In this course we will be creating a simple game using Unity, based on what we learned in the Intro to Game Development course.

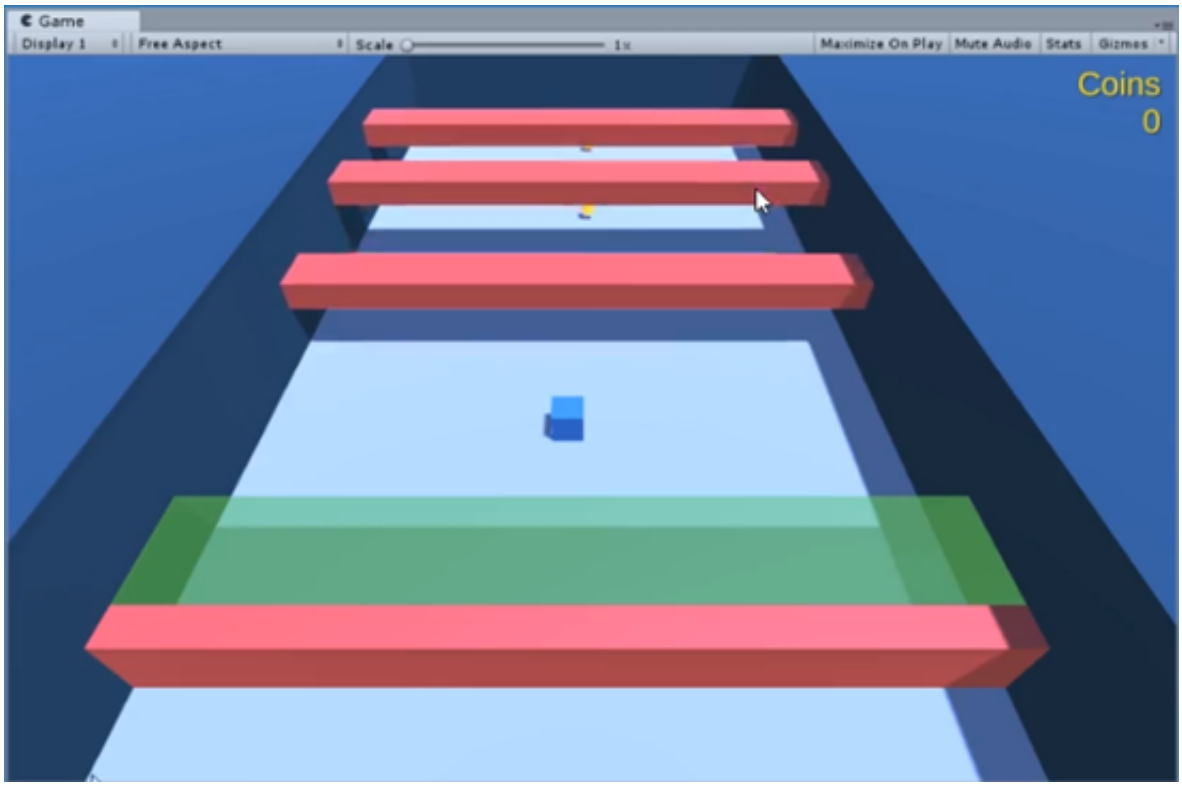
We will develop and build a very basic game idea so that we can understand exactly how all these things that we have learned so far tie in together to make an actual game.

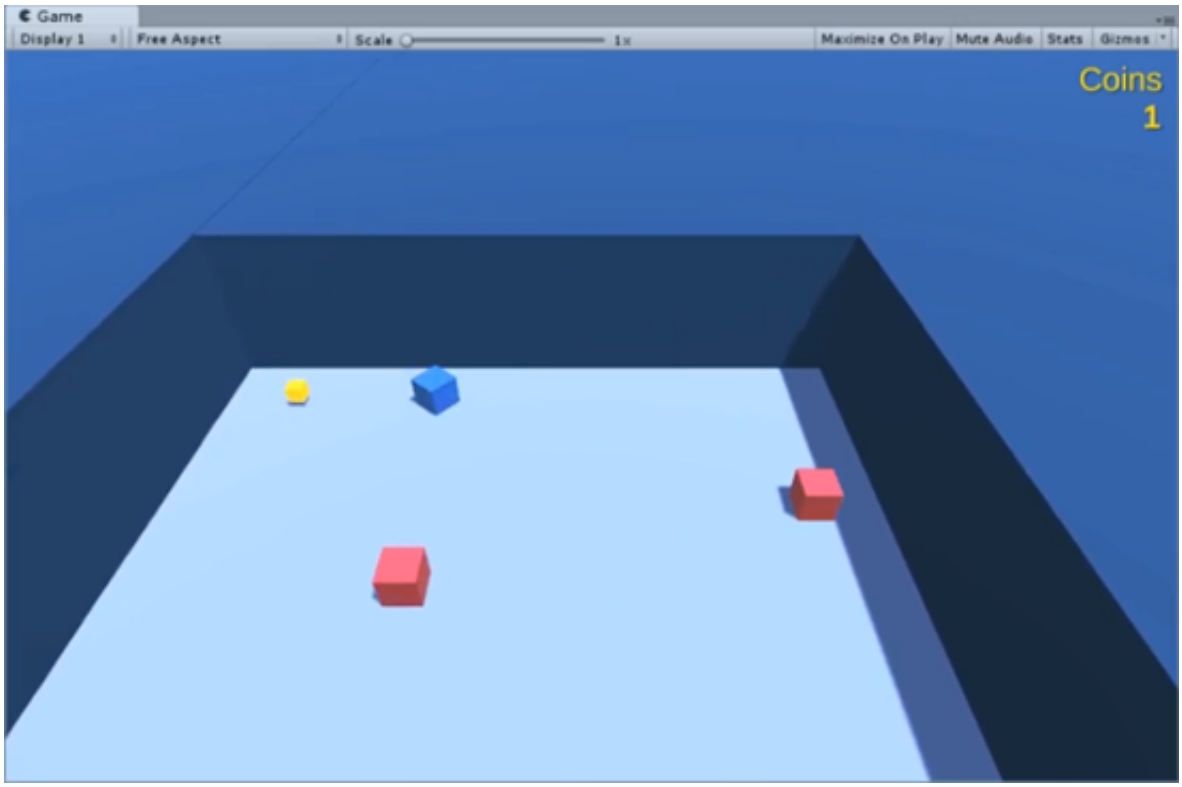
What is the Idea??

The idea is:

- Control a cube that can move and jump.
- Collect all the coins.
- **Watch out for enemies!**
- Simple attack for clearing enemies.
- With all the coins, go to level goal to progress to next level.

See the example of the game below in the screenshot:





The enemies will be moving using a **way point system.**

If the player does run into and enemy they will die and the game will be restarted.

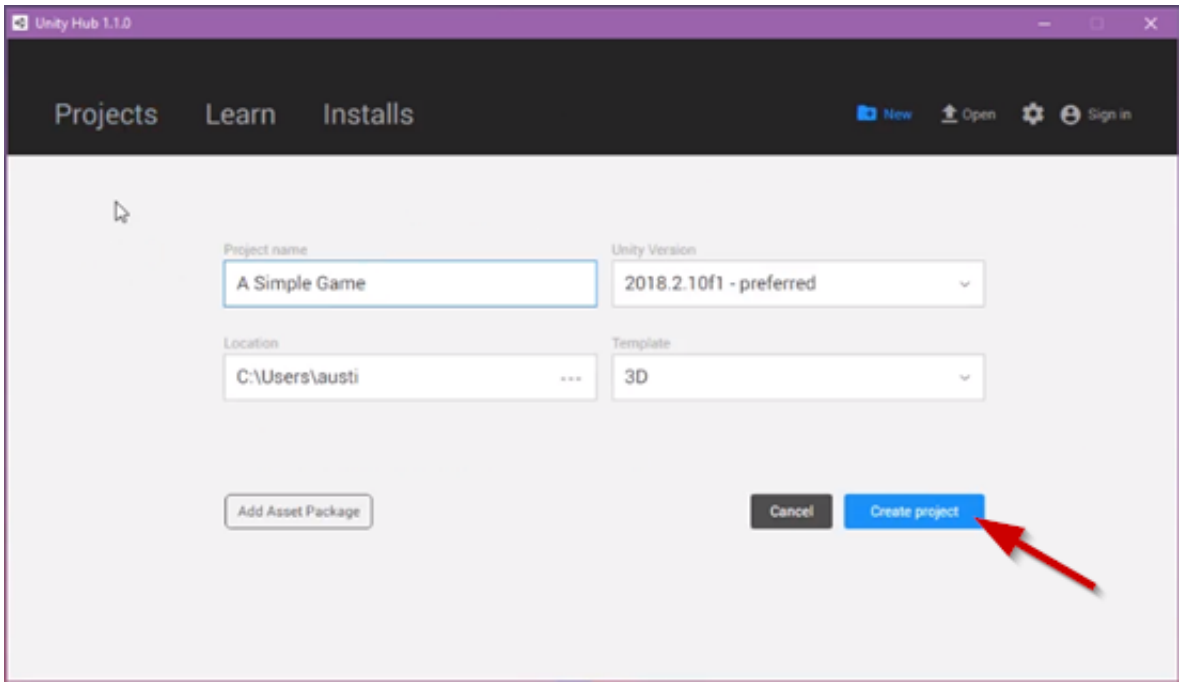
There will be a main menu and a pause menu as well in this game.

Even though this will be a very simple game in nature, making it will teach you a whole lot about Unity.

The First Steps

Create a new Unity 3D project with Unity Hub. The version of Unity that will be used for this course is Unity 2018.2.12f1.

The Project Name will be **“A Simple Game.”**



Click the **Create Project button** and the Unity editor will open.

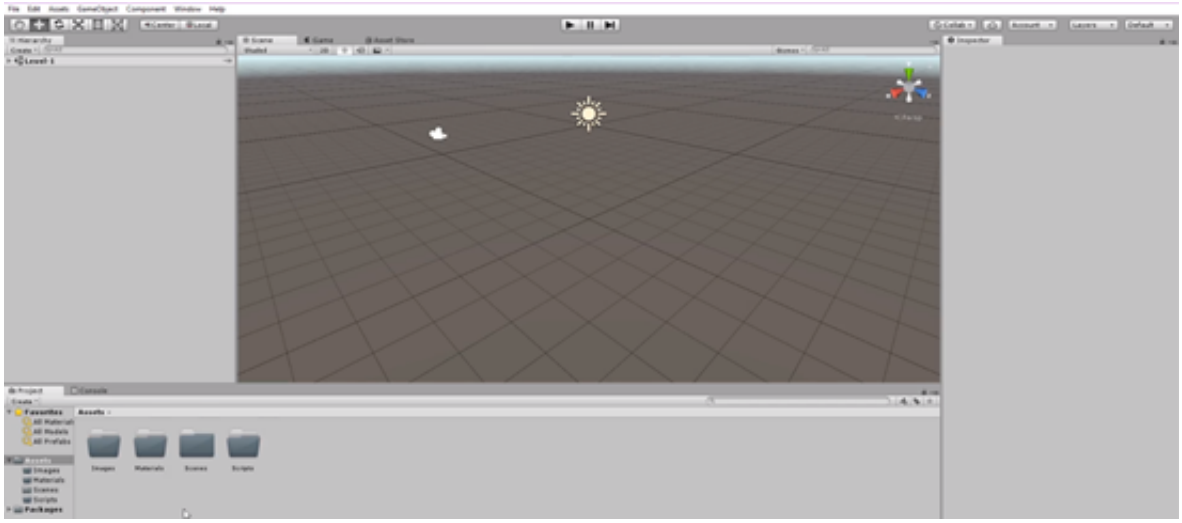
Once in the editor go ahead an create a new folder in the Assets folder and name the new folder **“Scripts.”**

Create a new folder for **“Materials.”**

There is already a Scenes folder in the project, and this is automatically created by Unity when you create a new project. This folder has the default scene in it.

The **SampleScene** can be renamed to **“Level-1.”**

Create a new folder and name this one **“Images.”**

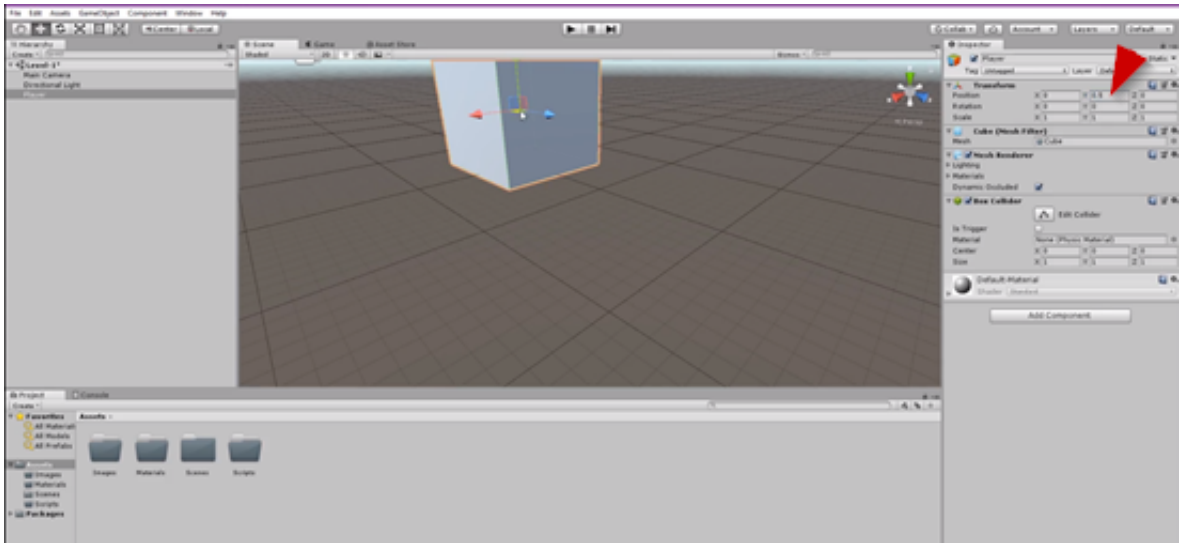


Creating the Player Object

We can create our player object by **right clicking in the Hierarchy>3D Object>Cube.**

The cube will represent the player. **Rename** the Cube object to **“Player.”**

We want the Player to be sitting on the ground, so we need to adjust the position of the **Player**. Change the **Y value on the position to 0.5.**



This is it for the setup, in the next lesson we will setup the scene and we will add in the ground, some walls, and adding in some obstacles.

In this lesson we will be setting up our level layout.

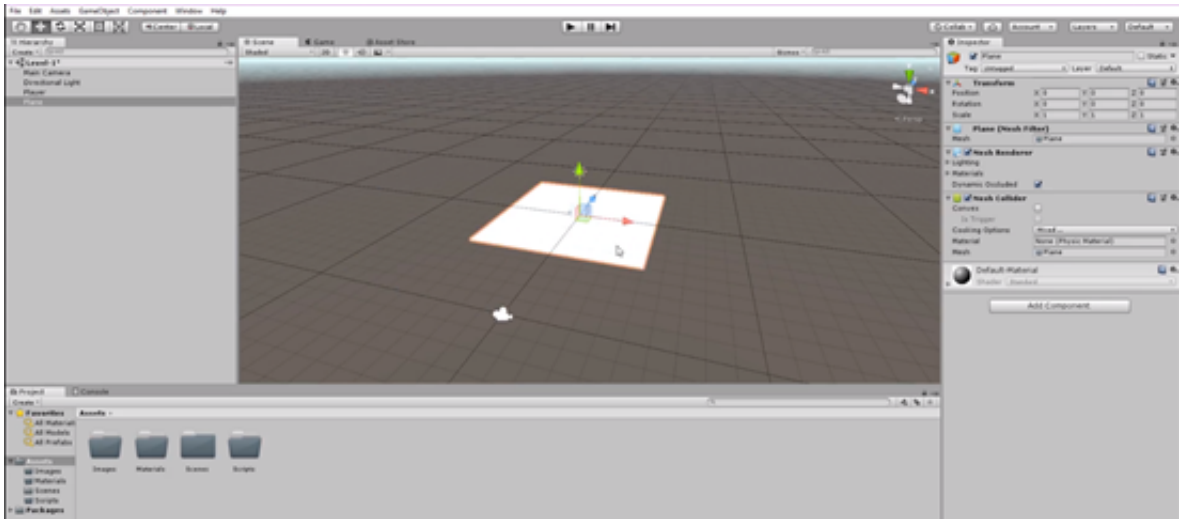
Level Layout

- Move, rotate, scale, and snap objects
- Place coins
- Place enemies
- Create walls
- Place goal

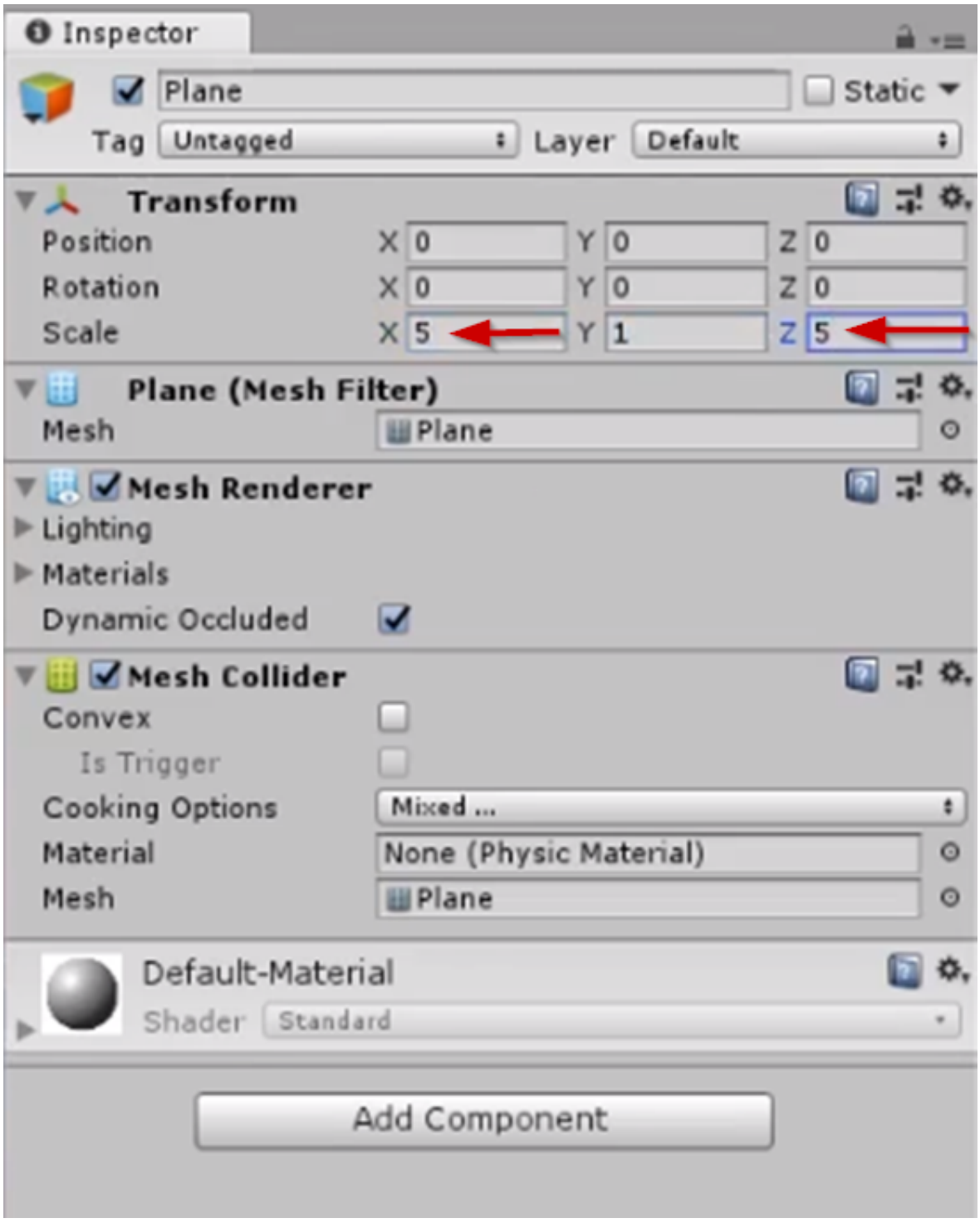
In order to do all of this in Unity we will take everything that we have learned from the previous course and apply it to the laying out of the level.

Make sure you have the **“A Simple Game”** project open in the Unity editor and go ahead and **right click >3D Object>Plane.** We will be using the Plane object as the ground. Position the Plane at 0,0,0.

The player should now be positioned right on top of the ground now.

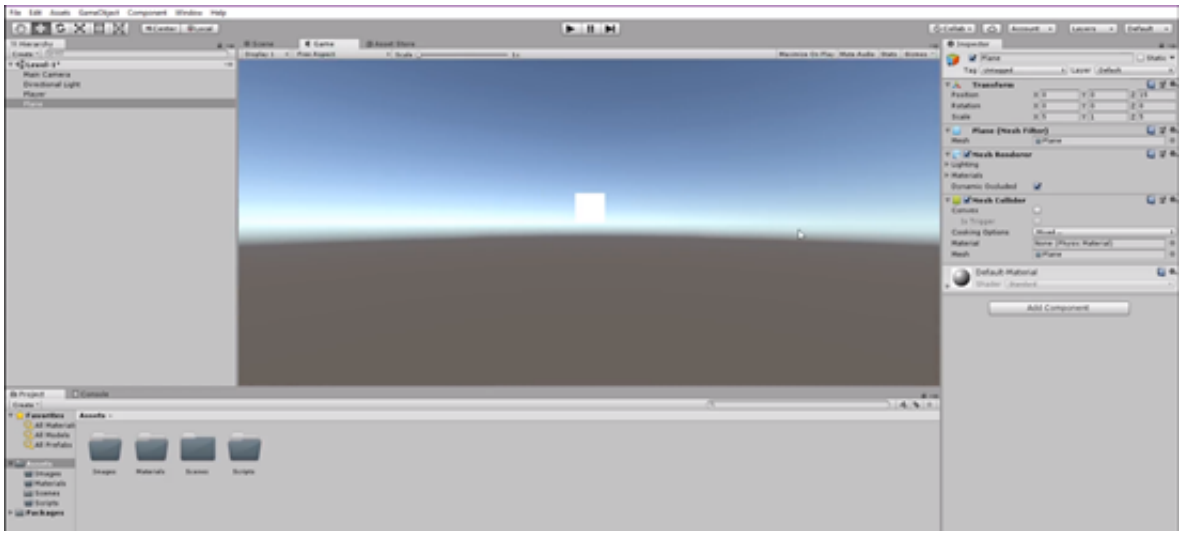


The ground isn’t going to be big enough for us so we need to scale it up. **Adjust the scale** on the plane to be **5 on the X and 5 on the Z. The Y value can be left at 1.**

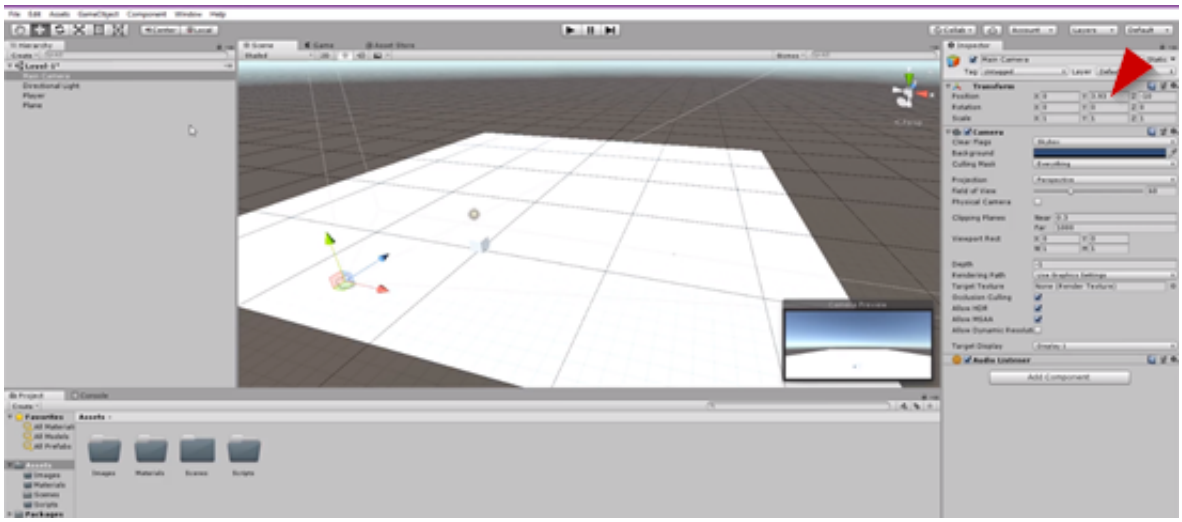


Then adjust the **Z value on the position for the Plane to be 15.**

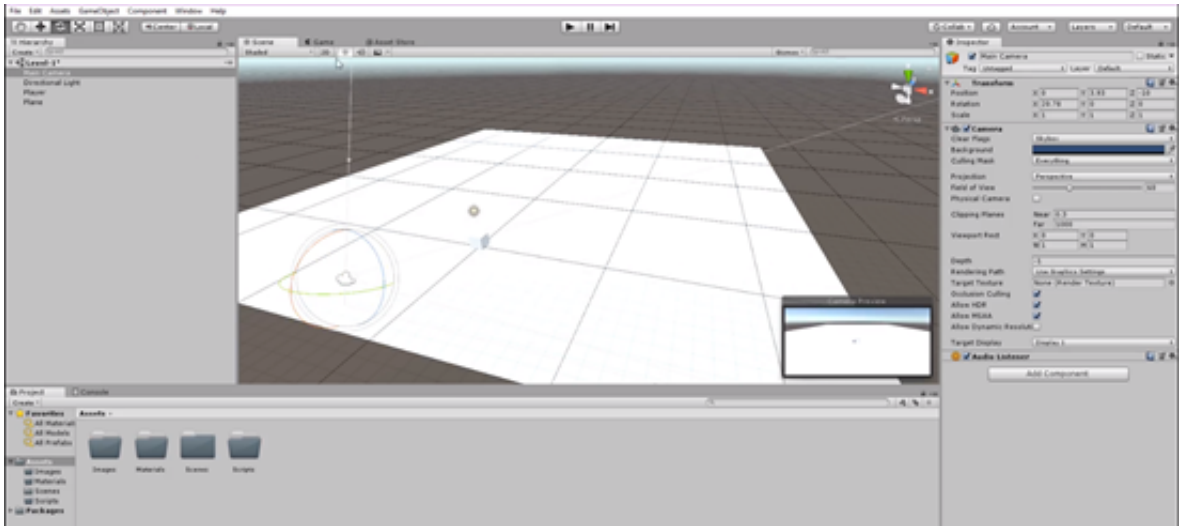
If you switched to the game view now you wouldn't even be able to see the ground below the player and this is because of the angle we're looking at.



Now adjust the position of the **Main Camera by placing the Y value on position to 3.93.**



Rotate the Main Camera on the X axis so that the Main Camera is now looking down. **The X axis value should now be at 20.78.**

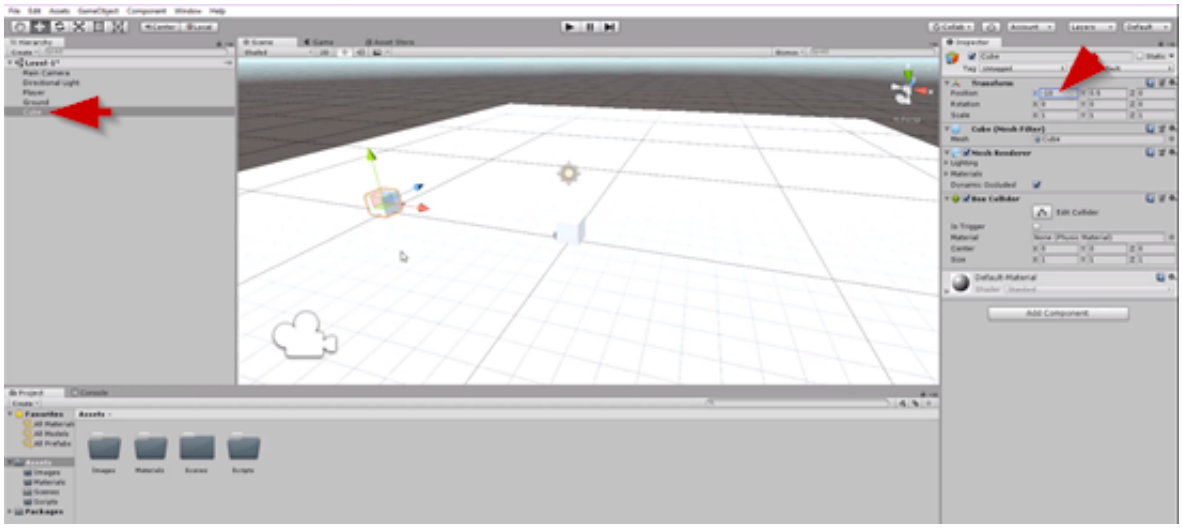


Rename the Plane to **“Ground.”**

We will now create some walls, these walls will be used to “encase” the level and keep the player from falling off.

Create a 3D Object>Cube, double click on the Cube in the Hierarchy to easily locate it after creation, and you will see that its actually way down below the ground.

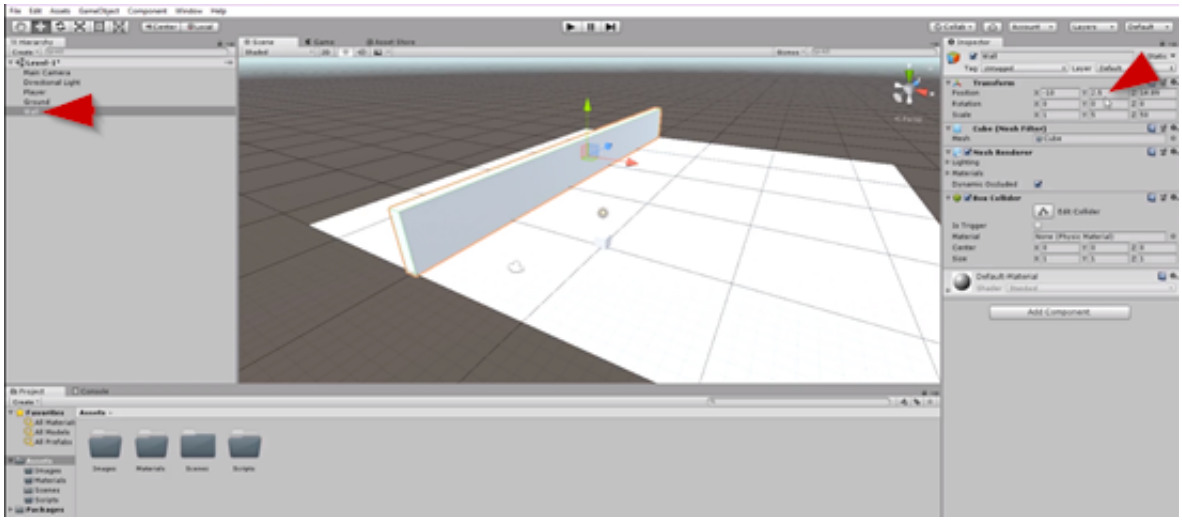
Adjust the position on the Transform component for the cube to **be 0 on the X, 0.5 on the Y, and 0 on the Z.** Now the Cube we just created is right where the Player is, and now we want the drag the cube over to right where we want the walls to be. **Make sure the Cube is now at -10 the X position.**



Now this isn’t much of a wall right now, so we need to **scale it along the Z axis and make sure its about 50 for the Z value. Position it at 15 on the Z axis.**

Rename the Cube to **“Wall.”**

Adjust the **scale** on the Y axis for the wall to be **5**. Adjust the Y value for the position to be **2.5.**

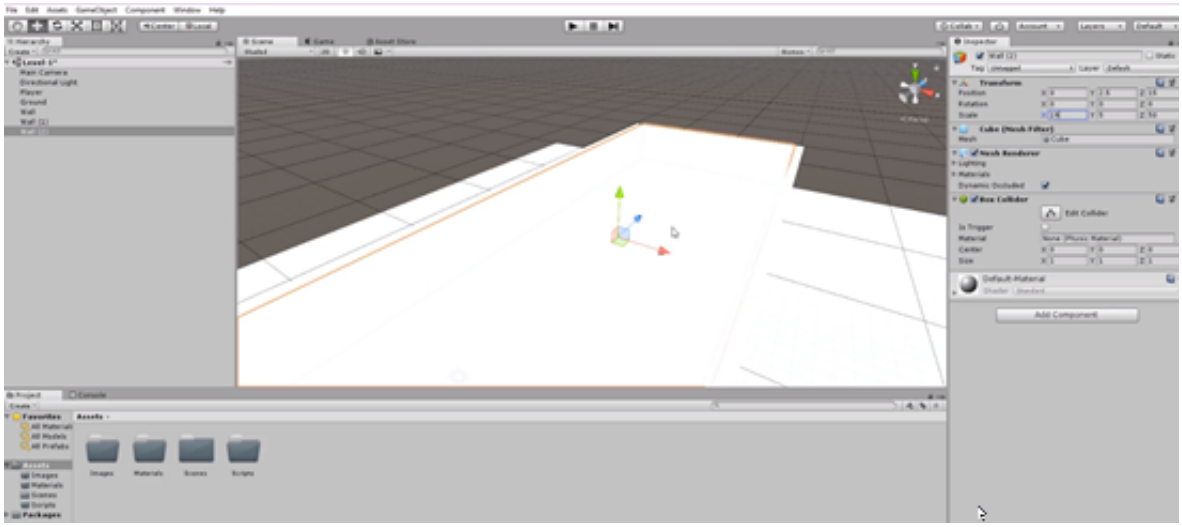


Now **duplicate the Wall object** so we will now have a second wall. You can **duplicate an object by hitting Ctrl + D** on the keyboard, and drag the Wall(1) game object over to the other side of the

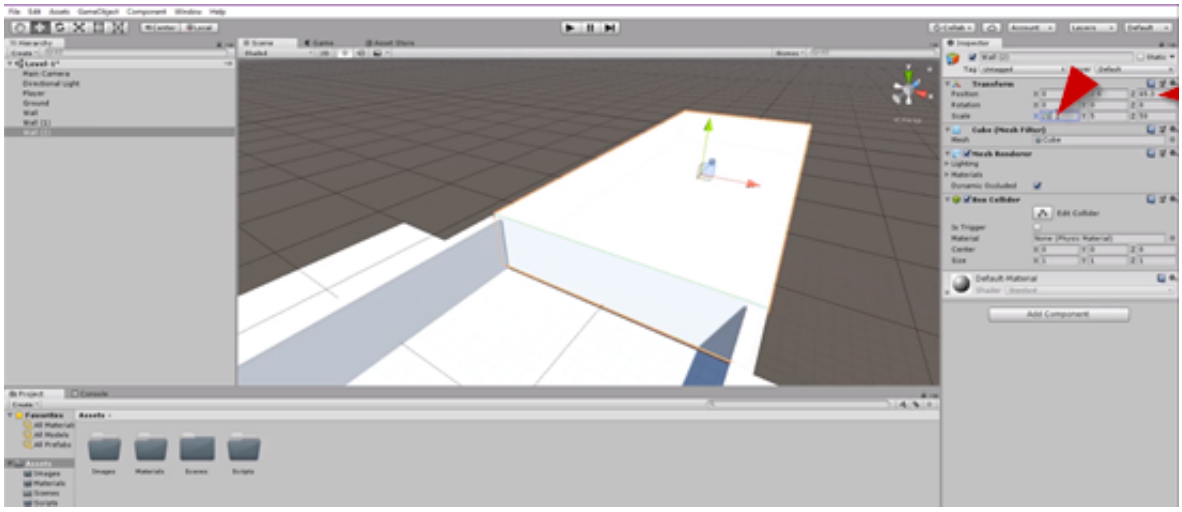
player. **The position on the X axis should be 10.**

The reason we are using such precise numbers here is because you don't want to have a mess of a level.

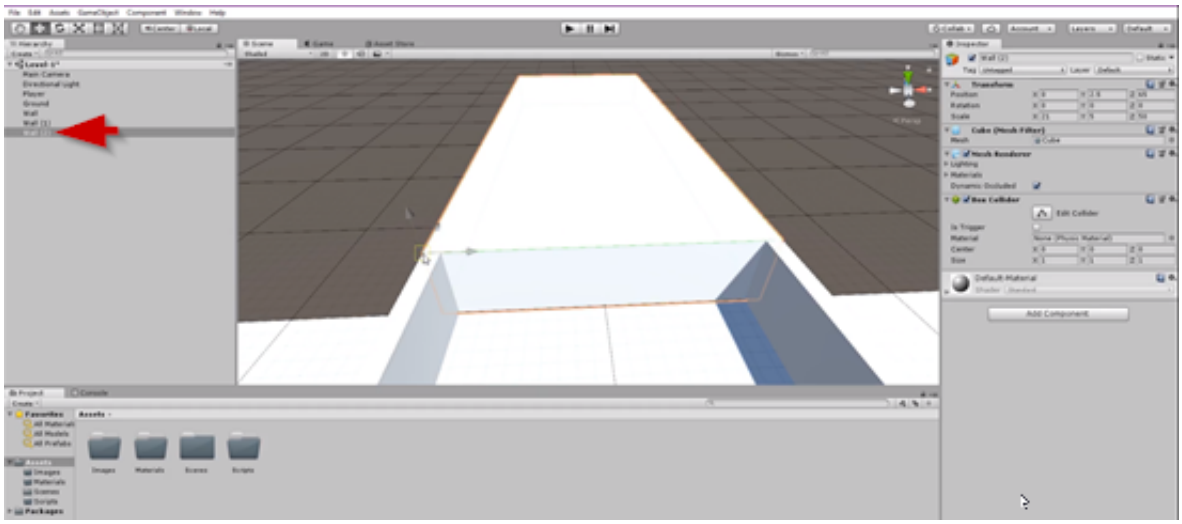
Now we need to cap off the walls, and there are a couple of ways to do this. Make a **duplicate of Wall(1)** and you will now have **Wall(2)**, make sure to **position Wall(2) at 0 on the X axis**. So we could rotate Wall(2) or scale it about the X axis to cap off the wall so what I will do is **scale it up to 19 on the X axis.** This will put it right up against both walls in the scene and it'll be nice and snug.



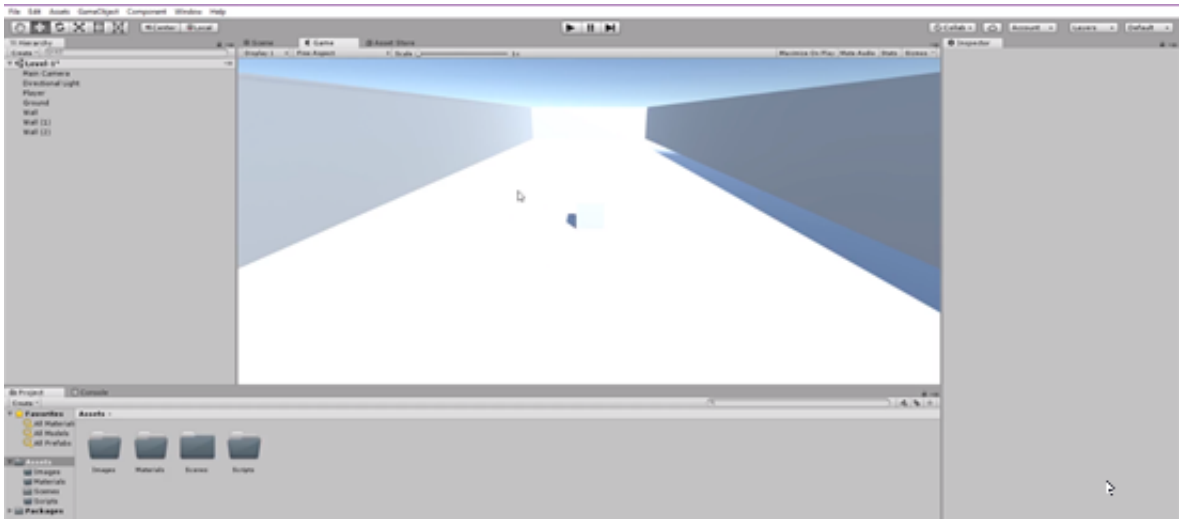
Then what we want to do is drag **Wall(2)** out and make the **scale** on the **X value to be 21.**



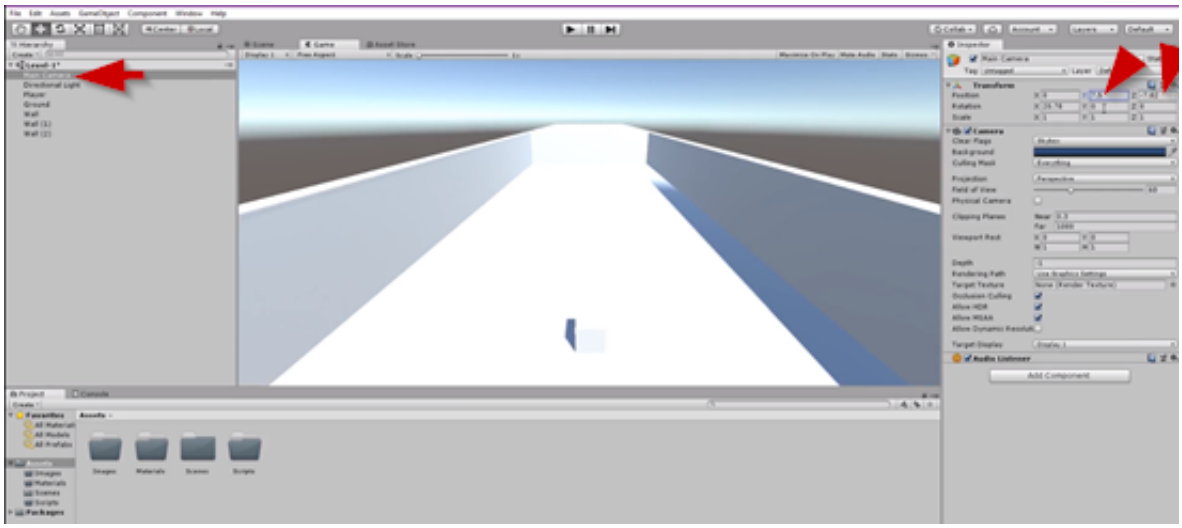
You can use the **vertex snapping tool** and snap it to the vertex.



Now looking at the game view you should see this in the scene:

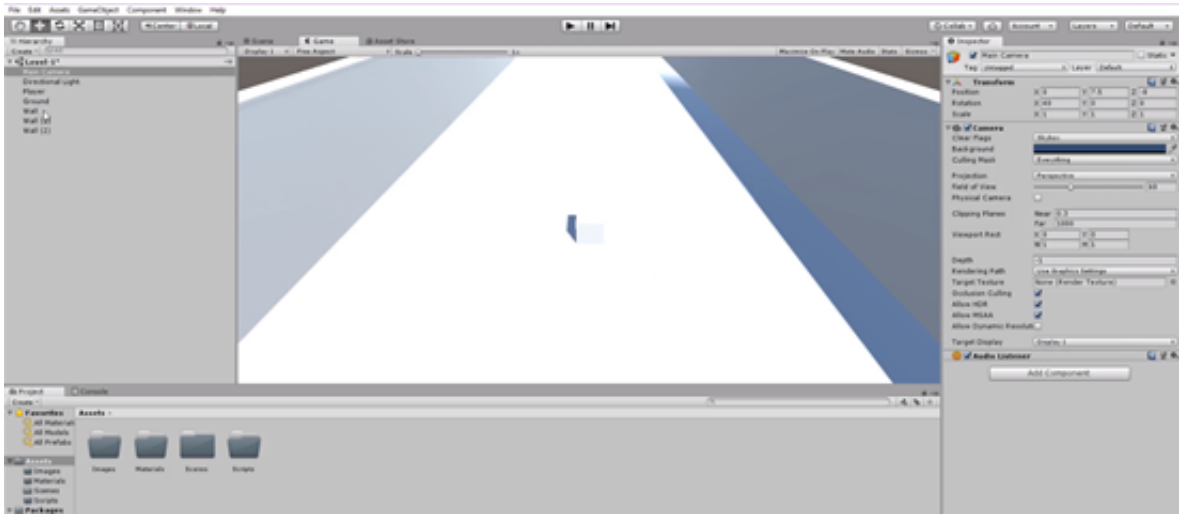


Now select the **Main Camera** and **adjust the position on the Y value to be 7.5 and the Z should now be at -7.82.**



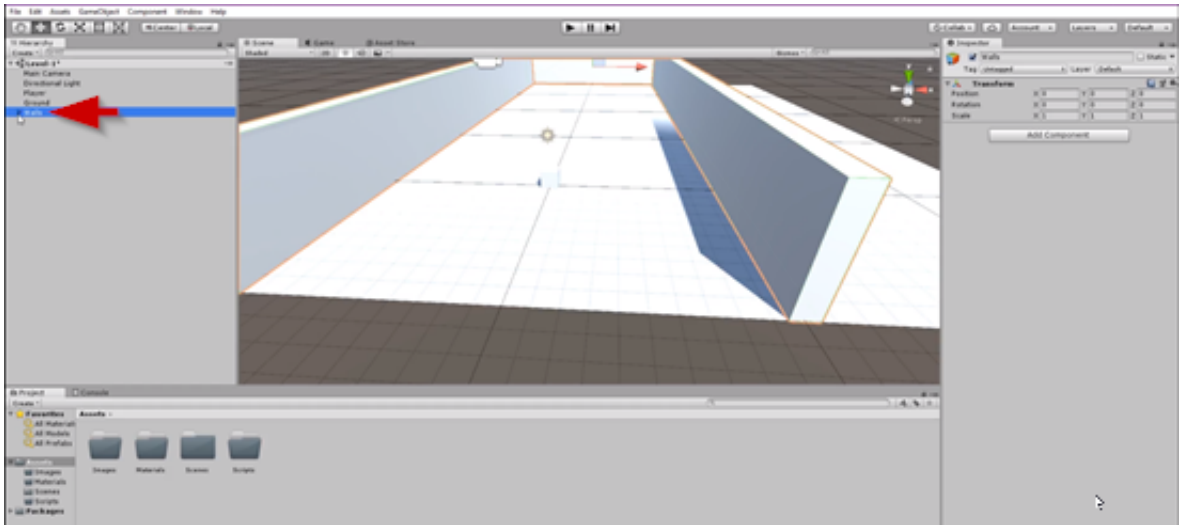
Rotate the **Main Camera on the X axis to 40.** Adjust the position to be **7.5 on the Y value and**

-8 on the Z value.



