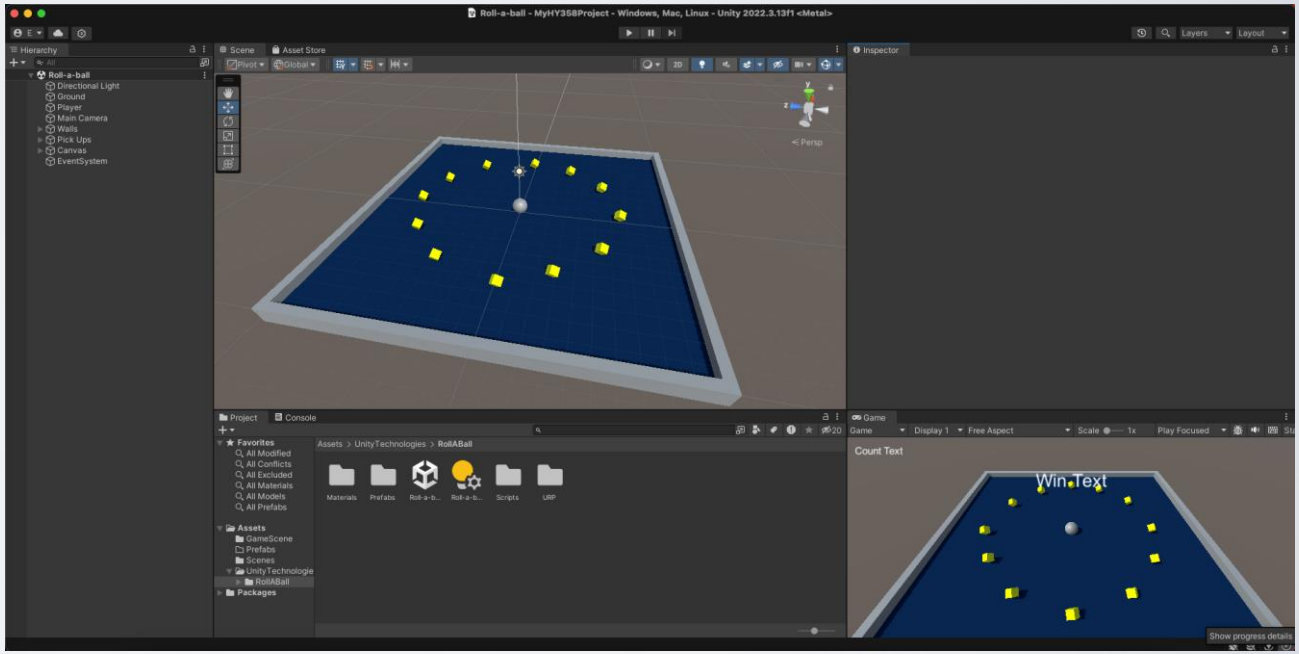
Unity Tutorial

Introduction

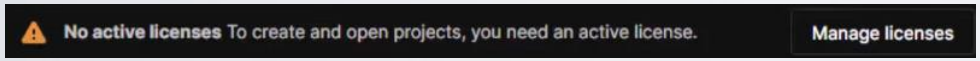
- What is **Unity**?
  - Unity is a popular commercial 2D/3D Game Engine.
  - Widely used by many indie developers as well as huge industries in the gaming sector such as Blizzard, which has developed the successful card game Hearthstone using Unity. (read more about it, here)

- But, why Unity?
  - Free for personal use
  - Cross-platform
  - Easy to learn
  - Lots of resources available
  - Simplifies game development
  - Active forums.

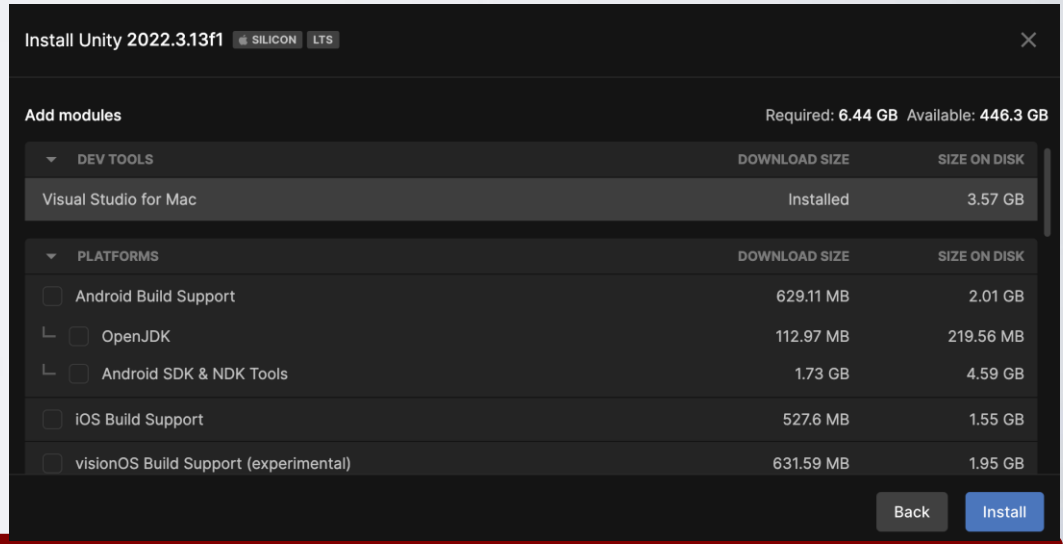


Installation

- Get the latest version of Unity Hub from here and install it following the wizard.
- After installation, Unity Hub will recommend you the latest Unity version to install.
- During installation:
  - You can choose to install several other components such as Visual Studio Community 2022
  - Even build support for other platforms.
- After installation is complete, create an account and sign in, in order to be able to use the asset store and download a lot of resources for free.
- In case you see this:

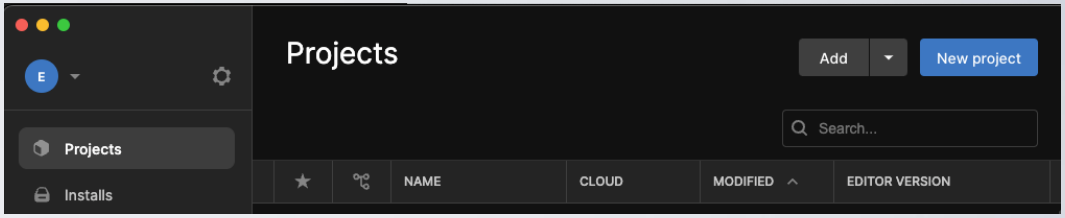


- Click "Manage licenses" and then click "Get a free personal license".



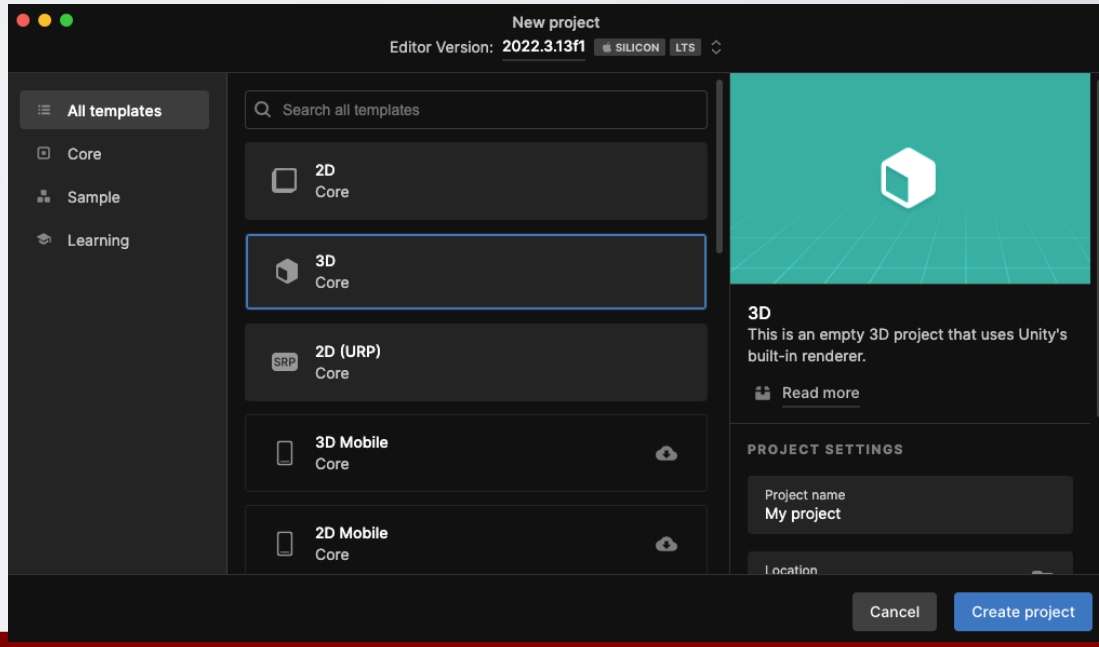
Basics

- Creating an Empty Project.



- Open Unity Hub and press New Project on the upper right corner.

- Choose an appropriate name for your Project and a location to be saved. Make sure 3D Core is selected if you wish to create a 3D game.

