Lab 3

Programming: Flappy Bird

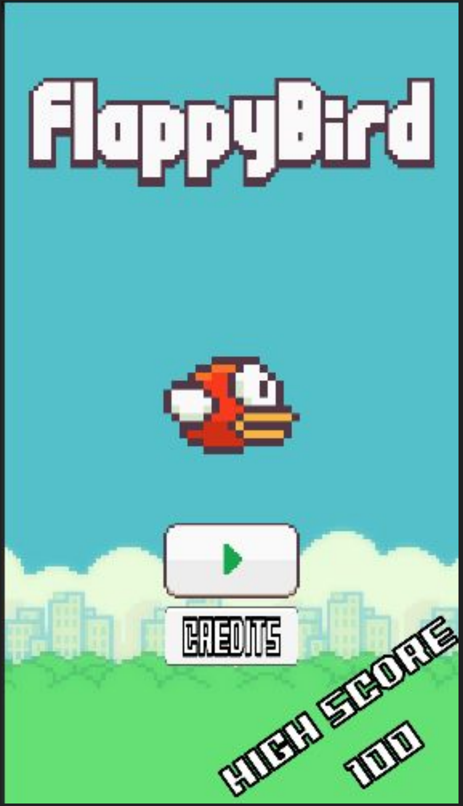
Lab Goals

In this lab, you will finish the Flappy Bird game by programming its functionality, from moving between scenes to keeping track of the score and playing the game.

Menu Scene

We will start with the Menu Scene, and there are three things you need to do:

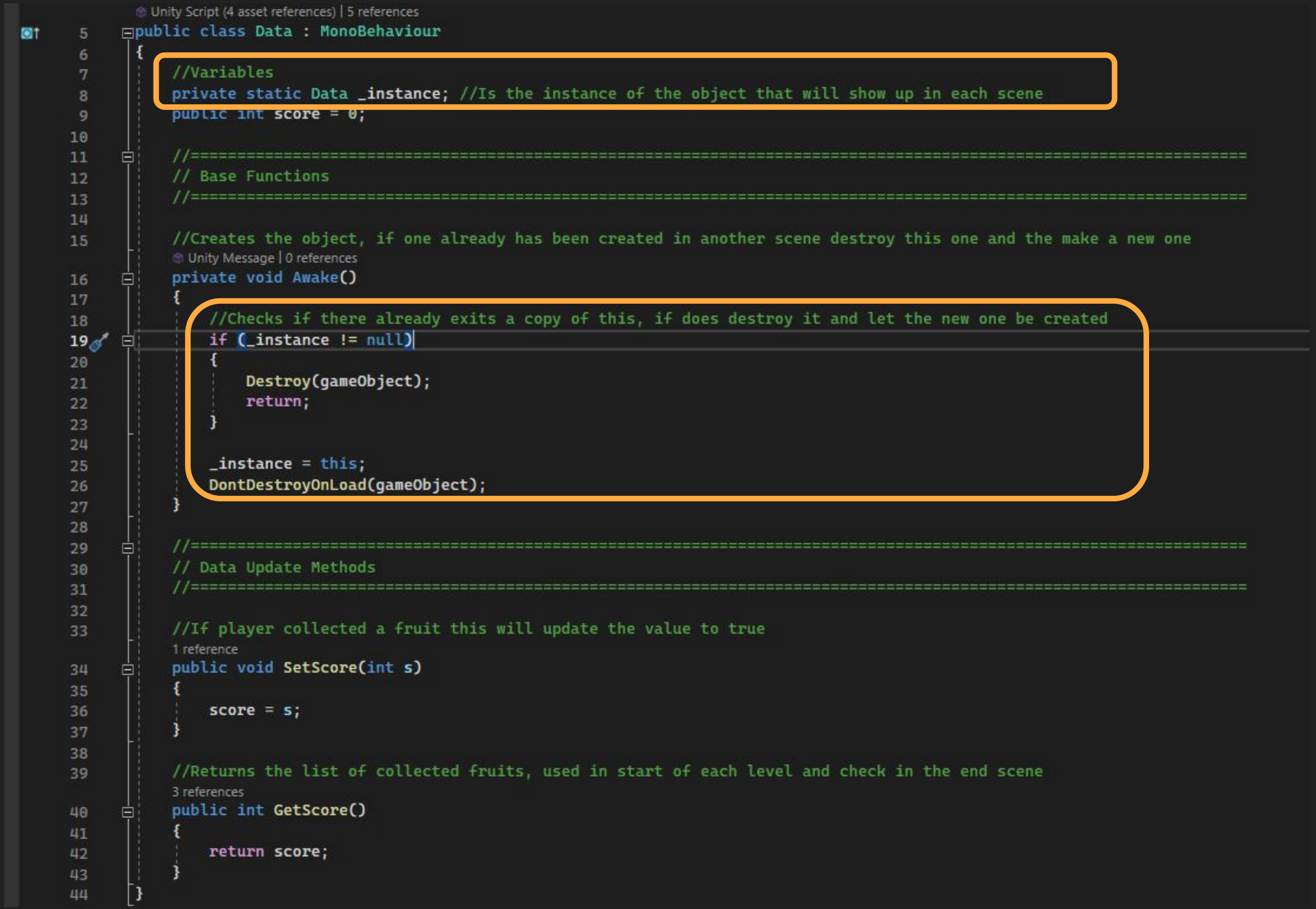
1. Create a MenuScript that will allow you to move between the Menu Scene and the Game when you press the play button.
2. Create a Data Script that will keep track of the player's highest score between the scenes.



Data

You will create an Awake function, which means that any time the object the script is attached to is turned on, it will perform the specified action. Inside this function, you will write the highlighted code.

