# Week 12: Scoring Mechanism and End Condition

# Welcome to Week 12!

## Scoring Mechanism and End Condition

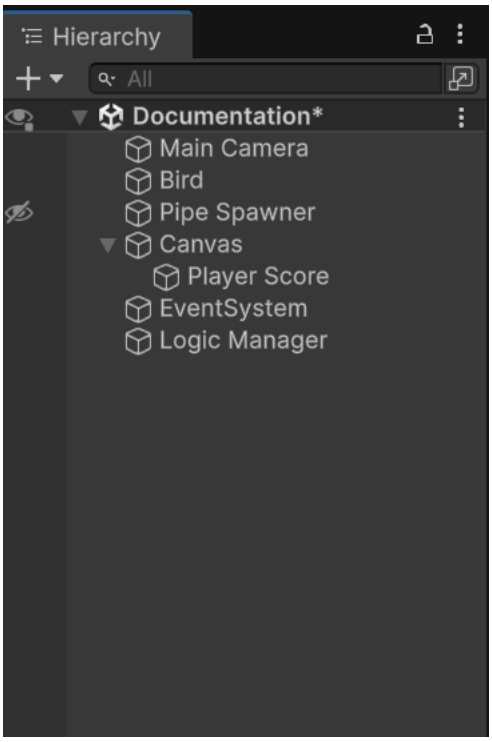


### 12.1—Introduction

This week continues development of the Unity project begun previously, with an emphasis on game progression and completion mechanics. The focus is on implementing a scoring system, detecting when the game ends, and allowing the player to restart the game. These systems work together to provide feedback, challenge, and closure to the gameplay experience.

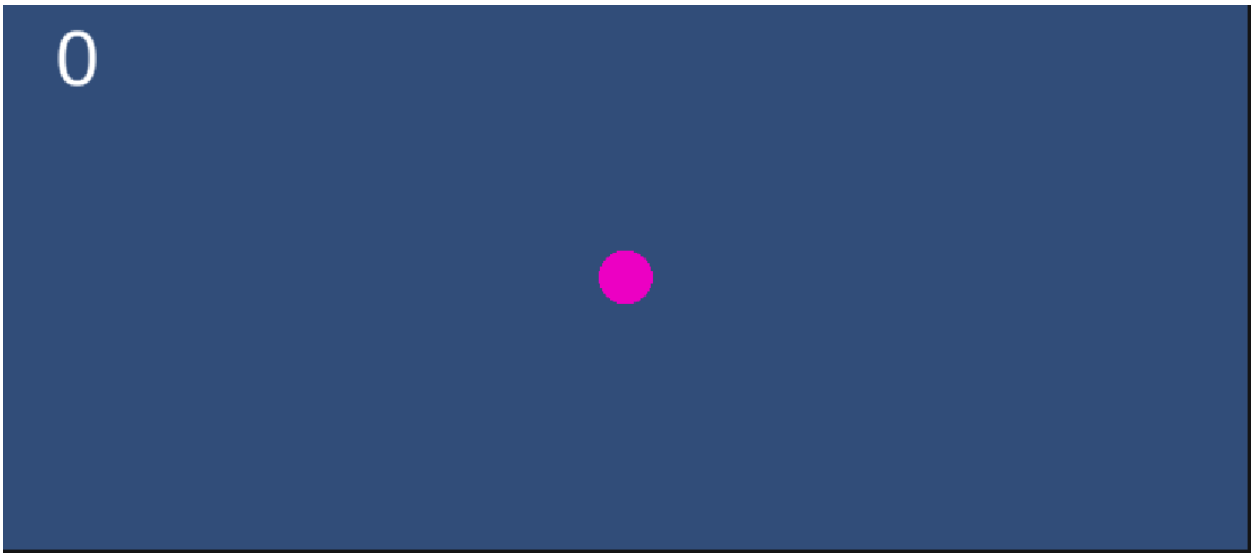
### 12.2—Score UI

The scoring interface begins with the addition of a Text (TMP) UI element, created by selecting GameObject -> UI -> Text – TextMeshPro. When this object is added, Unity automatically creates a Canvas and places the text object as its child. All user interface elements must exist within a Canvas to render correctly on screen. The Text (TMP) object is renamed “Player Score” for clarity.