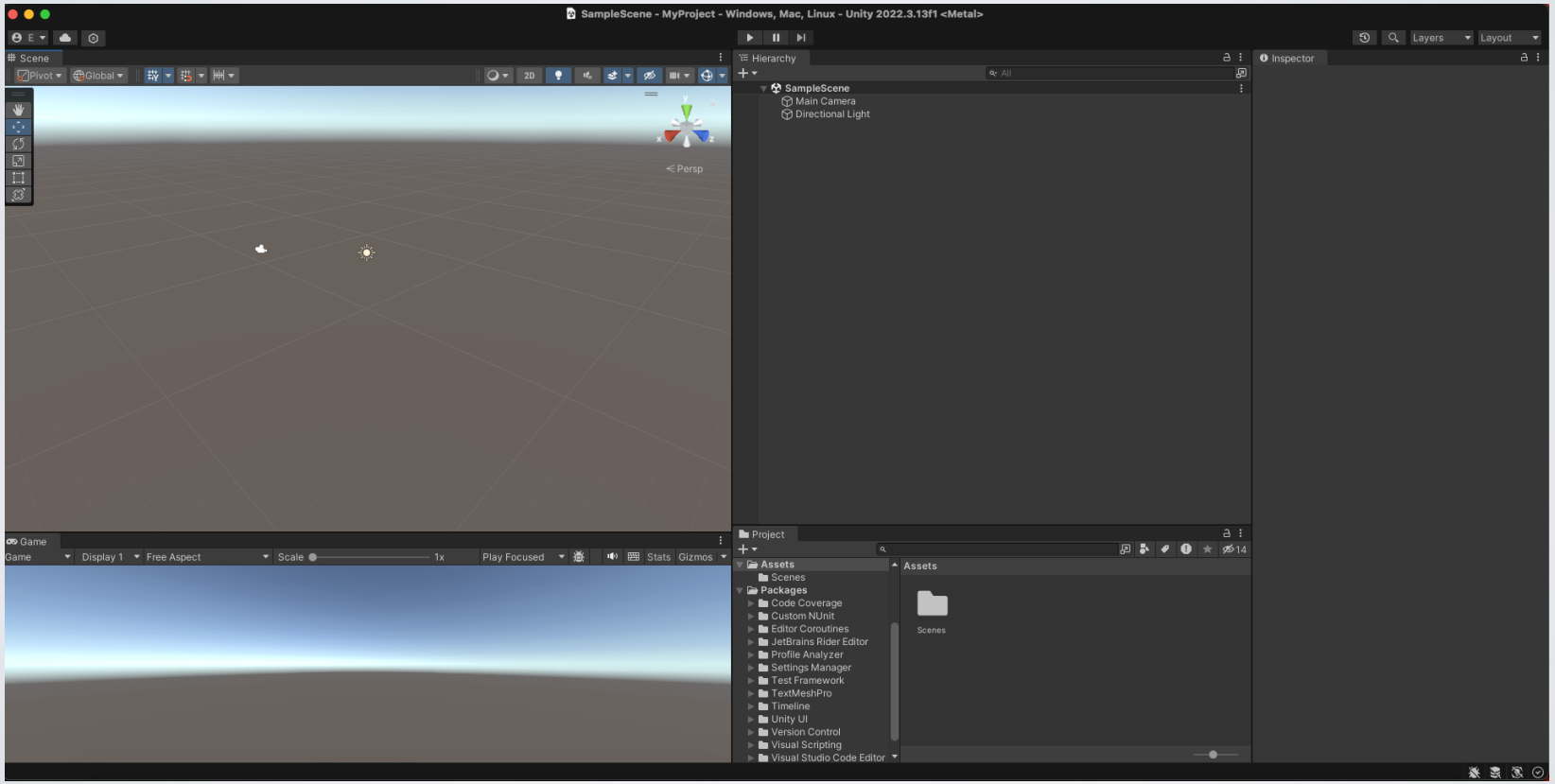


Afterwards click “Create Project” and you are ready to start.

Basics

- Getting to know the editor – Arranging tabs

- In Unity you can re-arrange all tabs such as Scene, Game view and Inspector to your own preference and liking.
- An easy to use and learn order is displayed on the image below.



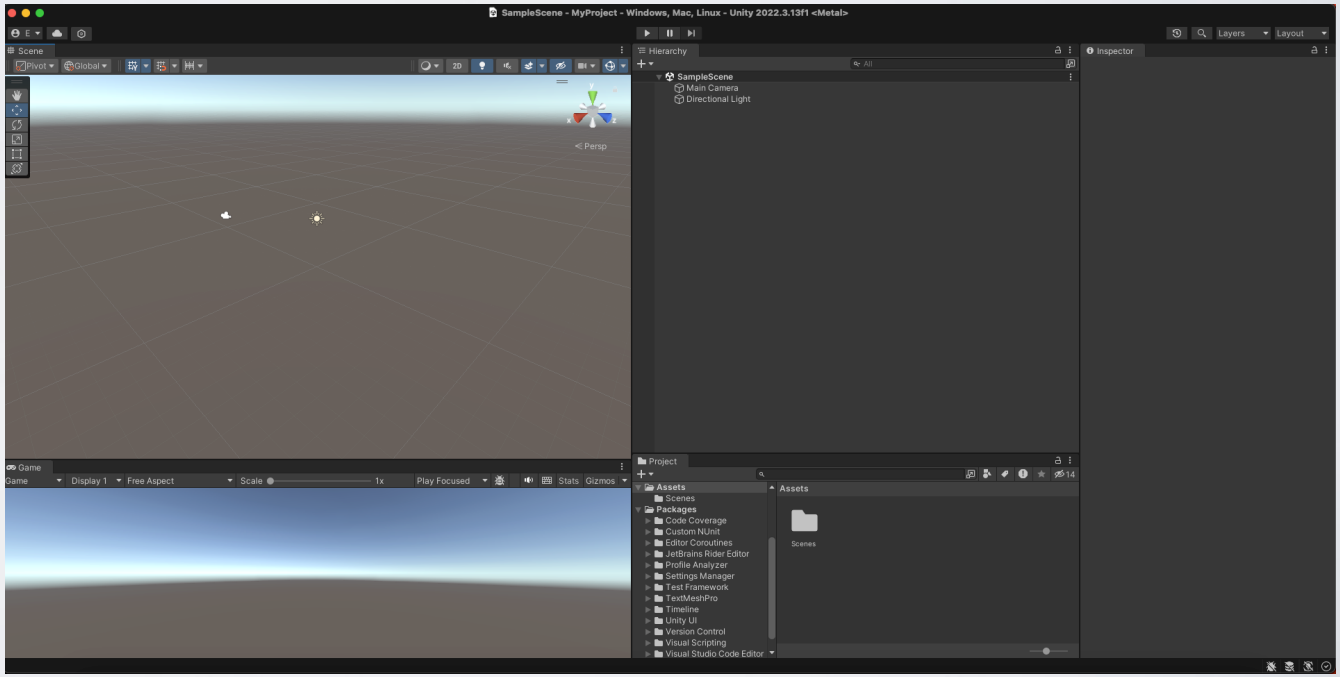
Basics

- Getting to know the editor – Arranging tabs

- On the left we have our scene view.
- Below, we have our game view.
- Right next is our Hierarchy which

contains our Scene components, cameras, game objects etc.

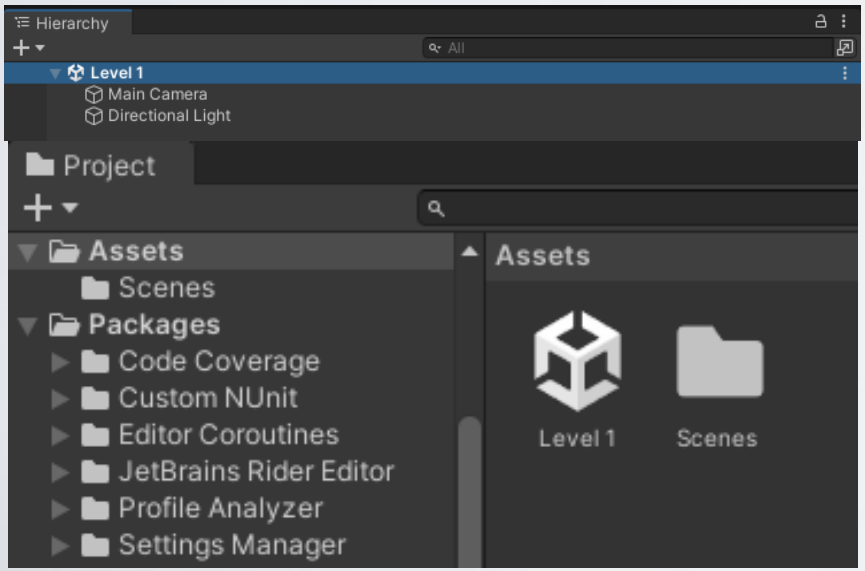
- Below is our Project tab which contains all project folders, files, models, scripts etc.
- Last, on the far right is our inspector which shows different attributes/details for a selected component.

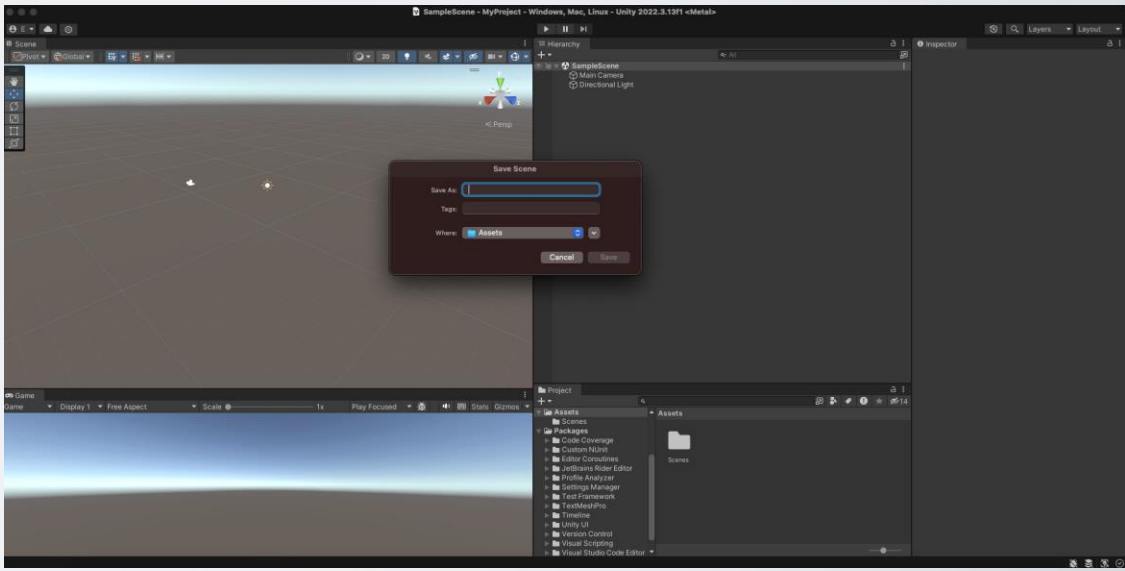


Basics

- Getting to know the editor – Saving your Scene

- As you can see under Hierarchy tab is our scene SampleScene and contains only two default components, a Camera and a Directional Light.
- You can choose to name this scene by right clicking on SampleScene and selecting “Save Scene As”, a pop up will appear in which you can select an appropriate name such as “Level 1” and Save your scene.
- As soon as you save it, you will notice that now “SampleScene” is changed to the name you gave and below in the Project tab a unity file with your scene has been created as shown below.



