Introduction to Unity

Iftekharul Islam

Files for Today's Session

- Project files and slides in PPT format (with animations):

  https://tinyurl.com/cs690-lab3

Outline

- **Introduction to Unity**
- **Setting up the Scene**
- **Moving the Player**
- **Moving the Camera**
- **Detecting Collision**
  - (Please review this section from the slides and the mentioned links. I couldn’t go over it during the lab as I was not feeling well that day.)

Unity

- Unity is a cross-platform game development system.
- Unity features a complete toolkit for designing and building games.
- In this session, we will present the basic elements of Unity.
- A good starting point to learn Unity: https://learn.unity.com/

Download and Installation

- Download: https://unity.com/download
- Installation tutorial: https://www.youtube.com/watch?v=ewiw2tcfen8
- We will be using **Unity 2022.3.19f1** for the lecture.

Section

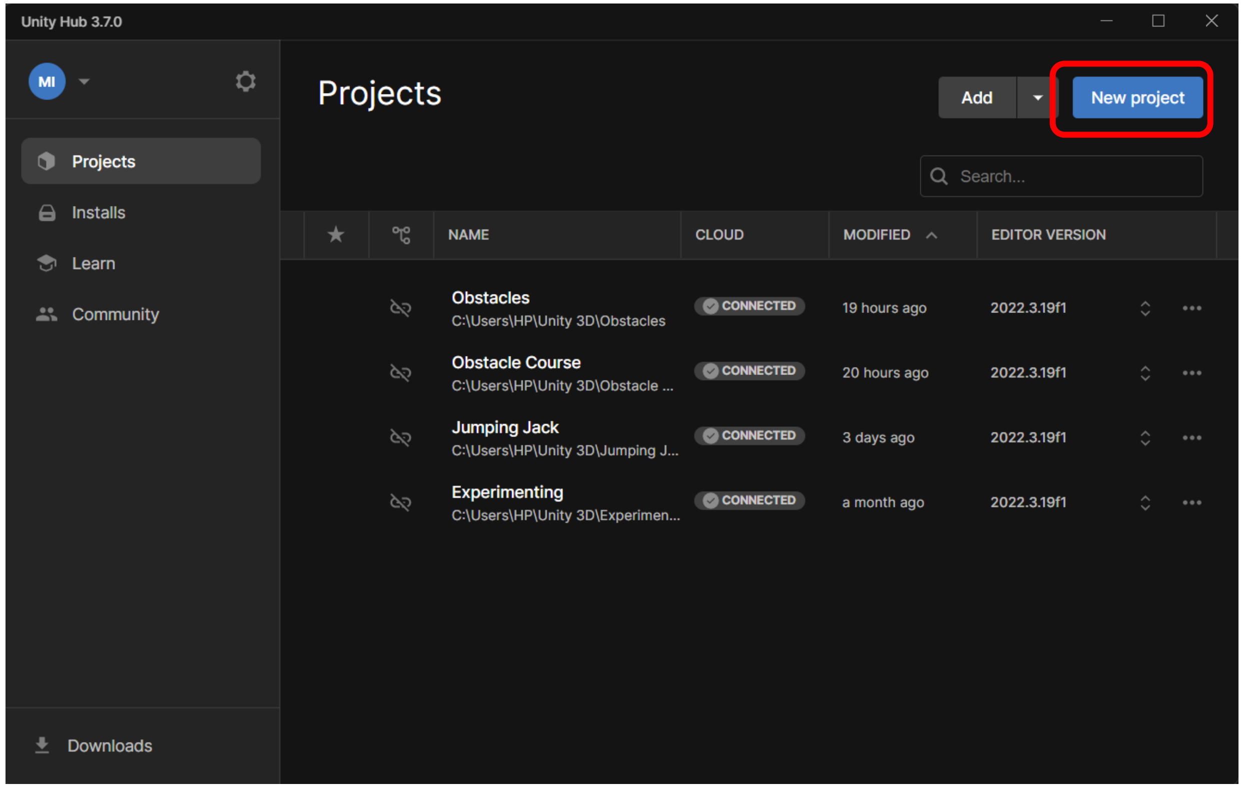
Setting up the Scene

Create a new Unity project

- **Select a project template.**
  - Open Unity Hub and log in using your Unity account.
  - Select **New Project**.
  - Select the **3D** template and then select **Download template** if it is not already downloaded**.**

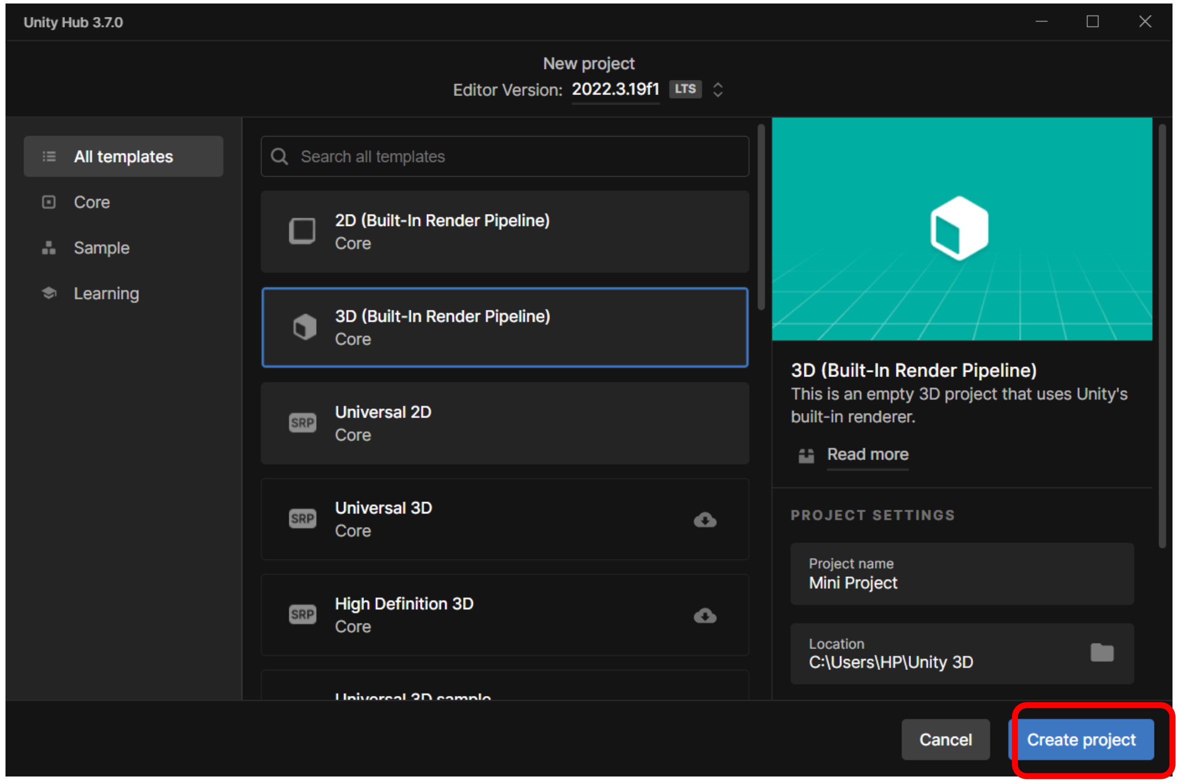
- **Create the new project.**
  - In the Project Settings, enter “Mini Project” in the **Project Name** box, select the **Location** box, and select a local folder of your choice.
  - Select **Create Project** and wait for the Unity Editor to open.

Create a new Unity project



Projects

NAME	CLOUD	MODIFIED	EDITOR VERSION
Obstacles C:\Users\HP\Unity 3D\Obstacles	CONNECTED	19 hours ago	2022.3.19f1
Obstacle Course C:\Users\HP\Unity 3D\Obstacle ...	CONNECTED	20 hours ago	2022.3.19f1
Jumping Jack C:\Users\HP\Unity 3D\Jumping J...	CONNECTED	3 days ago	2022.3.19f1
Experimenting C:\Users\HP\Unity 3D\Experimen...	CONNECTED	a month ago	2022.3.19f1



New project
Editor Version: 2022.3.19f1 LTS

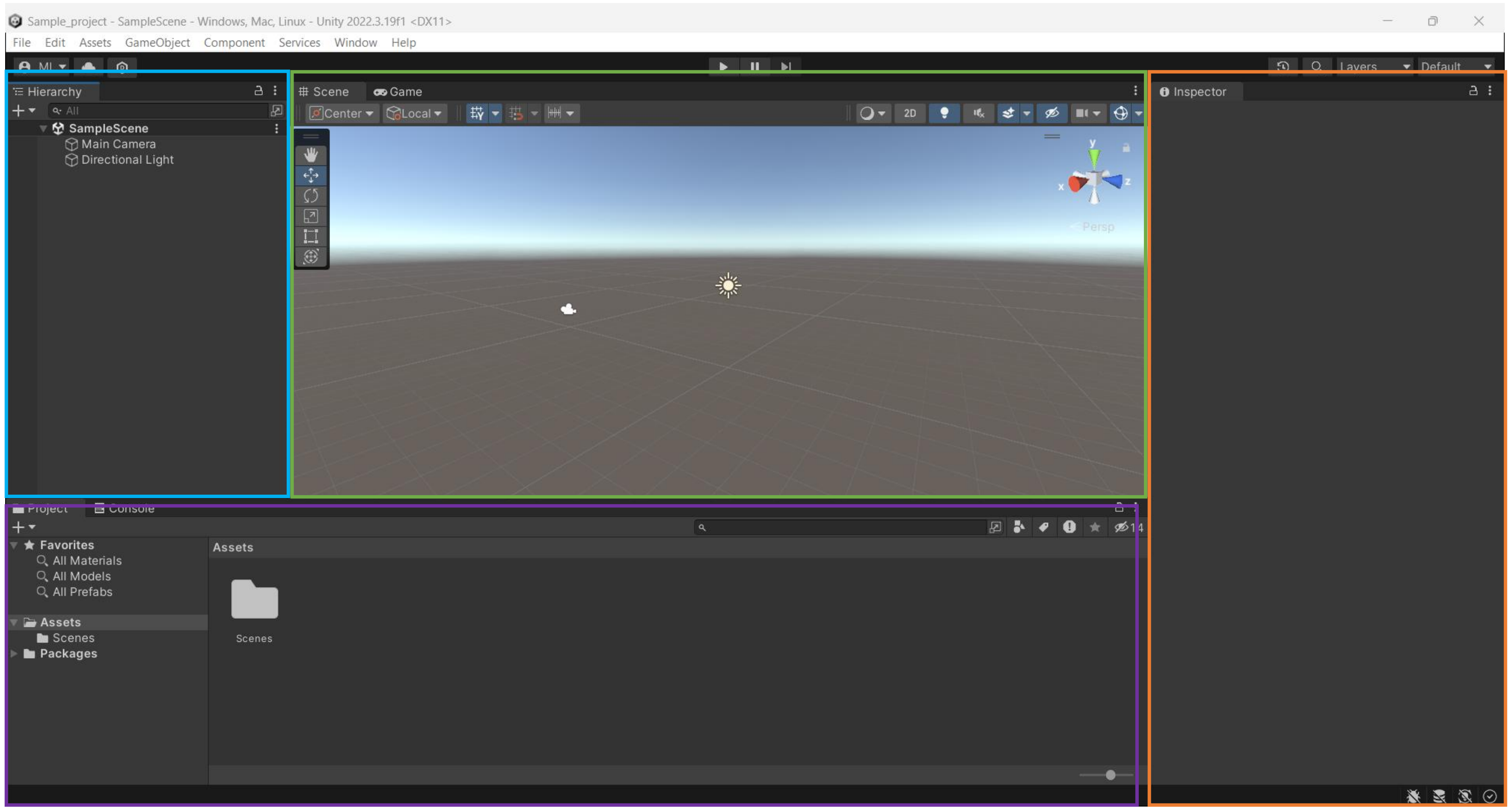
- 2D (Built-In Render Pipeline) — Core
- 3D (Built-In Render Pipeline) — Core
- Universal 2D — Core
- Universal 3D — Core
- High Definition 3D — Core
- Universal 3D sample

3D (Built-In Render Pipeline)
This is an empty 3D project that uses Unity's built-in renderer.

PROJECT SETTINGS
Project name: Mini Project
Location: C:\Users\HP\Unity 3D

Overview of the Unity IDE

- **Scene/game view**
  - Build/play scene
- **Hierarchy**
  - Manage game objects
- **Inspector**
  - Manage components
- **Project window**
  - Manage assets

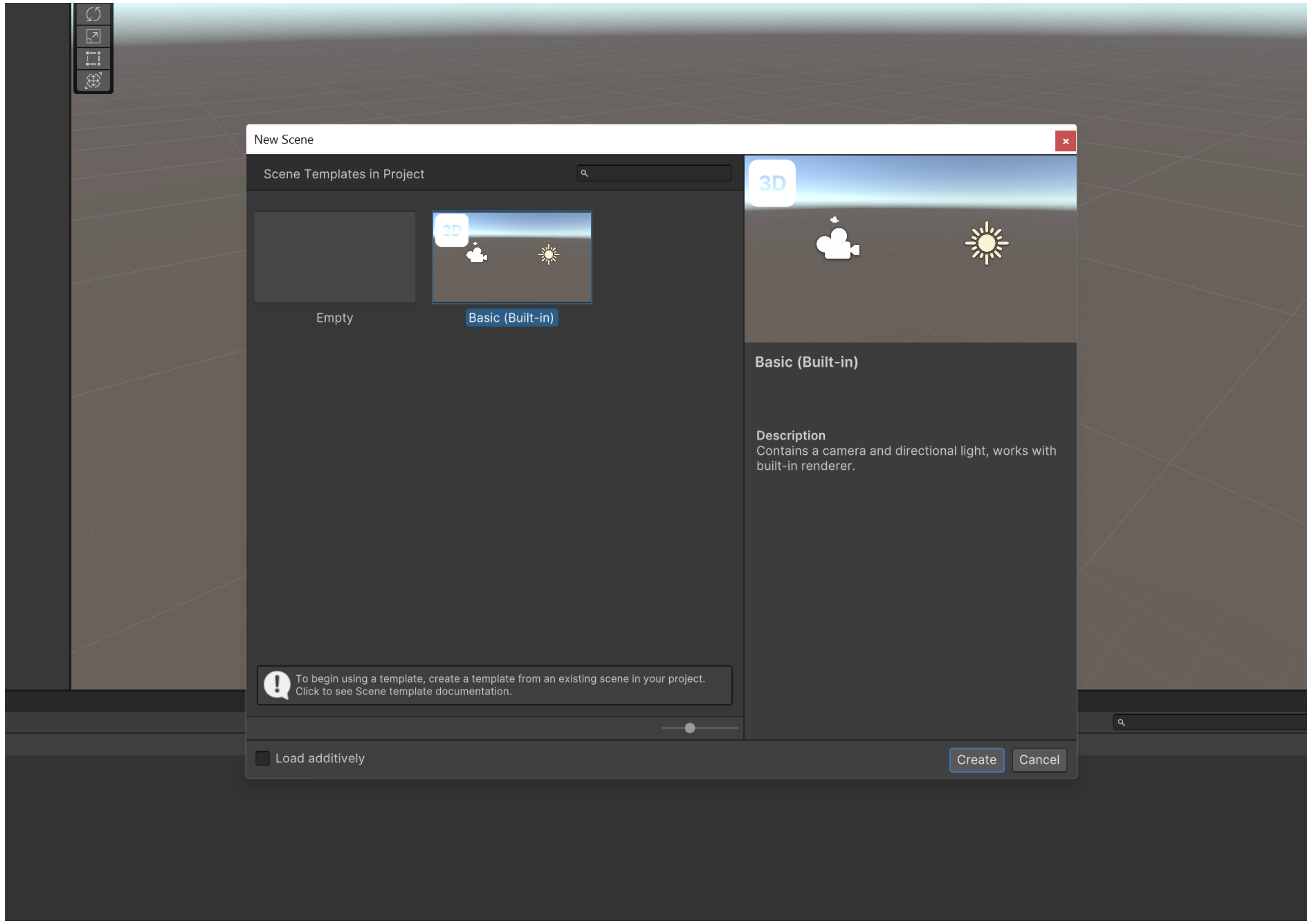


Create a new Scene

- **Set up your workspace.**
  - Make sure the **Default** Layout is selected from the **Layout** dropdown for tutorial consistency.

- **Create a new scene from a template.**
  - To create a new scene, select **File** > **New Scene**.
  - Select the **Basic (Built-in)** template, then select **Create**.

Create a new Scene


New Scene

Scene Templates in Project

Empty

Basic (Built-in)

Basic (Built-in)

Description
Contains a camera and directional light, works with built-in renderer.

To begin using a template, create a template from an existing scene in your project.
Click to see Scene template documentation.

