HELICOPTER GAME

OOC MINIPROJECT

1

2

3

4

5

6

7

8

9

Problem Statement

The helicopter game is a simple yet highly engaging game that challenges players to control the vertical movement of a helicopter while avoiding obstacles and terrain. Despite its straightforward design, recreating the game presents a variety of challenges that require the use of fundamental programming concepts and efficient design practices. The key problem is to develop a browser-based game that:

- Implements smooth, real-time control of a helicopter's movement.
- Generates a dynamic, obstacle-filled environment.
- Provides an interactive and responsive user experience.
- Ensures stable performance with minimal resource consumption.

C++ Libraries used

SFML LIBRARIES

- sf::Texture loads and manages images.
- sf::Sprite represents textures on the screen.
- sf::RenderWindow creates and manages the game window

STL LIBARIES

For Dynamic Array Declaration

STANDARD LIBRARIES

<vector>: For managing dynamic arrays of obstacles.
<iostream>: For input and output operations, such as error messages.

CUSTOM CLASSES

"Helicopter.h": For defining the Helicopter class.
"Obstacle.h": For defining the Obstacle class.
"LoginPage.h": For the login page functionality.

List of OOC concepts used in their project

1
2
3
4
5
6
7
8
9

OOC Concept	Explanation	Usage in the Project
Classes and Objects	A class is a blueprint for creating objects, which have attributes and methods to define behavior.	The game uses classes like `Helicopter`, `Obstacle`, `Game`, and `ScoreManager`, each representing a specific entity. Objects control helicopter movement, obstacle generation, and game state.
Encapsulation	Bundling data and methods within a class and restricting direct access to some attributes to protect the object's state.	The `Helicopter`, `Obstacle`, and `Game` classes hide internal implementation details and expose only necessary methods for interaction. The helicopter's position is controlled via methods,

List of OOC concepts used in their project

Inheritance	Allows one class to inherit properties and behaviors from another class, promoting code reuse.	A base class `GameObject` holds common functionality, which is inherited by subclasses like `Helicopter` and `Obstacle`. They extend the base class with additional functionality, such as controlling movement and detecting collisions.
Constructor & Destructor	A constructor initializes the object's state when instantiated, while a destructor cleans up when the object is destroyed.	Classes such as `Helicopter` and `Obstacle` have constructors that set their initial state (position, velocity, etc.). Destructors manage memory when objects are removed from the game, especially if dynamic memory allocation is involved.
Abstraction	Simplifies complex systems by only exposing essential details and hiding the underlying implementation.	The player interacts with simple controls (e.g., a key press), without knowing the underlying physics or collision detection. Details such as obstacle generation and scoring are managed within respective classes.

Tasks performed

1. Game Architecture Design: Structured game components (helicopter, obstacles, score tracking).
2. Helicopter Class Implementation: Developed class for helicopter movement (gravity, user input).
3. Obstacle Class Creation: Generated and managed obstacles, ensuring randomization.
4. Collision Detection System: Implemented logic to detect collisions with obstacles and terrain.
5. Game Loop Development: Created main event loop for updating game state and rendering.
6. Score Tracking and Management: Developed scoring system and high score tracking.
7. Rendering the Game: Built rendering engine for smooth display of game elements.
8. User Input Handling: Set up controls for helicopter movement based on user actions.
9. User Interface Design: Polished UI with current score, high score, and game messages.
10. Audio Integration: Added sound effects and background music for enhanced gameplay.

Screenshots of output


SCORE: 0

Conclusion

The helicopter game project demonstrates the effective use of Object-Oriented Programming (OOP) concepts to create a simple yet engaging browser-based game. By leveraging OOP principles such as classes, encapsulation, abstraction,etc.The project achieves a modular, reusable, and maintainable code structure. Each game entity, such as the helicopter and obstacles, is designed with clear responsibilities and behaviors, encapsulated in their respective classes. This not only simplifies the development process but also makes it easier to extend and modify the game in the future.

Through this project, developers gain practical experience in handling real-time gameplay mechanics like user input, collision detection, and dynamic obstacle generation. The use of abstraction allows for hiding complex logic, enabling players to focus on the game without worrying about internal details. Overall, the project serves as an excellent learning platform for understanding core OOP concepts in game development, while also showcasing the ability to create interactive and entertaining experiences with fundamental programming techniques.