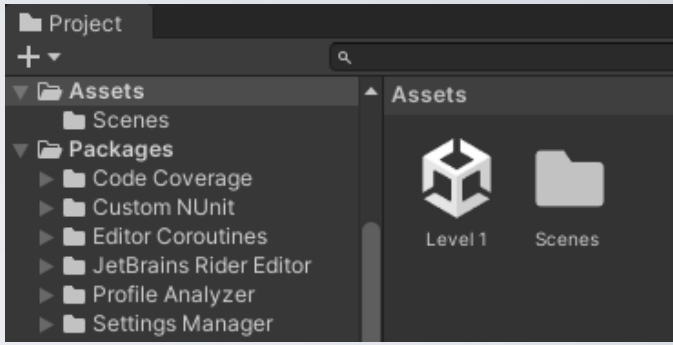


Basics

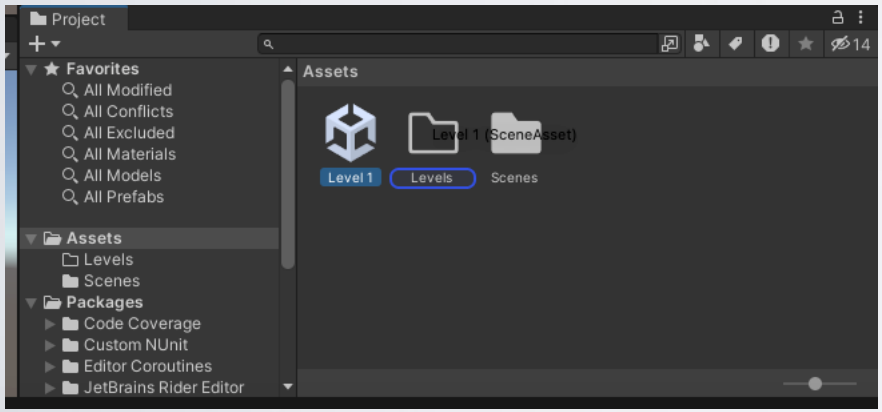
- Getting to know the editor – Managing Project files

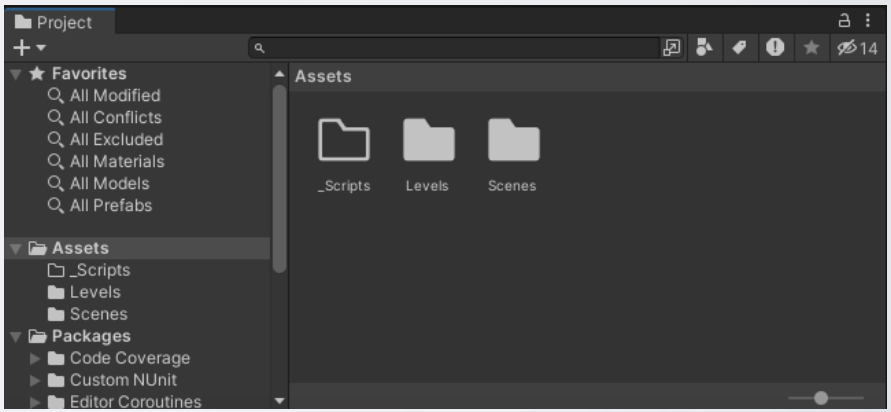
- Now you can create a folder named “Levels” by clicking on the Create button under Project tab and selecting “Folder”.

- You can now drag-n-drop Level 1 inside the newly created folder as shown below in order to keep a consistency and a good project organization between different components.



- Also, create a folder named “_Scripts” in order to store all scripts which you are creating during your game development



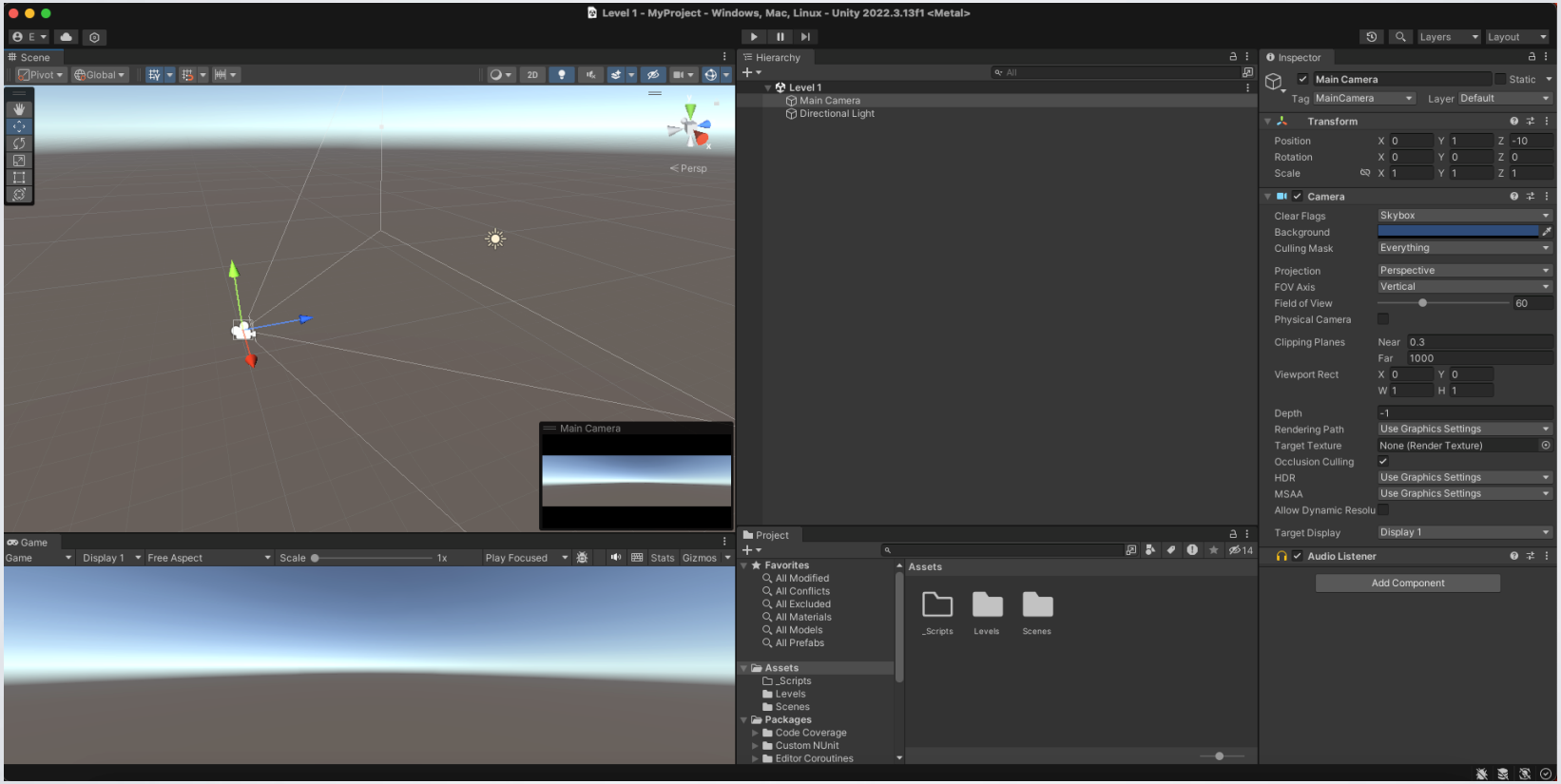


Basics

- Getting to know the editor – Inspector

- Inspector is a very useful tool offered by Unity from which you can manage all your Game Components, their states, attributes, textures and much more.

- For example, start by selecting your Main Camera from the Hierarchy tab, as shown in the image. All camera attributes will be displayed on the Inspector window.



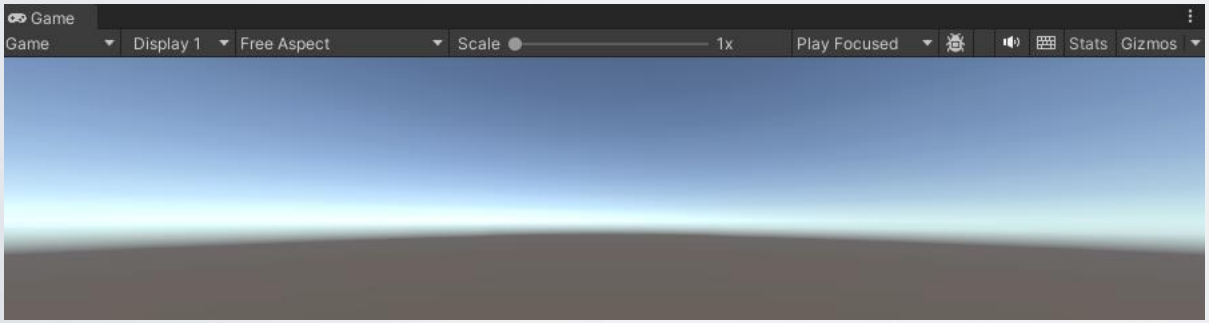
Basics

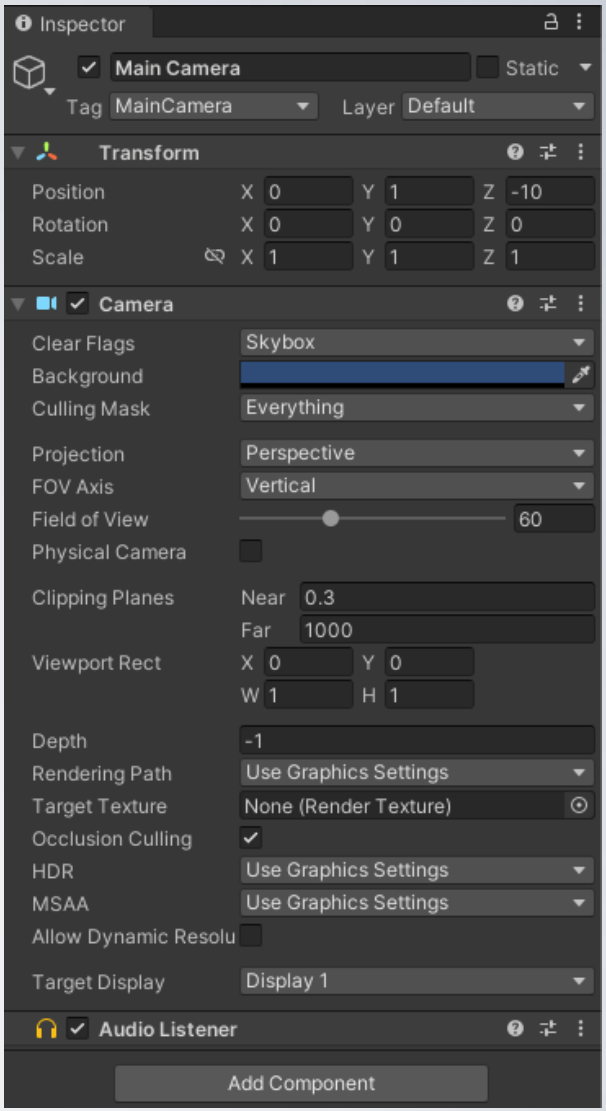
- Getting to know the editor – Inspector

- As you can see you can change your camera's position in the scene, rotate and scale it as well as change more technical attributes such as the field of view and the near and far clipping planes.

- Also, as soon as you have selected your camera on your Scene, you will notice that the camera is also selected on your scene view from which can see it's direction and can change it's position by simply dragging one of the colored arrows, specifying the axis of movement.