Now if you look in the Hierarchy we now have three wall objects, and what we want to do is **create an empty object. Right click in the Hierarchy>Create Empty.** Because this is an empty game object it will have no other components attached it but a transform component. **We can use this empty game object to organize the Hierarchy.**

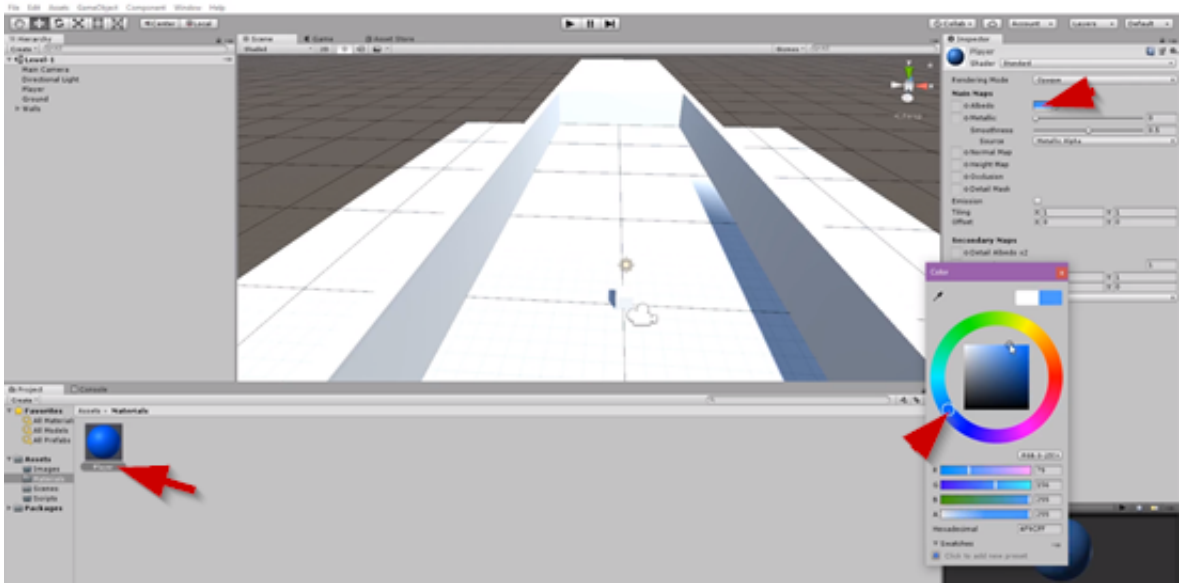
Select all the wall game objects and **drag and drop them into the empty game object.** Then the empty game object will contain all the walls we created. First make sure the empty game object is **positioned at 0,0,0.** Once you have the wall objects placed inside the empty game object you can rename it to **“Walls”** and you can **collapse it in the Hierarchy by hitting the arrow next to it.**



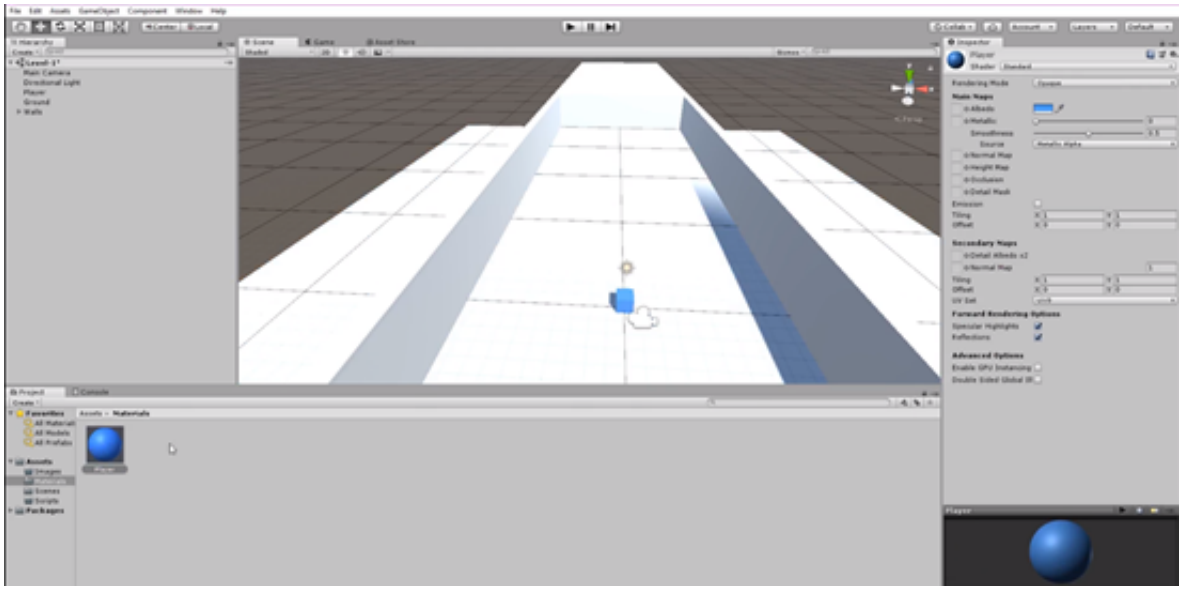
Now the Hierarchy is more organized and we can even move all the walls at once.

Creating Materials

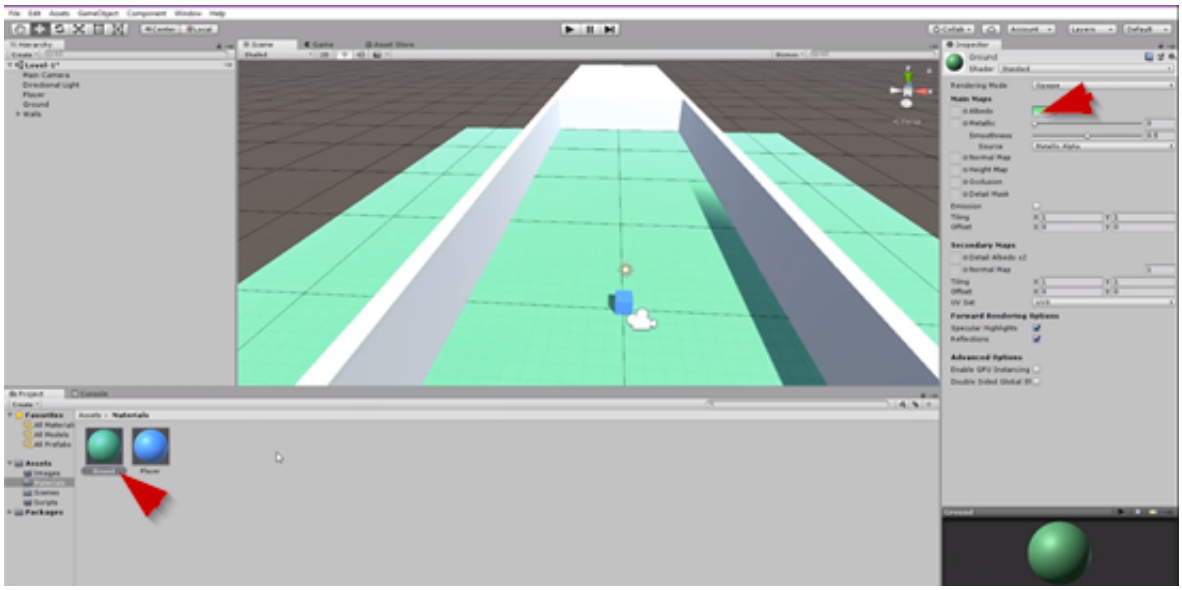
Navigate to the Materials folder and **create a new material and name it “Player” you can make the Player blue for now using the color picker under Albedo.**



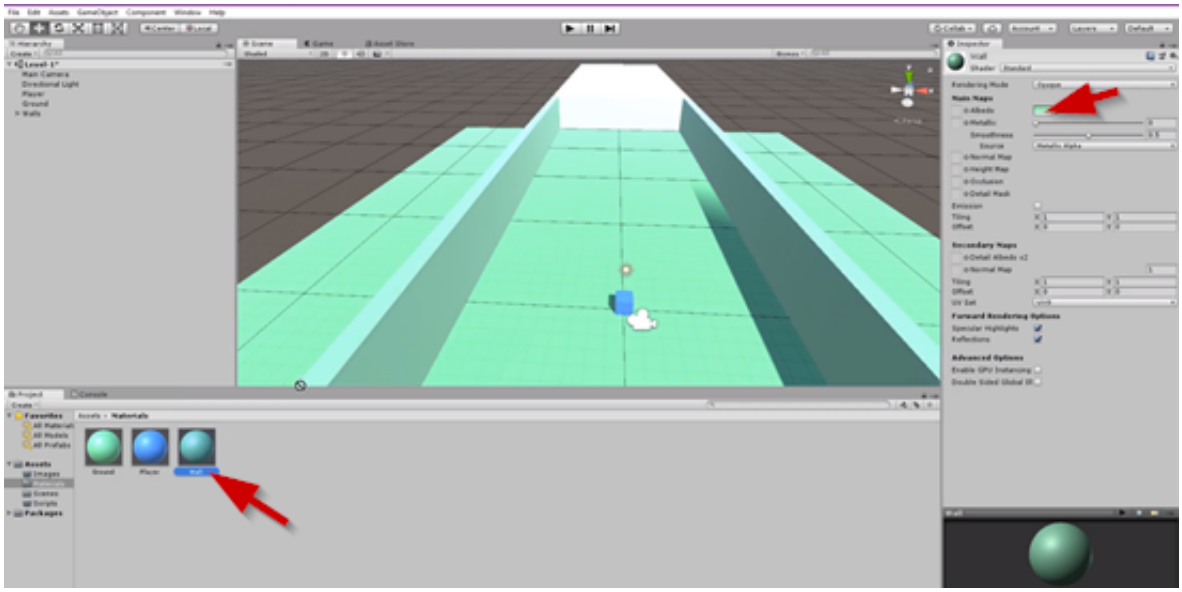
Now **drag and drop the Player material** directly onto the **Player** in the scene, and we will now have a blue player.



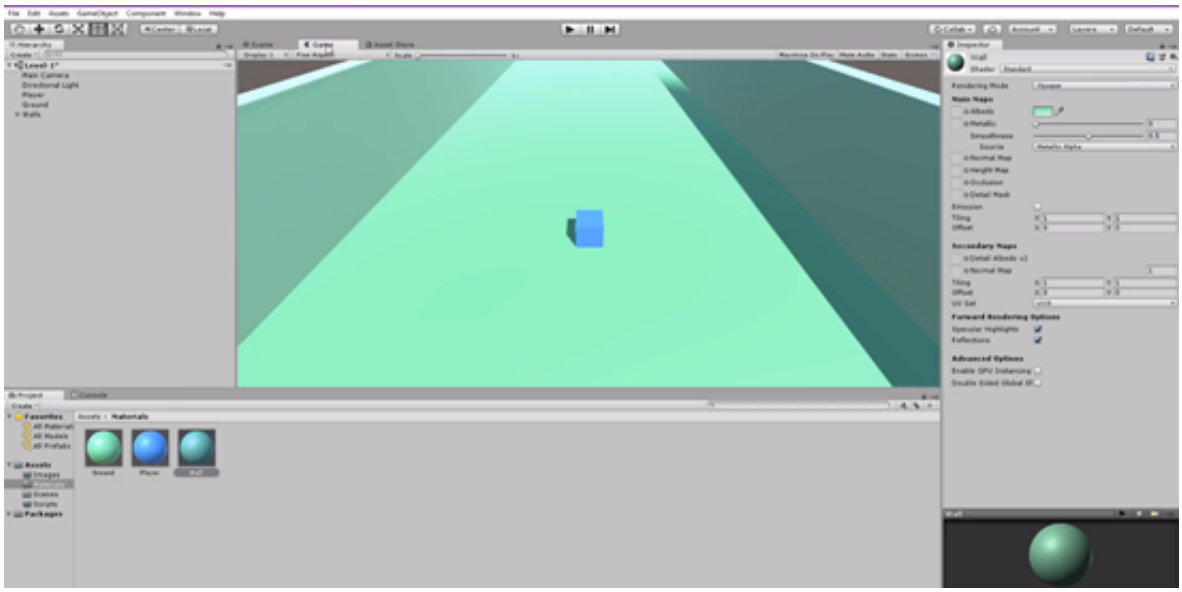
Duplicate the Player material using **Ctrl + D**, and name the new material **“Ground”** make the ground a green color using the color picker under Albedo, **drag and drop the Ground material onto the ground object in the scene.**



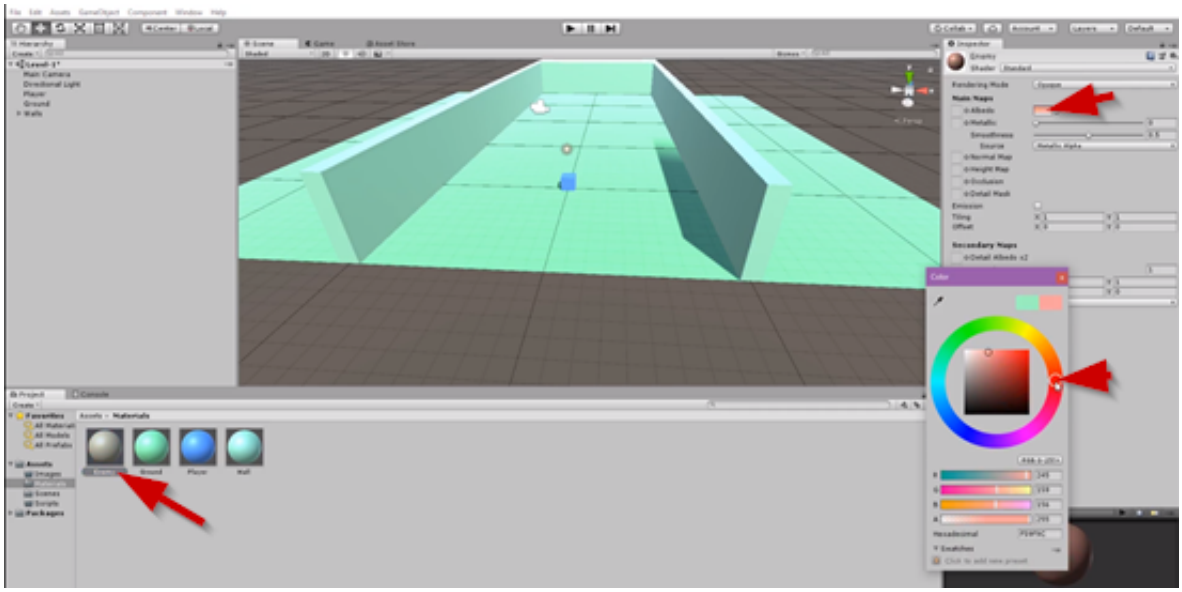
Duplicate the Ground material and name this one **“Wall”** and make the walls a similar color to the ground using the color picker under Albedo, and drag and drop the wall material onto the walls in the scene.



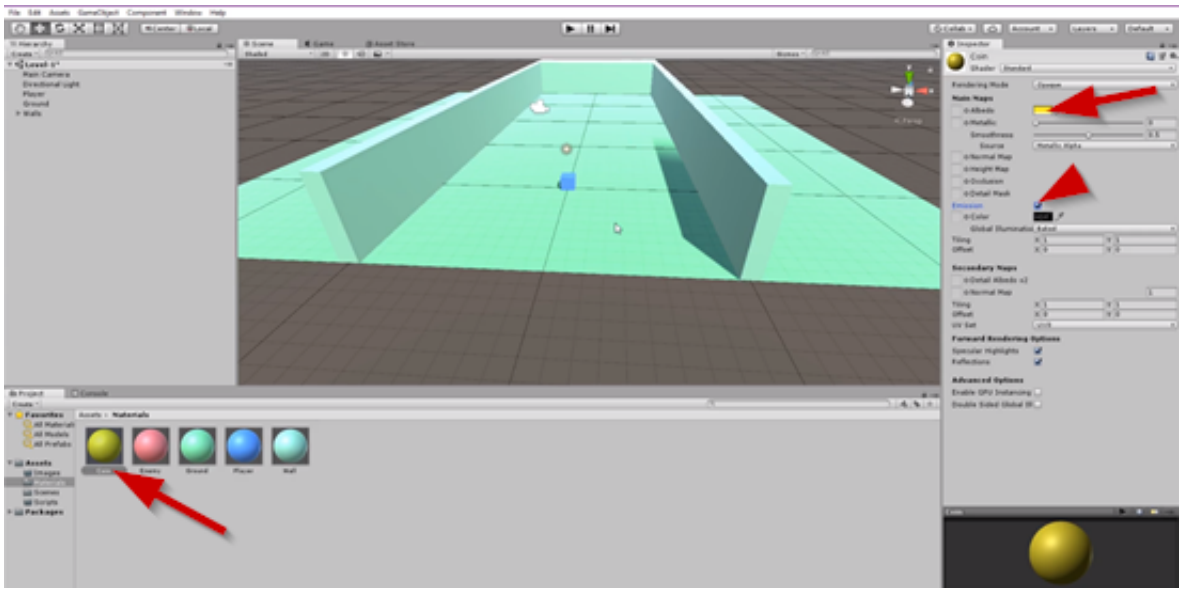
Now take a look in the Game view:



Now **create an enemy material** and we will assign this material to all the enemies we will be creating. **Duplicate the Wall material and rename the material to “Enemy” and make it a red color using the color picker under Albedo.**

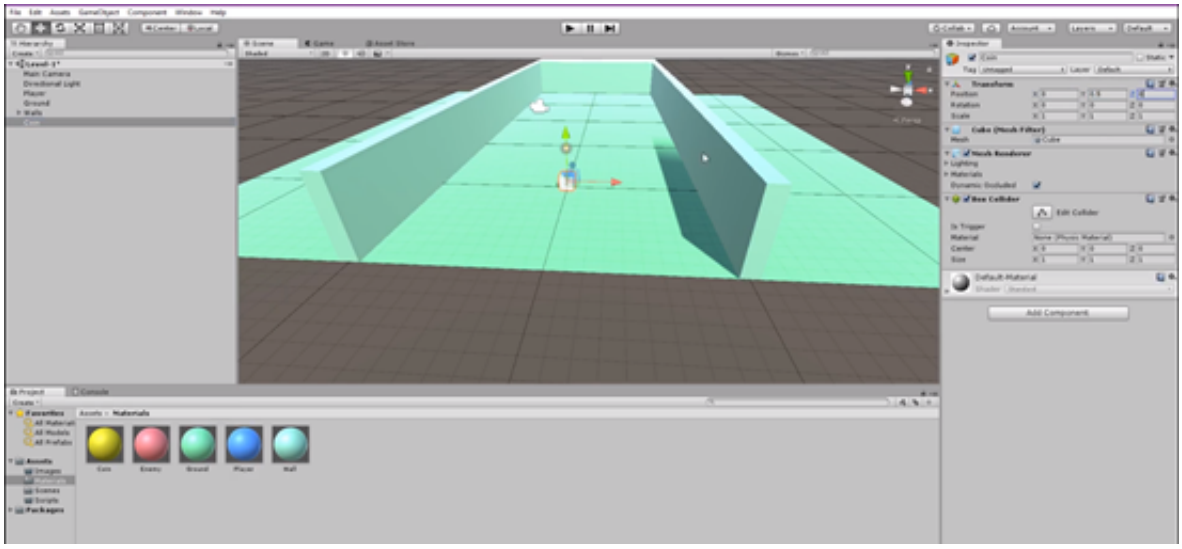


Then **duplicate** the Enemy material and **rename this material to “Coin”** these coins will be a bright yellow color and instead of just being a yellow color they will emit a yellow color, so make sure Emission is toggled and later on we will make changes to the emission later on as we progress with creating the project.

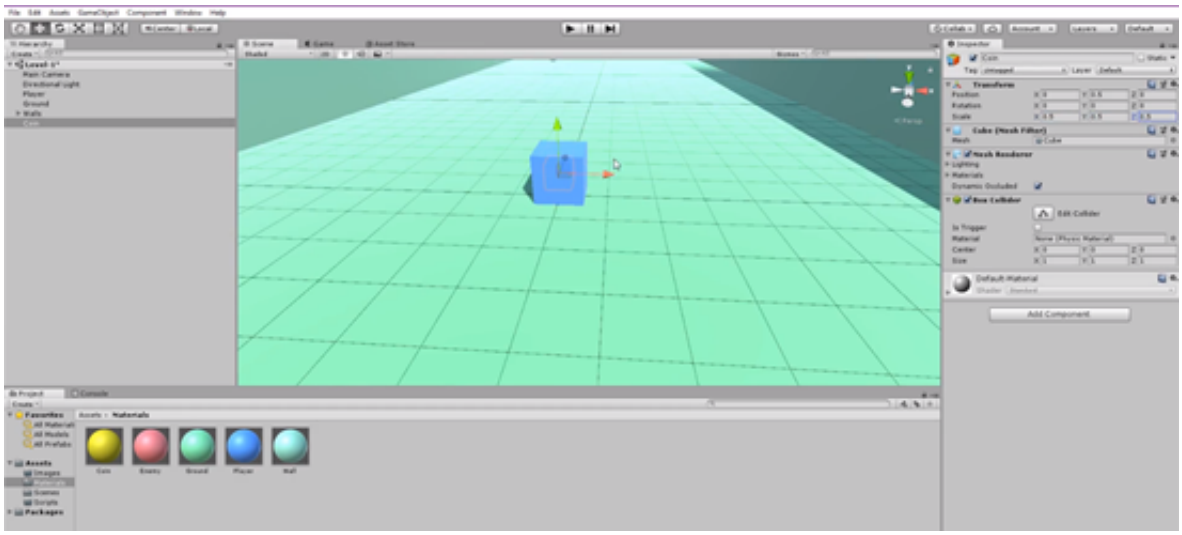


Coin Creation

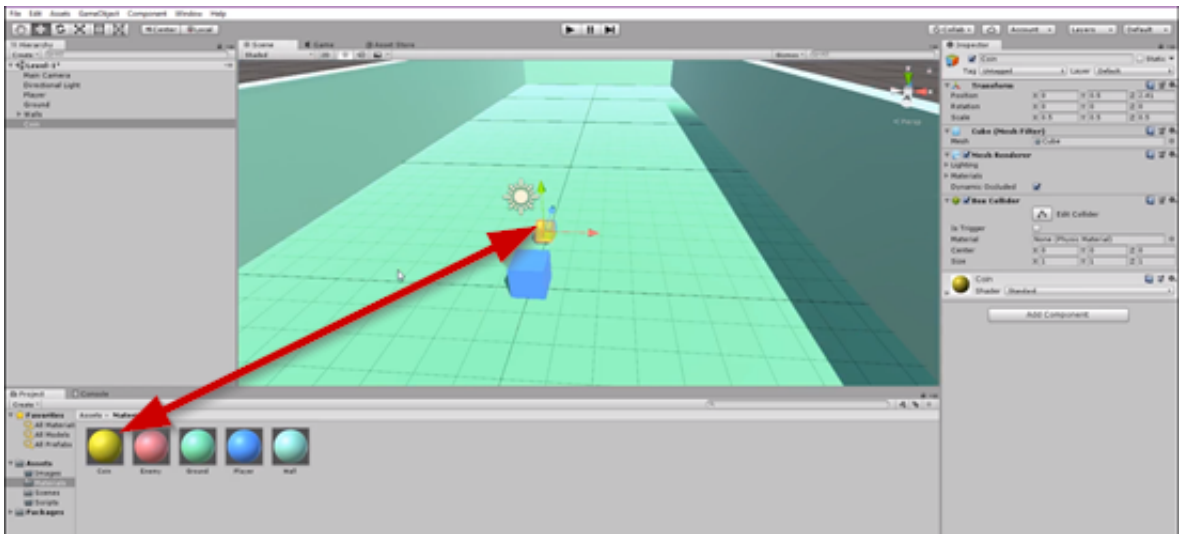
Right click in the Hierarchy>3D Object>Cube and rename it to **“Coin.” Position** it to **0 on the X, 0.5 on the Y, and 0 on the Z.**



The coin will not be the same size of the player so we need to adjust the scale and make it half the size of the player. **The scale can be adjusted to 0.5 on the X, 0-.5 on the Y, and 0.5 on the Z.**

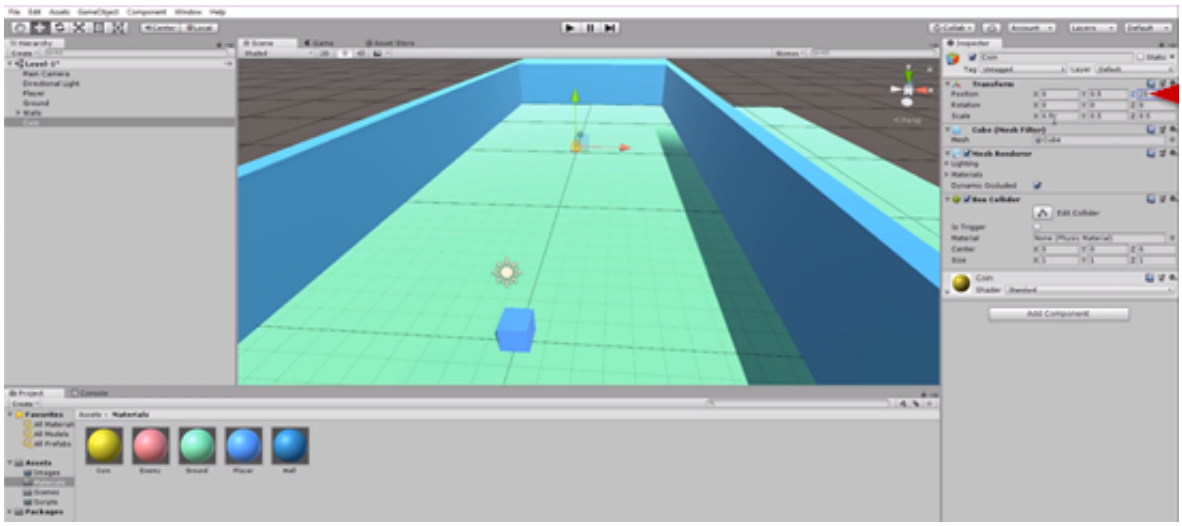


Drag the coin out in front of the player and **assign the Coin material to it by dragging and dropping it onto the coin.**

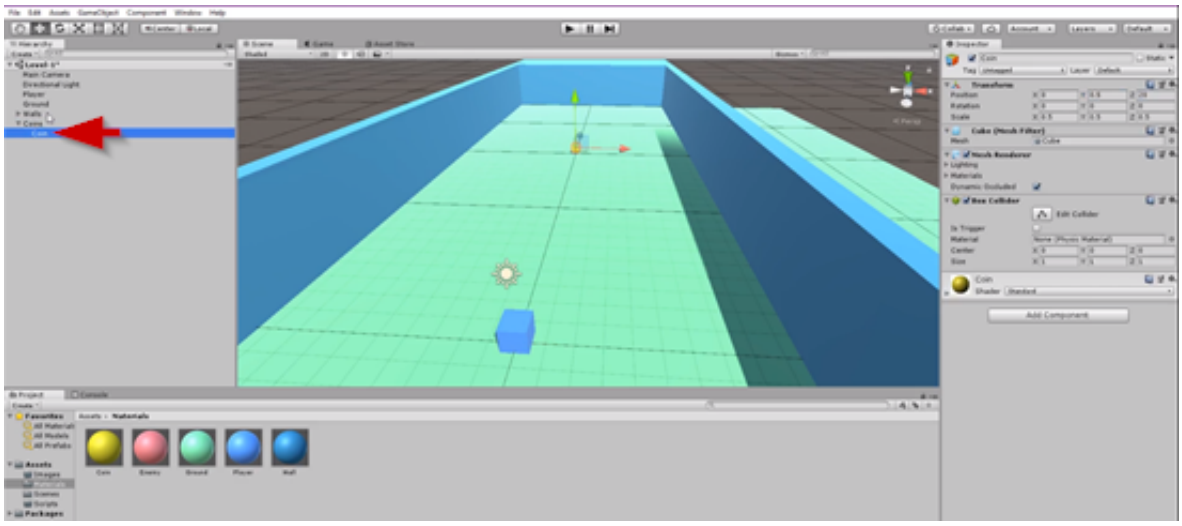


If you are not happy with the color of the walls you may change them.

Adjust the position on the **Coin on the Z axis to 20.**



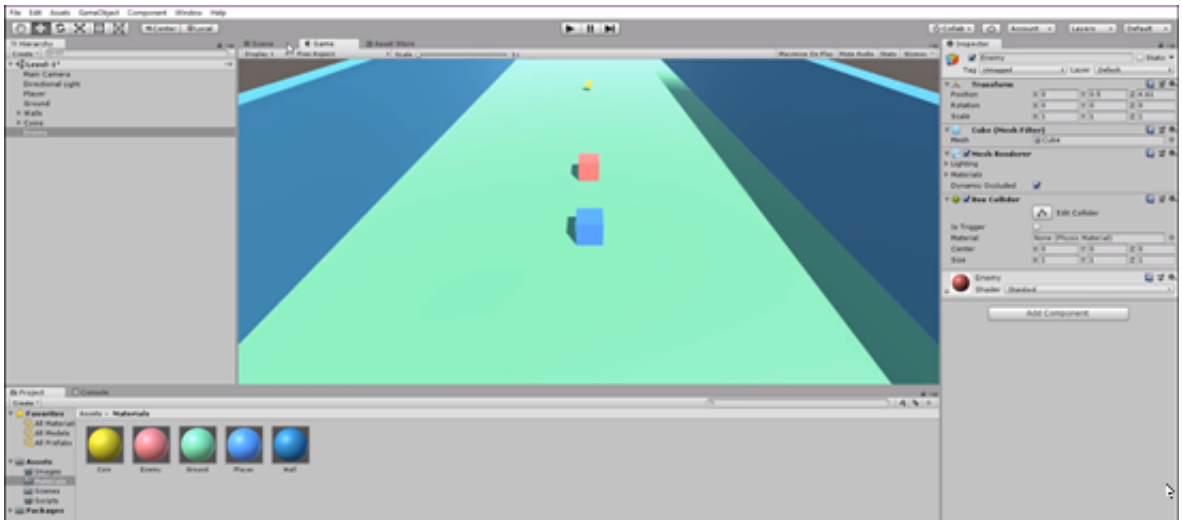
Create an empty game object and **rename it to “Coins”** we will use this to organize the coins in the scene. **Adjust the position of the Coins game object to 0,0,0.** Drag and drop the Coin into the Coins game object in the Hierarchy.



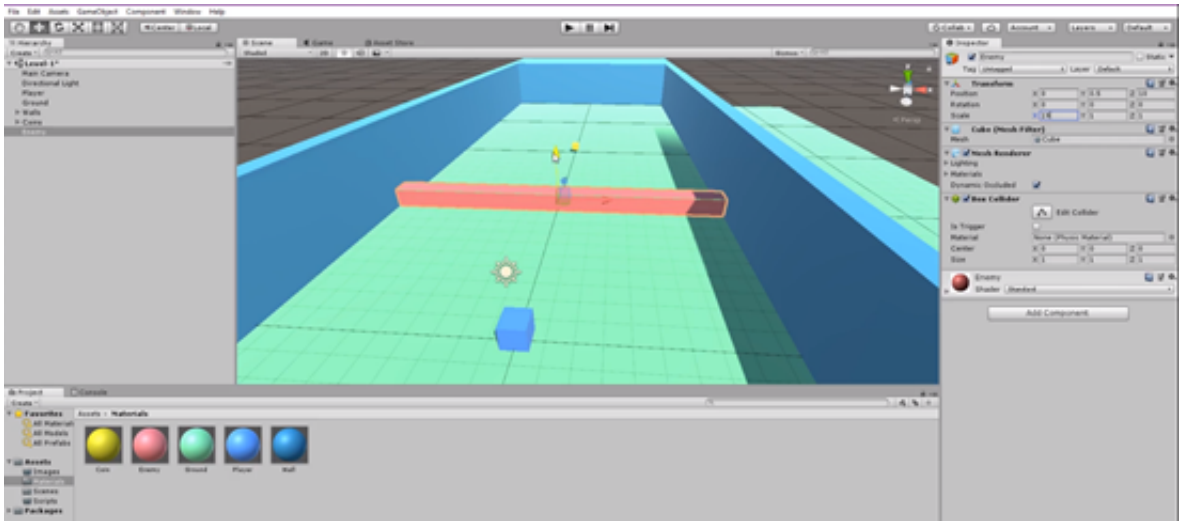
We are doing this because if you have a lot of coins in the levels you want to have some type of organization for your hierarchy.

Creating the First Enemy

Create the enemy now, and we will use a 3D cube for this, **rename** it to **“Enemy”** make sure the **position is at 0 for the X, 0.5 on the Y, and 0 on the Z value.** Assign the enemy material to the enemy in the scene.



Position the **Enemy at 10 on the Z axis.** Scale it to **19 on the Z**, so that it fits right inside the walls.



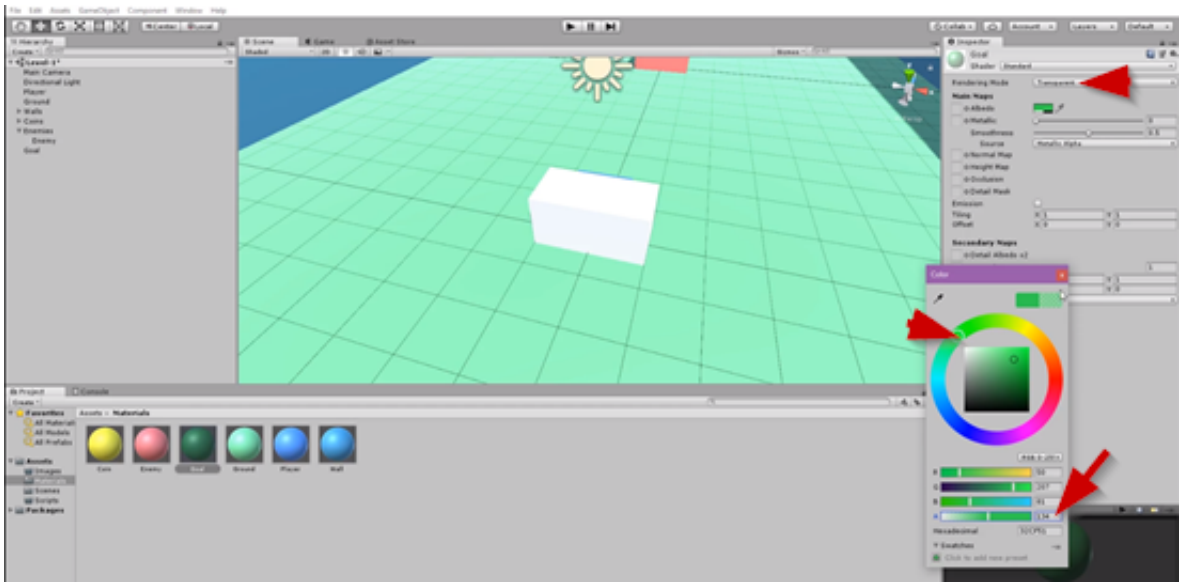
In order to keep all our enemies organized we will **create an empty game object and rename it to “Enemies”** make sure its **positioned at 0,0,0.** Drag and drop the Enemy we have in the Hierarchy into it. Make sure the **Enemy is at position 0.5 for the Y and 100 for the Z.**

Goal Creation

The last thing we need to create is the goal we need to get back to once we have collected all the coins in the level.

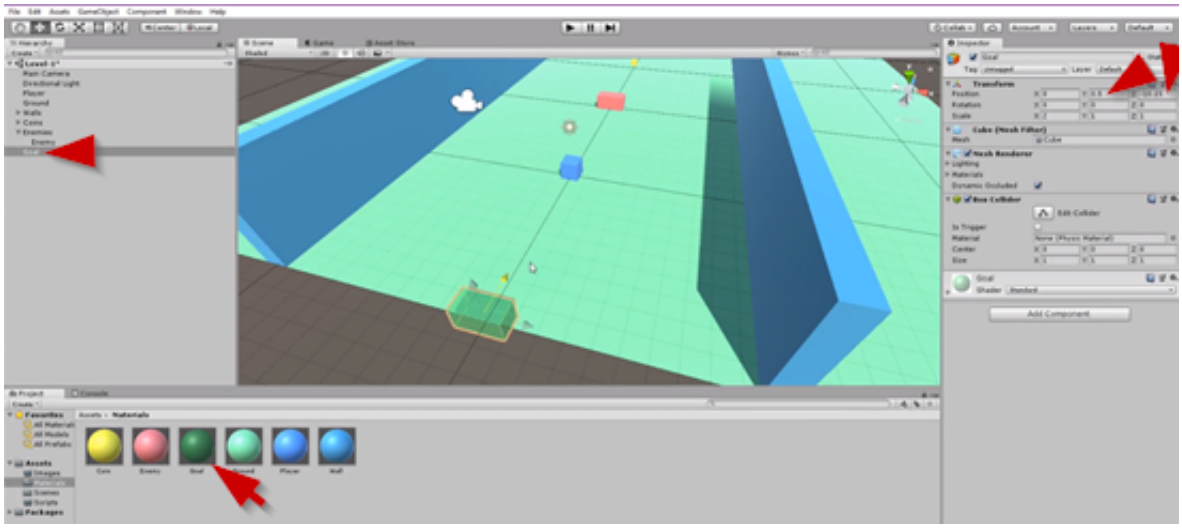
Keep in mind thought that you may put your goal anywhere in the level. In my case though I am going to put the goal back here behind the player so that the player will have to go collect the coins and then make it back to the goal.

Create a 3D Cube and **rename it to “Goal” the Goal will be positioned at 0 for now**, and we will create a new material for the goal, but it will be **slightly transparent.** You can duplicate the ground material and name it to **“Goal”** make the goal a different green color and **change the Rendering Mode from Opaque to Transparent** and bring the **Alpha value** down in the color picker, **134 on the Alpha** should be good.

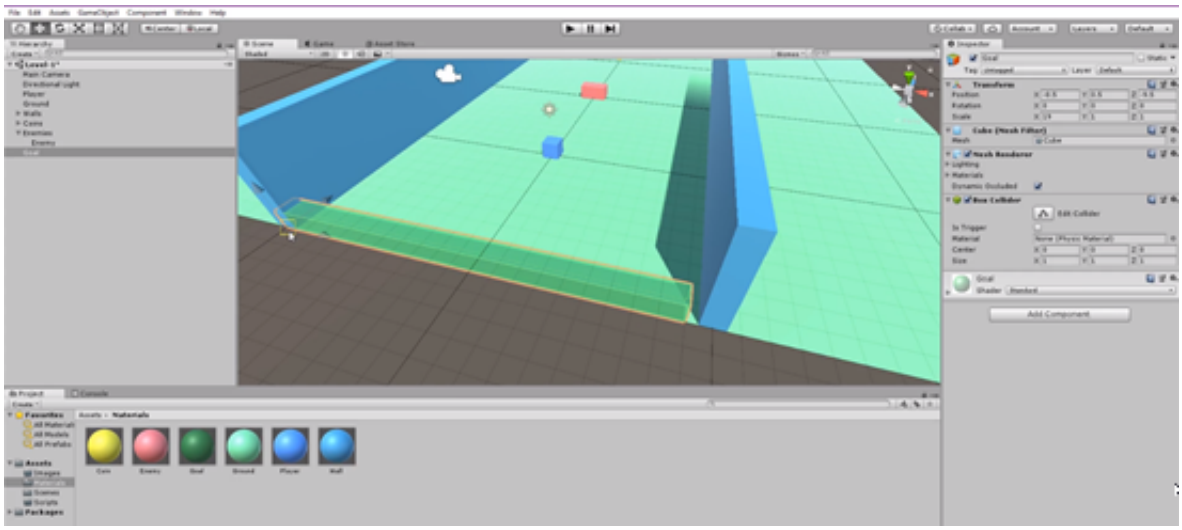


Assign the transparent material to the goal object in the scene.

Position the Goal wherever you want in the scene.



Adjust the scale on the **X value for the Goal to be 19** so it fits snugly inside of the walls, and use the **vertex snapping tool** to make sure it fits even more snugly between the walls.

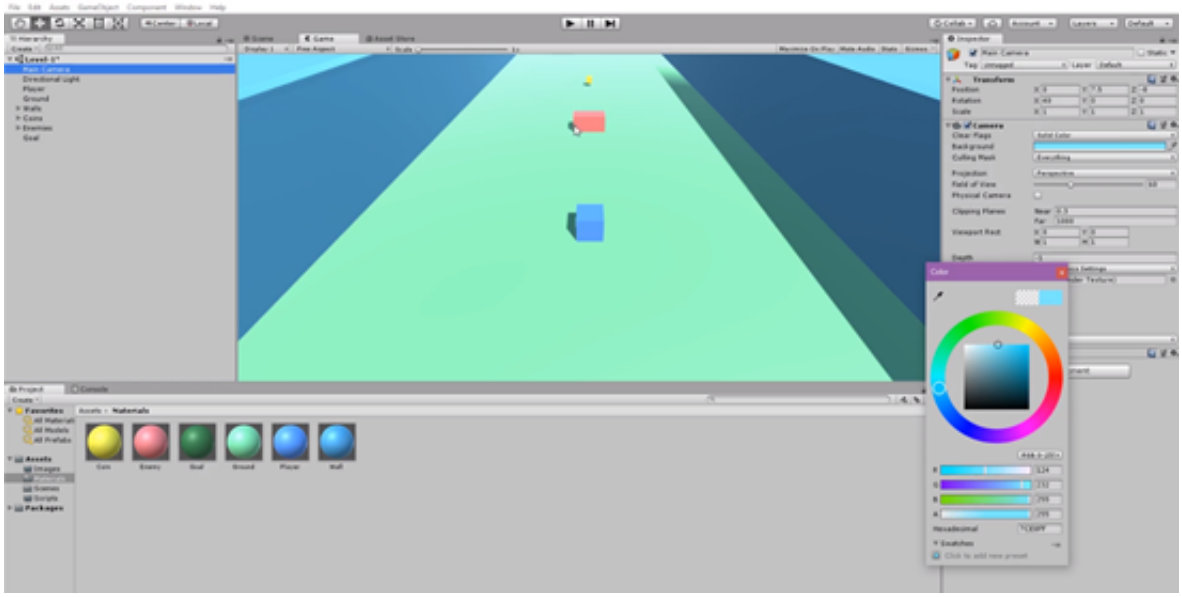


You can even make the Goal a little taller by adjusting the scale for the **Y value to be 3**. Use the vertex snapping tool again.

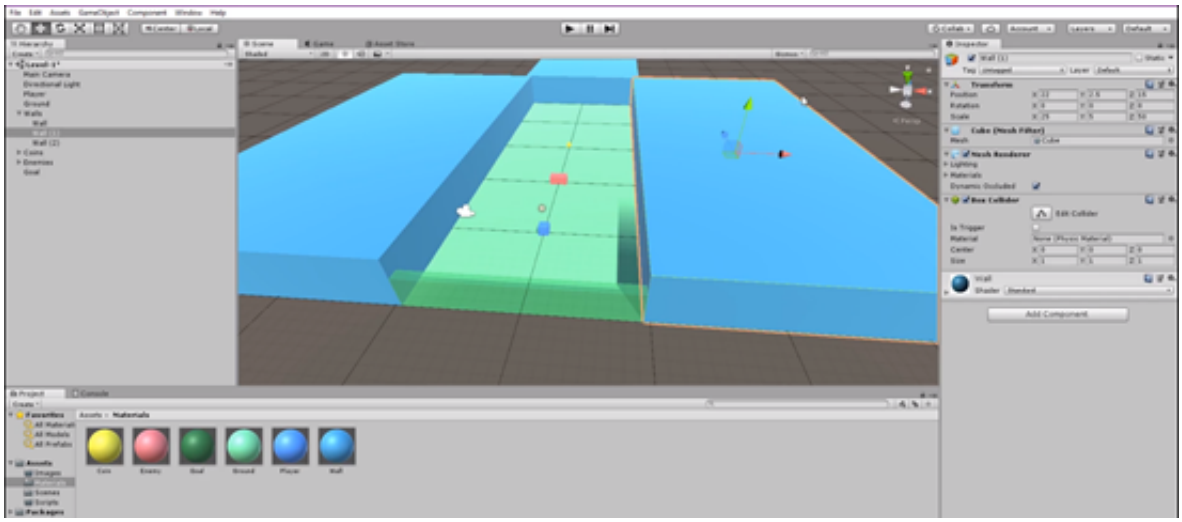
This way the Player shouldn't be able to accidentally jump over the goal.

This should be it for the level layout.

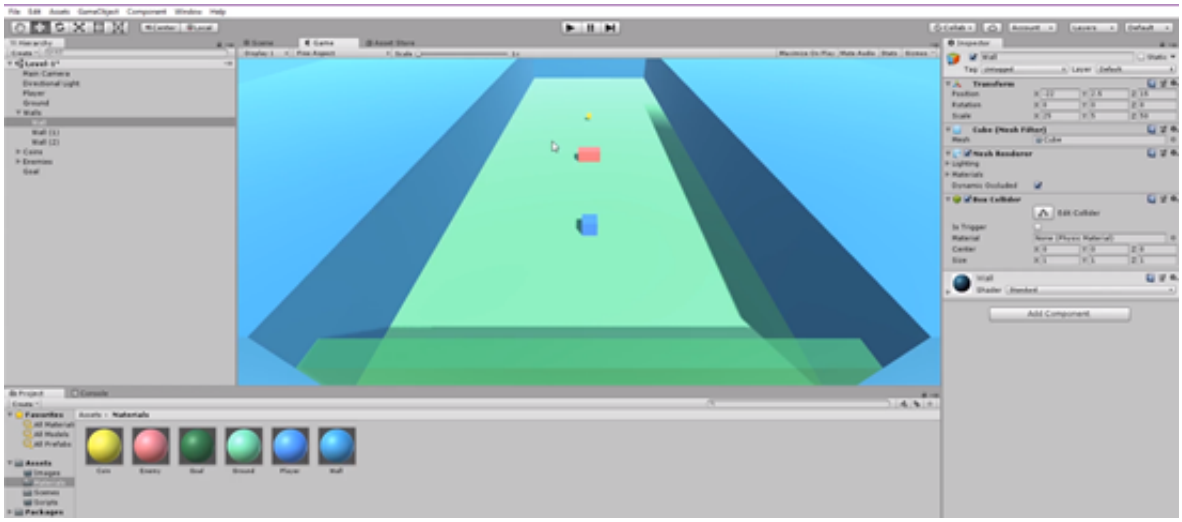
What we can do now is change up the **Skybox** a little to make it look better. So **select** the **Main Camera** in the Hierarchy and **change the Solid Color to match the sides of the walls.**



If you wanted to make the walls a little wider you can and use the vertex snapping tool to make sure they are placed correctly after adjusting the size.