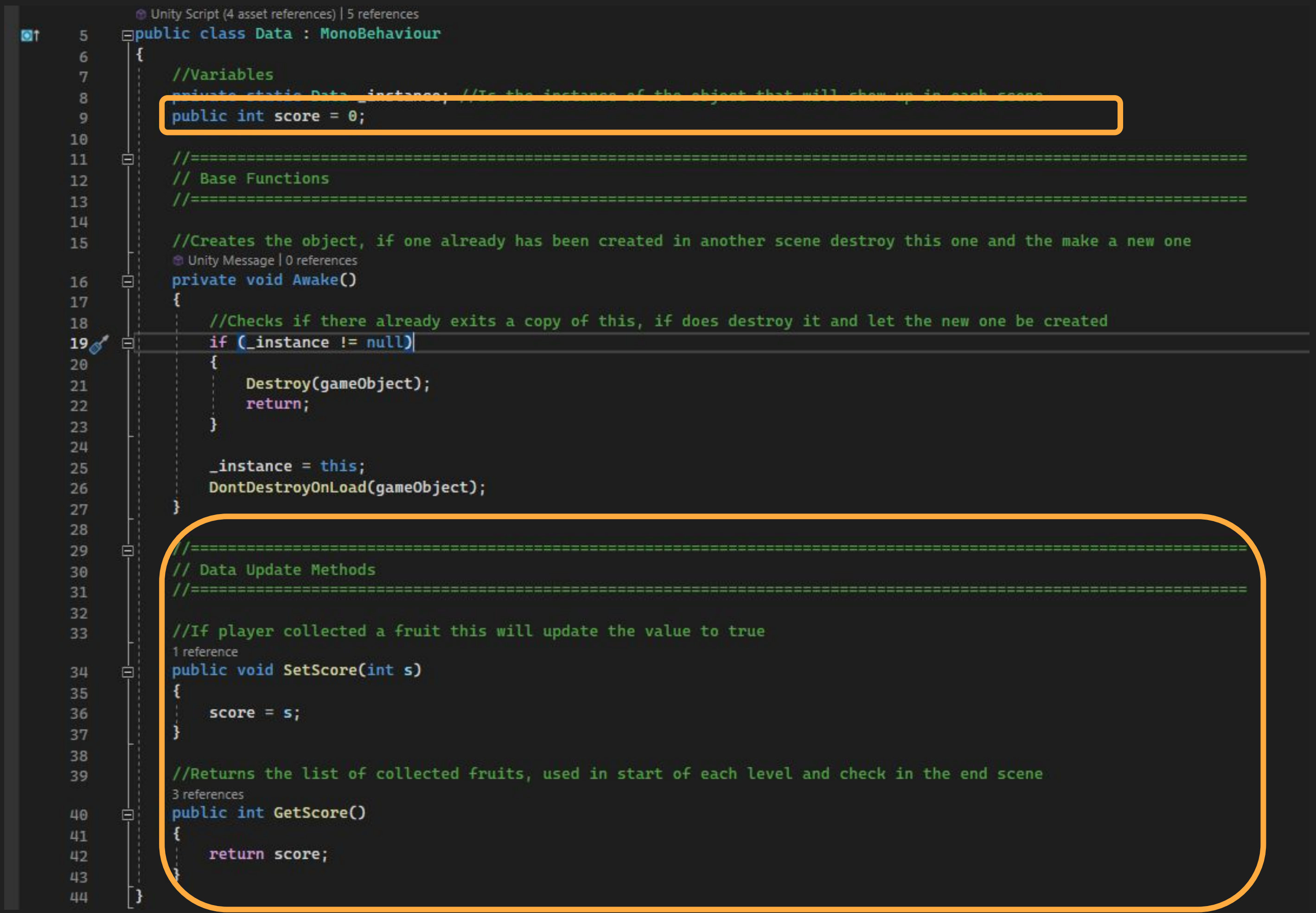
This code is responsible for preserving the game object between scenes and serving as storage for the high score.



```
Unity Script (4 asset references) | 5 references
public class Data : MonoBehaviour
{
    //Variables
    private static Data _instance; //Is the instance of the object that will show up in each scene
    public int score = 0;

    //===========================================================================================================
    // Base Functions
    //===========================================================================================================

    //Creates the object, if one already has been created in another scene destroy this one and the make a new one
    Unity Message | 0 references
    private void Awake()
    {
        //Checks if there already exits a copy of this, if does destroy it and let the new one be created
        if (_instance != null)
        {
            Destroy(gameObject);
            return;
        }

        _instance = this;
        DontDestroyOnLoad(gameObject);
    }

    //===========================================================================================================
    // Data Update Methods
    //===========================================================================================================

    //If player collected a fruit this will update the value to true
    1 reference
    public void SetScore(int s)
    {
        score = s;
    }

    //Returns the list of collected fruits, used in start of each level and check in the end scene
    3 references
    public int GetScore()
    {
        return score;
    }
}
```

Data

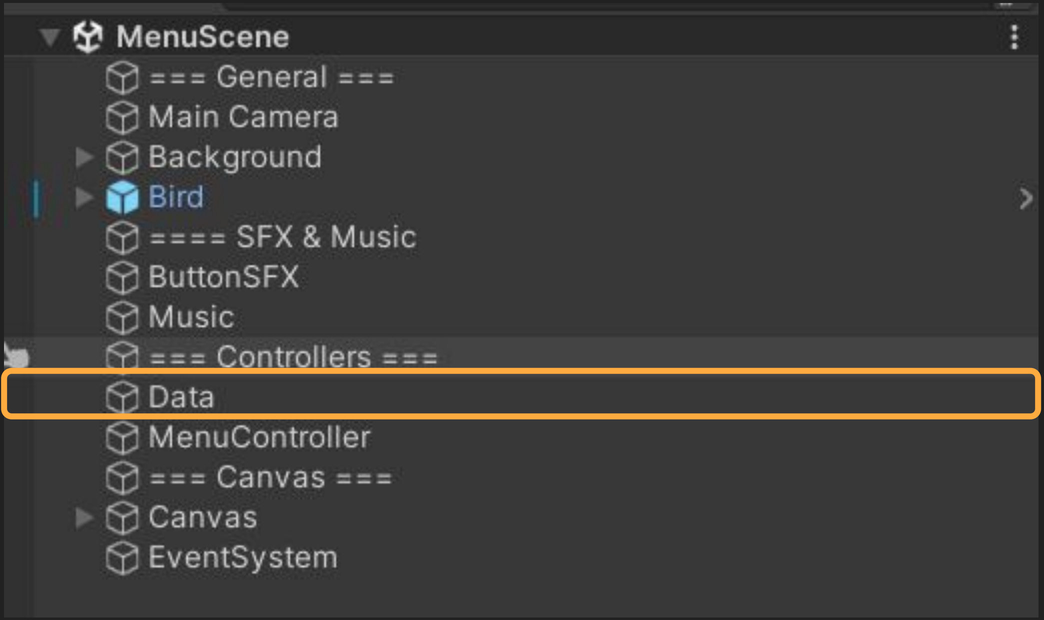
Next, you are going to create the 'score' variable and 'Set' and 'Get' functions. These functions will be called by other scripts to update the text shown in the main menu.



