Welcome to the Course

In this course, we'll be creating an in-game leaderboard using Microsoft Azure's PlayFab. The leaderboard will be connected to a small game where as the player, you need to collect 10 orbs in as little time you can. The previous course where we created a login/register system is required before attempting this one. We went over creating our PlayFab app, setting up Unity etc. It's highly recommended you complete that course first.

What new PlayFab features are we going to look at in this course?

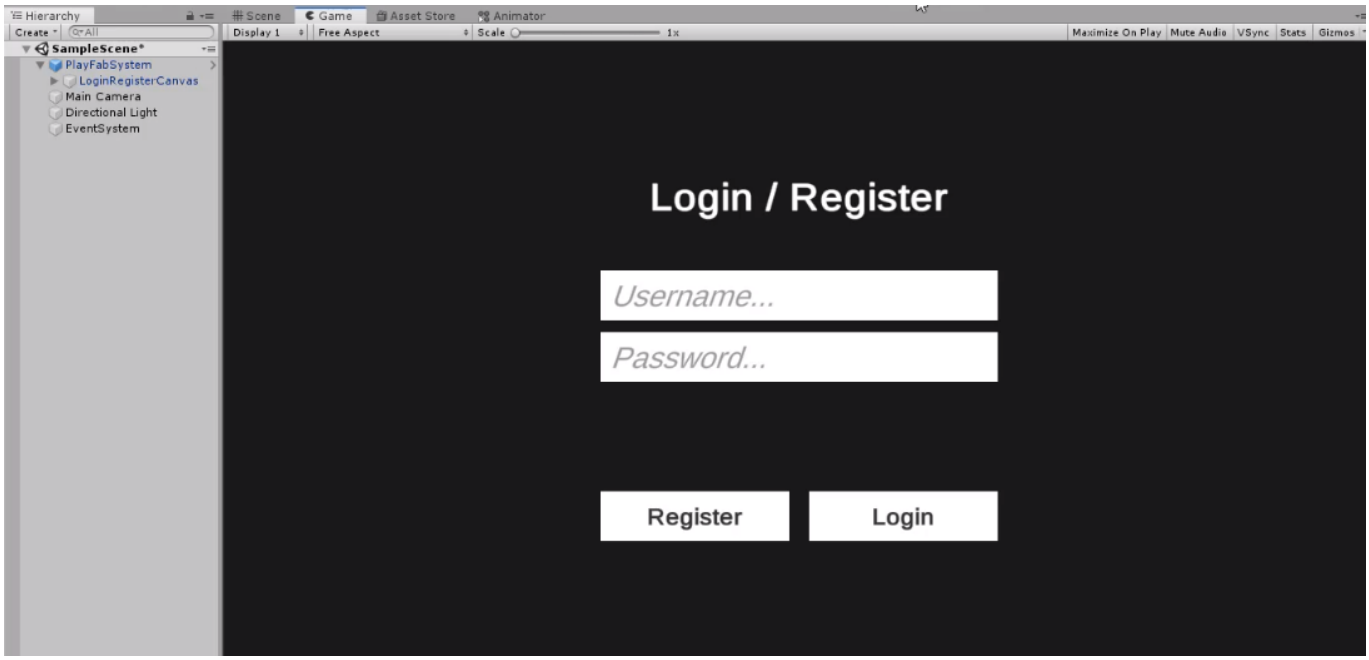
- Creating, adding and updating leaderboards.
- Cloud scripting, which will allow us to run code server-side (preventing client-side abuse).

Now, let's get started on the course!

Unity Setup

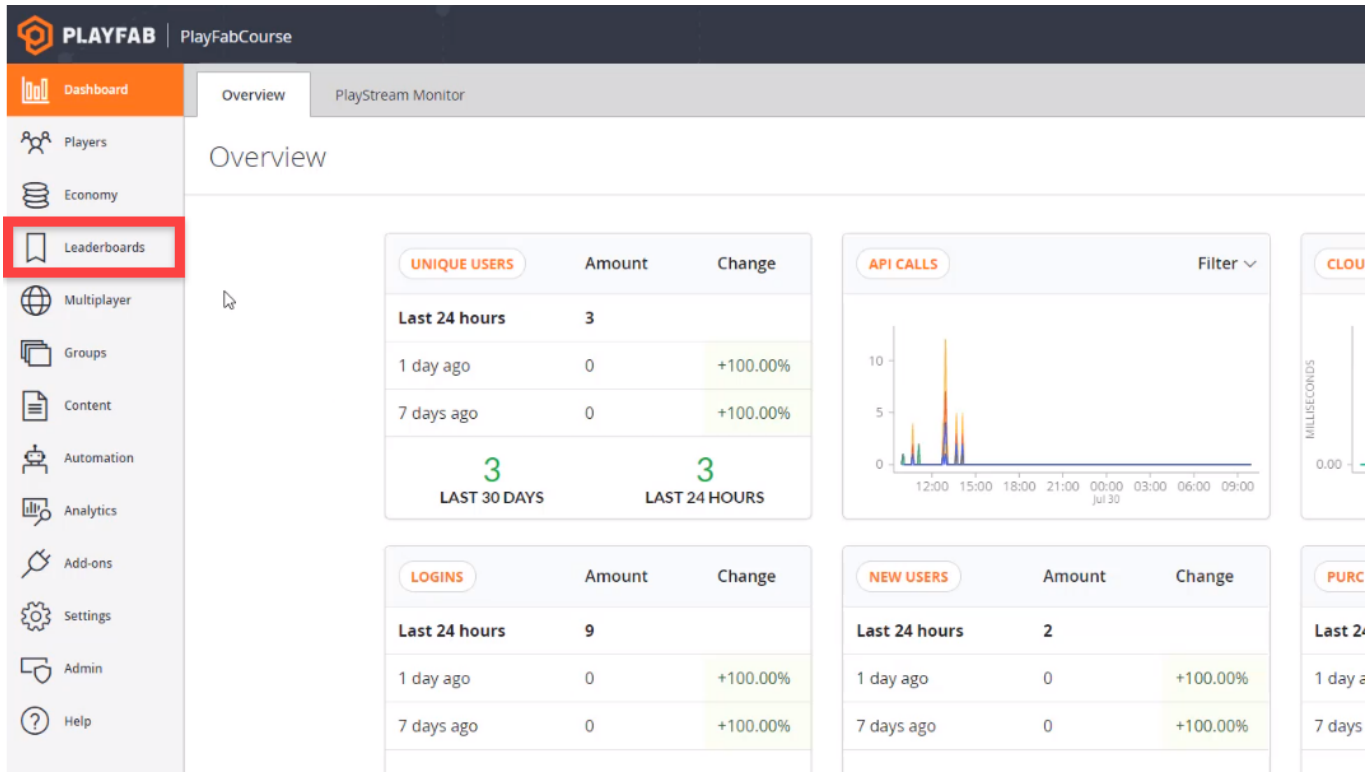
In Unity, import the PlayFab SDK and editor extension packages. Go through the same setup steps as we done in the first course. We're also going to re-use the login/register system created in the last course. You can either copy the files from the project, or download the course files for this course.

Let's drag the **PlayFabSystem** prefab into the scene and change the camera color to black/dark grey. Also make sure to add an EventSystem game object so that we can interact with the UI (right click Hierarchy > *UI* > *Event* System).

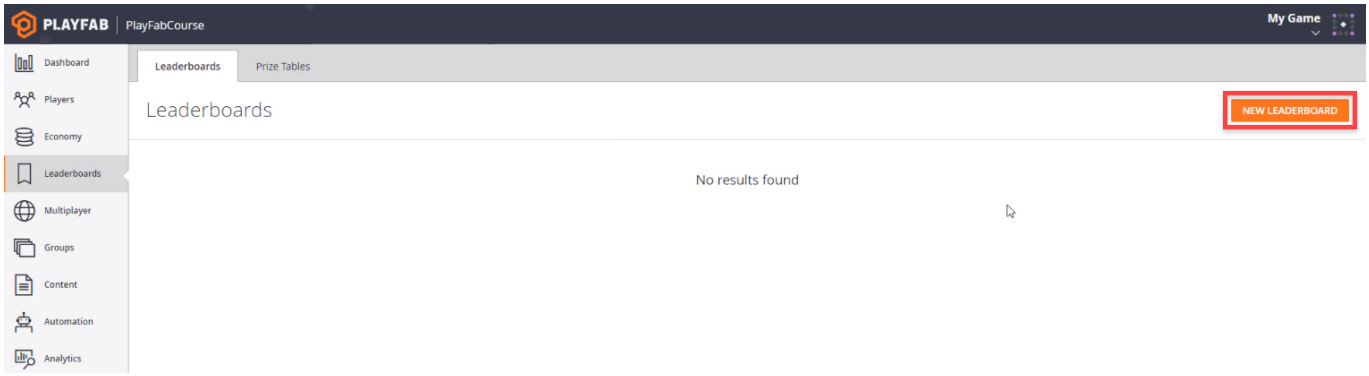


Creating the Leaderboard

In our PlayFab dashboard, let's click on the **Leaderboards** button to go to the Leaderboard page.