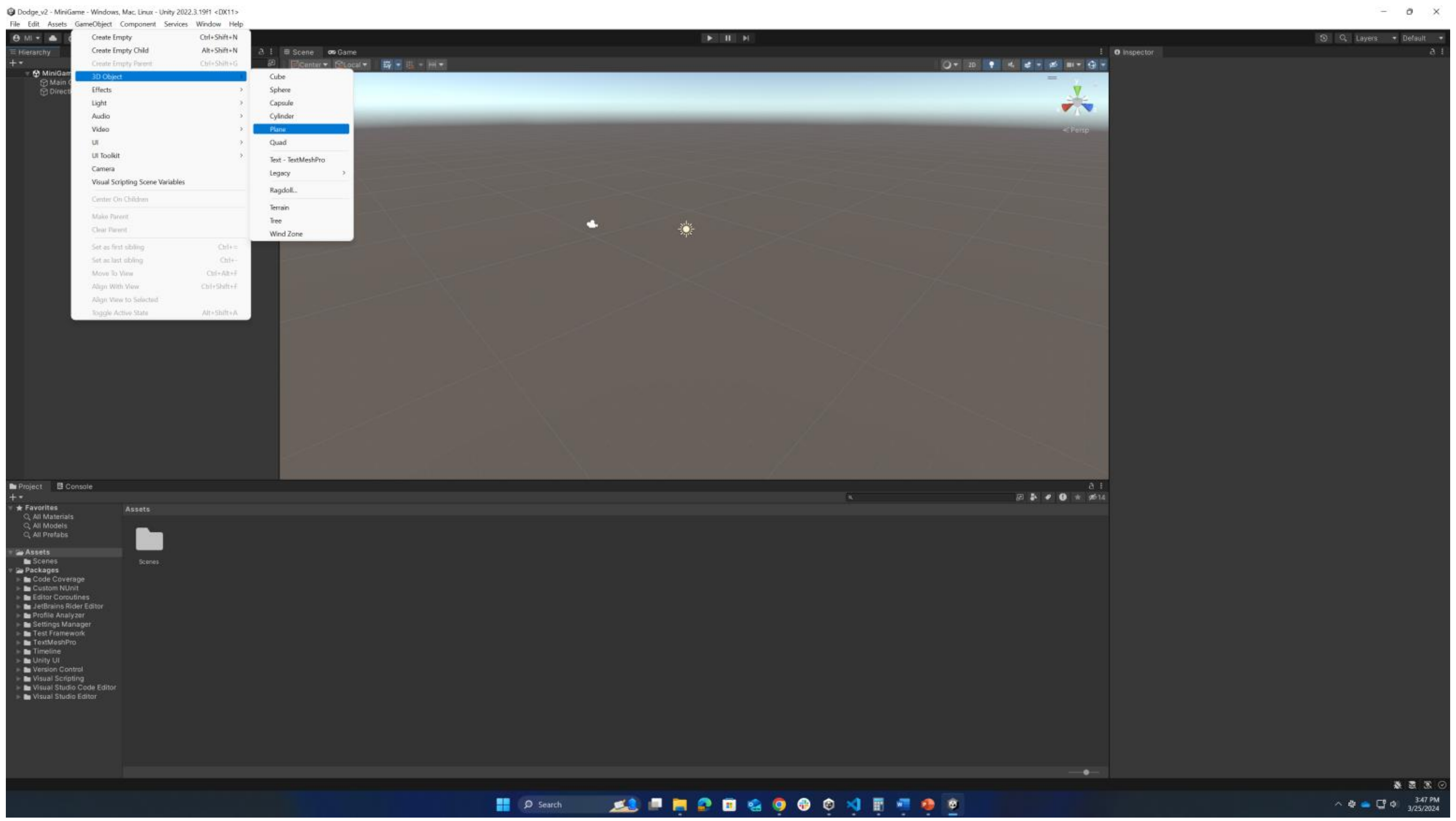
Load additively

Create a new Scene

- **Save the scene.**
  - Select **File** > **Save As**.
  - Name the scene "MiniGame".
  - Save the scene in a new folder named “Scenes”.

Create a primitive plane

- **Create a Plane GameObject.**
  - From the main menu, select **GameObject** > **3D Object** > **Plane**.
  - Alternatively, in the **Hierarchy** window, select the **Add** menu (**+**) > **3D Object** > **Plane**.
  - At the top of the **Inspector** window, rename the newly created **Plane** GameObject "Ground".



Mini Challenge: 3 Minutes

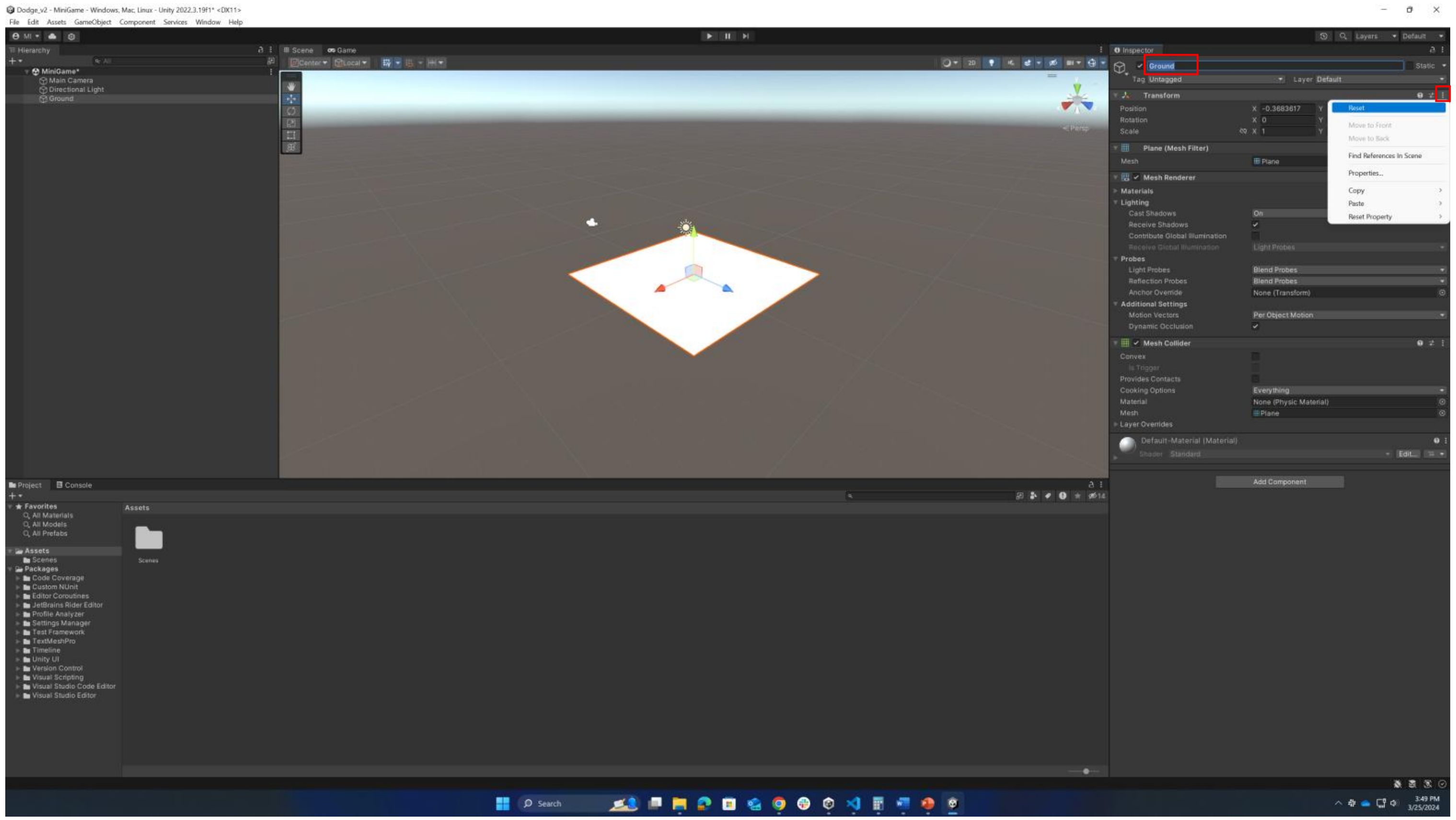
- Create a red cube object. Name it **Player**.
- Place it above the sphere.
- The cube should fall on top of the sphere when we hit play.

Mini Challenge: 2 Minutes

- Add left and right movement functionality to the player. The applied sideways force should be 100f.

Create a primitive plane

- **Reset the position of the Plane.**
  - With the **Ground** GameObject selected, in the upper-right corner of the Transform component, select the vertical **More** (⋮) menu .
  - Select **Reset** for the **Transform** component of the **Ground** GameObject. This action places the GameObject at the origin point (0, 0, 0) in the scene.



Create a primitive plane

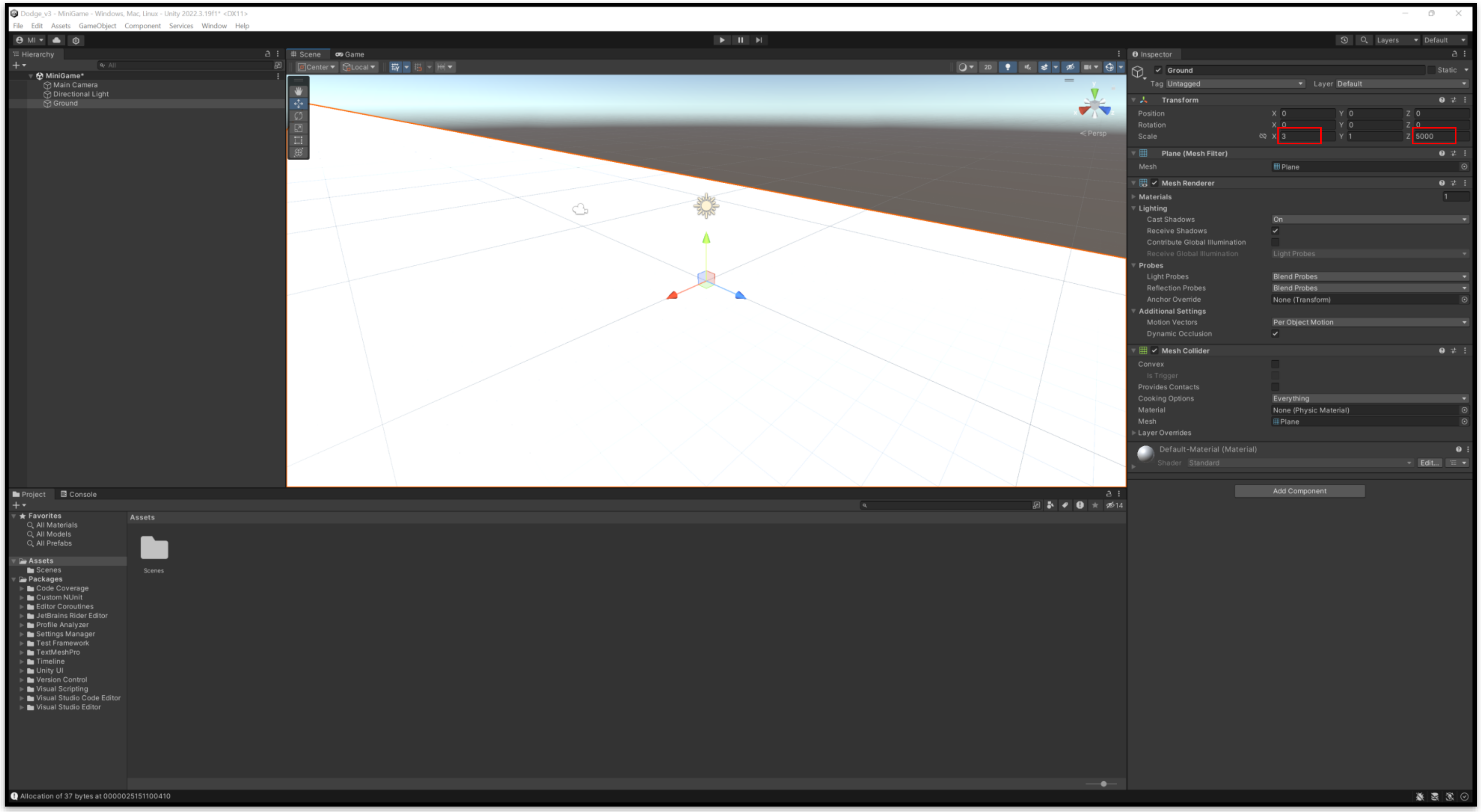
- **Frame the Plane in the Scene view.**
  - Ensure the **Ground** GameObject is selected and position the cursor in the **Scene** view.
  - Press the **F** key to frame the entire GameObject nicely within the **Scene** view.
  - Alternatively, select **Edit** > **Frame Selected** from the main menu.

Scale the Ground Plane

- **Increase the scale of the Ground Plane.**
  - With the **Ground** GameObject selected, activate the **Scale** tool by pressing the **R** hotkey.
  - We can drag the **X** (red) and **Z** (blue) handles to increase the size of the Plane.

- **Set precise scale values**
  - In the **Transform** component, set the **X** and **Z Scale** values to **3** and **5000** respectively.

Scale the Ground Plane



Dodge_v3 - MiniGame - Windows, Mac, Linux - Unity 2022.3.19f1* <DX11>

MiniGame*
- Main Camera
- Directional Light
- Ground

Inspector

Ground — Tag: Untagged — Layer: Default

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	3	1	5000

Plane (Mesh Filter)

Mesh	Plane

Mesh Renderer

Materials	1
Lighting	
Cast Shadows	On
Receive Shadows	✓
Contribute Global Illumination	
Receive Global Illumination	Light Probes
Probes	
Light Probes	Blend Probes
Reflection Probes	Blend Probes
Anchor Override	None (Transform)
Additional Settings	
Motion Vectors	Per Object Motion
Dynamic Occlusion	✓

Mesh Collider

Convex	
Is Trigger	
Provides Contacts	
Cooking Options	Everything
Material	None (Physic Material)
Mesh	Plane
Layer Overrides	

Default-Material (Material) — Shader: Standard

Allocation of 37 bytes at 0000025151100410

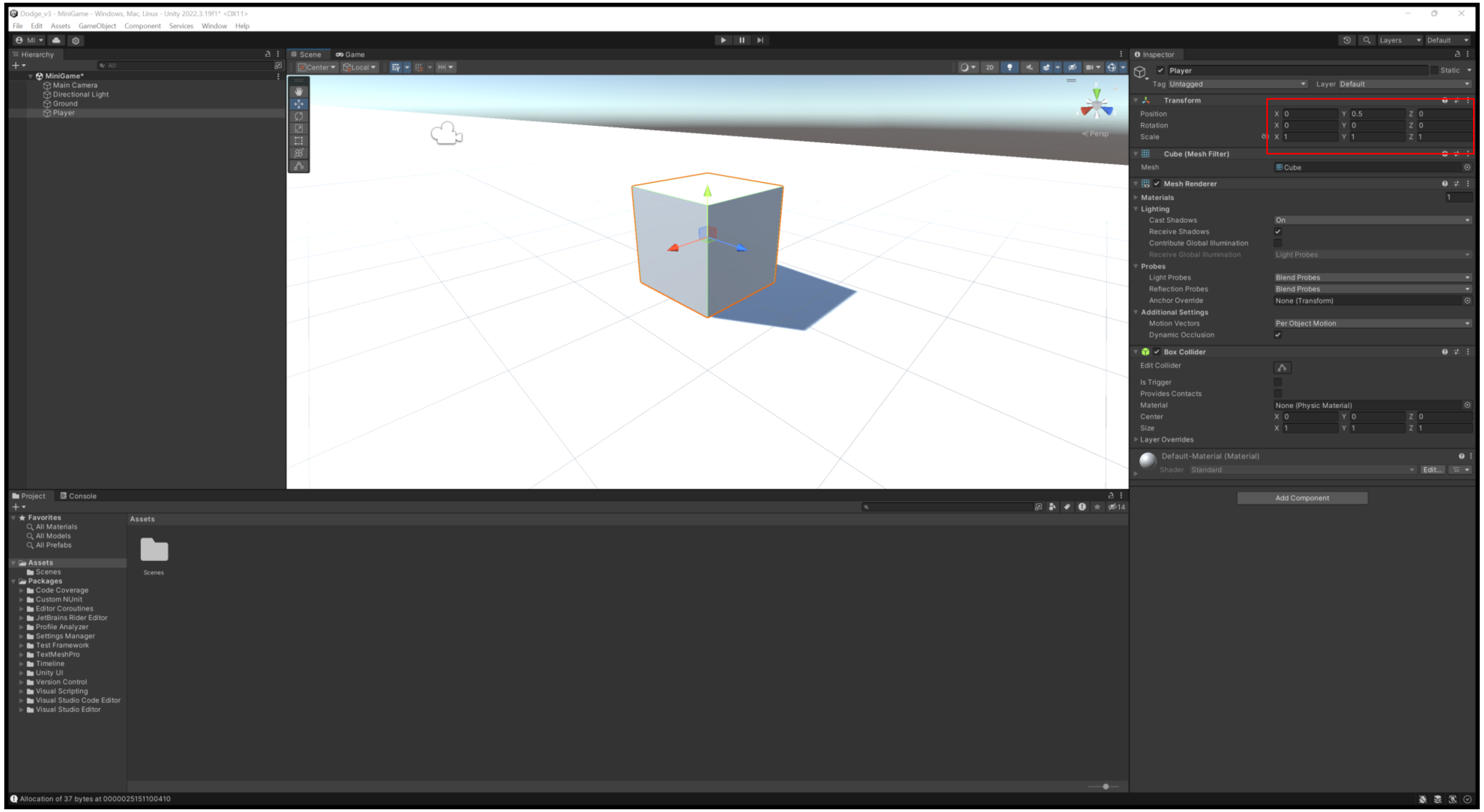
Create a Player GameObject

- **Create a Player cube.**
  - In the **Hierarchy** window, **right-click** > **3D Object** > **Cube.**
  - Rename the newly created **Cube** GameObject "Player".

- **Position the Player Sphere at the origin.**
  - In the **Inspector** window, reset the **Transform** component of the **Player** GameObject to position it at the origin point (0, 0, 0) of the scene.
  - Press the **F** key in the **Scene** view to frame the **Player** GameObject in the **Scene** view.

- **Move the Player GameObject up to sit on the Plane.**
  - In the **Transform** component for the **Player** GameObject, set the **Y Position** value to **0.5**.