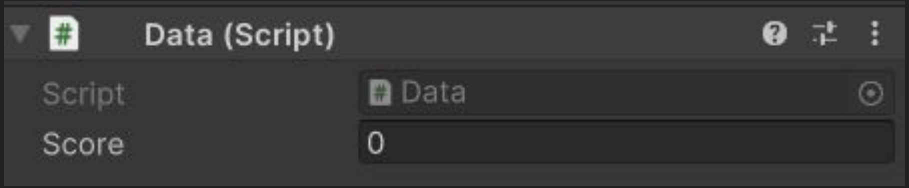
```
Unity Script (4 asset references) | 5 references
public class Data : MonoBehaviour
{
    //Variables
    private static Data _instance; //Is the instance of the object that will show up in each scene
    public int score = 0;

    //==========================================================================================================
    // Base Functions
    //==========================================================================================================

    //Creates the object, if one already has been created in another scene destroy this one and the make a new one
    Unity Message | 0 references
    private void Awake()
    {
        //Checks if there already exits a copy of this, if does destroy it and let the new one be created
        if (_instance != null)
        {
            Destroy(gameObject);
            return;
        }

        _instance = this;
        DontDestroyOnLoad(gameObject);
    }

    //==========================================================================================================
    // Data Update Methods
    //==========================================================================================================

    //If player collected a fruit this will update the value to true
    1 reference
    public void SetScore(int s)
    {
        score = s;
    }

    //Returns the list of collected fruits, used in start of each level and check in the end scene
    3 references
    public int GetScore()
    {
        return score;
    }
}
```

Data

Once you have created and saved your script, you will attach it to the 'Data' game object in the Menu Scene. When you click 'play,' you will notice that the game object is moved to its own subsection where it won't be destroyed when loaded into the next scene.

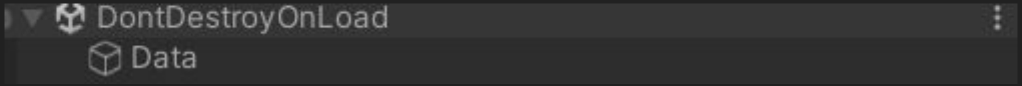


MenuScene
=== General ===
Main Camera
Background
Bird
==== SFX & Music
ButtonSFX
Music
=== Controllers ===
Data
MenuController
=== Canvas ===
Canvas
EventSystem



Data (Script)

Script	Data
Score	0

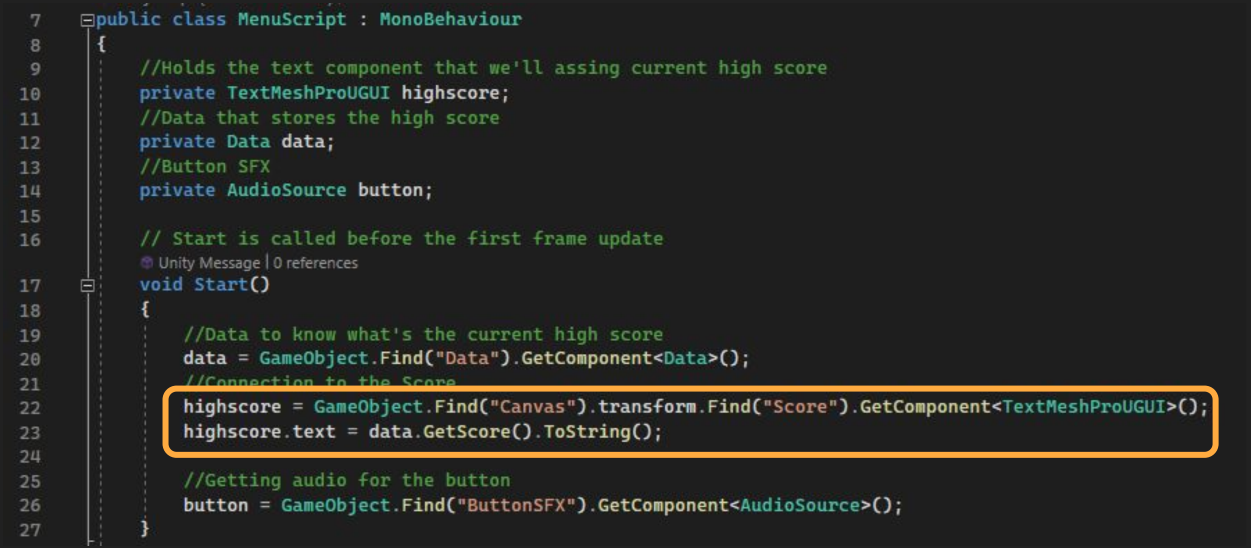


DontDestroyOnLoad
Data

Menu Script

In the menu script, we will begin by setting up variables and connecting them to game objects and components. Here, we are using the GameObject.Find function to search the hierarchy for the 'Data,' 'Score,' and 'ButtonSFX' game objects so that we can use the scripts that are attached to them.

Once we have attached them, you can see that we are using the GetScore function we created in the 'Data' script to set the value of the TextMeshPro.



```
public class MenuScript : MonoBehaviour
{
    //Holds the text component that we'll assing current high score
    private TextMeshProUGUI highscore;
    //Data that stores the high score
    private Data data;
    //Button SFX
    private AudioSource button;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        //Data to know what's the current high score
        data = GameObject.Find("Data").GetComponent<Data>();
        //Connection to the Score
        highscore = GameObject.Find("Canvas").transform.Find("Score").GetComponent<TextMeshProUGUI>();
        highscore.text = data.GetScore().ToString();

        //Getting audio for the button
        button = GameObject.Find("ButtonSFX").GetComponent<AudioSource>();
    }
```

