Project Report - FLAPPY BIRD UNITY

Srimanth Lashkar
ID: SE23UCSE213

1 Introduction

This project is a 2D Endless Runner game developed using the **Unity Game Engine**. The objective of the game is to control a bird character, navigating it through an infinite series of procedural pipe obstacles without colliding. The project focuses on real-time physics simulation, collision detection, object instantiation, and score management.

The main motivation for this project was to recreate the mechanics of the classic viral hit "Flappy Bird" to understand the core principles of 2D game development in Unity. It serves as a practical implementation of Rigidbody mechanics and the fundamental Unity game loop.

2 Software and Tools Used

- **Unity Game Engine** (Version 2021.3 or later) – For game physics, rendering, and scene management.
- **Visual Studio / VS Code** – For writing C# scripts.
- **C# (C Sharp)** – The programming language used for all game logic.
- **Unity UI System** – For the HUD (Heads-Up Display), Score, and Game Over screens.
- **Windows OS** – Development environment.

3 Project Structure

The project is organized into the following main folders within the Unity Assets directory:

- **Scenes/:** Contains the `Game.unity` (main gameplay level) and `Start Menu.unity` (title screen).
- **Scripts/:** Contains all C# logic files:
  - `BirdScript.cs`: Controls player physics and input (Flap mechanics).

    - `PipeSpawnerScript.cs`: Manages the endless generation of obstacles at random heights.
    - `LogicScript.cs`: Handles global game states, scoring, and UI screens.
    - `PipeMoveScript.cs`: Handles the leftward movement and cleanup of pipe objects.
- **Prefabs/:** Stores the `Pipe` prefab (a pre-configured game object containing top/bottom pipes and a score trigger).
- **Sprites/:** Contains 2D image assets for the bird, pipes, clouds, and background.

4 Core Features and Mechanics

- **Physics-Based Movement:** The bird uses Unity's `Rigidbody2D` component. Gravity is applied automatically, and pressing `Spacebar` applies an upward velocity vector to simulate flapping.
- **Infinite Procedural Generation:** The `PipeSpawnerScript` instantiates new Pipe obstacles at set intervals. The Y-position is randomized to create variation in difficulty.
- **Collision System:** The game uses `BoxCollider2D` (Pipes/Ground) and `CircleCollider2D` (Bird). A collision triggers the Game Over state, while passing through a hidden "Trigger" collider increments the score.

5 User Interface Controls

The UI provides immediate feedback to the player and control over the game state:

- **Score Display:** A text element updates dynamically (+1) every time the bird successfully passes a pipe pair.
- **Game Over Screen:** Appears upon collision, freezing the game and offering a "Restart" button.
- **Start Menu:** A simple entry scene allowing the user to launch the game.

6 How to Re-create the Project

To re-create this project:

1. **Install Unity Hub** and create a new 2D Project.
2. **Import Assets:** Drag the bird, pipe, and background sprites into the Project window.
3. **Setup the Bird:** Add a `Rigidbody2D` and `CircleCollider2D` to the bird sprite. Attach `BirdScript.cs`.

4. **Create the Pipe Prefab:**
    - Create an empty GameObject with two pipe sprites (top and bottom).
    - Add `BoxCollider2D` to the pipes and a Trigger Collider in the gap between them.
    - Save this as a Prefab.
5. **Scripting:** Write the C# scripts (`BirdScript`, `PipeSpawner`, `LogicScript`) and attach them to the relevant objects in the Hierarchy.
6. **UI Setup:** Create a Canvas, add Text for the score, and Buttons for Restart/Quit. Connect the button `OnClick()` events to `LogicScript` functions.
7. **Run and Test:** Press Play to test physics settings (gravity scale, flap strength) and spawn rates.

7 Conclusion

This project demonstrates fundamental game development concepts such as **Object Pooling** (via instantiation), **Physics Simulations**, and **UI Event Systems** using Unity. The result is a fully functional arcade game that challenges the player's reaction time while maintaining a seamless game loop. It successfully integrates visual creativity with logical programming to recreate a polished gameplay experience.