Create a Player GameObject



Player — Static
Tag: Untagged — Layer: Default

Transform
Position X 0 Y 0.5 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Cube (Mesh Filter)
Mesh: Cube

Mesh Renderer
Materials: 1
Lighting
Cast Shadows: On
Receive Shadows: ✓
Contribute Global Illumination: ☐
Receive Global Illumination: Light Probes
Probes
Light Probes: Blend Probes
Reflection Probes: Blend Probes
Anchor Override: None (Transform)
Additional Settings
Motion Vectors: Per Object Motion
Dynamic Occlusion: ✓

Box Collider
Edit Collider
Is Trigger: ☐
Provides Contacts: ☐
Material: None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1
Layer Overrides

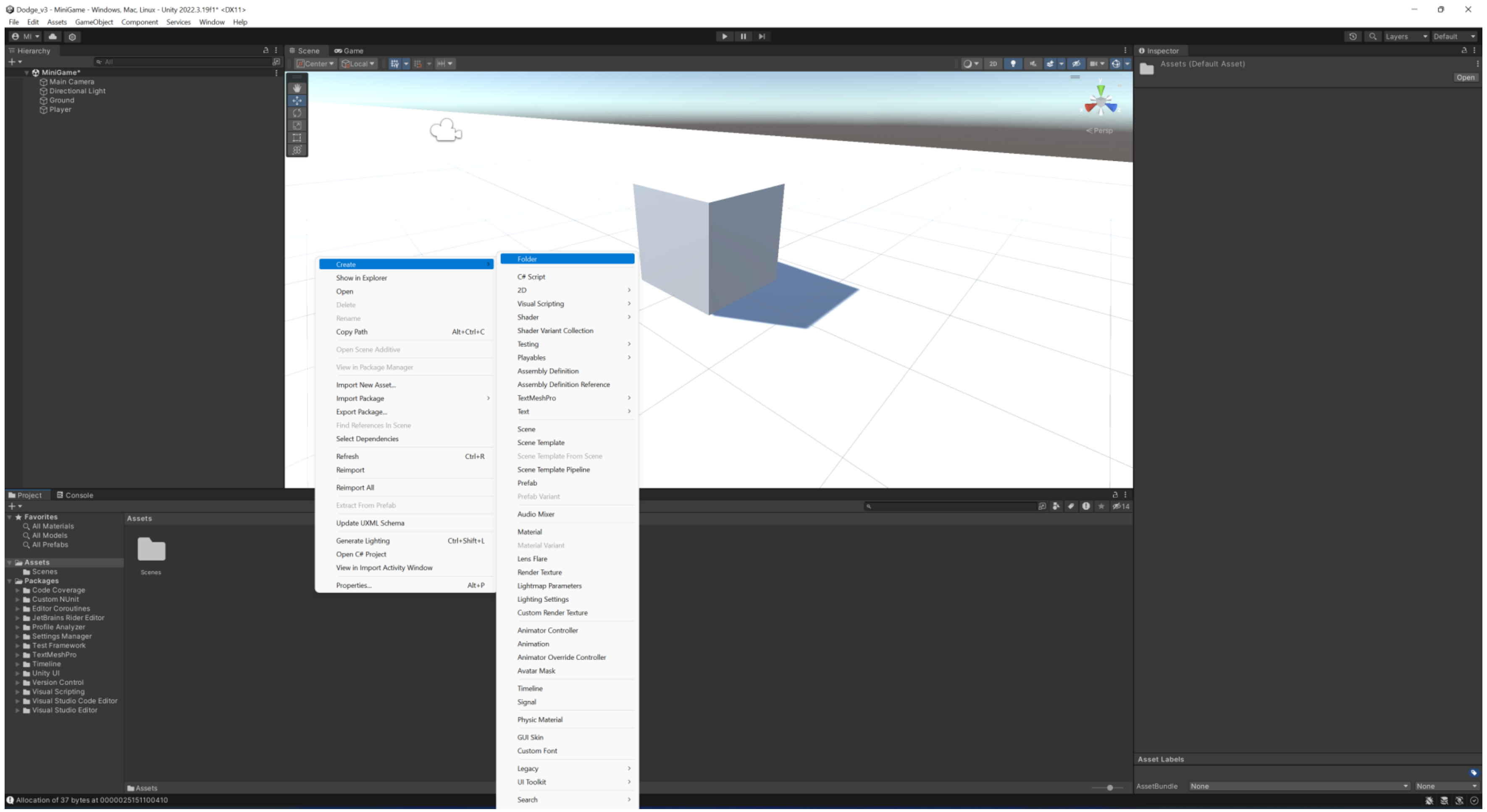
Default-Material (Material)
Shader: Standard

Allocation of 37 bytes at 0000025151100410

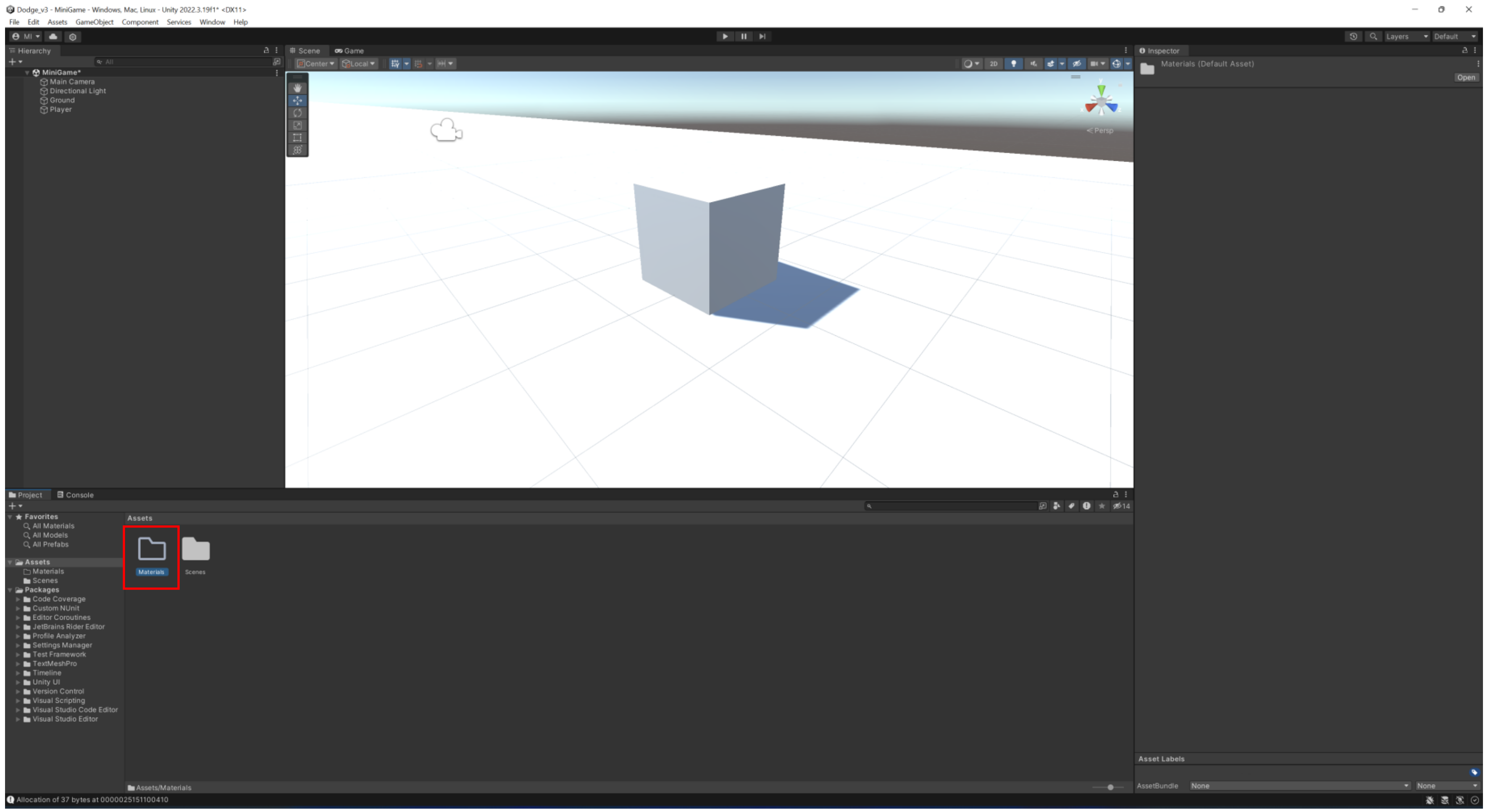
Add colors with Materials

- **Create a new Materials folder.**
  - In the **Project** window, **right-click** > **Create** > **Folder** to make a new folder.
  - Rename the new folder "Materials".

Add colors with Materials



Add colors with Materials

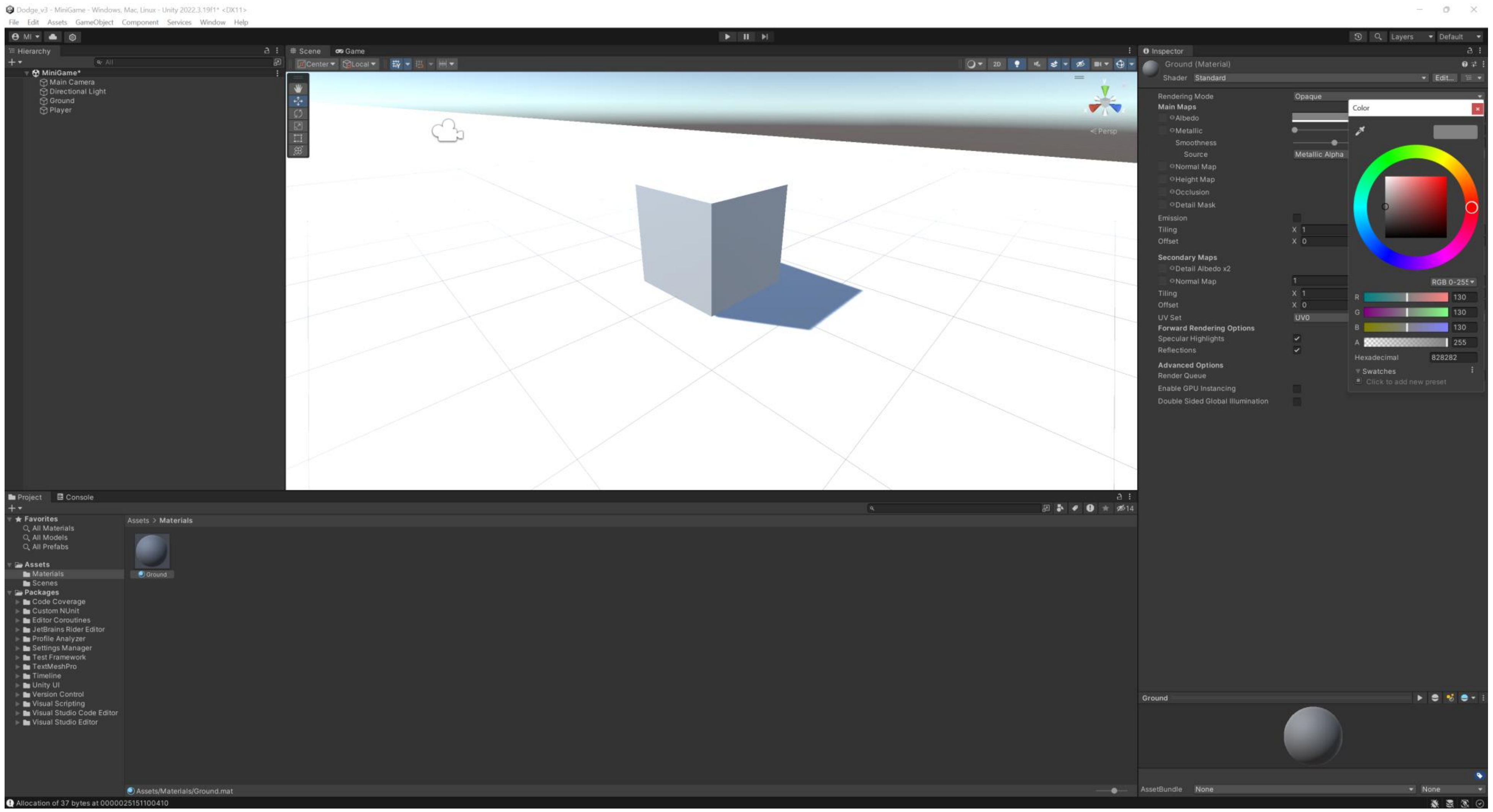


Allocation of 37 bytes at 0000025151100410

Add colors with Materials

- **Create a new Background material.**
  - In the newly created **Materials** folder, **right-click** > **Create** > **Material** to make a new material, then name it "Ground".
  - In the **Inspector** window, use the foldout (triangle) to expand the **Surface Inputs** module, and select the **Base Map** color picker.
  - Change the color to a pale gray with RGB values of **130**, **130**, and **130**.
  - Make sure the **Metallic Map** is set to **0** and the **Smoothness** is set to around **0.25** for a matte finish.
  - Apply the **Ground** material to the **Ground** GameObject by dragging it from the **Project** window onto the Ground GameObject in the **Scene** view.

Add colors with Materials

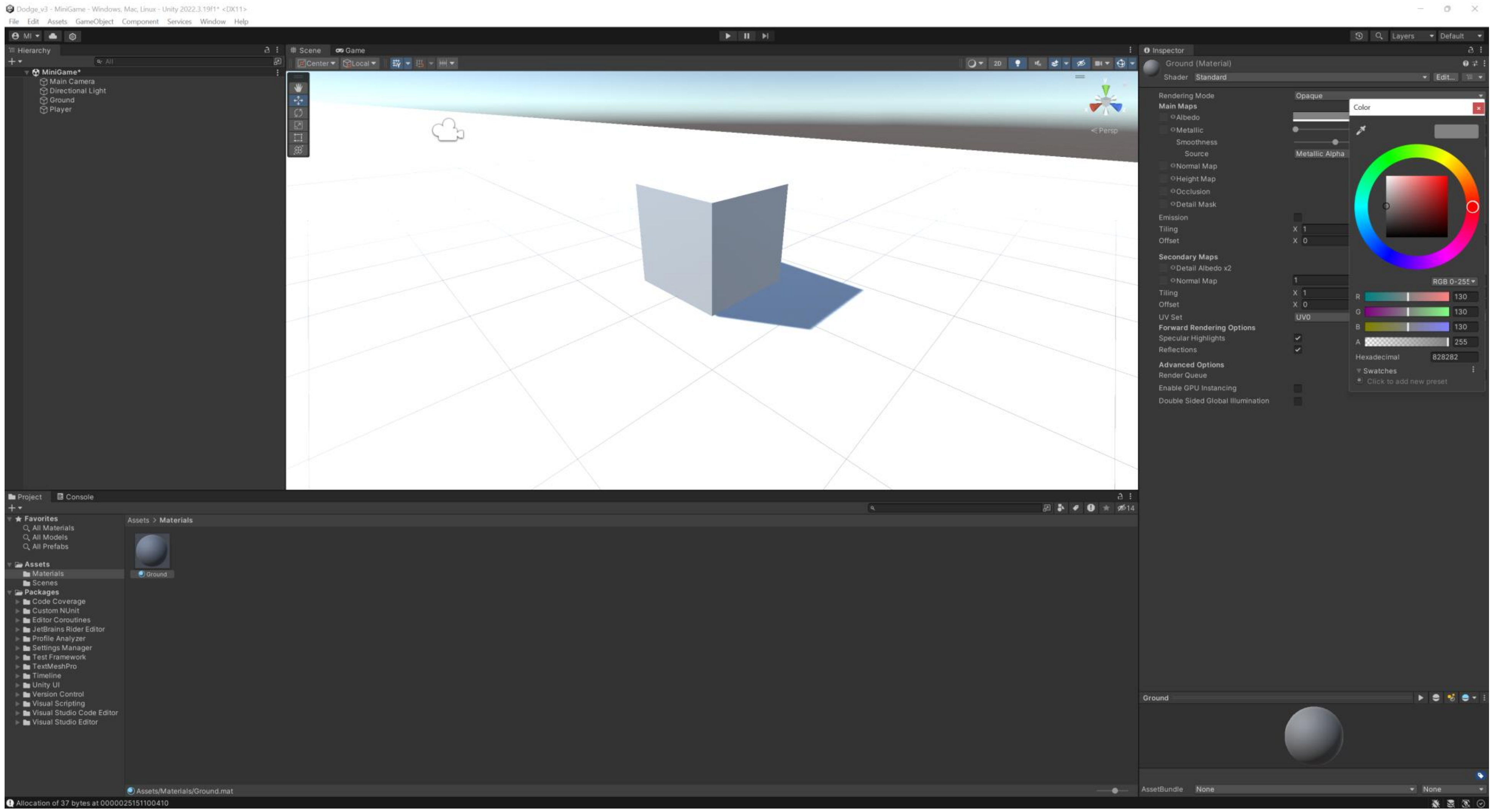


Ground (Material)
Shader: Standard
Rendering Mode: Opaque
Main Maps
Albedo
Metallic
Smoothness
Source: Metallic Alpha
Normal Map
Height Map
Occlusion
Detail Mask
Emission
Tiling: X 1
Offset: X 0
Secondary Maps
Detail Albedo x2
Normal Map: 1
Tiling: X 1
Offset: X 0
UV Set: UV0
Forward Rendering Options
Specular Highlights ✓
Reflections ✓
Advanced Options
Render Queue
Enable GPU Instancing
Double Sided Global Illumination

RGB 0-255
R 130
G 130
B 130
A 255
Hexadecimal 828282

Assets/Materials/Ground.mat
Allocation of 37 bytes at 0000025151100410

Add colors with Materials


Dodge_v3 - MiniGame - Windows, Mac, Linux - Unity 2022.3.19f1* <DX11>

File Edit Assets GameObject Component Services Window Help

Ground (Material)