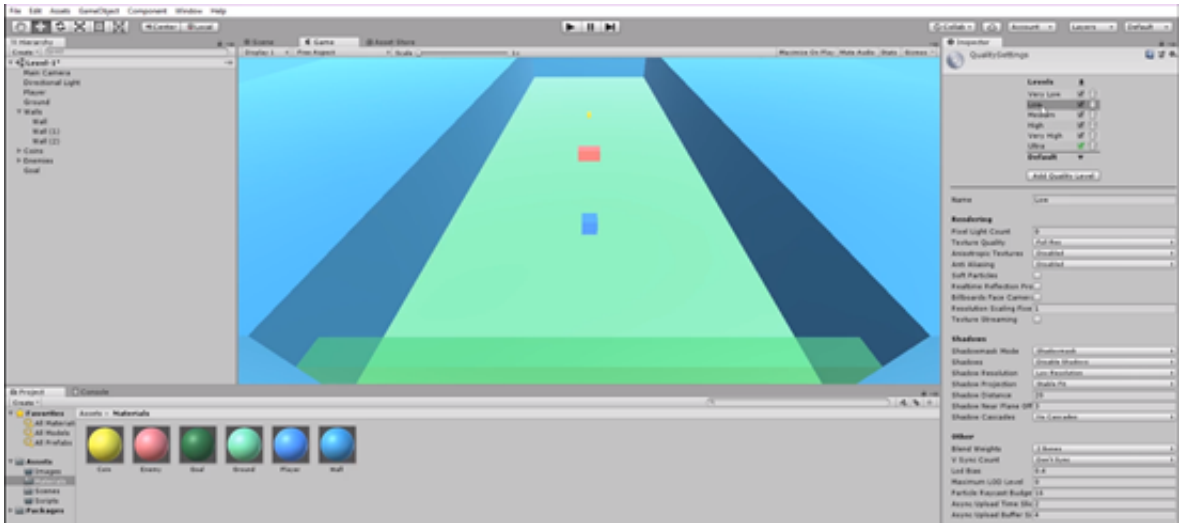


Now if you take a look at the level with the Game view you will see that we have these wide walls:

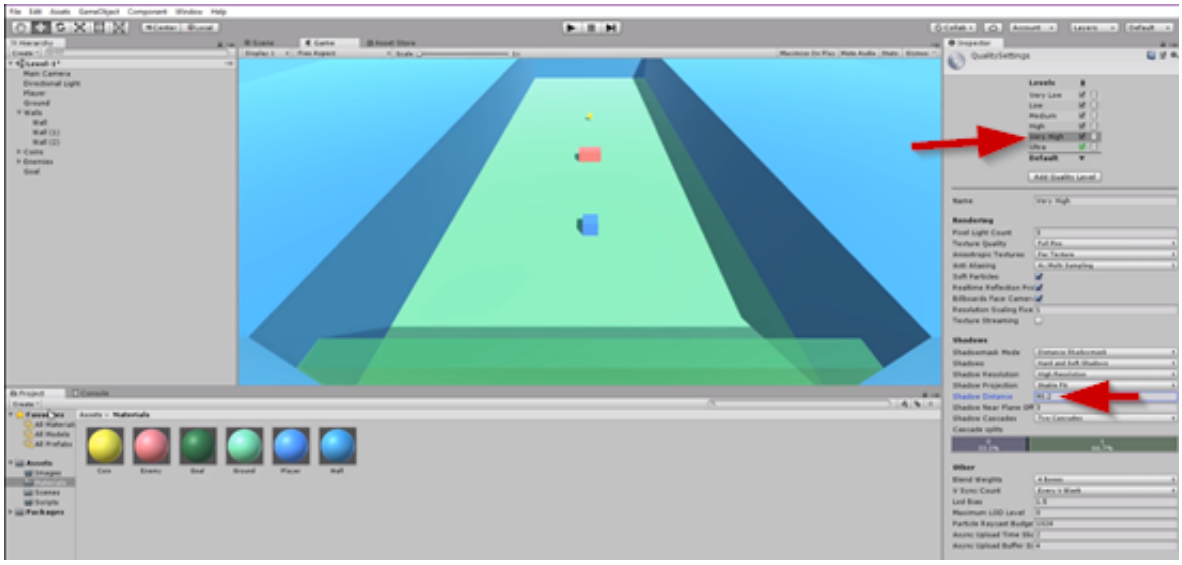


Taking Care of the Shadows

Select **Edit>Project Settings>Quality** and in this menu we can decide what settings have what. So if you selected the **Low quality setting** you would see all the settings listed below:

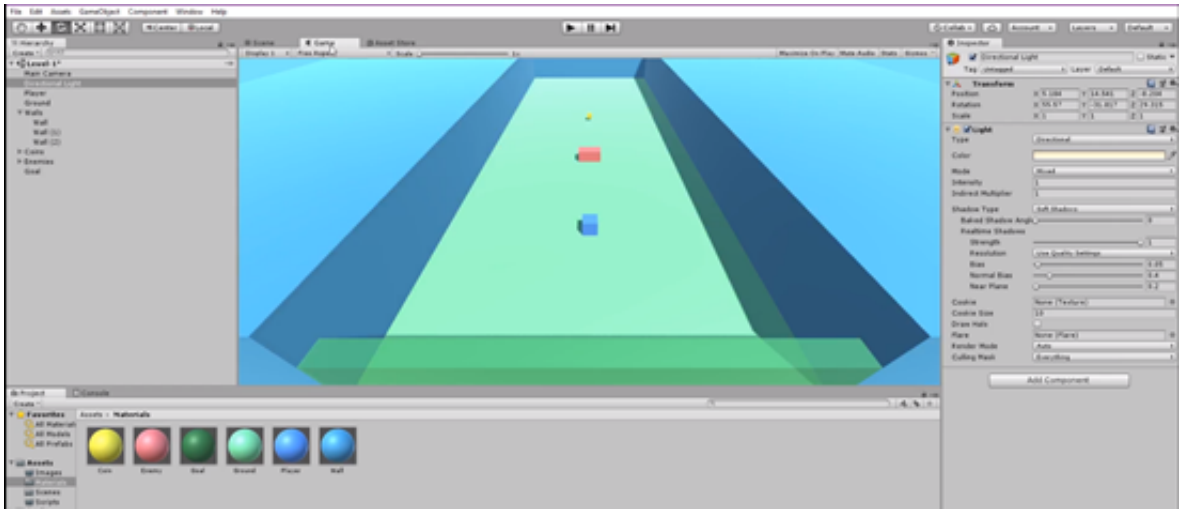


And we will probably be using the **Very High setting** for this project so then you can adjust the **Shadow Distance** so that we can see all the shadows in the scene:

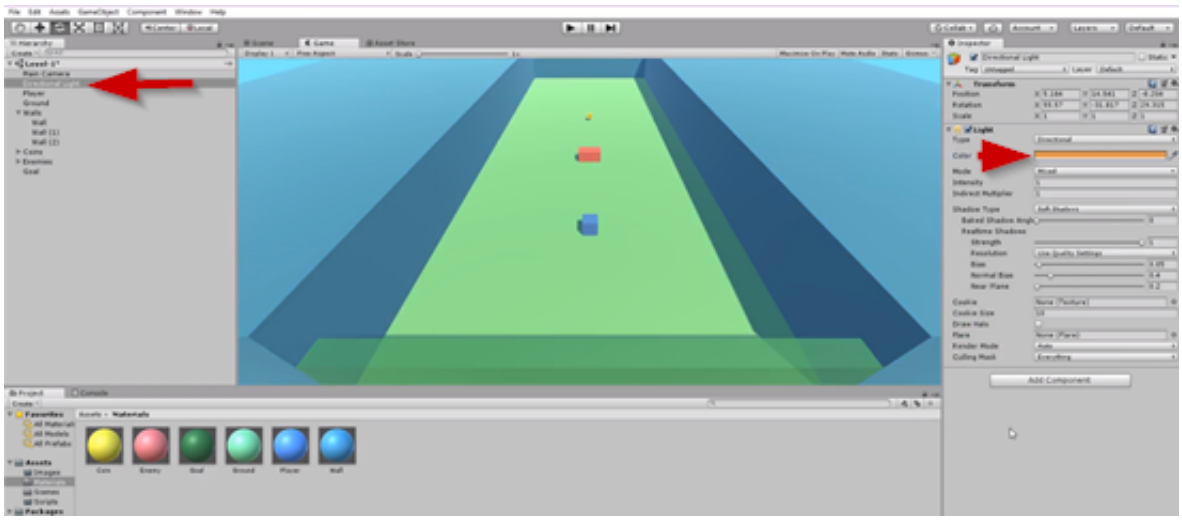


Set the **Shadow Distance to 100.**

The last thing we need to do is adjust the lighting, **select** the **Directional Light in the Hierarchy** and feel free to adjust the light however you would want it for your game.



You can also c**hange the color of the light using the color picker.**



In this lesson we will be getting our player moving around the level based on inputs.

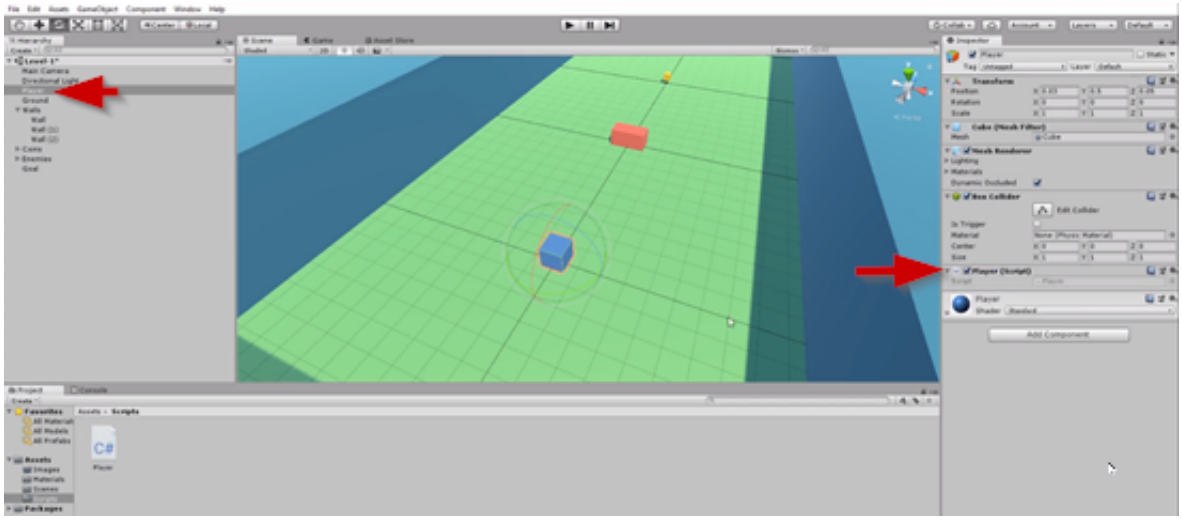
Player Movement

- Move the player object
- Calculate movement based on input
- Collide with objects using a rigidbody

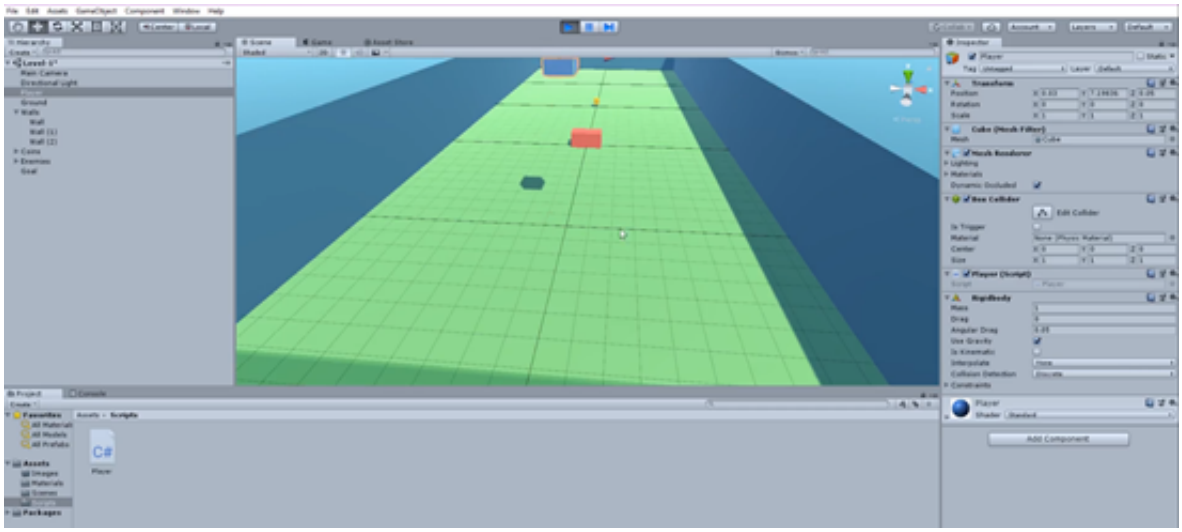
Navigate to the **Scripts folder** in the Project window and we will create our first C# script for this project.

Name the new script you create to **“Player”** this Player script will be handling everything our player does.

Attach the Player script to the Player object by dragging and dropping it onto the Player.



Select the **Player** game object in the **Hierarchy** and **attach a Rigidbody component** to it using the **Add Component** button in the **Inspector window.**



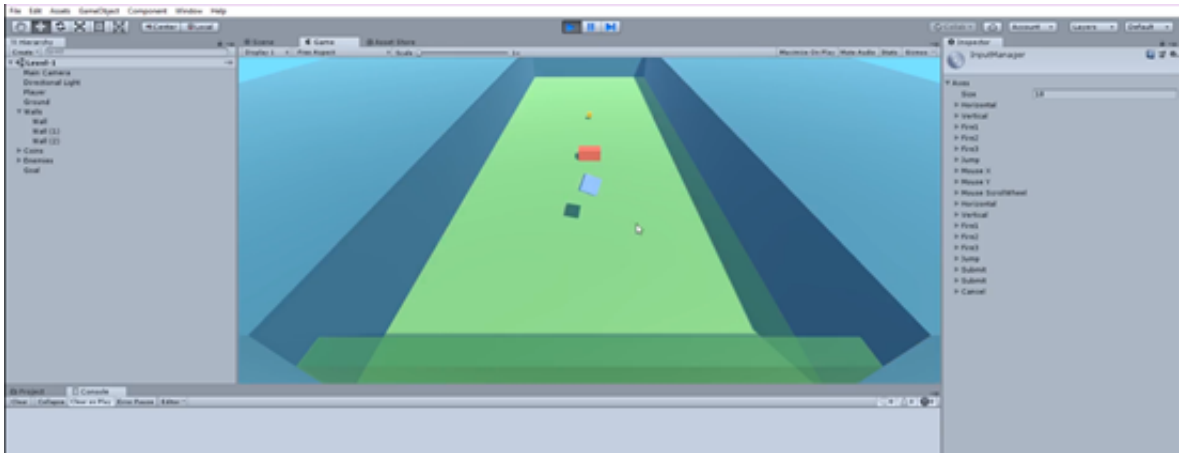
Open up the Player script in the code editor.

See the code below and follow along:

```
[SerializeField]
private Rigidbody playerBody;

private Vector3 inputVector;

// Use this for initialization
 void Start ()
    {
                playerBody = GetComponent<Rigidbody>();
}
// Update is called once per frame
 void Update ()
    {
        inputVector = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vert
ical"));
        playerBody.velocity = inputVector;
```

Save the script and navigate back to the Unity editor.

Hit the **Play button** and you should see that the Player is falling slowly to the ground.



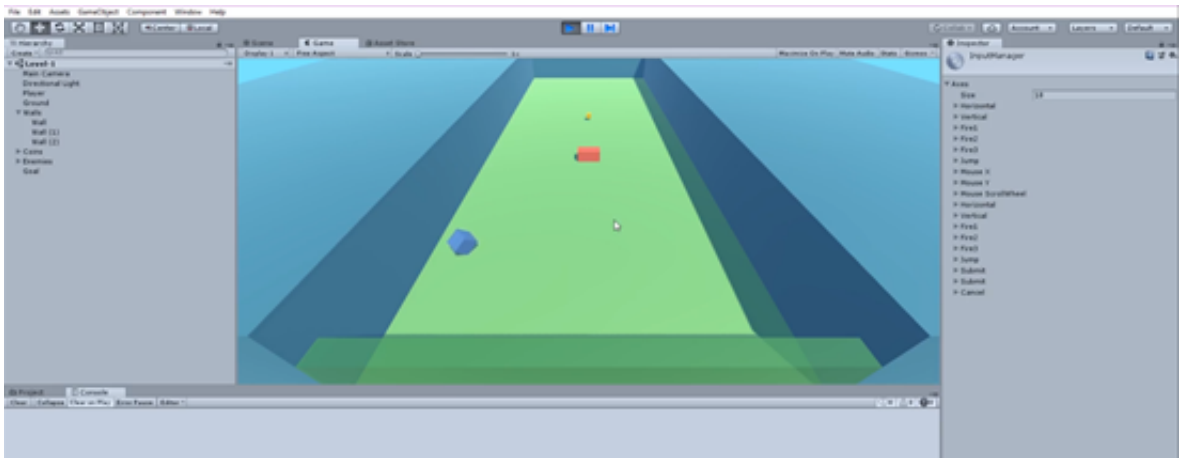
But, you are able to hold and use the arrows keys to move the Player in the direction you would like to move.

Now we can fix the falling we are having by adding some code to what we already have written.

Open the Player script in the code editor, see the code below and follow along:

```
// Update is called once per frame
 void Update ()
    {
        inputVector = new Vector3(Input.GetAxis("Horizontal"), playerBody.velocity.y,
 Input.GetAxis("Vertical"));
```

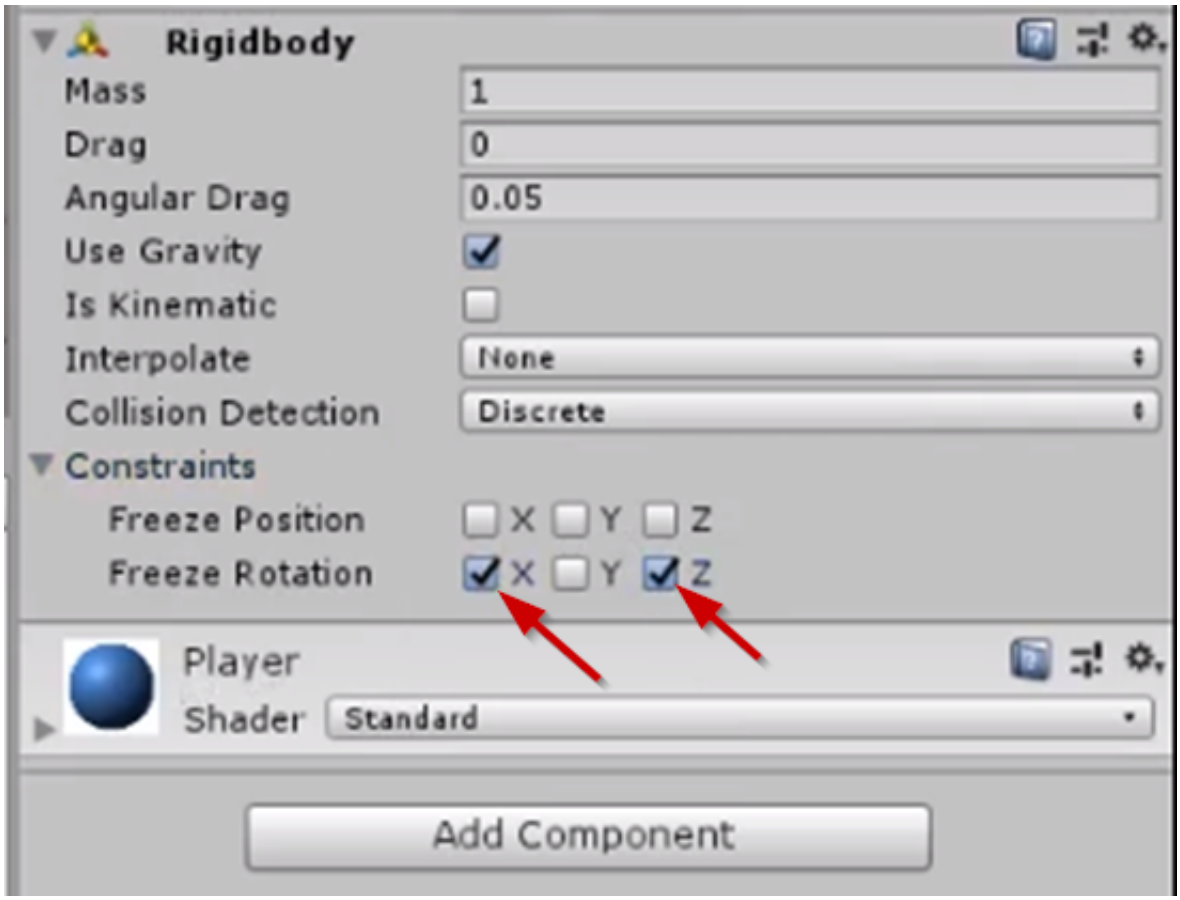
Now the falling is taken care of and the Player is stuck to the ground. You can still move the Player with the arrow keys and the Player will tumble around the level as you move.



We can fix this problem we are having with the tumbling, **select the Player game object in the Hierarchy** and go to **Constraints** which in under the **Rigidbody component.**

There will be a **Free Position** and **Freeze Rotation** options that have all **three axes.** By **toggling** any of the options are you can constrain the movement on the respective axis.

With this game we only want to be able to rotate on the Y axis, we don't want to be tumbling around anymore. **Toggle the X and Z options for Freeze Rotation.**



Hit the Play button and test out the movement for the Player, you can move left, right, up, and down. The Player is also no longer tumbling.

The Player is moving very slowly though, and this is an easy fix. What we can do is multiple the Input

vector by **10** to increase the speed of the player.

Open the Player script up in the code editor, see the code below and follow along:

```
// Update is called once per frame
 void Update ()
    {
        inputVector = new Vector3(Input.GetAxis("Horizontal") * 10f, playerBody.veloc
ity.y, Input.GetAxis("Vertical") * 10f);
```

Save the script and test the changes out by hitting the Play button.

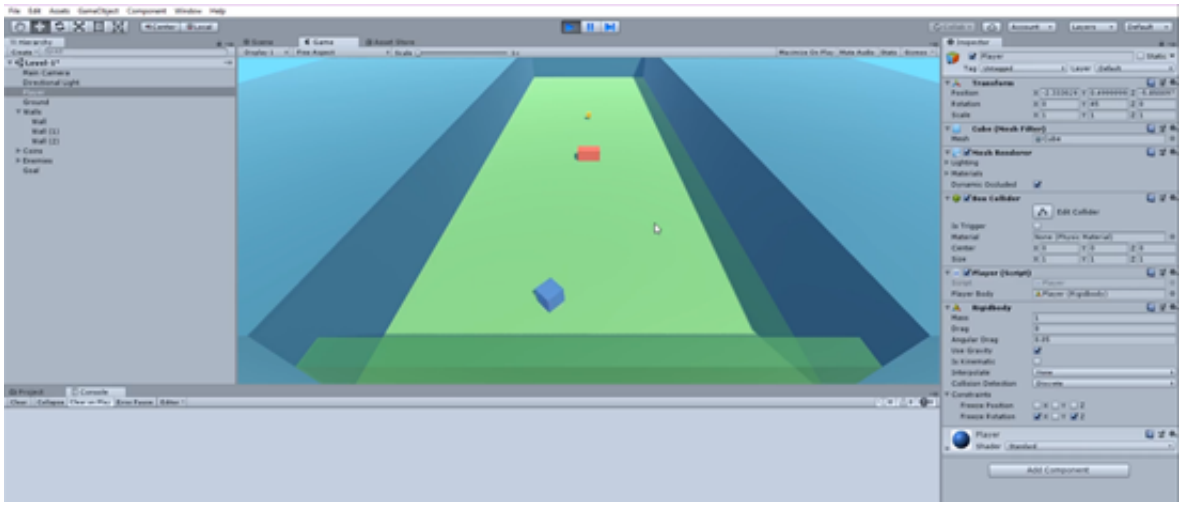
The Player is now moving much faster and in a snappy manner.

One thing you will notice though is that the Player is always facing the same way when moving. What we need to do is rotate the Player like you would expect if you were walking around in the world. In order to do this we can use a method called the look at from transform component.

See the code below and follow along:

```
// Update is called once per frame
 void Update ()
    {
        inputVector = new Vector3(Input.GetAxis("Horizontal") * 10f, playerBody.veloc
ity.y, Input.GetAxis("Vertical") * 10f);
        transform.LookAt(transform.position + new Vector3(inputVector.x, 0, inputVect
or.z));
        playerBody.velocity = inputVector;
    }
```

Save the script and test out the changes by hitting the Play button.



Open the Player script up in the code editor. We need to move some code into FixedUpdate.

See the code below and follow along:


```
private void FixedUpdate()
   {
       playerBody.velocity = inputVector;
   }
```

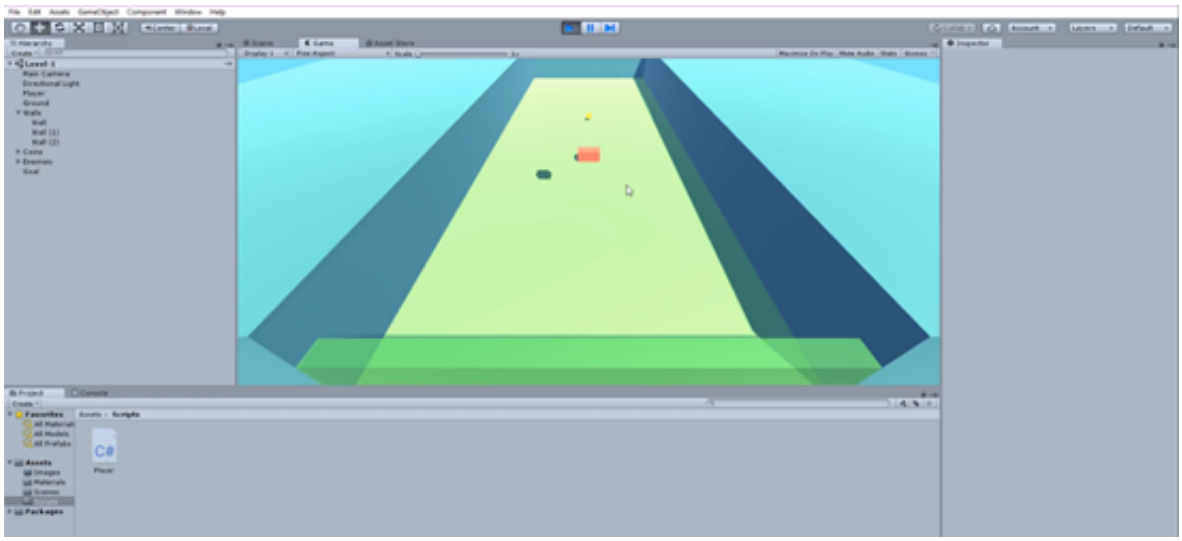
In this lesson, we are going to setup the Player jump.

- Check if the player is on the ground.
- Check if “Jump” is pressed.
- Apply instant Y force to player.

Open the **Player script**, see the code below and follow along:

```
[SerializeField]
private Rigidbody playerBody;

private bool jump;

private Vector3 inputVector;

// Use this for initialization
 void Start ()
    {
        playerBody = GetComponent<Rigidbody>();
    }

// Update is called once per frame
 void Update ()
    {
        inputVector = new Vector3(Input.GetAxis("Horizontal") * 10f, playerBody.veloc
ity.y, Input.GetAxis("Vertical") * 10f);
        transform.LookAt(transform.position + new Vector3(inputVector.x, 0, inputVect
or.z));
        if (Input.GetButtonDown("Jump"))
        {
            jump = true;
        }
    }

private void FixedUpdate()
    {
        playerBody.velocity = inputVector;
        if (jump)
        {
            playerBody.AddForce(Vector3.up * 20f, ForceMode.Impulse);
            jump = false;
        }
    }
```

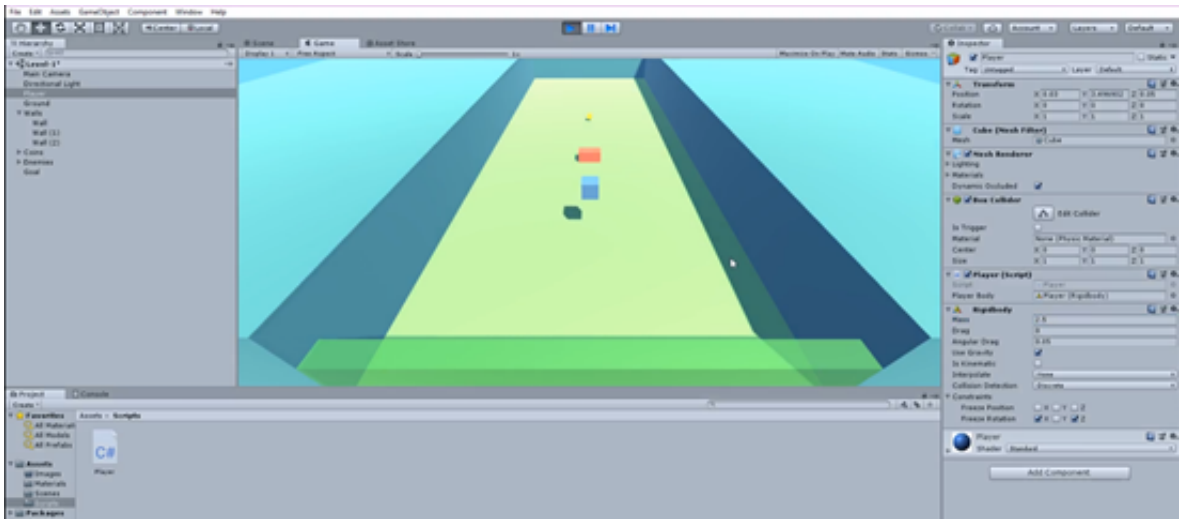
Save the script and navigate back to the Unity editor and **hit the Play button** to test out the changes.



The player is able to jump really high though.

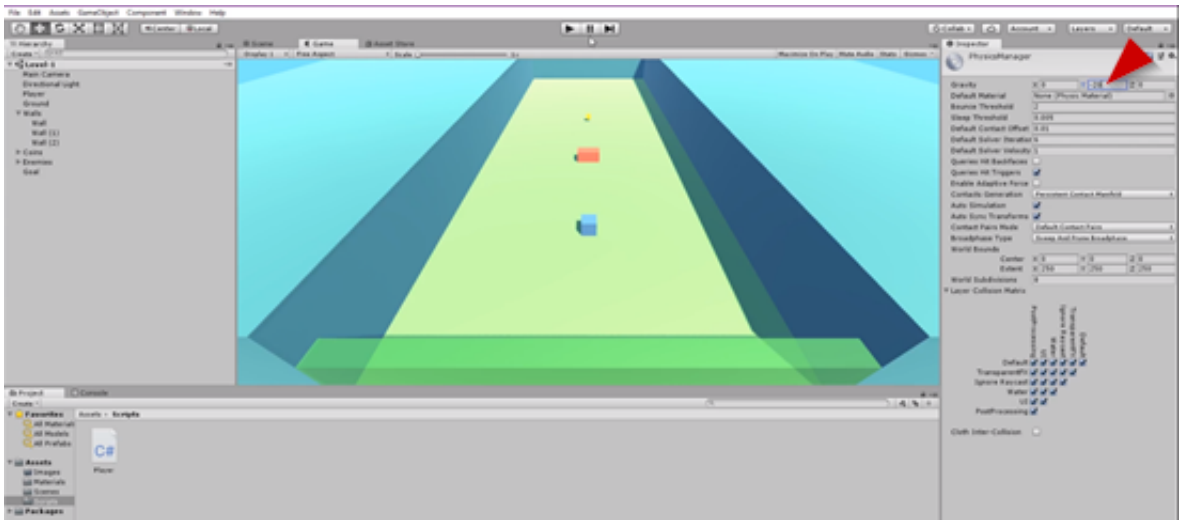
We will need tune how we want the jump to feel. We can adjust some settings on the **Rigidbody component** that is attached to the Player.

Select the Player and increase the Mass to 2.5.

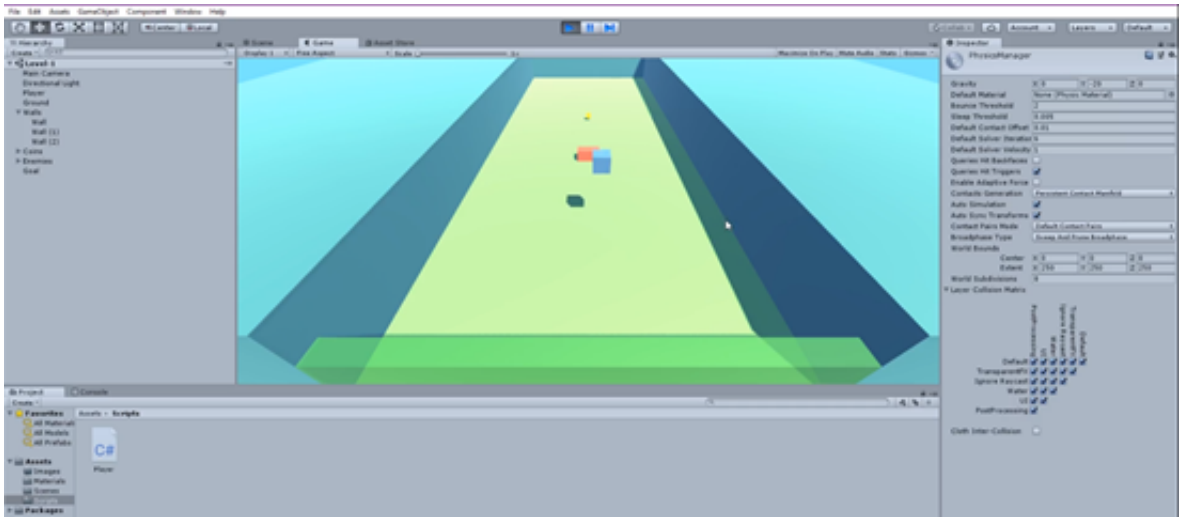


The problem now is that the Player can just keep jumping and jumping. **The Mass can be changed to 1.5** and now navigate to **Edit>Project Settings>Physics.**

We have a bunch of stuff now that we can see in the Inspector. What we need to do is **adjust the gravity property on the Y axis to be -20.**



Hit the **Play button** and test out the changes.



The Player now falls to the ground faster and it is a bit snappier.

Now we can come up with the fix for the endless jumping the Player can do.

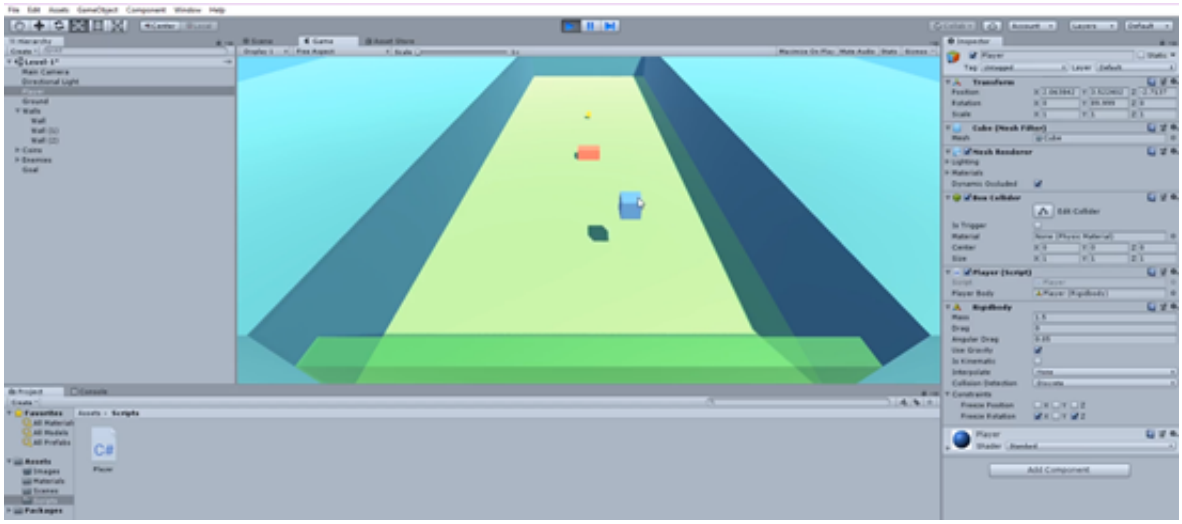
We need to know if the Player is actually on the ground. So we need a way to check for this and we can use a ray cast for this.

See the code below and follow along:

```
[SerializeField]
private Rigidbody playerBody;

private bool jump;

private Vector3 inputVector;

// Use this for initialization
 void Start ()
    {
        playerBody = GetComponent<Rigidbody>();
```

```
    }

// Update is called once per frame
 void Update ()
    {
        inputVector = new Vector3(Input.GetAxis("Horizontal") * 10f, playerBody.veloc
ity.y, Input.GetAxis("Vertical") * 10f);
        transform.LookAt(transform.position + new Vector3(inputVector.x, 0, inputVect
or.z));
        if (Input.GetButtonDown("Jump"))
        {
            jump = true;
        }
    }

private void FixedUpdate()
    {
        playerBody.velocity = inputVector;
        if (jump && IsGrounded())
        {
            playerBody.AddForce(Vector3.up * 20f, ForceMode.Impulse);
            jump = false;
        }
    }

bool IsGrounded()
    {
        float distance = GetComponent<Collider>().bounds.extents.y + 0.01f;
        Ray ray = new Ray(transform.position, Vector3.down);
        return Physics.Raycast(ray, distance);
    }
```

Save this script and navigate back to the Unity editor and **hit the Play button** to test the changes.



We can now just jump once when the space key is pressed. No more endless jumping.

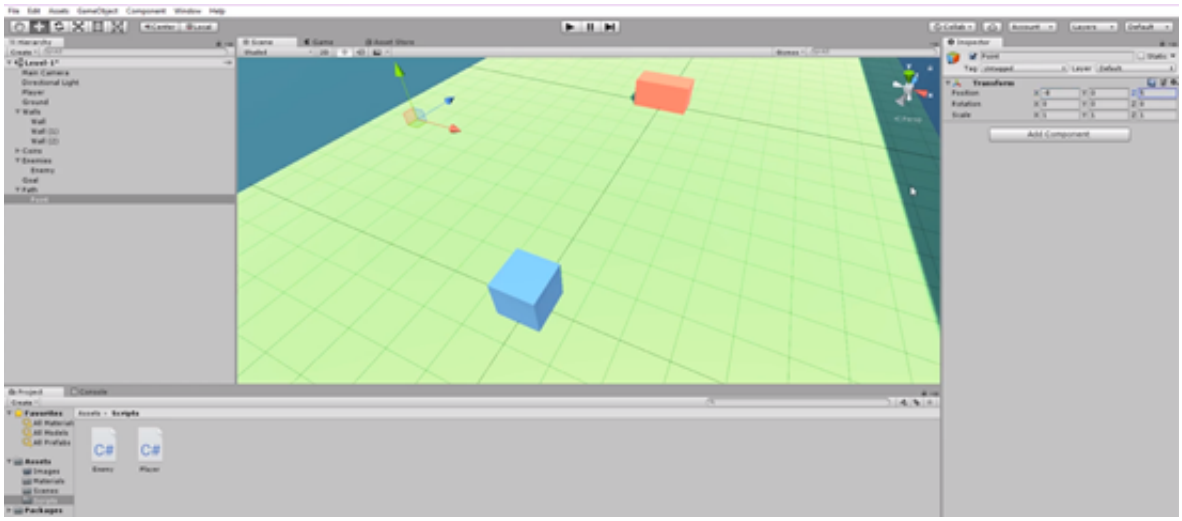
In this lesson we are going to setup the enemy movement system that will use a waypoint system.

- Create movement waypoints.
- Move from/to waypoints(a>b>c).

Create a new **C# script** and name it **“Enemy”** attach this script to the Enemy in the scene, by dragging and dropping it onto the Enemy.

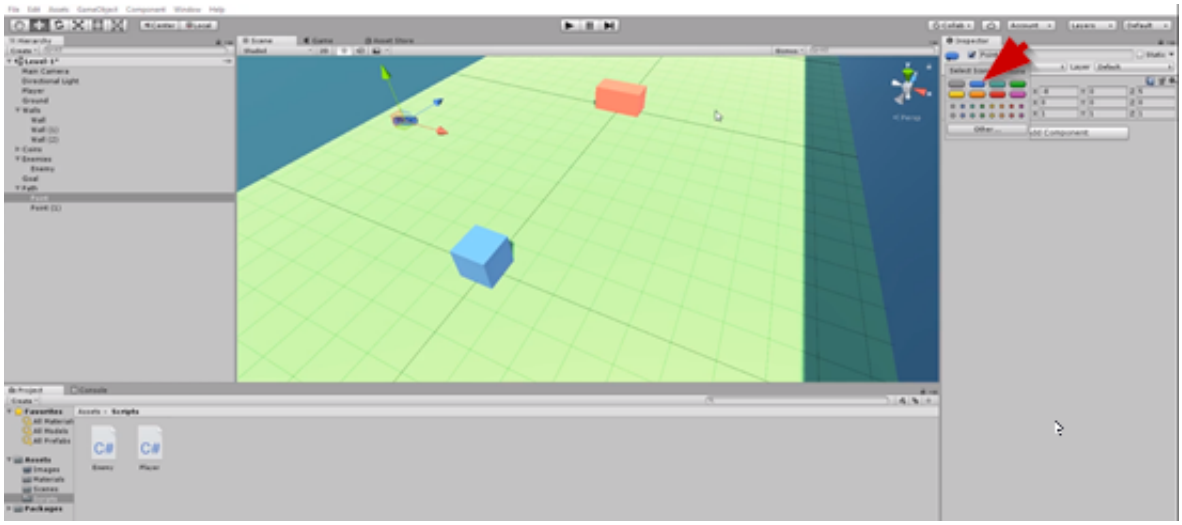
What we need to do now is layout some waypoints for our enemy to move between.

Create a new empty game object and rename it to **“Path”** and **position it at (0,0,0).** Add an empty game object to Path, this will make the Path a parent to this object. Call the child object **“Point” and position Point at X = -8, Y = 0, and Z = 5.**

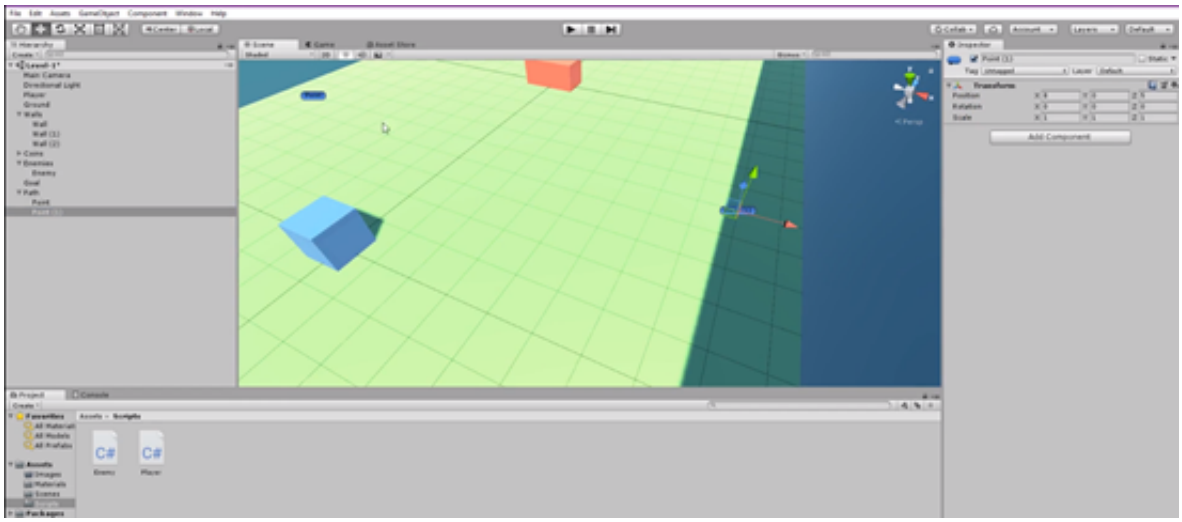


Duplicate Point and then **position Point(1) at X = 8, Y =0, and Z = 5.** We now have two positions and we cannot see the waypoint, so we can combat this by adding an icon on to the waypoints so we can easily locate them in the scene.

Select the Point game object in the Hierarchy and in the Inspector window you can see an **object icon. Select the object icon and choose the blue icon.**



Now we can visually see Point in the scene because of the blue icon. Do the same thing for Point(1).



And now we can click on the blue icons to select them in the scene if we need to.

We need to **create an array of waypoints** in order to get the enemy to move through them.

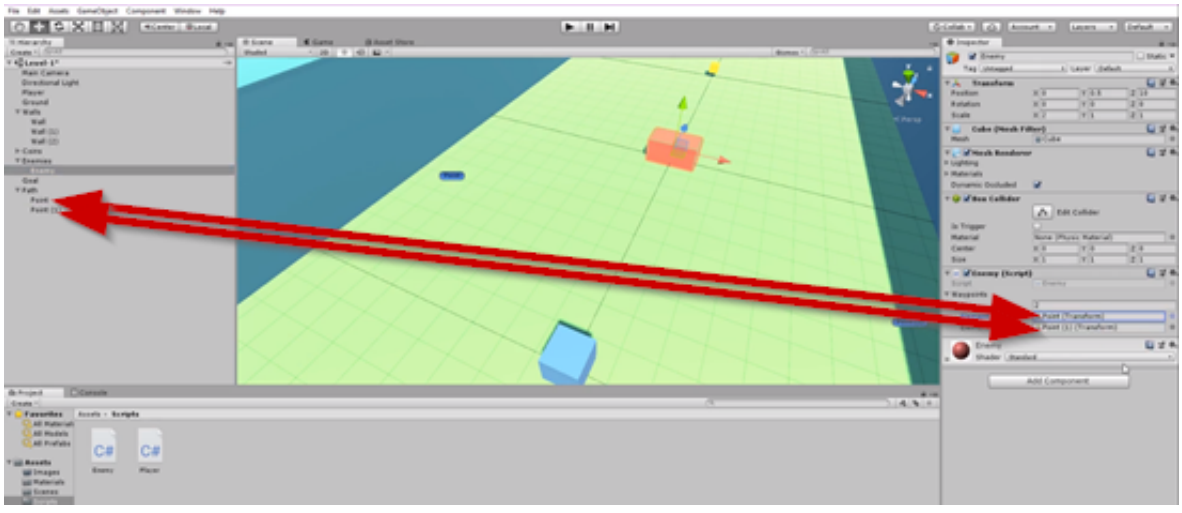
Open the Enemy script up in the code editor and see the code below and follow along:

```
[SerializeField]
private Transform[] waypoints;
```

Save the script and navigate back to the Unity editor.

Select the Enemy in the Hierarchy and look at the **Inspector window.** Find the Enemy(script) component and you should now see the Waypoints array there. If you **click on the drop down arrow you will see that it says Size 0**. What this means is that the array has no elements in it currently. We can say that this array will have 2 elements in it just by **typing 2 into the field there in the Inspector.** So now you will see there is a spot now for Element 0 and Element 1.

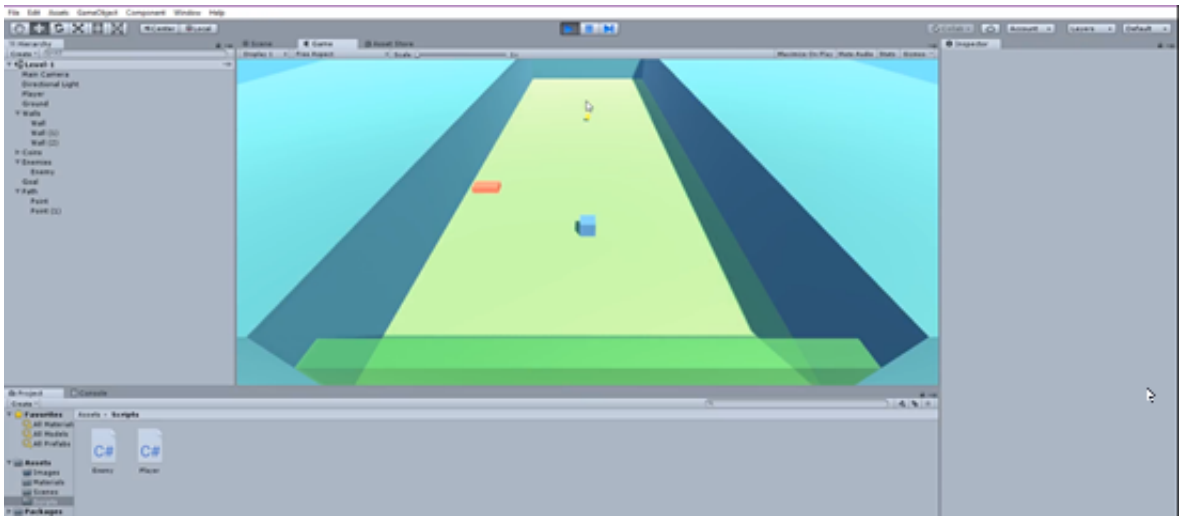
The first waypoint will be **Element 0 and the second waypoint will be Element 1.** Just drag and drop the Point object into the Element 0 field. Drag and drop Point(1) into the Element 1 field.



