TITLE:FLAPPY BIRD IN UNITY USING C#

INTENDED GAME SYSTEMS

The game's intended platform was mobile devices, specifically iOS and Android smartphones and tablets. It was designed to be easy to pick up and play, featuring one-touch controls.

TARGET AUDIENCE

The game appeals to a wide audience due to its simplicity and accessibility. It particularly targets casual gamers, both young and old, who enjoy quick, challenging, and addictive gameplay experiences on mobile devices.It can be played as a leisure game of all age groups.

INTENDED ESRB RATING

Flappy Bird, as a simple and non-violent arcade-style game with no objectionable content, would likely have received an "E for Everyone" rating from the Entertainment Software Rating Board (ESRB). This rating is typically assigned to games that are considered suitable for all ages, containing content that is generally suitable for players of all ages, including young children.

GAME STORY

The game of Flappy Bird is simple. There is a bird on the screen and you tap the screen to make the bird flap, guiding him through a series of pipes.

Flappy Bird is a mobile game that gained immense popularity for its simplicity and addictive gameplay. The player controls a bird, navigating it through a series of obstacles by tapping the screen to make the bird flap its wings and ascend. The objective is to fly as far as possible, passing through narrow gaps between pipes or columns without touching them. This is done by using the spacebar(here) so that player increases their score the further they make it.

GAME FLOW

1. **Start Screen:** The game begins with a simple start screen displaying the title/logo of Flappy Bird and a "Start" button.
2. **Gameplay:** Upon tapping "Start," the player enters the gameplay mode. The bird automatically starts flying, and the player's interaction involves tapping the screen to make the bird flap and ascend. The objective is to navigate the bird through gaps between obstacles (pipes or columns) by timing the taps to maintain flight.
3. **Obstacle Navigation:** Pipes or columns appear on the screen, creating gaps of varying sizes. The challenge lies in maneuvering the bird through these gaps by tapping to adjust its altitude. The player must avoid colliding with the obstacles, as a collision ends the game.
4. **Score Display:** As the player successfully navigates through obstacles, they earn points. The score is displayed either in real-time during gameplay or at the end of each session.
5. **Game Over:** When the bird collides with an obstacle, the game ends abruptly. A "Game Over" screen is displayed, showcasing the player's final score alongside options to restart the game or return to the main menu.

CHALLENGES

Precise Timing: The primary challenge is timing the taps accurately to keep the bird afloat and guide it through the narrow gaps between obstacles. The player must navigate the bird with precision to avoid collisions**.**

REWARDS

No In-Game Rewards: Unlike many modern games with extensive reward systems or unlockable content, Flappy Bird doesn't offer in-game rewards beyond the satisfaction of achieving a high score. The gameplay itself serves as the main reward, challenging players to improve and surpass their previous performances.

CHARACTER

In the game "Flappy Bird," the main character is a small bird. The bird is depicted as a simple, cartoonish character with a round body, small beak, and large, expressive eyes. Its appearance is minimalistic, featuring basic colors and simple animation. Key characteristics of the Flappy Bird character include:

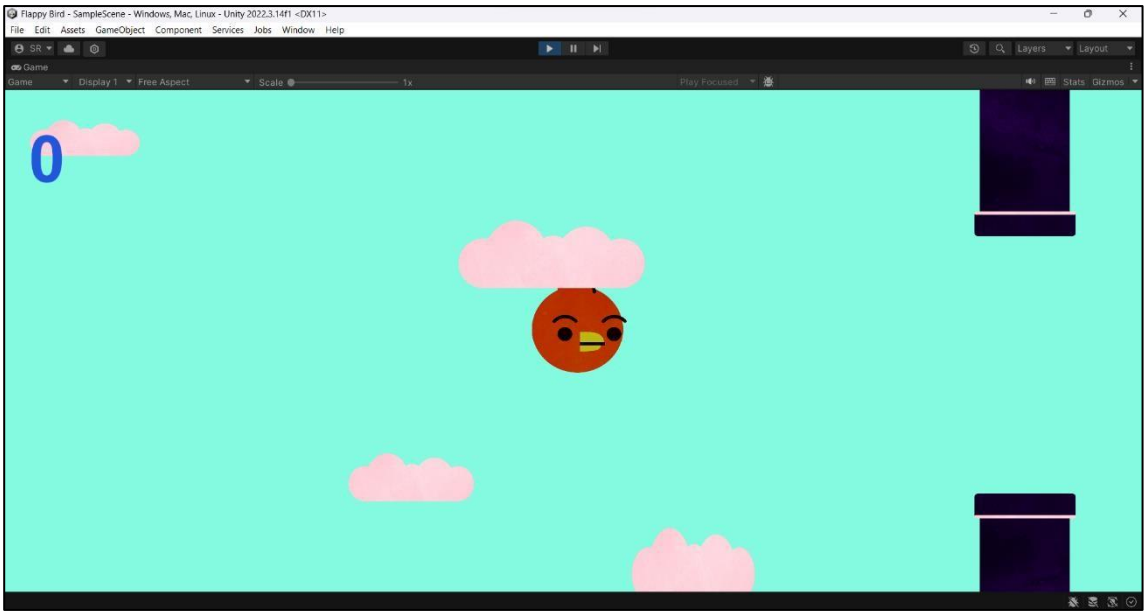
1. **Bird-like Attributes:** The character embodies the traits of a typical bird, albeit in a stylized and simplified manner.
2. **Vulnerability:** The bird appears vulnerable due to its small size and the perilous environment filled with obstacles. Its vulnerability adds to the challenge of the game, as players must skillfully guide the bird to prevent collisions.
3. **Endearing and Relatable:** Despite its simplicity, the character's design and movements evoke a sense of charm and relatability. Its struggle to fly through the challenging course resonates with players, contributing to the emotional investment in the game.
4. **Iconic Status:** The Flappy Bird character achieved iconic status within the gaming community, becoming instantly recognizable due to the game's immense popularity during its peak. Even with its straightforward design, the bird's image became widely associated with the game itself.

Overall, the Flappy Bird character's simplicity, vulnerability, and recognizable appearance played a significant role in the game's appeal and contributed to its lasting impact on the mobile gaming landscape.

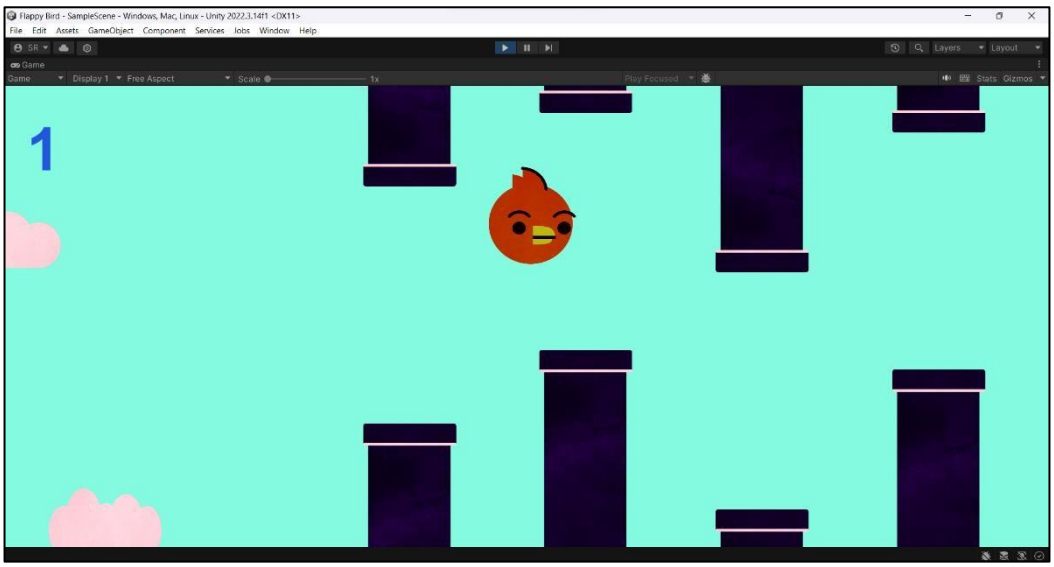
GAMEPLAY

The gameplay of Flappy Bird is simple yet challenging, focusing on navigating a bird through a series of obstacles. Here's a breakdown of the gameplay mechanics:

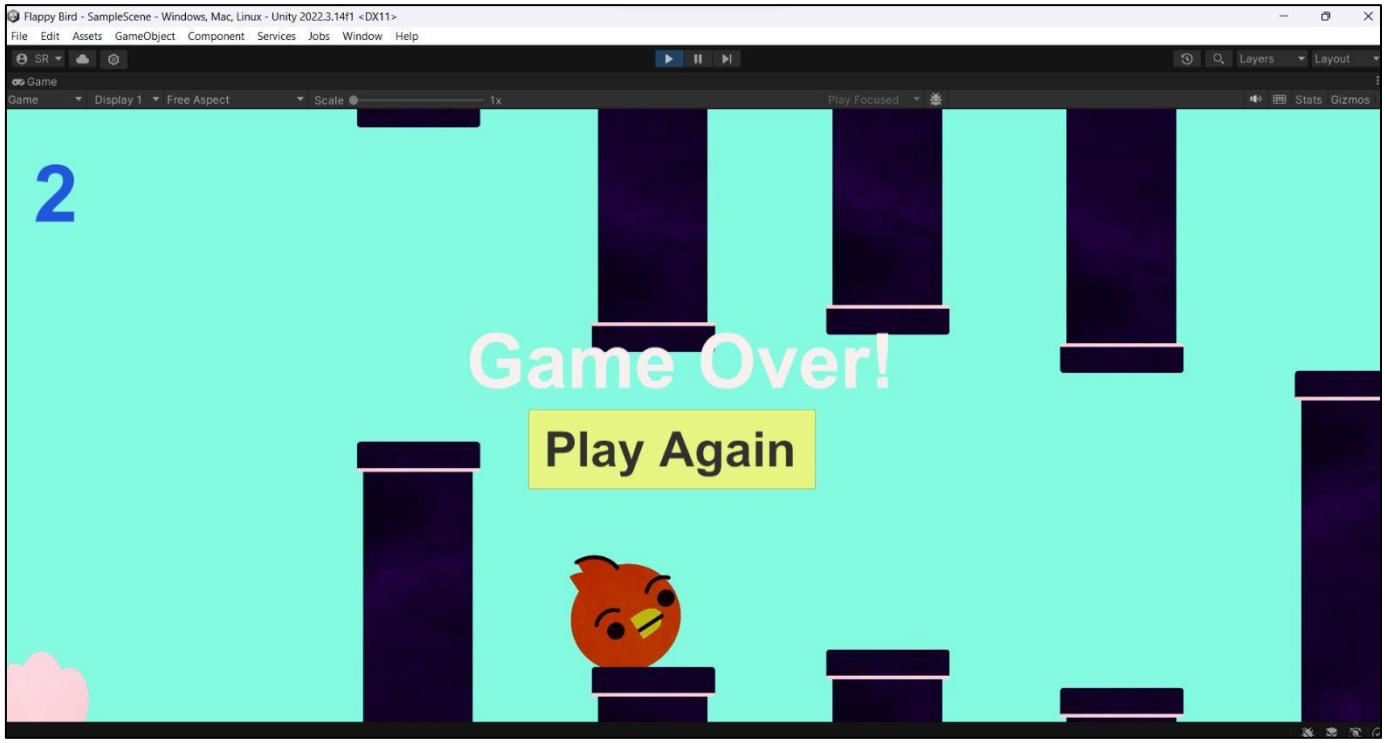
1. **Start and Control:** The game begins as the player taps the screen to start. The primary control mechanism involves tapping the screen (or clicking the mouse) to make the bird flap its wings. Each tap causes the bird to ascend momentarily.



2. **Flight and Obstacles:** The bird continuously moves horizontally from left to right across the screen. As it progresses, obstacles in the form of pipes or columns appear with gaps in between. The objective is to guide the bird through these gaps by tapping to adjust its altitude and prevent collisions.
3. **Timing and Precision:** The challenge lies in the precise timing of taps to maneuver the bird through the narrow openings between obstacles. Each successful pass through a gap earns the player points.
4. **Collision and Game Over:** Colliding with any part of the obstacles results in the game ending abruptly. Upon collision, the bird falls to the ground or hits an obstacle, triggering a "Game Over" screen.
5. **Scoring System:** Points are awarded for each obstacle successfully passed. The player's score is displayed either in real-time during gameplay or at the end of each session, encouraging players to compete for higher scores.