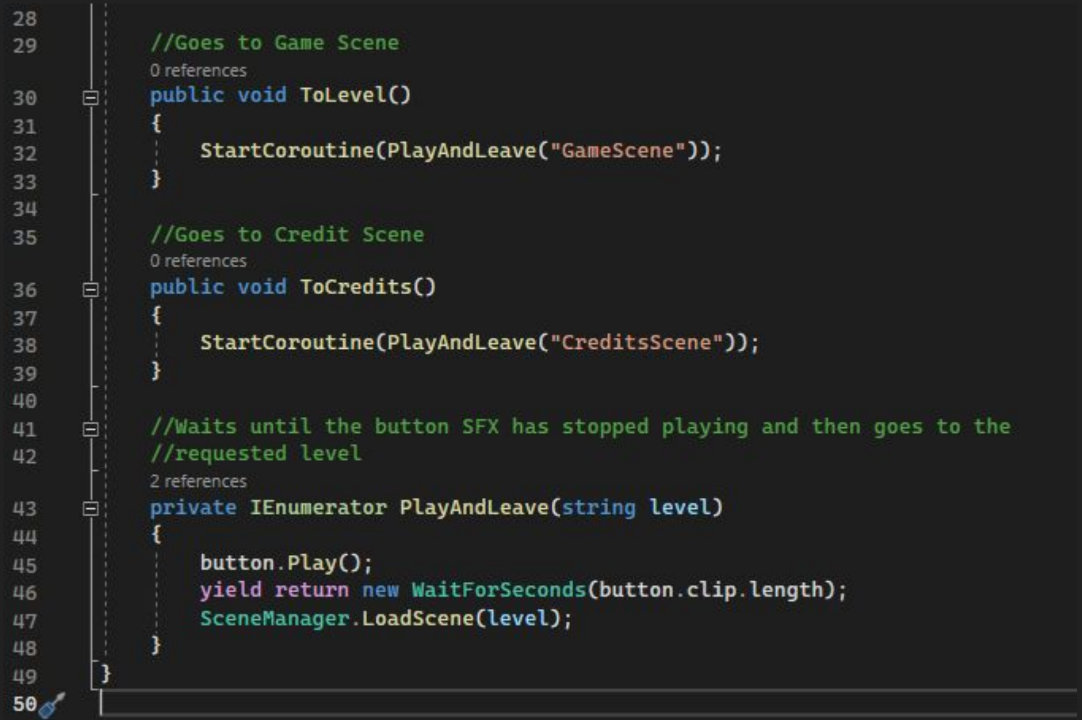
Menu Script

Once we have all the components set up we start using them.

We create two functions ToLevel and ToCredits that will be used by buttons.

You will notice that they are using something called Coroutine. That means we're splitting the code and the game will continue to run while we execute this code on the side.

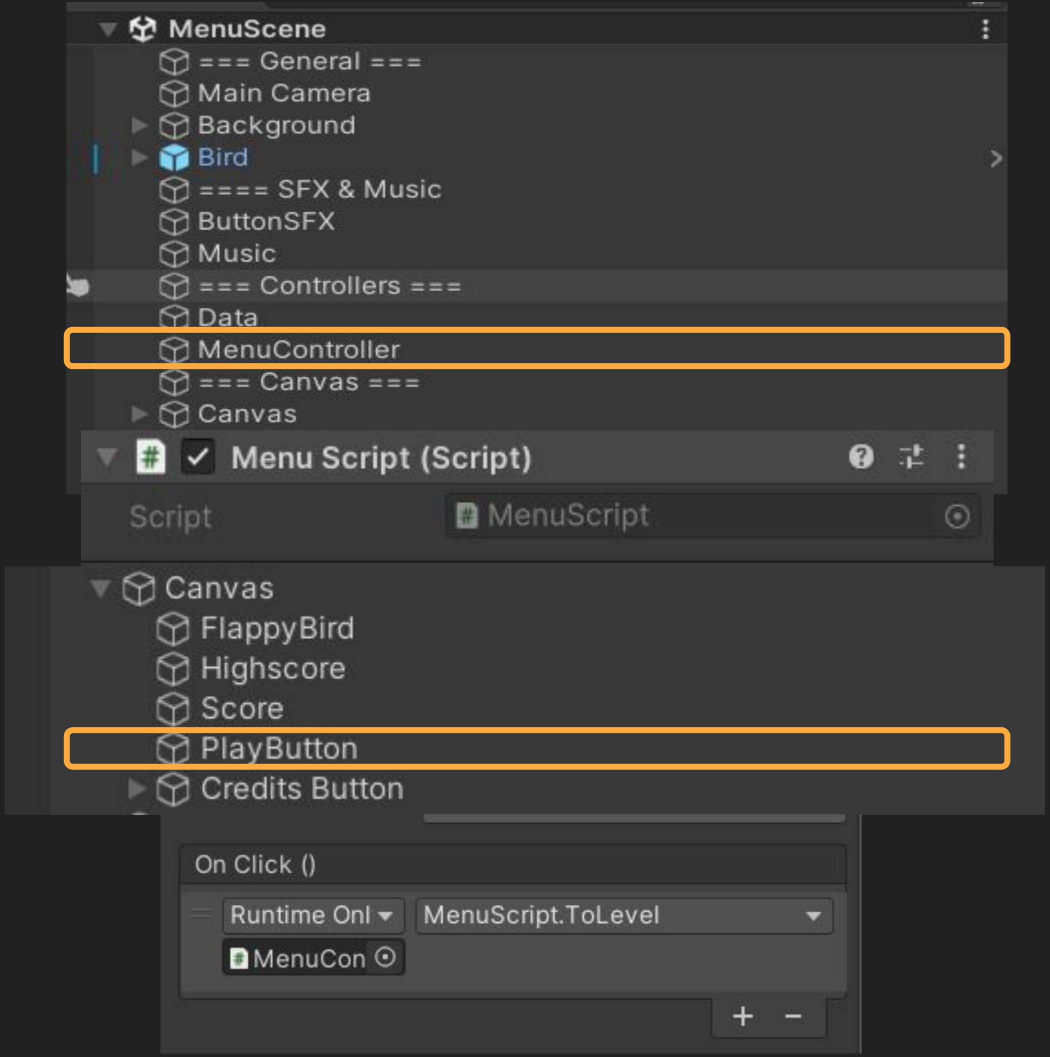
Inside the Courtine you will find that we play a SFX, and then as part of the courtuine we will wait till the SFX is done playing, and use the SceneManager to load into the chosen Scene.



```
    //Goes to Game Scene
    0 references
    public void ToLevel()
    {
        StartCoroutine(PlayAndLeave("GameScene"));
    }

    //Goes to Credit Scene
    0 references
    public void ToCredits()
    {
        StartCoroutine(PlayAndLeave("CreditsScene"));
    }

    //Waits until the button SFX has stopped playing and then goes to the
    //requested level
    2 references
    private IEnumerator PlayAndLeave(string level)
    {
        button.Play();
        yield return new WaitForSeconds(button.clip.length);
        SceneManager.LoadScene(level);
    }
}
```