Shader Standard

Rendering Mode Opaque

Main Maps

Albedo

Metallic

Smoothness

Source Metallic Alpha

Normal Map

Height Map

Occlusion

Detail Mask

Emission

Tiling X 1

Offset X 0

Secondary Maps

Detail Albedo x2

Normal Map 1

Tiling X 1

Offset X 0

UV Set UV0

Forward Rendering Options

Specular Highlights ✓

Reflections ✓

Advanced Options

Render Queue

Enable GPU Instancing

Double Sided Global Illumination

Color

RGB 0-255

R 130

G 130

B 130

A 255

Hexadecimal 828282

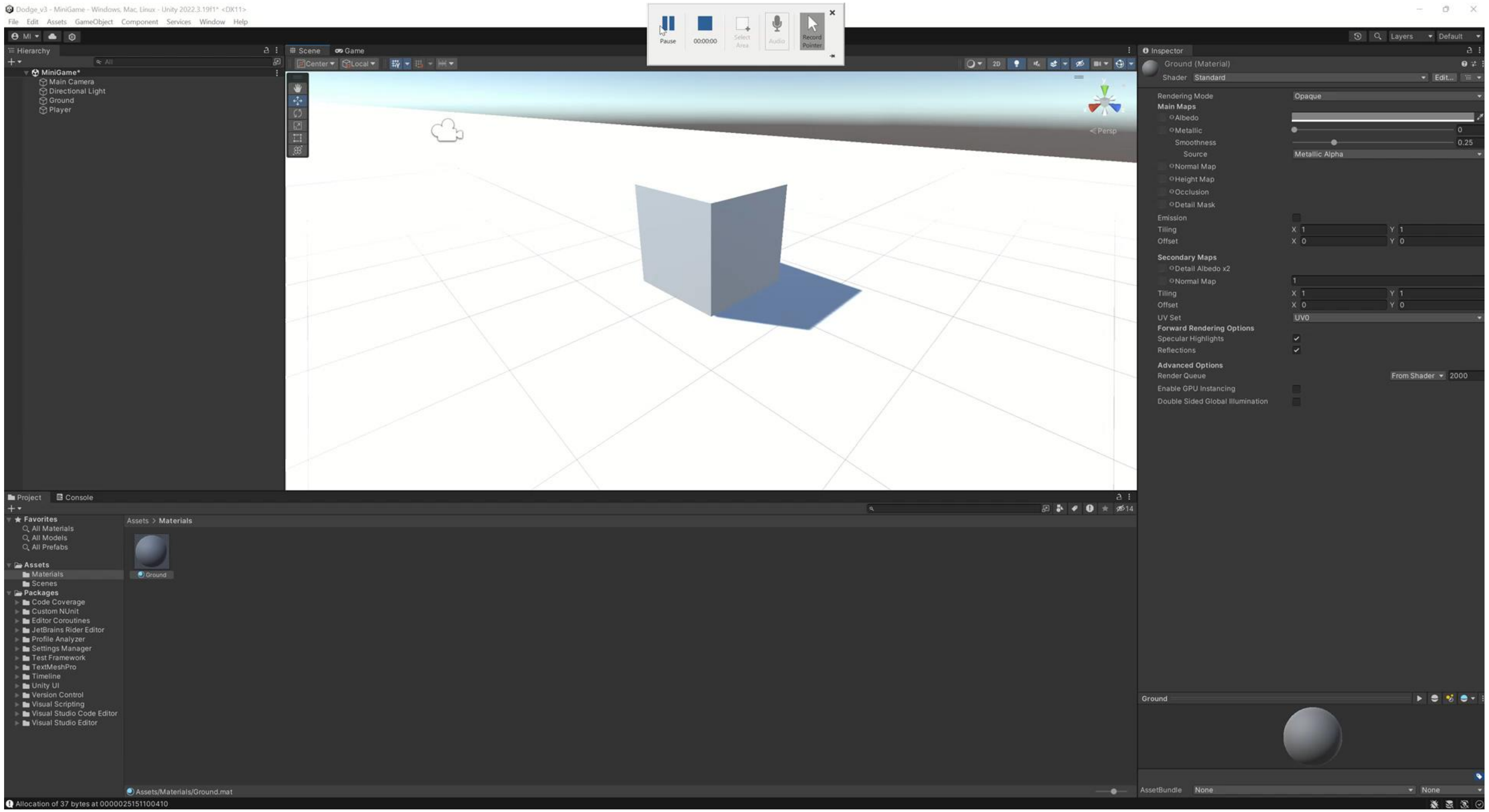
Swatches

Click to add new preset

Assets/Materials/Ground.mat

Allocation of 37 bytes at 0000025151100410

Add colors with Materials



Ground (Material)
Shader Standard

Rendering Mode Opaque
Main Maps
Albedo
Metallic 0
Smoothness 0.25
Source Metallic Alpha
Normal Map
Height Map
Occlusion
Detail Mask
Emission
Tiling X 1 Y 1
Offset X 0 Y 0

Secondary Maps
Detail Albedo x2
Normal Map 1
Tiling X 1 Y 1
Offset X 0 Y 0
UV Set UV0
Forward Rendering Options
Specular Highlights ✓
Reflections ✓

Advanced Options
Render Queue From Shader 2000
Enable GPU Instancing
Double Sided Global Illumination

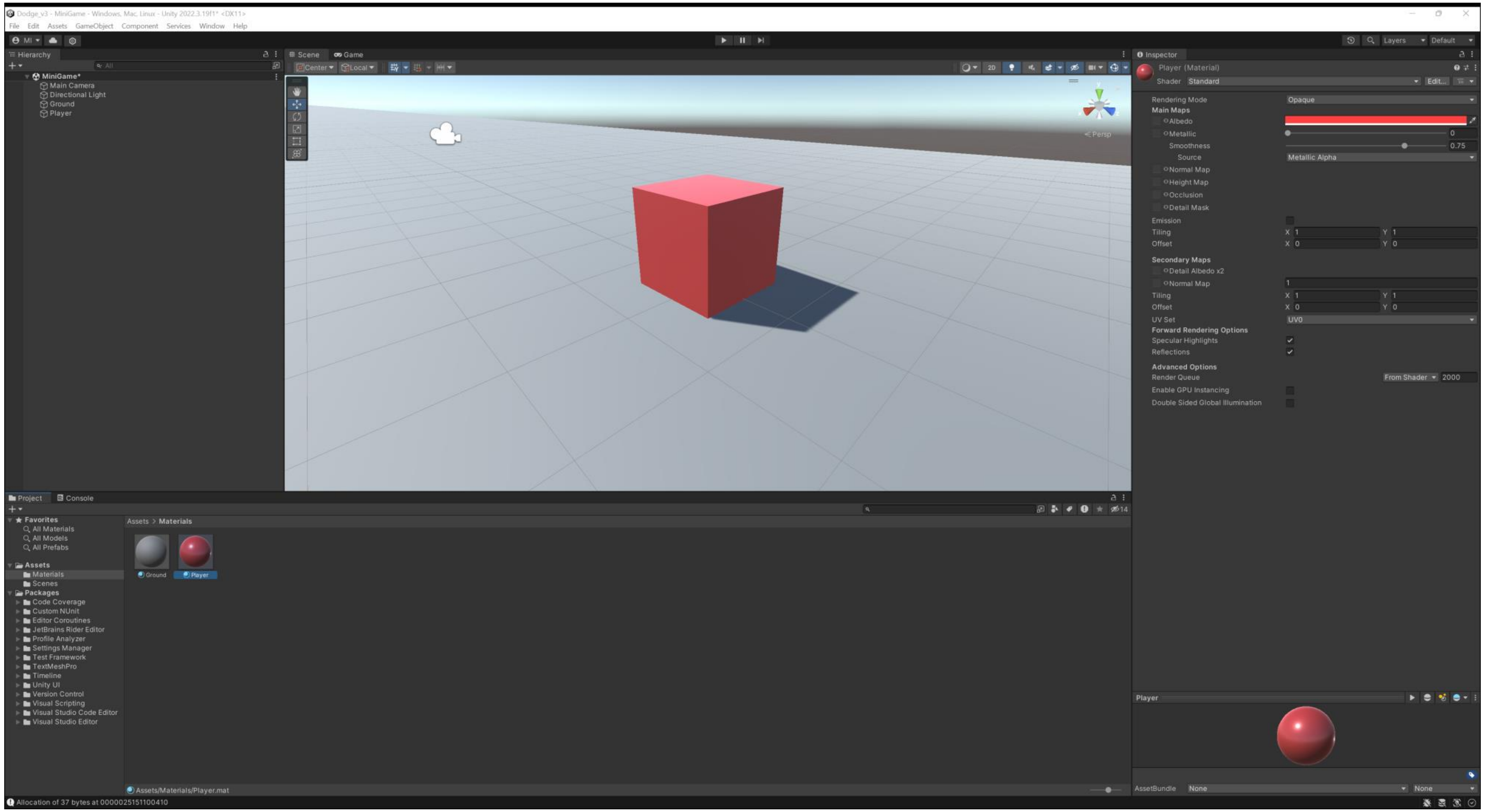
Assets/Materials/Ground.mat

Allocation of 37 bytes at 0000025151100410

Add colors with Materials

- **Creating a new Player Material.**
  - In the **Materials** folder, create a new material and name it "Player".
  - In the **Inspector** window, adjust the **Base Map** color for the **Player** material — set the RGB values to **255**, **65**, and **65** for a matte red.
  - Set the **Metallic Map** to **0** and change the **Smoothness** to **0.75** for a shiny finish.
  - Apply the **Player** material to the **Player** GameObject by dragging it onto the sphere in the **Scene** view.

Add colors with Materials



Player (Material)
Shader: Standard

Rendering Mode: Opaque
Main Maps
Albedo: 🟥
Metallic: 0
Smoothness: 0.75
Source: Metallic Alpha
Normal Map
Height Map
Occlusion
Detail Mask
Emission: ☐
Tiling: X 1 Y 1
Offset: X 0 Y 0
Secondary Maps
Detail Albedo x2
Normal Map: 1
Tiling: X 1 Y 1
Offset: X 0 Y 0
UV Set: UV0
Forward Rendering Options
Specular Highlights: ✓
Reflections: ✓
Advanced Options
Render Queue: From Shader 2000
Enable GPU Instancing: ☐
Double Sided Global Illumination: ☐

Player
AssetBundle: None None

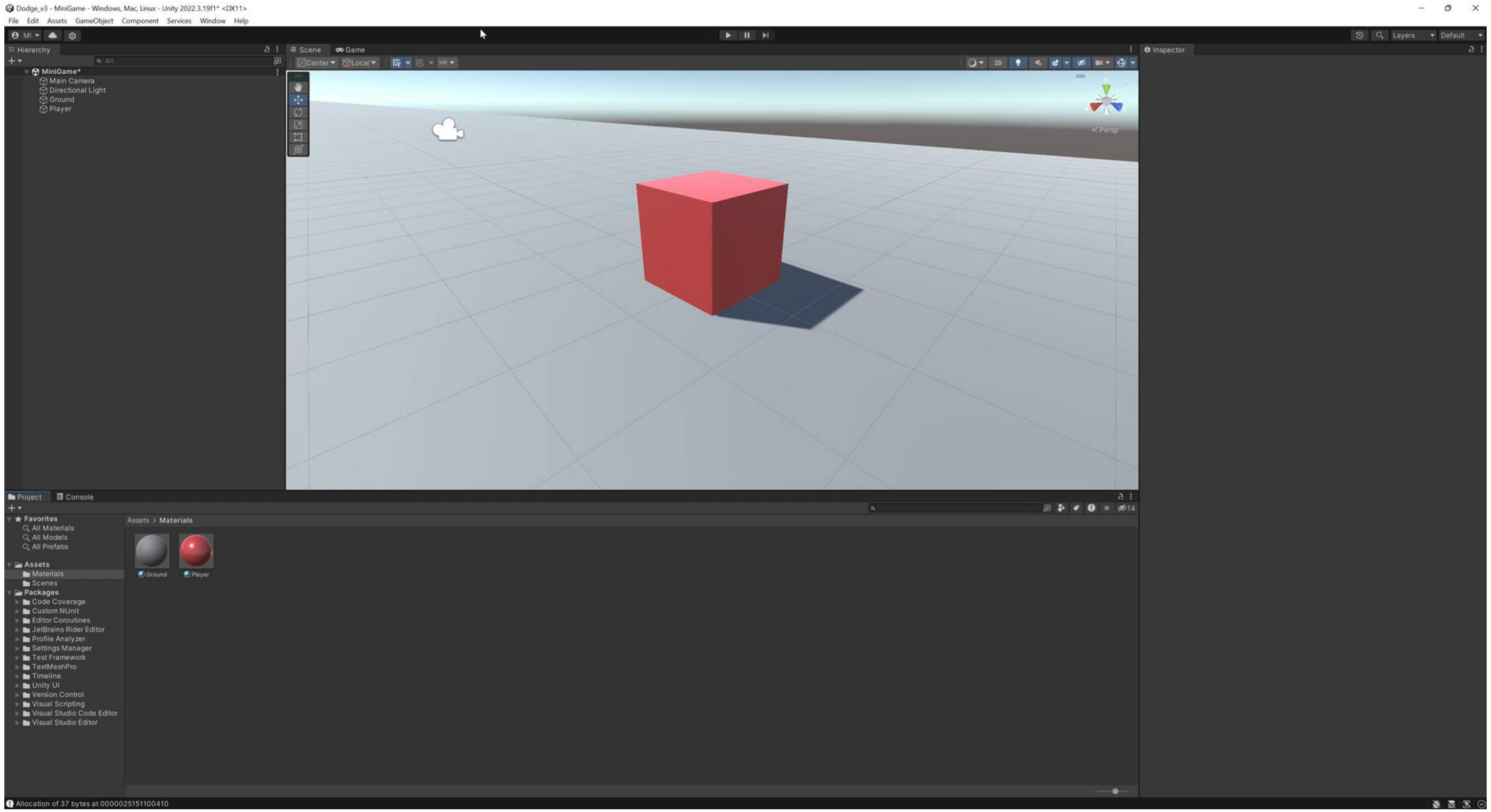
Assets/Materials/Player.mat

Allocation of 37 bytes at 0000025151100410

Add colors with Materials

- **Dock the Game View to the right of Scene View.**
  - Left click and hold the game view tab. Move it to the right of the Scene view so that we can see both tabs side by side.

Add colors with Materials

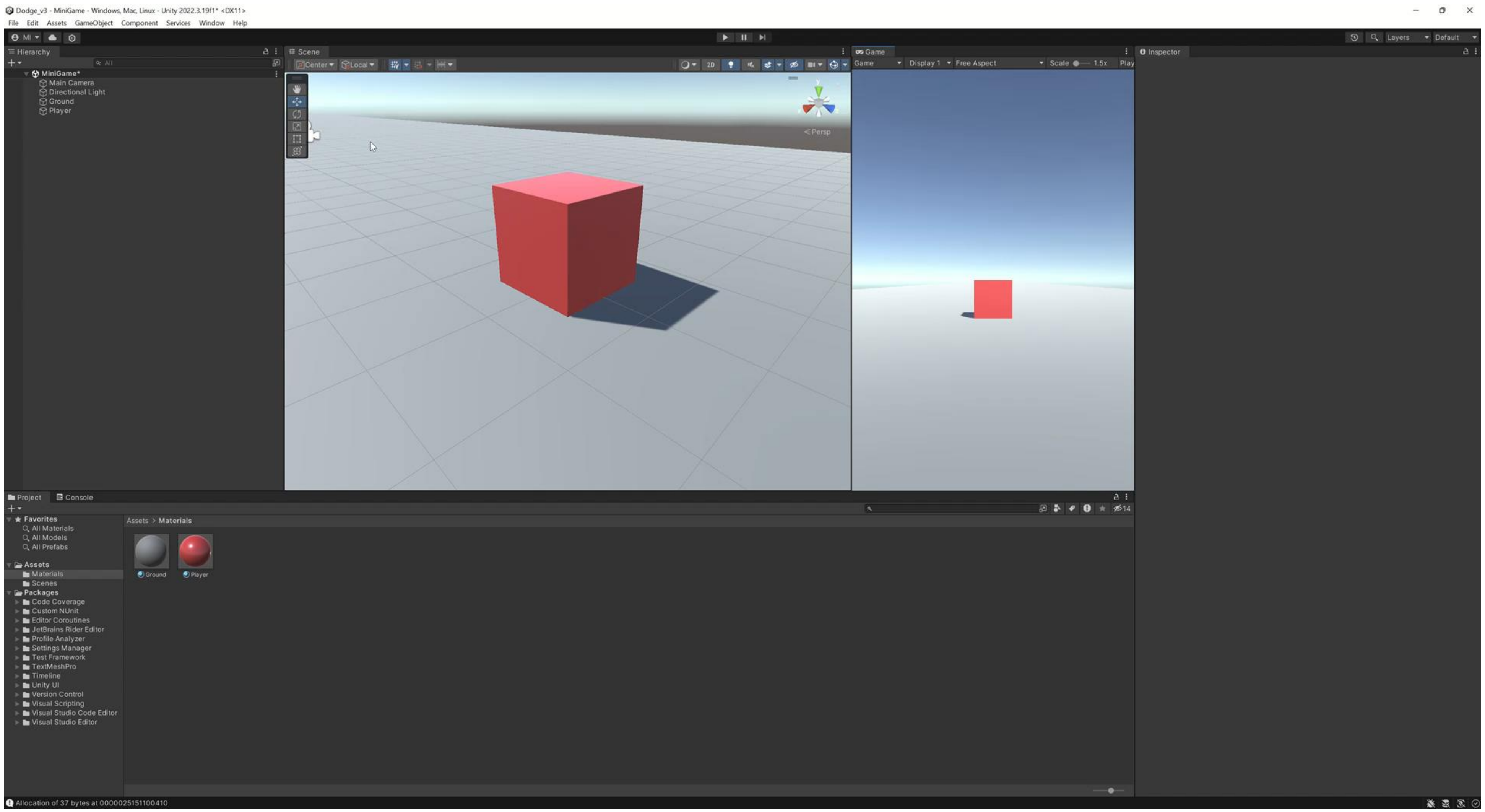


Allocation of 37 bytes at 0000025151100410

Add colors with Materials

- **Change the skybox.**
  - In the **Hierarchy** window, select the **Main Camera**.
  - In the **Inspector** tab, change the skybox of the Camera component to **Solid Color**. set the RGB values of Background to **0**, **220**, and **255** for a light blue.

Add colors with Materials



Allocation of 37 bytes at 0000025151100410

Section

Moving the Player

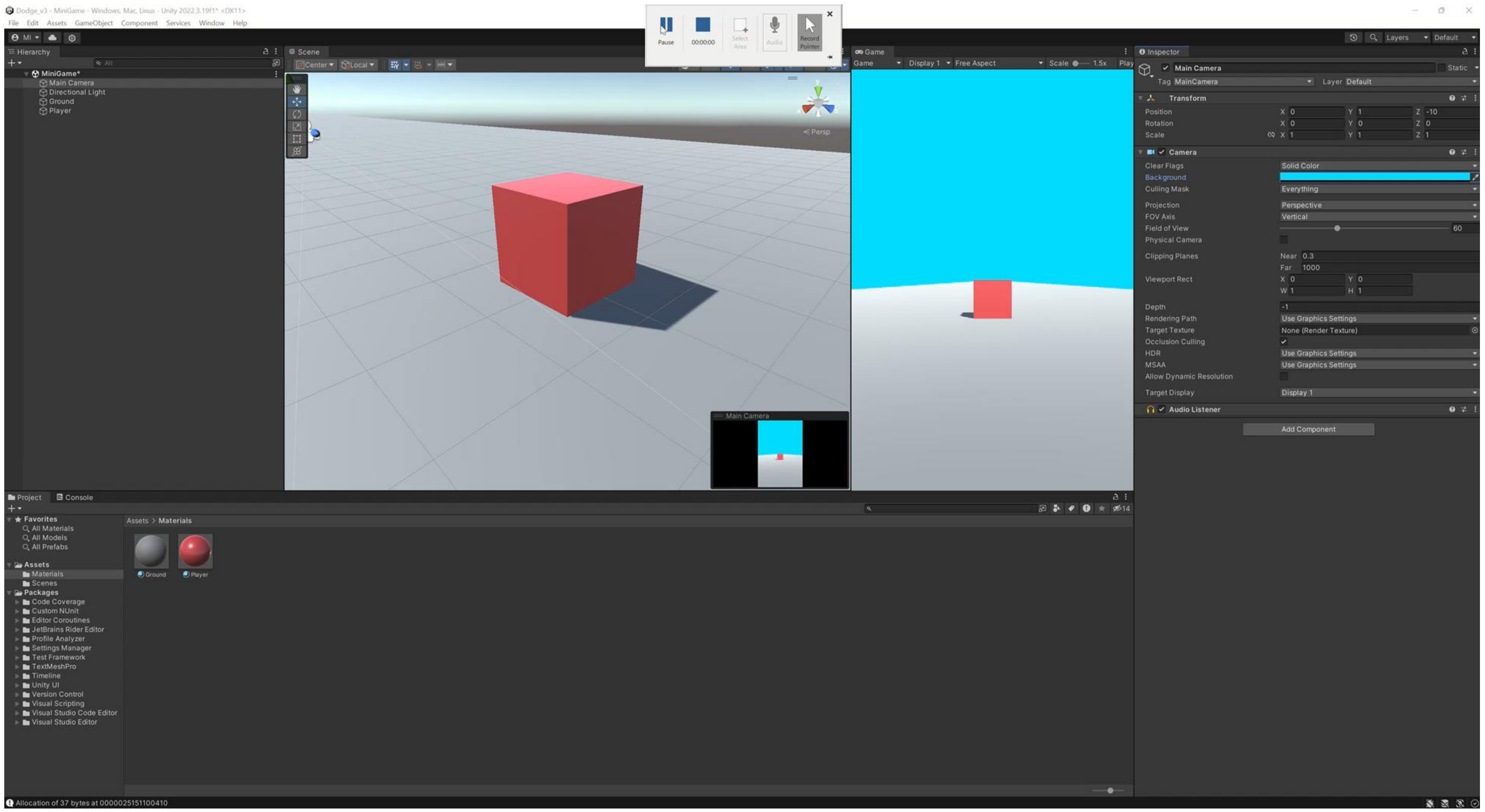
Moving the Player

- (Note: Make sure the **play button** is **unclicked**. Otherwise, all the changes we make will be lost.)