Open the Enemy script back up in the code editor and see the code below and follow along:

```
[SerializeField]
private Transform[] waypoints;
private Vector3 targetPosition;

// Use this for initialization
    void Start ()
    {
        targetPosition = waypoints[0].position;
 }

 // Update is called once per frame
 void Update ()
    {
        transform.position = Vector3.MoveTowards(transform.position, targetPosition,
.5f);
    }

}
```

Save the script and navigate back to the Unity editor and hit the Play button to see how the waypoints will work.



So we saw the enemy move to the waypoints, but the enemy is also in the ground when doing that. We can fix this by putting the **Path object position for the Y value at 0.5.**

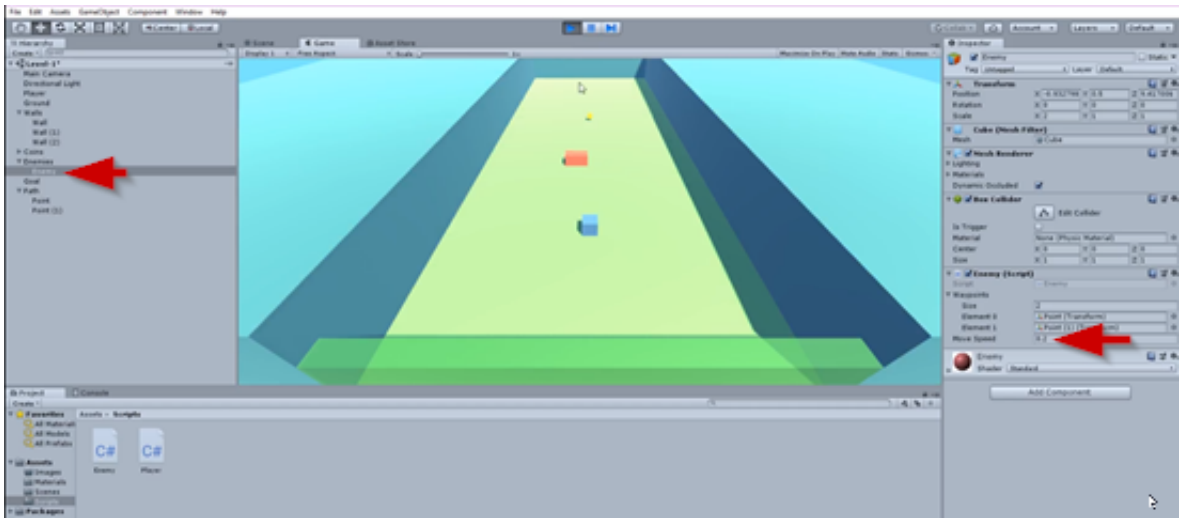
The movement for the enemy is also too fast. We can fix this via code. Go ahead and **open the Enemy script** back up in the code editor.

See the code below and follow along:

```
[SerializeField]
private Transform[] waypoints;
private Vector3 targetPosition;
[SerializeField]
private float moveSpeed;

// Use this for initialization
```

```
    void Start ()
    {
        targetPosition = waypoints[0].position;
 }

 // Update is called once per frame
 void Update ()
    {
        transform.position = Vector3.MoveTowards(transform.position, targetPosition,
.5f * moveSpeed);
    }

}
```

Save the script and navigate back to the Unity editor to **assign 0.2 to the Move Speed value in the Inspector for the Enemy.**



We can **add an attribute** to adjust the move speed with a slider. Open the enemy script back up in the code editor.

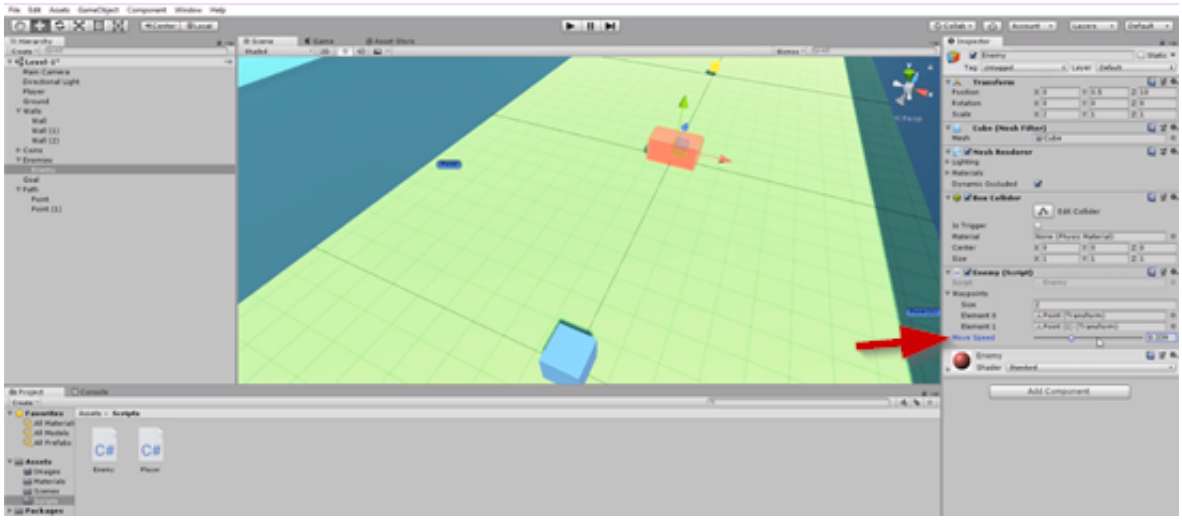
See the code below and follow along:

```
[SerializeField]
private Transform[] waypoints;
private Vector3 targetPosition;
[SerializeField]
[Range(0,1f)]
private float moveSpeed;

// Use this for initialization
    void Start ()
    {
        targetPosition = waypoints[0].position;
 }

 // Update is called once per frame
```



```
 void Update ()
    {
        transform.position = Vector3.MoveTowards(transform.position, targetPosition,
.5f * moveSpeed);
```

Save the script and navigate back to the Unity editor. You can now adjust the speed of the Enemy using the slider in the Inspector.



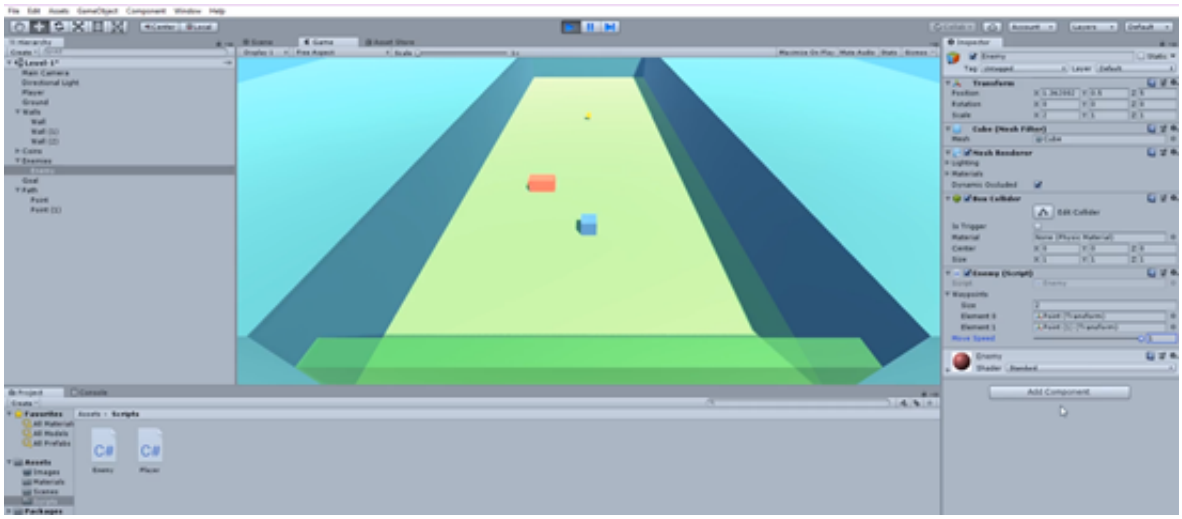
What we want to do now is have the Enemy go from the first waypoint to the second waypoint and we want the enemy to keep doing that over and over.

Open the enemy script back up in the code editor and see the code below and follow along:

```
[SerializeField]
    private Transform[] waypoints;
    private Vector3 targetPosition;
    [SerializeField]
    [Range(0,1f)]
    private float moveSpeed;
    private int waypointIndex;
    // Use this for initialization
    void Start ()
    {
        targetPosition = waypoints[0].position;
 }

 // Update is called once per frame
 void Update ()
    {
        transform.position = Vector3.MoveTowards(transform.position, targetPosition,
.5f * moveSpeed);
        if (Vector3.Distance(transform.position, targetPosition) < .25f)
        {
            if (waypointIndex >= waypoints.Length-1)
            {
                waypointIndex = 0;
            }
```

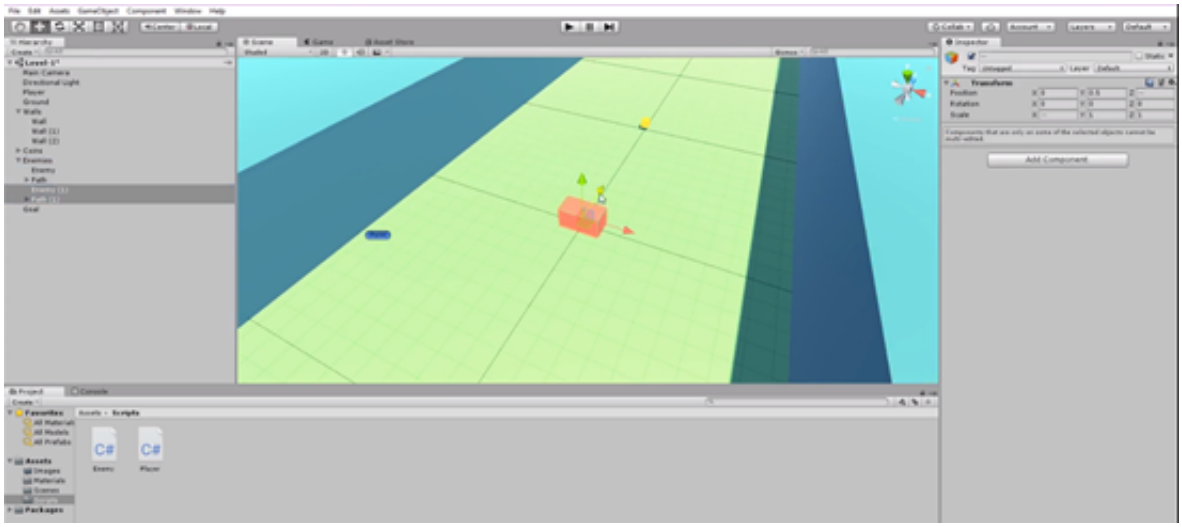
```
            else
            {
                waypointIndex++;
            }
            targetPosition = waypoints[waypointIndex].position;
        }
    }
}
```

Save the script and navigate back to the Unity editor to test the changes by hitting the **Play button.**



Now we want to take Path and put that game object inside Enemies in the Inspector.

Then **select both Enemy and Path and duplicate them by hitting Ctrl + D.**

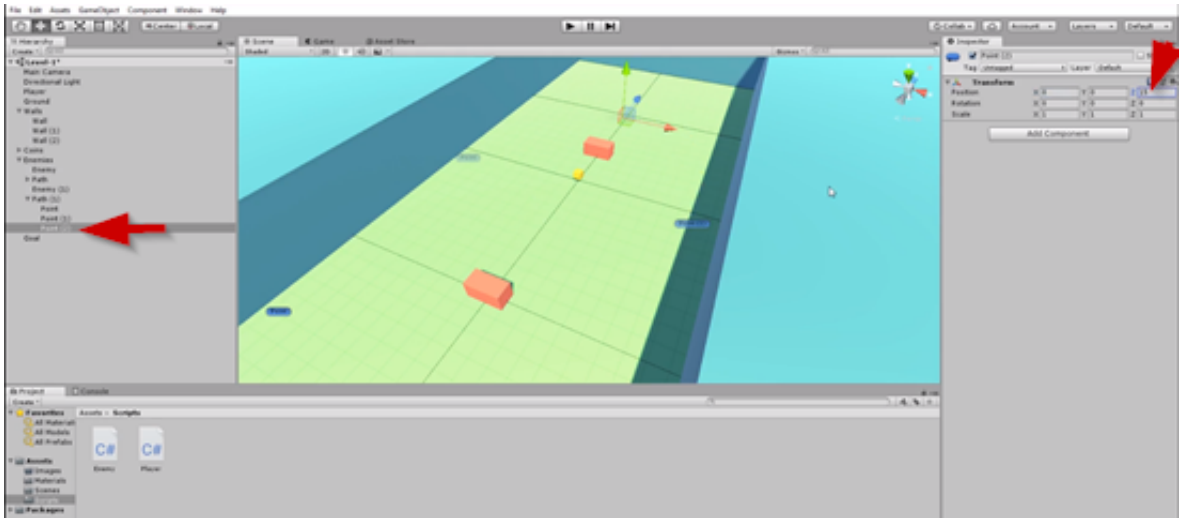


Now what this does is allow us to drag the second enemy along with the waypoints that belong to it somewhere else in the scene.

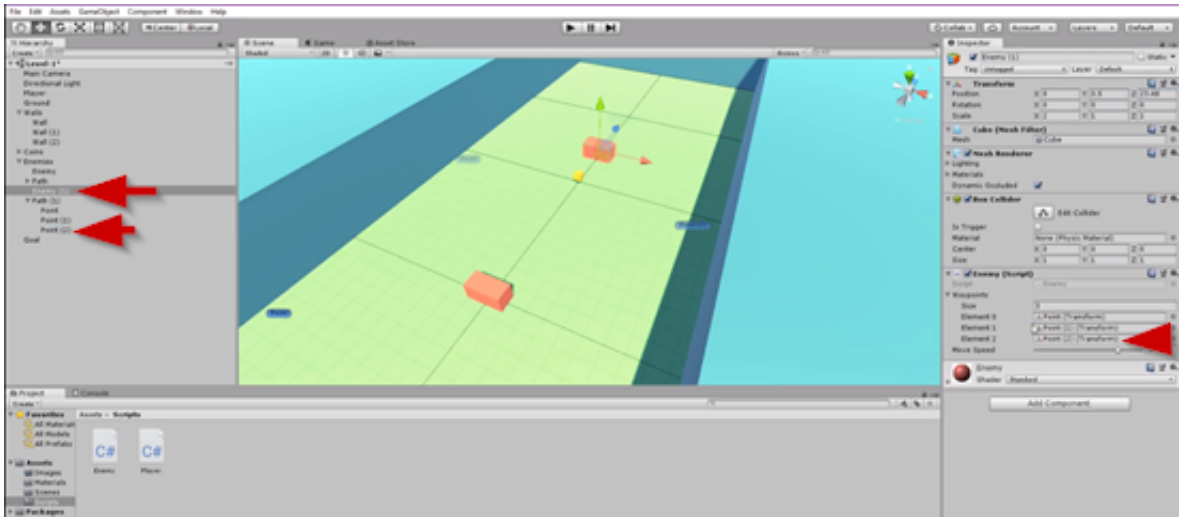
I am putting **Enemy(1) at position (0,0.5, 23.48).** Now we need to setup the waypoints for this

enemy because all we have currently is the duplicated path from the first enemy.

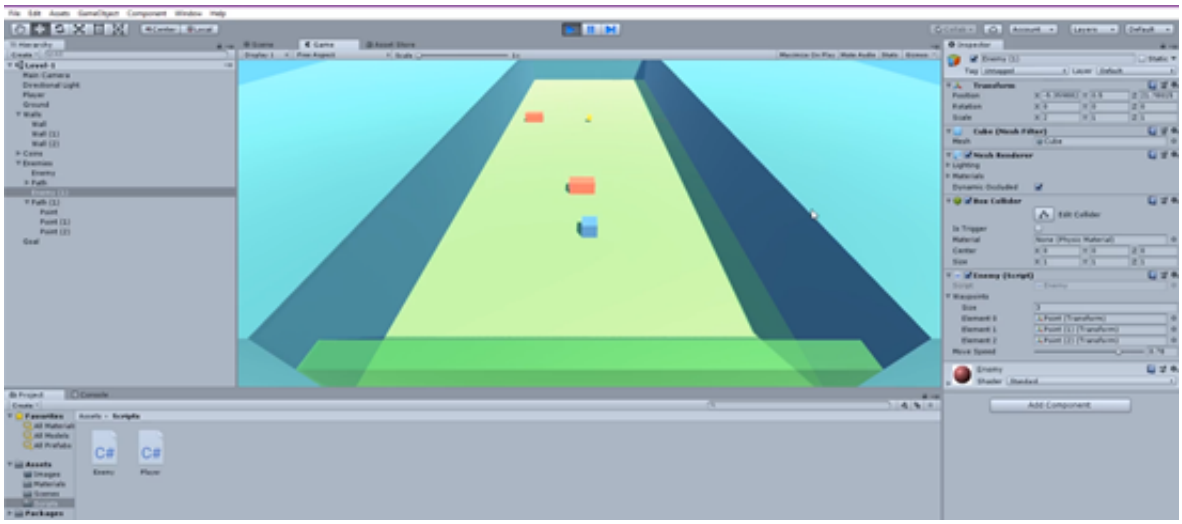
But, I am actually going to **add another point to the Enemy(1) path.** This will give this enemy 3 points. **Point(2) will be at position (0,0,15).**



Point(2) needs to be added to the waypoint array in the Inspector.



Hit Play and you will see both enemies moving along their waypoint paths.



Notice that we did not have to touch any code to create the second enemy and its waypoints path. That is what is so good with designing systems that work like this in your projects you create.

In this lesson, we will write the code for the game manager.

- Handles loading levels.
- Handles exiting the game.

Create a new C# script and name it “Game” and we need to add this script to an object in our scene. So **create an empty game object and rename it to “Game”** attach the Game script to the Game object.

Open the Game script up in the code editor, see the code below and follow along:

```
[SerializeField]
   private int level;
   [SerializeField]
   private bool lastLevel;
   private int nextLevel;


// Use this for initialization
void Start ()
   {
       nextLevel = level + 1;
}
```

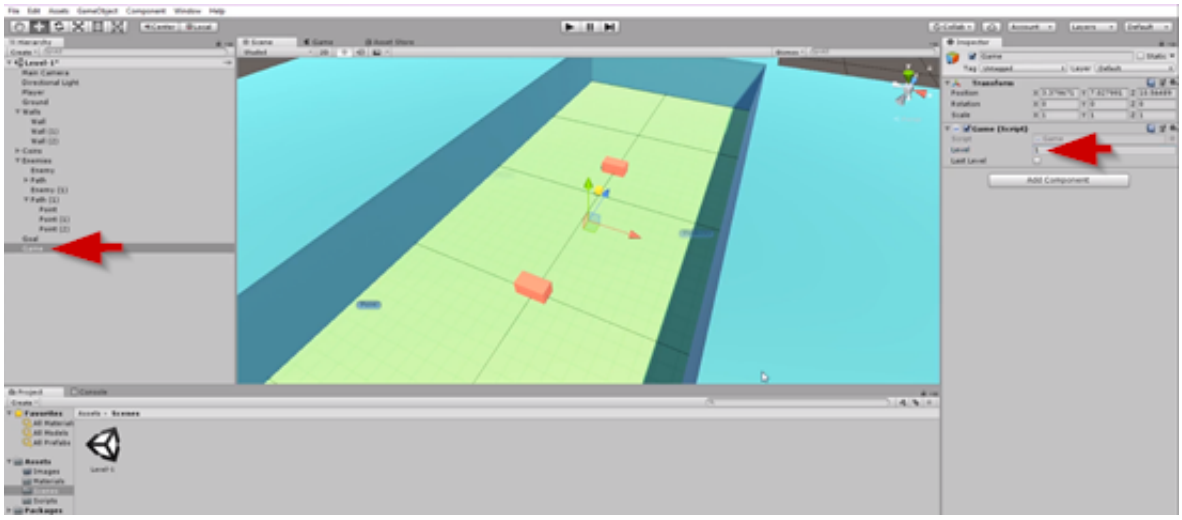
Navigate back to the Unity editor and take a look at the Scenes folder. We have one scene currently in that folder and its called **“Level-1”** we are going to use the number in the name of the level here to handle loading levels.

```
using UnityEngine.SceneManagement;

public class Game : MonoBehaviour {
    [SerializeField]
    private int level;
    [SerializeField]
    private bool lastLevel;
    private int nextLevel;


 // Use this for initialization
 void Start ()
    {
        nextLevel = level + 1;
 }

    public void LoadLevel(string levelName)
    {
        SceneManager.LoadScene(levelName);
    }

    public void LoadNextLevel()
    {
        if (!lastLevel)
        {
```

```
            string sceneName = "Level-" + nextLevel;
            LoadLevel(sceneName);
        }
        else
        {
            // go to main menu
            LoadLevel("Main-Menu");
        }
    }

    public void ReloadCurrentLevel()
    {
        LoadLevel("Level-" + level);
    }

    public void Quit()
    {
        Application.Quit();
    }
}
```

Save the script and navigate back to the Unity editor.

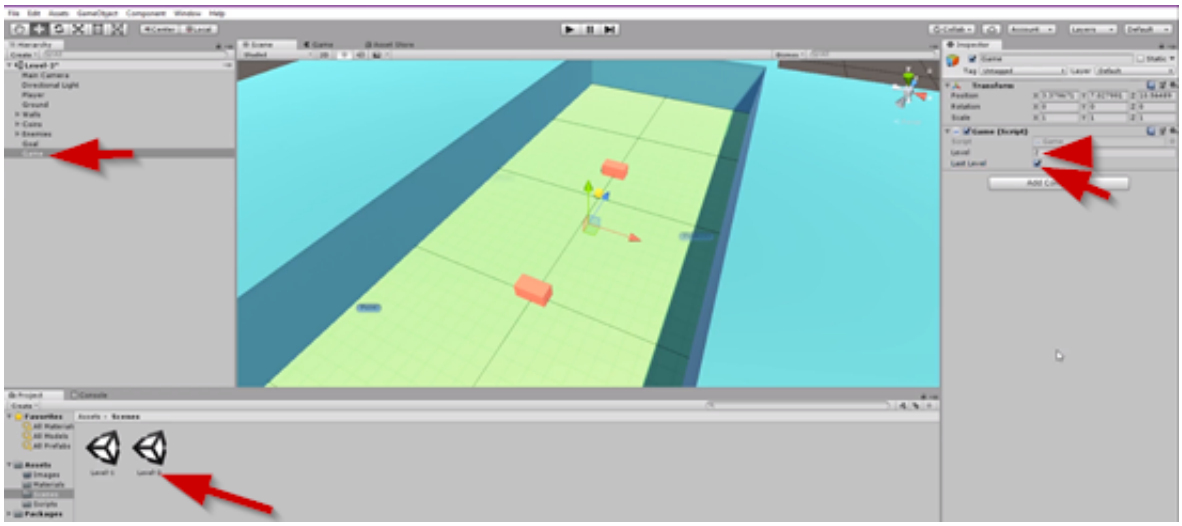
Select the Game object in the Hierarchy and look at the Inspector window. In the **Level field change it from 0 to 1** and it is not the last level so do not check the toggle option.



Save the scene and project.

Duplicate Level-1 and you will then have a Level-2 in the Scenes folder.

Now **open Level-2** up and **select** the **Game object in the Hierarchy.** Change the **Level value from 1 to 2.**



You can setup this level however you would like.

In this lesson we will be checking for collisions between the enemies and player.

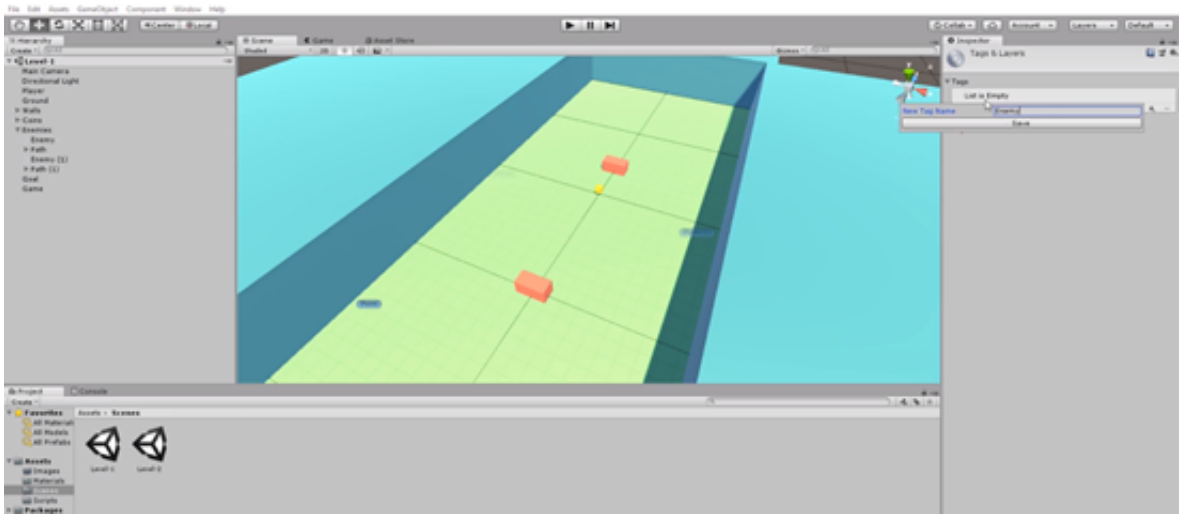
Colliding With Enemies

- Check to see if player collided with an enemy.
- Use object tags.
- Reload current level.

We will be adding the code for **collision detection** between the Player and Enemy to the **Player script**, but first we need to setup the tags.

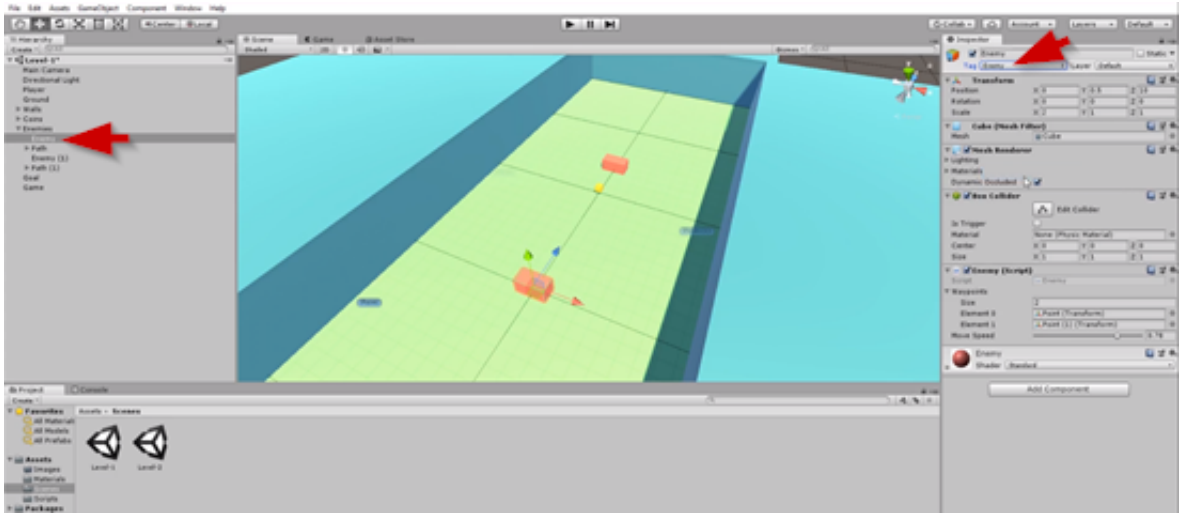
Select Enemy from the Hierarchy and look at the Inspector window, we have a **tag drop down box and select Add Tag**... from the menu. This will bring you to the **Tags and Layers menu**.

Click on the Plus icon to add a new tag and name the new tag **“Enemy.”**



Click the **Save button.**

Make sure to **tag the Enemy** in the scene with new tag you created by selecting it from the drop down menu.



Now **tag the Enemy(1) with the same tag.**

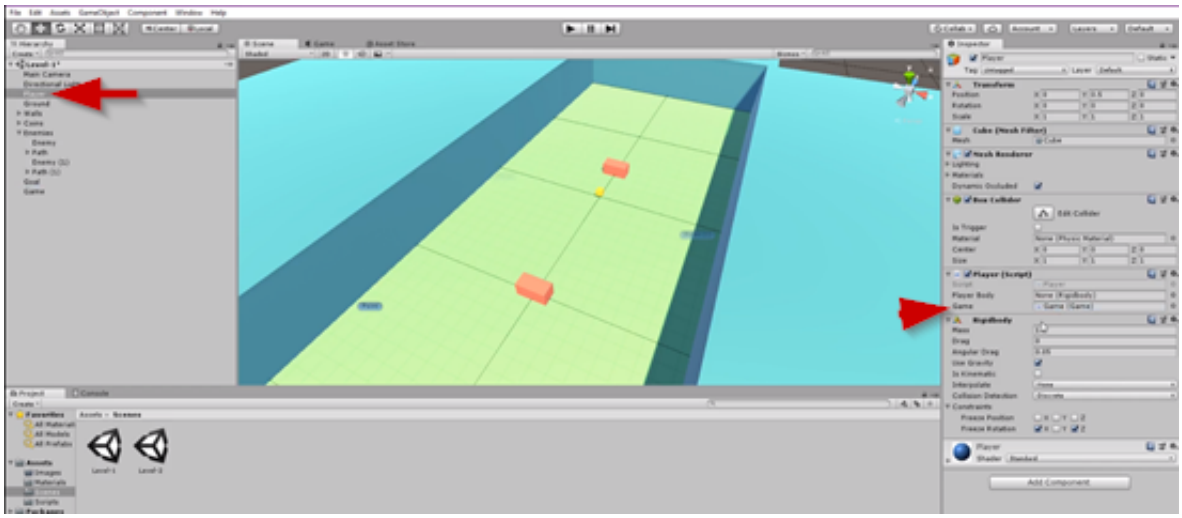
We now need to add the collision code using the tag we just created.

Open the Player script and see the code below and follow along:

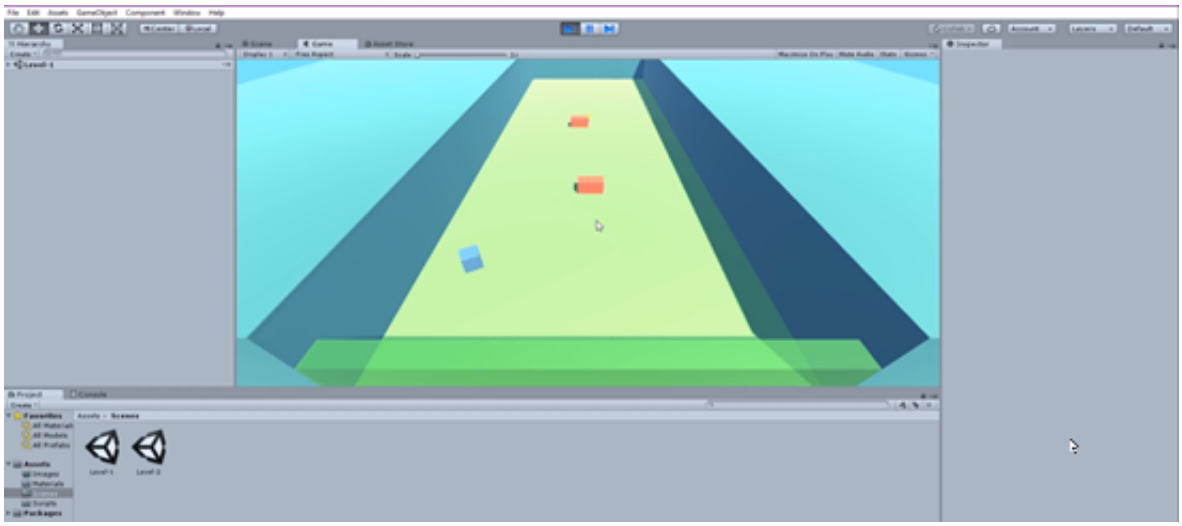
```
[SerializeField]
private Game game;

private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Enemy"))
        {
            game.ReloadCurrentLevel();
        }
    }
```

Save the script and navigate back to the Unity editor.

Assign the reference for the Game field in the **Inspector on the Player script component** by dragging and dropping Game onto the open field in the Inspector.



Test the changes out by **hitting the Play button.**



If the player collides with an Enemy the level is restarted.

Open the Player script back up in the code editor. See the code below and follow along:

```
private Game game;

// Use this for initialization
 void Start ()
    {
        game = FindObjectOfType<Game>();
        playerBody = GetComponent<Rigidbody>();
 }
```

We added this to the code so that we don’t have to assign the reference in the Inspector.

In this lesson we will be setting up the coin collision so that the player can start collecting coins.

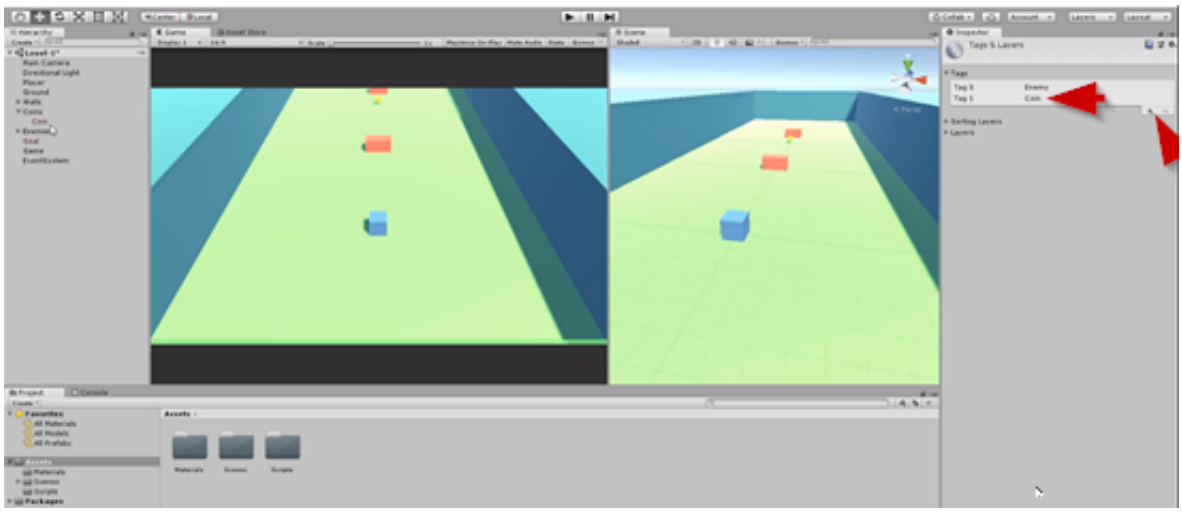
- Check for collisions with coins.
- Increment a counter value.
- Display coin count in the UI.
- Goal collision check coin count.
- Load next level if enough.

Adding Some More Tags

Once you have the project open we need to setup some more tags for the coin collection.

Select the coin game object from the **Hierarchy** and in the **Inspector** click the Tag drop down menu and then **select** the **Add Tag**... option.

Create the new tag and call it **“Coin.”**



Now **select** the **Goal game object** from the **Hierarchy** and create a **new tag** for this called **“Goal.”**

We will be checking for collision using the tags, and we will do this in the Player script.

Open the Player script up in the code editor and we need to add some code to it. See the code below and follow along:

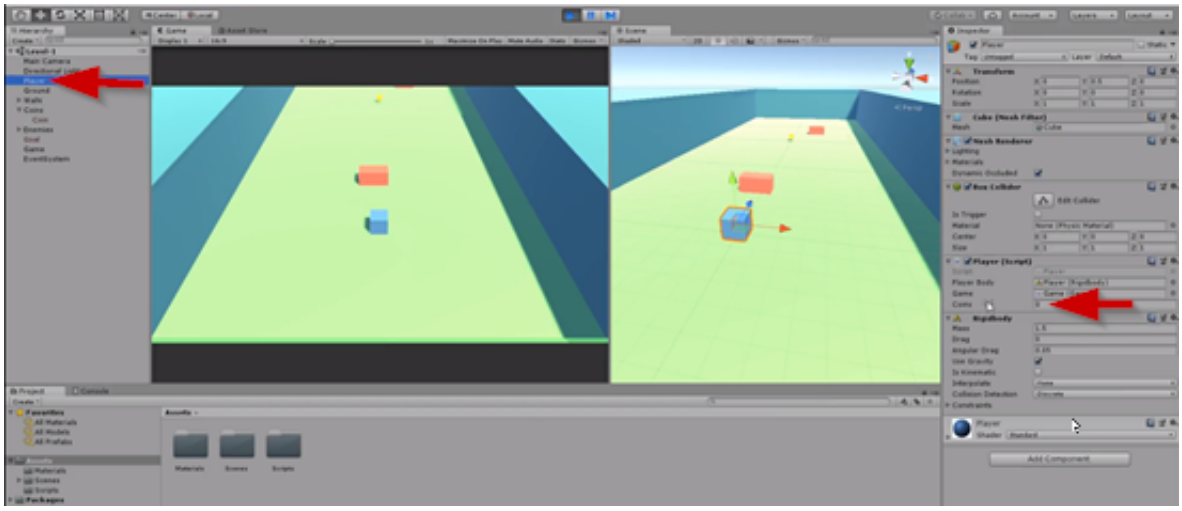
```
[SerializeField]
private int coins;

 private void OnTriggerEnter(Collider other)
    {
        switch (other.tag)
        {
            case "Coin":
                coins++;
                //update the UI
                break;
            case "Goal":
                // check for completion
                break;
```

```
            default:
                break;
        }
    }
```

Save this script and navigate back to the Unity editor. We can check to see if this is working because we will be able to see the value in the Inspector.

Hit the **Play** button.

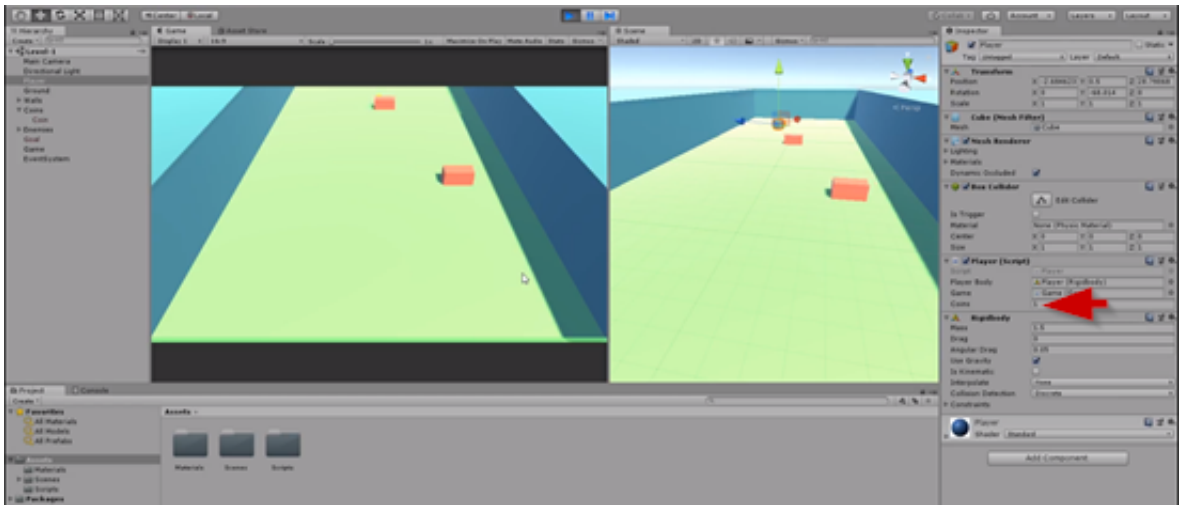
Select the **Player in the Hierarchy** and look at the **Inspector** where the Coins value field is located and it should say **0** right now because we haven't collected any coins yet.



Now if you try to collect the Coin you will see that nothing happens. This is because **the coin wasn't tagged with the "Coin" tag.** So go head and make sure this is done.

Go ahead and make sure the **Goal is tagged as "Goal"** as well.

Hit the **Play** button again and test out the changes.



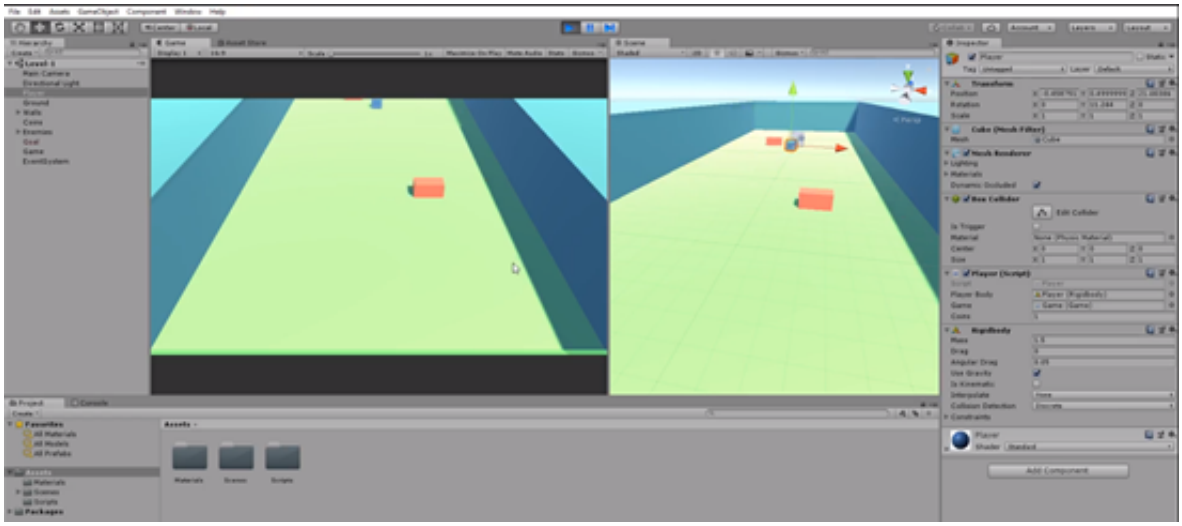
You will see that when the player collides with the Coin now that the Coins value increases by one.

You will notice that the coin is still there after we collide with it, so we need to setup a destroy command in the code so that once we collide with the coin it will disappear from the scene.

Open the Player script back up, and see the code below and follow along.

```
private void OnTriggerEnter(Collider other)
    {
        switch (other.tag)
        {
            case "Coin":
                coins++;
                //update the UI
                 Destroy(other.gameObject);
                break;
            case "Goal":
                // check for completion
                break;
            default:
                break;
        }
    }
```

Save this script and navigate back to the Unity editor. Test out the changes after you hit the Play button.



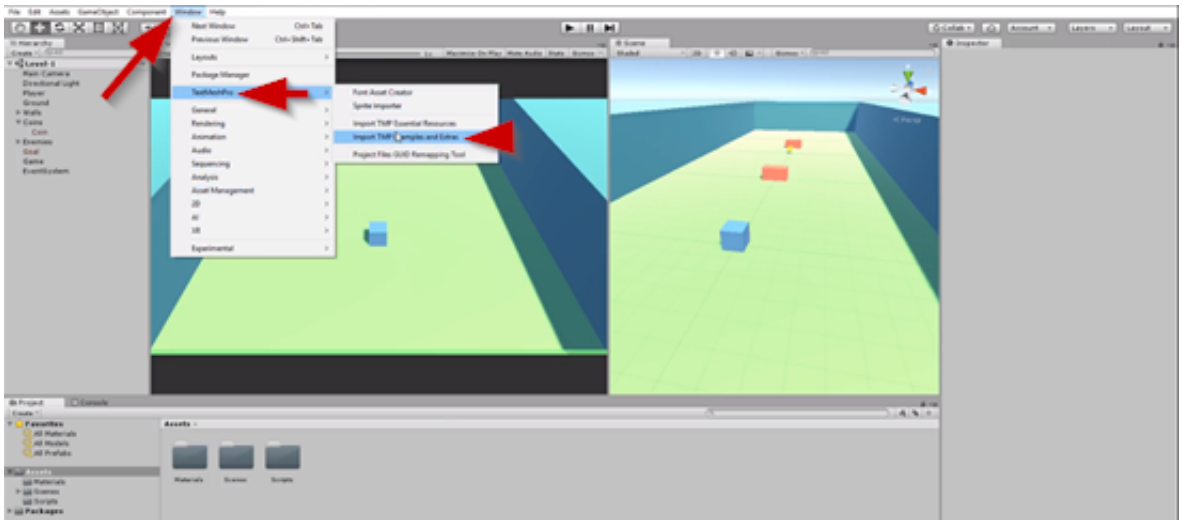
Now you will see that the coin has been destroyed once its been collected by the player.

Setting Up The U.I.

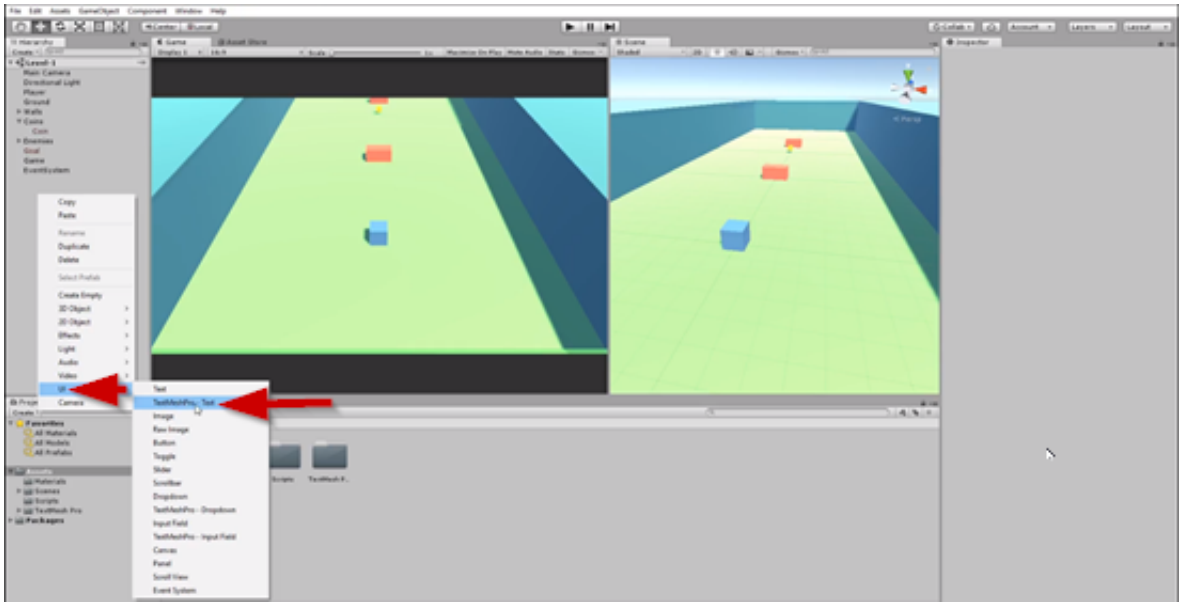
We can setup the text element that will display the amount of coins we have collected in the top right corner of the game.