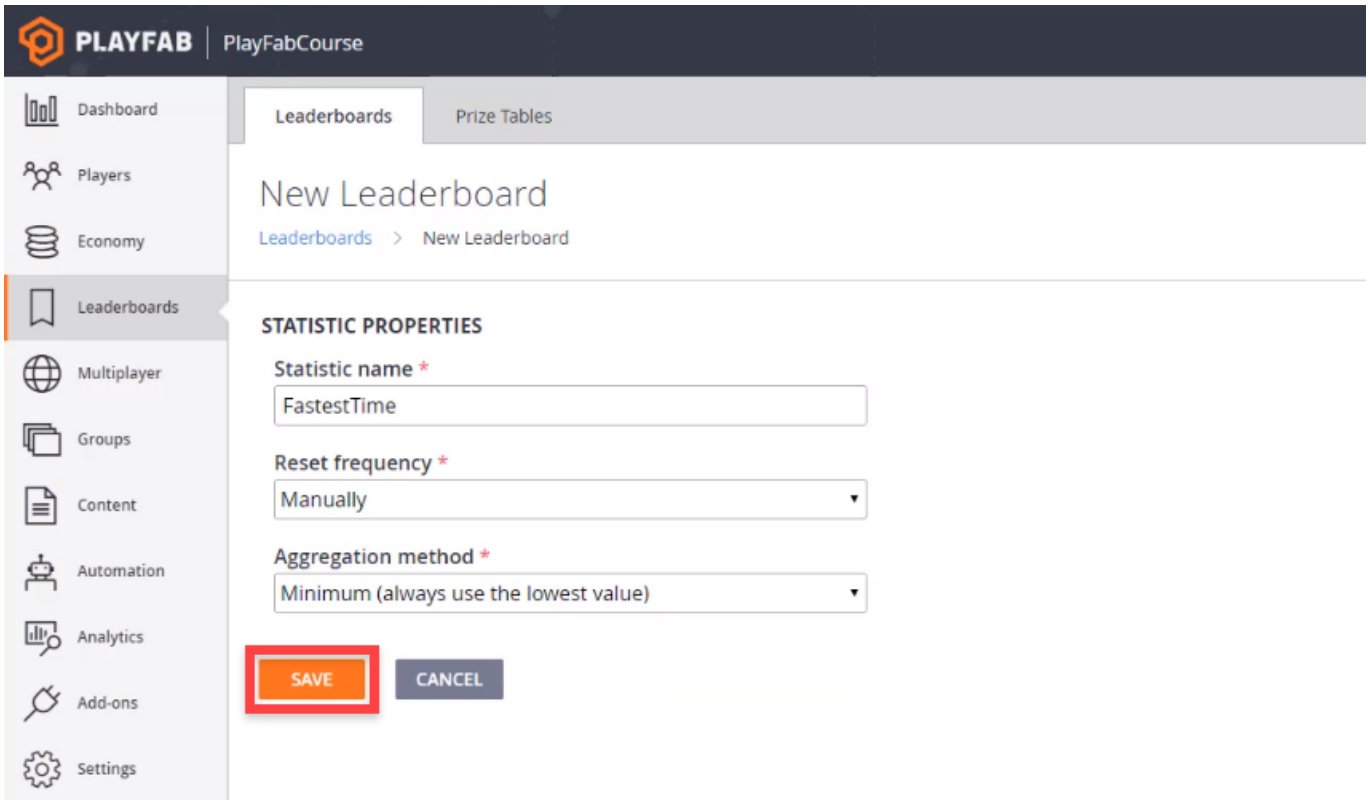
Here, we want to click on the **New Leaderboard** button.



You'll see that commonly, a leaderboard is known as a **statistic**.

- Set the **Statistic name** to *FastestTime*
- Set the **Reset frequency** to *Manually*
- Set the **Aggregation method** to *Minimum (always use the lowest value)*

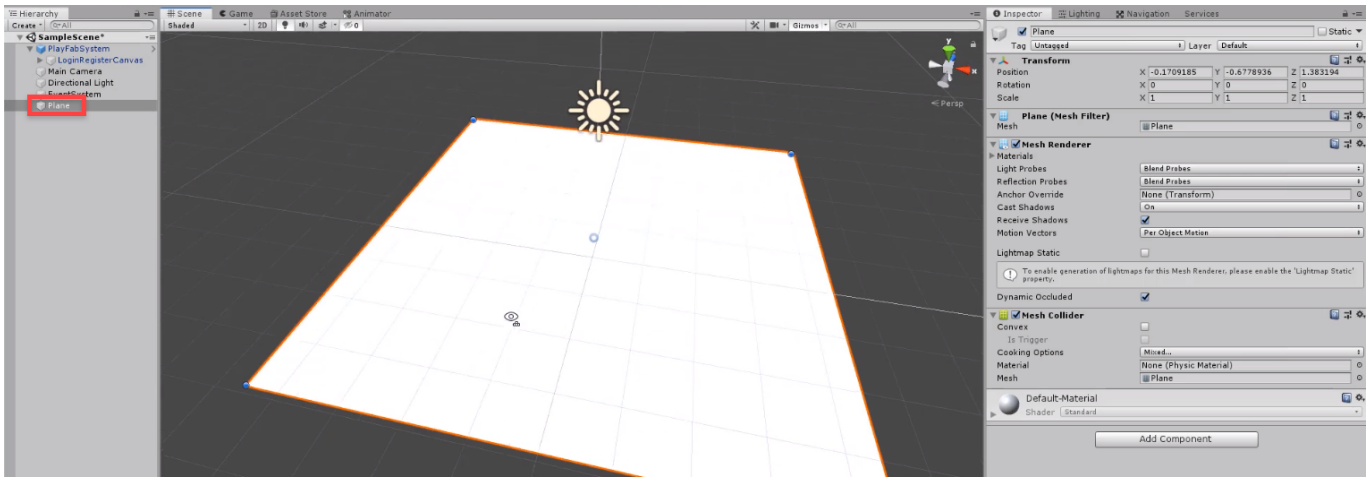
Click the **Save** button.



In the next lesson, we'll be creating the player object we can move around.

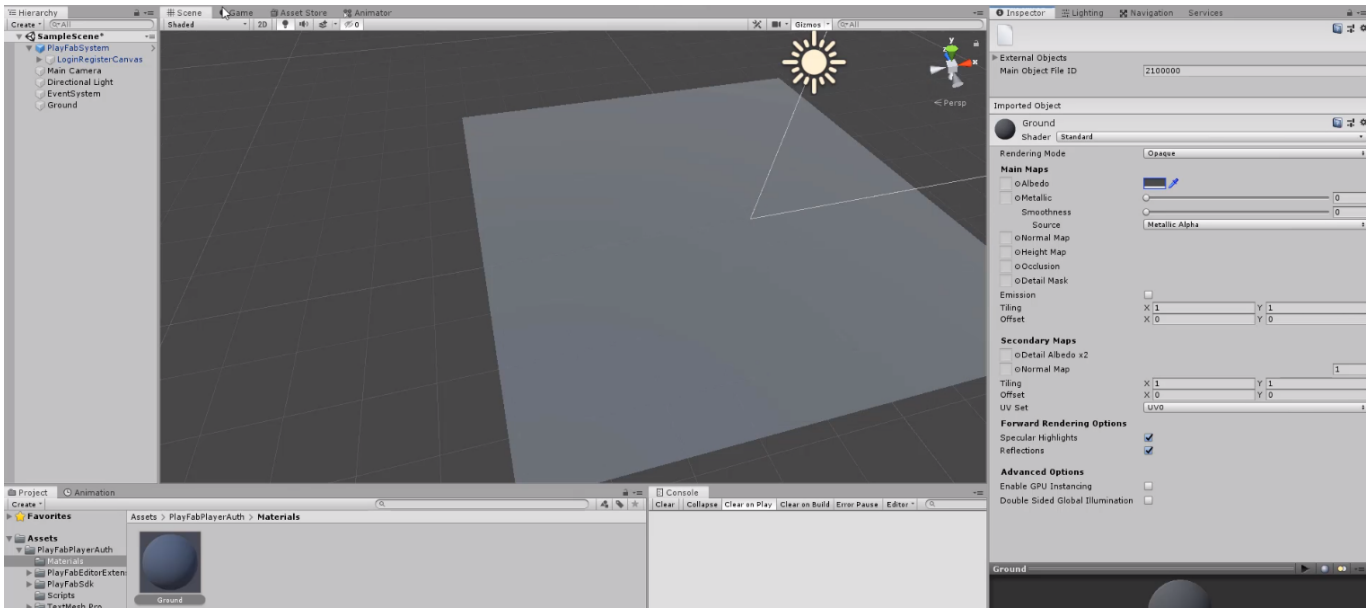
Scene Setup

Create a new Plane object for the ground.