Add a Rigidbody to the Player

- **Add a Rigidbody component.**
  - Select the **Player** GameObject in the **Hierarchy** window.
  - In the **Inspector** Window, select **Add Component**, then search for "Rigidbody" and add the **Rigidbody** component to the **Player** GameObject.
- **Note**: Make sure to select **Rigidbody** and not **Rigidbody 2D**.

Add a Rigidbody to the Player



Inspector

Main Camera — Static
Tag MainCamera — Layer Default

Transform

Position	X 0	Y 1	Z -10
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Camera

Clear Flags	Solid Color
Background	🟦
Culling Mask	Everything
Projection	Perspective
FOV Axis	Vertical
Field of View	60
Physical Camera	☐
Clipping Planes	Near 0.3 / Far 1000
Viewport Rect	X 0 Y 0 / W 1 H 1
Depth	-1
Rendering Path	Use Graphics Settings
Target Texture	None (Render Texture)
Occlusion Culling	✓
HDR	Use Graphics Settings
MSAA	Use Graphics Settings
Allow Dynamic Resolution	☐
Target Display	Display 1

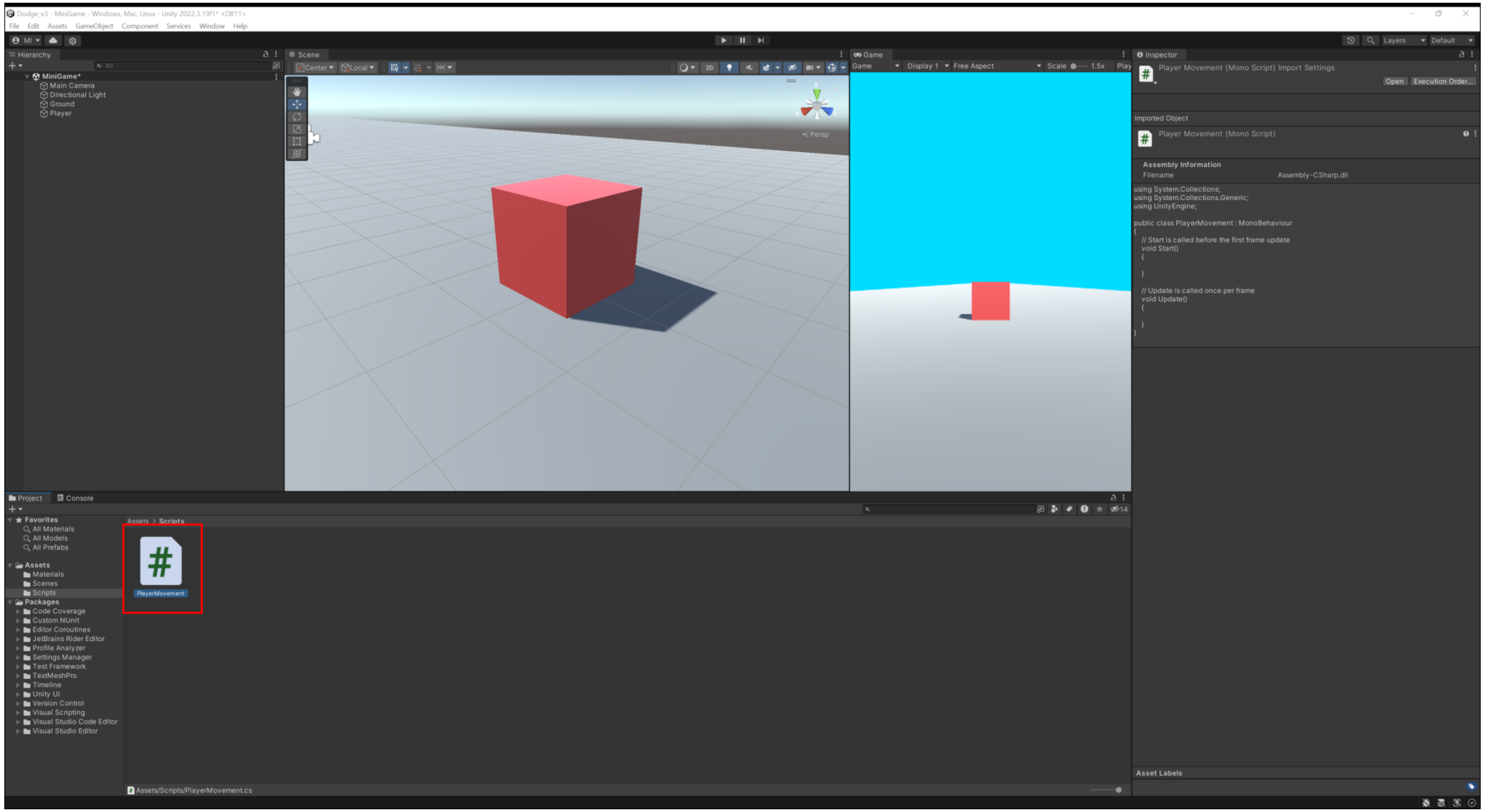
Audio Listener

Allocation of 37 bytes at 0000025151100410

Create a new script

- **Create a new PlayerMovement script.**
  - In the **Project** window, create a new folder named "Scripts".
  - With the **Player** GameObject selected, select **Add Component** > **New Script,** then name the new script "PlayerMovement".
  - The created script asset will be at the root level of the **Assets** folder by default. Move the new **PlayerMovement** script asset into the **Scripts** folder.

Create a new script



Player Movement (Mono Script) Import Settings

Imported Object

Player Movement (Mono Script)

Assembly Information

Filename: Assembly-CSharp.dll

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

public class PlayerMovement : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}
```

Create a new script

- **Open the script in a script editor.**
  - Double-click the script asset in the **Project** window to open it in your preferred script editor, usually **VS Code**.
  - **Start() and Update() Functions:** These are special functions within a script that Unity calls automatically during the game's lifecycle.
    - **Start()** is called only once when the object is first created or enabled in the scene.
    - **Update()** is called repeatedly every frame, making it ideal for continuously updating the object's behavior throughout the game.

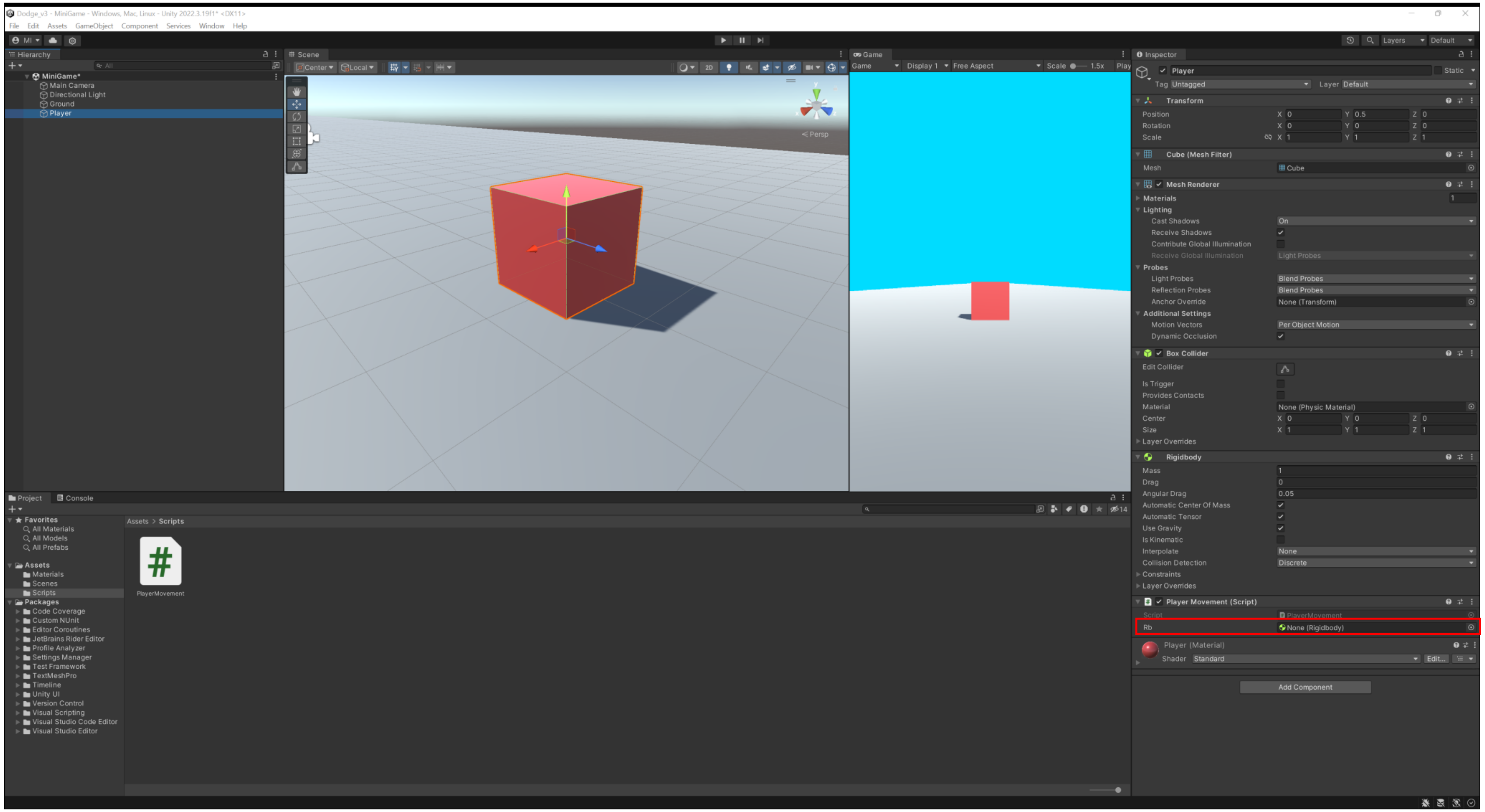
Assign a new Rigidbody variable

- Above the **Start** function, add the following line of code to declare a new variable and save the script (Ctrl + S):

```
public Rigidbody rb;
```

- Next, go to the Unity Editor. After the script gets compiled, you can see that the Player Movement component of the Player object has a Rb field added to it.

Assign a new Rigidbody variable



Player — Static
Tag: Untagged — Layer: Default

Transform
- Position: X 0, Y 0.5, Z 0
- Rotation: X 0, Y 0, Z 0
- Scale: X 1, Y 1, Z 1

Cube (Mesh Filter)
- Mesh: Cube

Mesh Renderer ✓
- Materials: 1
- Lighting
  - Cast Shadows: On
  - Receive Shadows: ✓
  - Contribute Global Illumination: ☐
  - Receive Global Illumination: Light Probes
- Probes
  - Light Probes: Blend Probes
  - Reflection Probes: Blend Probes
  - Anchor Override: None (Transform)
- Additional Settings
  - Motion Vectors: Per Object Motion
  - Dynamic Occlusion: ✓

Box Collider ✓
- Edit Collider
- Is Trigger: ☐
- Provides Contacts: ☐
- Material: None (Physic Material)
- Center: X 0, Y 0, Z 0
- Size: X 1, Y 1, Z 1
- Layer Overrides

Rigidbody
- Mass: 1
- Drag: 0
- Angular Drag: 0.05
- Automatic Center Of Mass: ✓
- Automatic Tensor: ✓
- Use Gravity: ✓
- Is Kinematic: ☐
- Interpolate: None
- Collision Detection: Discrete
- Constraints
- Layer Overrides

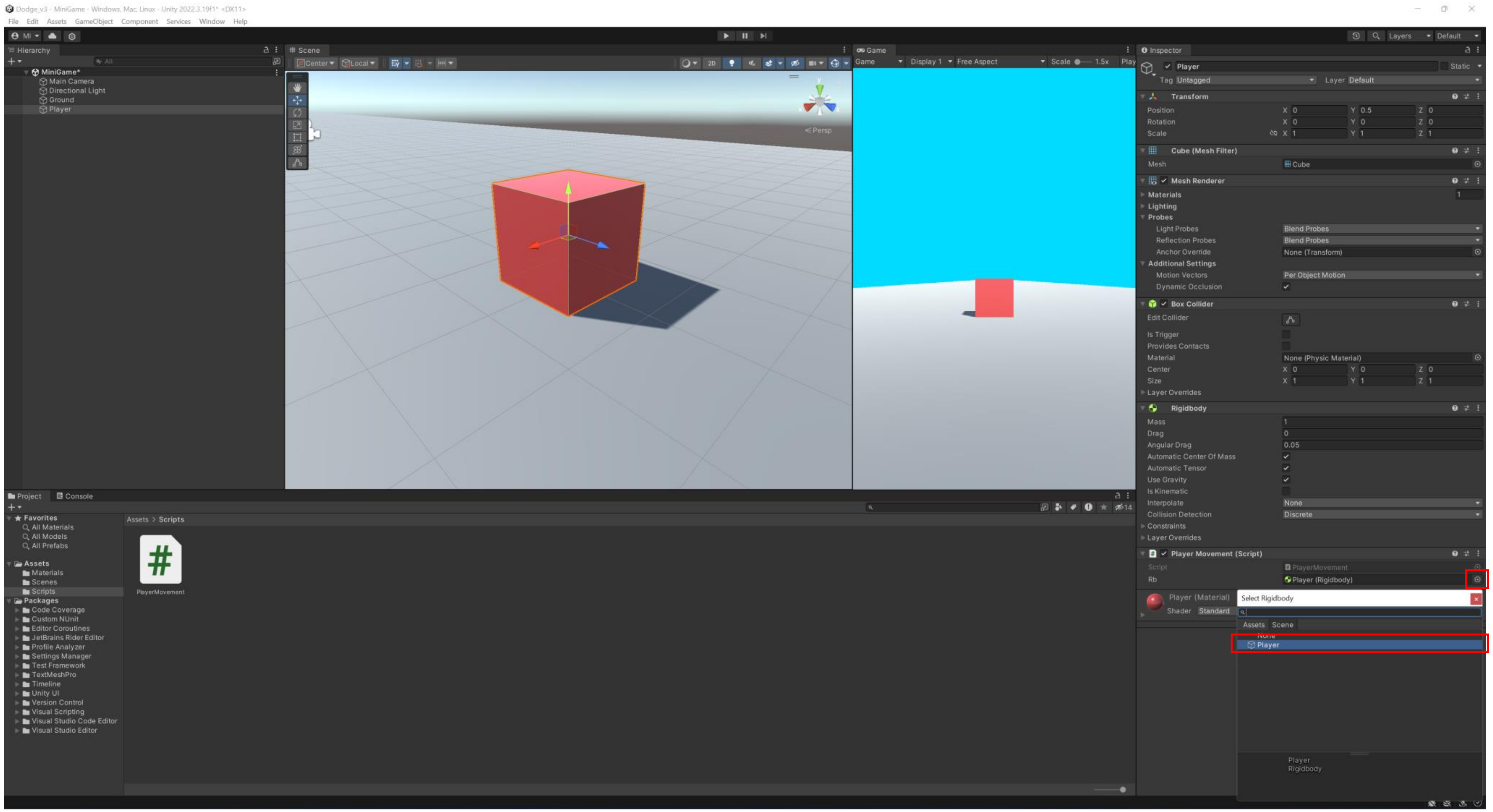
Player Movement (Script) ✓
- Script: PlayerMovement
- Rb: None (Rigidbody)

Player (Material)
Shader: Standard — Edit...

Referencing the Player's Rigidbody

- Click on the small round button on the right side of the Rb field and select Player from the list (or you can drag the Player object from the Hierarchy panel to the Rb field).
- This creates a reference to the Rigidbody component of the Player object from the rb variable in our PlayerMovement script.
- This allows us to use the rb variable to modify the Rigidbody of the Player.

Assign a new Rigidbody variable



Player — Tag Untagged — Layer Default

Transform
Position X 0 Y 0.5 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Cube (Mesh Filter)
Mesh: Cube

Mesh Renderer
Light Probes: Blend Probes
Reflection Probes: Blend Probes
Anchor Override: None (Transform)
Motion Vectors: Per Object Motion
Dynamic Occlusion: ✓

Box Collider
Material: None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1

Rigidbody
Mass: 1
Drag: 0
Angular Drag: 0.05
Automatic Center Of Mass: ✓
Automatic Tensor: ✓
Use Gravity: ✓
Is Kinematic:
Interpolate: None
Collision Detection: Discrete

Player Movement (Script)
Script: PlayerMovement
Rb: Player (Rigidbody)

Player (Material) — Shader Standard

Select Rigidbody
Assets Scene
None
Player

Player
Rigidbody

Apply force to the Player

- In our game, we want the player to move forward automatically and move the left or to the right when we press the left or right button on our keyboard respectively.

Apply force to the Player

- **Physics Control:** Let's explore some key properties and methods of the Rigidbody component:
  - **useGravity**: Enable or disable gravity for the object.
  - **AddForce**: Apply a force to the object, causing it to move in a specific direction.
  - **Time.deltaTime**: This value is used to ensure your force calculations are frame-rate independent, leading to smoother and more consistent physics behavior across different machines.

Apply force to the Player

- Add the following lines below **`public Rigidbody rb;`**:

```
public float forwardForce = 1000f;

public float sidewaysForce = 100f;
```

- (Note: We can change the values of these variables in the Unity editor later if needed.)

- We will not need the Start() function here. So we can remove it.

- In the **Update** function body, add the following code:

```
rb.AddForce(0f, 0f, forwardForce * Time.deltaTime);
```

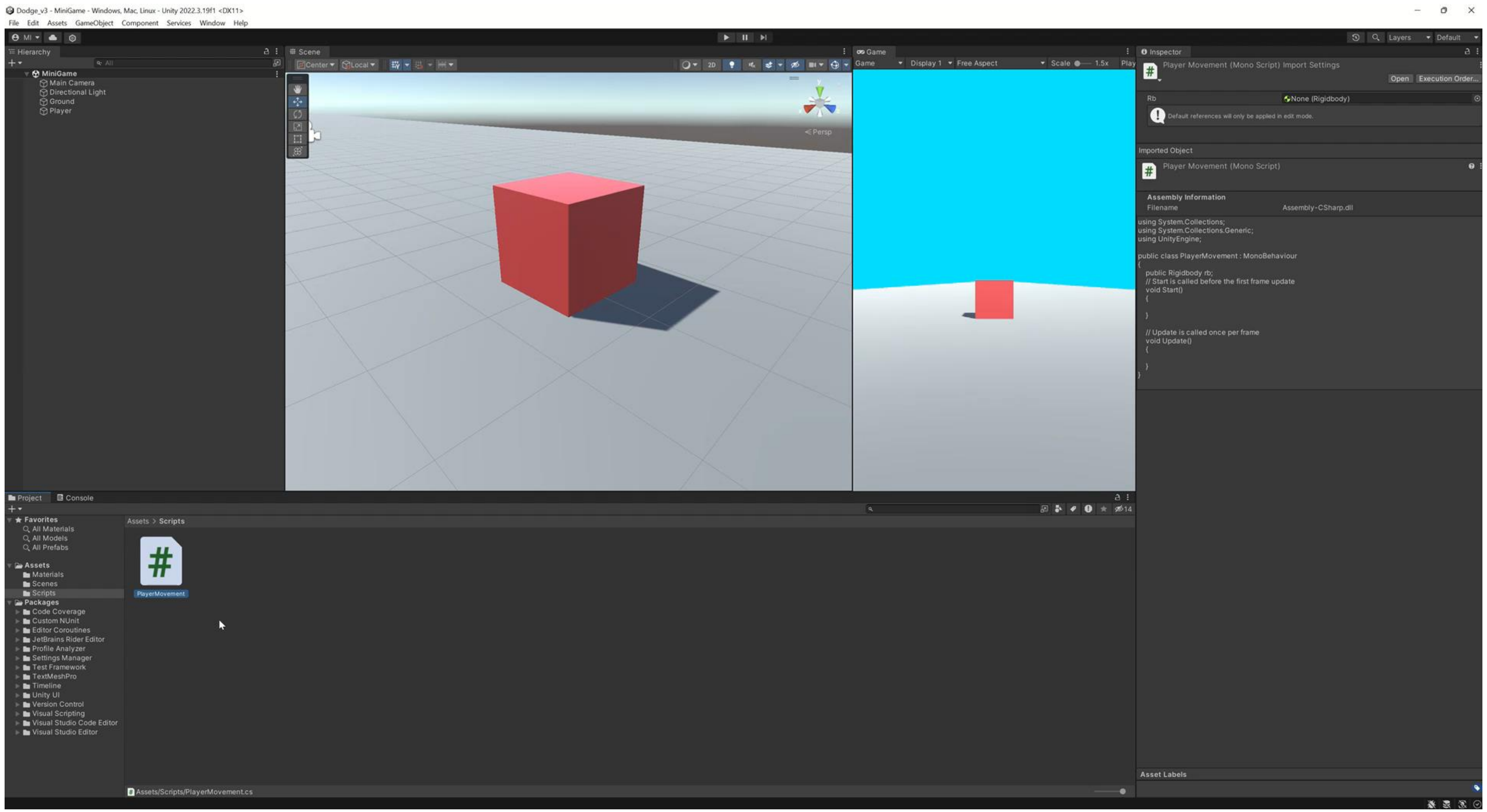
represent the force applied in the x, y and z directions respectively

Apply force to the Player

- If we now go to the Unity editor and click the play button, we will see the player moving forward.

- Make sure to unclick the play button afterward.

Apply force to the Player



Player Movement (Mono Script) Import Settings

Rb: None (Rigidbody)

Default references will only be applied in edit mode.

Imported Object

Player Movement (Mono Script)

Assembly Information
Filename: Assembly-CSharp.dll

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

public class PlayerMovement : MonoBehaviour
{
    public Rigidbody rb;
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}
```

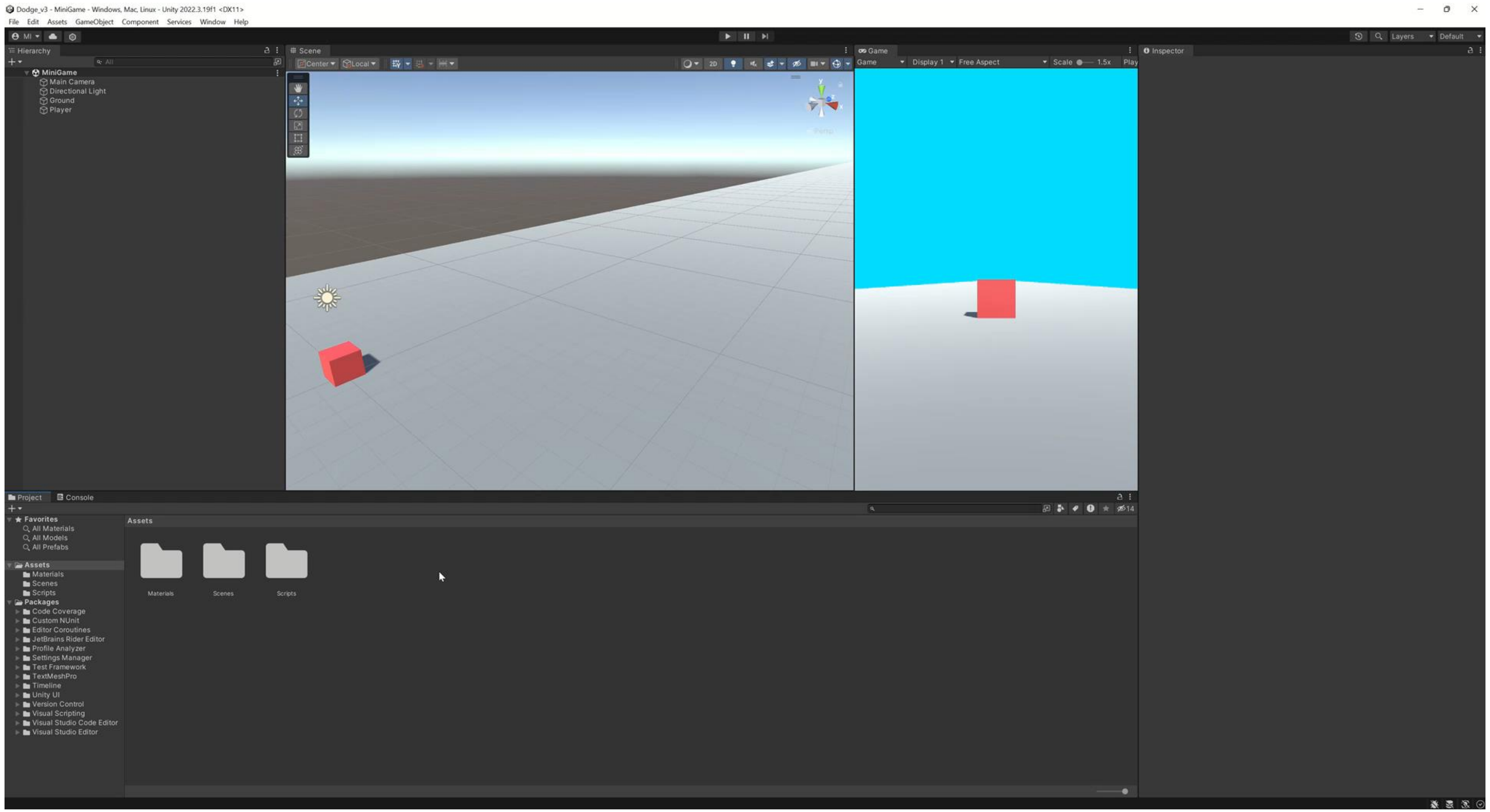
Apply force to the Player

- To address the uncontrolled spinning or rotating of the player object, follow these steps:

1. Create a Slippery Physic Material:
   - ➢ In the Project panel, make sure you are in the Asset folder
   - ➢ Right-click in the Project panel and choose "Create" > "Physic Material".
   - ➢ Name it “Slippery" and set both dynamic and static friction values to 0.

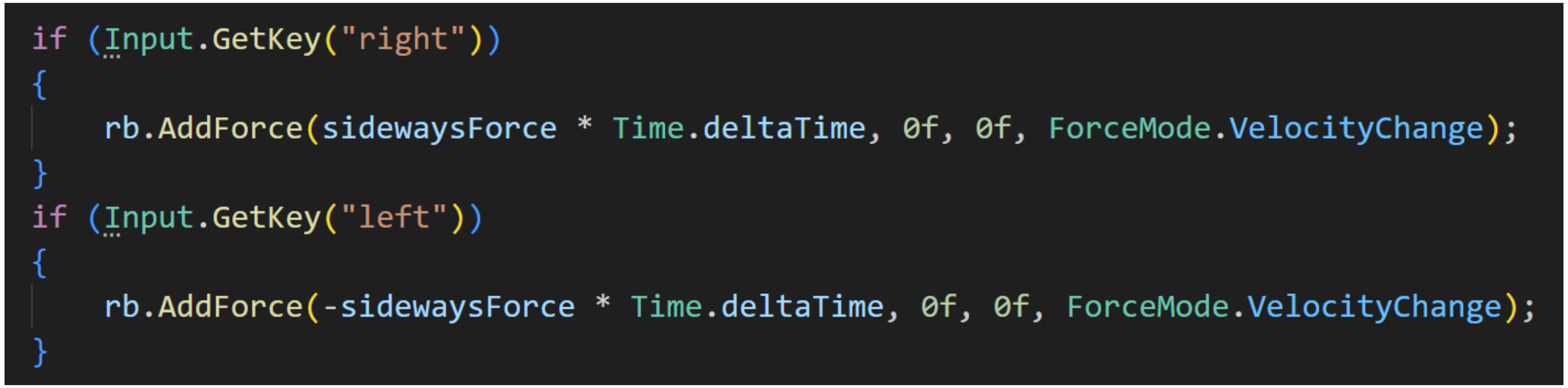
2. Apply the Material to the Ground:
   - ➢ Select the ground object in the Scene view.
   - ➢ Drag and drop the “Slippery" material onto the ground object.
   - ➢ This reduces friction between the player and the ground, allowing for smoother movement and preventing uncontrolled spinning.

Apply force to the Player



Add sideways movement

- To incorporate sideways movement to our player, add the following code segment in the Update function:



```
if (Input.GetKey("right"))
{
    rb.AddForce(sidewaysForce * Time.deltaTime, 0f, 0f, ForceMode.VelocityChange);
}
if (Input.GetKey("left"))
{
    rb.AddForce(-sidewaysForce * Time.deltaTime, 0f, 0f, ForceMode.VelocityChange);
}
```

Add sideways movement

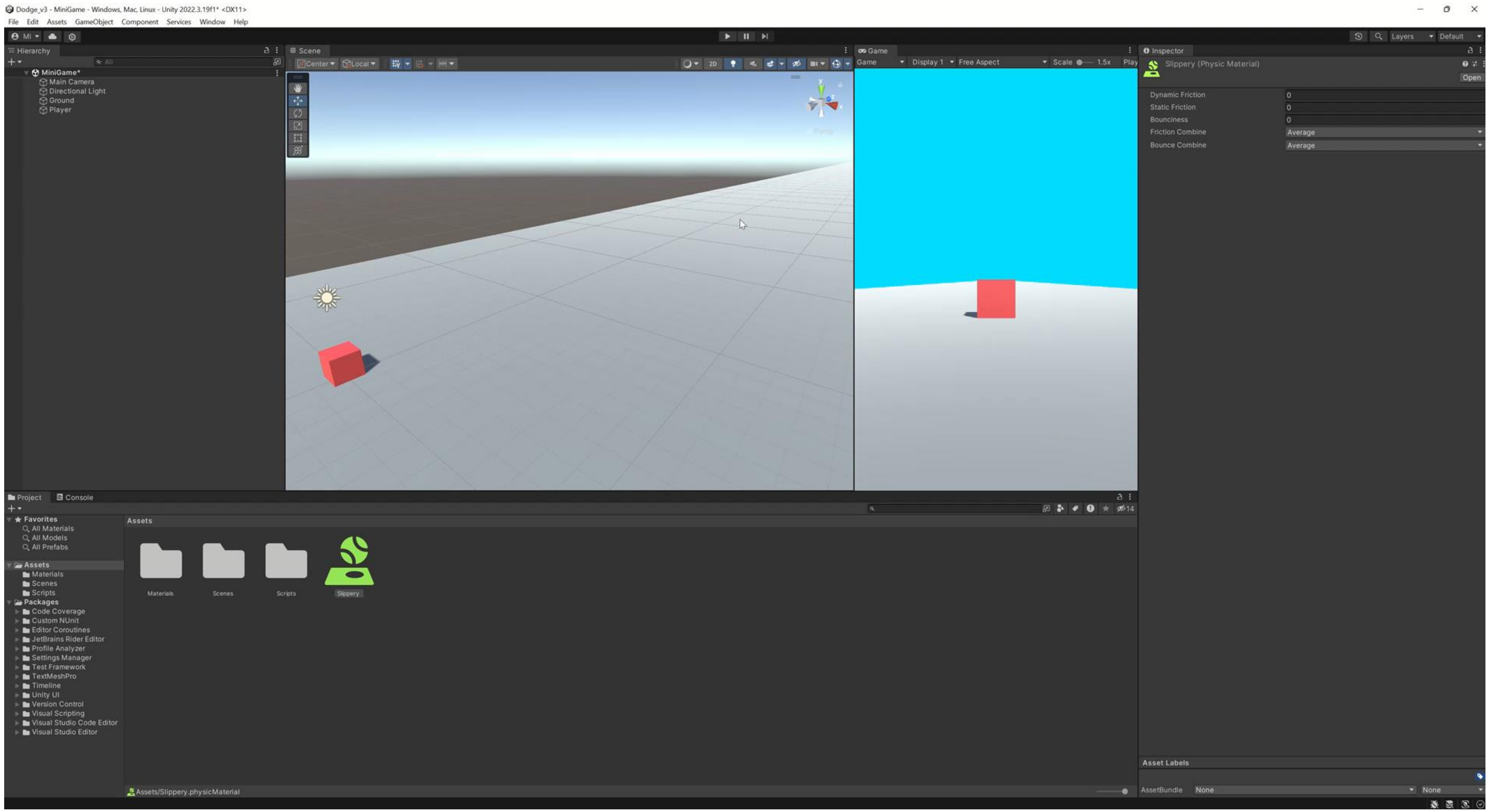
- To add sideways movement to our player, we've introduced a new variable **`sidewaysForce`** with a value of **`100f`**, which determines the strength of the sideways force.

- In the **`Update()`** function, we've added conditions to check if the 'right' or 'left' arrow keys are pressed. If the 'right' arrow key is pressed, we apply a force to the right using **`rb.AddForce(sidewaysForce * Time.deltaTime, 0f, 0f, ForceMode.VelocityChange);`**. Similarly, if the 'left' arrow key is pressed, we apply a force to the left using **`rb.AddForce(-sidewaysForce * Time.deltaTime, 0f, 0f, ForceMode.VelocityChange);`**.

Add sideways movement

- Here's what's happening in these lines:
  - **`Input.GetKey("right")`** and **`Input.GetKey("left")`** check if the 'right' or 'left' arrow keys are being pressed.
  - **`rb.AddForce()`** applies a force to the player object.
  - **`sidewaysForce * Time.deltaTime`** determines the strength of the sideways force applied.
  - The **`0f, 0f`** values in the **`AddForce()`** method mean there's no force applied in the vertical or forward/backward direction, ensuring the movement remains purely sideways.
  - **`ForceMode.VelocityChange`** ensures that the force is applied instantly, allowing for immediate movement in response to key presses."

