MINISTRY OF E DUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY OF UKRAINE

" IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE"

Amons Oleksandr

Basics of video game development

Laboratory works

Creating a breakout game with Unity

Kulubecioglu Mehmet

IM-14 FIOT

Kyiv

IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE

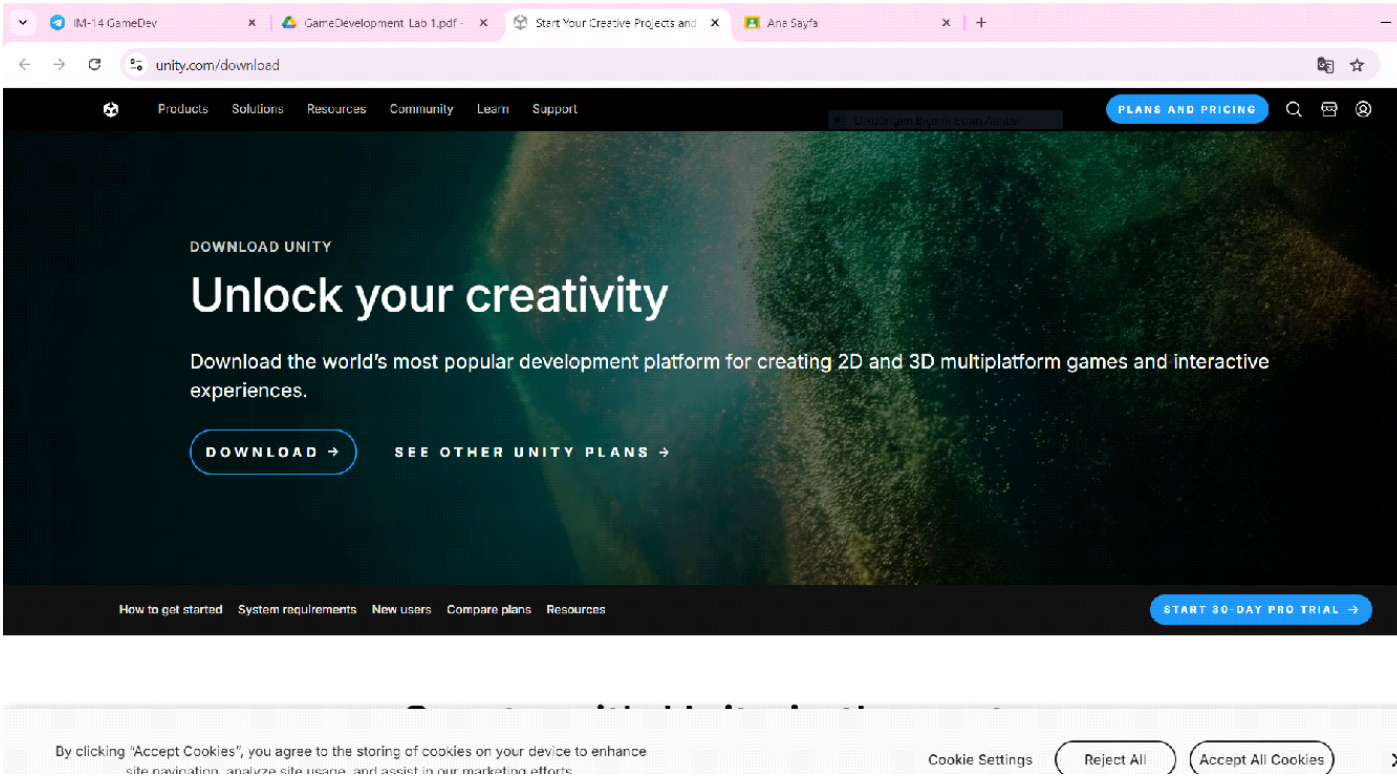