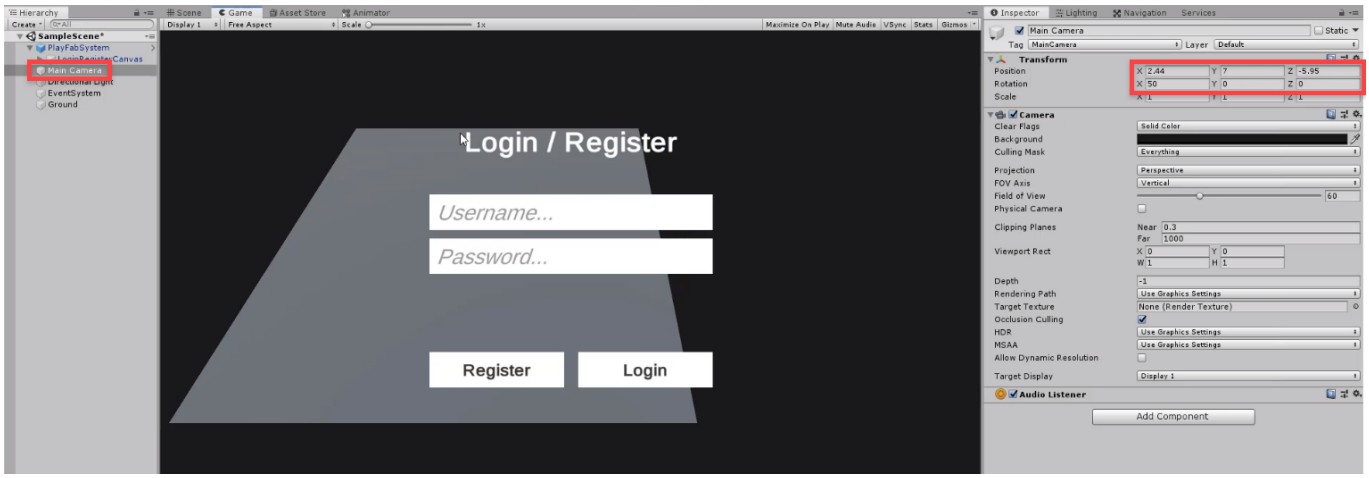
Create a new folder called **Materials** and inside of that, create a new material called **Ground**. Apply this to the ground plane.

- Set the **Color** to grey
- Set the **Smoothness** to *0*



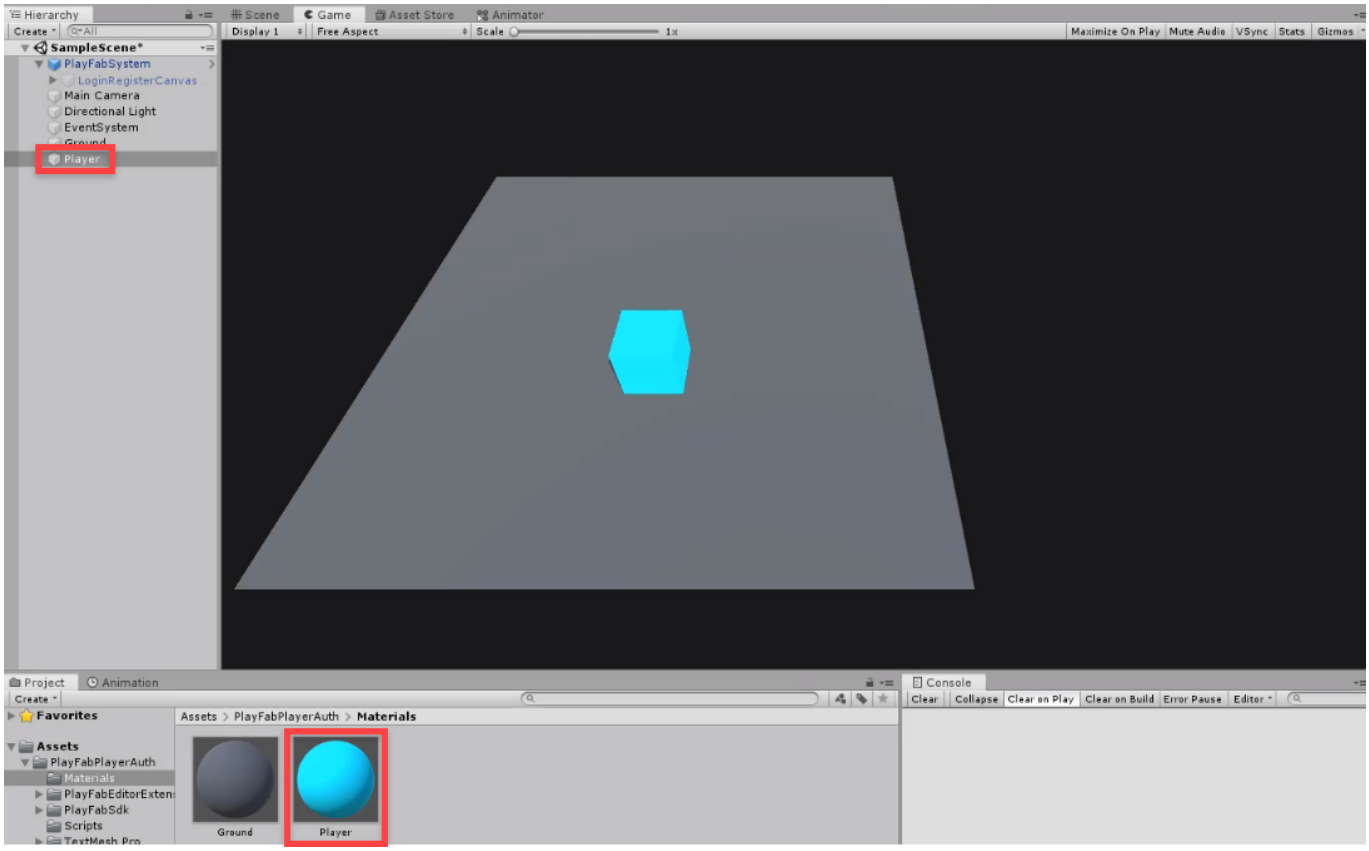
Select the camera and move/rotate it into a better position.

- Set the **Position** to *2.44, 7, -5.95*
- Set the **Rotation** to *50, 0, 0*



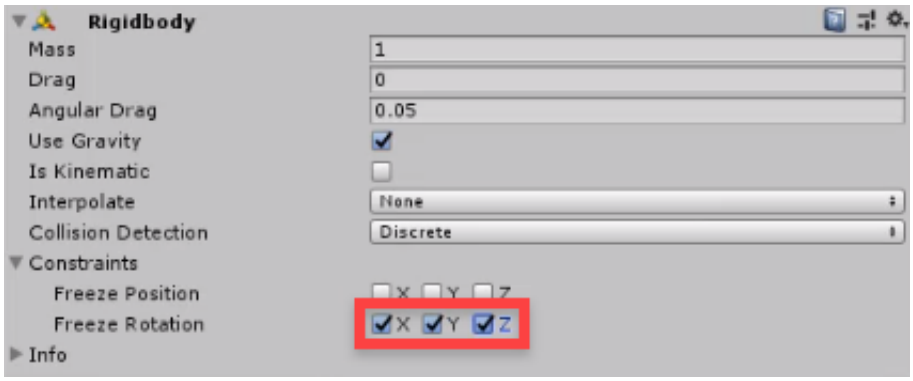
Player Object

Create a new cube object called **Player**. Also create a new material called **Player**, attach it and set the color to light blue.



We're going to be moving the player around with physics, so let's add a **Rigidbody** component.

- Enable the **Freeze Rotation** on all axis'

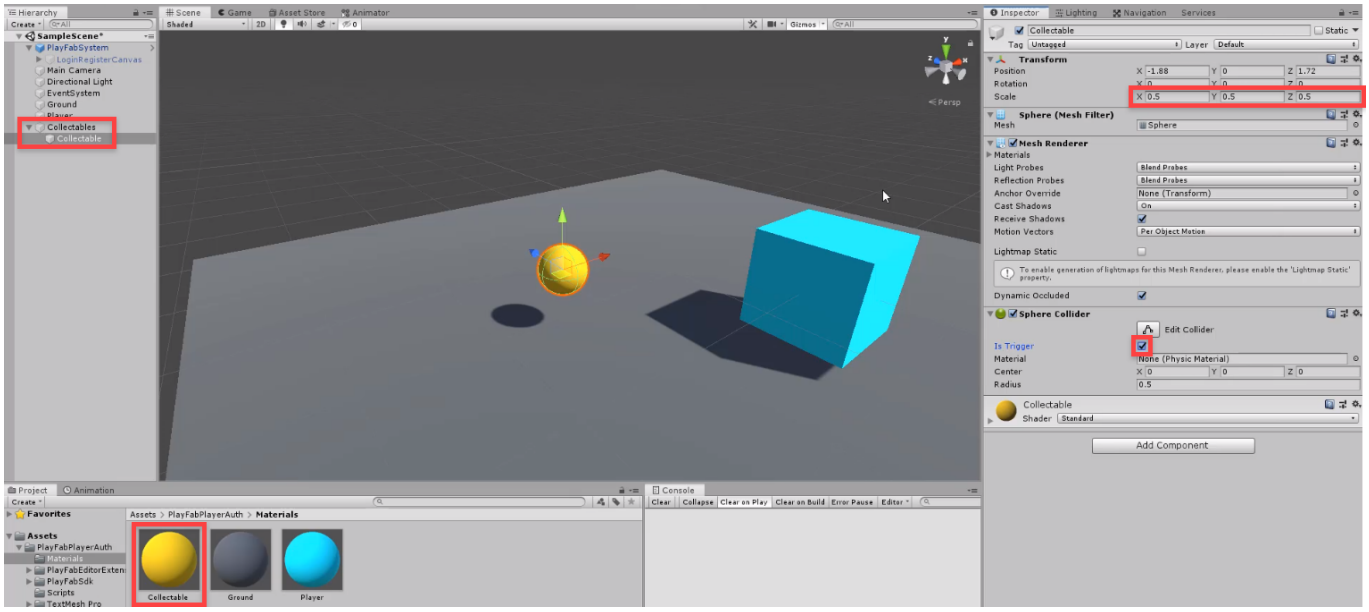


Collectables

Let's now create the things that the player will collect in order to set their time. Create a new empty GameObject called **Collectables**. This will hold the objects.