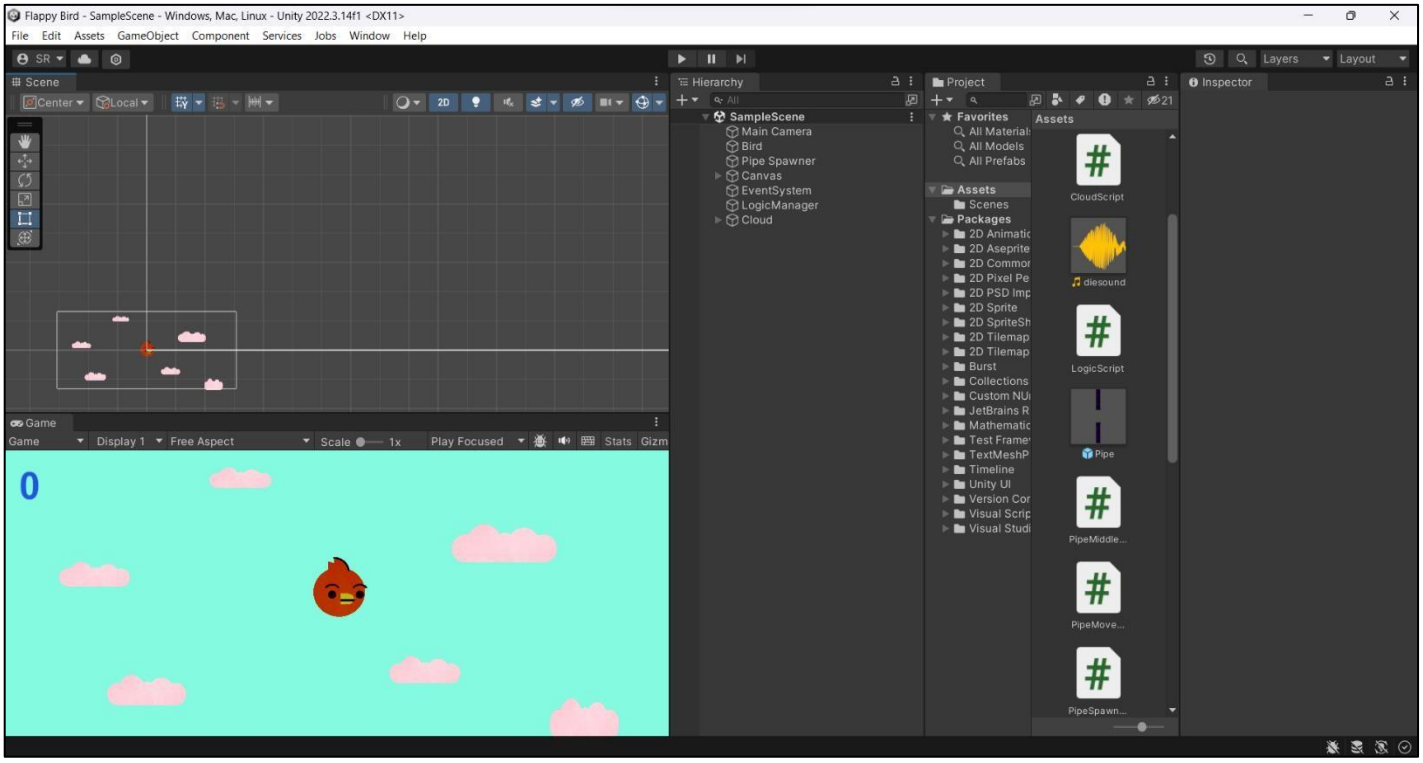


6. **Replayability:** Upon reaching a game-over state, the player is offered the option to restart the game instantly, prompting them to try again and improve their performance.



7. **Simple Visuals and Feedback:** The game features basic 2D graphics with a side-scrolling view. Visual and auditory feedback elements, such as sound effects for flapping wings and collision alerts, provide immediate feedback to the player during gameplay.

UNITY ENGINE DESIGN



C# SCRIPTS USED

1.Bird Script

This script is written in C# and is designed to control the behavior of a bird character in a 2D game using Unity game engine.

Main components in the script:

Rigidbody2D myRigidbody: A reference to the Rigidbody2D component attached to the GameObject. It is used to control the bird's physics, such as movement and collisions.

float flapStrength: Represents the force applied when the bird flaps/jumps. It's a configurable value in the Unity Editor.

LogicScript logic: A reference to another script named LogicScript, which presumably controls the game logic, such as scoring, game over conditions, etc.

bool birdIsAlive: A boolean variable that tracks whether the bird is alive or not. It starts as true.

Start(): Called at the beginning of the script execution. It finds the GameObject with the tag "Logic" and gets its LogicScript component.