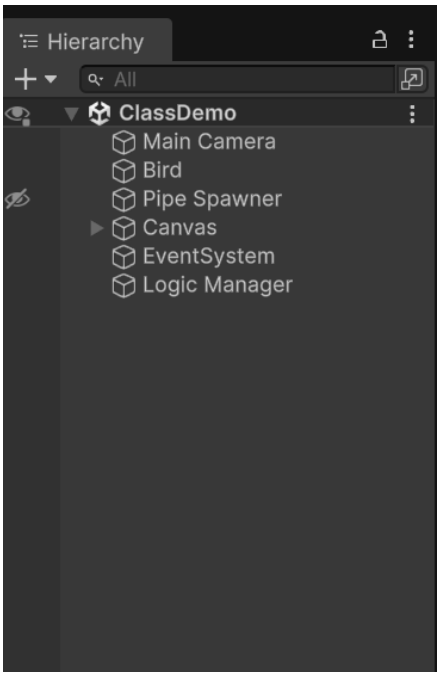

With the Canvas selected, the Canvas Scaler component is adjusted so that the UI Scale Mode property is set to “Scale With Screen Size.” This setting ensures that UI elements maintain consistent proportions across different screen sizes and aspect ratios. The Reference Resolution is then set to 1920 x 1080, which serves as the baseline resolution for scaling calculations.

Next, the Player Score text object is selected in the Hierarchy. In the TextMeshPro—Text (UI) component, the default text value is changing from “New Text” to 0, indicating an initial score of zero. The font size is increased to ensure readability during gameplay. The position of the score display is adjusted using the Rect Transform component so that it appears in the upper-right corner of the screen. The easiest way to see these changes are in the Game view rather than the Scene view.




## 12.3—Logic Manager

An empty GameObject named “Logic Manager” is added to the scene. This object serves as a centralized controller for game rules and flow. Often referred to as a Game Manager, this type of object is responsible for tracking score, handling win or loss conditions, controlling UI states, managing transitions such as restarting the game, and more. Centralizing this logic helps keep individual scripts simpler and easier to maintain.




## 11.4—Logic Manager Scripting

A new C# script named “LogicManager” is created and attached to the Logic Manager GameObject. Unlike earlier scripts, the default template methods are removed to allow only relevant functionality to be defined.

Two variable are added to the script: an integer named “playerScore” to store the current score, and a TMP_Text variable named “scoreText” to reference the UI text element that displays the score. Adding the TMP_Text variable requires including the TMPro namespace at the top of the script using a using directive. Using directives allow access to classes within a namespace without repeatedly writing the full namespace path.

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using NUnit.Framework;

public class LogicManager : MonoBehaviour
{
    public int playerScore;
    public TMP_Text scoreText;
}
```

After saving the script, the Player Score Text (TMP) object is dragged from the Hierarchy into the scoreText field in the LogicManager component within the Inspector. This creates a reference between the script and the UI element.

A function named addScore() is added to the script. This method increases the player’s score and updates the score display. The score value is incremented and converted to a string before being assigned to scoreText.text, since UI text fields only accept character and string values.

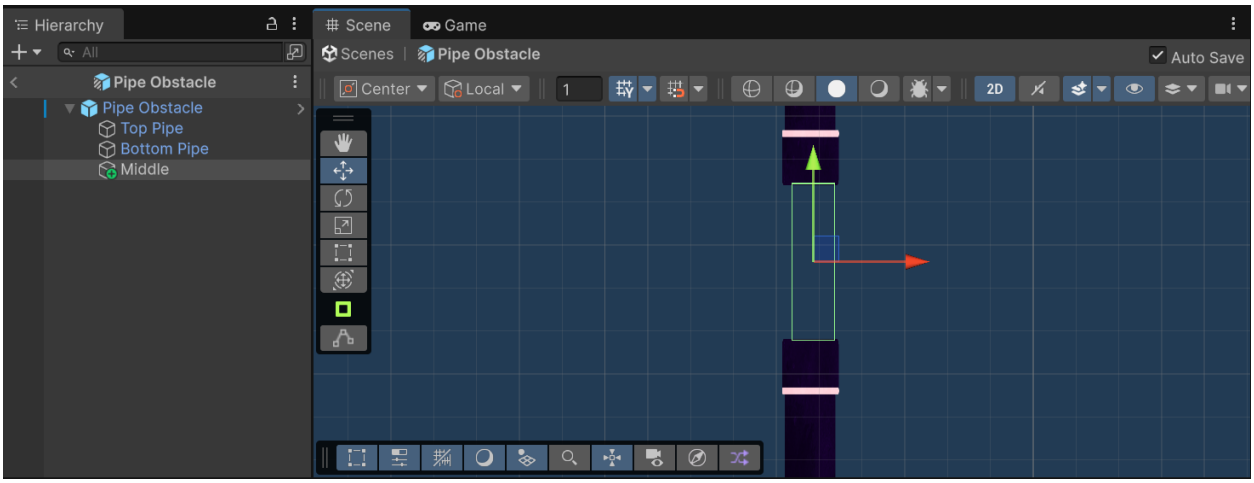
To allow testing without playing the game, a ContextMenu attribute is added above the function. This creates an option in the Inspector that manually triggers the function.

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using NUnit.Framework;

public class LogicScript : MonoBehaviour
{
    public int playerScore;
    public TMP_Text scoreText;

    [ContextMenu("Increase Score")]
    public void addScore()
    {
        scoreText.text = playerScore.toString();
    }
}
```

## 12.5—Pipe Collision

The Pipe Obstacle prefab is opened by double-clicking it in the Assets folder. Entering Prefab Mode is indicated by the editor background changing color and the Hierarchy displaying only the prefab's internal objects.