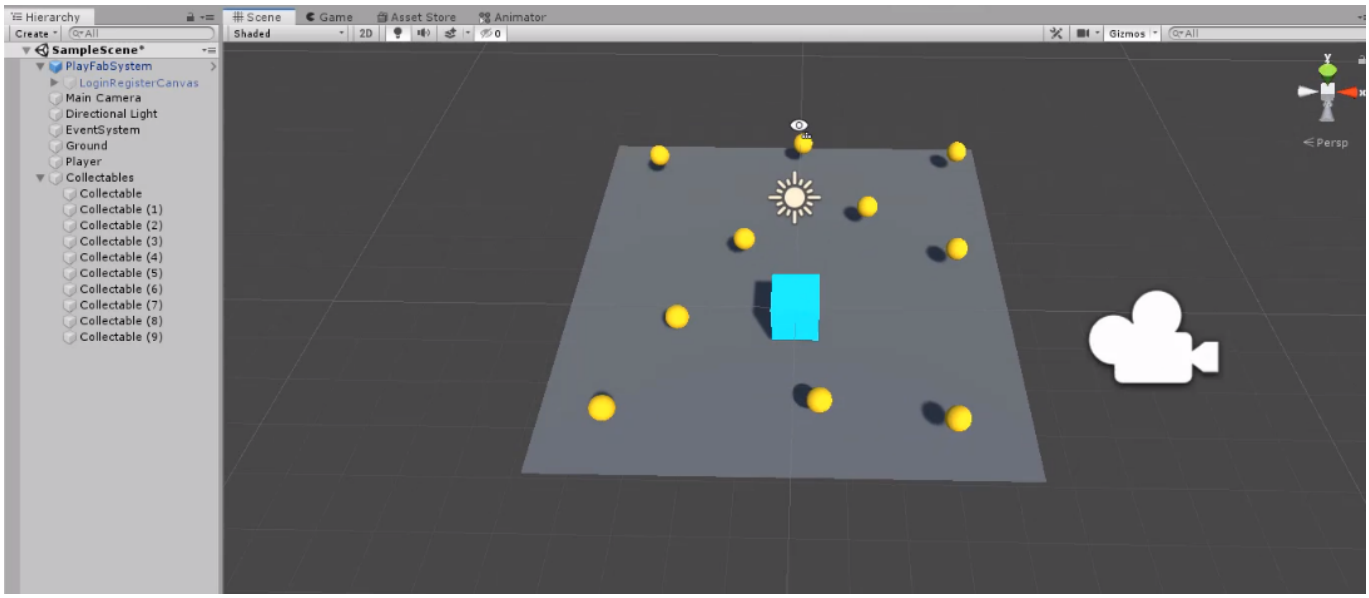
As the child of that, create a new sphere object called **Collectable**.

- Set the **Scale** to *0.5*
- Enable **Is Trigger**
- Create a new yellow material called **Collectable** and apply it to the sphere



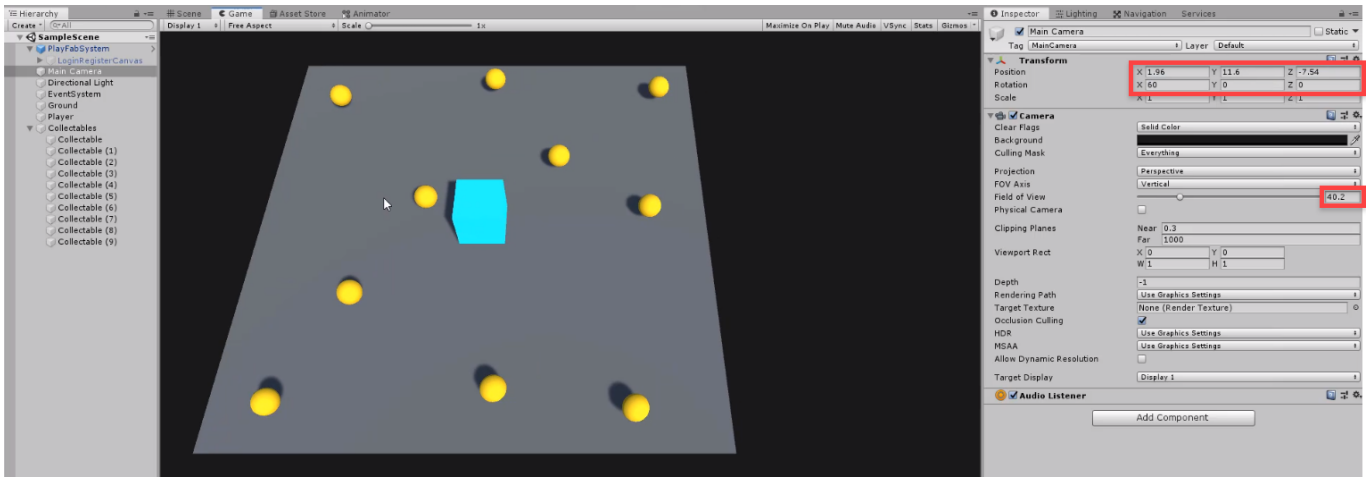
To know if we hit a collectable, let's create a new tag called **Collectable**. Assign this to the collectable object.

Now, duplicate the collectable 9 times and position them randomly around the ground.



We can finally change our camera a bit.

- Set the **Position** to *1.96, 11.6, -7.54*
- Set the **Rotation** to *60, 0, 0*
- Set the **Field of View** to *40*



PlayerController Script

Create a new C# script called **PlayerController** and attach it to the player object. Let's start with the libraries we'll be using.

```
using TMPro;
```

Then we can create our variables.

```
public float speed;
private Rigidbody rig;

private float startTime;
private float timeTaken;

private int collectablesPicked;
public int maxCollectables = 10;

private bool isPlaying;
```

In the **Awake** function, let's get the rigidbody component.

```
void Awake ()
{
    rig = GetComponent<Rigidbody>();
}
```

In the **Update** function, we'll be doing most of our actions.

```
void Update ()
{

}
```

For moving, let's start by getting the horizontal and vertical inputs.

```
float x = Input.GetAxis("Horizontal") * speed;
float z = Input.GetAxis("Vertical") * speed;
```

Then we can set the velocity.

```
rig.velocity = new Vector3(x, rig.velocity.y, z);
```

In the editor, set the variables:

- Speed = 3
- Max Collectables = 10

Then press play and test out the movement.

Collecting Orbs

Back in the script, let's create the **OnTriggerEnter** function, which gets called when we collide with an object. We'll add one to the **collectablesPicked** variable and destroy the collectable.

```
void OnTriggerEnter (Collider other)
{
    if(other.gameObject.CompareTag("Collectable"))
    {
        collectablesPicked++;
        Destroy(other.gameObject);

        if(collectablesPicked == maxCollectables)
            End();
    }
}
```

The **Begin** function starts the timer.

```
public void Begin ()
{
    startTime = Time.time;
    isPlaying = true;
}
```

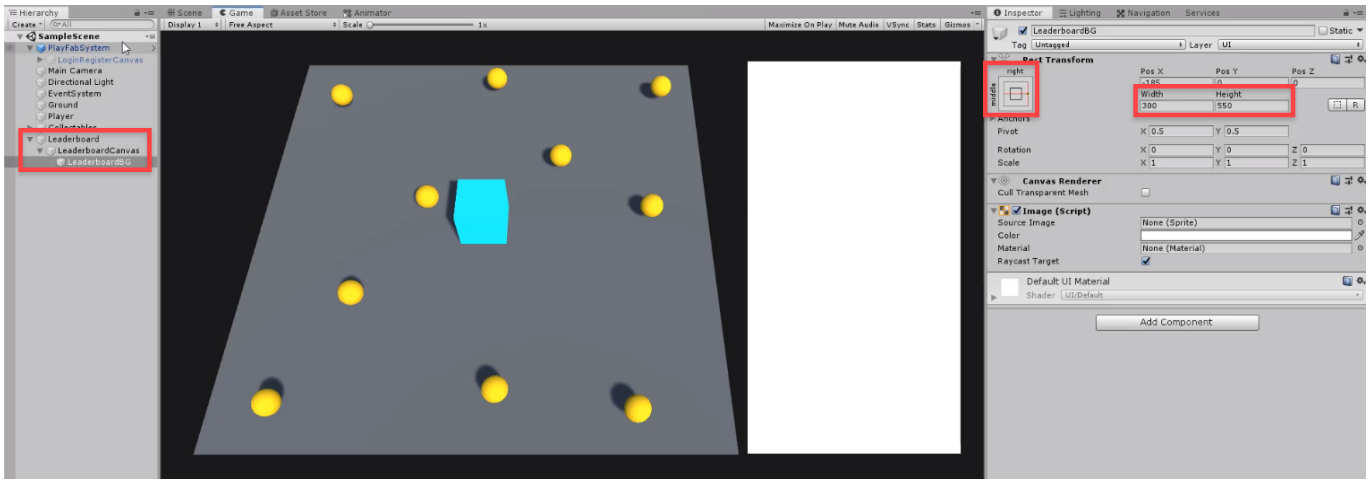
The **End** function gets called when the timer has ended.

```
void End ()
{
    timeTaken = Time.time - startTime;
    isPlaying = false;
}
```

