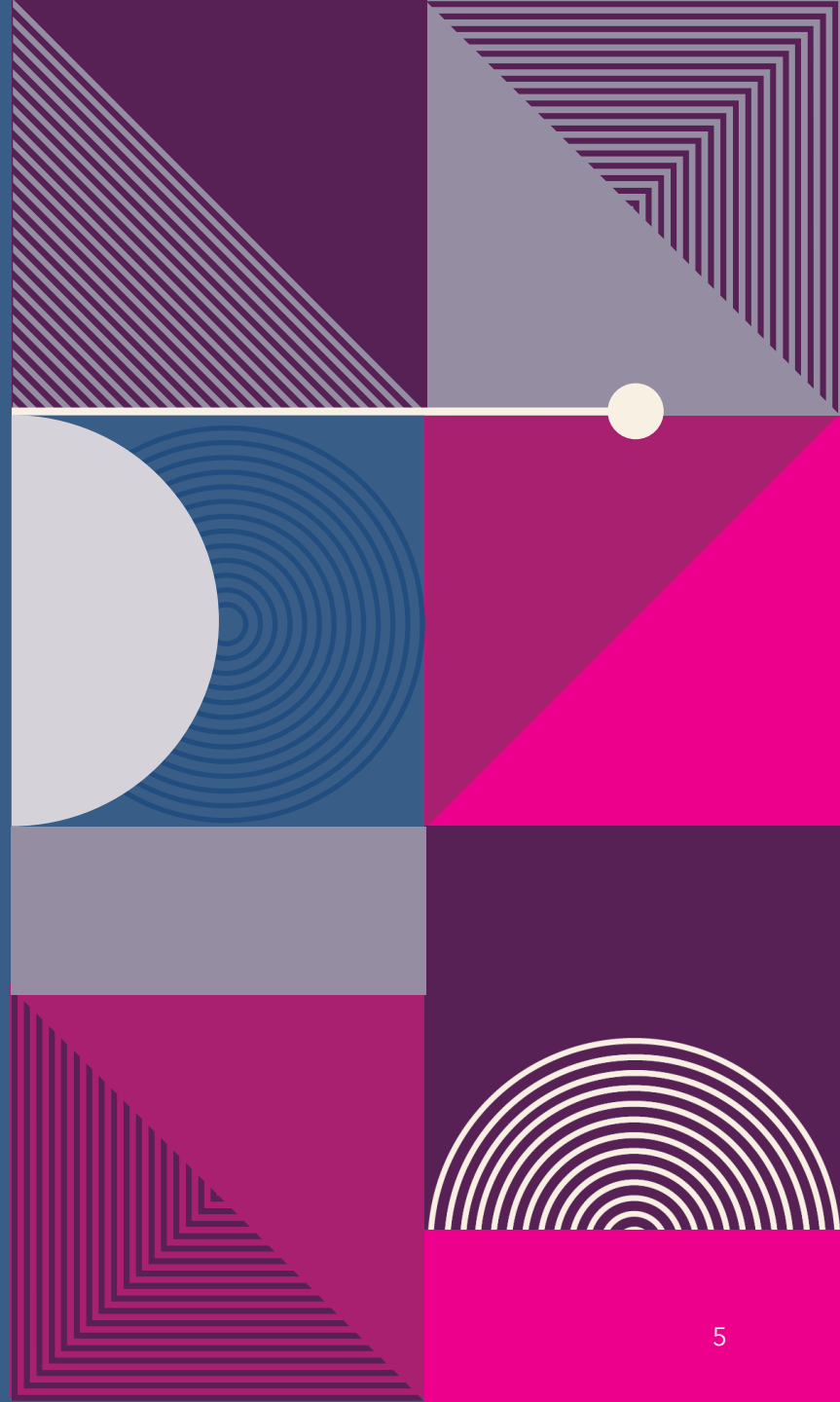
3. time.h (#include <time.h>):

This library provides functions for manipulating date and time information.It includes functions for measuring time intervals, generating random numbers, and formatting time values.The rand() function, used for generating random numbers, is included from this library.