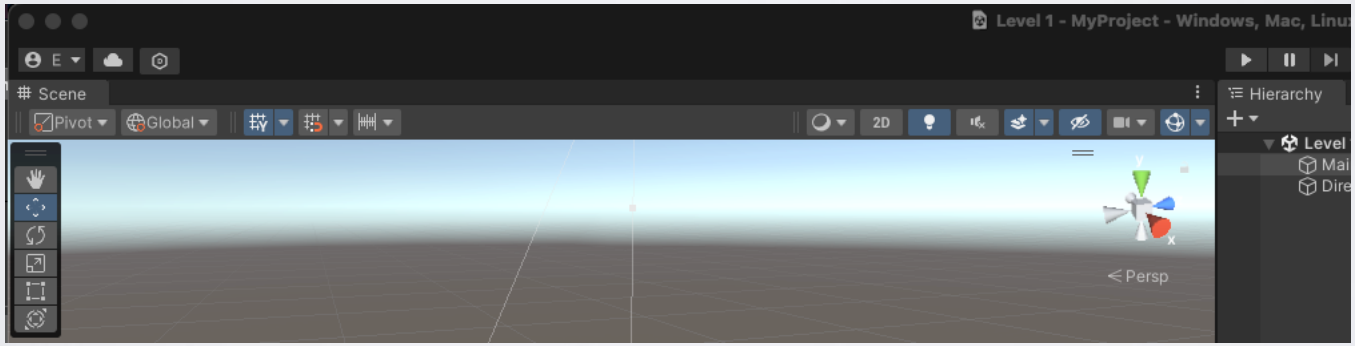
- Meanwhile, your Game view below displays the current look at of your camera.





Basics

- Getting to know the editor – Extra Options
- As seen in the image below, there are also extra and important tools that Unity offers.
- For example if you click the "hand" button (left column) you can navigate by dragging through your scene. Moreover, by clicking the "cross arrows" and having your camera selected you can translate its position.
- Even more, the third option offers the ability to rotate your game object, and the next is scaling.
- Finally, Unity offers Play and Pause buttons from which you can "play" and "pause" your scene and see what you have created from user/player perspective by playing the game.



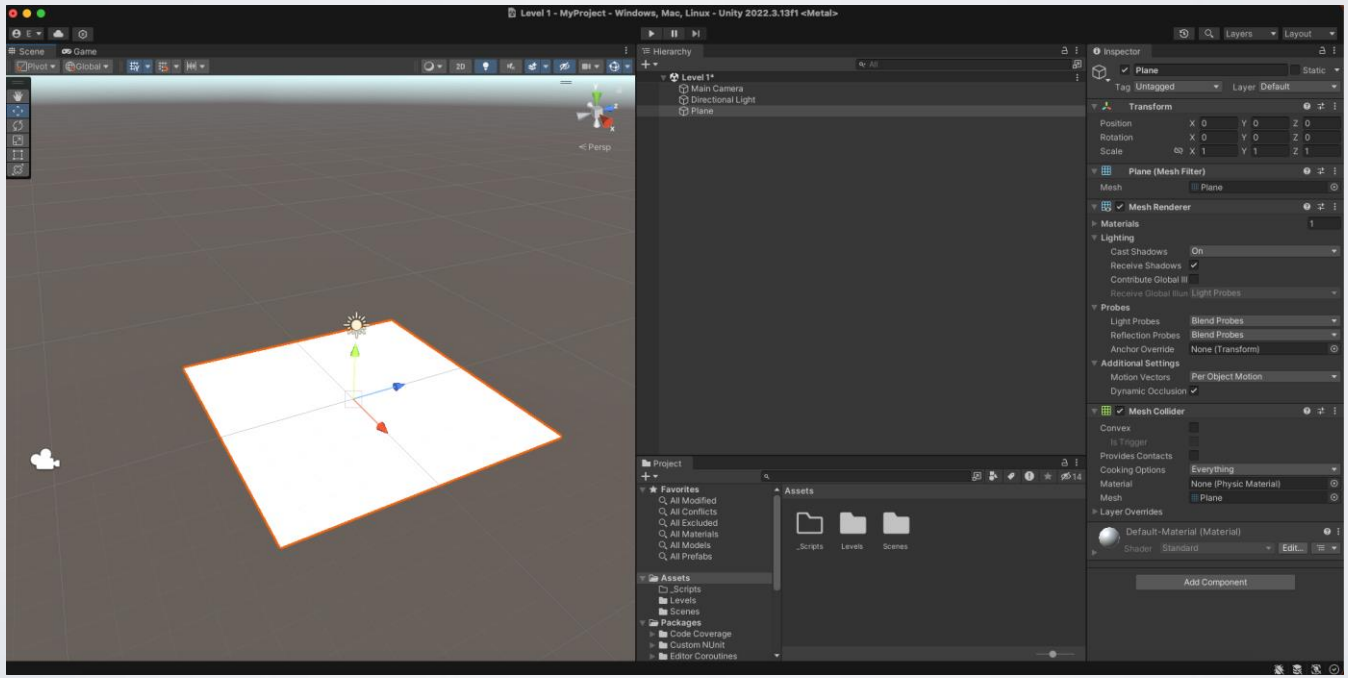
***Take note that Unity has many more options and toolkits for you to explore and create even more fascinating content than you can using just those presented in this tutorial.

Developing a Simple Game

- After setting up Unity and getting familiar with the editor you are ready to start developing your own game.

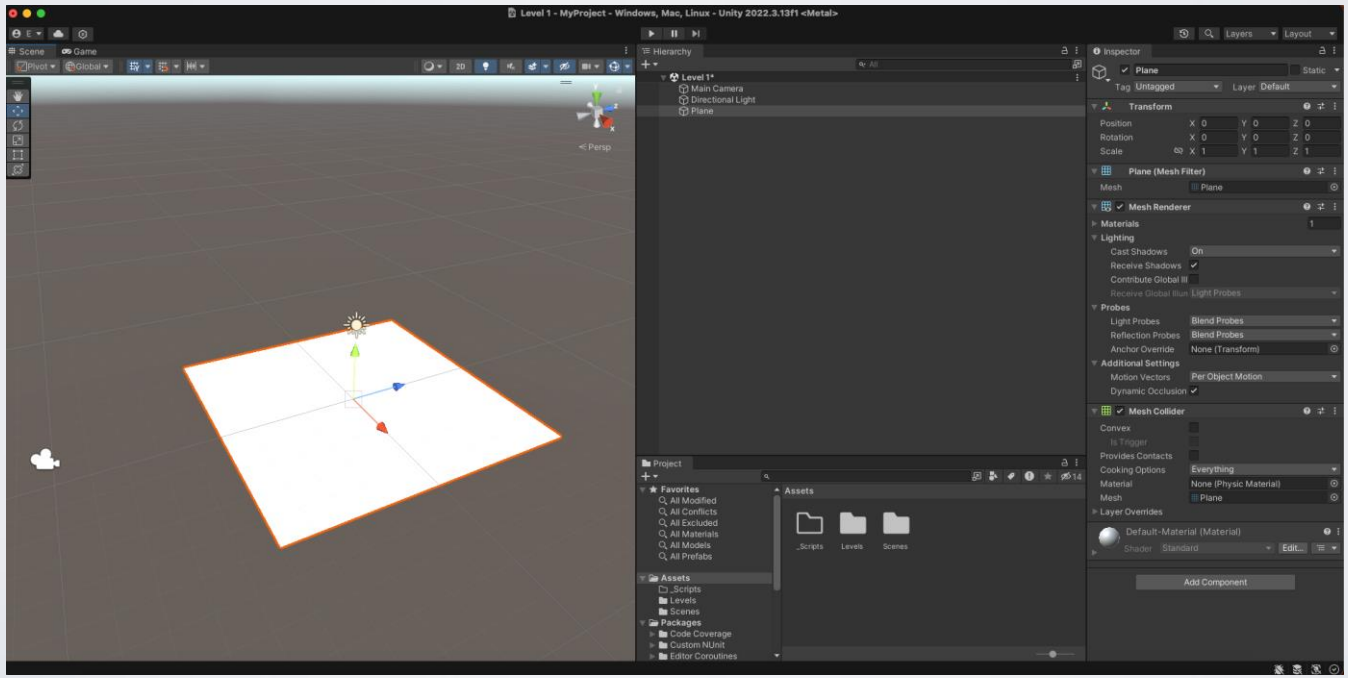
- You can start with the creation of a plane, simply by clicking on the "+" button under the hierarchy tab in your scene, selecting 3D Object and then Plane. This will add a plane to your scene.

- Reset its position to 0,0,0 by selecting the plane in the hierarchy menu, going to inspector and either entering manually the value in the corresponding position boxes, or by clicking on the three dots in the upper right and selecting "Reset".