Add sideways movement



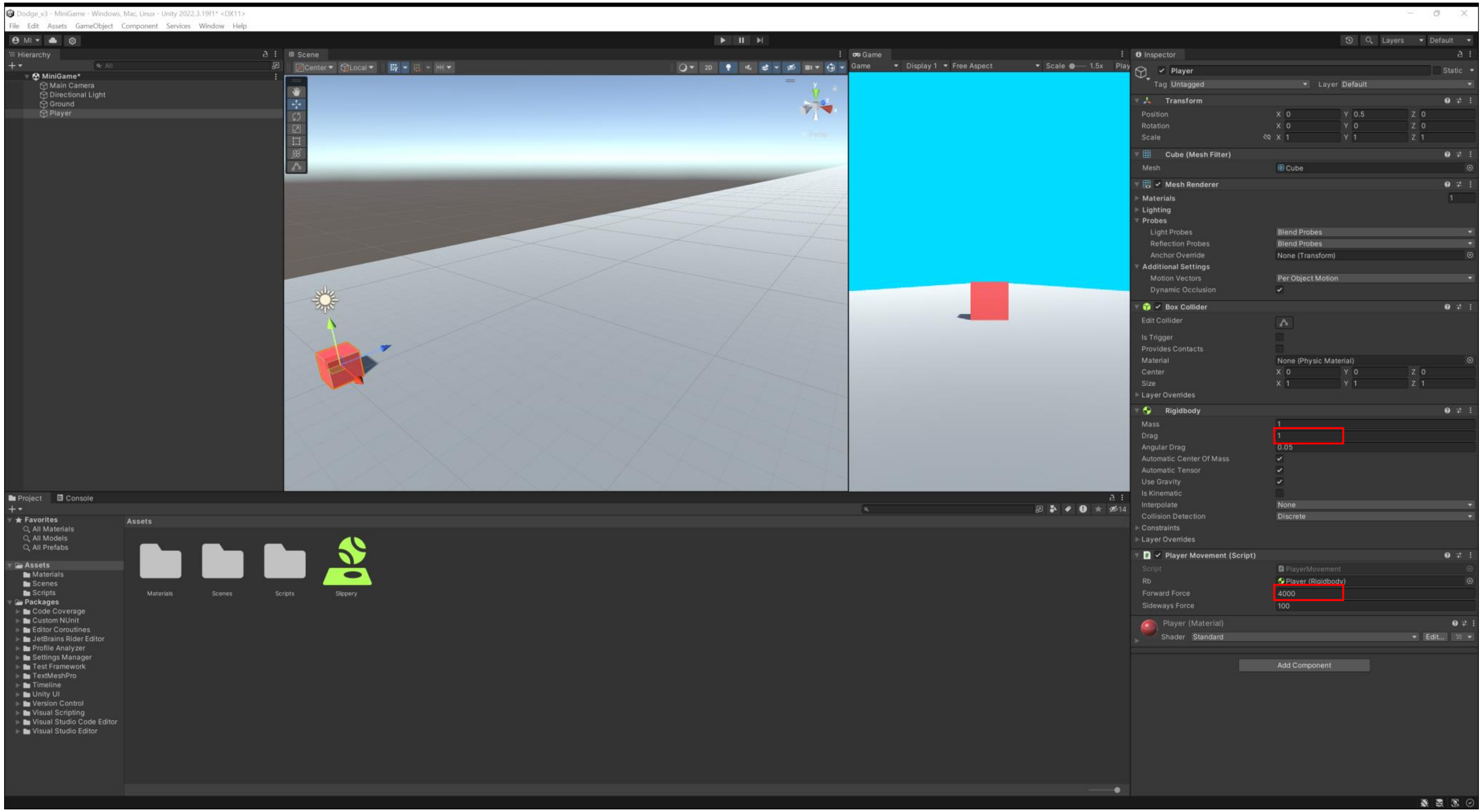
Slippery (Physic Material)

Dynamic Friction	0
Static Friction	0
Bounciness	0
Friction Combine	Average
Bounce Combine	Average

Add sideways movement

- The forwardForce looks to be slow when we hit the play button. Let’s change it to 4000.
- Also set the drag value to 1 from 0 in the Rigidbody component of the Player. This adds air resistance and make the player’s movement smoother.

Add sideways movement



Player — Tag: Untagged — Layer: Default

Transform

	X	Y	Z
Position	0	0.5	0
Rotation	0	0	0
Scale	1	1	1

Cube (Mesh Filter)
Mesh: Cube

Mesh Renderer
Materials: 1
Light Probes: Blend Probes
Reflection Probes: Blend Probes
Anchor Override: None (Transform)
Motion Vectors: Per Object Motion
Dynamic Occlusion: ✓

Box Collider
Is Trigger: ☐
Provides Contacts: ☐
Material: None (Physic Material)
Center: X 0 Y 0 Z 0
Size: X 1 Y 1 Z 1

Rigidbody
Mass: 1
Drag: 1
Angular Drag: 0.05
Automatic Center Of Mass: ✓
Automatic Tensor: ✓
Use Gravity: ✓
Is Kinematic: ☐
Interpolate: None
Collision Detection: Discrete

Player Movement (Script)
Script: PlayerMovement
Rb: Player (Rigidbody)
Forward Force: 4000
Sideways Force: 100

Player (Material) — Shader: Standard

Section

Moving the Camera

Making the Camera follow the Player

- **Approach 1:** Attach camera as child of the player object.
  - **Drawback:** Whenever the player rotates, the camera will also rotate, potentially causing a disorienting experience for the user.

- **Approach 2:** Script for camera follow

Making the Camera follow the Player

- **Create the CameraMovement script.**
  - With the **Main Camera** GameObject selected in the **Hierarchy** window, select **Add Component** > **New script** in the **Inspector** window.
  - Name your new script “CameraMovement”.
  - In the **Project** window, move the script from the root **Assets** folder into the **Scripts** folder.
  - Open the new script for editing.

- **Declare player and offset variables.**
  - Inside the first curly brace, add the following lines of code to declare two new variables:

```
public Transform player;
 Public Vector3 offset;
```

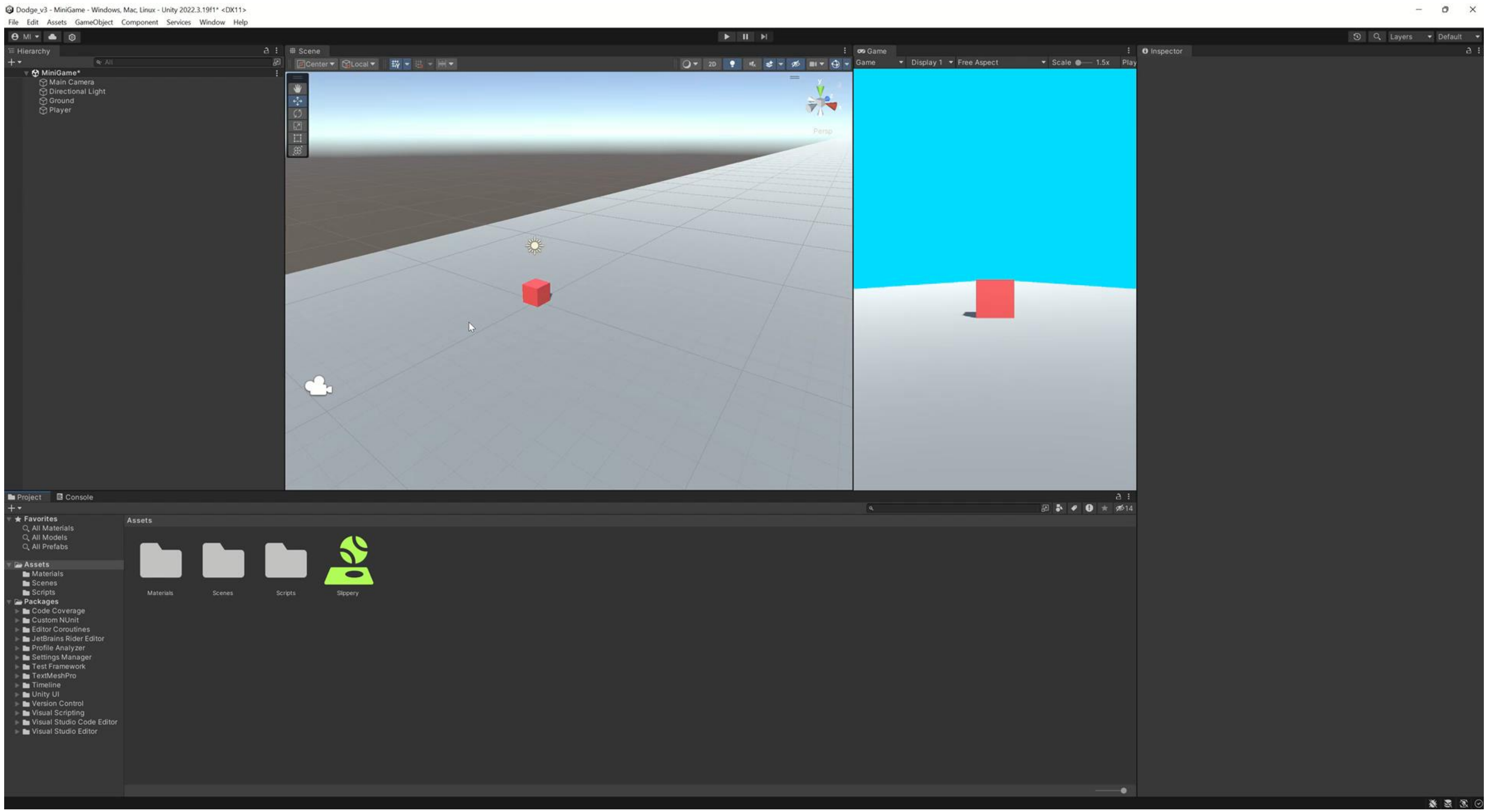
Making the Camera follow the Player

- **Set the camera position in LateUpdate.**
  - In the **Update** function, inside the first curly brace, add the following line of code:

```
transform.position = player.transform.position + offset;
```

- **Assign the Player GameObject variable in the Inspector window and Set the offset value.**
  - Make sure you have saved the **CameraController** script, then return to Unity.
  - Drag the **Player** GameObject from the **Hierarchy** window into the **Player** slot in the **CameraController** component.
  - For the offset, Set X = 0, Y = 2 and Z = -8.

Making the Camera follow the Player



Section

Creating Obstacles and Detecting Collisions

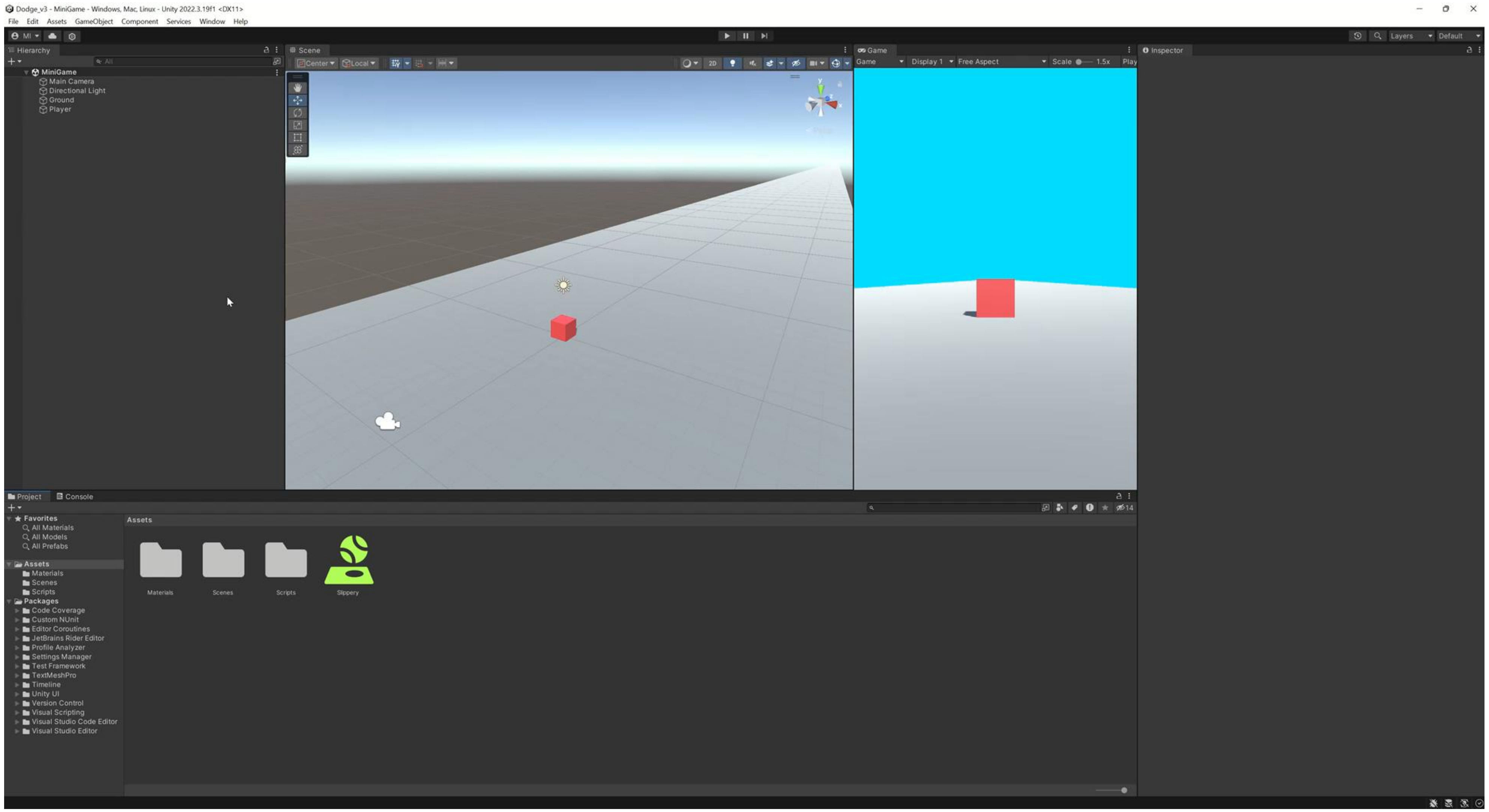
Creating an Obstacle

- **Create an Obstacle:** Create a new cube object in the scene to represent an obstacle the player can collide with.
  - (Select the main camera and set its Y position = 3 to get a better view in the Game View.)
  - Reset obstacle's position in the (Transform Component in the Inspector tab).
  - Then set its Y position = 0.5 and Z position = 35.

- **Obstacle Properties:** Customize the obstacle's appearance by adjusting its color (R = 60, G = 60, B = 60), smoothness (= 0.75, optional), and size (Scale → X = 3).

- **Adding Rigidbody and Mass:** Attach a Rigidbody component to the obstacle. Adjust the obstacle's mass to influence its behavior during collisions. A higher mass (such as 2) will make the obstacle less movable upon impact with the player.

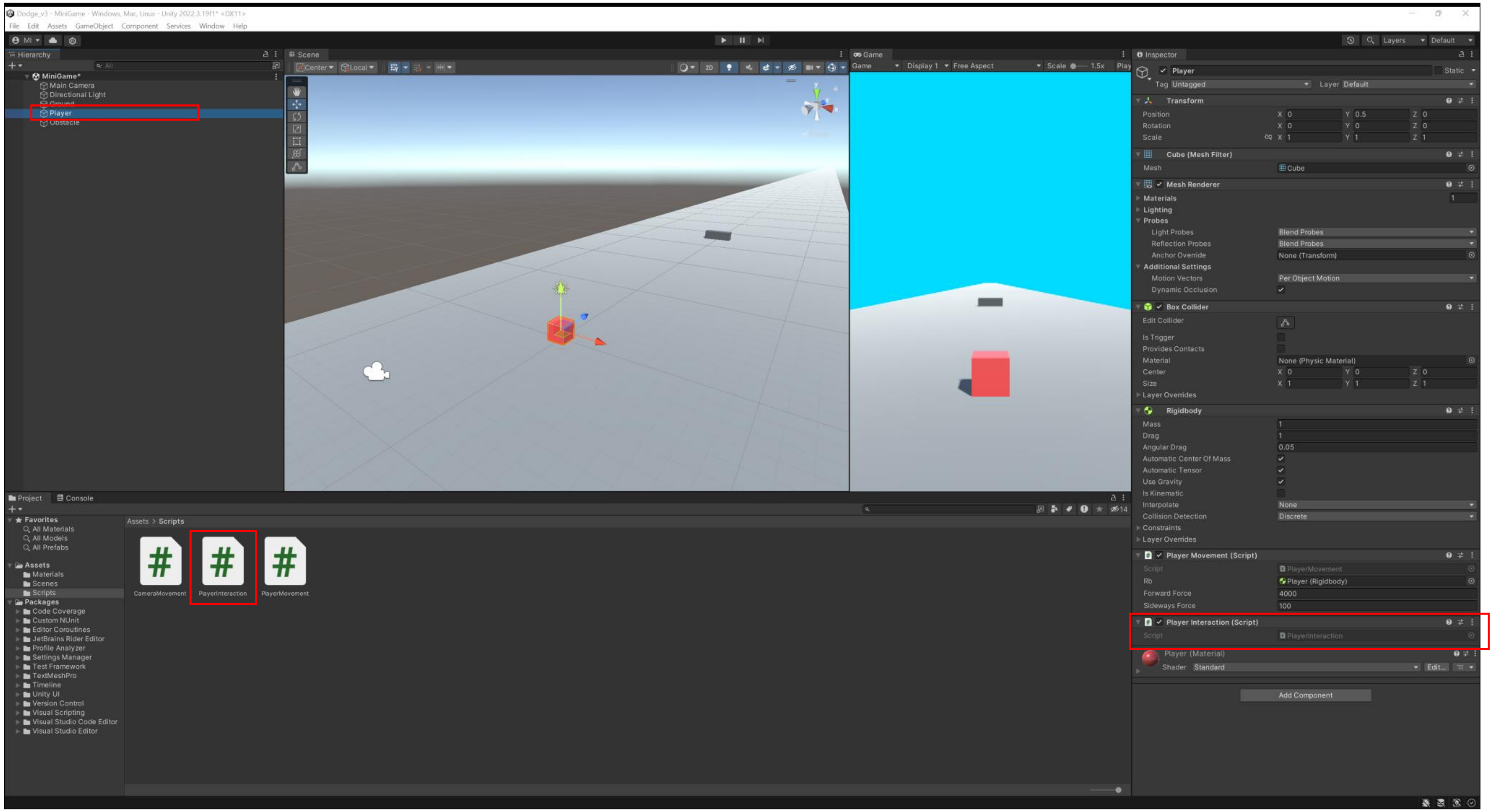
Creating an Obstacle



Detecting Collision

- **Scripting Collisions (Separate Script)**
  - While collision detection can be implemented in the player movement script, it's often cleaner to create a dedicated script for better organization.
  - Select the Player object and add a new script to it named **PlayerInteraction**.
  - Place **PlayerInteraction.cs** in the Scripts folder.