CLASSES

Gameplay class: This represents the base class for gameplay functionality. Provides methods for drawing the game border, updating the score, managing the car, detecting collisions, and handling erasure.

Enemy class class: Serves as the base class for enemy-related functionality. Defines methods for drawing, generating, erasing, and resetting enemies on the screen



DERIVED CLASSES

- **Easy Class**
- **Medium Class**
- **Hard Class**

1. **easyClass class:** - Inherits from both gamePlay and enemyClass. - Implements the play() function for easy difficulty level, defining specific gameplay logic such as enemy speed and behavior.
2. **mediumClass class:** - Inherits from both gamePlay and enemyClass. - Implements the play() function for medium difficulty level, adjusting gameplay parameters for increased challenge compared to the easy level.
3. **hardClass class:** - Inherits from both gamePlay and enemyClass. - Implements the play() function for hard difficulty level, providing the most challenging gameplay experience with faster enemies and increased obstacles.

FUNCTIONS

- **gotoxy(int x, int y):** Positions the console cursor at the specified coordinates (x, y) for printing characters or other console manipulations.
- **setcursor(bool visible, DWORD size):** Controls the visibility and size of the console cursor, enhancing user experience by adjusting cursor appearance as needed.
- **gameover():** Clears the console screen, displays a game over message, and waits for user input before returning to the main menu.
- **drawBorder():** Draws the border of the game area on the console screen, providing visual boundaries for gameplay elements.

FUNCTIONS

- **updateScore():**Updates and displays the player's score on the console screen during gameplay.
- **drawCar():**Draws the player's car on the console screen, allowing the player to visually perceive their position in the game.
- **eraseCar():**Clears the area occupied by the player's car on the console screen, removing its previous position.
- **collision():**Checks for collisions between the player's car and enemy cars, returning 1 if a collision occurs and 0 otherwise.
- **drawEnemy(int ind), genEnemy(int ind), eraseEnemy(int ind), resetEnemy(int ind):**Handle the drawing, generation, erasing, and resetting of enemy cars on the console screen, respectively.
- **play() (virtual in base classes):**Represents the main gameplay logic, to be implemented differently in derived classes for various difficulty levels.

ADDITIONAL EXPLANATION

- **Color Settings:** The code uses system("color") to set the background and foreground colors of the console window. Colors are specified using hexadecimal codes or predefined color names.

- **User Input Handling:** The game uses GetAsyncKeyState() to detect keyboard input for controlling the car and exiting the game.

- **Random Enemy Generation:** Enemies are randomly generated on the screen within a specified range.

OOP CONCEPTS

I. Inheritance:

Explanation: Inheritance is a fundamental OOP concept where a class (child or derived class) can inherit attributes and behaviors from another class (parent or base class). It facilitates code reuse and promotes a hierarchical structure in object-oriented design.

Example: In this code, classes like easyClass, mediumClass, and hardClass inherit functionality from both gamePlay and enemyClass. They inherit common gameplay and enemy-related functionalities from these base classes.

OOP CONCEPTS

2. Polymorphism:

Explanation: Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables a single interface to represent different types of objects and allows methods to be dynamically bound at runtime, facilitating code flexibility and extensibility.

Example: The play() function in the base classes gamePlay and enemyClass is declared as virtual. This allows derived classes to override this function with their own implementations, enabling different behavior for each difficulty level.

OOP CONCEPTS

3. Encapsulation:

Explanation: Encapsulation is the bundling of data and methods that operate on that data into a single unit (class). It promotes information hiding and protects the internal state of an object from external interference.

Example: The classes gamePlay and enemyClass encapsulate related functionality such as drawing, updating, and managing game elements (car, enemies). Data members and member functions within these classes are encapsulated, ensuring data integrity and modularity.

OOP CONCEPTS

4. Abstraction:

Explanation: Abstraction involves hiding complex implementation details and presenting only essential features of an object to the outside world. It enables developers to focus on what an object does rather than how it does it, promoting code simplification and ease of use.

Example: The base classes gamePlay and enemyClass provide an abstraction of common gameplay and enemy-related functionality. They hide implementation details such as console manipulation and collision detection, allowing derived classes to focus on specific implementations for different difficulty levels.

THANK YOU