Creating the Leaderboard UI

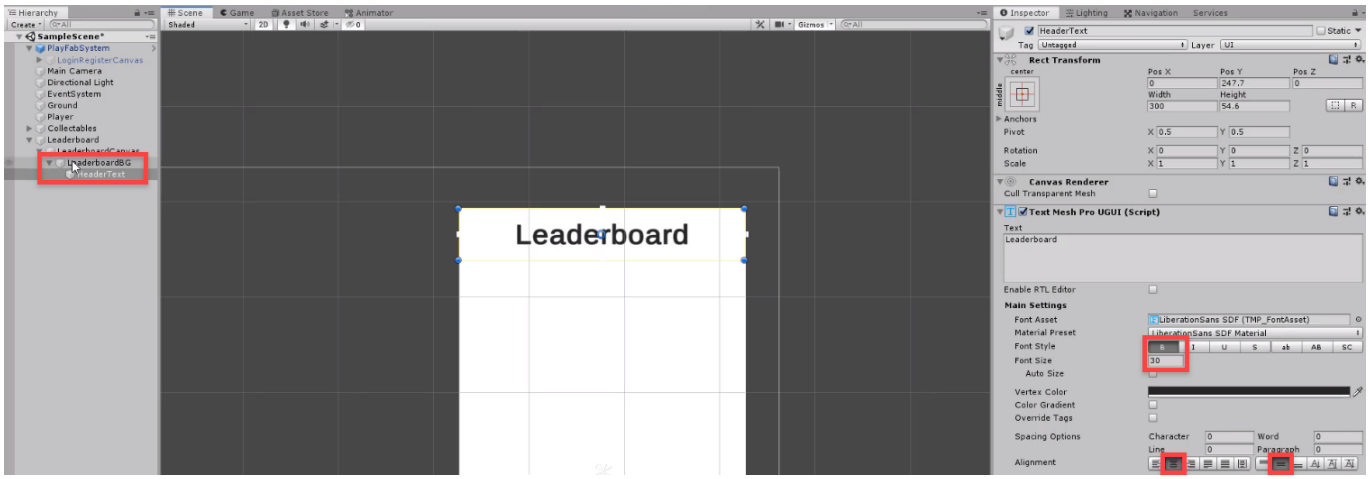
Create a new empty GameObject called **Leaderboard**. As a child of that, create a new canvas called **LeaderboardCanvas**. As a child of the canvas, create a new Image called **LeaderboardBG**.

- Set the **Anchoring** to *middle-right*
- Set the **Width** to *300*
- Set the **Height** to *550*

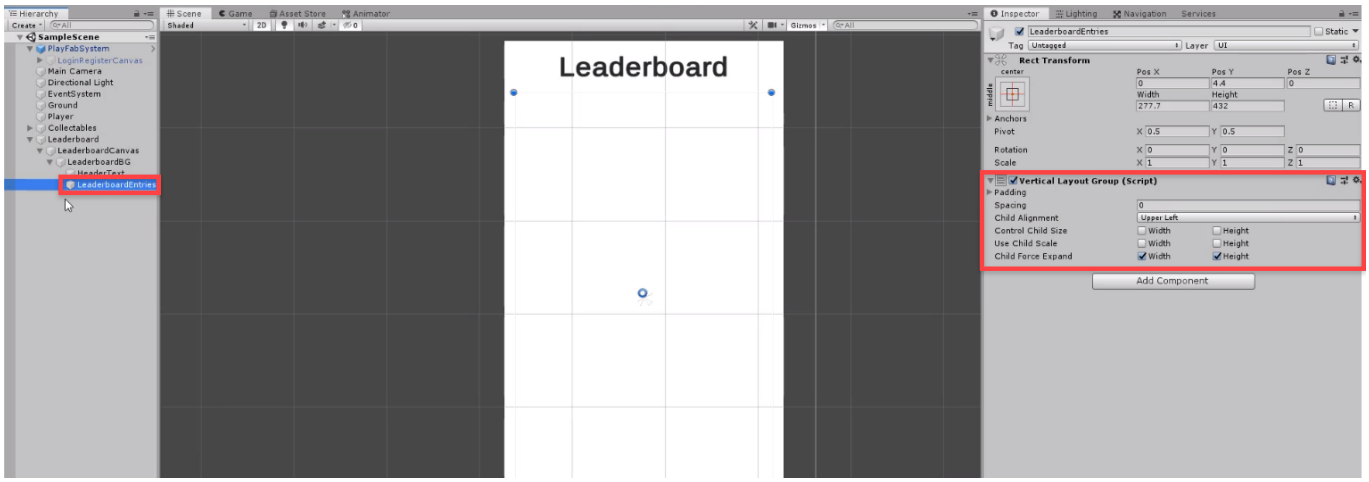


Then as a child of the image, create a new Text – TextMeshPro element.

- Set the **Font Style** to *Bold*
- Set the **Font Size** to *30*
- Set the **Alignment** to *middle-center*

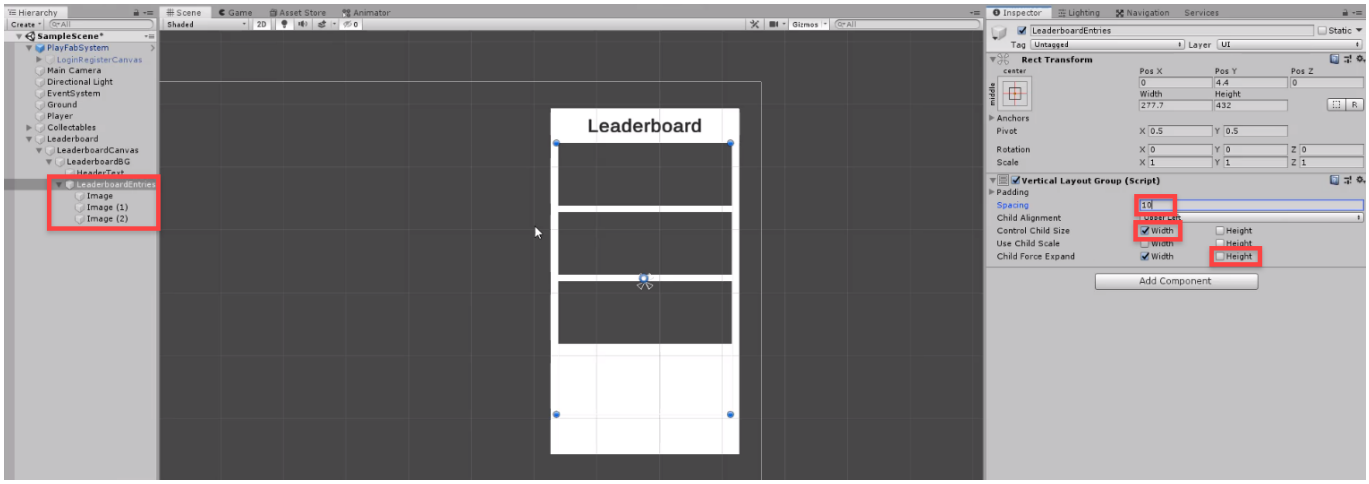


Next, we can create a new empty GameObject called **LeaderboardEntries**. Attach a **Vertical Layout Group** component.

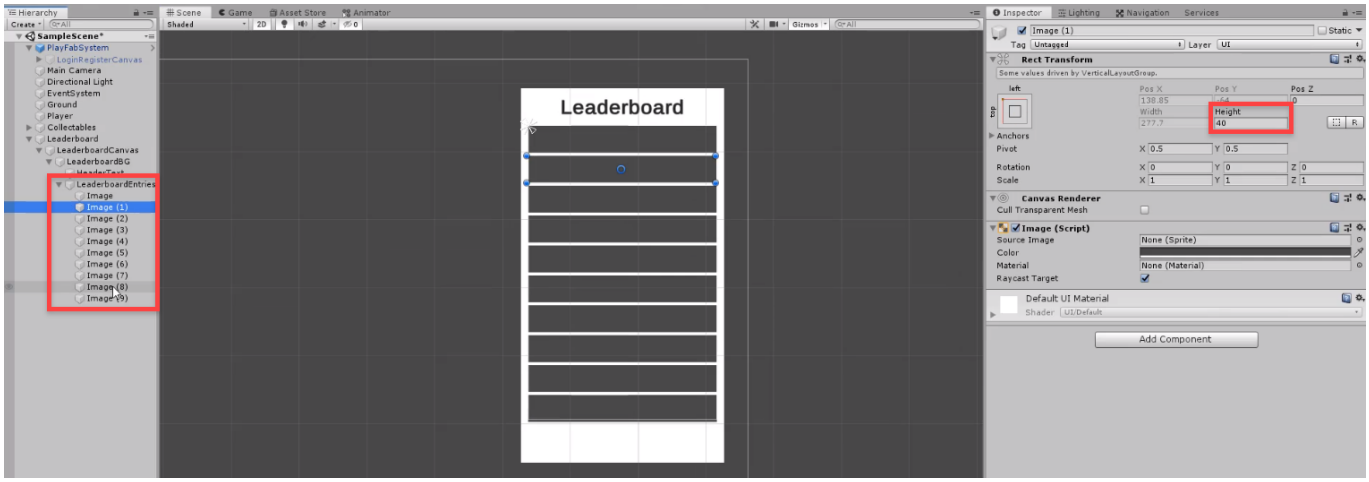


Create a new image as a child of the empty object (duplicate it a few times). This should automatically position and scale the images so that they're ordered. Let's change some things on the vertical layout group component.