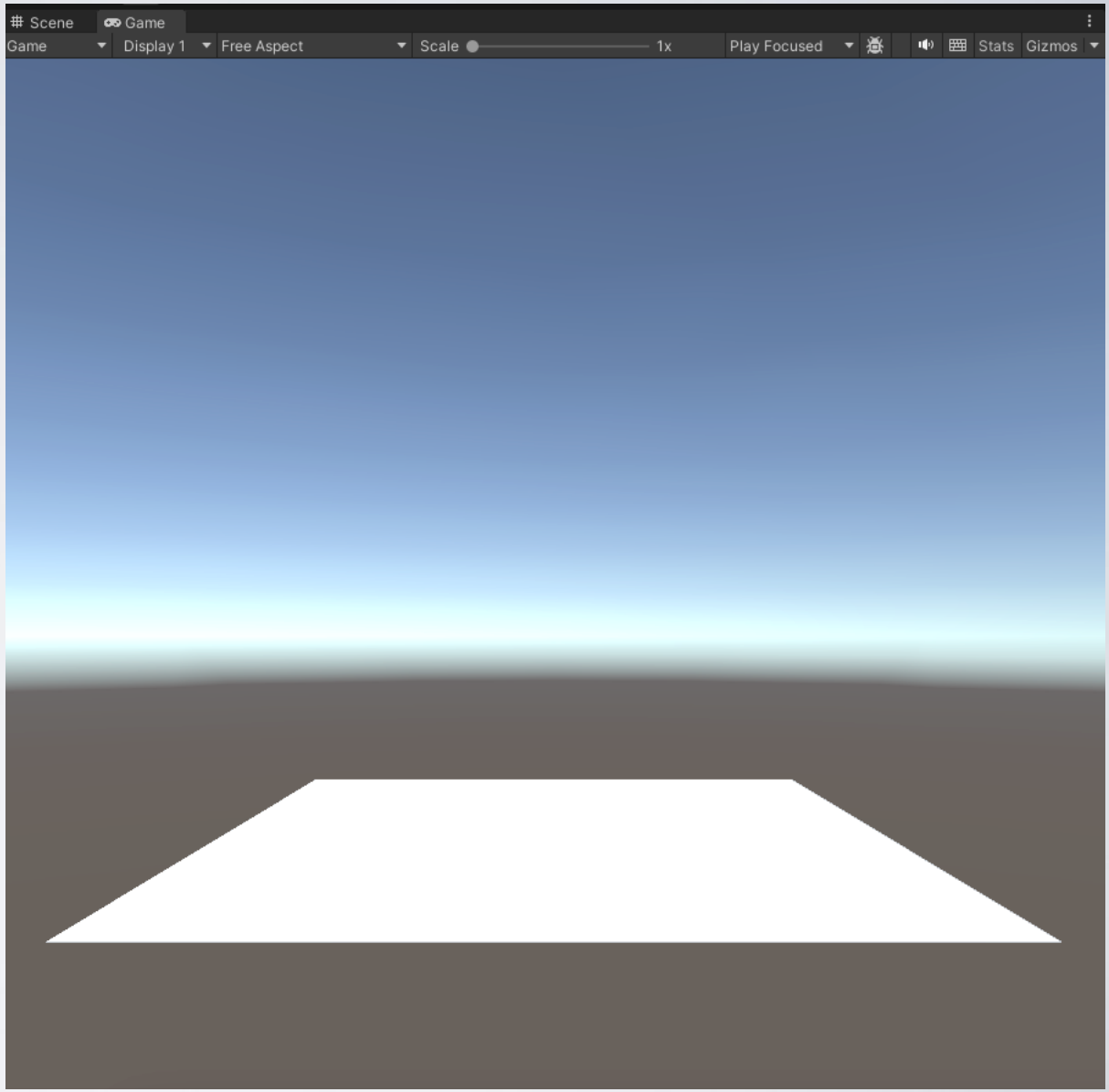
Developing a Simple Game

- As you can see this newly created plane has more components in the inspector than you expected.
- One of them worth mentioning is the Mesh Collider which is responsible for physics and collision detection to our game objects in order to provide a more realistic experience.
- Mesh Renderer on the other hand concerns the rendering of the game object. For example, as seen on the image it has attributes for lighting and material. If you deselect the Mesh Renderer tick-box you will notice that the plane disappears from the scene.



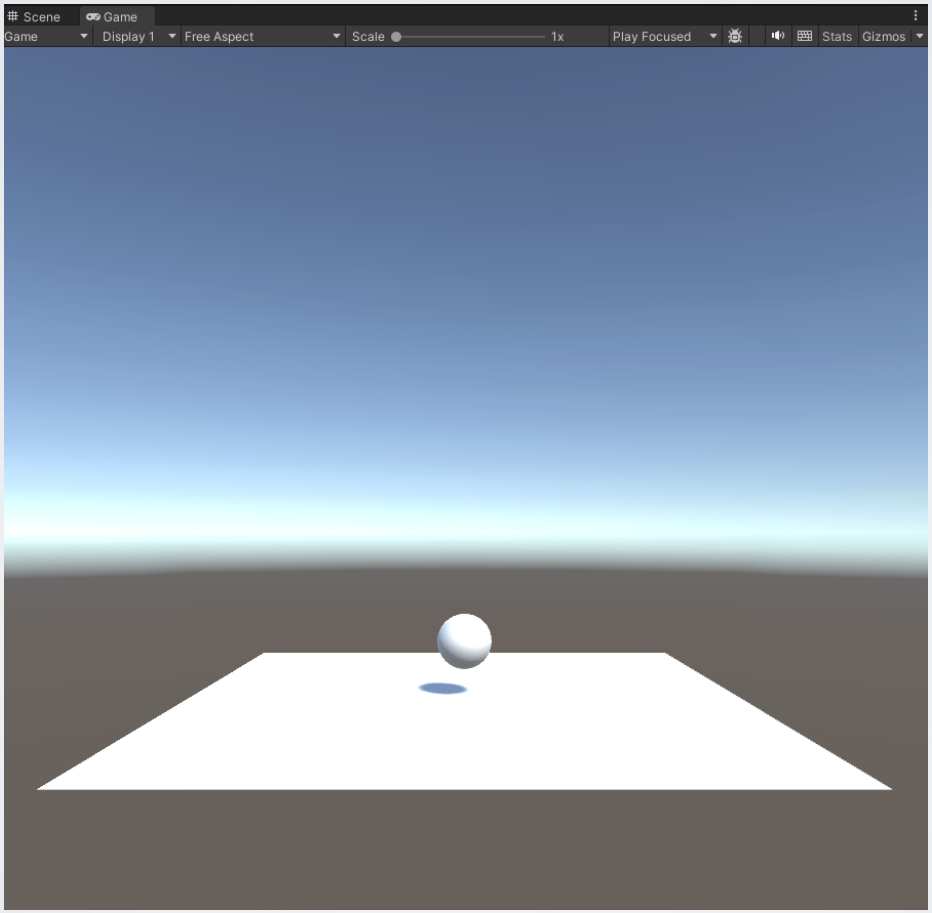
Developing a Simple Game

- Now you can select your Main Camera and set its Position at (0, 3, -14) and it's Rotation at (-4, 0, 0) in order for the plane to be viewable by the camera.
- Also, you can do so with the light depending on your personal preference.
- By now, you should have a game view looking similar to the one on the right.



Developing a Simple Game

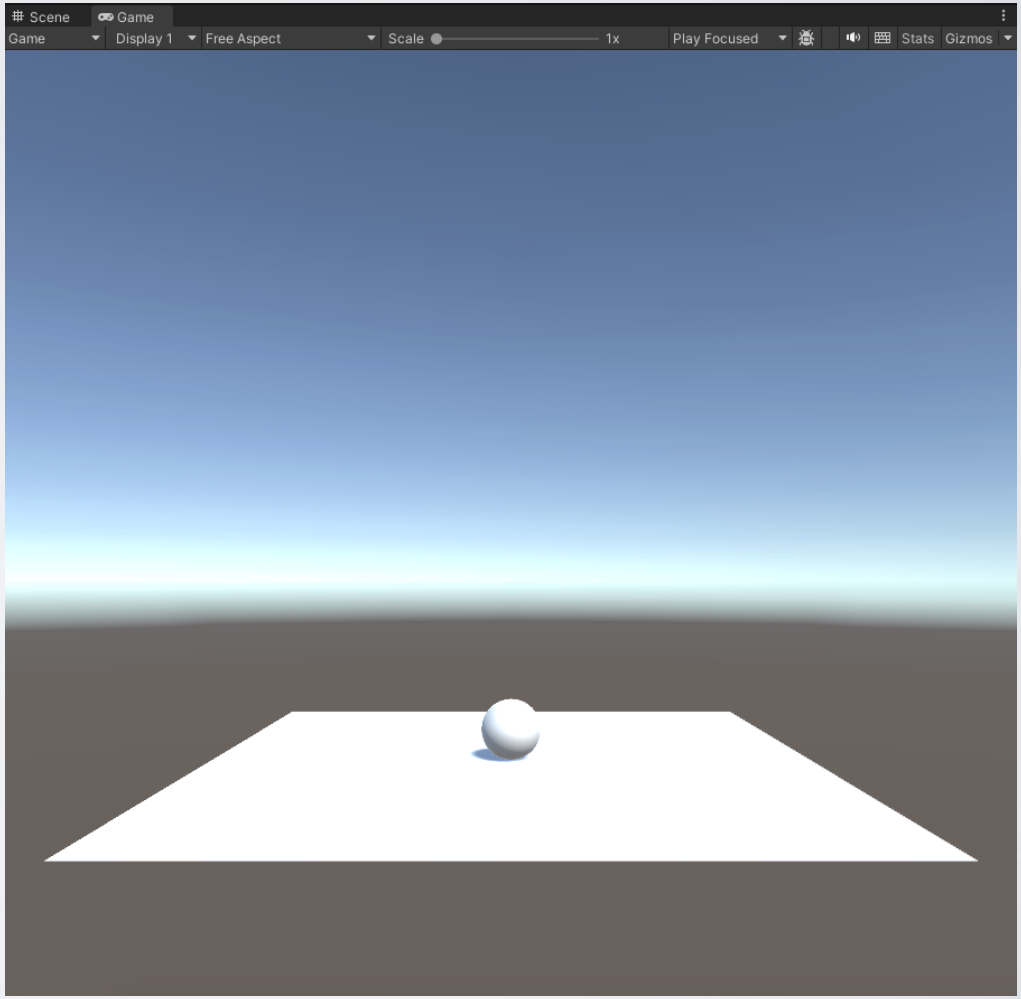
- Proceed by adding a simple Sphere and set its position to (0, 1, 0) so that our Sphere remains on top of the plane.
- Hit Play and notice that nothing happens.
- This occurred because our Sphere is not a rigid body and thus according to Unity it does not obey natures gravitational law.



Developing a Simple Game

- In order to fix that, select the sphere, go to the Inspector window, press "Add Component" button, navigate to Physics in the dropdown menu and select Rigidbody from the list.

- Quickly you will notice that a rigidbody is now attached to our sphere, containing some physical attributes such as mass.



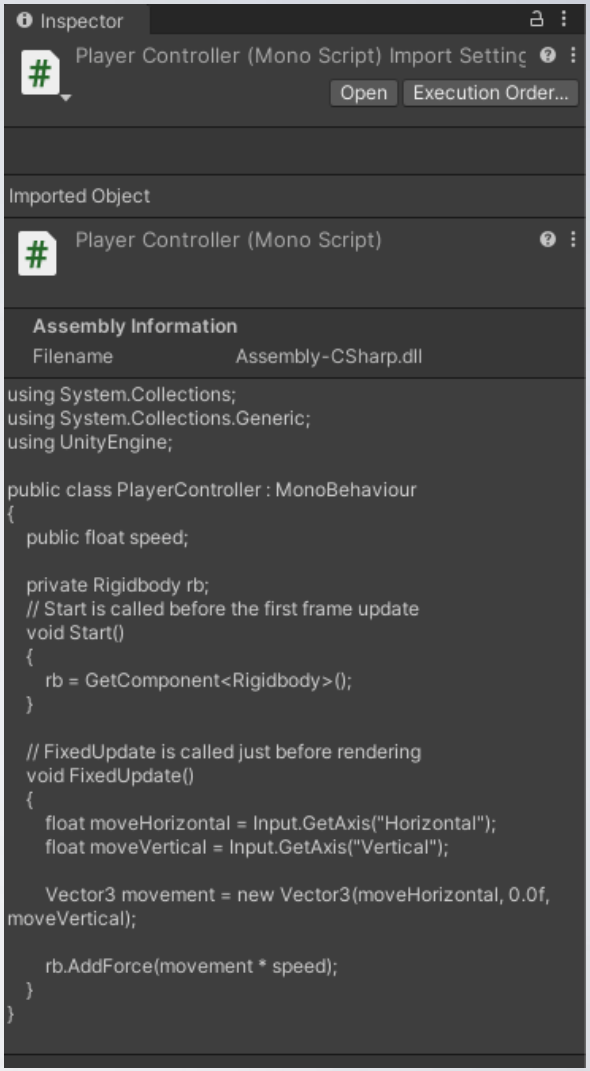
- Now hit Play again and notice that our sphere complies with gravitational law and falls onto the plane.