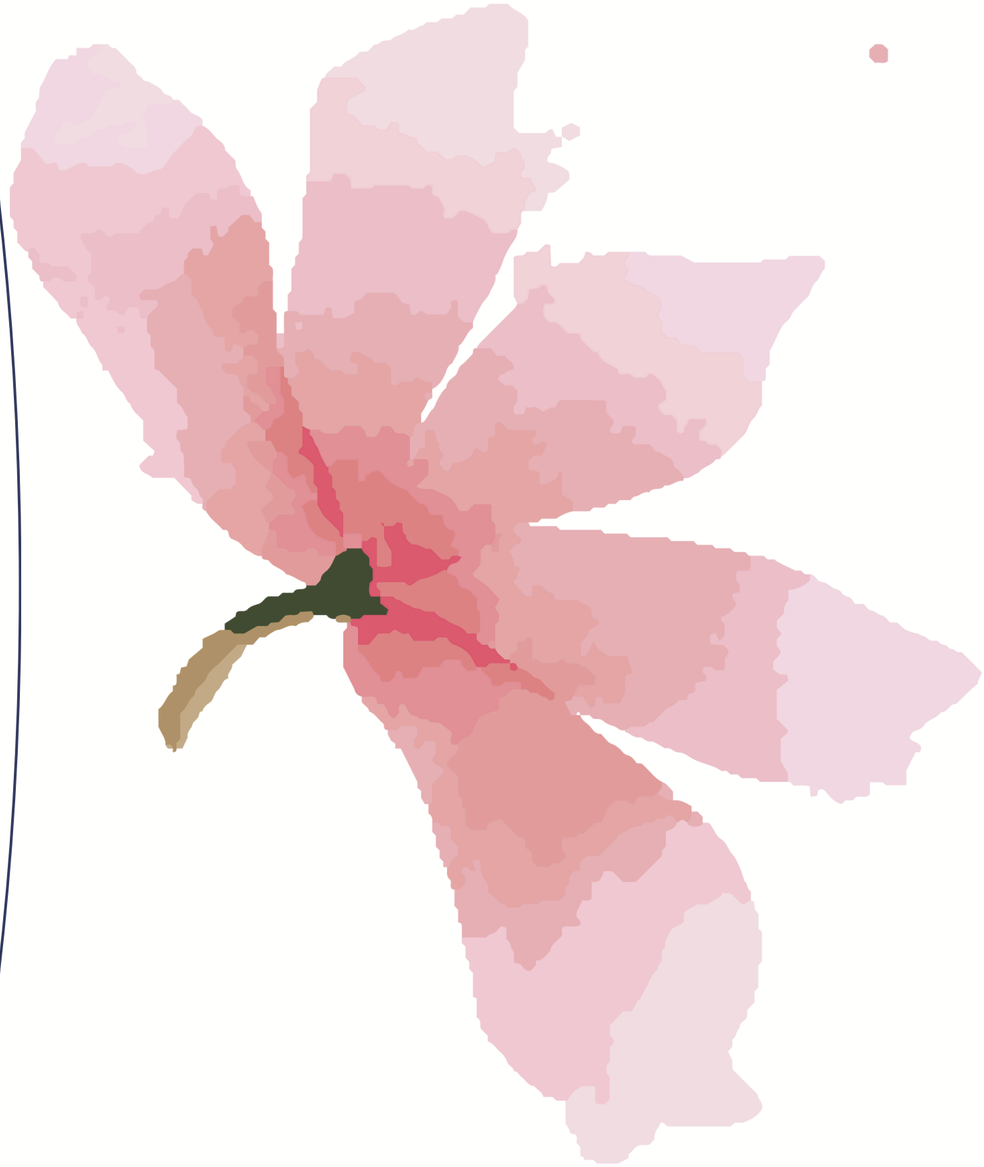
References

1] Raph Koster, William V. Wright, A Theory of Fun for Game Design, Paraglyph Press, 2005.

[2] Peter Millef, Collision Detection in 2D Games, Journal of Game Design and Development, vol. 10, pp. 34-56, 2010.

[3] Jeannie Novak, Game Development Essentials: Game Project Management, Cengage Learning, 2012.

[4] Bruce Sutherland, C++ Game Development Primer, Apress, 2014.



Thank you

Group Members :

- **Jinesha Gandhi - 38**
- **Aarushi Nishant Singh - 44**
- **Gaurika Nawani - 61**
- **Praveet Gupta - 59**