Detecting Collision



Player — Static
Tag Untagged — Layer Default

Transform
Position X 0 Y 0.5 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Cube (Mesh Filter)
Mesh Cube

Mesh Renderer
Materials 1
Lighting
Probes
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Additional Settings
Motion Vectors Per Object Motion
Dynamic Occlusion ✓

Box Collider
Edit Collider
Is Trigger
Provides Contacts
Material None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1
Layer Overrides

Rigidbody
Mass 1
Drag 1
Angular Drag 0.05
Automatic Center Of Mass ✓
Automatic Tensor ✓
Use Gravity ✓
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Layer Overrides

Player Movement (Script)
Script PlayerMovement
Rb Player (Rigidbody)
Forward Force 4000
Sideways Force 100

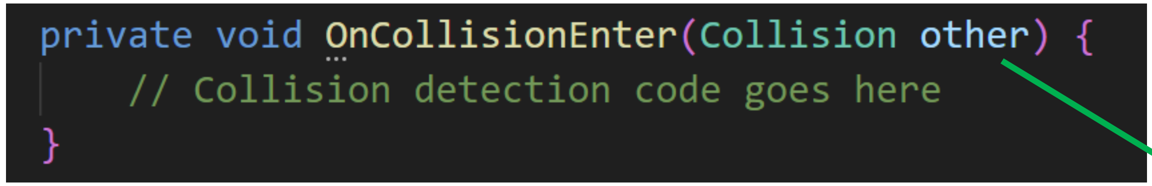
Player Interaction (Script)
Script PlayerInteraction

Player (Material)
Shader Standard

Add Component

OnCollisionEnter Function

- Unity provides built-in functions for handling collisions. It’s basic structure is as follow:



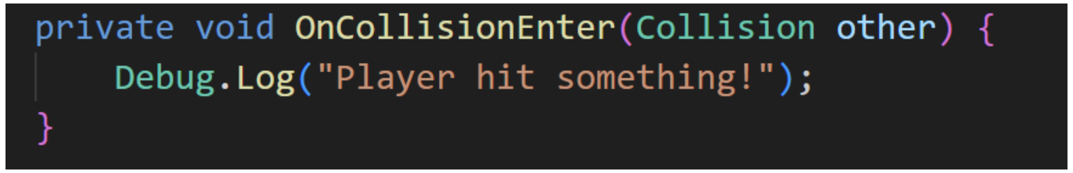
```
private void OnCollisionEnter(Collision other) {
    // Collision detection code goes here
}
```

Refers to the object with which the player collided/came into contact.

- The OnCollisionEnter() function gets called whenever the player object collides with another object in the scene.
- We can use Debug.Log statements within this function to verify collision detection. Initially, this might show a collision with the ground since it is a game object as well.

OnCollisionEnter Function

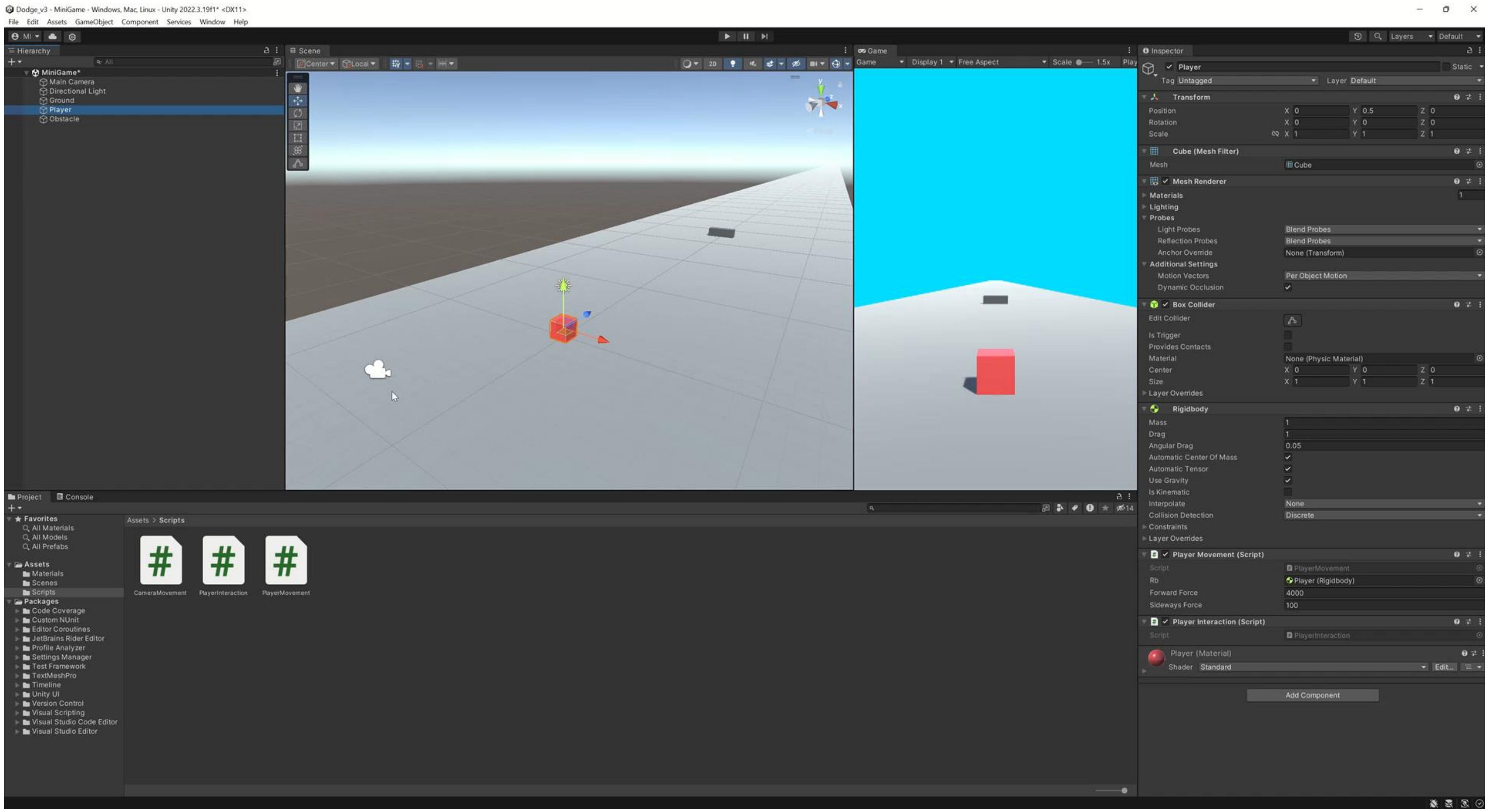
- We can use Debug.Log statements within this function to verify collision detection. Initially, this might show a collision with the ground since it is a game object as well.
- In the PlayerInteraction.cs script:
  - Remove the Start function (We won’t need it here).
  - Within the first curly braces, add the following code segment (above Update fucntion).



```
private void OnCollisionEnter(Collision other) {
    Debug.Log("Player hit something!");
}
```

- Head back to Unity and click the play button. Select the console tab on the lower portion of the editor.
- We will see that the message “Player hit something!” is displayed. The message gets displayed twice, one for the ground (which we don’t want for the game) and the other one for the obstacle.

OnCollisionEnter Function



Player — Tag: Untagged — Layer: Default

Transform
- Position: X 0, Y 0.5, Z 0
- Rotation: X 0, Y 0, Z 0
- Scale: X 1, Y 1, Z 1

Cube (Mesh Filter)
- Mesh: Cube

Mesh Renderer
- Materials: 1
- Lighting
- Probes
  - Light Probes: Blend Probes
  - Reflection Probes: Blend Probes
  - Anchor Override: None (Transform)
- Additional Settings
  - Motion Vectors: Per Object Motion
  - Dynamic Occlusion: ✓

Box Collider
- Edit Collider
- Is Trigger: ☐
- Provides Contacts: ☐
- Material: None (Physic Material)
- Center: X 0, Y 0, Z 0
- Size: X 1, Y 1, Z 1
- Layer Overrides

Rigidbody
- Mass: 1
- Drag: 1
- Angular Drag: 0.05
- Automatic Center Of Mass: ✓
- Automatic Tensor: ✓
- Use Gravity: ✓
- Is Kinematic: ☐
- Interpolate: None
- Collision Detection: Discrete
- Constraints
- Layer Overrides

Player Movement (Script)
- Script: PlayerMovement
- Rb: Player (Rigidbody)
- Forward Force: 4000
- Sideways Force: 100

Player Interaction (Script)
- Script: PlayerInteraction

Player (Material) — Shader: Standard

Identifying Colliding Objects

- In our game, we need to detect when the player collides with obstacles. This allows us to respond accordingly, such as halting player movement upon collision.

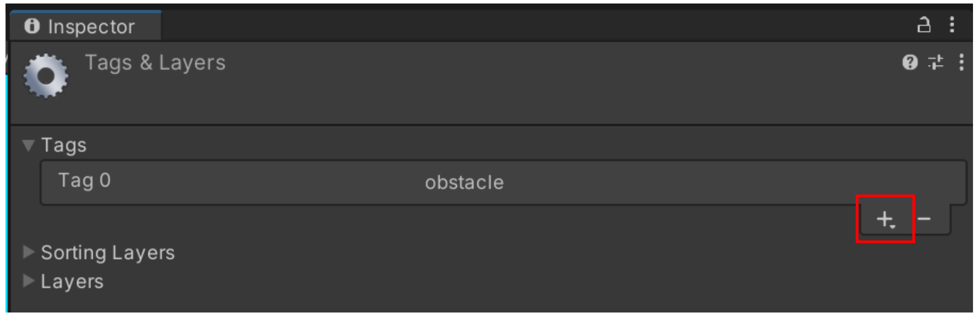
- While using the collider's name for identification can be a starting point, it's not a reliable approach for complex games.
  - Names can be easily duplicated or forgotten, leading to potential issues.

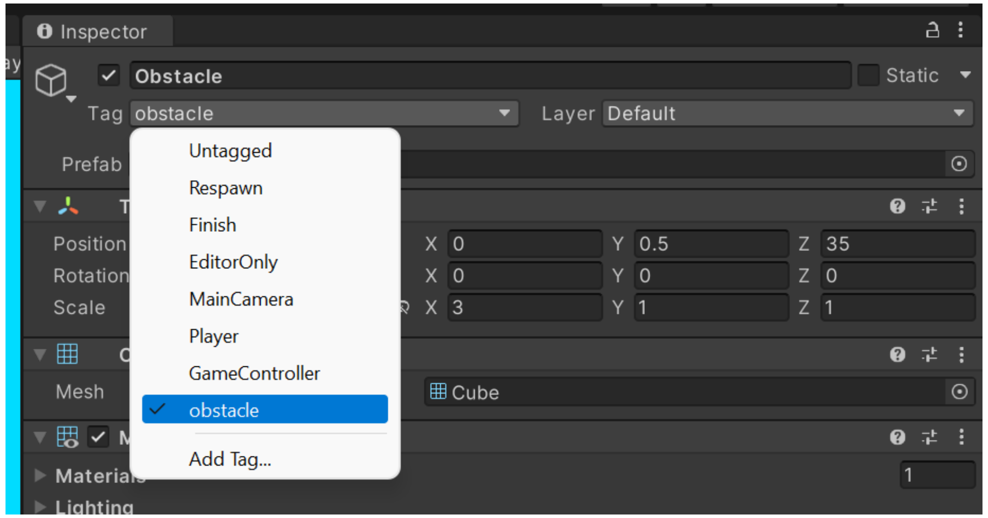
- **Tags** provide a more robust way to identify objects in Unity.
  - Assign a unique tag to the obstacle object (e.g., “obstacle").

Identifying Colliding Objects

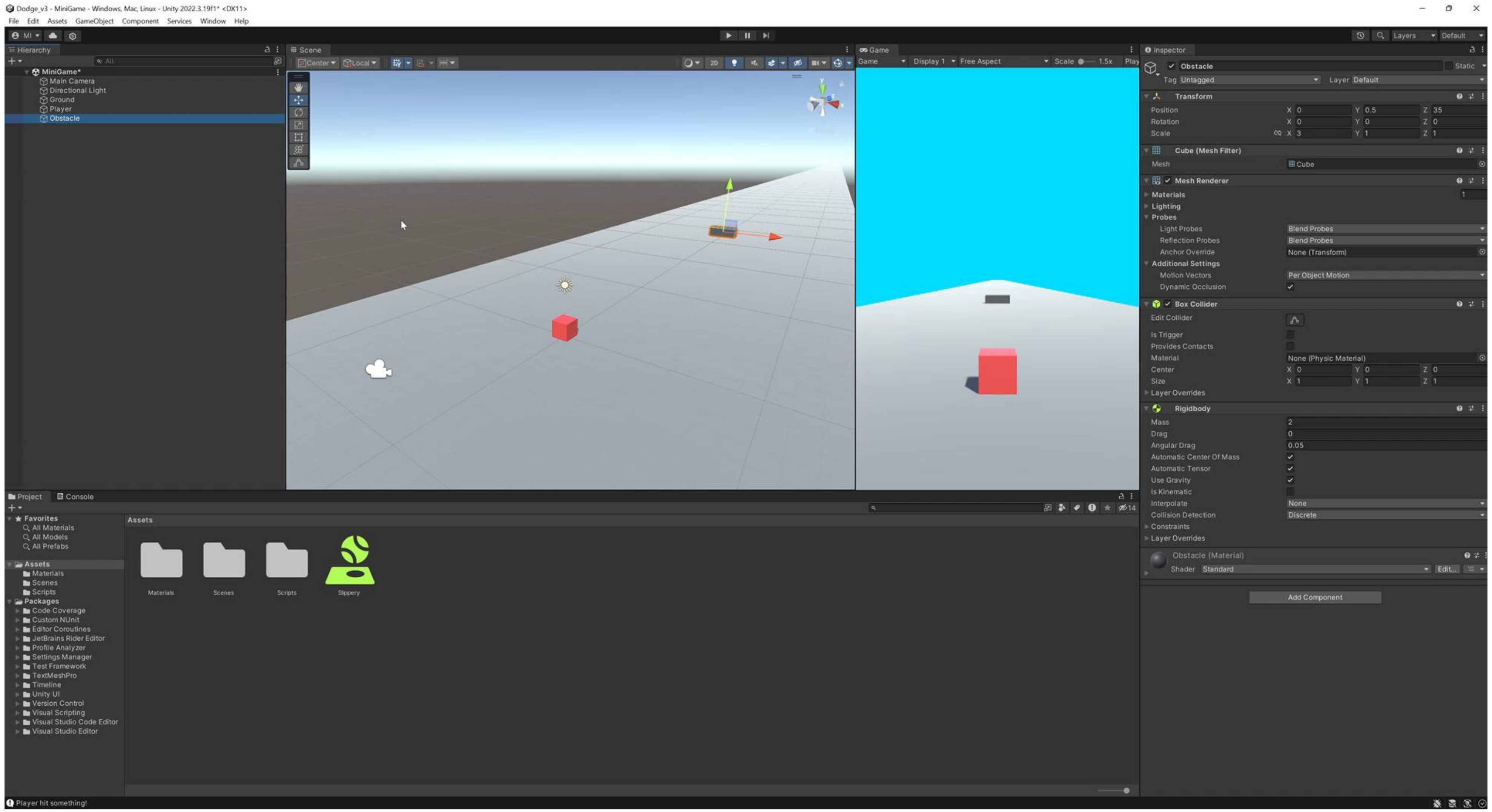
- **Create a new tag.**
  - Select the Obstacle object in the Hierarchy panel.
  - At the top of the **Inspector** window, from the **Tag** dropdown menu, select **Add Tag**.
  - Select the **Add** (**+**) button to add a new tag, then name it "**obstacle**".
  - **Important:** This is case sensitive, so be careful — it needs to be exactly the same spelling and capitalization that you use in the script later on.

- **Apply the tag to the Obstacle object.**
  - With the **Obstacle** object still selected, use the **Tag** dropdown menu to select the new "obstacle" tag from the list.





Identifying Colliding Objects


Dodge_v3 - MiniGame - Windows, Mac, Linux - Unity 2022.3.19f1* <DX11>
File Edit Assets GameObject Component Services Window Help
Hierarchy
Scene
Game
Inspector
Project
Console
Materials
Scenes
Scripts
Slippery

Hierarchy:
- MiniGame*
  - Main Camera
  - Directional Light
  - Ground
  - Player
  - Obstacle

Inspector: Obstacle (Tag: Untagged, Layer: Default)

Transform

	X	Y	Z
Position	0	0.5	35
Rotation	0	0	0
Scale	3	1	1

Cube (Mesh Filter)
Mesh: Cube

Mesh Renderer
Materials: 1
Light Probes: Blend Probes
Reflection Probes: Blend Probes
Anchor Override: None (Transform)
Motion Vectors: Per Object Motion
Dynamic Occlusion: ✓

Box Collider
Is Trigger: ☐
Provides Contacts: ☐
Material: None (Physic Material)
Center: X 0, Y 0, Z 0
Size: X 1, Y 1, Z 1

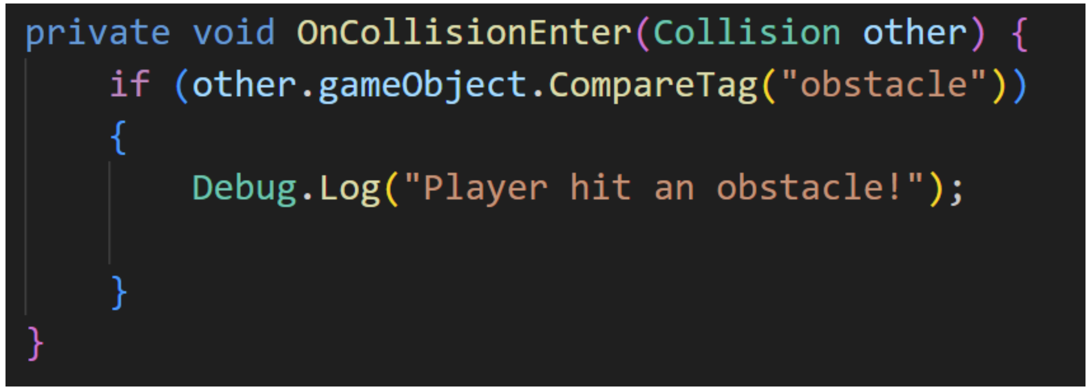
Rigidbody
Mass: 2
Drag: 0
Angular Drag: 0.05
Automatic Center Of Mass: ✓
Automatic Tensor: ✓
Use Gravity: ✓
Is Kinematic: ☐
Interpolate: None
Collision Detection: Discrete

Obstacle (Material)
Shader: Standard

Add Component

Player hit something!

Identifying Colliding Objects



```
private void OnCollisionEnter(Collision other) {
    if (other.gameObject.CompareTag("obstacle"))
    {
        Debug.Log("Player hit an obstacle!");

    }
}
```

Acknowledgement

- The contents of these slides have been adopted primarily from the following two sources:
    - https://learn.unity.com/project/roll-a-ball
    - https://brackeys.com/