- Finally, place our sphere at 0.5 in the Y axis in order to simply sit over our plane and let's proceed to add some movement to it.

Developing a Simple Game

- In order to make our Sphere a moving game object we need to create a script and attach it to the Sphere.
- Scripting comes in two flavors when it comes to Unity, C# and JavaScript. You can choose the one you are already familiar with.
- In case you are not familiar with either JavaScript or C# but you do have solid knowledge of C++ then it is strongly suggested that you pick up C# as it will come more easy to catch up with.
- Scripting in Unity is one of the most important parts that you have to master in order to create good content.

Please do make sure to watch at least all beginner tutorials on Unity Scripting from here.

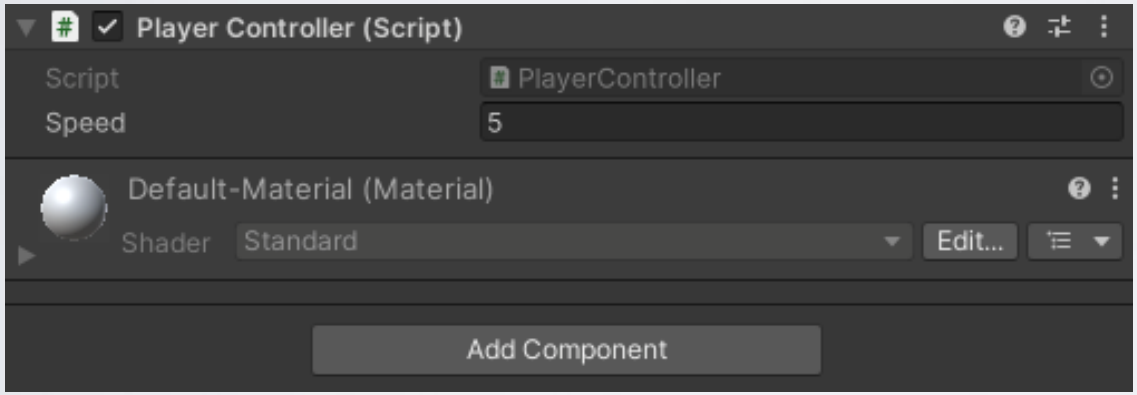


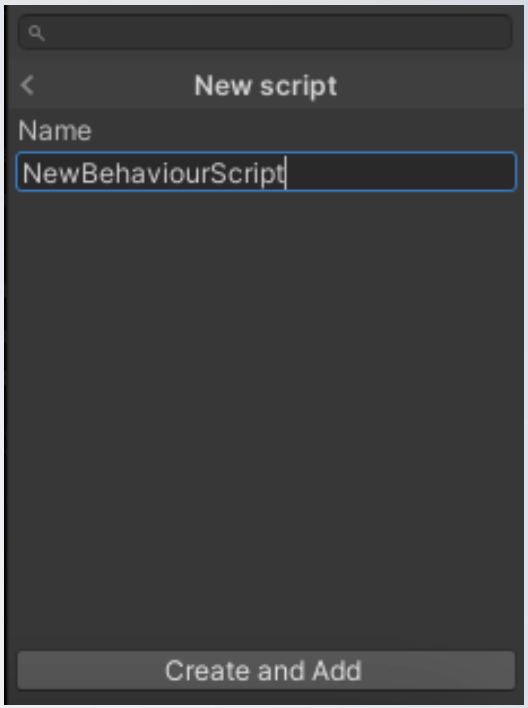
Developing a Simple Game

- Now, back to adding our script to the game object. Select the Sphere from the Hierarchy go to the inspector, press the button “Add Component” and either type in the search box “script” or search for “New Script” in the dropdown menu and select it.

- Proceed by selecting the language of your choice and name the script with a relevant name (i.e. “PlayerController”).

- Afterwards, click “Create and Add” and you will see that the newly created script is attached to the sphere in the Inspector tab.



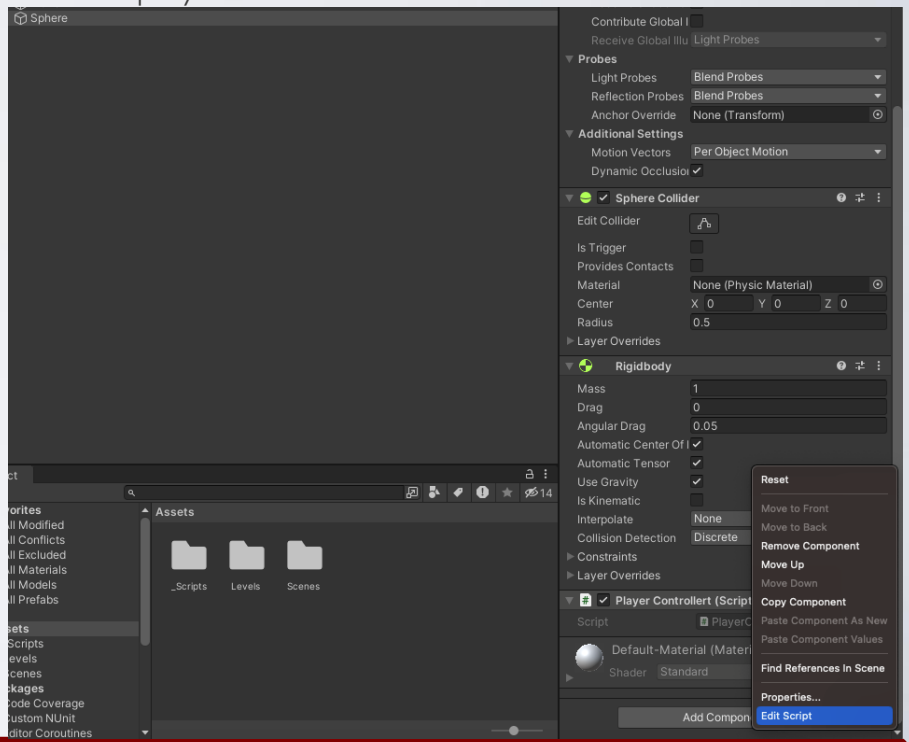


- Finally, under the Project window drag and drop the script into the corresponding “_Scripts” folder.

Developing a Simple Game

- Opening and editing the script.
- In order to edit our newly created script you either have to double-click it from the Project/_Scripts folder or in the Sphere inspector click the little gear and press the “Edit Script” button.
- This will open up Visual Studio, or your preferred IDE (which can be set in Unity Editor Preferences) with your script in it.
- When the script opens in the editor you will see the exact same code displayed below.

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

0 references
public class PlayerController : MonoBehaviour {

    // Use this for initialization
    0 references
    void Start () {

    }

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

    }
}
```



Developing a Simple Game

- Opening and editing the script.
- On top there will be some “using” instructions which are similar to “include” as in C/C++, leave them be.
- You can see that we have a public class which inherits from another class called MonoBehaviour. This class provides us with basic functionality as is the Start and Update functions.
- As the comments in the code clearly indicate, Start is called in the beginning of the game and can be used similarly as a constructor, assigning basic values and attributes to our game component.
- On the other hand, Update is called once per frame in order to do the necessary actions and calculations required for our game.

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

0 references
public class PlayerController : MonoBehaviour {

    // Use this for initialization
    0 references
    void Start () {

    }

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

    }
}
```

Developing a Simple Game

- Opening and editing the script.

- In this case, we will use the Start function to initialize our declared variables.

- Next, using the Update function we will calculate the Sphere's movement based on User Input per frame.

- Bare in mind, that you can attach more than one scripts in a game object and even create classes that do not inherit from MonoBehaviour thus, do not have functionality as is Start and Update functions.

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

0 references
public class PlayerController : MonoBehaviour {

    // Use this for initialization
    0 references
    void Start () {

    }

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

    }
}
```

Developing a Simple Game

- Opening and editing the script.
- To begin with, let's add some variables to our script needed for the Sphere movement.
- Declare a public float speed variable on top and a private Rigidbody variable afterwards. Keep in mind that public variables are available for modification from inside the Unity Inspector.
- In the Start function we want to assign our Rigidbody variable with the Rigidbody of the sphere we created previously.

As for the speed variable, we will give it a value through the inspector, since it is declared as public.


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

0 references
public class PlayerController : MonoBehaviour {

    public float speed;

    private Rigidbody rb;

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

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

    }
}
```

Developing a Simple Game

- Opening and editing the script.
- Assign the speed variable an initial value of 3 through the inspector.
- Since we want our sphere to move in a physical way, change Update function to FixedUpdate as shown bellow. Fixed update is called right before rendering in order for other calculations such as physics to take place before.
- Afterwards, as shown in code you can see the basic movement calculations done for our sphere.

```
public class PlayerController : MonoBehaviour {

    public float speed;

    private Rigidbody rb;

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

    // FixedUpdate is called just before rendering.
    0 references
    void FixedUpdate () {

        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");

        Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);

        rb.AddForce(movement * speed);
    }
}
```

Developing a Simple Game

- Opening and editing the script.
- Input is a built-in function that Unity provides for handling events. We will use Input to get a float number on both Horizontal and Vertical directions.(Read more about it, here)
- Then, we create a direction vector and initialize it with moveHorizontal and moveVertical values we got from Input.GetAxis.
- You can clearly see why the Y axis field is zero, we do not want our sphere to move up and down.
- Finally, we add a force to our rigidbody which is a vector3 value pointing to a direction multiplied by the speed.
- This force will move the sphere towards the direction given.