- Set the **Spacing** to *5* (not seen in image)
- Enable the **Control Child Size – Width**
- Disable the **Child Force Expand – Height**

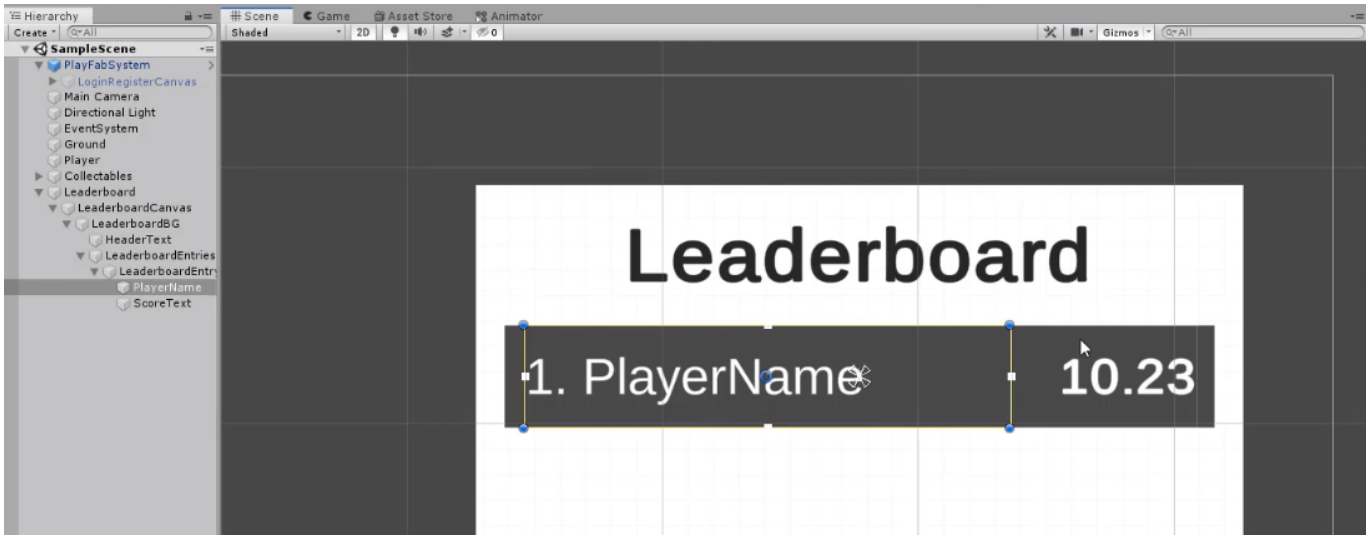


Set the image height to *40*.

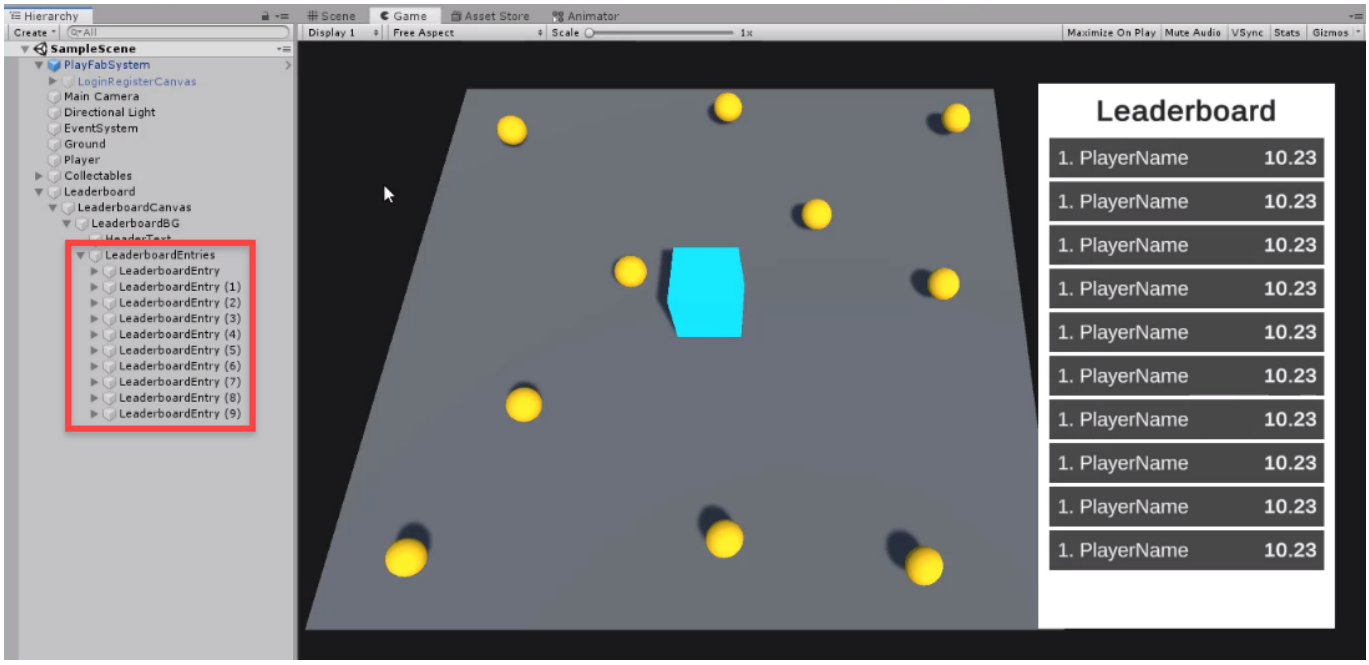


Each image is going to be an entry in the leaderboard. Let's create 2 new text elements.

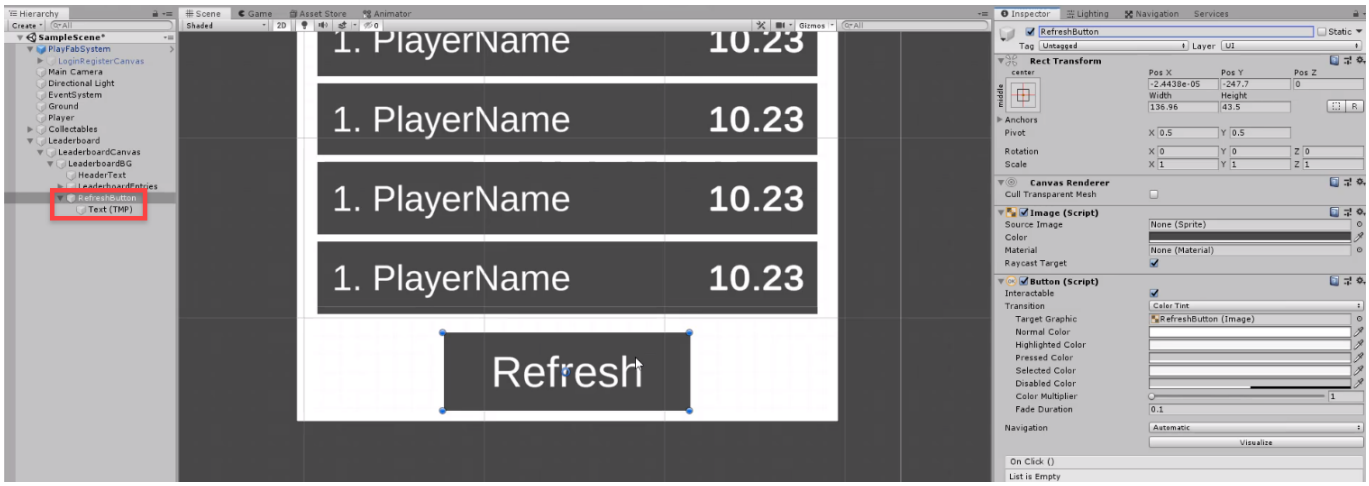
- **PlayerName** – display the player rank (1 – 10) as well as their display name.
- **ScoreText** – display the player's time.



Duplicate the leaderboard entry so that we have 10.



Let's also add a new Button – TextMeshPro which will refresh the leaderboard.