In order to do this we need to import a tool called **Text Mesh Pro.**

Select **Window>TextMeshPro>Import TMP Essential Resources**

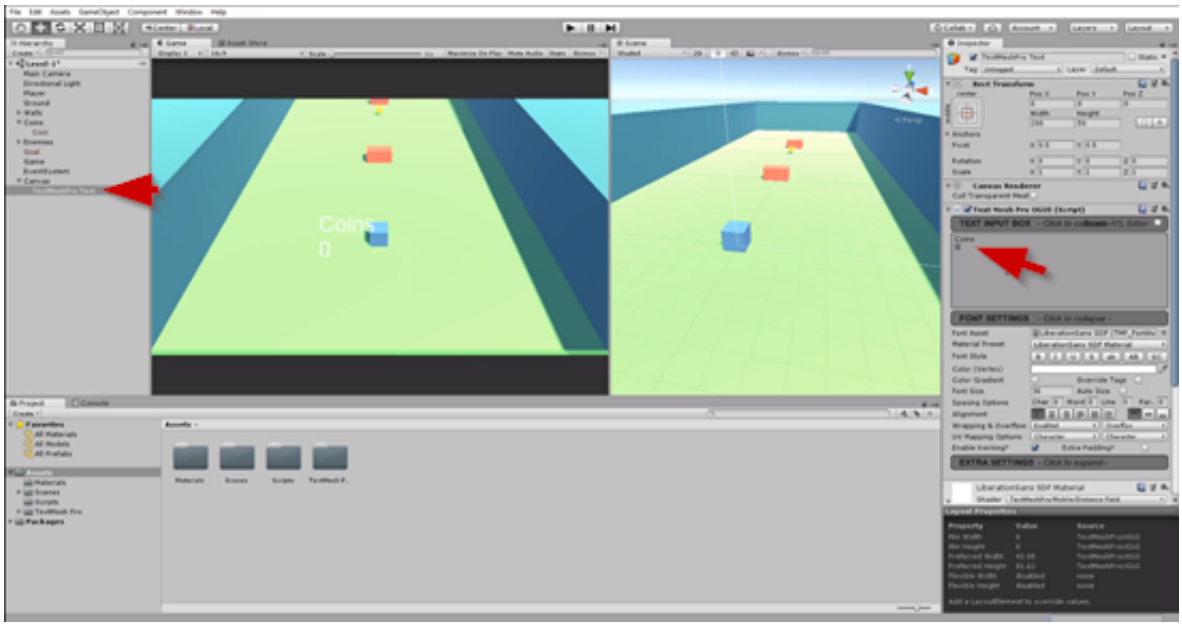


Right click in the Hierarchy>UI>TextMeshPro-Text

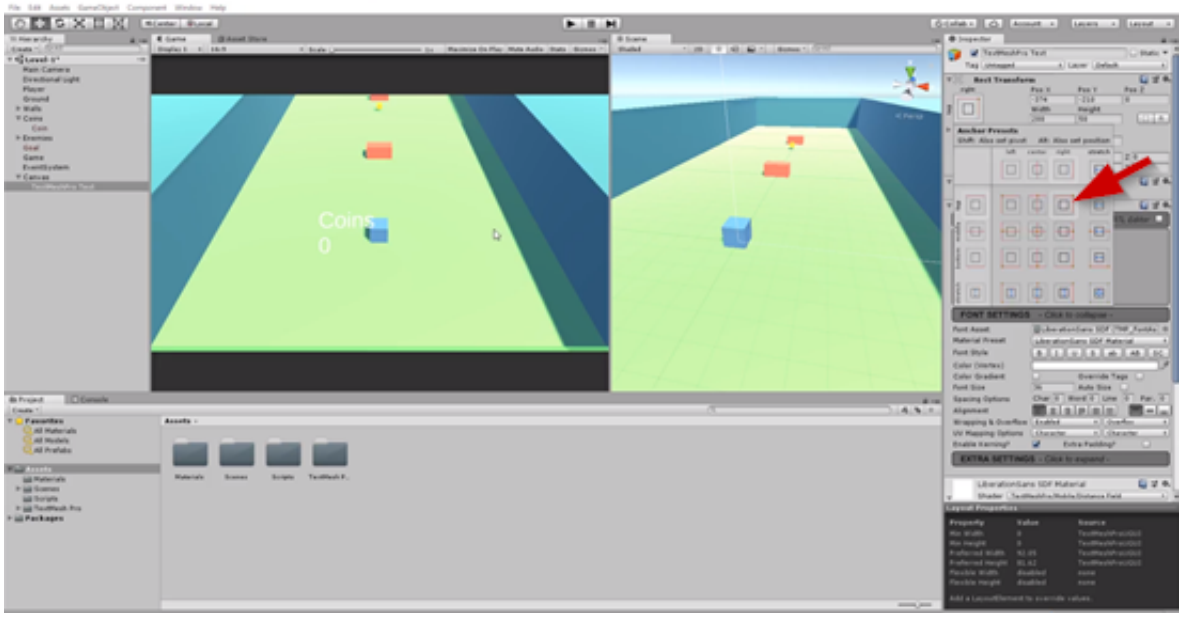


Now you will see that a Canvas element has been added to the **Hierarchy** and the TextMeshPro Text object is a child to that.

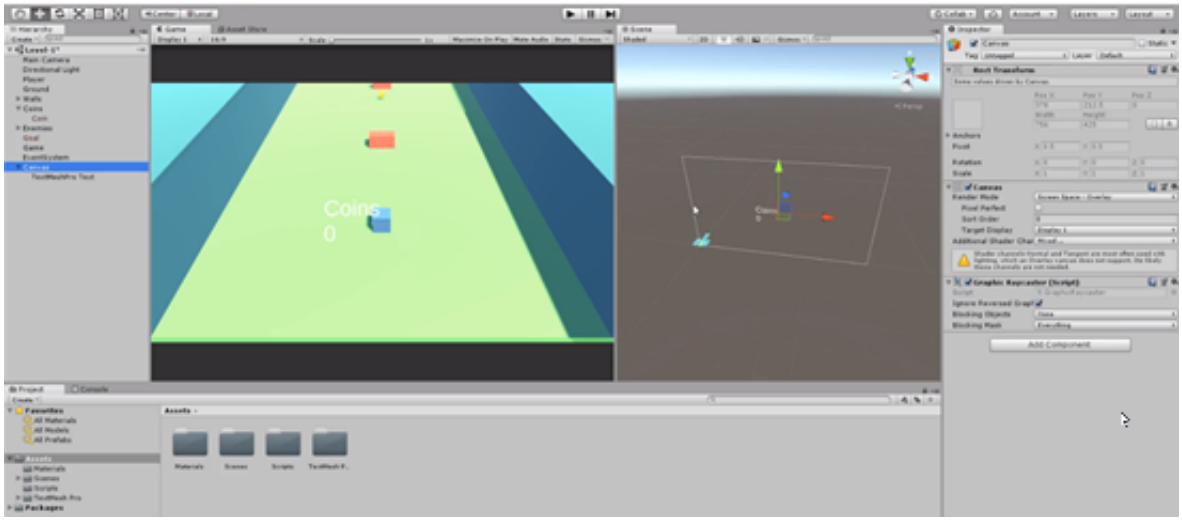
In the **Inspector** window with the TextMeshPro Text object selected enter the word **“Coins” and then “0” below that.**



We are going to want this to be in the top right corner of the game we need to set the Rect Transform anchor to be the top right corner, this just means that whenever the screen moves, its gonna move relative to the top right corner.

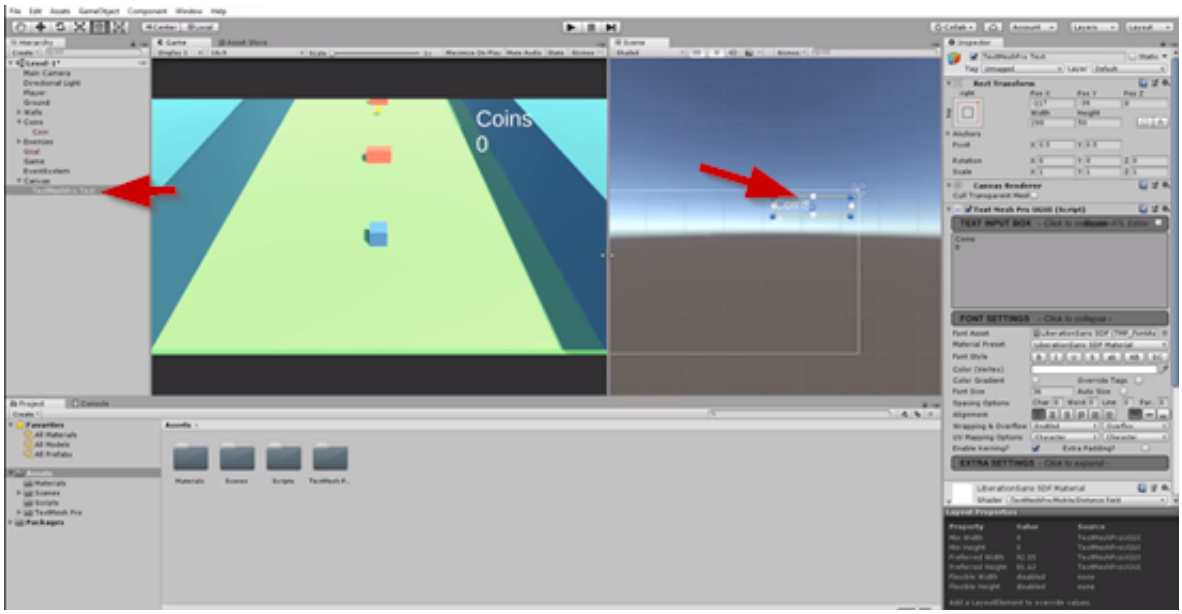


The canvas object is literally the canvas that we're going to paint our UI objects on, so if you hit the **"F" key or double click on it,** it's going to center in my frame.

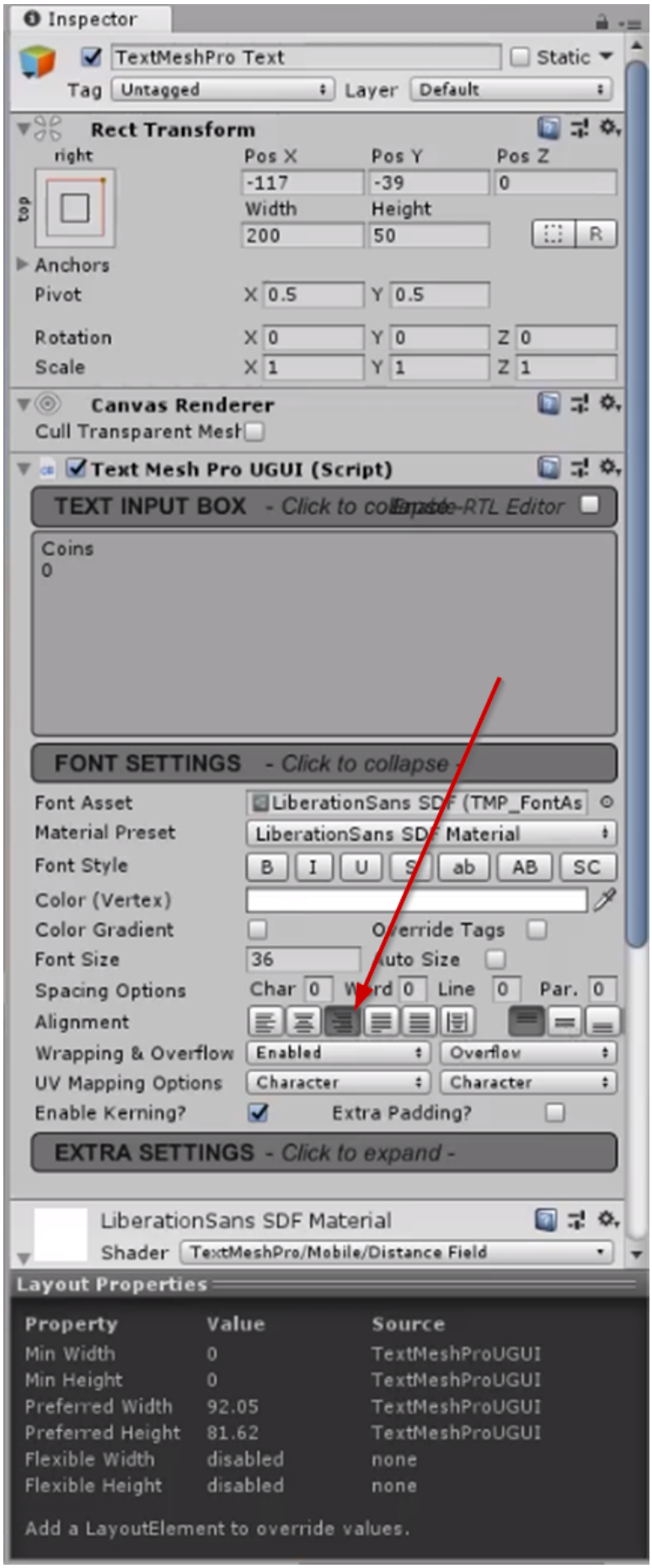


Now if you select the 2D tab in the Scene window you can go into 2D view and this is where the UI resides and it will match the view port. Stick with the **16:9** ratio when laying this out.

With the **TextMeshPro Text** object **selected** and the rect tool selected on the toolbar move it up into the **top right hand corner.**



Set the alignment up to be **right alignment** in the **Inspector** window.



You can also change the color of the text to your liking using the color picker.

I will use a yellow color.

Adjust the line height to be **-2 in the Inspector window.**

Enable the **Underlay** option in the **Inspector** window. Increase the **opacity to 182.**

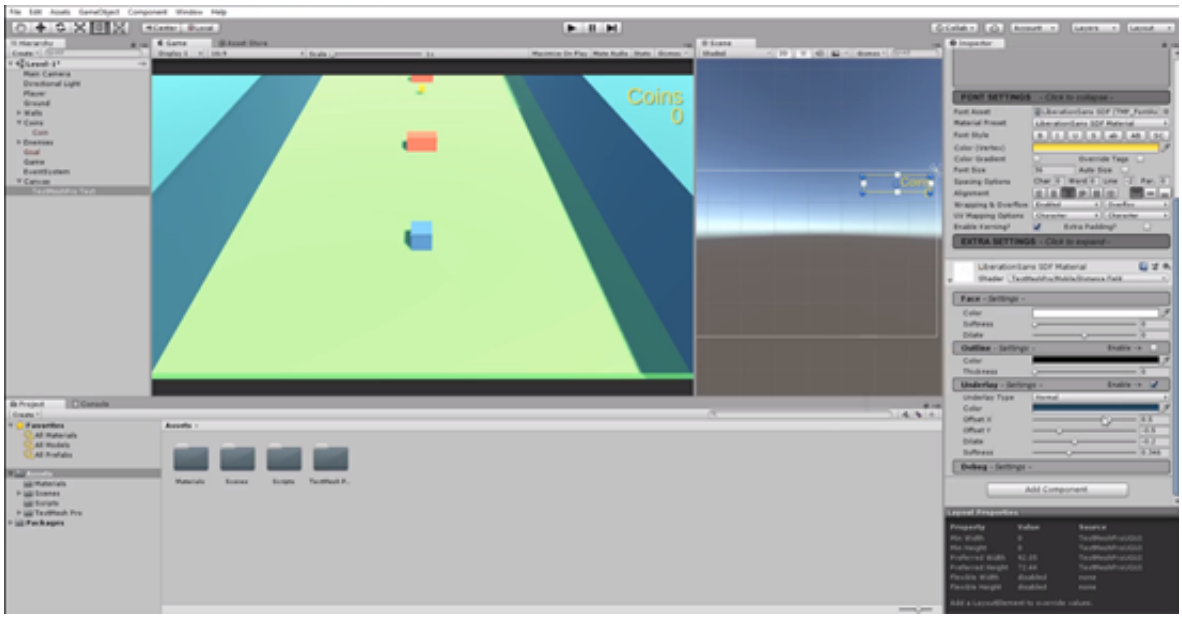
Set the **Offset X to 0.5.**

Set the **Offset Y to -.5.**

Adjust the **Dilate to -0.2.**

Adjust the **Softness to 0.346.**

Change the color to a dark blue color using the color picker.



We can now setup the UI to update as the coins are collected by the player.

We need to add some code to the Player script so **open the Player script** back up in the code editor.

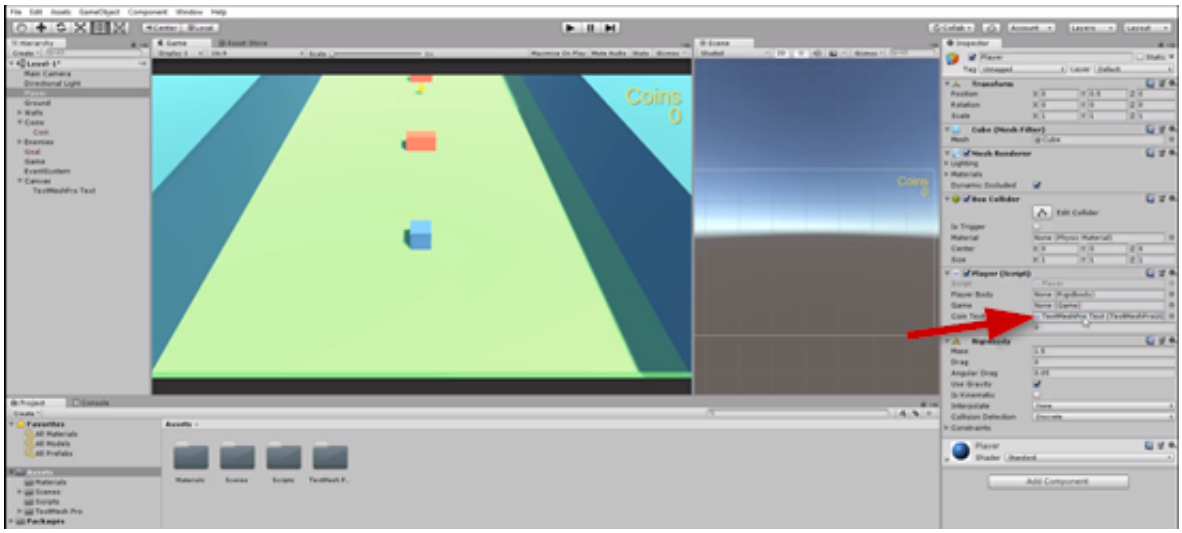
See the code below and follow along:

```
[SerializeField]
private TMPro.TextMeshProUGUI coinText;

private void OnTriggerEnter(Collider other)
    {
        switch (other.tag)
        {
            case "Coin":
                coins++;
                coinText.text = string.Format("Coins\n{0}", coins);
                Destroy(other.gameObject);
                break;
            case "Goal":
                // check for completion
               break;
            default:
                break;
        }
    }
```

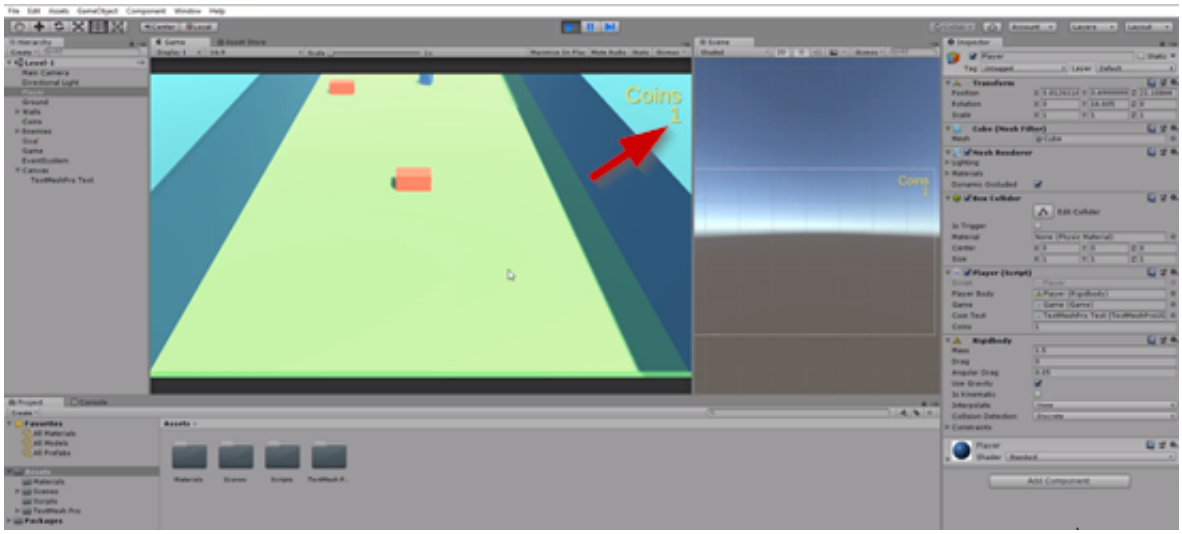
Save this script and navigate back to the Unity editor.

The reference to the coins has to be setup in the Inspector on the script component.

We need to **assign the TextMeshProUGUI by dragging and dropping the game object into the correct spot on the script component.**



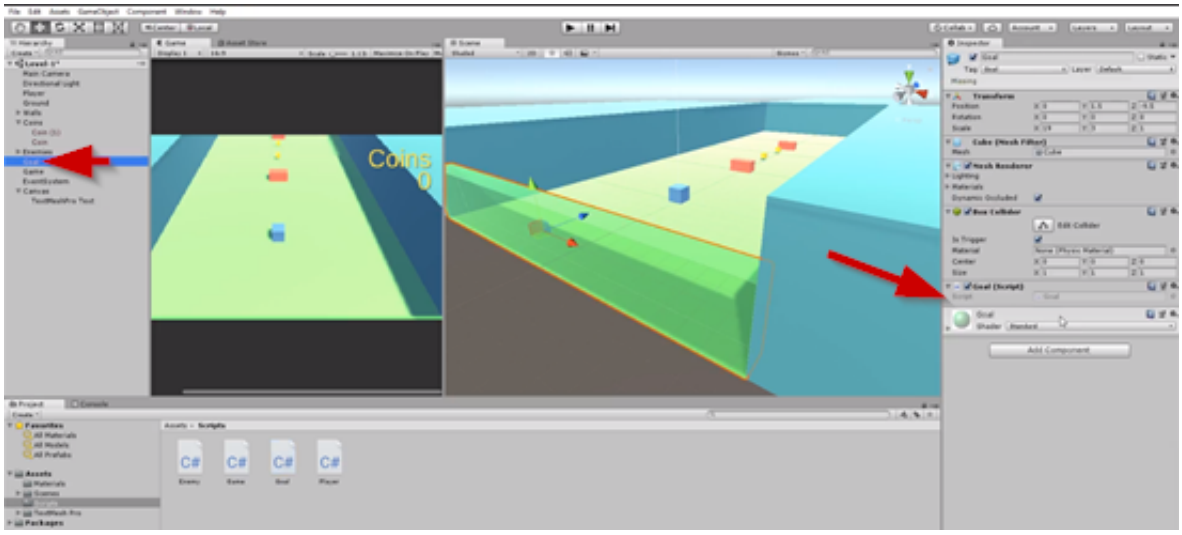
Hit the **Play** button and test out the changes.



Once the coin is collected the UI is being updated properly.

In this lesson we will setup the goal check completion.

Create a new C# script in the **Scripts folder** and call it **“Goal”** this script needs to be attached to the Goal game object in the scene.



Open the Goal script up in the code editor.

See the code below and follow along:

```
private int requiredCoins;
    private Game game;
 // Use this for initialization
 void Start () {
        game = FindObjectOfType<Game>();
        requiredCoins = GameObject.FindGameObjectsWithTag("Coin").Length;
 }

    public void CheckForCompletion(int coinCount)
    {
        if (coinCount >= requiredCoins)
        {
            game.LoadNextLevel();
        }
        else
        {
            Debug.Log("You do not have enough coins!");
        }
    }
```

Save this script and **open the Player script** up in the code editor.

See the code below and follow along:

```
private void OnTriggerEnter(Collider other)
    {
        switch (other.tag)
```



```
        {
            case "Coin":
                coins++;
                coinText.text = string.Format("Coins\n{0}", coins);
                Destroy(other.gameObject);
                break;
            case "Goal":
                other.GetComponent<Goal>().CheckForCompletion(coins);
                break;
            default:
                break;
        }
    }
```

Save this script and navigate back to the Unity editor to test out he changes by hitting the **Play** button.

Once the Player has collected all the coins in the level and then collides with the Goal they will proceed to the next level.

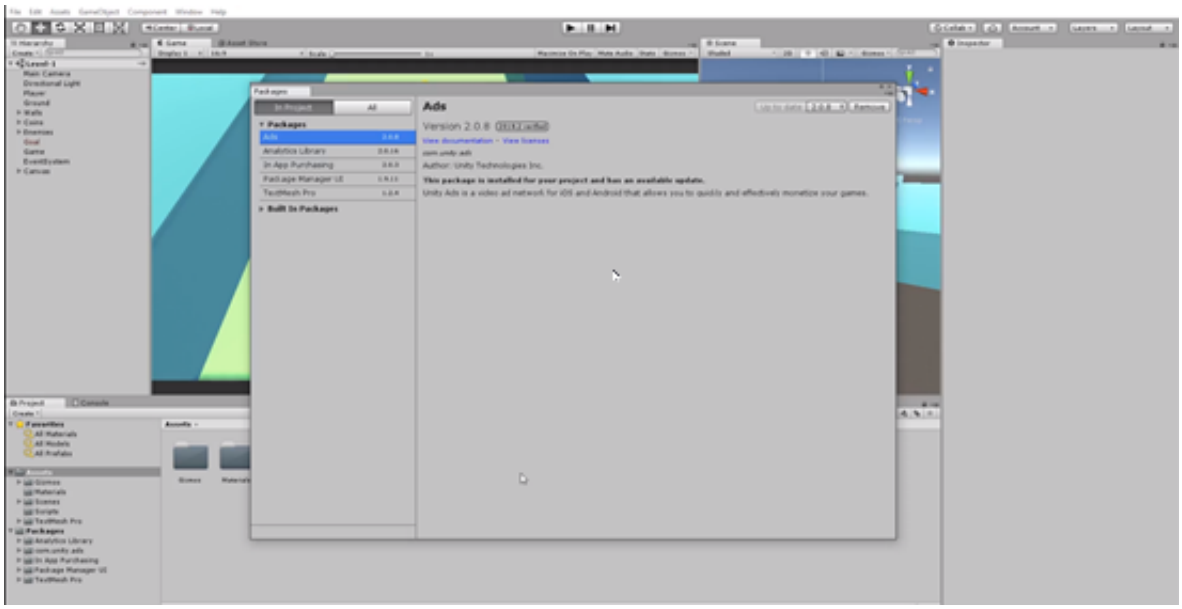
Everything that is done in the first level will be copy and pasted into the other levels we create.

In this lesson we will be setting up the camera system using the Cinemachine tool that comes with Unity.

Camera System

- Import Cinemachine tool
- Setup a “virtual camera”
- Make the camera follow player around
- Get latest preview release to fix an issue

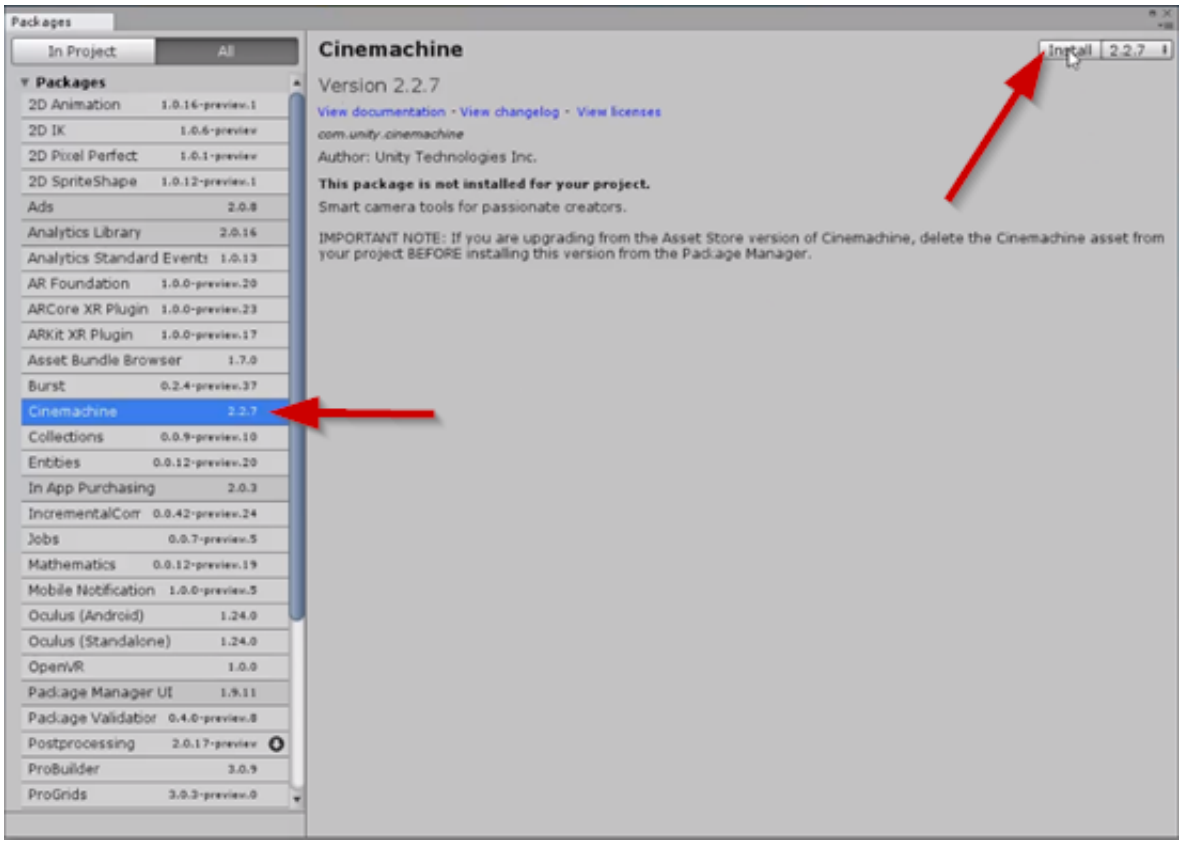
In the Unity editor **select Window>Package Manager.**



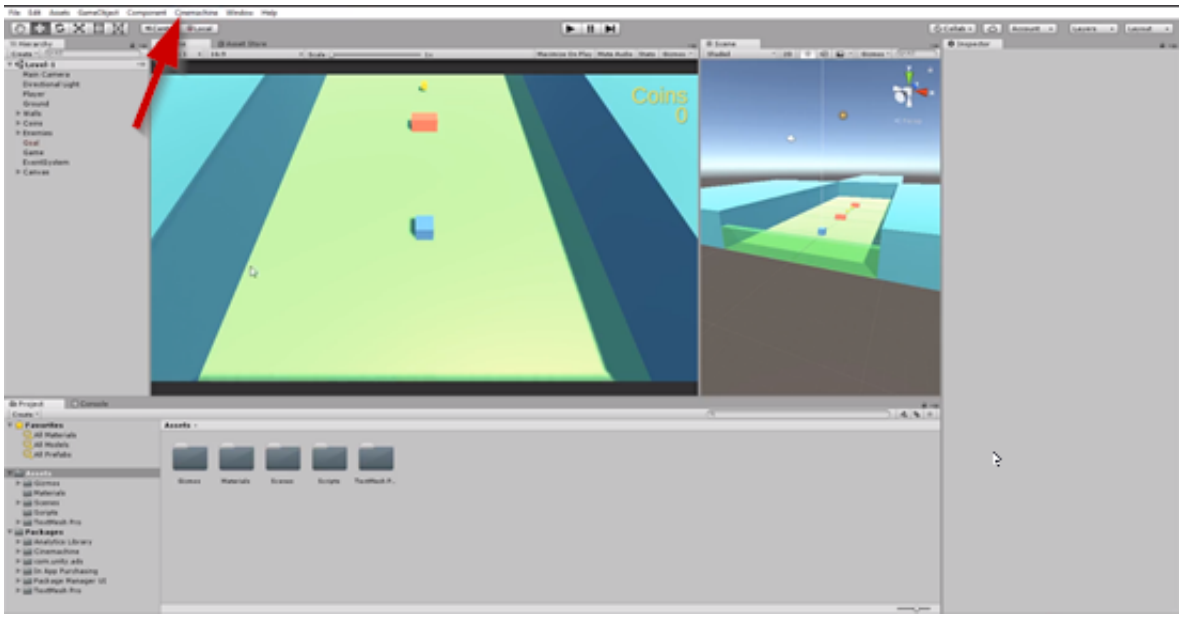
This is where we can handle all the packages in Unity.

Click on the **All tab** and scroll down the list to **“Cinemmachine”** if you have access to any version later than **2.2.7** then you will be good to go, and won’t have to deal with the issue mentioned earlier in the video.

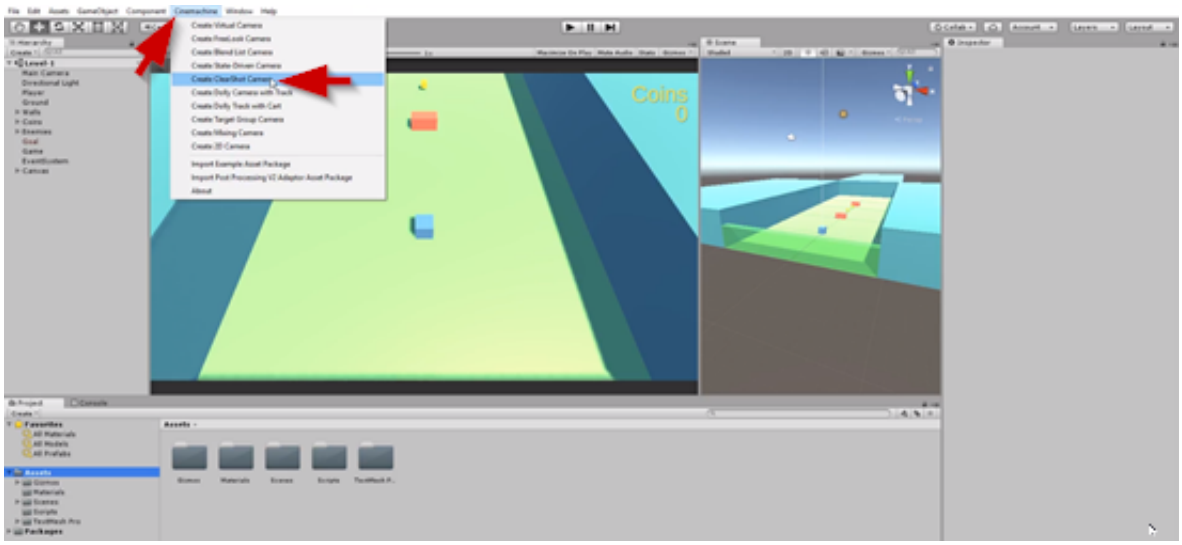
Click on the **Install button.**



Once Cinemachine is installed you will have the **Cinemachine button** inside the Unity editor.



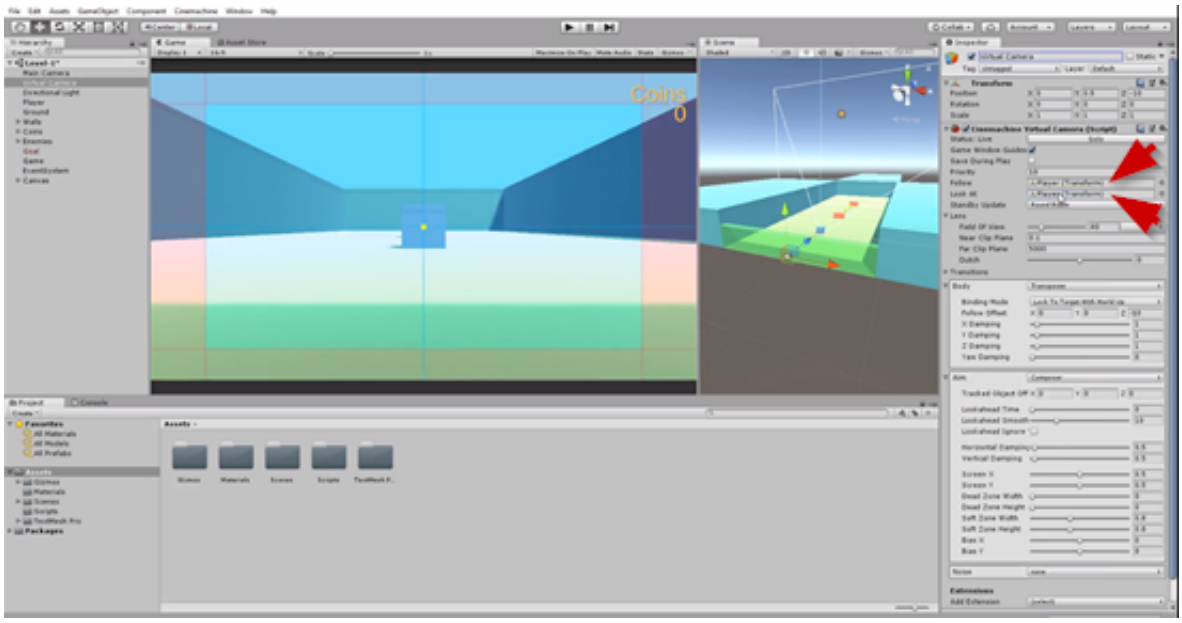
Select the **Create Virtual Camera option** from the Cinemachine drop down menu.



Once you have selected that option there will be a bunch of options that pop up in the **Inspector.**

Rename the **CM vcam1** to **“Virtual Camera”** right now all we care about is the **Follow and the Look At properties.**

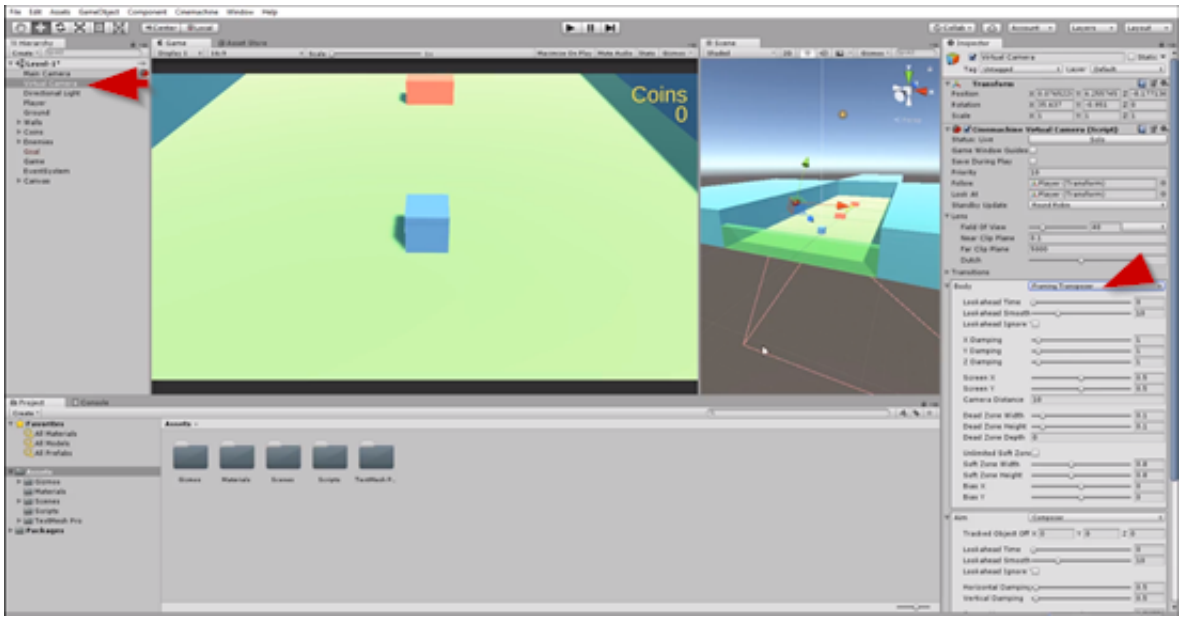
We need to setup the references for these options so take the **Player and drag and drop it onto the Follow field and the Look At field.**



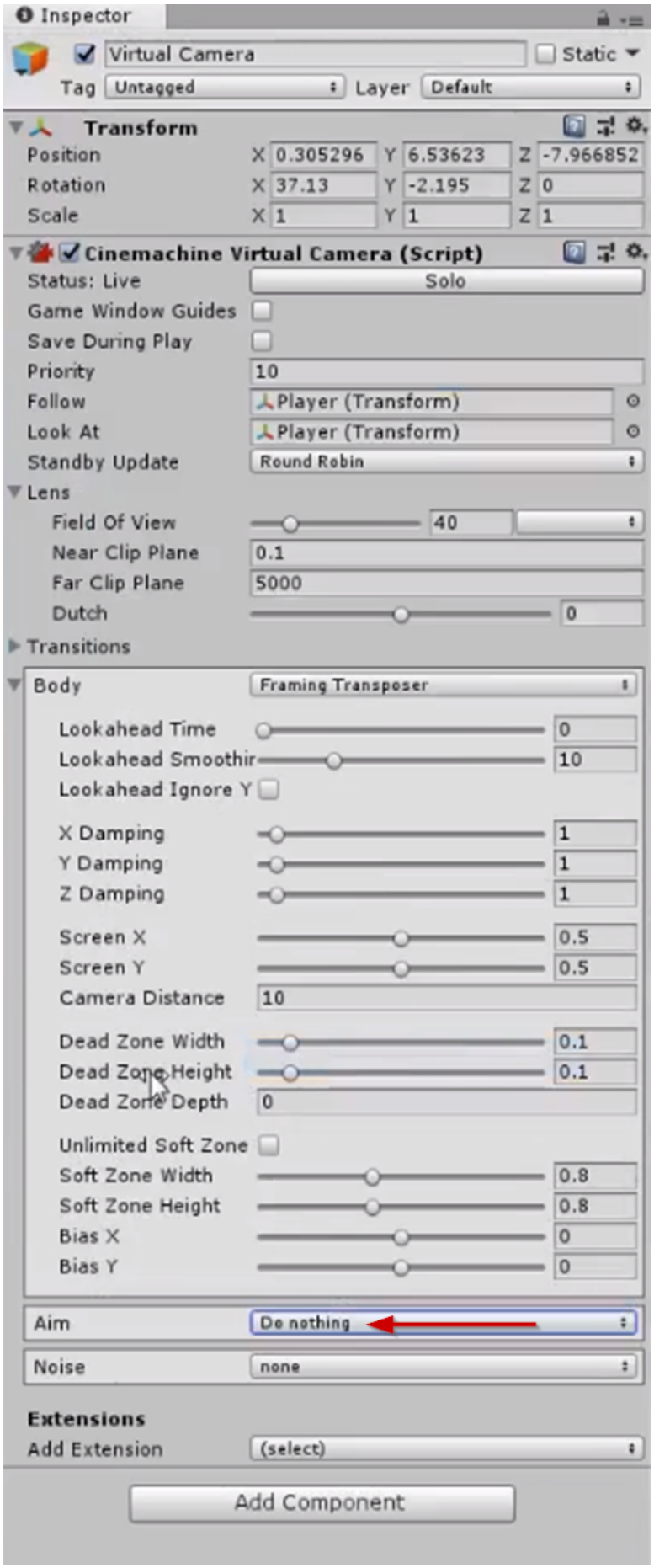
Now one of the issues we will have right away is if you play the game is that the game in unplayable due to the way the camera rotates.

So we can solve this issue by making sure that the camera does not rotate with the player.

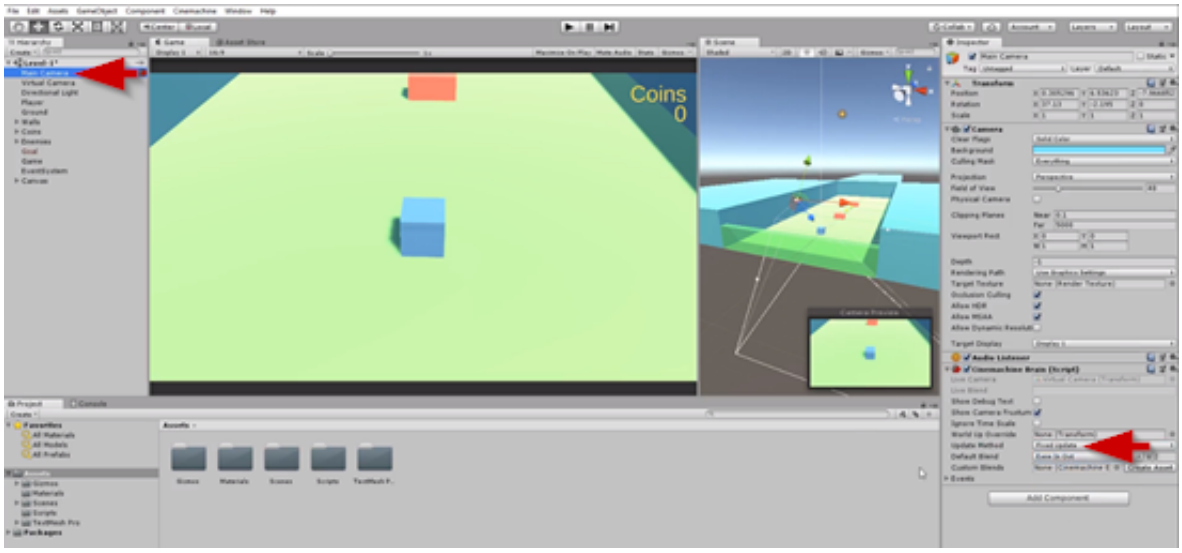
Select the **Virtual Camera from the Hierarchy** and in the **Inspector** where the Body option is set this to be **“Framing Transposer.”**



Now the **Aim** needs to be **set to “Do Nothing” in the Inspector.**



Hit the **Play** button and test out the changes. You will see that the camera is following the player and the rotation is gone, but there is a jittering issue. To fix this issue if you are not using the 2.2.8 version or later of Cinemachine **select the Main Camera in the Hierarchy** and in the **Inspector** select the **Fixed Update option for the Update Method.**



Now we need to update the packages manifest folder of data, and this is for the 2.2.7 version, if you have version 2.2.8 or later this is not needed.