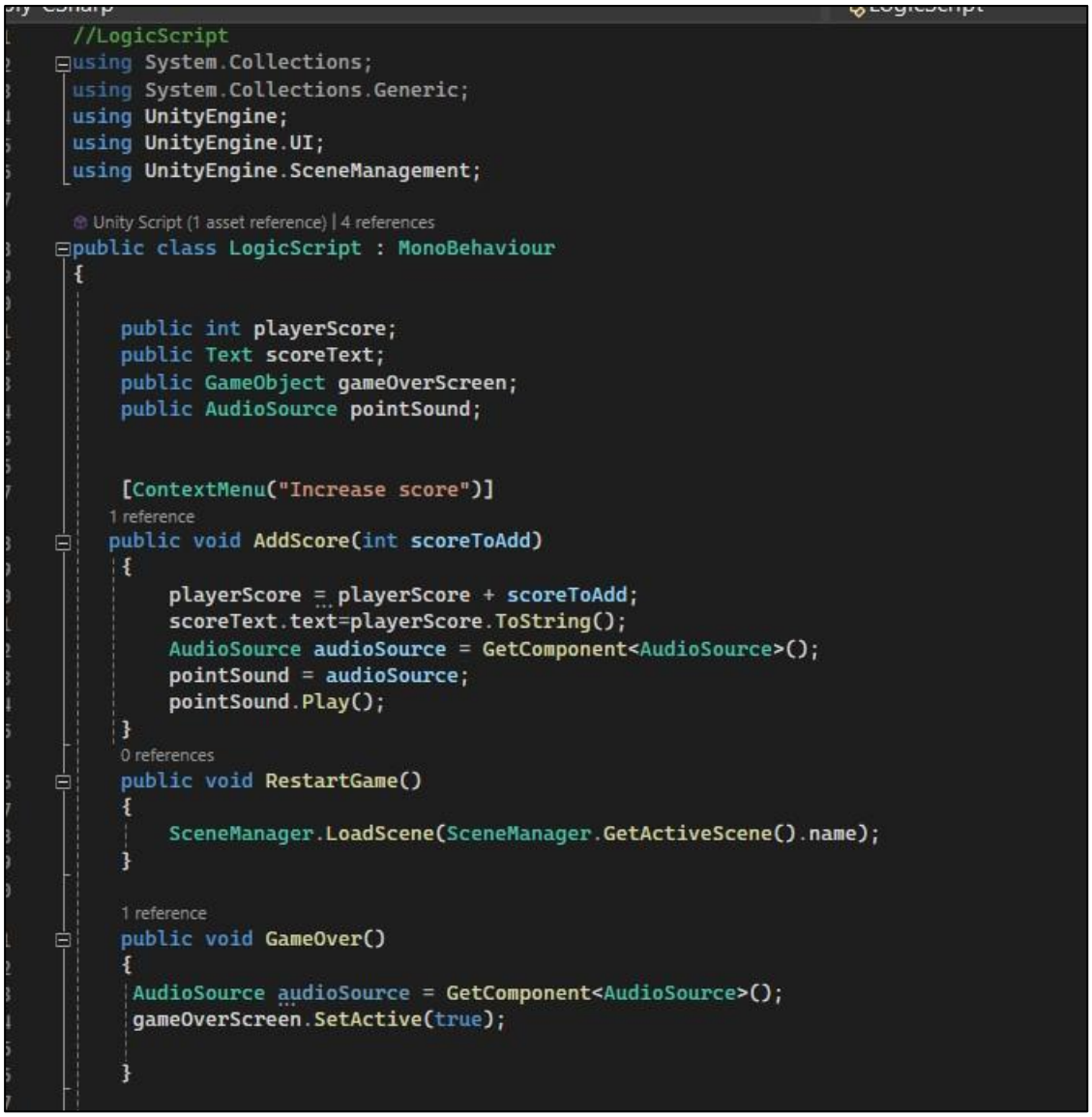
Update(): Called once per frame. It checks for player input (Space key press) and ensures the bird is alive before allowing it to flap/jump by changing its Rigidbody2D velocity.

OnCollisionEnter2D(Collision2D collision): Called when a collision occurs in 2D space. When the bird collides with something, it triggers the GameOver() function in the LogicScript and sets birdIsAlive to false.