Menu Script

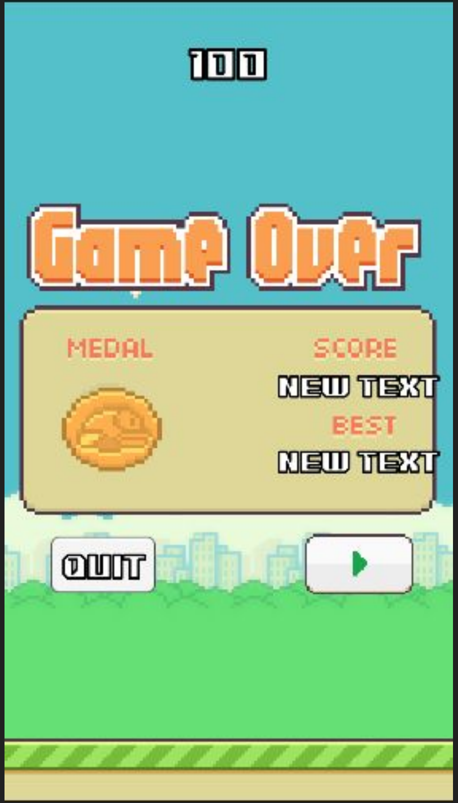
After you've created the Menu Script, attach it to the 'MenuController' game object. Once you've attached it, go to 'Canvas' -> 'PlayButton' and, in the 'OnClick' section, add the 'MenuController' object and select the 'MenuScript.ToLevel' function. This way, when you click on the button, you'll be moved to the level scene.

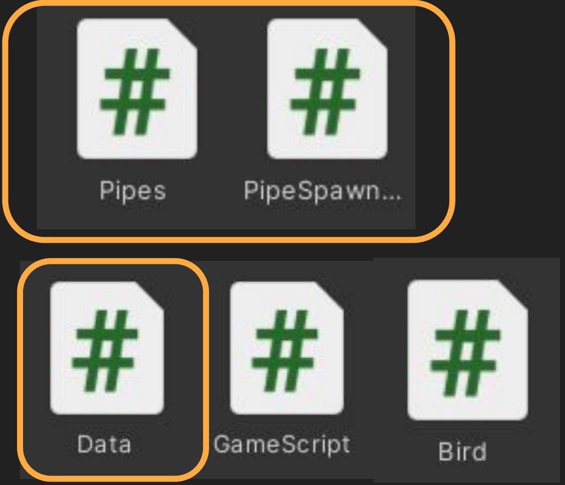


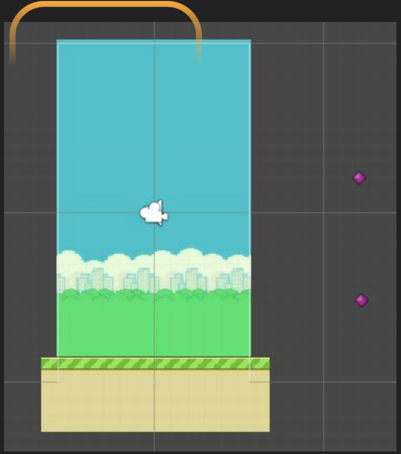
Game Scene

In the game scene, we will utilize these five scripts. We will reuse the 'Data' script to preserve the values. Additionally, we will create a script for the pipes to move and a separate one to generate new pipes.

Two of the scripts, the 'Bird' script and the 'GameScript,' have already been created for you.

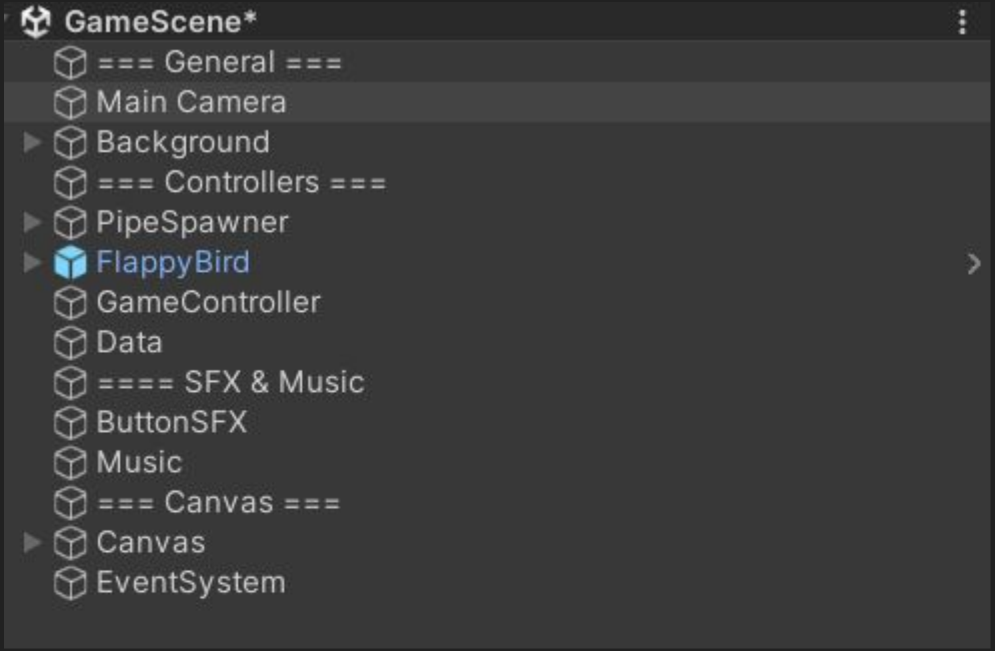


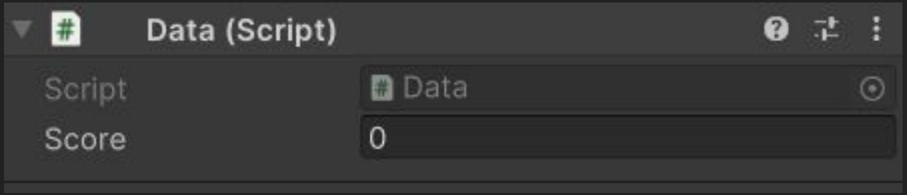




Data

We have already created this script, so we will reattach it to the 'Data' object that is in this scene.