Leaderboard Script

Create a new C# script called **Leaderboard** and attach it to the Leaderboard object. Let's start with the libraries we'll be using.

```
using PlayFab;
using PlayFab.ClientModels;
using TMPro;
```

Then we can create our variables.

```
public GameObject leaderboardCanvas;
public GameObject[] leaderboardEntries;
```

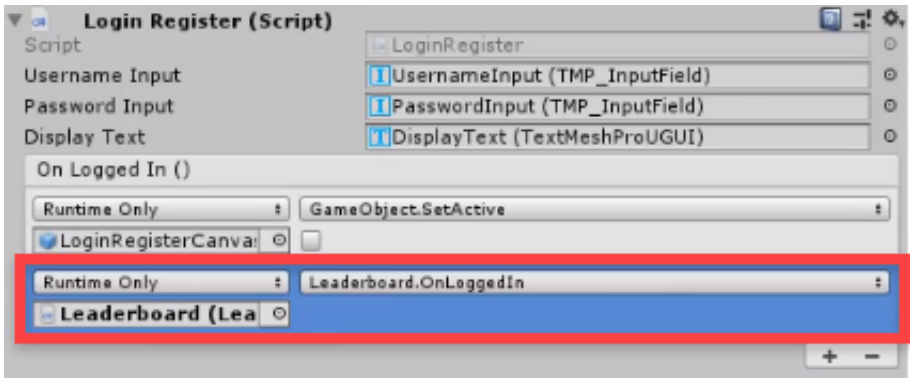
Let's also create an instance of the script.

```
public static Leaderboard instance;
void Awake () { instance = this; }
```

The **OnLoggedIn** function gets called when we log into our PlayFab account.

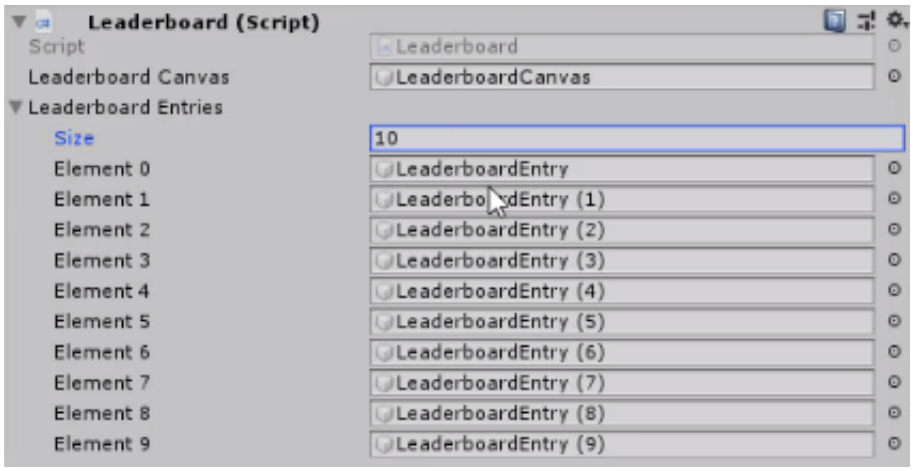
```
public void OnLoggedIn ()
{
    leaderboardCanvas.SetActive(false);
    DisplayLeaderboard();
}
```

Back in the editor, we can add the function to the LoginRegister script's OnLoggedIn function.



Let's also disable the leaderboard canvas by default and make sure that the login register canvas is enabled.

On the Leaderboard component, make sure to fill in the properties.



The **DisplayLeaderboard** function sends the API to get the server and gets us the leaderboard.

```
public void DisplayLeaderboard ()
{

}
```

Let's start with creating the class to hold our API info.

```
GetLeaderboardRequest getLeaderboardRequest = new GetLeaderboardRequest
{
    StatisticName = "FastestTime",
    MaxResultsCount = 10
};
```

Then we can call the API.

```
PlayFabClientAPI.GetLeaderboard(getLeaderboardRequest,
    result => UpdateLeaderboardUI(result.Leaderboard),
    error => Debug.Log(error.ErrorMessage)
);
```

The **UpdateLeaderboardUI** function gets called when we get the API request back with a list of players in the leaderboard. This function will display the leaderboard on-screen.

```
void UpdateLeaderboardUI (List<PlayerLeaderboardEntry> leaderboard)
{

}
```

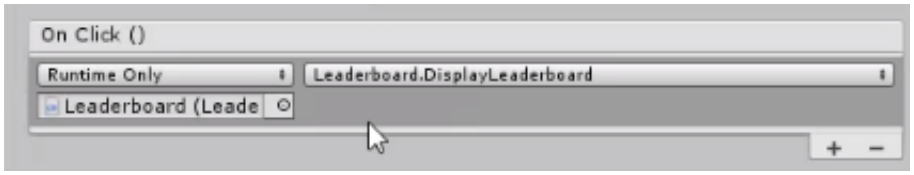

Displaying the Leaderboard In-Game

Let's start filling in the **UpdateLeaderboardUI** function.

```
for(int x = 0; x < leaderboardEntries.Length; x++)
{
    leaderboardEntries[x].SetActive(x < leaderboard.Count);
    if(x >= leaderboard.Count) continue;

    leaderboardEntries[x].transform.Find("PlayerName").GetComponent<TextMeshProUGUI>(
).text = (leaderboard[x].Position + 1) + ". " + leaderboard[x].DisplayName;
    leaderboardEntries[x].transform.Find("ScoreText").GetComponent<TextMeshProUGUI>()
.text = (-(float)leaderboard[x].StatValue * 0.001f).ToString();
}
```

Let's also hook up the Refresh button.



Setting a Leaderboard Entry

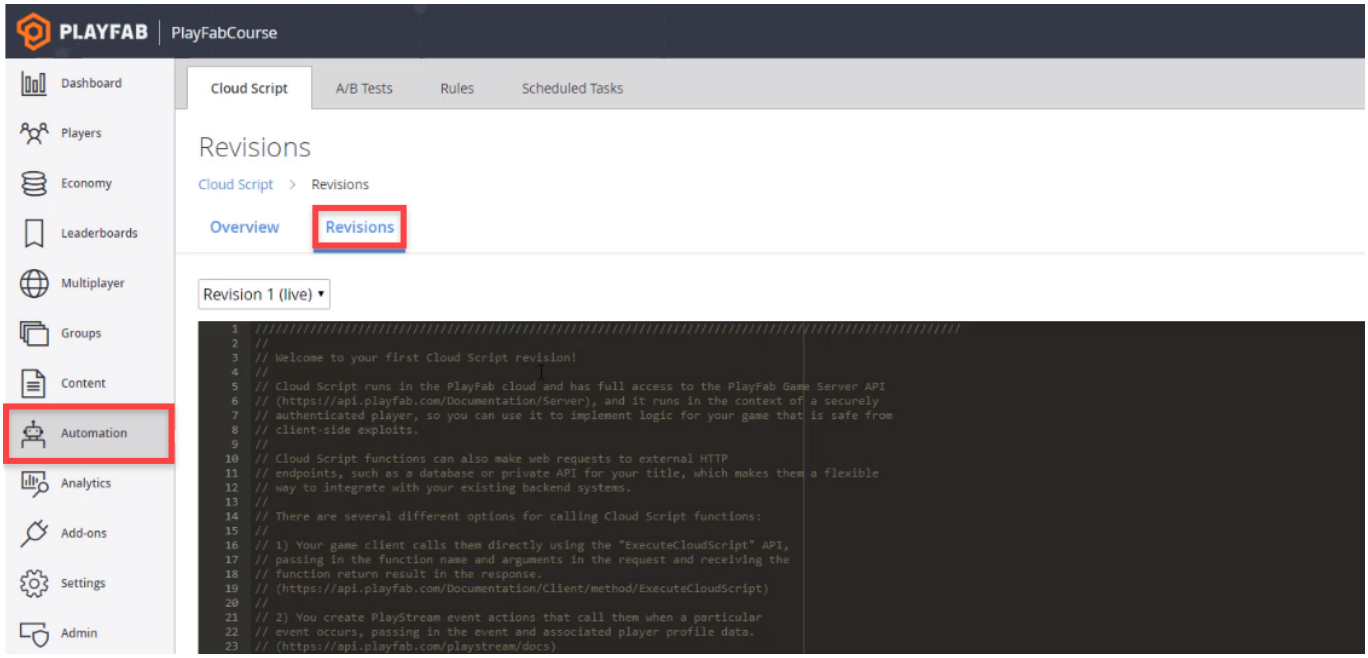
In the **Leaderboard** script, let's create a new function called **SetLeaderboardEntry**.

```
public void SetLeaderboardEntry (int newScore)
{

}
```

We're not going to directly add a new entry to the leaderboard here, as it could be a security issue. Instead of adding a new leaderboard entry on the client side, we'll do it server side. This is done using PlayFab's **Cloud Script**. This is a script in the PlayFab app that allows us to create functions we can call.

Go to the PlayFab dashboard and select the **Automation** page. Here, click on **Revisions** to go to the cloud script.