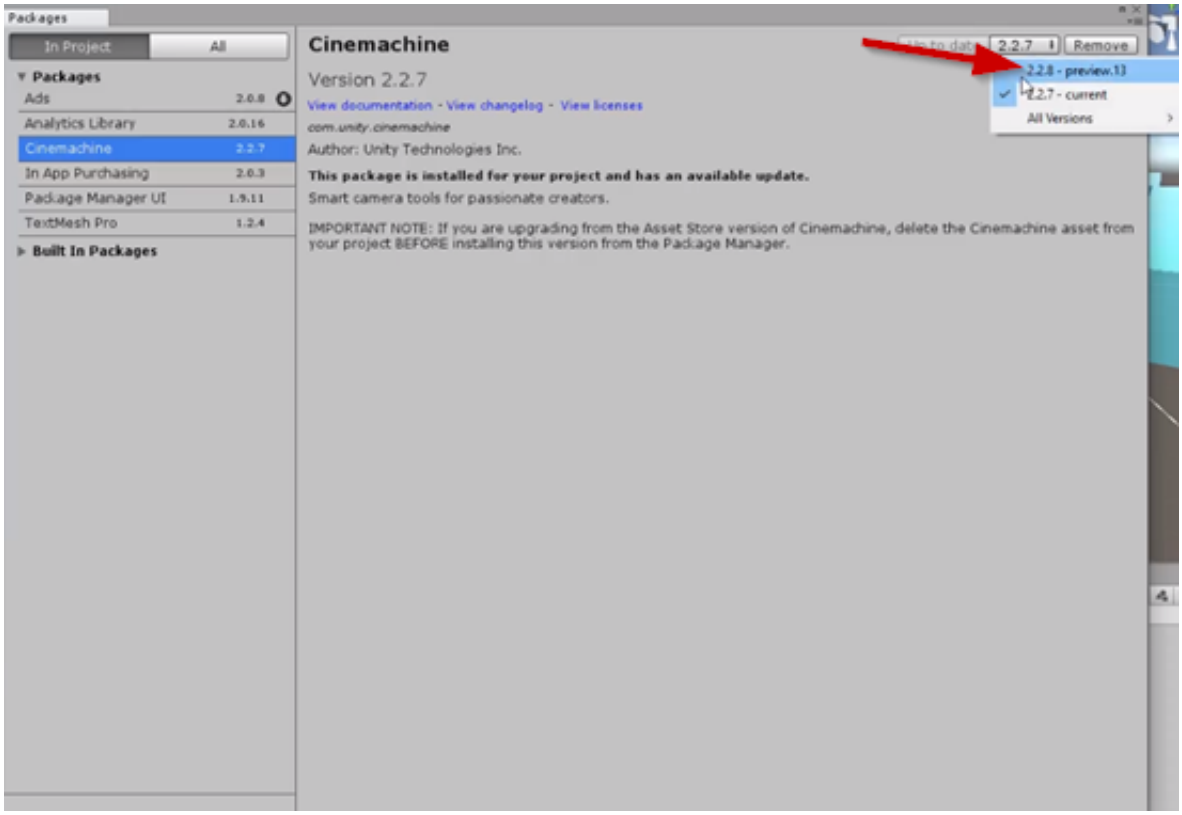
Right click in the **Packages folder>Show In Explorer>Packages.**

Once in here there will be a **Manifest folder.** Double click this and open it up. There will be a list of all the packages in this JSON file.

You will need to add this line of text **right above the dependencies line “registry”: “https://staging-packages.unity.com”,**

Save this file using Ctrl+S and then you can close the file.

Now you need to go to the **Cinemachine package window** and select the **2.2.8 version** from the drop down menu.

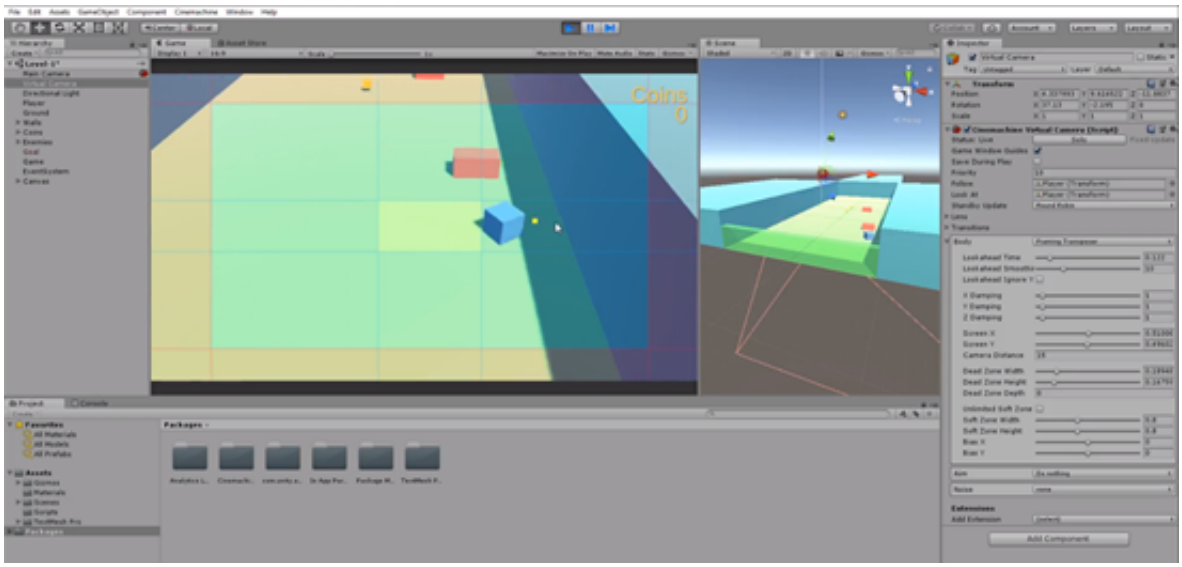


C**lick on File>Build Settings>Player Settings,** and in the Inspector where the Configuration section is there will be a Scripting Define Symbols field and in this field you can type **“CINEMACHINE_EXPERIMENTAL_DAMPING”**

Hit the **Play** button and test the changes.

Select the V**irtual Camera in the Hierarchy** and adjust the **Camera Distance to 15 in the Inspector.**

Adjust the **Look ahead Time to 0.122.**



Adjust the **Camera Distance to 20.**

Feel free to play with the camera settings till you get the camera how you would like it to be.

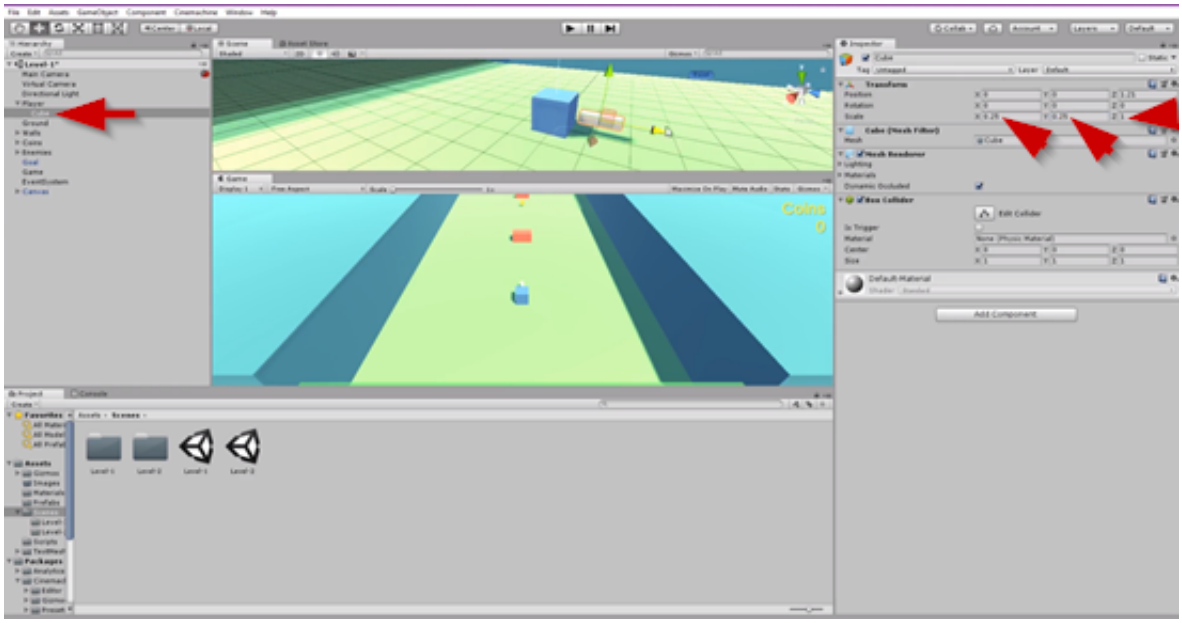
In this lesson we will be setting up the player's sword attack. The sword attack will work so that if the player is near an enemy the player can always press the attack button to swing the sword and hopefully kill the enemy.

- Create our player sword
- "Attack" to "swing" it
- Check for collisions with enemies.

In the Unity editor **select the Player** game object in the **Hierarchy**. **Right click and create 3D Object>Cube.**

This **cube** will be a **child to the Player**. This cube will be a simple sword. Bring the sword out to the front of the player, and the scale needs to be adjusted a bit.

Make the **scale on the X axis to be .25, the Y axis is .25, and keeping it at 1 on the Z axis** should be fine.



You can adjust the sword to look like its in the Player's right hand. Adjust the **position on the X axis to be 0.31.**

Rename the Cube to **"Sword."**

Open up the Player script, and we will be adding in a method that's going to handle enabling and disabling the sword so that we can actually check for those collisions.

See the code below and follow along:

```
private GameObject sword;

private void PerformAttack()
    {
        if (!sword.activeSelf)
        {
            sword.SetActive(true);
        }
    }
```


Save this script, this method will not handle disabling the sword itself, we need the sword to handle that part so whenever it becomes active we want the sword to count down for however long it should be active. and once the countdown runs out, then it will deactivate itself until the Player tells the sword to attack again.

We need to **create a new C# script in the Scripts folder** that will be specifically for the sword.

Name the new script "Sword" and **open it** up in the code editor.

See the code below and follow along:

```
private float attackLength = .5f;
 // Update is called once per frame
 void Update () {
        attackLength -= Time.deltaTime;
        if (attackLength <= 0)
        {
            gameObject.SetActive(false);
            attackLength = .5f;
        }
 }
```

Save the script and open up the Player script once more we need to add the code to the Start function to setup the reference between the Player object and the Sword which is a child to the Player.

See the code below and follow along:

```
// Use this for initialization
 void Start ()
    {
        sword = transform.GetChild(0).gameObject;
        game = FindObjectOfType<Game>();
        playerBody = GetComponent<Rigidbody>();
 }
```

Now we need to setup the input for the attack so in the **Player script we need to add an if statement to check for the input in the Update function.**

See the code below and follow along:

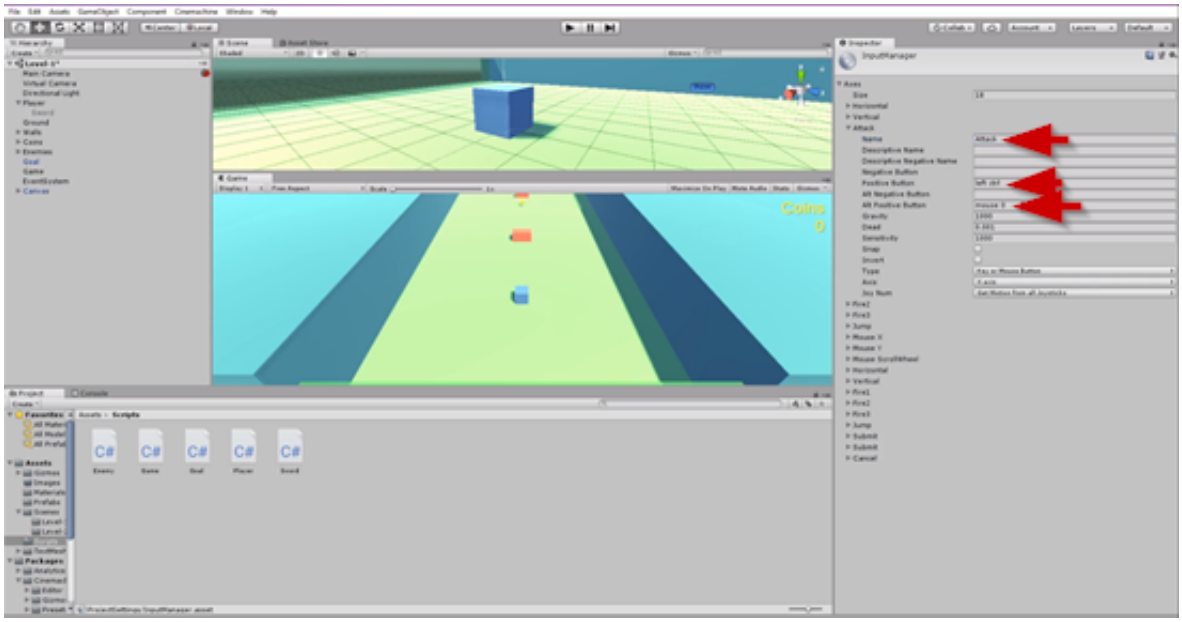
```
// Update is called once per frame
 void Update ()
    {
        inputVector = new Vector3(Input.GetAxis("Horizontal") * 10f, playerBody.veloc
ity.y, Input.GetAxis("Vertical") * 10f);
        transform.LookAt(transform.position + new Vector3(inputVector.x, 0, inputVect
or.z));
        if (Input.GetButtonDown("Jump"))
```



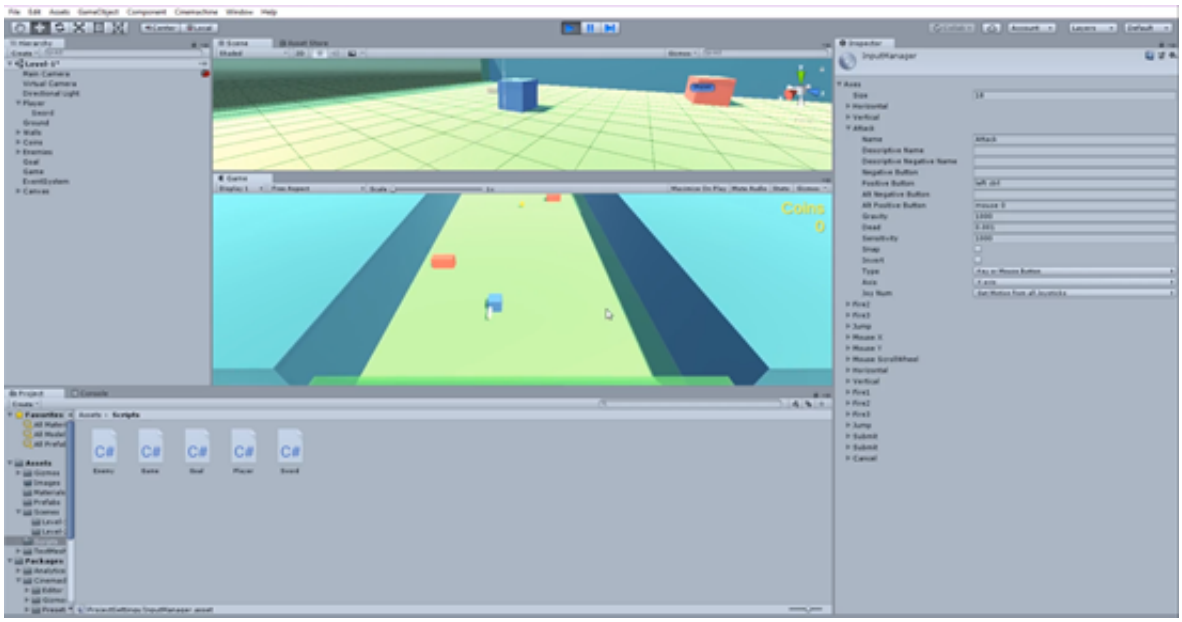
```
        {
            jump = true;
        }
        if (Input.GetButtonDown("Attack"))
        {
            PerformAttack();
        }
    }
```

Save the script, and navigate back to the Unity editor.

We need to **setup the attack button in the Project Settings,** so we need to **rename Fire1 to "Attack"** so now whenever the left control or mouse zero is pressed the Player will attack.



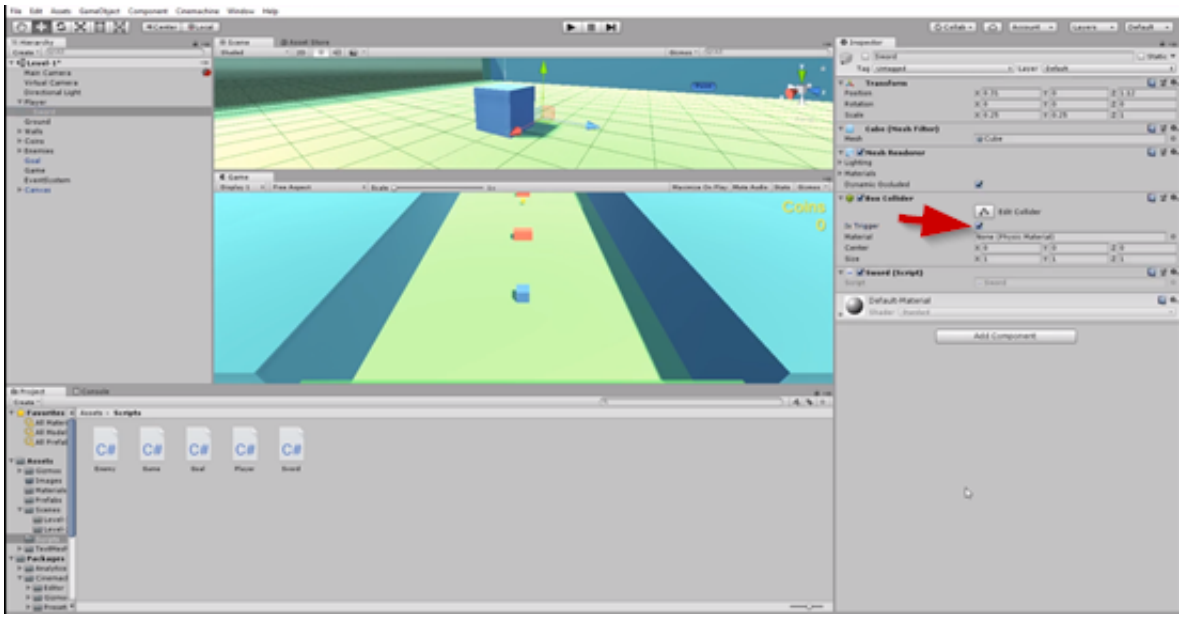
We can now try the Sword attack out, hit the Play button and press the left control key or the left mouse button and you can see the Sword be enabled on the Player.



But, we now have a new issue. The sword never disables, why is this happening? That is because the Sword game object doesn't have the Sword script component attached to it. Go ahead and attach the script to the Sword by dragging and dropping it onto the Sword game object.

Now **hit the Play button** and the Sword will enable and disable as it should properly. The sword is a child of the Player so it will move around and rotate along with the Player.

Now what we need to do is setup the sword so that if it hits an enemy while its enabled we can kill some enemies with it. We can set this up by using collision detection. First, make the Sword a trigger by t**oggling the Is Trigger option on the Box Collider component in the Inspector.**



We can do the collision detection in the Sword script using an if statement.

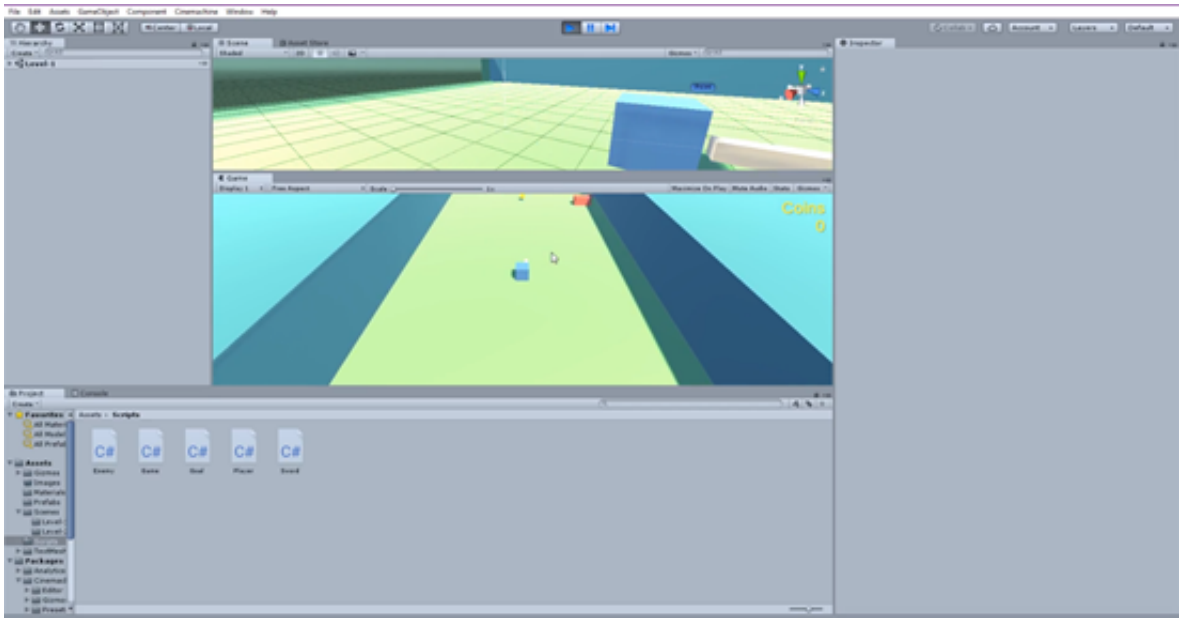
Open the Sword script back up in the code editor.

See the code below and follow along:


```
private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Enemy"))
        {
            Destroy(other.gameObject);
        }
    }
```

Save the script and navigate back to the Unity editor to test the changes out by hitting the **Play button.**

Now when the Player attacks with the sword and the sword collides with an enemy the enemy is destroyed.



In this lesson, we are going to set up the pause menu in the game.

In-Game Menu (pause)

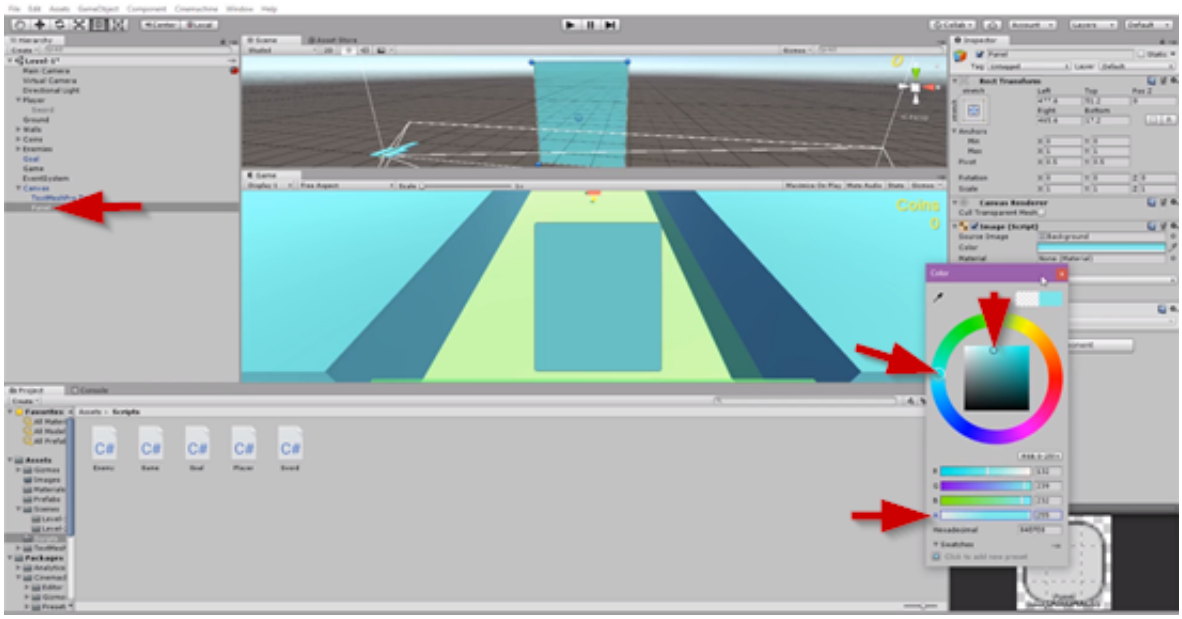
- Create a menu
- Enable menu when button pressed
- Add buttons for actions

In the Unity editor **select the Canvas element, right click>Panel .**

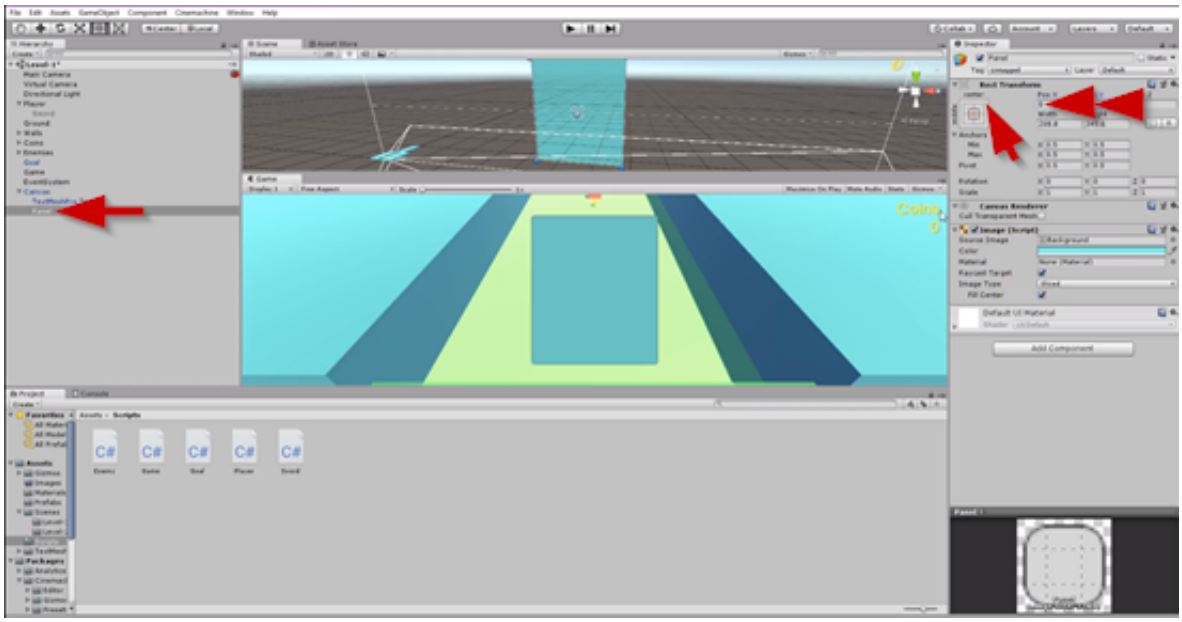
A panel is just an object that has a 2D image component attached to it that we can resize and use as sort of a window, for our UI.

We will be using this panel to organize our buttons for the pause menu. Position and resize the panel in the middle of the screen. Use the rect tool to adjust the size of it.

Adjust the color of the panel, I am going to **choose a blue color that will kind of go with the game, and I will make the Alpha full.**

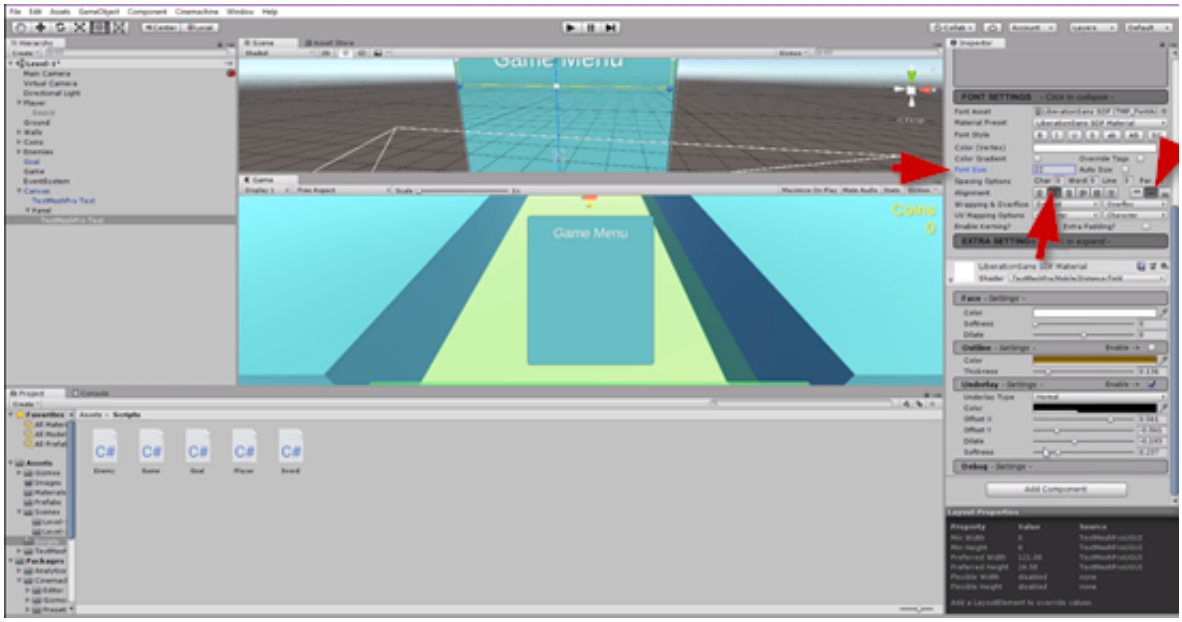


Make sure the panel is **centered anchored**. Make sure the **position is at 0,0,0.**

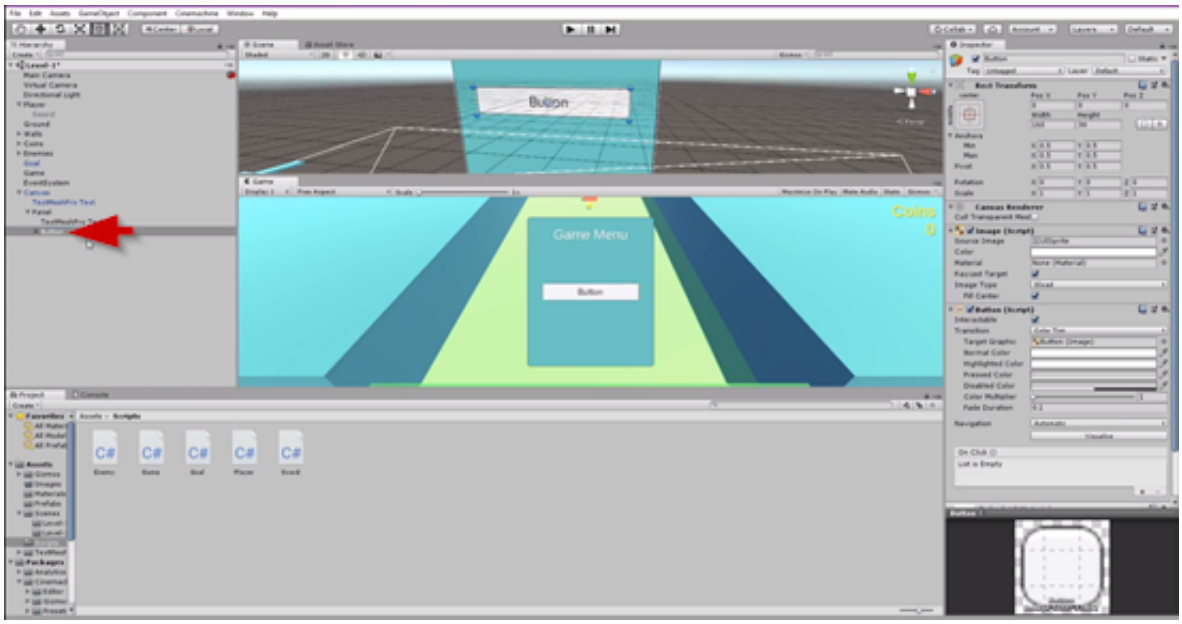


Add a TextMeshPro Text element to the Panel. This needs to be centered for alignment and centered for both the horizontal and vertical positions.

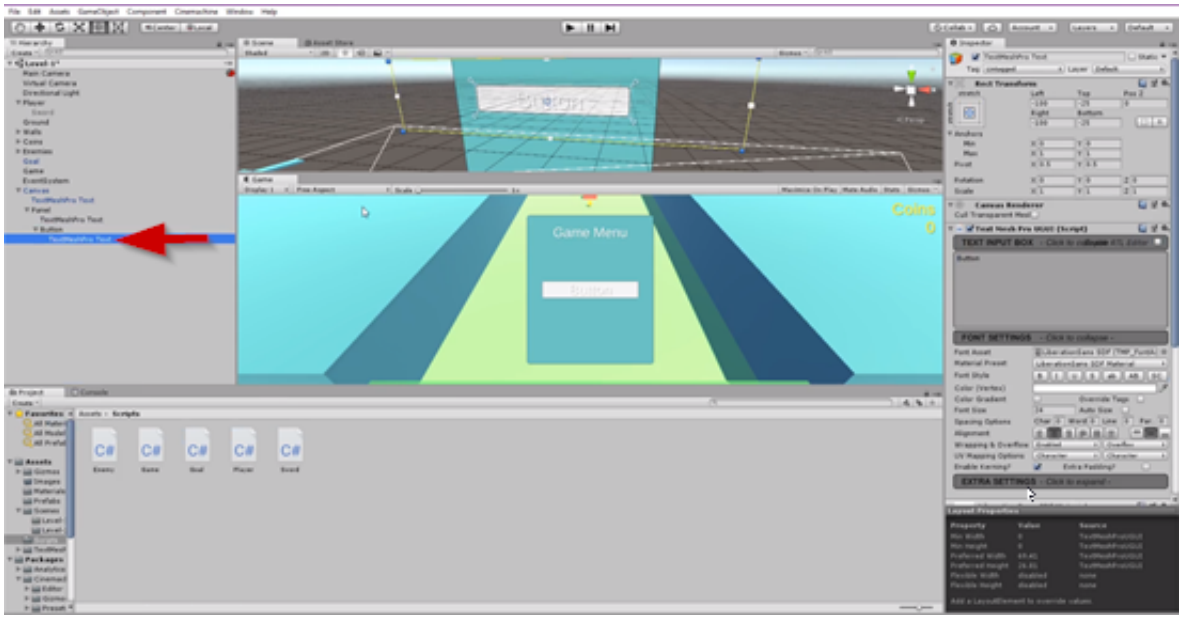
Then just drag the text element up to the top of the panel. In the **Text Input Box type “Game Menu” and change the font size to 22.**



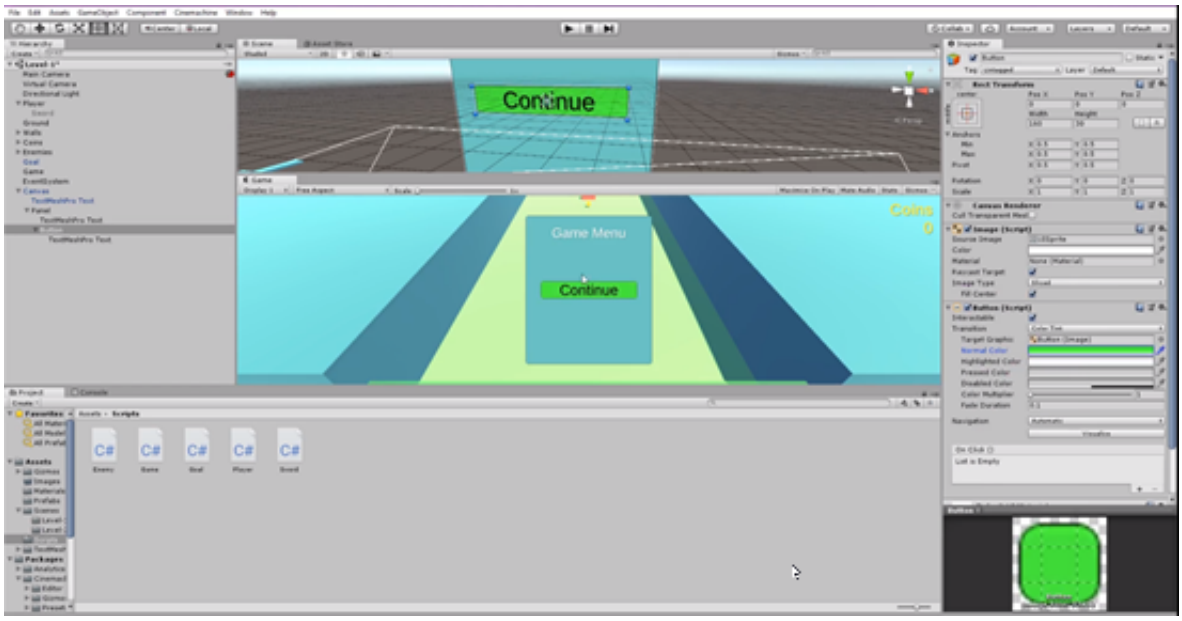
Below the Game Menu text element we want to add some buttons. With the **Panel selected in the Hierarchy Right Click>UI>Button.**



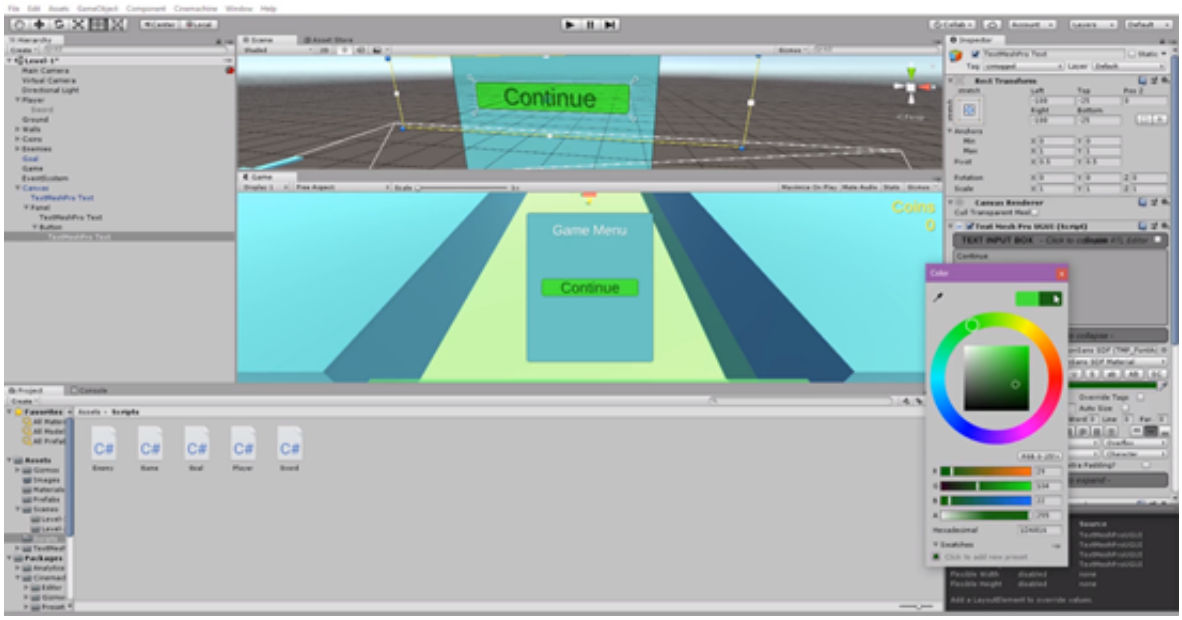
There is a text element on the button, and its a child object to the button itself. This text element on the button is a built-in text from the Unity built-in text. We can **delete this text element and we can add a TextMeshPro-Text element to the Button.**



In the **Text Input Box type “Continue”** and change the actual color for the button to a green color.



Change the **Continue text to be a darker green color using the color picker.**

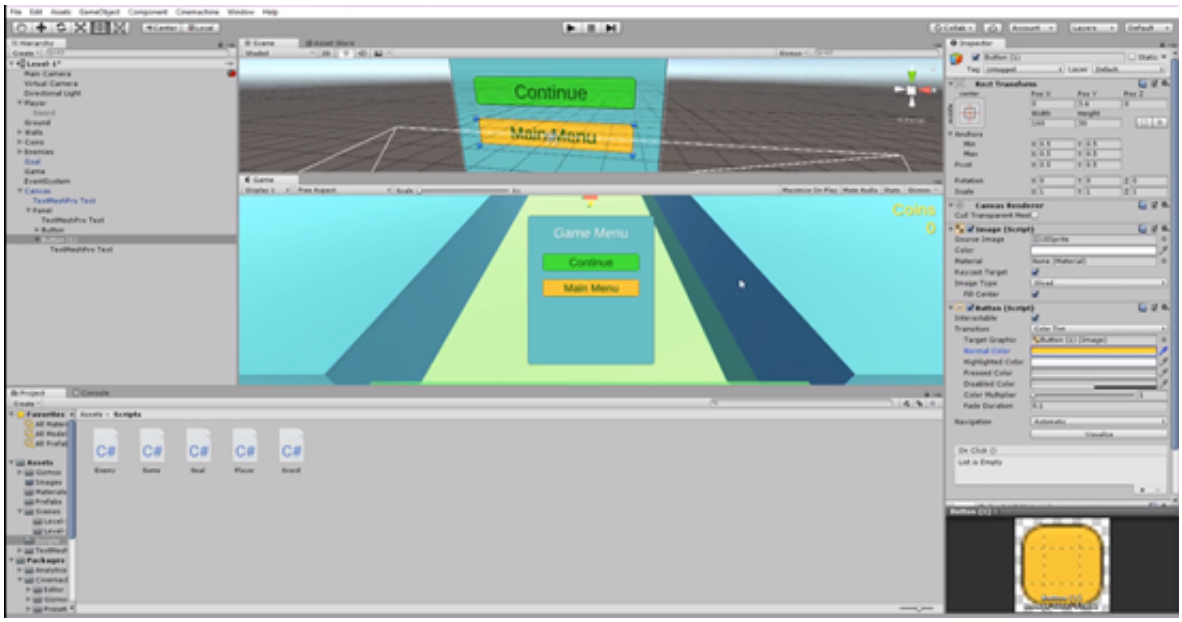


Adjust the font size to 18.

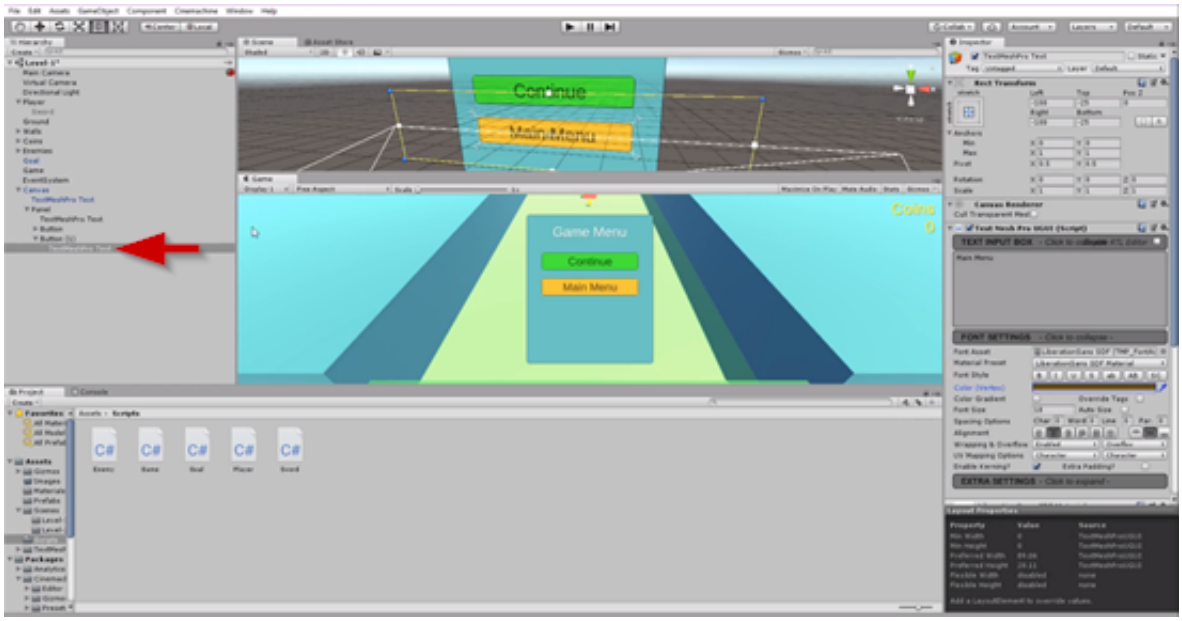
Position the button upwards right under the Game Menu text on the panel. **The Y position value should be 45.6.**

Now we need to add some more buttons to the panel, and we can do this by **duplicating the button we already have.**

Once we duplicate the button though, we'll **need to reposition it below the Continue button.** The **Y value should be 3.6.** Call this button **"Main Menu"** and you can change the color of this button if you like.



You can change the color of the Main Menu text to a darker color.



Duplicate the Main Menu button. Then reposition it below the Main Menu button, but we need to make sure we position the button that gives us the same offset from each button so we don't have a weird, badly layed out menu.

What we can do for this is **add a component to the Panel** that will handle organizing the elements in the panel for us so that we handle the spacing between the buttons much more effectively.

So **add a Vertical Layout Group component to the Panel**, this component will allow us to vertically lay out elements within our panel.

So **select the Padding option** on the Vertical Layout component in the Inspector and make these changes:

Left = 10

Right = 10

Top = 10

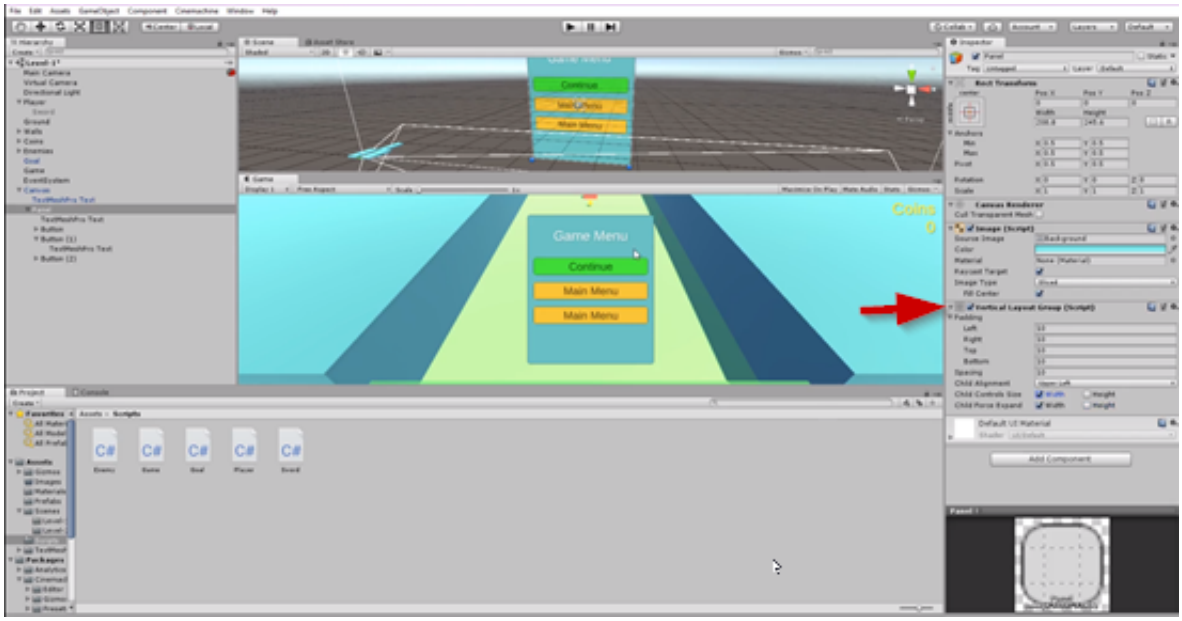
Bottom = 10

This will give us some cushion around the panel edges.