Overall, this script manages the bird's movement and collision detection, affecting the game's logic through the LogicScript when collisions occur or when the player initiates a jump with the space key.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class BirdScript : MonoBehaviour
{

    public Rigidbody2D myRigidbody;
    public float flapStrength;
    public LogicScript logic;
    public bool birdIsAlive = true;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

        logic = GameObject.FindGameObjectWithTag("Logic").GetComponent<LogicScript>();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space) == true && birdIsAlive == true)
        {

            myRigidbody.velocity = Vector2.up * flapStrength;
        }

    }
    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D collision)
    {
        logic.GameOver();
        birdIsAlive = false;
    }
}
```

2.Logic Script



```
//LogicScript
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

Unity Script (1 asset reference) | 4 references
public class LogicScript : MonoBehaviour
{

    public int playerScore;
    public Text scoreText;
    public GameObject gameOverScreen;
    public AudioSource pointSound;


    [ContextMenu("Increase score")]
    1 reference
    public void AddScore(int scoreToAdd)
    {
        playerScore = playerScore + scoreToAdd;
        scoreText.text=playerScore.ToString();
        AudioSource audioSource = GetComponent<AudioSource>();
        pointSound = audioSource;
        pointSound.Play();
    }
    0 references
    public void RestartGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }

    1 reference
    public void GameOver()
    {
    AudioSource audioSource = GetComponent<AudioSource>();
    gameOverScreen.SetActive(true);

    }
```

This script, named **LogicScript**, manages various aspects of the game logic. Main components in the script:

int playerScore: Represents the player's score.

Text scoreText: A reference to a UI Text component used to display the player's score.

GameObject gameOverScreen: A reference to a UI element that displays the game over screen.

AudioSource pointSound: An audio source that plays a sound when the player earns points.

AddScore(int scoreToAdd): A function that increases the player's score by the specified amount (scoreToAdd). It updates the score display in the UI and plays a sound effect associated with earning points.

RestartGame(): Reloads the current scene, effectively restarting the game. It uses Unity's SceneManager to reload the scene by its name.

GameOver(): A function called when the game ends or when certain conditions are met. It activates the game over screen by setting gameOverScreen to active, presumably displaying it to the player.

The script manages scoring, restarting the game, and triggering the game over screen. It seems to interact with UI elements (such as Text and GameObject) to update the score and display game over information to the player.

3.Pipe Move Script

```
//PipeMoveScript
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class PipeMoveScript : MonoBehaviour
{
    public float moveSpeed = 5;

    public float deadZone = -45;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        transform.position = transform.position + (Vector3.left * moveSpeed * Time.deltaTime);
        if (transform.position.x<deadZone)
        {
            Debug.Log("Pipe deleted");
            Destroy(gameObject);
        }

    }
}
```

This script, named PipeMoveScript, controls the movement and deletion of a pipe GameObject within the game.

Main components in the script:

float moveSpeed: Represents the speed at which the pipe moves horizontally. The pipe moves towards the left (Vector3.left) at this speed.

float deadZone: Indicates the threshold position beyond which the pipe is considered to have moved too far and needs to be destroyed.

Start(): This method is called at the start of the script execution but currently doesn't contain any code.

Update(): Called once per frame. This method controls the movement of the pipe by adjusting its position based on the move speed. It also checks if the pipe's x-position is less than the deadZone. If the pipe moves beyond this threshold, it logs a debug message and destroys the pipe GameObject using Destroy(gameObject).

Overall, this script ensures that the pipe moves horizontally and gets deleted when it reaches a certain position (deadZone) to manage memory and prevent unnecessary game object accumulation outside the playable area.

4.Pipe Spawn Script

This script, named PipeSpawnScript, is responsible for spawning pipes at certain intervals within the game.

```
//PipeSpawnScript
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class PipeSpawnScript : MonoBehaviour
{
    public GameObject pipe;
    public float spawnRate = 2;
    private float spawnTime = 0;
    public float heightOffset = 10;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        SpawnPipe();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {   if (spawnTime < spawnRate)
        {