An empty child GameObject named "Middle" is added to the prefab. A Box Collider 2D component is attached, and Is Trigger option is checked. The collider is resized and positioned so that it fills the gap between the top and bottom pipes. This trigger zone detects when the bird successfully passes through the obstacle.




## 12.6—Pipe Middle (Gap) scripting

A new C# script named "PipeMiddle" is created and attached to the Middle GameObject. This script is responsible for detecting when the bird successfully passes through the gap between the pipes and informing the Logic Manager to increase the player's score. By placing this responsibility on the Middle GameObject, the scoring event is tied directly to the trigger area located between the top and bottom pipes.

In the script, a public variable with LogicManager data type called "logic" is declared. After saving the script and returning to the Unity Interface, dragging the Logic Manager GameObject into the logic property doesn't work. Unity does not automatically connect scripts that resides on separate GameObjects. In order for one script to communicate with another on a different GameObject, a reference must be established through code.

To establish communication between the two scripts, a reference must be created through code. This is accomplished by assigning a unique tag to the Logic Manager GameObject. In the Inspector panel, a new tag named GameLogic is created and applied to the Logic Manager GameObject. Once this tag has been assigned, the Start() method in the PipeMiddle script is updated to locate the Logic Manager at runtime. The method GameObject.FindGameObjectWithTag("GameLogic") searches the scene and returns the GameObject that carries the specified tag.  After obtaining this GameObject reference, the

GetComponent<LogicScript>() method retrieves the LogicScript component attached to it. This component is then assigned to the logic variable, successfully linking the PipeMiddle script to the Logic Manager and enabling communication between them.

```
using UnityEngine;

public class PipeMiddleScript : MonoBehaviour
{
    public classLogicScript logic;
     void Start()
     {
       logic =
GameObject.FindGameObjectWithTag("GameLogic").GetComponent<LogicScript>();
     }
    void Update()
    {

    }
}
```

## 12.7—Ontriggerenter2d method

An OnTriggerEnter2D() function is added to the PipeMiddle script. This method runs automatically when another collider enters the trigger zone. To ensure only the bird increases the score, the bird is assigned to a dedicated Bird layer.

The method checks whether the colliding object belongs to the Bird layer. If so, the score is increased through the Logic Manager.

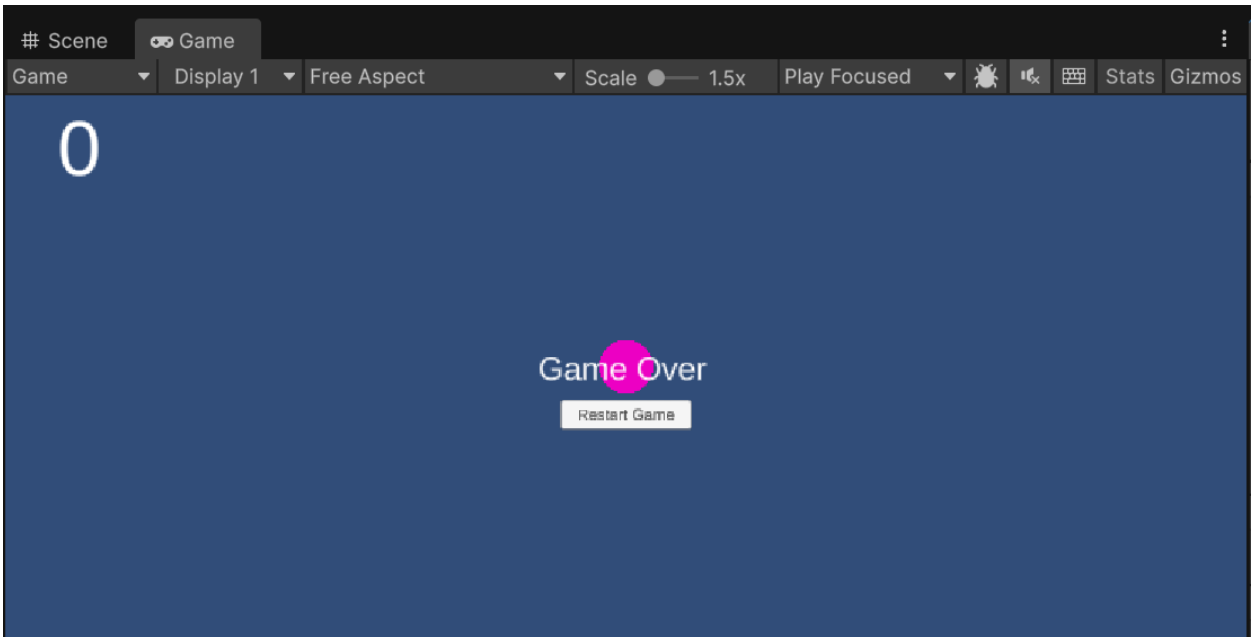
```
private void OnTriggerEnter2D(Collider2D collision)
 {
     if (collision.gameObject.layer == 3)
     {
         Debug.Log("The player that passed through is: "
+(collision.gameObject.name).ToString());
         logic.addScore(1);
     }
 }
```