This is **JavaScript**. It's similar in some ways to C#, but differs mainly in the fact that it's less strict. When we create a variable, we define no data type. Let's create a new function at the top of the script called **UpdateHighscore**.

```
handlers.UpdateHighscore = function (args, context)
{

}
```

First, we can get our variables.

```
var score = args.score;
```

Then i'm going to create another function. This one will check whether the score is possible.

```
function ScoreIsPossible (score)
{
    var trueScore = -score;

    if(score > -1000)
        return false;
    else
        return true;
}
```

Back in the **UpdateHighscore** function, we can check this.

```
if(!ScoreIsPossible(score))
    return null;
```

Just like in Unity, we need to create an object to hold the info we're going to send to the API.

```
var request =
{
    PlayFabId: currentPlayerId,
    Statistics: [{ StatisticName: "FastestTime", Value: score }]
};
```

Then, we can send the request.

```
var result = server.UpdatePlayerStatistic(request);
```

Saving the Cloud Script

Let's start by saving the cloud script. Click the **Save As Revision** button.

Then we need to deploy it to the live version. Click on the **Deploy Revision** button.

Back in Unity

Back in the **Leaderboard** script, let's fill in the **SetLeaderboardEntry** function.

```
ExecuteCloudScriptRequest request = new ExecuteCloudScriptRequest
{
    FunctionName = "UpdateHighscore",
    FunctionParameter = new { score = newScore }
};

PlayFabClientAPI.ExecuteCloudScript(request,
    result => DisplayLeaderboard(),
    error => Debug.Log(error.ErrorMessage)
);
```

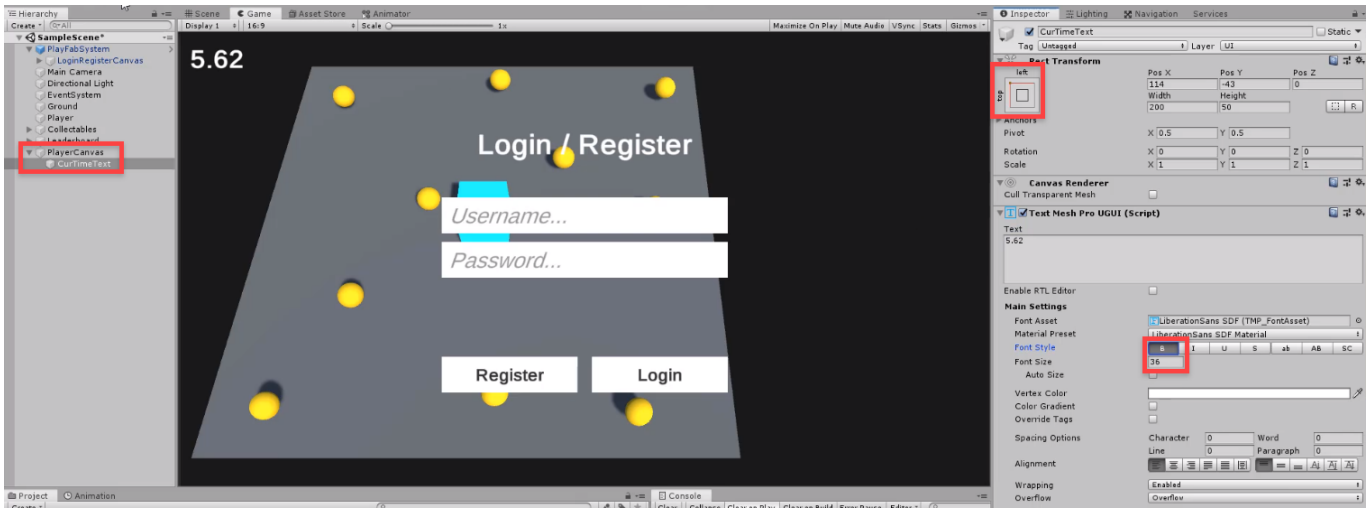
Let's do a quick fix in the **UpdateLeaderboardUI** function. Where we set the score text, let's change the end of the line of code to only display the number up to 2 decimal places.

```
... text = (-(float)leaderboard[x].StatValue * 0.001f).ToString("F2");
```

Player Canvas

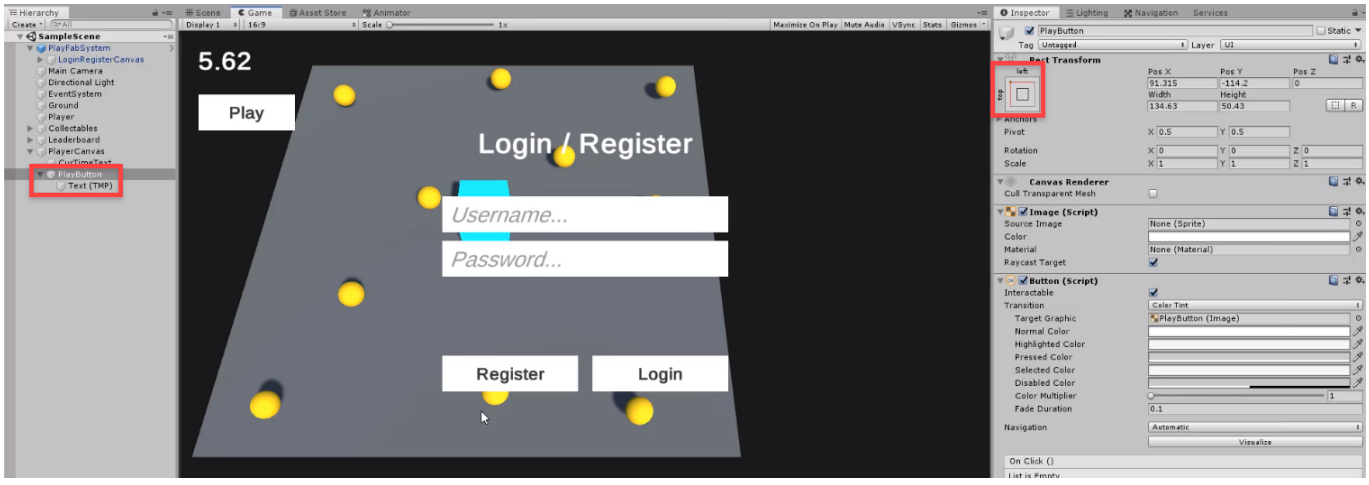
Create a new canvas called **PlayerCanvas**. Create a new Text – TextMeshPro element called **CurTimeText**.

- Set the **Anchoring** to *top-left*
- Set the **Font Style** to *Bold*
- Set the **Font Size** to *36*



Then create a new button called **PlayButton**.

- Set the **Anchoring** to *top-left*



PlayerController Script

In the **PlayerController** script, let's add some new variables.

```
public GameObject playButton;
public TextMeshProUGUI curTimeText;
```

In the **Begin** function, let's hide the play button.

```
playButton.SetActive(false);
```

And show it in the **End** function.

```
playButton.SetActive(true);
```

Let's also set a new leaderboard entry in the **End** function.

```
Leaderboard.instance.SetLeaderboardEntry(-Mathf.RoundToInt(timeTaken * 1000.0f));
```

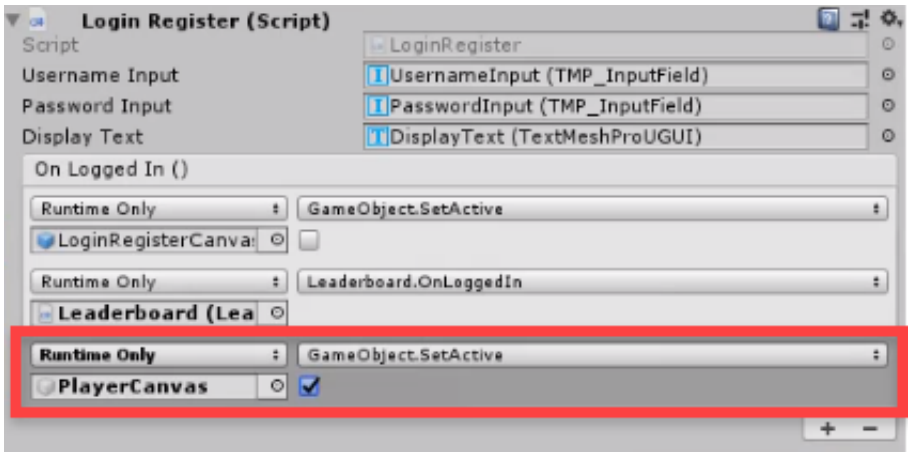
Inside the **Update** function, let's update the curTimeText.

```
curTimeText.text = (Time.time - startTime).ToString("F2");
```

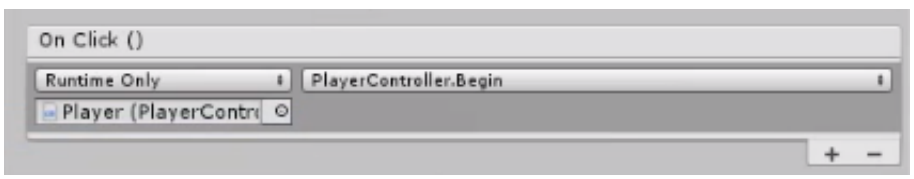
We don't want to move or update the text while not playing, so let's add some code at the top of the **Update** function.

```
if(!isPlaying)
    return;
```

Back in the editor, let's disable the player canvas and enable it in the OnLoggedIn function.



Let's also connect the Play button to the player controller's Begin function.



Congratulations on Finishing the Course

Let’s have a look at what we made and learned:

- We created a game in Unity that was connected to an online leaderboard.
- We used PlayFab’s leaderboard system.
- Cloud scripting was used to prevent client-side abuse.

Thanks for watching and good luck with your future games!