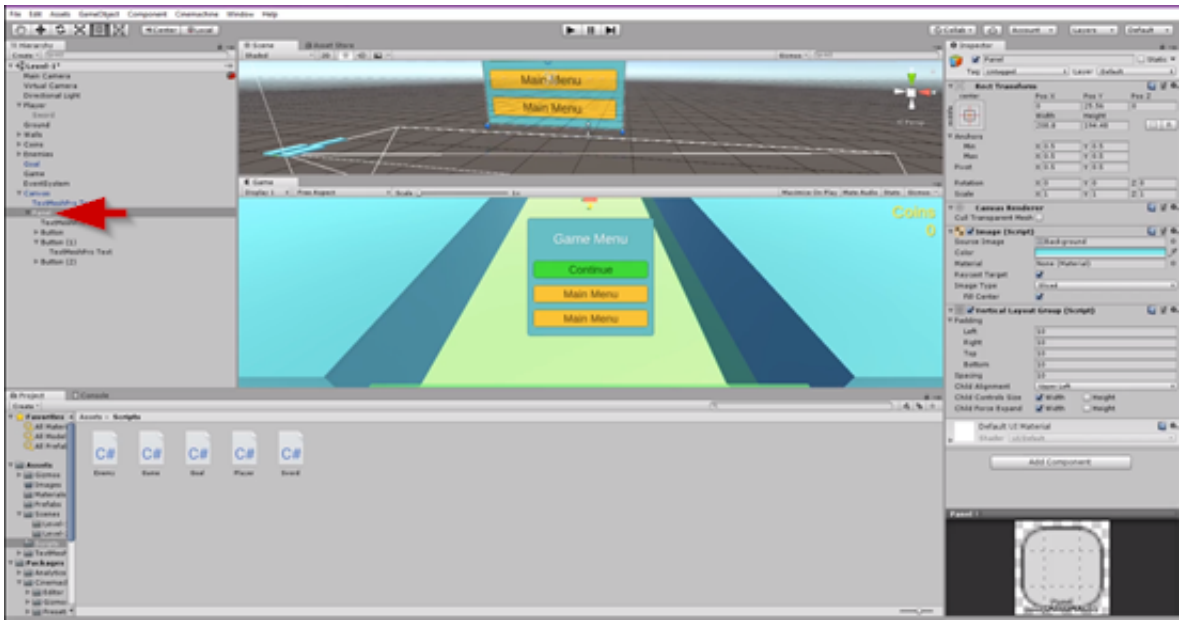
Change the **Spacing option to be 10.**

Uncheck the option for **Child Force Expand Height.**

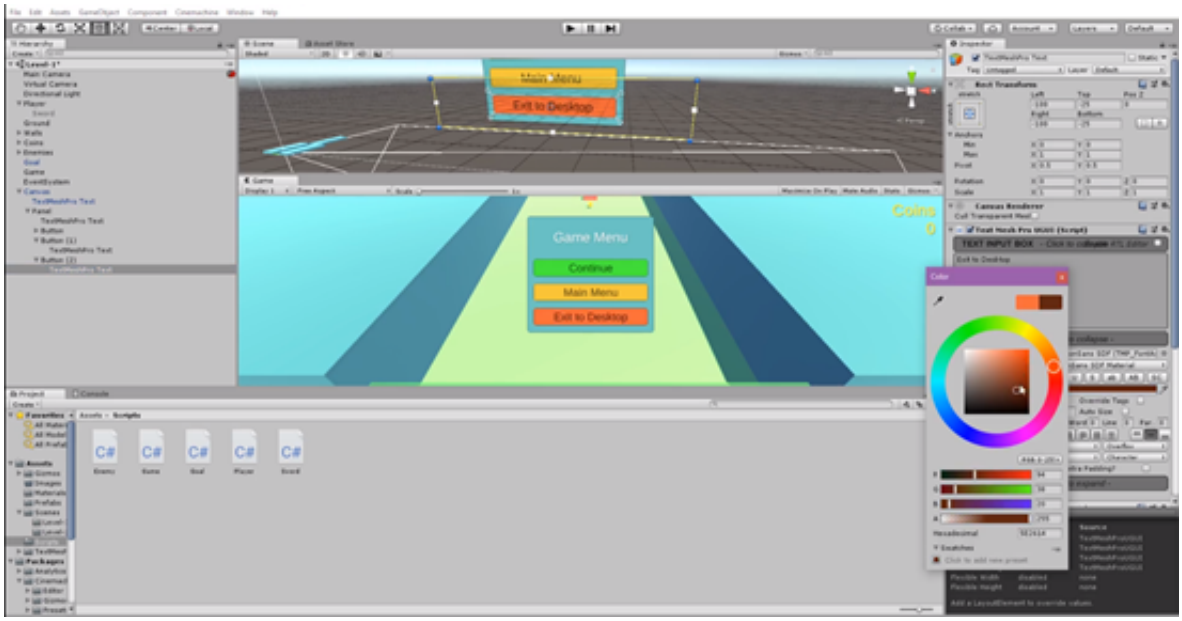
Now **toggle** the option for **Child Controls Size Width.**



Now just **resize the panel** to match the buttons.



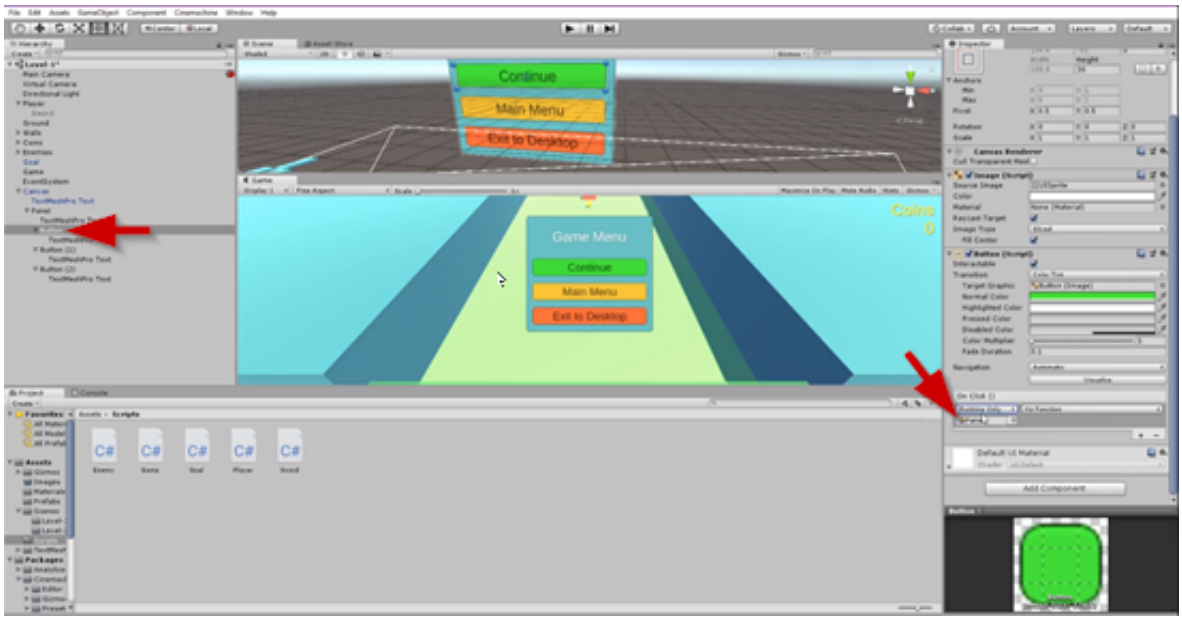
Change the text for the last button to say **“Exit to Desktop”** change the color of this button to be red since its the exit button.



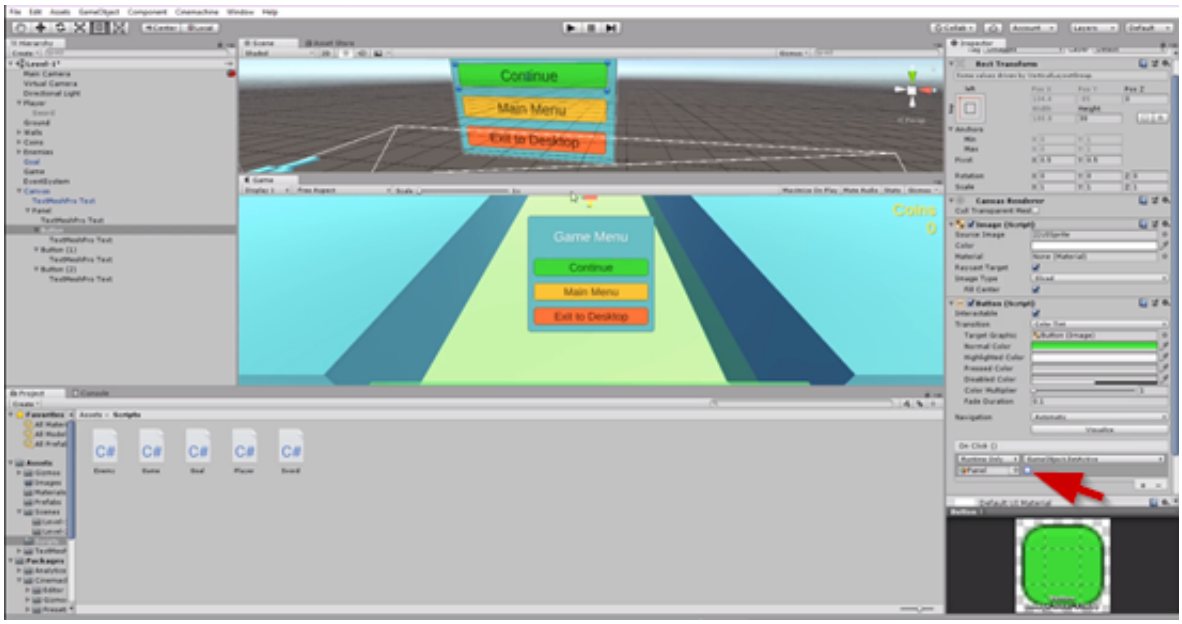
Now we want these buttons to do things when they are clicked on.

Select the **Button game object which is the Continue button** and look at the **Inspector**, we want the **On Click** event to be setup so we can control what happens when this button is clicked on.

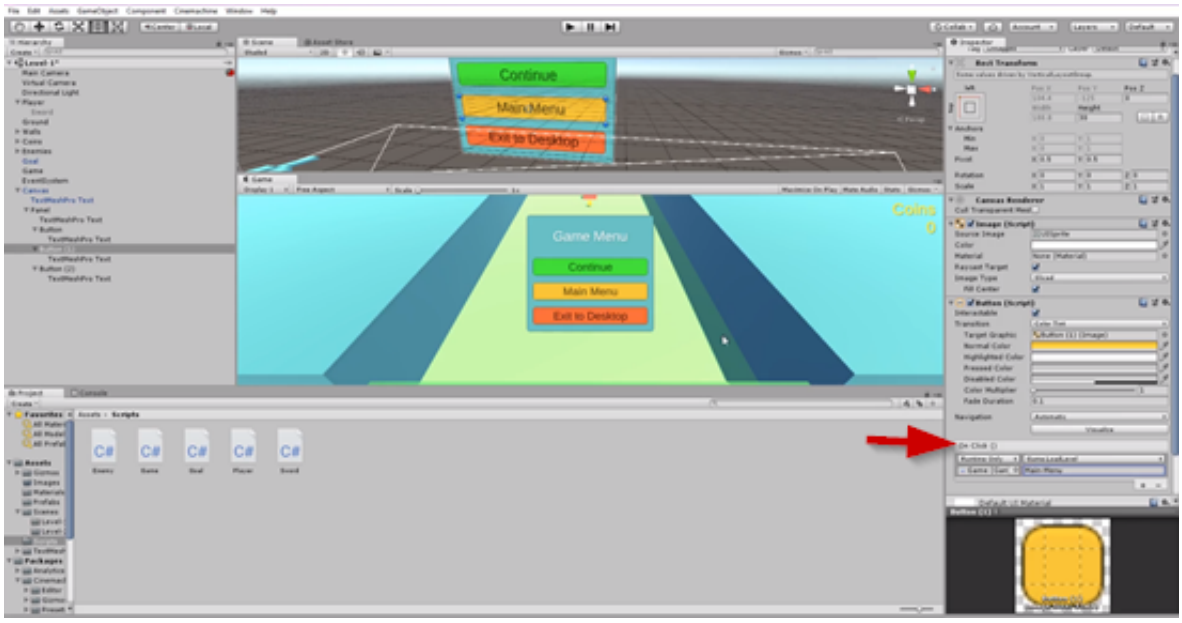
Make sure the **Runtime Only option is selected** and then we need to **drag and drop the Panel into the None(Object field).**



Now if you select the **No Function drop down menu you can choose from a list of components that are on the panel,** and since we are just going to disable this panel we can select the active state to be false by toggling the check box.

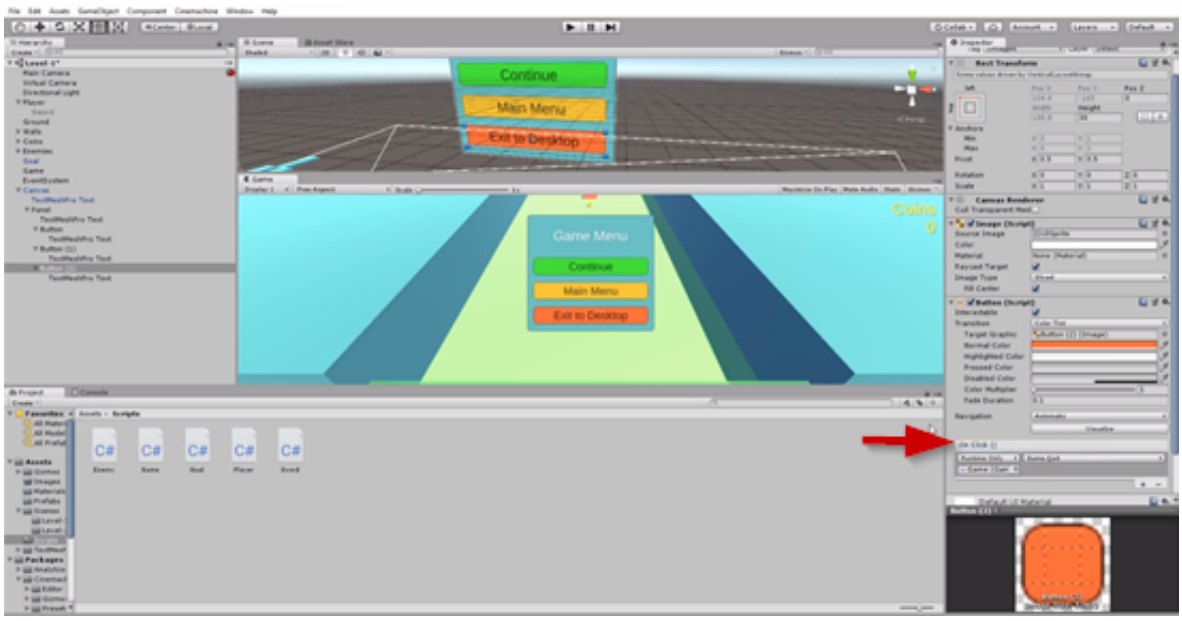


Now for the B**utton(1)** which is the **Main Menu button** we can **set up the On Click event**, and we will use the **Game Manager for this object**, so drag and drop the Game game object and into the field. The **Game.LoadLevel function is the function we will use for this and we can pass in the string "Main-Menu"** so now whenever we go to click on Main Menu it'll go through the load level method it will pass in Main Menu as the level to load.



Now for the third button which is the **Exit to Desktop button** we will go through the **Game object** as well for this one because the Quit method is in the Game script.

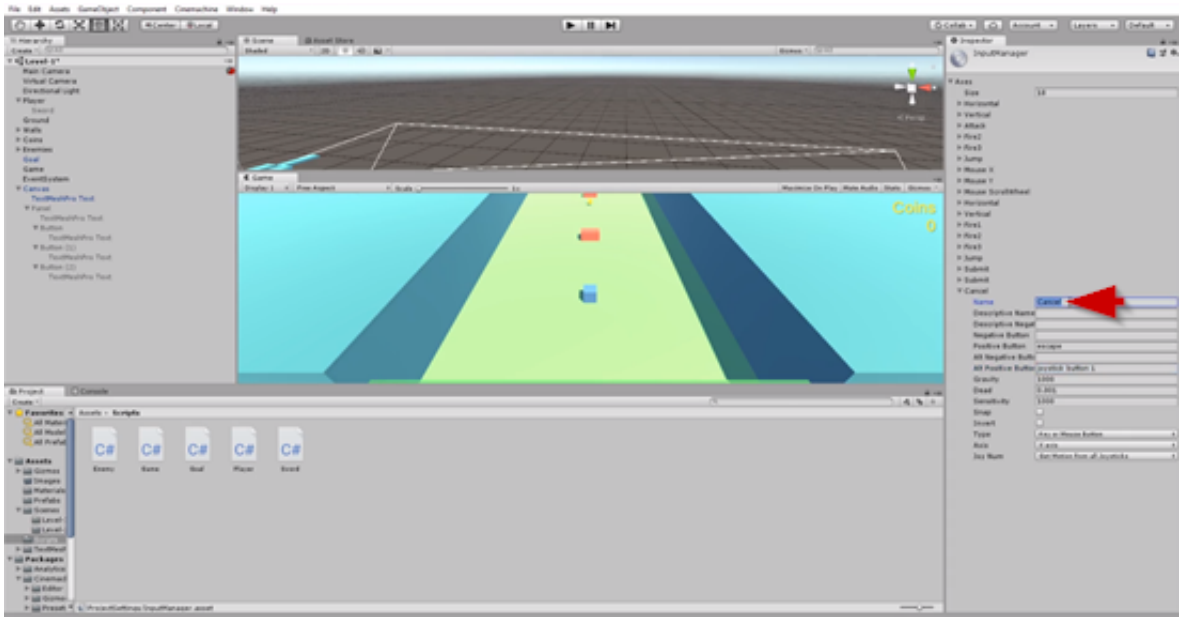
So add the **Game object by dragging and dropping it into the object field, and select Game.Quit for the function on the On click event.**



The last thing we need to do to get the in-game menu to work is the ability to open it. **The entire panel by default is disabled.**

So what we want to do is setup the menu to enable when the **Escape key is pressed.**

Select **Edit>Project Settings>Input**, we can use the Cancel option for this.



We need to handle this all through code, and we can **create a new script called “GameMenu” open this script up in the code editor.**

See the code below and follow along:

```
// Update is called once per frame
 void Update () {
  if (Input.GetButtonDown("Cancel"))
        {
```



```
            gameobject.SetActive(!gameObject.activeSelf);
        }
 }
```

Save this script and navigate back to the Unity editor.

Attach the GameMenu script to the Panel game object.

You will see the issue we're gonna run into, we are not going to be able to enable it whenever it's disabled. If you hit the escape key, nothing happens. This is because the Panel by default is disabled.

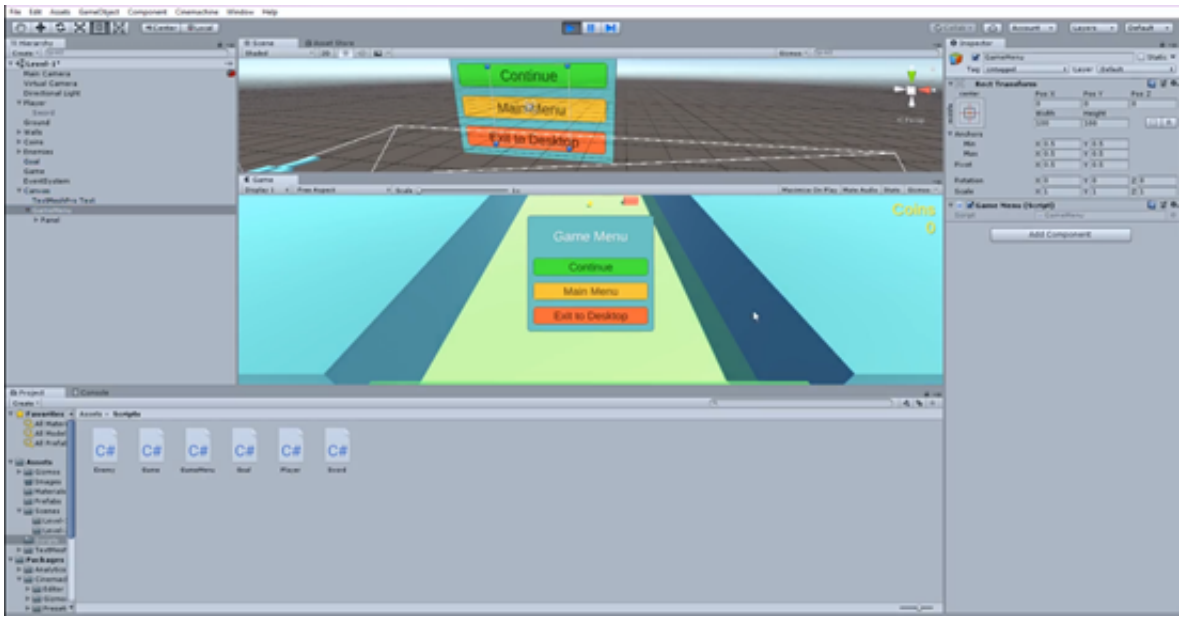
So what we need to do is put the Game script onto an object that will not be enabled and disabled. The way we can do this is create an empty game object in the canvas and call it "Game Menu" and put Panel directly inside Game Menu. Now remove the Game script component from the Panel and attach the Game script to the Game Menu object.

Open the GameMenu script up in the code editor, see the code below and follow along:

```
private GameObject menu;
 // Use this for initialization
 void Start () {
        menu = transform.GetChild(0).gameObject;
    }

 // Update is called once per frame
 void Update () {
  if (Input.GetButtonDown("Cancel"))
        {
            menu.SetActive(!menu.activeSelf);
        }
 }
```

Save the script and navigate back to the Unity editor.

Hit the **Play button** and we will be able to disable and enable the menu.



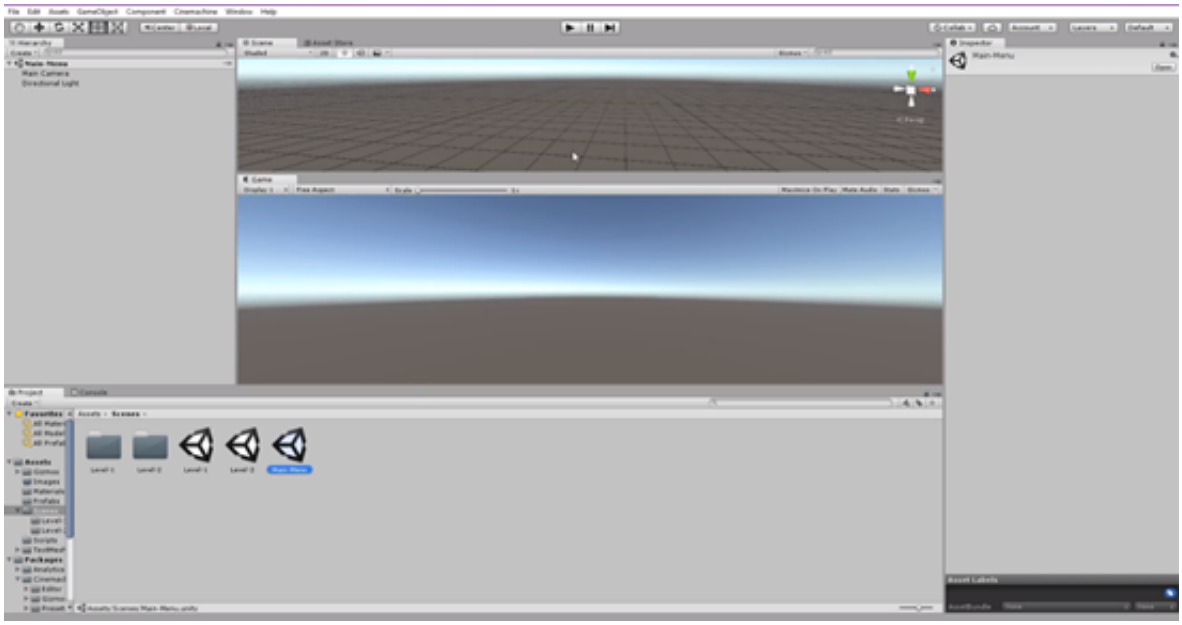
In the next lesson we will setup the Main Menu button toe work properly.

In this lesson we will create the Main Menu scene. This scene will be a separate scene, and from there we will be able to click Play and its going to load the first scene. We will also be able to click Exit or Quit.

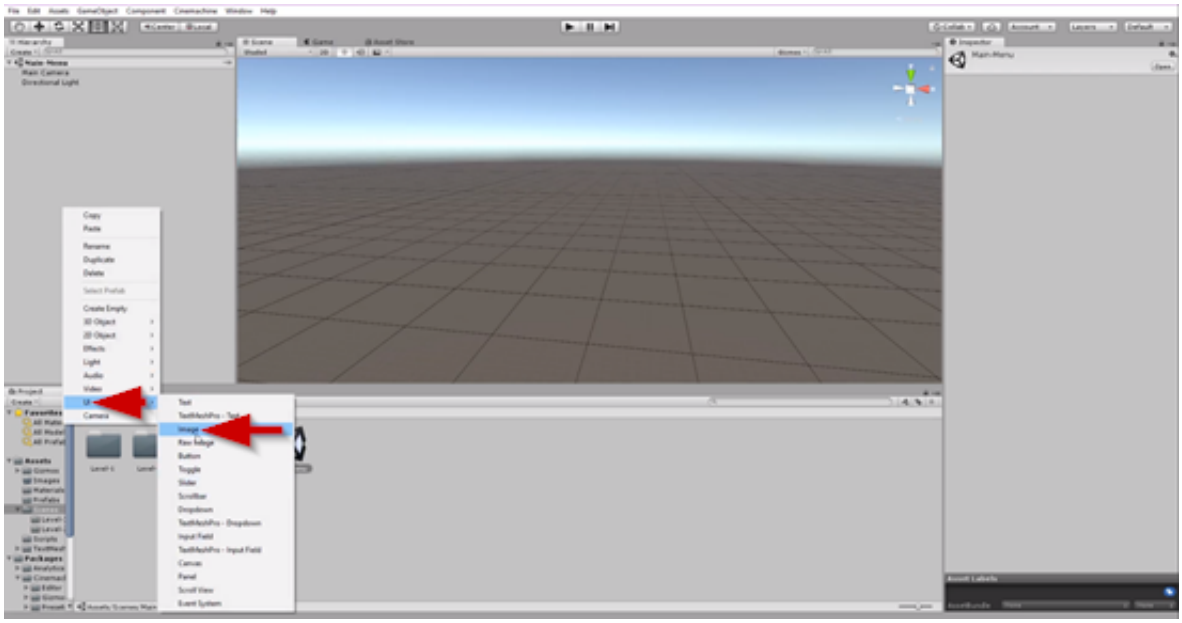
Navigate to the Scenes folder and create a new scene and name it **“Main-Menu”** because this is what the pause/in-game menu is looking for.

Make sure you **save the scene you are currently in.**

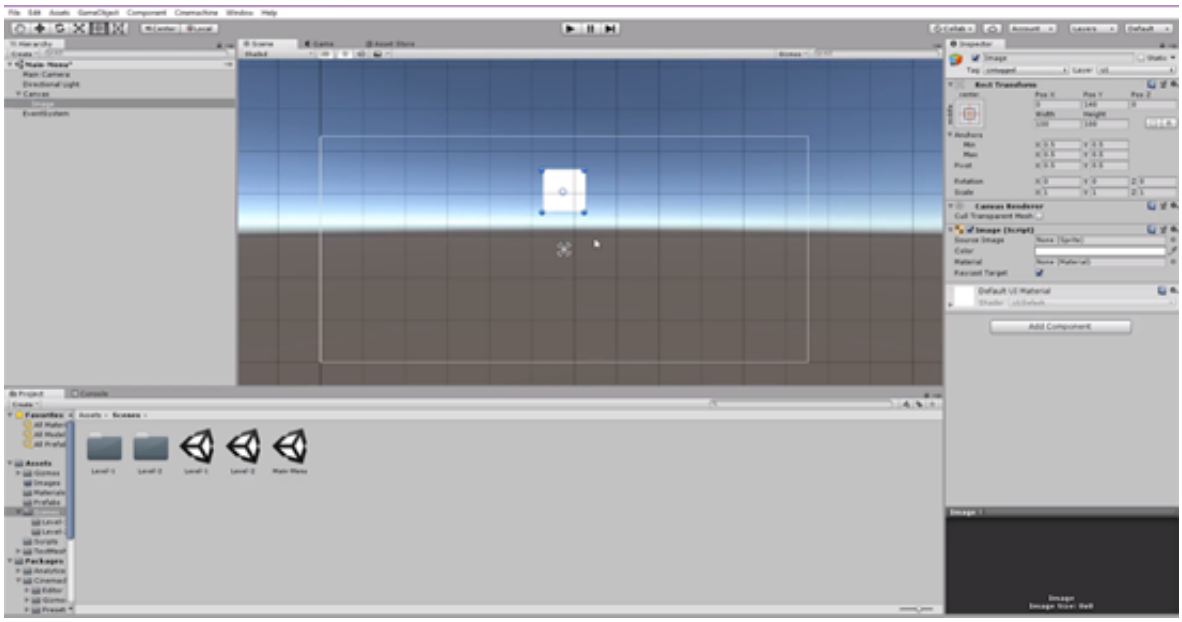
Open up the Main-Menu scene, and you should see this:



Right click in the Hierarchy and **Create a UI>Image.**

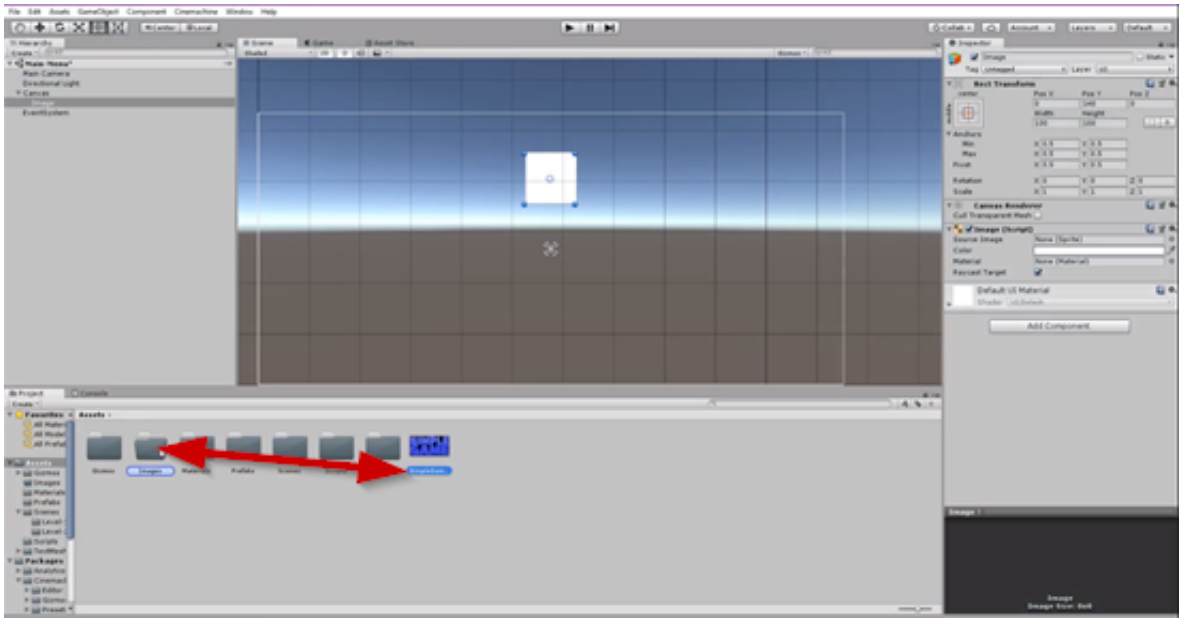


Toggle the 2D mode back on and what we are going to do is add a little logo up top.

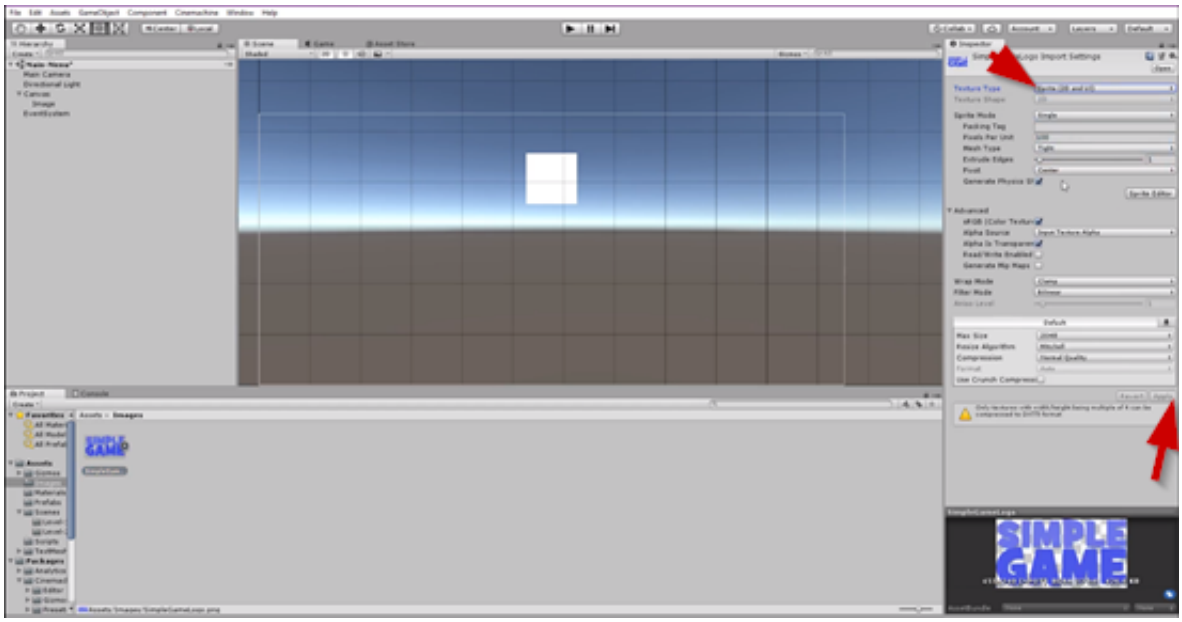


We will use the Simple Game Logo from the course assets for the image.

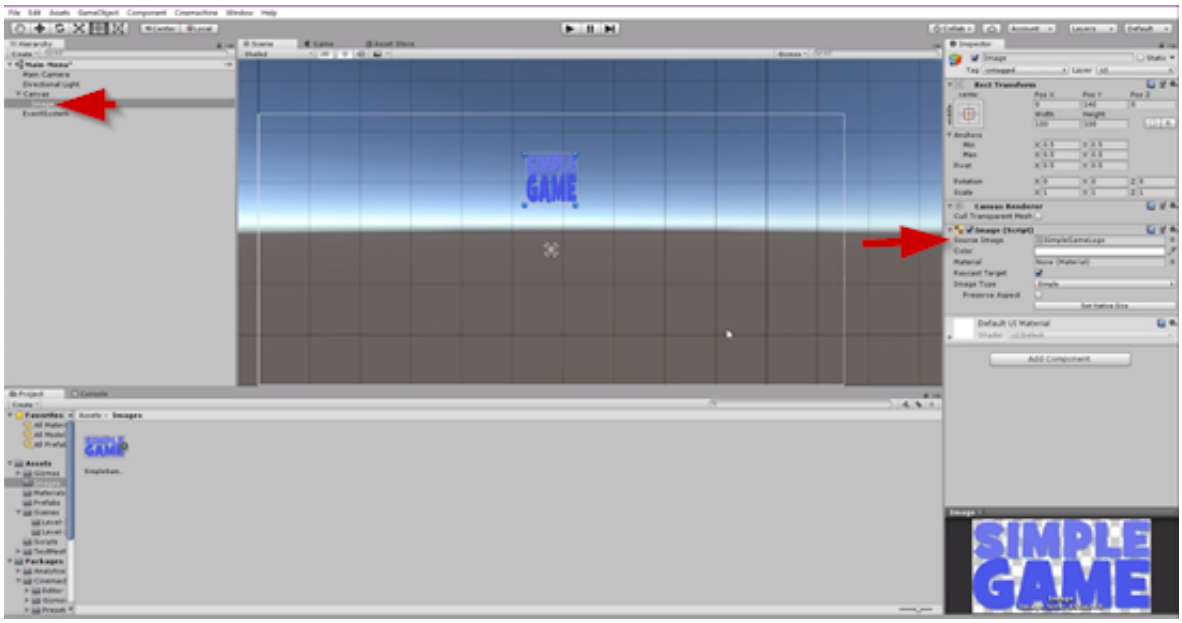
Add this image to the Images folder in the Assets folder.



Now inside the images folder **select the logo and set it to be Sprite for the Texture Type and then hit the Apply button.**



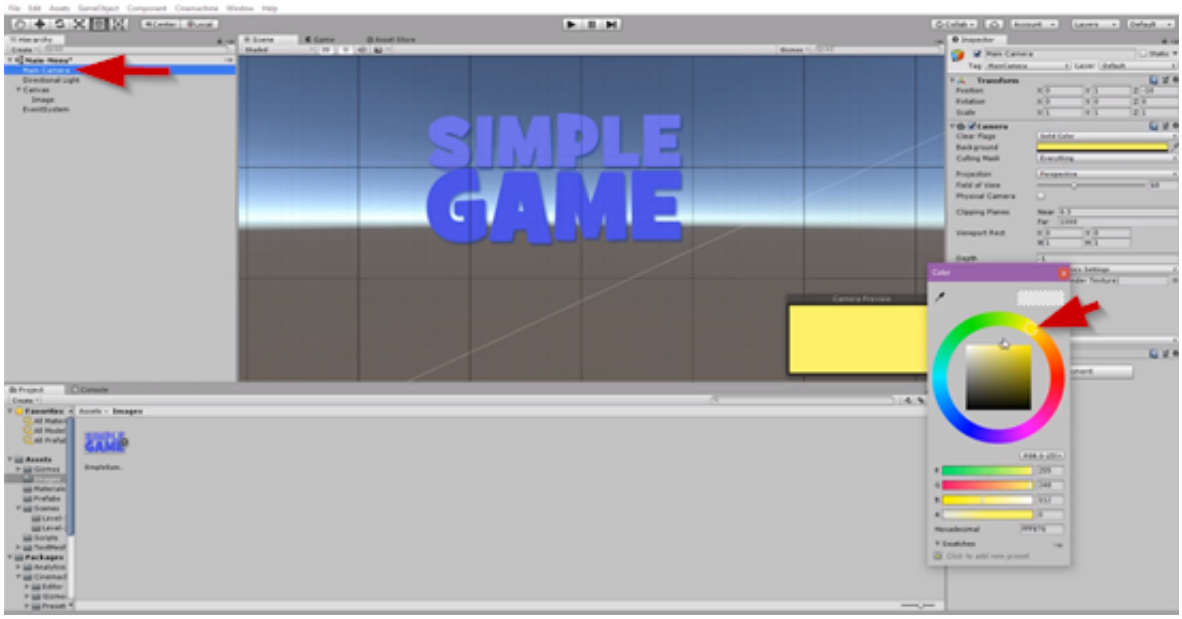
Now **drag the logo onto the Source Image field for the Image(Script) component in the Inspector.**



Toggle the **Preserve Aspect option.**

Center the Logo in the middle of the screen.

Select the Main Camera in the Hierarchy and set the **Clear Flags option to be Solid Color.** I will choose a cool like yellow color for the background.



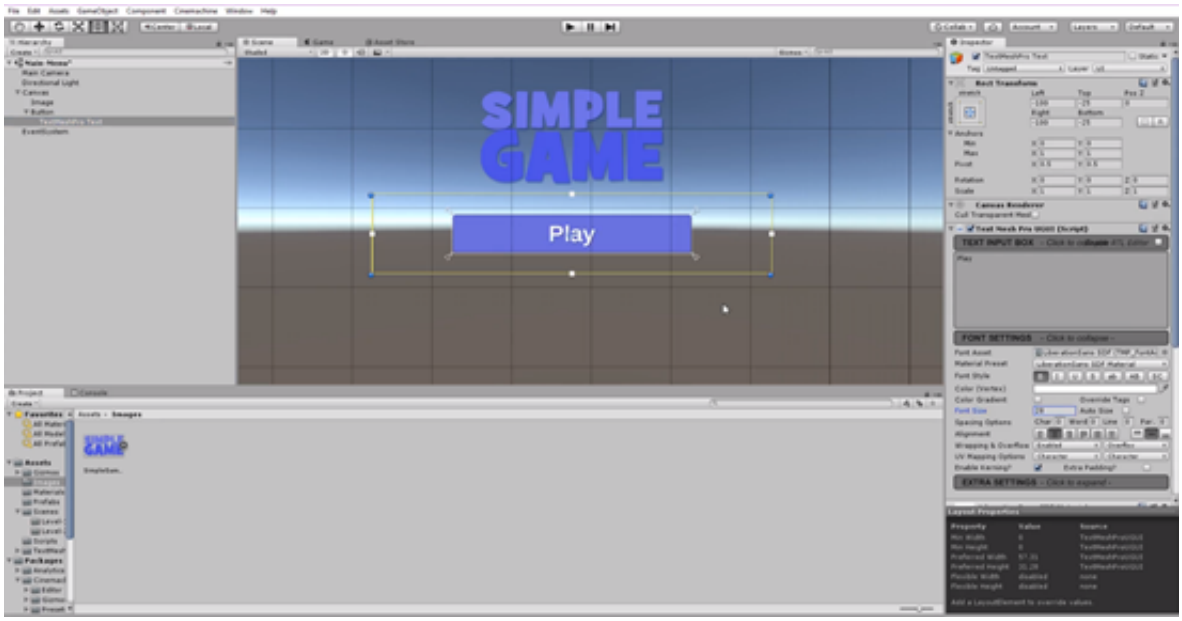
We will add a couple of buttons. **Select the Canvas game object in the Hierarchy and right click> UI>Button.**

Again, **delete the Text component** it has because we will use the Text Mesh Pro-text component for the text on the button.

So **create a Text Mesh Pro-Text object as a child to the Button.**

In the Text Input Box type **“Play.”**

You can make the button a blue color and leave the font color at white if you like. **Change the font size to 28**, and make it **Bold.**



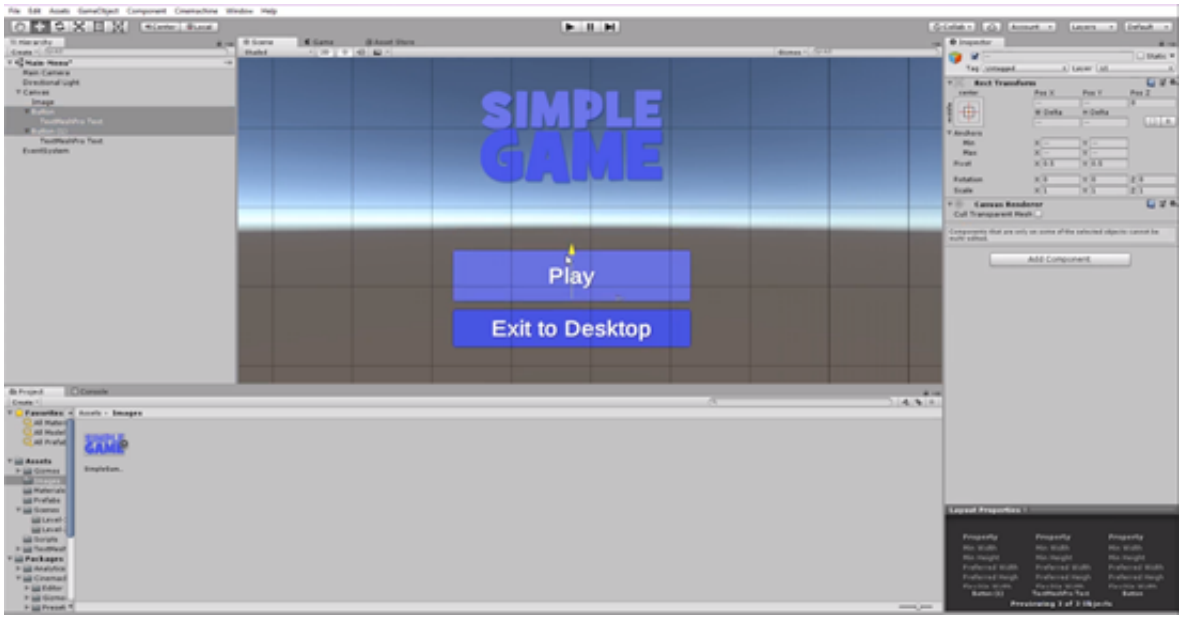
Now we can **duplicate the Play button** and this will be the **Exit to Desktop button.**

Reposition the Exit to Desktop button below the Play button.

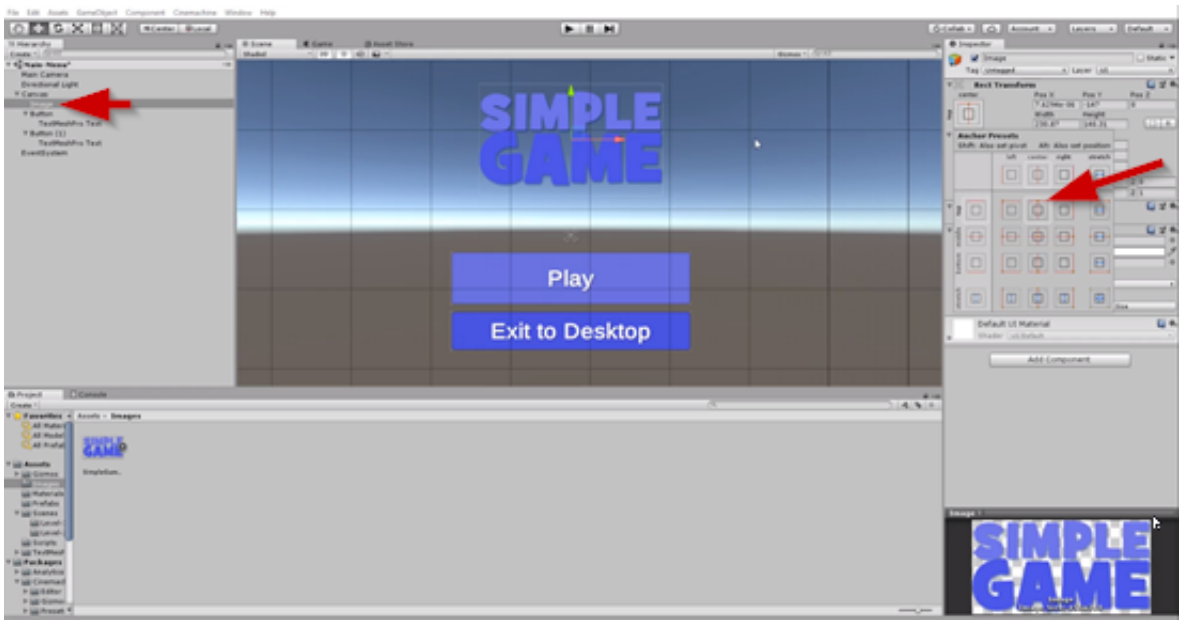
The Game view should now look like this:

You can also change the size of the Play button so its the more dominating button.