## 12.8—Game Over UI

To create the game over interface, an empty GameObject named “Game Over Screen” is added as a child under the Canvas. This will serve as a container for all user interface elements that should appear when the game ends.

A Text (TMP) GameObject is added as a child of the Game Over Screen GameObject. In the TextMeshPro-Text (UI) component, the displayed text is changed to “Game Over.” The Rect Transform component is adjusted by modifying the Pos X and Pos Y values so that the text is centered on the screen.

A Button GameObject is added as another child of the Game Over Screen to allow the player to restart the game. The button's position and size are customized by changing the Pos X, Pos Y, Width, and Height values in the Rect Transform component to ensure it appears below the Game Over text.




## 12.9—Button Scripting

Restart logic is added directly to the Logic Manager rather than creating a separate script. A method named restartGame() is defined, which reloads the current scene using SceneManager.LoadScene().

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using NUnit.Framework;

public class LogicScript : MonoBehaviour
{
    public int playerScore;
    public TMP_Text scoreText;

    public void addScore()
    {
        playerScore+=1;
        scoreText.text = playerScore.toString();
    }
    public void restartGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

In the Button's OnClick() function, the Logic Manager is assigned as the target object, and the restartGame() function is selected from the function list.

To control visibility of the Game Over screen, a GameObject variable named gameOverScreen is added to the Logic Manager. A gameOver() function activates this screen when called.

```
public void gameOver()
    {
        gameOverScreen.SetActive(true);
    }
```

The Game Over Screen should be initially disabled in the Hierarchy so it does not appear during normal gameplay.

## 12.10—Modifying Bird Script

The Bird script is updated so that the Rigidbody2D does not begin simulating physics immediately when the scene loads. In the Start() function, the Rigidbody2D's simulated property is set to false.

```
private void Start()
    {
        myRigidbody.simulated = false;
    }
```

An OnCollisionEnter2D() function is then added to the detect collisions with obstacles. If the collision is not with an object on the Bird layer, player input is disabled and the gameOver() method is called on the Logic Manager.

```
 private void OnCollisionEnter2D(Collision2D collision)
 {
     if (collision.gameObject.layer != 3)
     {
         OnDisable();
         logic.gameOver();
     }
 }
```

With these systems in place, the game now tracks score, detects successful navigation through obstacles, ends the game upon collision, and allows for a full restart through the UI.

## 12.11—Exercises

### Reinforcement

> The reinforcement section is used to check the understanding of the information that was presented in the PowerPoint/document. Please answer the questions in your own words.

R—12.1 Why must all UI elements in Unity be placed inside a Canvas in order to appear on screen?

R—12.2 What is the purpose of setting the Canvas Scaler's UI Scale Mode to "Scale With Screen Size"? What is a Reference Resolution of 1920 x 10180 used, and how does it affect UI scaling?

R—12.3 What is the role of the Logic Manager (or Game Manager) in a Unity project?

R—12.4 Why is centralizing game logic in a dedicated GameObject considered good design practice?

R—12.5 How does ContextMenu attribute assist in testing functionality during development?

R—12.6 Why can scripts on different GameObject not automatically communicate with each other?

R—12.7 How does checking the collision object's layer help control when the score increases?

R—12.8 How does the Button's OnClick() event connect the UI to the Logic Manager?

R—12.9 How do the scoring system, collision detection, and restart functionality work together to complete the gameplay loop?

***Project***

> The project section's tasks are used in work toward creating the final Flying Bird game project for the end of the course.

P—12.1 Implement a scoring UI by creating a TextMeshPro text element that displays the player's score. Configure the Canvas to scale with screen size, set an appropriate reference resolution, initialize the score display to 0, and position it in the upper-right corner of the screen.

P—12.2 Implement a Logic Manager class that maintains and updates the player's score. Define variables for storing the score and referencing the UI text element and implement a method that increments the score and updates the displayed value according.

P—12.3 Implement a trigger-based scoring mechanism within the Pipe Obstacles prefab by adding a child GameObject with a Box Collider 2D configured as a trigger. Attach a script that detects when the bird passes through the gap and invokes the Logic Manager's score update method.