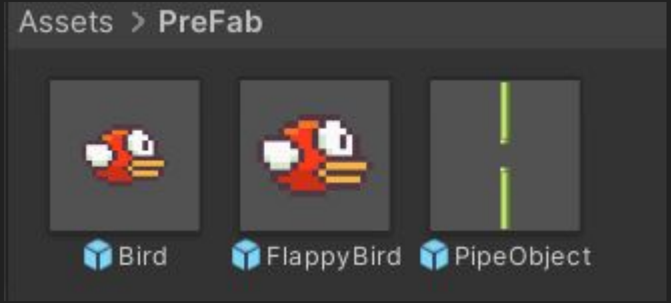


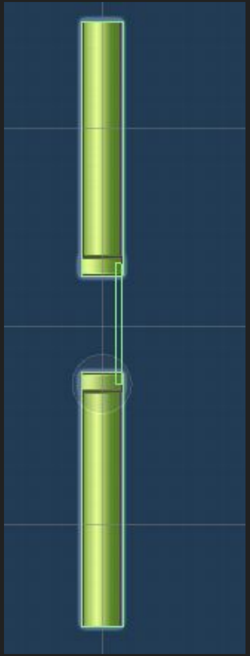
Script Data
Score 0

Pipe

In your 'Prefabs' folder, you will find a 'Pipe Object' prefab with two pipes and a Trigger Collider in the middle.

We will create a script that will move this object to the left and destroy it after a short period of time.

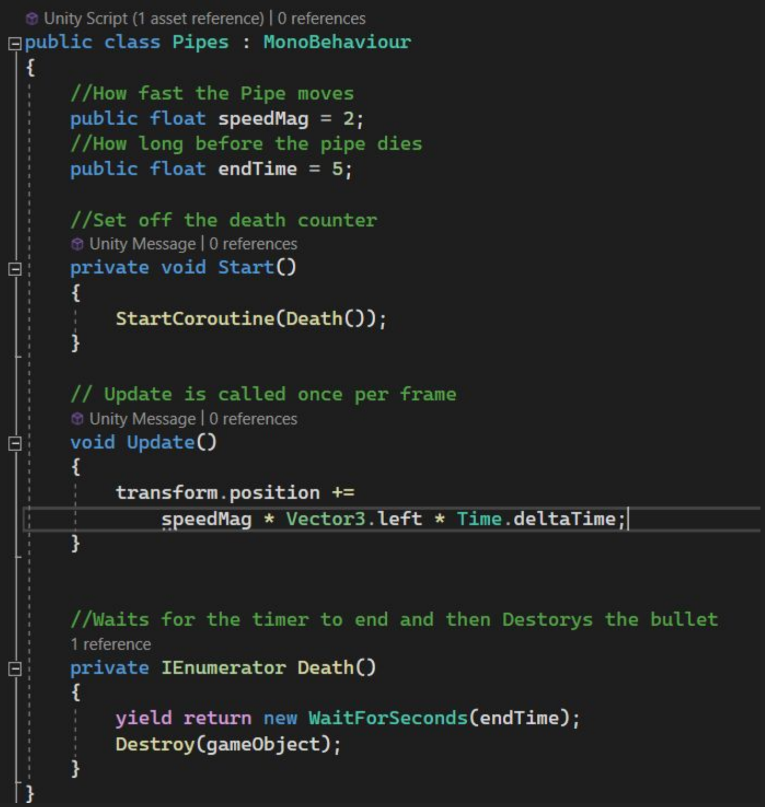




Pipe

There are two sections to this. The first is the movement of the object, which is handled in the 'Update' function. We use the predefined speed, multiplied by a vector that faces left, and then multiply it by 'deltaTime,' which represents the internal clock of the game.

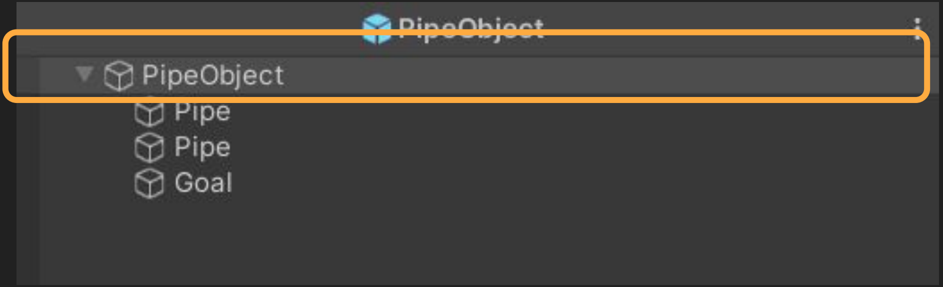
The second section involves a coroutine that runs alongside the object's movement. It waits for a specified time of 5 seconds before destroying the object.

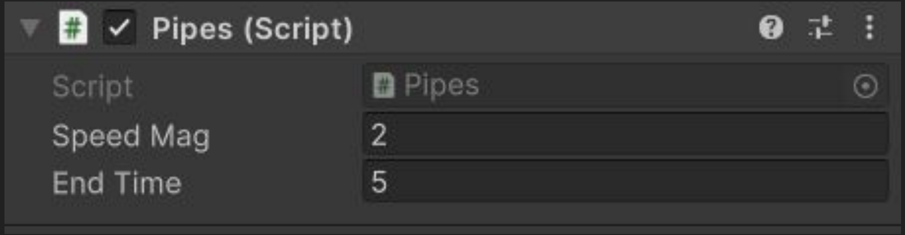


```
Unity Script (1 asset reference) | 0 references
public class Pipes : MonoBehaviour
{
    //How fast the Pipe moves
    public float speedMag = 2;
    //How long before the pipe dies
    public float endTime = 5;

    //Set off the death counter
    Unity Message | 0 references
    private void Start()
    {
        StartCoroutine(Death());
    }

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


    //Waits for the timer to end and then Destorys the bullet
    1 reference
    private IEnumerator Death()
    {
        yield return new WaitForSeconds(endTime);
        Destroy(gameObject);
    }
}
```

Pipe

After you've created and saved your script, open the Prefab and add the script to the 'Pipe Object Parent'



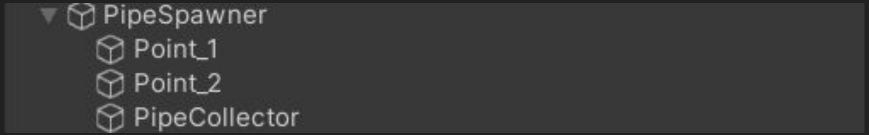


Script: Pipes
Speed Mag: 2
End Time: 5

Pipe Spawner

We have the PreFab but now we have to make a lot of them appear.