2024

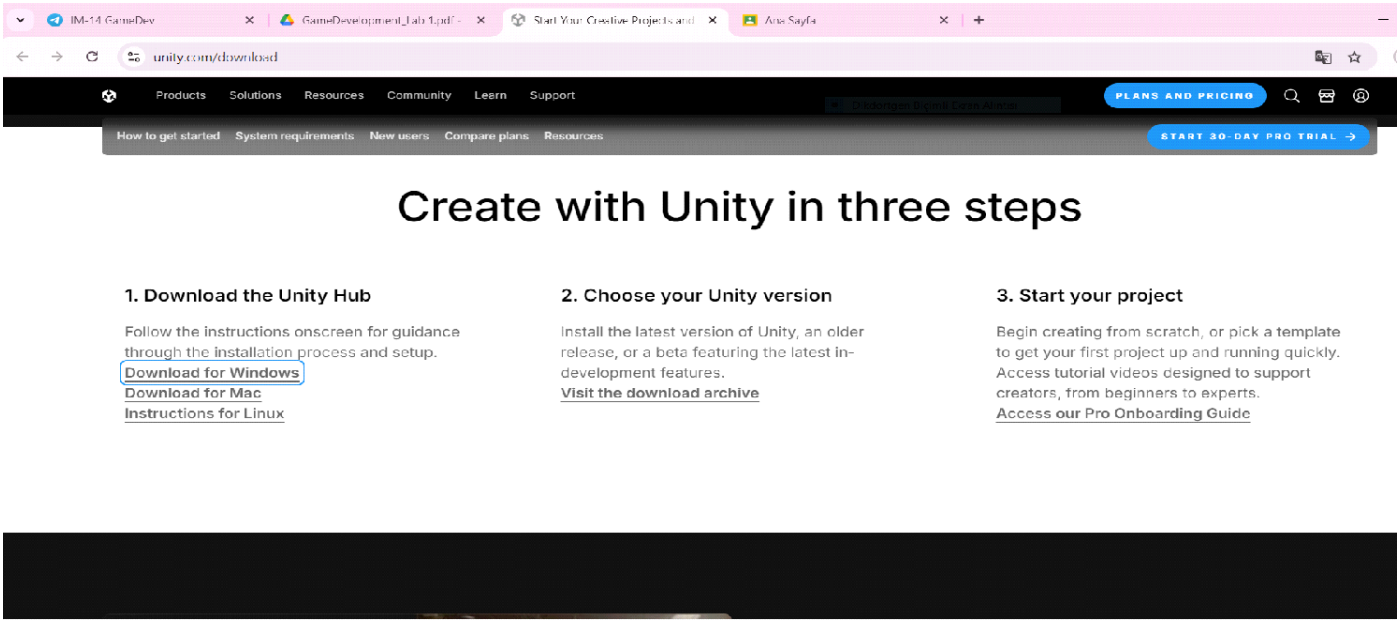
1. Tasks

• Install Unity
• Create a new Unity project.
• Set up a simple scene with basic objects (e.g., terrain, skybox, basic shapes like cubes and spheres).
• Use assets for create main Scene.
The result: Designed first level scene for the game

2. Progress of the lab

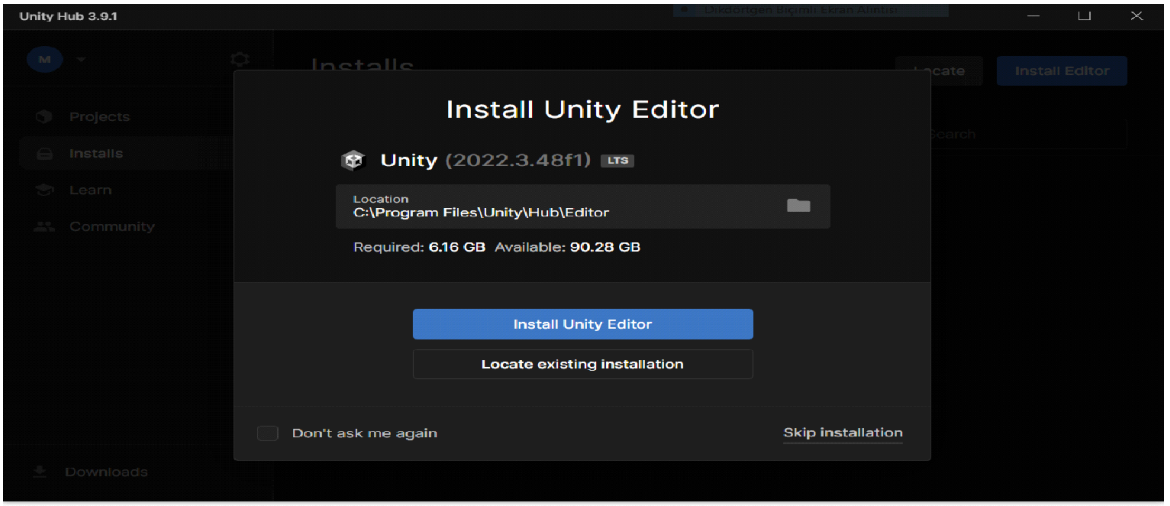
Install Unity.