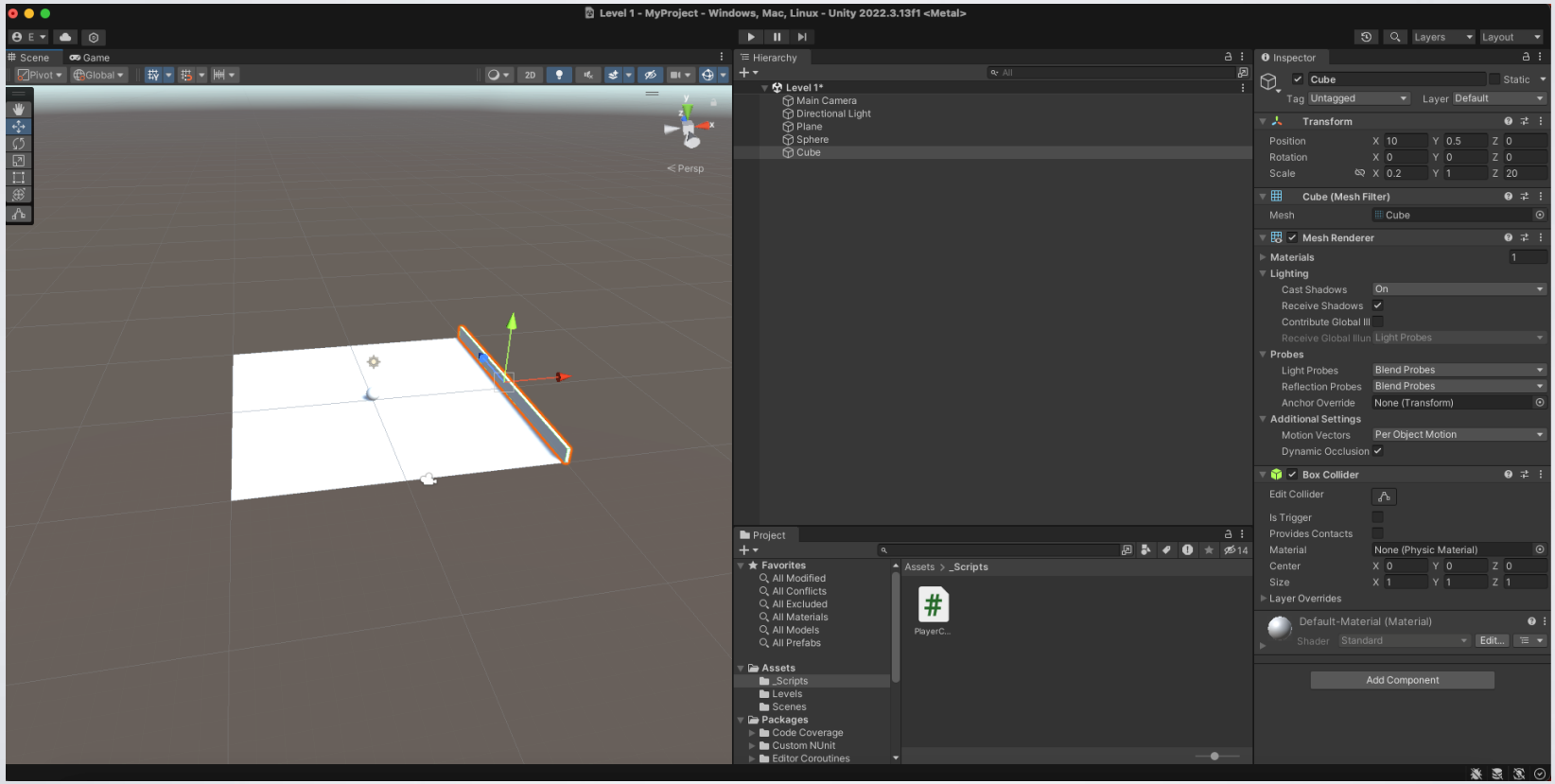
```
public class PlayerController : MonoBehaviour {

    public float speed;

    private Rigidbody rb;

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

    // FixedUpdate is called just before rendering.
    0 references
    void FixedUpdate () {

        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");

        Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);

        rb.AddForce(movement * speed);
    }
}
```

Developing a Simple Game

- Save the script and go back to Unity.
- If you have completed all previous steps correctly and hit the Play button you can now move the sphere by pressing either the arrow or the WASD keyboard keys.
- Congrats!
- But wait, there is more…
- You will quickly start noticing that your plane is pretty small in size and that your sphere falls of the ground. Let's increase it in size and build some walls around it to prevent the sphere from falling….

Developing a Simple Game

- Start by scaling your plane with value 2 in every axis.
- Then proceed to create a cube object by the Hierarchy tab.
- Translate it by (10, 0.5, 0) to be just right of the plane and starting from the same height.
- Afterwards, scale it by (0.2, 1, 20). This will be our East Wall…

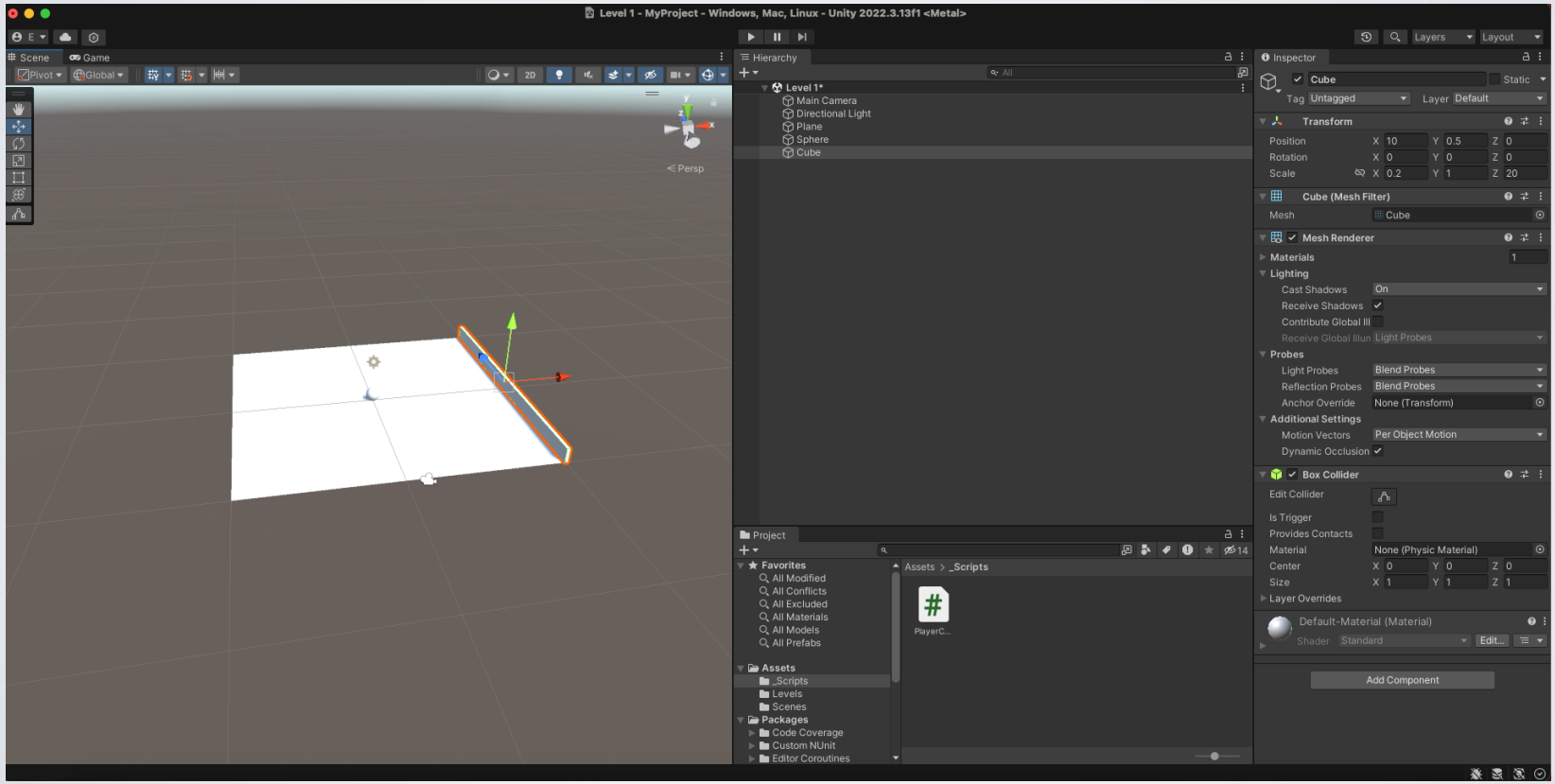


Developing a Simple Game

- Now you can just duplicate the Cube object by right clicking it in the hierarchy menu and selecting Duplicate. Do this 3 times and you will end up with 4 “Walls”. Translate the first one by -10 in the X axis to represent the Left wall.

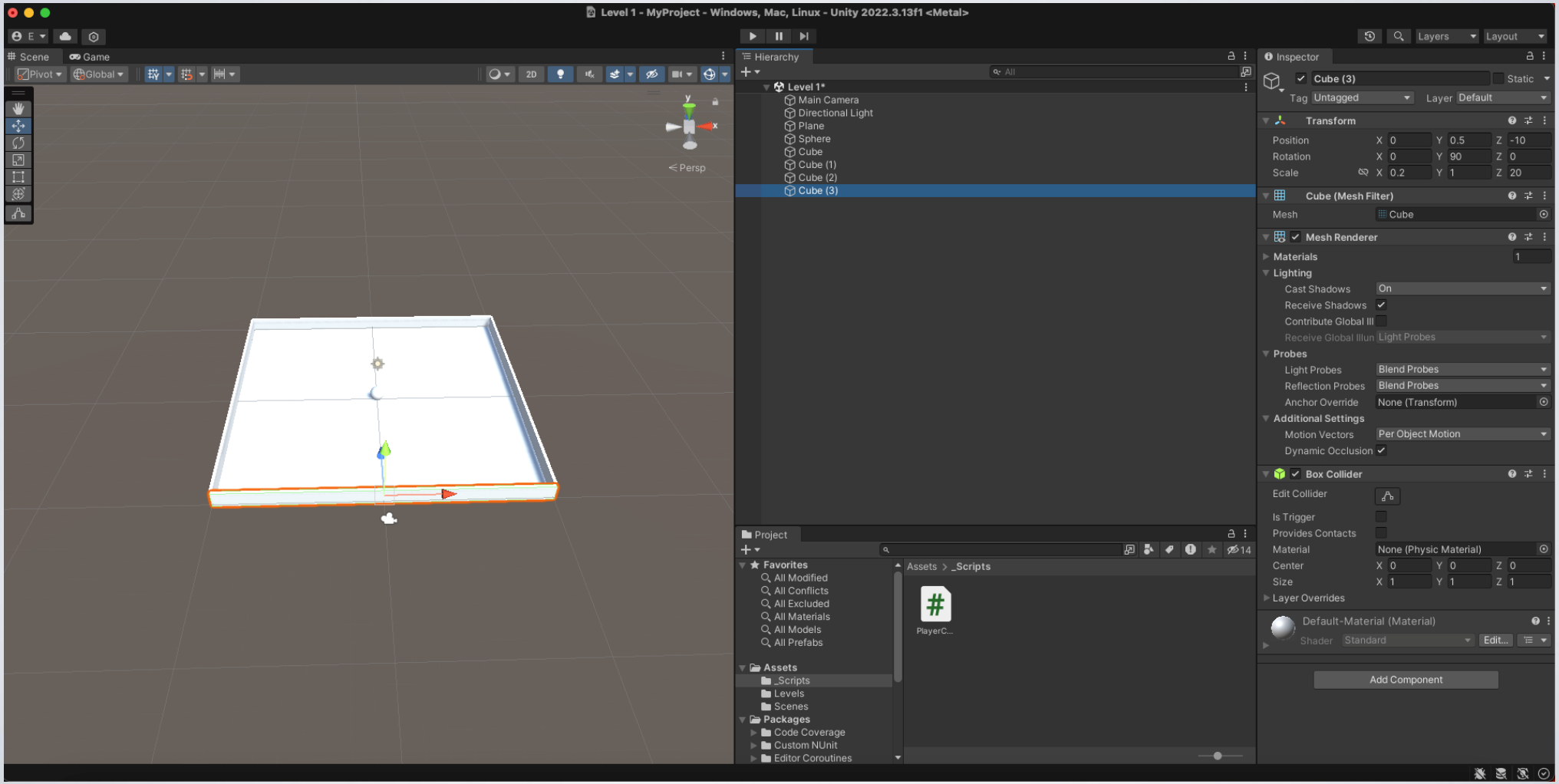
- After that, rotate the third cube by 90 degrees in Y axis and translate it by -10 in the Z axis to represent our North wall.