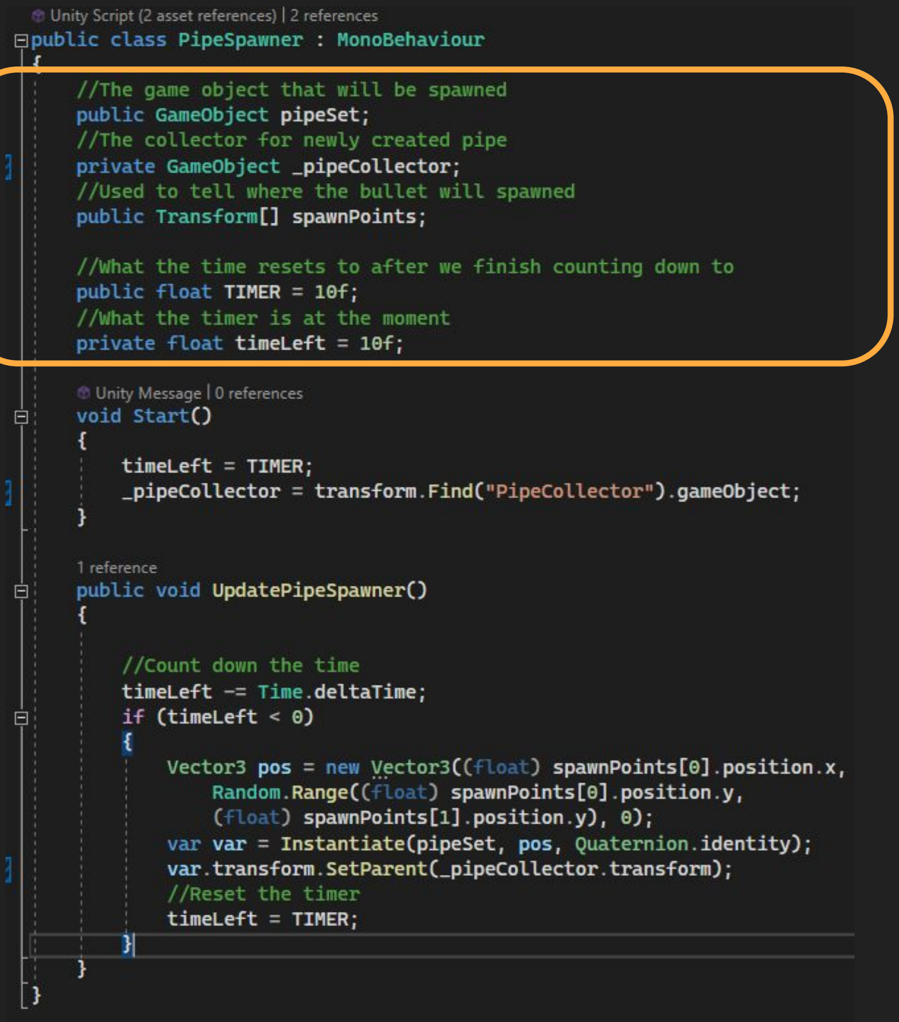
So in the game scene we have the piper spawner. The piper spawner is defied by the game object that holds everything together, 2 points which we will use to randomize the placement of the Pipe prefab, and a pipe collector where we will put the newly instantiated objects into.


```
▼ PipeSpawner
    Point_1
    Point_2
    PipeCollector
```

Pipe Spawner

We're going to have a few public variables here:

1. pipeSet is the prefab from which we will create more copies.
2. pipeCollector is the place where we will position the newly created pipes.
3. spawnPoints are the two spots in between where we'll physically spawn the pipeSet.
4. timeLeft tells us how long until we spawn a new pipe.
5. TIMER is the time that timeLeft is going to reset to.