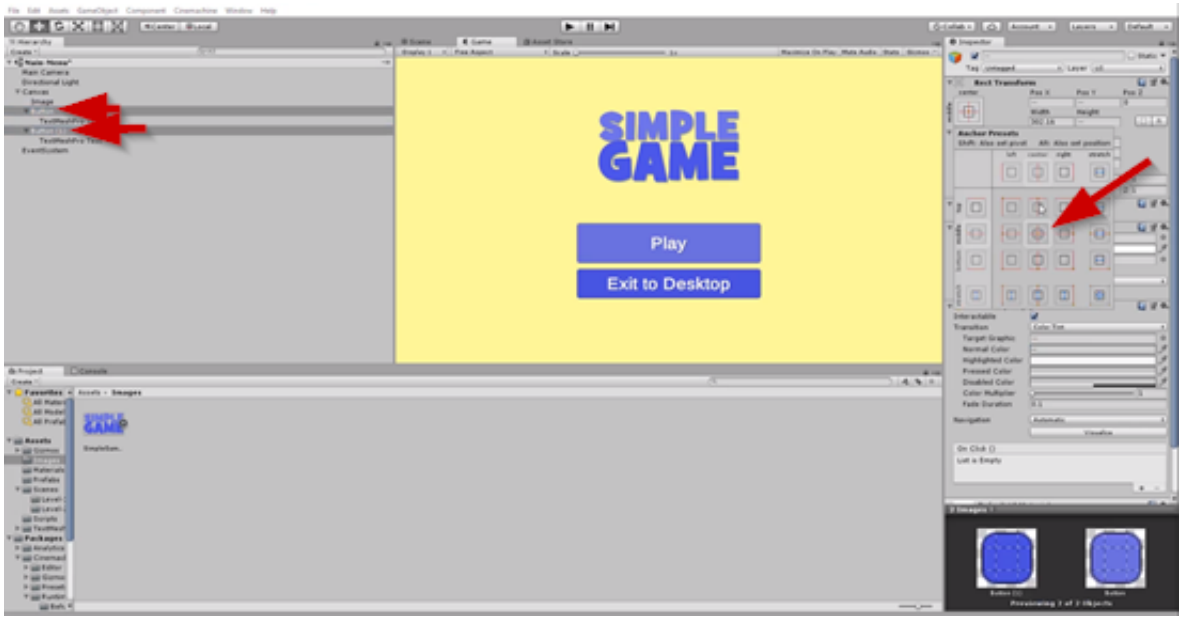
Then reposition both buttons a little lower.



Make sure to **anchor** the logo image to the **top and center.**



Also **anchor** both the buttons to the **top center.**



Keep in mind you can design your menu however you like.

Now what we want to do is have Play load the first level when its clicked on.

What we need to do now is **create an empty game object and name it “Game” attach the Game script to this object.**

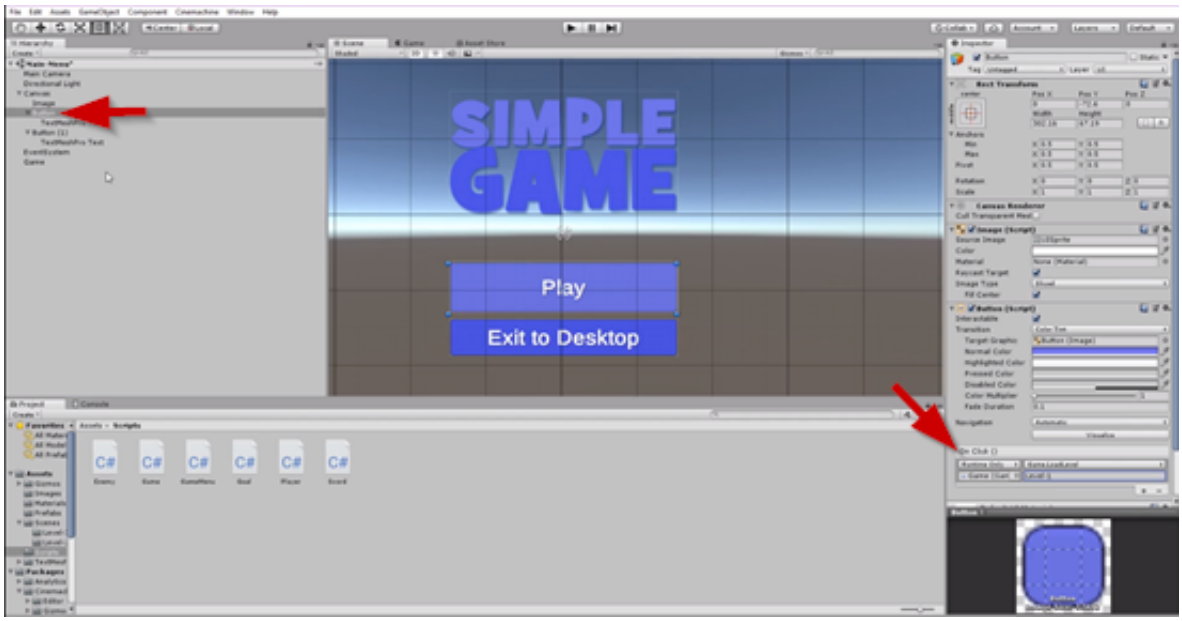
Leave **Level at 0 and do not check the Last Level option.**

No we need to **setup** the **Play** button so it works with an **On Click event.**

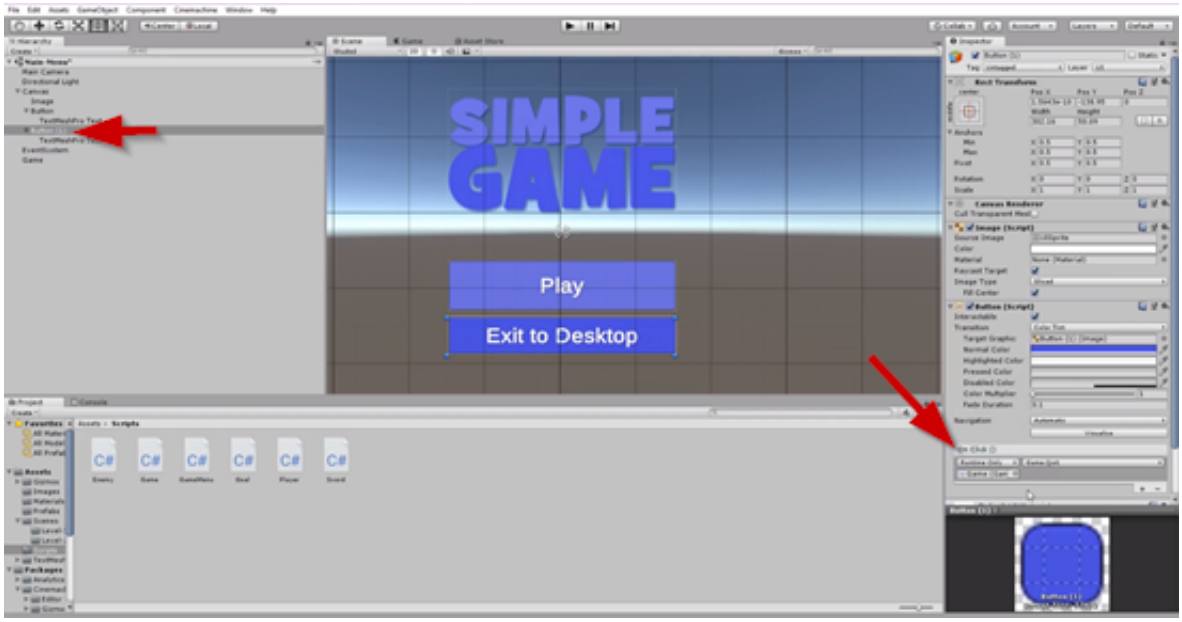
Drag and drop the Game object into the object field on the Play button.

Select the function to be **Game.LoadLevel.**

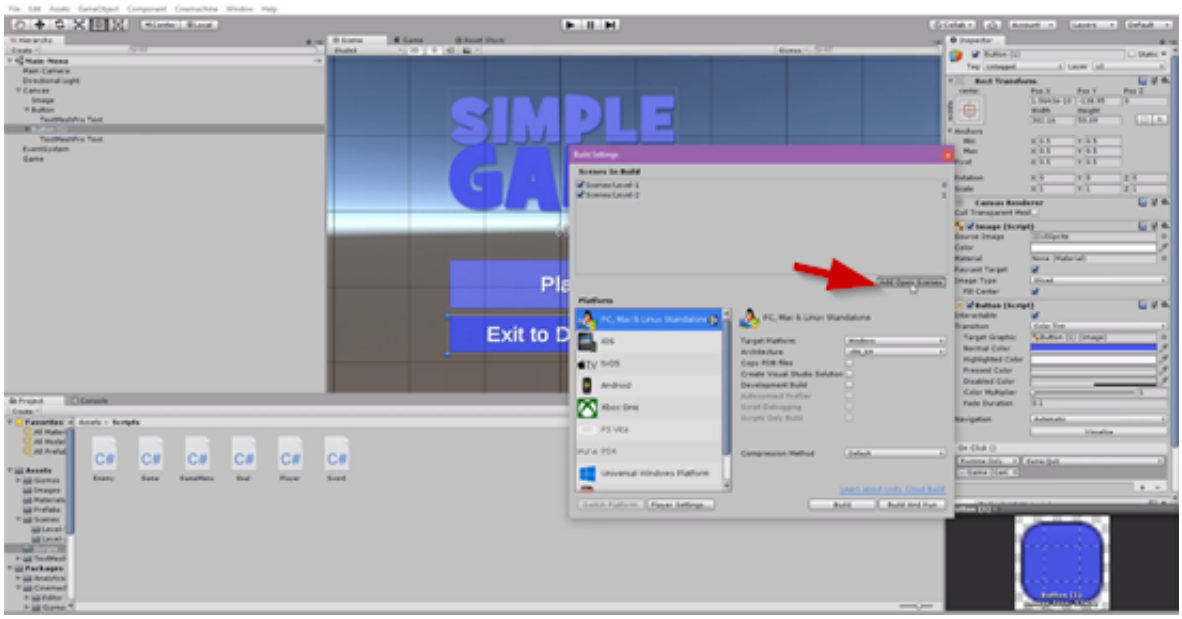
Then type **“Level-1”** in the field.



The Exit to Desktop button will be setup the same way except the function we need for the On Click event will be **Game.Quit.**



Now go to **Build Settings and add the Main-Menu** to the list here.

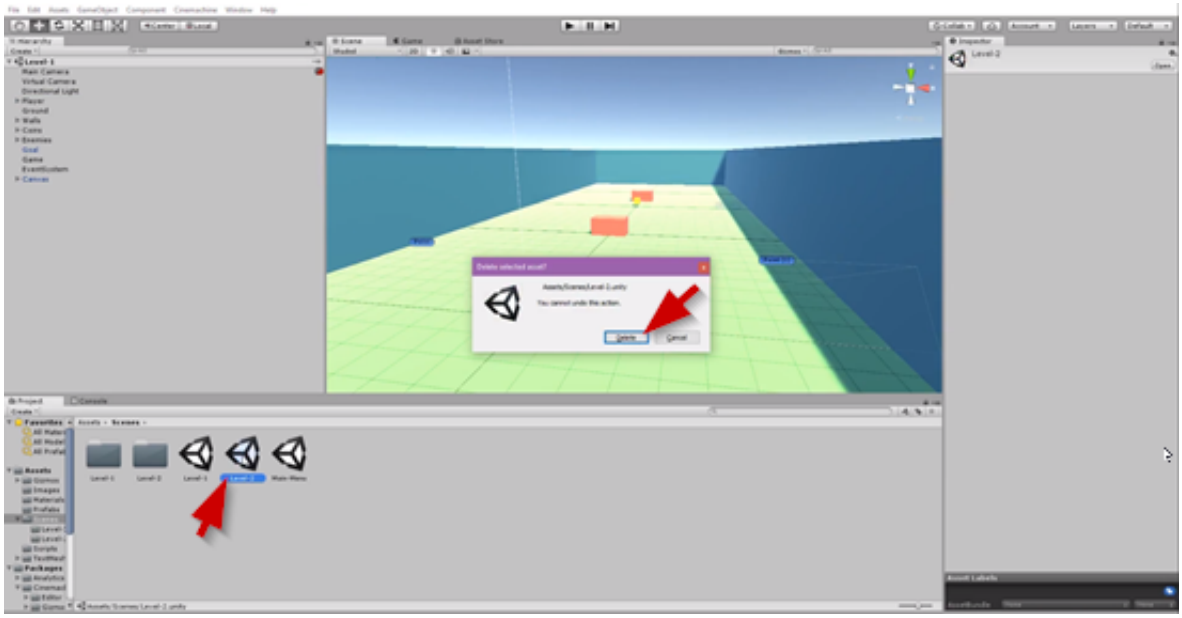


Hit the **Add Open Scenes button** to add the Main-Menu scene to the build.

Now if you **hit the Play button to test out the changes** you will see that all the menus and buttons are functioning properly.

In this lesson we will start setting up some more levels to add to our simple game.

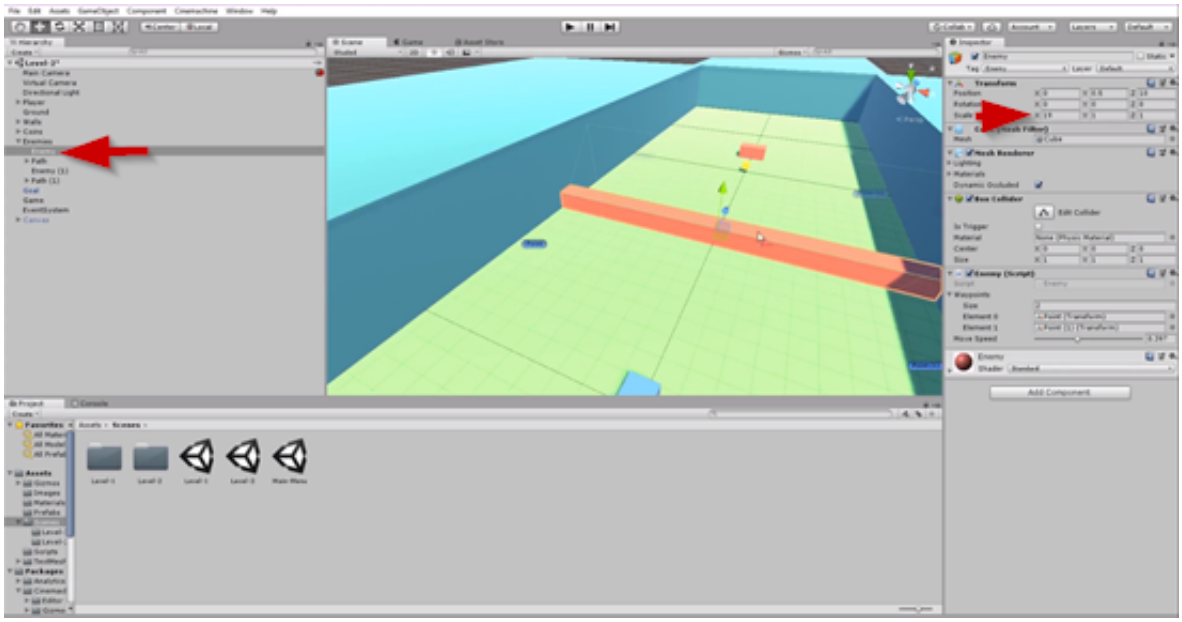
Start off by **deleting Level-2 entirely from the Scenes folder.**



We want to make sure that all the changes we had previously made to **Level-1** are contained in all the new levels we create.

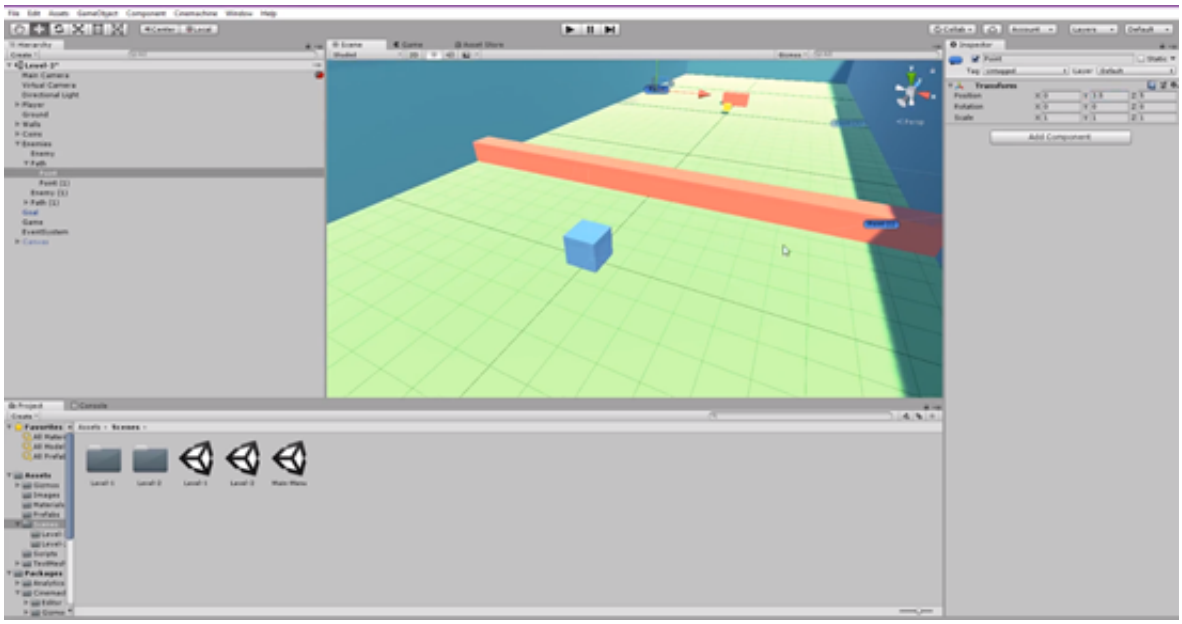
Duplicate Level-1, and **open** up **Level-2.**

We can make a change to the first enemy where we **scale it up** so that it takes up the whole width of **19.**

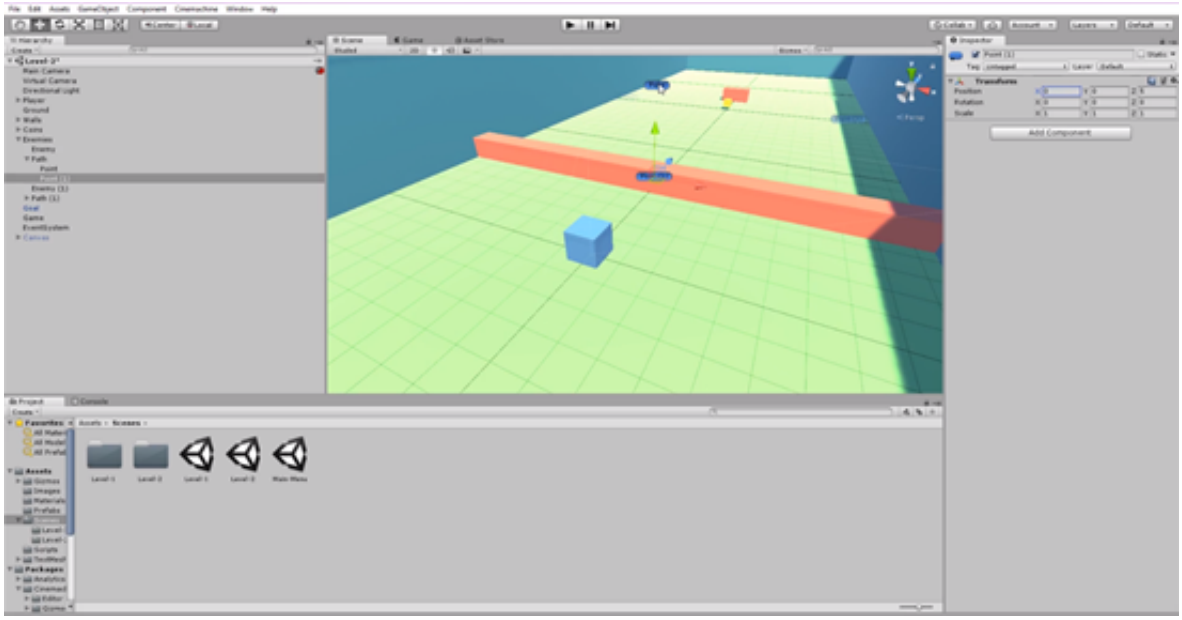


Reposition the enemy to be at **5** on the **Z axis.**

Change the path that this enemy is following. **Select the Path and then Point adjust the position so that the X Axis is at 0, the Y Axis is at 3.5, and the Z axis is still at 5.**

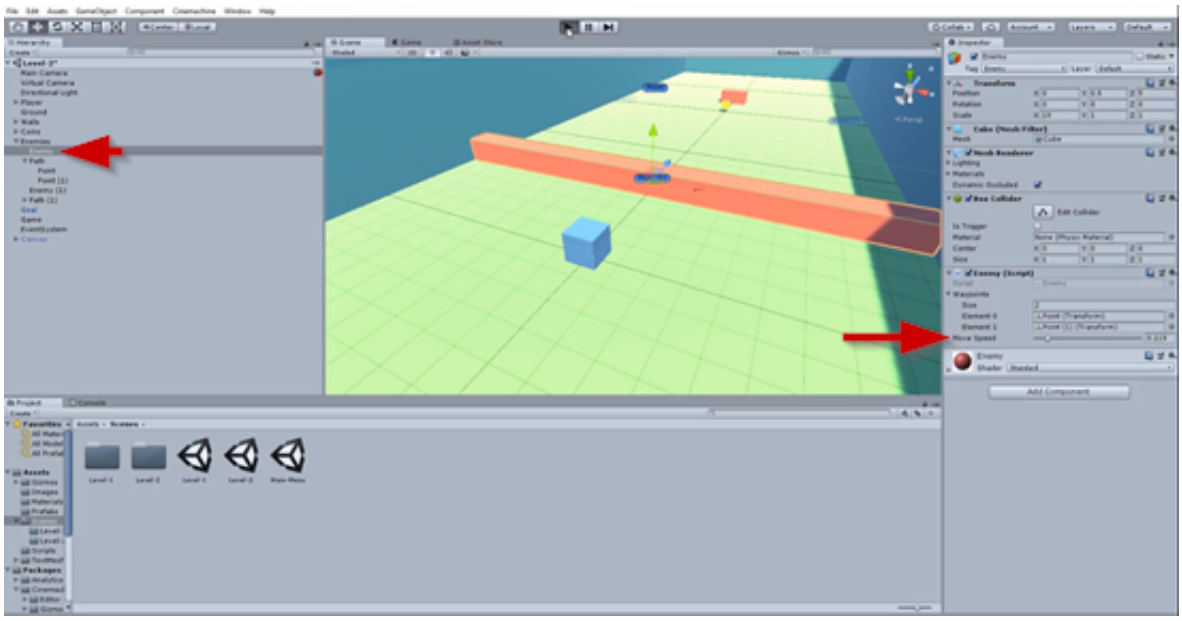


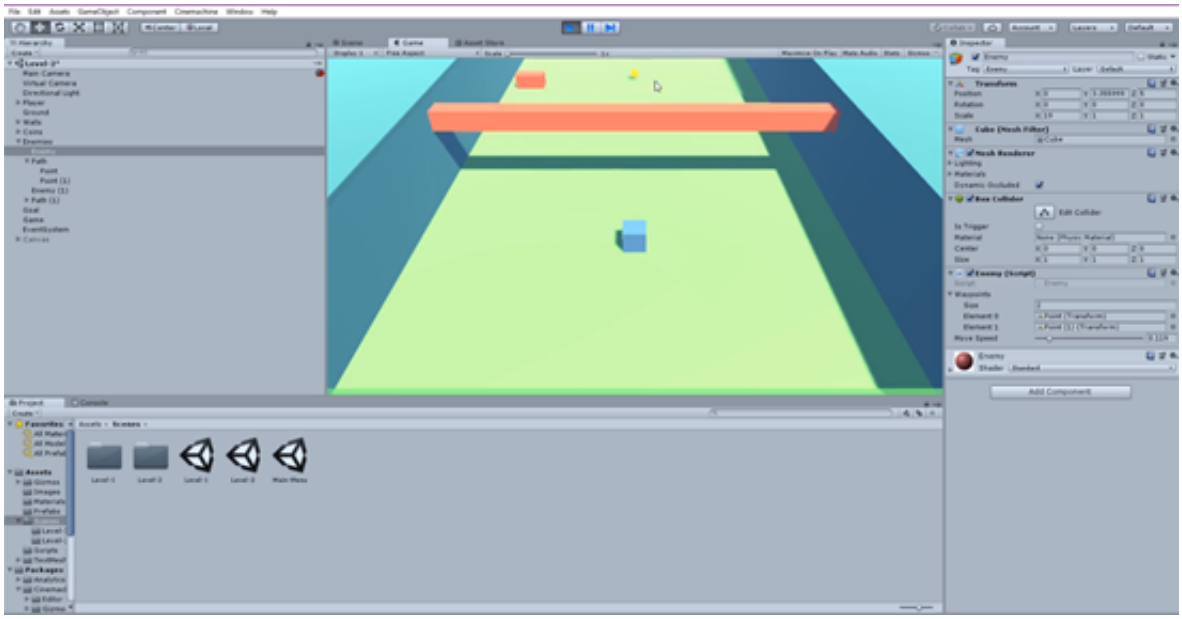
Select Point(1) and adjust the **position** so that the **X Axis is at 0, the Y Axis is at 0, and the Z Axis is still at 5.**



Now if you **hit the Play button** and check out the changes.

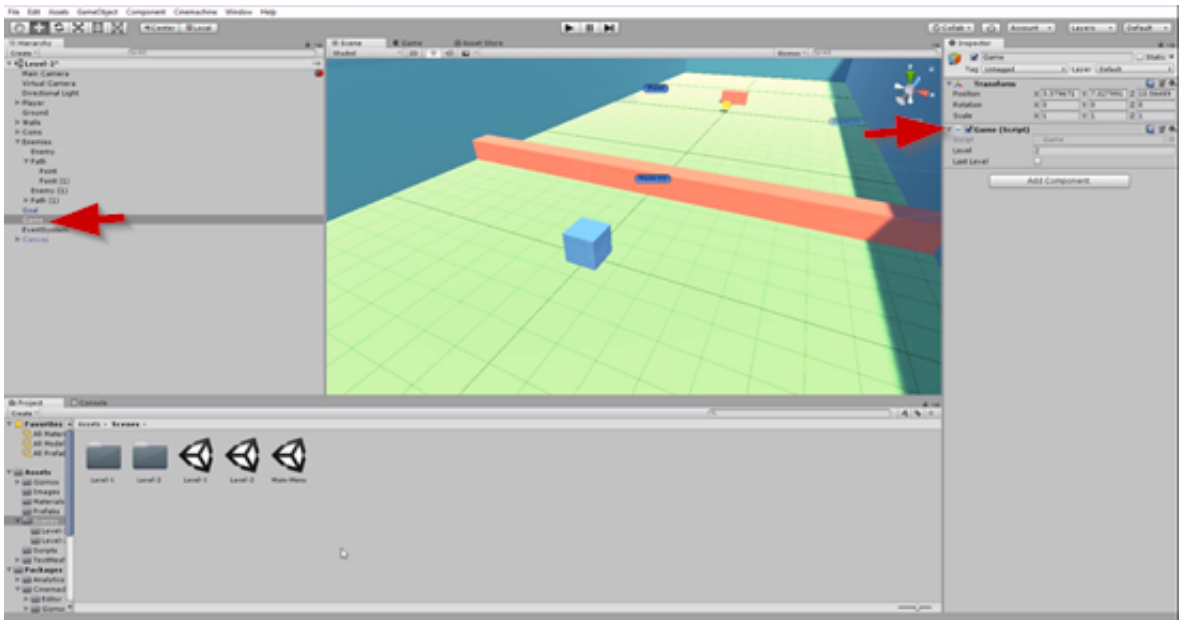
The enemy is moving very fast, and it will be pretty impossible to get past so we can **adjust the speed of it by reducing it to 0.119.**





Now we added some different challenge to this level.

Select the **Game object in the Hierarchy** and in the **Level field type “2”** there and this will not be the last level so **do not toggle that option.**

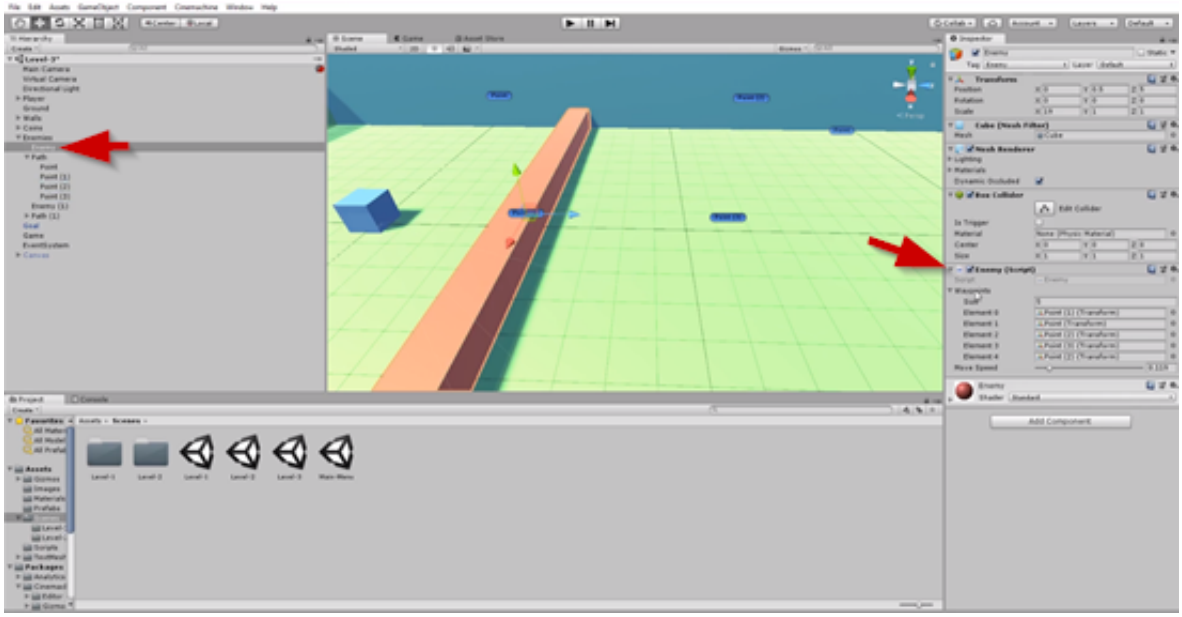


Save the changes made to Level-2 and **duplicate Level-2.**

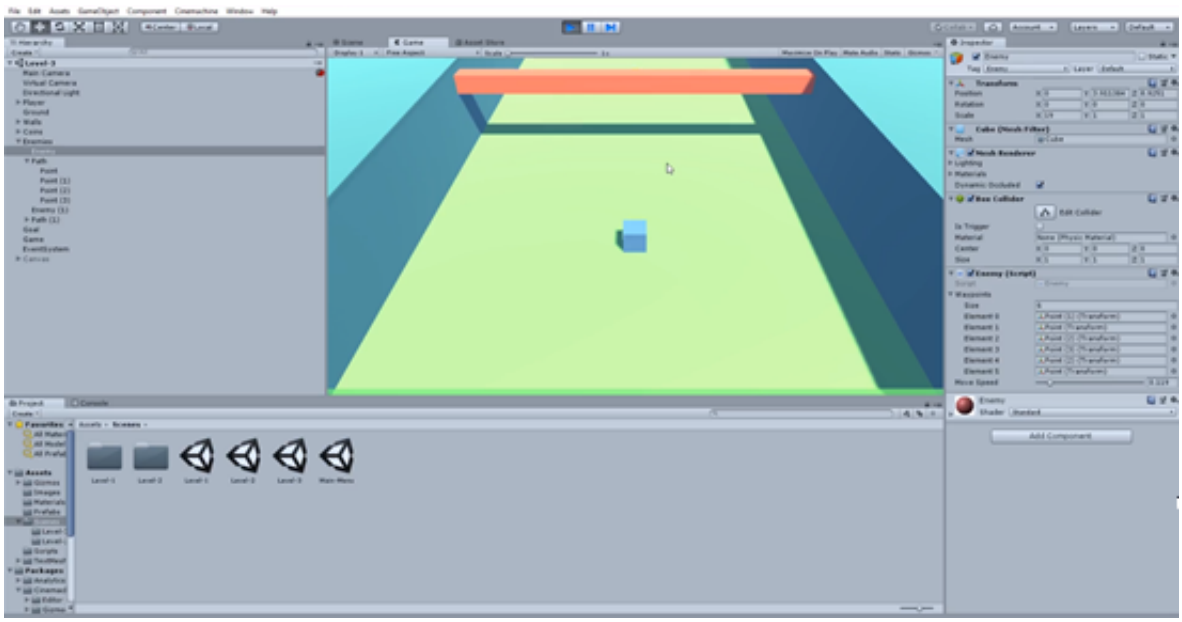
The new level is named **“Level-3”** and what we want to do here in this level is just create something a bit more advanced. We can **adjust the waypoints and speeds of the enemies to make the levels more challenging.**

If you start to **add more waypoints** to the enemy make sure you **add them to the array on the Enemy(script) component.**

This can all be done by dragging and dropping them onto the script component. Just make sure that they are in the correct order.

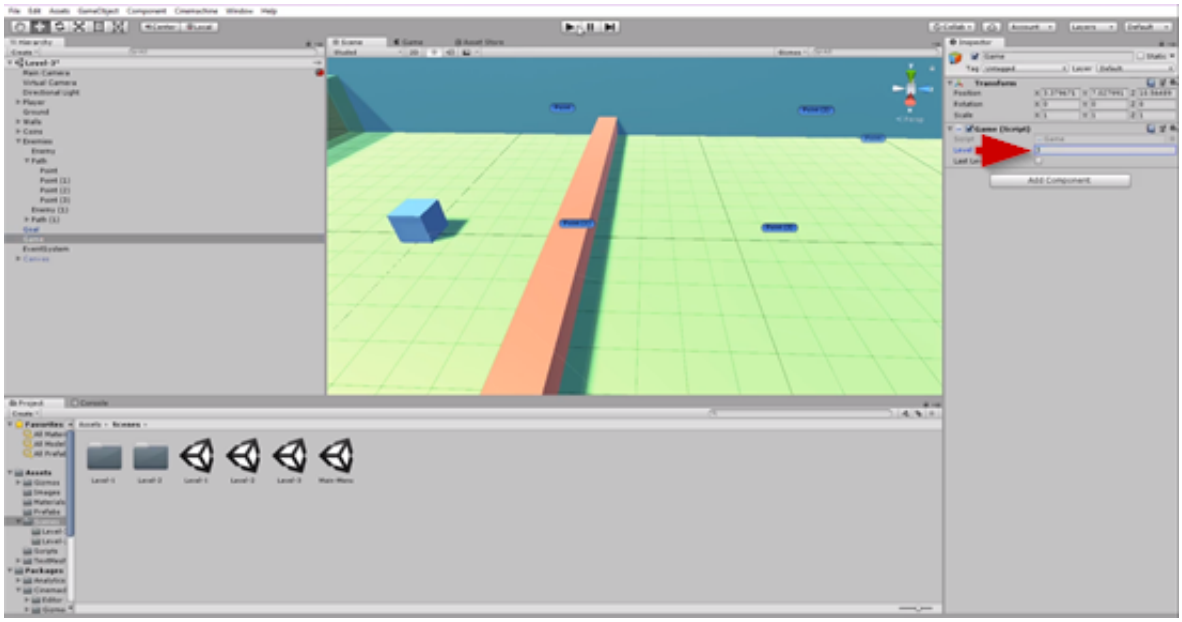


Hit **Play** to test the changes.

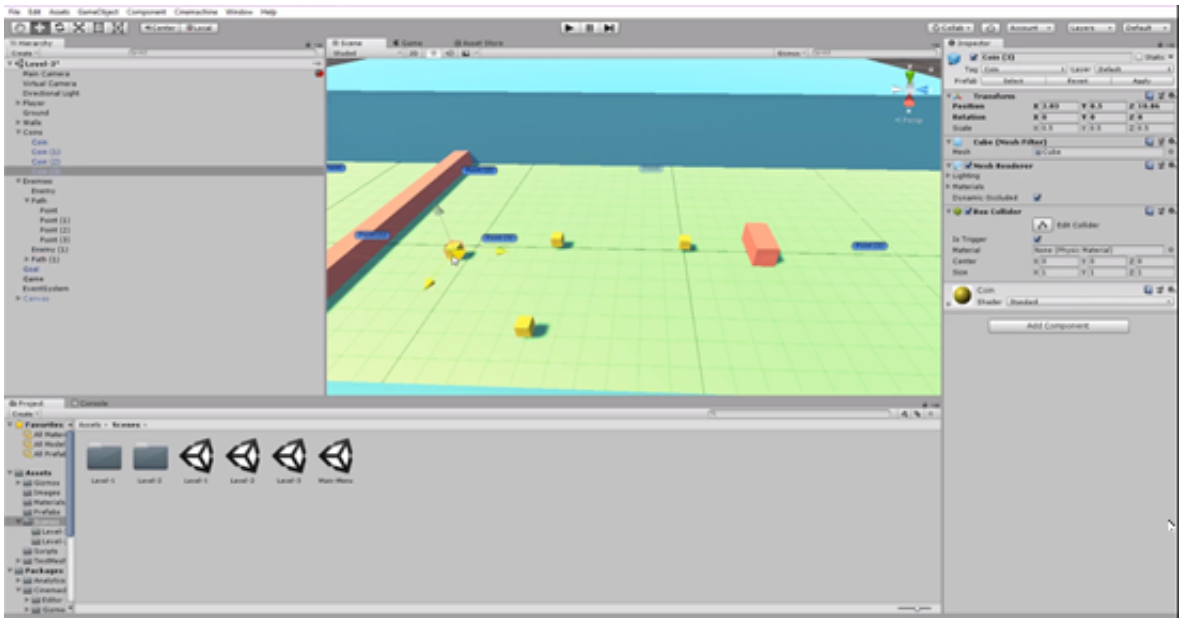


Keep in mind that you design the levels how ever you like.

Make sure to **add 3 to the Level field on the Game(Script) component.**

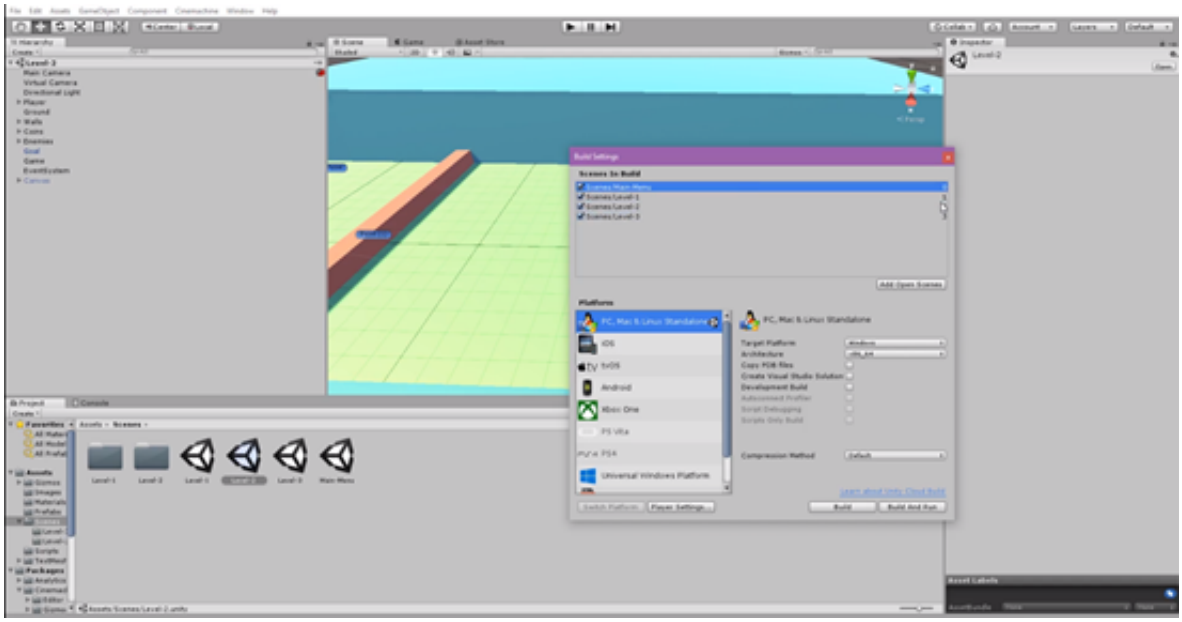


Add some more coins to the third level. We have a system in place that will allow us to ad as many coins as we would like to each level.



Save the third level and try to run through the game from the Main Menu to the third level.

Make sure that all the n**ewly created levels have been added to the Build Settings.**



You now have all the tools needed to create some challenging levels. Make sure you do check the last level option for the very last level you add to the build settings. If you don't you will get an error in the Console window.

Go ahead and add some more levels to the simple game.

The very last thing to do is **adjust the code in the Game script so that the Main Menu is in fact loaded properly.**

Open up the Game script and see the code below to follow along:

```
public void LoadNextLevel()
```



```
    {
        if (!lastLevel)
        {
            string sceneName = "Level-" + nextLevel;
            LoadLevel(sceneName);
        }
        else
        {
            // go to main menu
            LoadLevel("Main-Menu");
        }
    }
```

All of the final version of the six scripts created in this course are posted below, if you are having issues with any of your code feel free to compare your scripts to these final versions.

The entire **Game.cs script** is here:

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

public class Game : MonoBehaviour {
    [SerializeField]
    private int level;
    [SerializeField]
    private bool lastLevel;
    private int nextLevel;


 // Use this for initialization
 void Start ()
    {
        nextLevel = level + 1;
 }

    public void LoadLevel(string levelName)
    {
        SceneManager.LoadScene(levelName);
    }

    public void LoadNextLevel()
    {
        if (!lastLevel)
        {
            string sceneName = "Level-" + nextLevel;
            LoadLevel(sceneName);
        }
        else
        {
            // go to main menu
            LoadLevel("Main-Menu");
```



```
        }
    }

    public void ReloadCurrentLevel()
    {
        LoadLevel("Level-" + level);
    }

    public void Quit()
    {
        Application.Quit();
    }
}
```

The entire **GameMenu.cs script** is here:

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

public class GameMenu : MonoBehaviour {
    private GameObject menu;
 // Use this for initialization
 void Start () {
        menu = transform.GetChild(0).gameObject;
    }

 // Update is called once per frame
 void Update () {
  if (Input.GetButtonDown("Cancel"))
        {
            menu.SetActive(!menu.activeSelf);
        }
 }
}
```

The entire **Sword.cs script** is here:

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

public class Sword : MonoBehaviour {

 private float attackLength = .5f;
 // Update is called once per frame
 void Update () {
        attackLength -= Time.deltaTime;
        if (attackLength <= 0)
```

```
        {
            gameObject.SetActive(false);
            attackLength = .5f;
        }
 }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Enemy"))
        {
            Destroy(other.gameObject);
        }
    }
}
```

The entire **Player.cs script** is here:

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

public class Player : MonoBehaviour
{
    [SerializeField]
    private Rigidbody playerBody;
    [SerializeField]
    private TMPro.TextMeshProUGUI coinText;
    private GameObject sword;
    private Game game;
    private bool jump;
    [SerializeField]
    private int coins;

    private Vector3 inputVector;
 // Use this for initialization
 void Start ()
    {
        sword = transform.GetChild(0).gameObject;
        game = FindObjectOfType<Game>();
        playerBody = GetComponent<Rigidbody>();
 }

 // Update is called once per frame
 void Update ()
    {
        inputVector = new Vector3(Input.GetAxis("Horizontal") * 10f, playerBody.veloc
ity.y, Input.GetAxis("Vertical") * 10f);
        transform.LookAt(transform.position + new Vector3(inputVector.x, 0, inputVect
or.z));
        if (Input.GetButtonDown("Jump"))
        {
            jump = true;
        }
```

```
    if (Input.GetButtonDown("Attack"))
    {
        PerformAttack();
    }
}

private void FixedUpdate()
{
    playerBody.velocity = inputVector;
    if (jump && IsGrounded())
    {
        playerBody.AddForce(Vector3.up * 20f, ForceMode.Impulse);
        jump = false;
    }
}

private void PerformAttack()
{
    if (!sword.activeSelf)
    {
        sword.SetActive(true);
    }
}

bool IsGrounded()
{
    float distance = GetComponent<Collider>().bounds.extents.y + 0.01f;
    Ray ray = new Ray(transform.position, Vector3.down);
    return Physics.Raycast(ray, distance);
}

private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("Enemy"))
    {
        game.ReloadCurrentLevel();
    }
}

private void OnTriggerEnter(Collider other)
{
    switch (other.tag)
    {
        case "Coin":
            coins++;
            coinText.text = string.Format("Coins\n{0}", coins);
            Destroy(other.gameObject);
            break;
        case "Goal":
            other.GetComponent<Goal>().CheckForCompletion(coins);
            break;
        default:
            break;
    }
}
```

```
}
```

The entire **Enemy.cs script** is here:

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

public class Enemy : MonoBehaviour {
    [SerializeField]
    private Transform[] waypoints;
    private Vector3 targetPosition;
    [SerializeField]
    [Range(0,1f)]
    private float moveSpeed;
    private int waypointIndex;
    // Use this for initialization
    void Start ()
    {
        targetPosition = waypoints[0].position;
 }

 // Update is called once per frame
 void Update ()
    {
        transform.position = Vector3.MoveTowards(transform.position, targetPosition,
.5f * moveSpeed);
        if (Vector3.Distance(transform.position, targetPosition) < .25f)
        {
            if (waypointIndex >= waypoints.Length-1)
            {
                waypointIndex = 0;
            }
            else
            {
                waypointIndex++;
            }
            targetPosition = waypoints[waypointIndex].position;
        }
    }
}
```

The entire **Goal.cs script** is here:

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

public class Goal : MonoBehaviour {
    private int requiredCoins;
```

```
    private Game game;
 // Use this for initialization
 void Start () {
        game = FindObjectOfType<Game>();
        requiredCoins = GameObject.FindGameObjectsWithTag("Coin").Length;
 }

    public void CheckForCompletion(int coinCount)
    {
        if (coinCount >= requiredCoins)
        {
            game.LoadNextLevel();
        }
        else
        {
            Debug.LogFormat("You need more coins! You have {0} out of {1}", coinCount
, requiredCoins);
        }
    }
}
```

This is the final lesson of the course and marks the completion of the course.

Congratulations!