- Finally, do the same for last cube and translate it by 10 in the Z axis for our South wall.



Developing a Simple Game

- If you completed all previous steps correctly, you will end up with something like this…

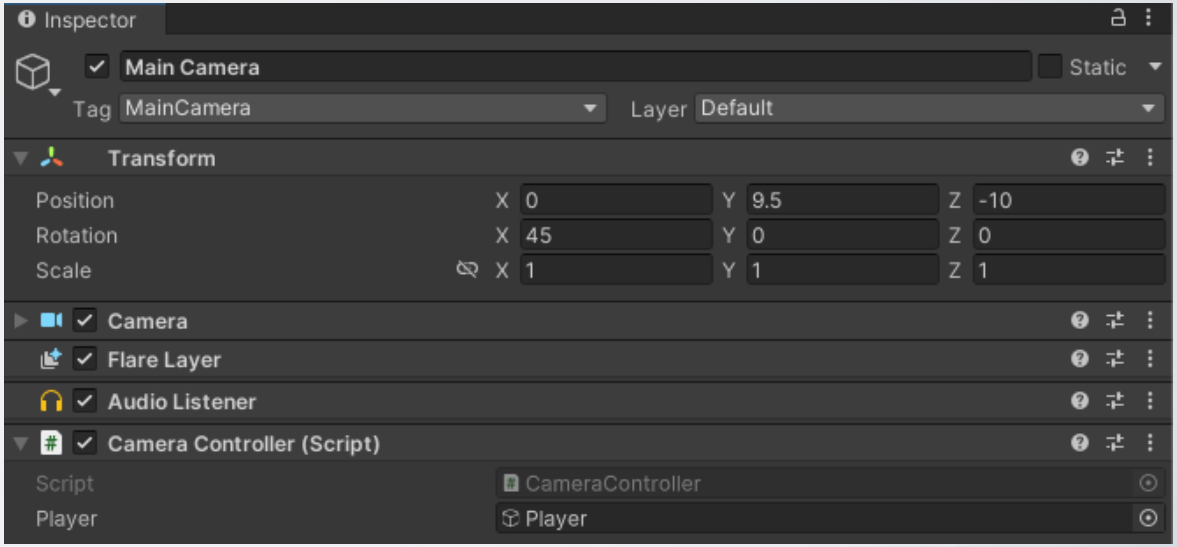


Developing a Simple Game

- The sphere now collides with the "Walls" and stays in place…
- But, what about the Camera???
  - True, our camera remains static which makes it an unpleasant user experience.
  - Let's fix it!!!

Developing a Simple Game

- We will add movement to our camera in a 3rd player perspective.
- To do so, we obviously need a script.
- Select our Main Camera from the inspector and add a script as we have done previously with the player controller script. You can name it CameraController.
- Now, open the script for editing…



Developing a Simple Game

- To begin with, declare a public GameObject variable named player.
- As you can tell, our GameObject variable is public because we are going to assign it our Sphere object as a value directly from the Inspector menu.
- Continue then by declaring a Vector3 variable named offset which is initially assigned at the Start function.

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

public class CameraController : MonoBehaviour {

    public GameObject player;

    private Vector3 offset;

    void Start()
    {
        offset = transform.position - player.transform.position;
    }

    void LateUpdate()
    {
        transform.position = player.transform.position + offset;
    }
}
```

Developing a Simple Game

- Offsets value is the camera's starting position minus the Sphere's starting position.

- We use an offset because we want our camera to follow our player from a distance.

Notice, that we use LateUpdate because we want our update function to be called last, after all other updates have been called to ensure that the sphere will have moved first before applying any transformations to the camera.

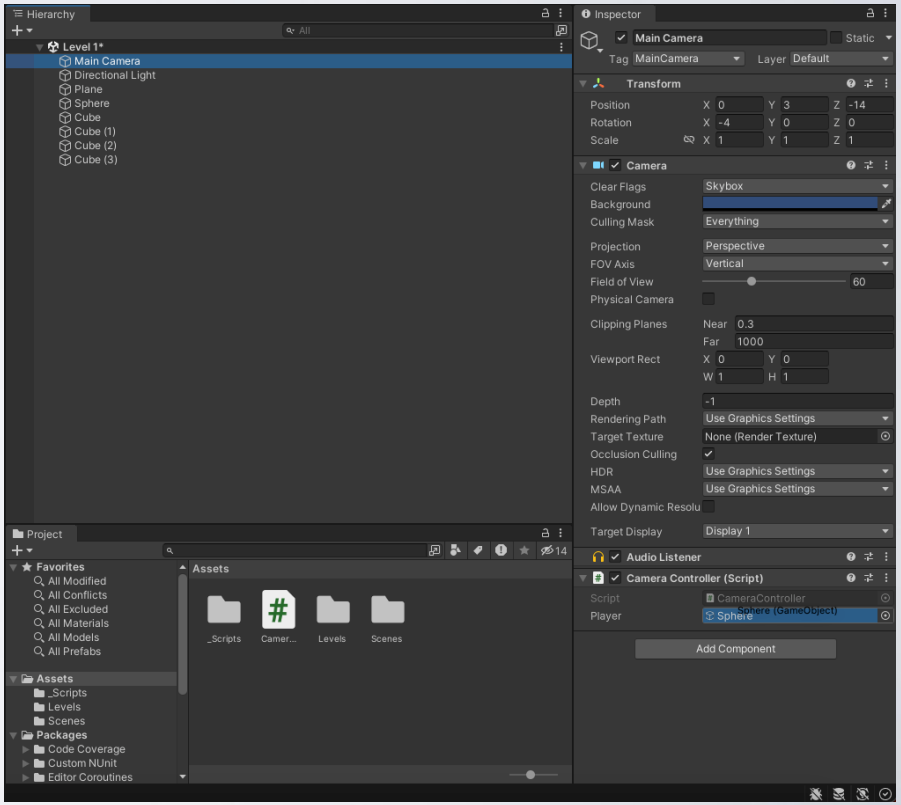
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraController : MonoBehaviour {

    public GameObject player;

    private Vector3 offset;

    void Start()
    {
        offset = transform.position - player.transform.position;
    }

    void LateUpdate()
    {
        transform.position = player.transform.position + offset;
    }
}
```

- Finally, we transform the camera's Position according to player's position transformation plus the offset we kept from the beginning.

Developing a Simple Game

- Save the script and go back to Unity.
- Before hitting Play, do not forget to drag the Sphere Game Object to the Main Camera script public variable as shown below.

Now you can hit Play and fortunately you will be pleased.



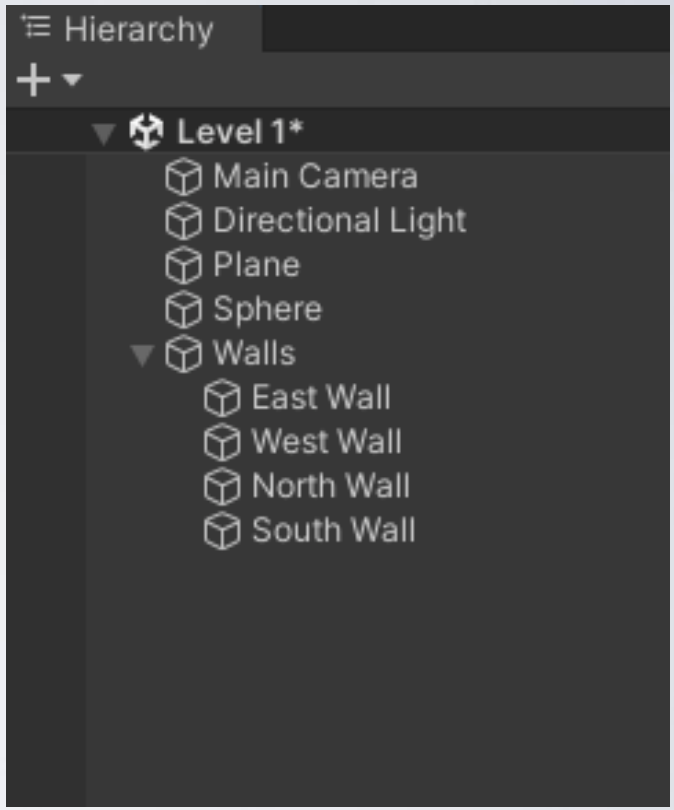
Developing a Simple Game

- Extra steps and tips.

- Please give appropriate names to all your Game Components/Objects (you will thank me later!) in order to avoid possible confusions.

- You can rename your components by right clicking on them and choosing rename.

- For example, you can name Plane as Ground, Sphere as Player, and each cube according to its position and rotation (i.e. North Wall).



Even better, create an Empty game object from the Hierarchy and add all Walls as children to it as on your right.

Keeping a project clean and organized is essential!!!

Organizational and management techniques and practices are a big plus in our industry.

Developing a Simple Game

- Final words…
- We didn't build a game in case you did not notice. We were not even close to it.
- A proper game needs game objectives, different levels and difficulties, a start and an end. Moving around a ball is not a game.
- But, we did start developing a real game called Roll-a-ball which has a complete tutorial that Unity provides here.
- In other words, you are already close to the final game and you should finish it at home as an exercise…
- Unity's tutorials are excellent and can give you a full insight of the Engine's capabilities, so it is highly recommended to try them out.

Thank you!