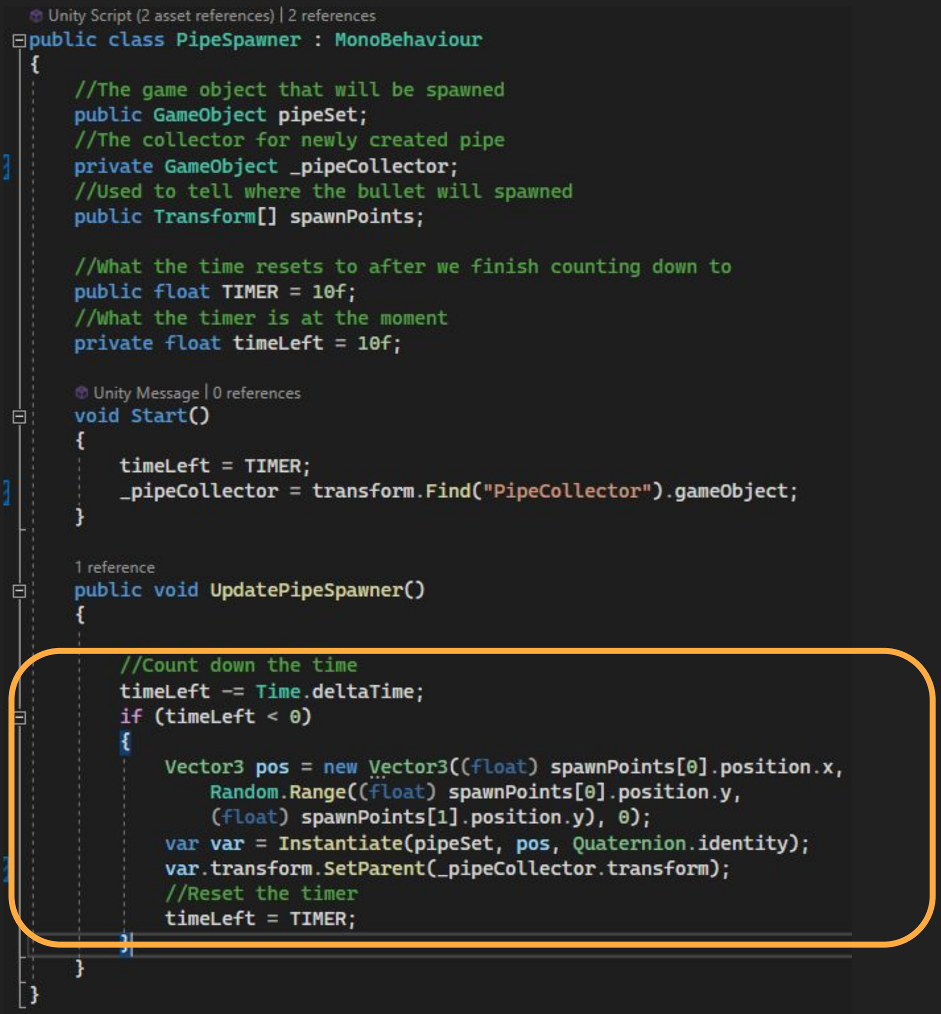


```
Unity Script (2 asset references) | 2 references
public class PipeSpawner : MonoBehaviour
{
    //The game object that will be spawned
    public GameObject pipeSet;
    //The collector for newly created pipe
    private GameObject _pipeCollector;
    //Used to tell where the bullet will spawned
    public Transform[] spawnPoints;

    //What the time resets to after we finish counting down to
    public float TIMER = 10f;
    //What the timer is at the moment
    private float timeLeft = 10f;

    Unity Message | 0 references
    void Start()
    {
        timeLeft = TIMER;
        _pipeCollector = transform.Find("PipeCollector").gameObject;
    }

    1 reference
    public void UpdatePipeSpawner()
    {

        //Count down the time
        timeLeft -= Time.deltaTime;
        if (timeLeft < 0)
        {
            Vector3 pos = new Vector3((float) spawnPoints[0].position.x,
                Random.Range((float) spawnPoints[0].position.y,
                (float) spawnPoints[1].position.y), 0);
            var var = Instantiate(pipeSet, pos, Quaternion.identity);
            var.transform.SetParent(_pipeCollector.transform);
            //Reset the timer
            timeLeft = TIMER;
        }
    }
}
```

Pipe Spawner

We will work with two functions. In the Start function, we preset the timeLeft to TIMER in case you've changed it in the inspector, and we grab the 'Pipe Collector' object.

In the Update function, we continuously decrease the timeLeft until it's less than 0. Once the time runs out, we find a position where we'll spawn the pipe, and we randomize the Y position between the two points. Then, we instantiate the pipe, meaning we create it. We set the 'Pipe Collector' to be the parent of the newly created pipe. Lastly, we reset the timer to allow the creation of the next pipe.

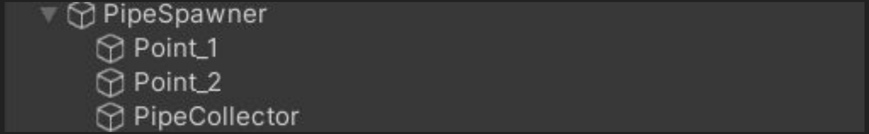


```
Unity Script (2 asset references) | 2 references
public class PipeSpawner : MonoBehaviour
{
    //The game object that will be spawned
    public GameObject pipeSet;
    //The collector for newly created pipe
    private GameObject _pipeCollector;
    //Used to tell where the bullet will spawned
    public Transform[] spawnPoints;

    //What the time resets to after we finish counting down to
    public float TIMER = 10f;
    //What the timer is at the moment
    private float timeLeft = 10f;

    Unity Message | 0 references
    void Start()
    {
        timeLeft = TIMER;
        _pipeCollector = transform.Find("PipeCollector").gameObject;
    }

    1 reference
    public void UpdatePipeSpawner()
    {
        //Count down the time
        timeLeft -= Time.deltaTime;
        if (timeLeft < 0)
        {
            Vector3 pos = new Vector3((float) spawnPoints[0].position.x,
                Random.Range((float) spawnPoints[0].position.y,
                (float) spawnPoints[1].position.y), 0);
            var var = Instantiate(pipeSet, pos, Quaternion.identity);
            var.transform.SetParent(_pipeCollector.transform);
            //Reset the timer
            timeLeft = TIMER;
        }
    }
}
```

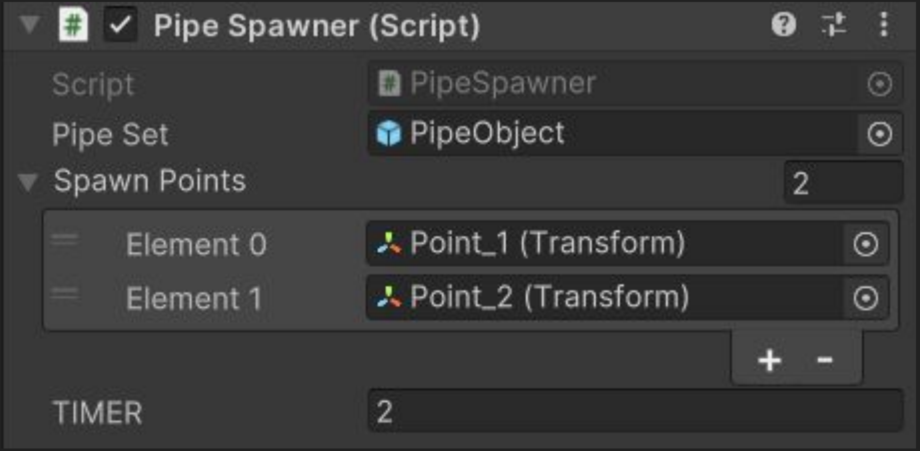
Pipe Spawner

After you've saved your script, attach it to the parent object 'PipeSpawner.' Within the 'PipeSpawner,' you have to add a few items, such as:

1. Set the 'PipeObject,' which is the prefab you will drag from the prefab menu.
2. Define the 'SpawnPoints' as 'Point_1' and 'Point_2.'
3. Change the 'TIMER' to 2.



PipeSpawner
Point_1
Point_2
PipeCollector



Pipe Spawner (Script)

Script	PipeSpawner
Pipe Set	PipeObject
Spawn Points	2
Element 0	Point_1 (Transform)
Element 1	Point_2 (Transform)
TIMER	2