- Under 1. Download the Unity Hub and select the option for your operating system.
- Launch the UnityHubSetup installer (file that you downloaded previously)
- Follow this wizard instructions to install the Unity Hub
- Create a Unity account (choose Unity Personal plan).
- Launch Unity Hub.

- Once you're in the Unity Hub, it will guide you to install the latest long-term support (LTS) version of the Unity Editor

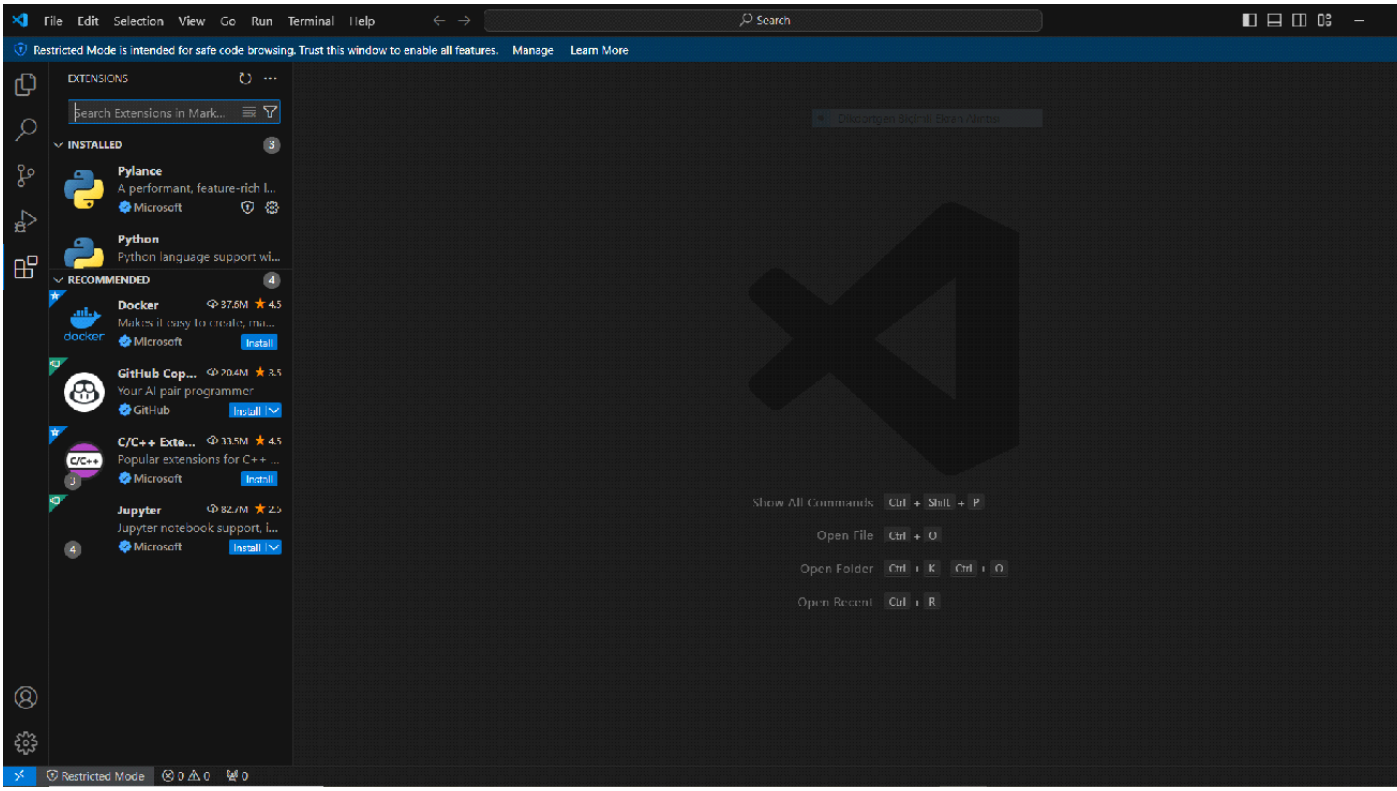


• Select Install Unity Editor to download and install the latest LTS version of the Unity Editor (for instance, I have installed Unity 2022.3.44f1). The download and installation of Unity Editor take some time. When the process is complete, this version of the Unity Editor will appear on the

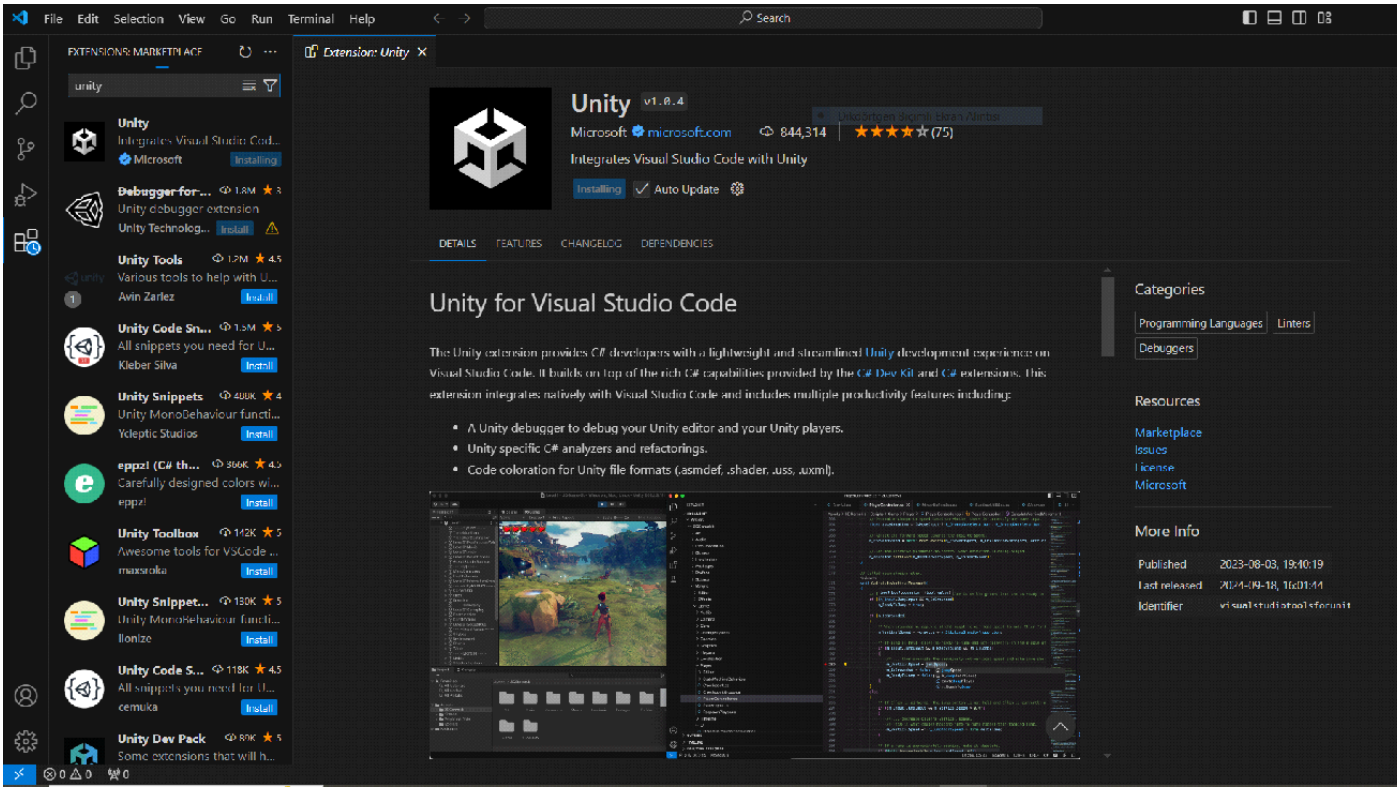
Installs tab of the Unity Hub.