            spawnTime = spawnTime+Time.deltaTime;
        }
        else {

            SpawnPipe();
            spawnTime = 0;
        }

    }
    2 references
    void SpawnPipe()
    {
        float lowestPoint = transform.position.y - heightOffset;
        float highestPoint = transform.position.y + heightOffset;
        Instantiate(pipe,new Vector3(transform.position.x, Random.Range(lowestPoint, highestPoint), 0),transform.rotation);
    }
}
```

Main components in the script:

public GameObject pipe: Reference to the pipe prefab that will be spawned. This prefab needs to be assigned in the Unity Editor.

public float spawnRate: Represents the time interval between spawning pipes.

private float spawnTime = 0: A timer to keep track of time between pipe spawns.

public float heightOffset = 10: Determines the variation in height for the spawned pipes.

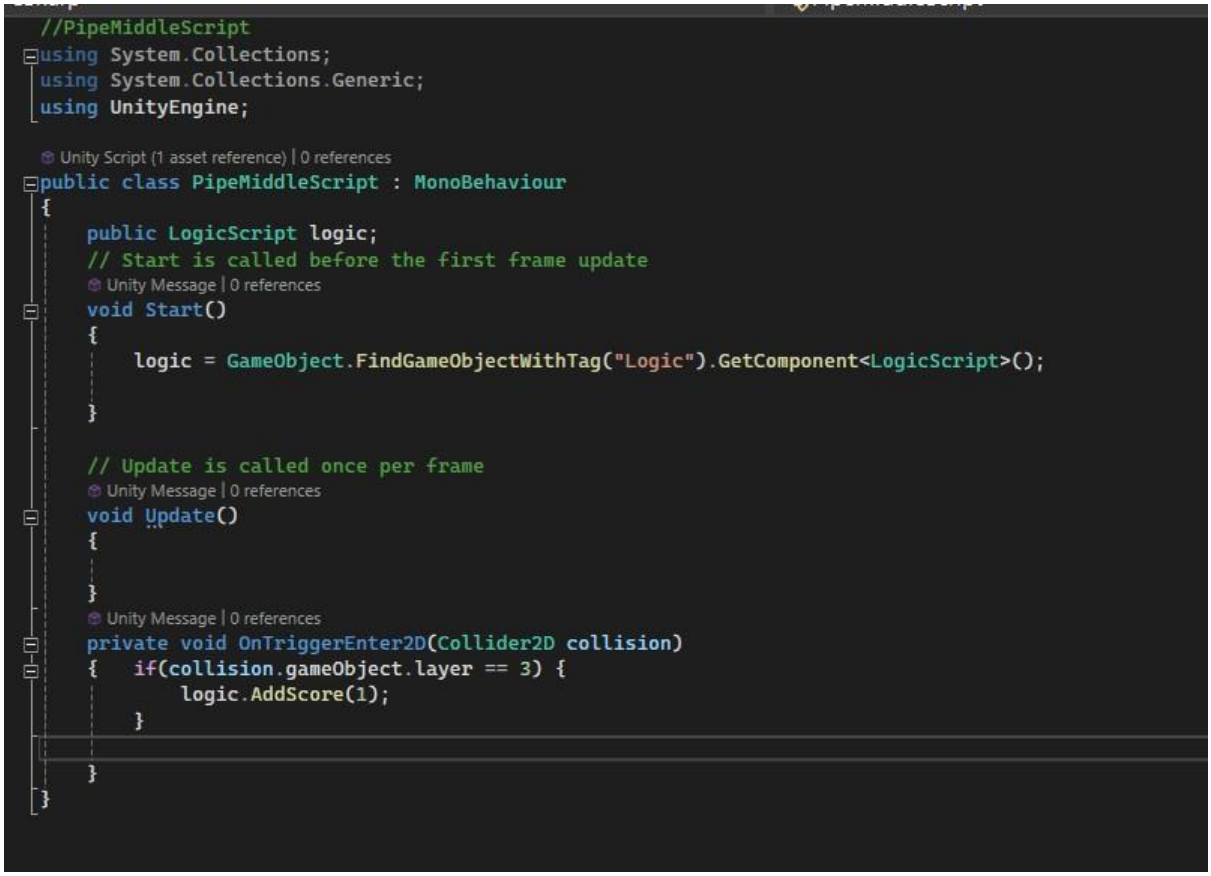
Start(): Called before the first frame update. It initiates the game by spawning a pipe immediately when the script starts.

Update(): Called once per frame. This method controls the timing for pipe spawning. If spawnTime is less than spawnRate, it increments spawnTime by the time passed in the frame. Once spawnTime reaches or exceeds spawnRate, it triggers the spawning of a new pipe using SpawnPipe() and resets the timer to 0.

SpawnPipe(): This method instantiates a new pipe prefab at a random height within the specified range (determined by lowestPoint and highestPoint variables).

Overall, this script manages the spawning of pipes at regular intervals and ensures they are placed at random heights within a specified range to create variety in the game environment

5.Pipe Middle Script



```
//PipeMiddleScript
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class PipeMiddleScript : MonoBehaviour
{
    public LogicScript logic;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        logic = GameObject.FindGameObjectWithTag("Logic").GetComponent<LogicScript>();

    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {

    }
    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {   if(collision.gameObject.layer == 3) {
            logic.AddScore(1);
        }

    }
}
```

This script, named PipeMiddleScript, manages the behavior of the middle part of the pipes in the game.

Main components in the script:

public LogicScript logic: A reference to the LogicScript used for managing game logic, particularly for scoring.

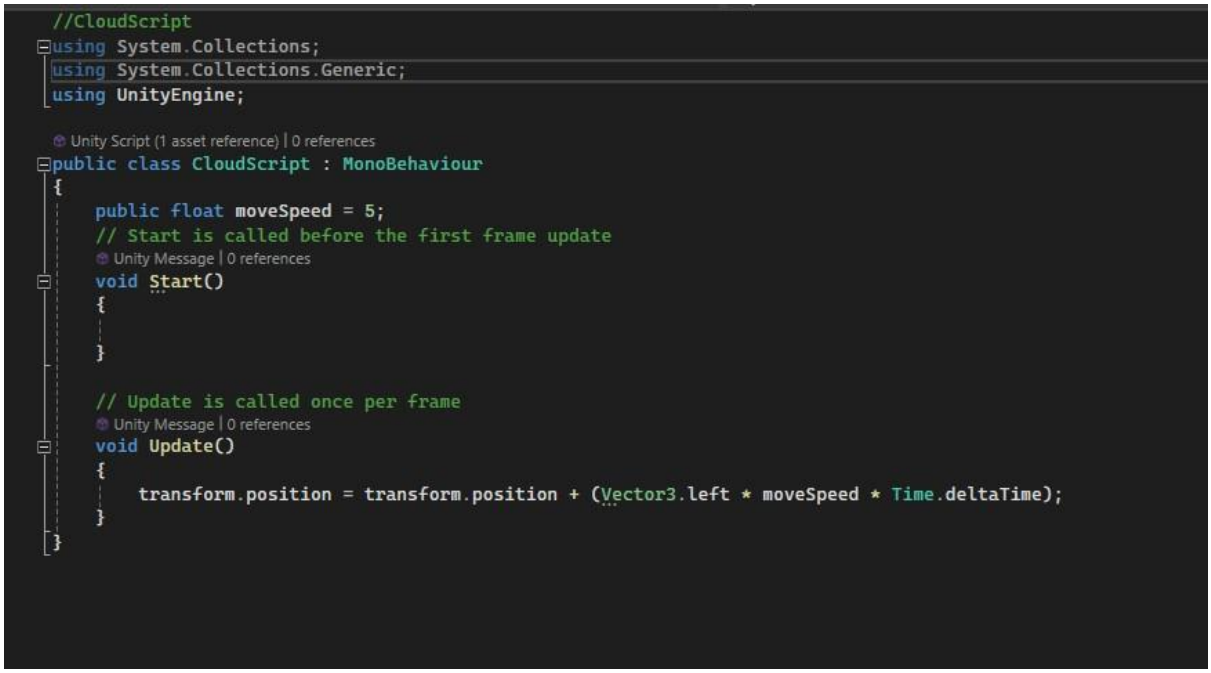
Start(): Called before the first frame update. It finds the GameObject with the tag "Logic" and gets its LogicScript component to establish a reference.

Update(): Called once per frame. Currently, this method doesn't contain any code.

OnTriggerEnter2D(Collider2D collision): Called when another Collider2D enters this collider trigger. In this case, it checks if the object that entered the trigger has a collider on the layer with index 3. If the condition is met, it assumes it's the player or another element that triggers a score increase, and it calls the AddScore() function from the LogicScript to increase the score by 1.

This script manages the logic for scoring when the bird collides with the middle part of the pipes, triggering an increment in the game score by calling the AddScore() function from the LogicScript.

6.Cloud Script



```
//CloudScript
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class CloudScript : MonoBehaviour
{
    public float moveSpeed = 5;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        transform.position = transform.position + (Vector3.left * moveSpeed * Time.deltaTime);
    }
}
```

This script, named CloudScript, controls the movement of cloud GameObjects within the game.

Main components in the script:

float moveSpeed: Represents the speed at which the cloud moves horizontally. The cloud moves towards the left (Vector3.left) at this speed.

Start(): This method is called at the start of the script execution but currently doesn't contain any code.

Update(): Called once per frame. This method controls the movement of the cloud by adjusting its position based on the move speed. It updates the cloud's position in the scene by adding a movement towards the left (Vector3.left) at a speed multiplied by Time.deltaTime. This multiplication by Time.deltaTime ensures smooth movement regardless of frame rate, as it makes the movement frame-rate independent.
Overall, this script enables the continuous horizontal movement of cloud GameObjects in the game scene, simulating a scrolling effect by moving towards the left at a specified speed.