Install Visual Studio Code:

I have already vs code

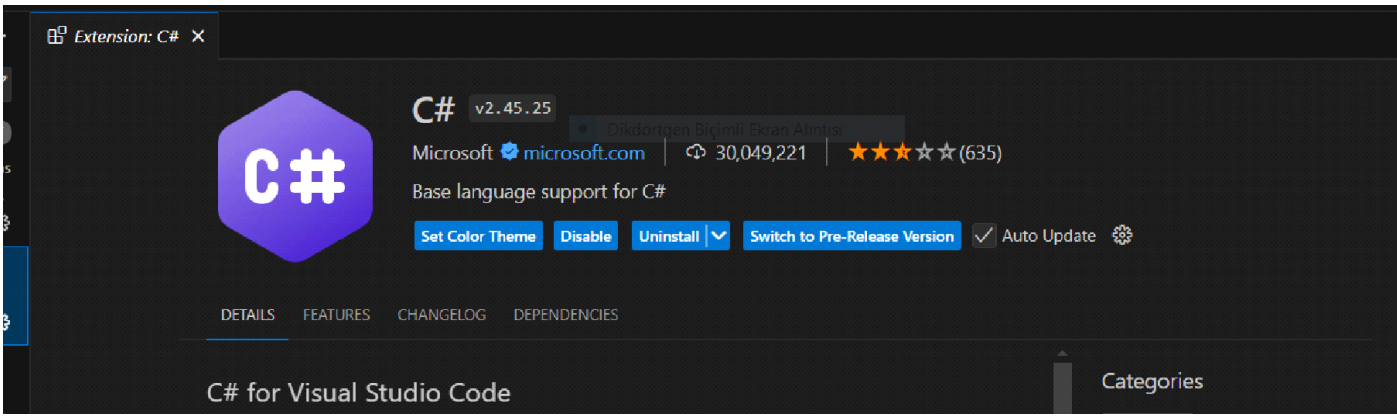


- Launch Visual Studio Code.
- In VS Code, open the Extensions view by clicking on the Extensions icon in the Activity Bar on the side of VS Code:



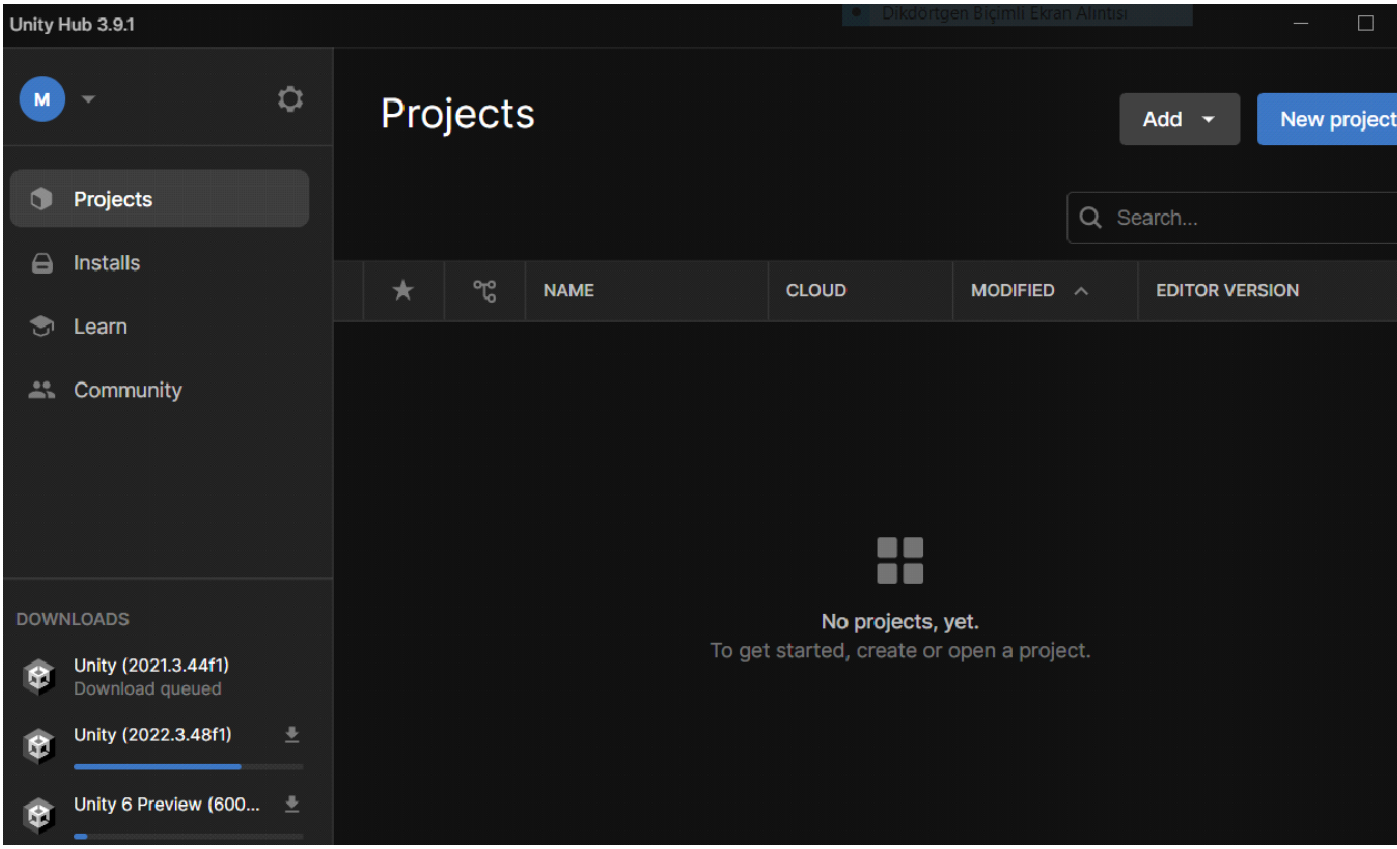
Need to install next extensions:

- Unity
- C#

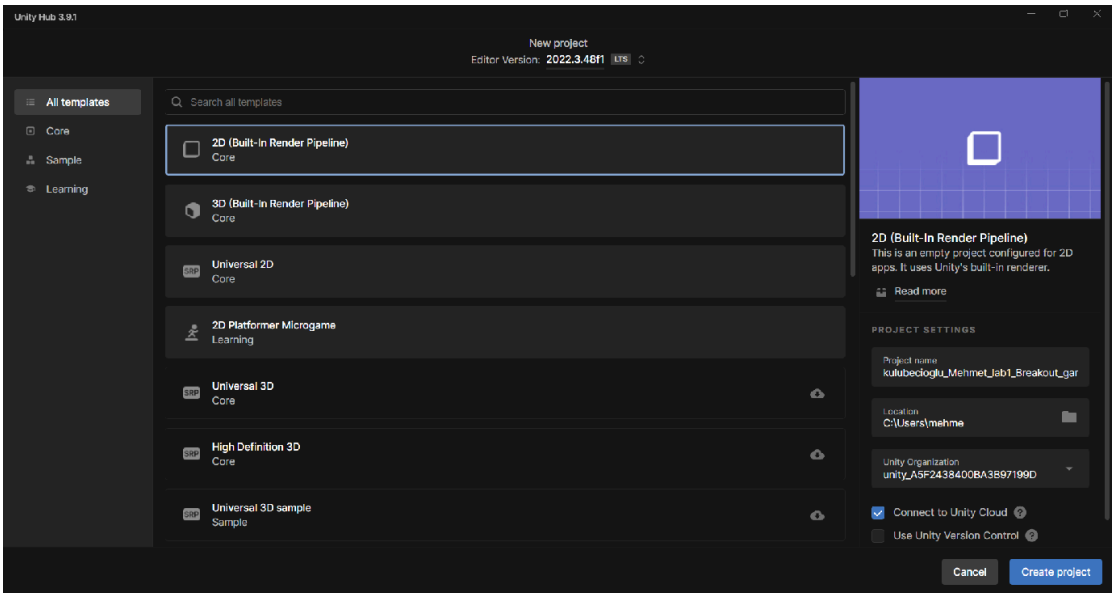


Create new Unity project

- Launch Unity Hub.
- Press the button “New project” on the Projects tab:



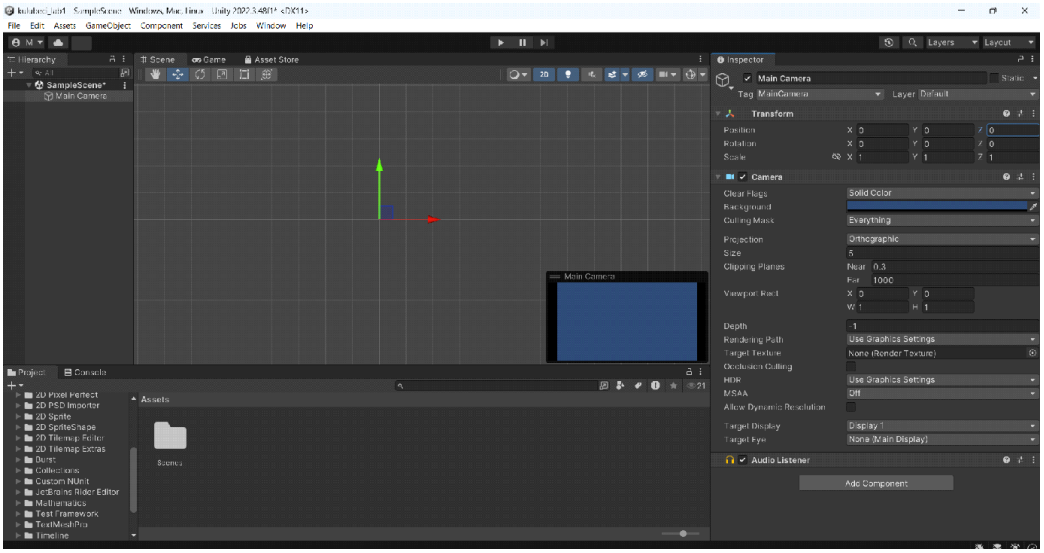
- Select template “2D (Built-In Render Pipeline). Core” (blocks 1 and 2 on the picture below)



- Enter the name of your project (without space) – block 3
- Select a location for your project – block 4

- Pres button “Create project”.
After pressing the button, creating a project is starting, and it takes time.
While the project is being created, you will see pictures like this:

When this project is created, the Unity window will open:



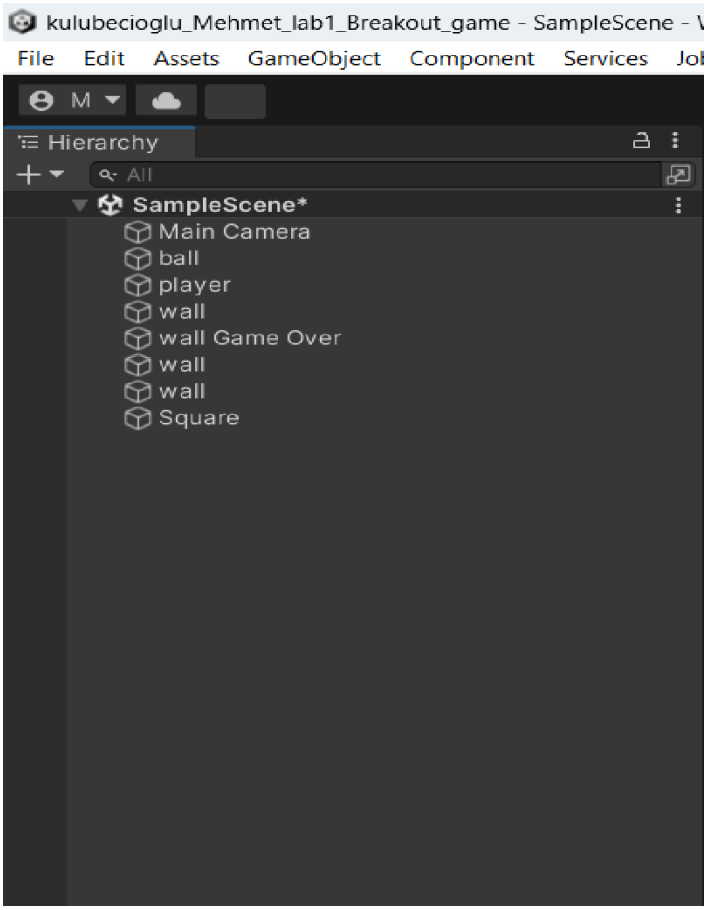
The project was created successfully.

I can open already created projects by double-clicking on a project. There are four main areas:

1. **Hierarchy Window:** This window contains all the objects in the current scene.

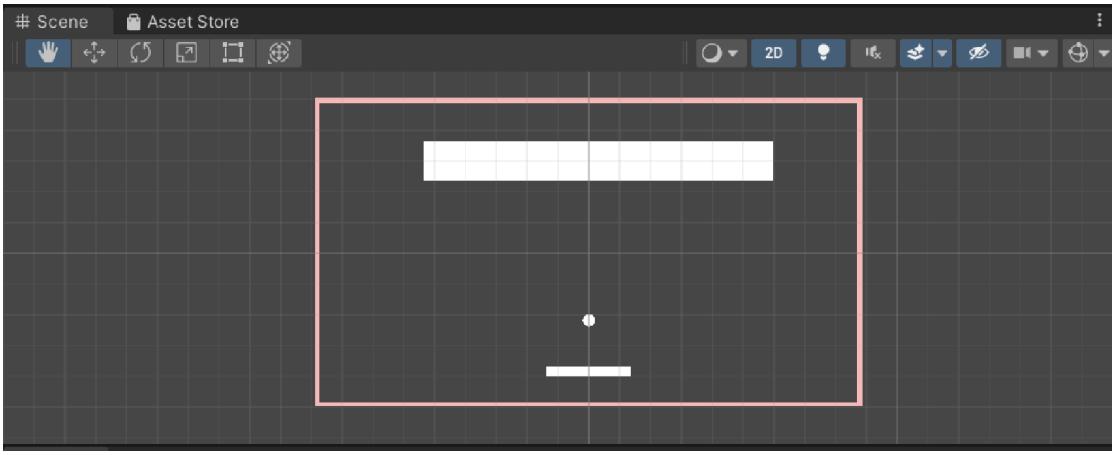
2. **Scene/Game View Window:** Here, I can edit my scenes and levels and test them in real-time.
3. **Project Window and Console Tab:**
    - **Project Window:** This section contains all the assets related to my project. 3D models, 2D textures, sound and music files, and anything else required for my game to look, sound, and play as I want it to.
    - **Console Tab:** Pressing the console tab brings up a debugging window called the Console. This window shows output and errors that may occur in my code.
4. **Inspector Window:** On the right is the inspector window, which shows all the data related to any selected object.
5. **Toolbar:** The vertical list of buttons in the Scene View is the Toolbar. This gives me the tools needed to manipulate objects (move, resize, rotate, etc.).
6. **Play, Pause, and Step Buttons:** I use these to test and debug my game project.

Hierarchy Window displays a list of all objects in the current scene in a hierarchical format. This window also allows for the creation of new GameObjects via the “Create” drop-down menu in the top-left corner. The search field allows me to search for specific GameObjects by name. In Unity, GameObjects can contain other GameObjects in what’s called a “parent-child” relationship. The Hierarchy Window will display these relationships in a helpful nested format.



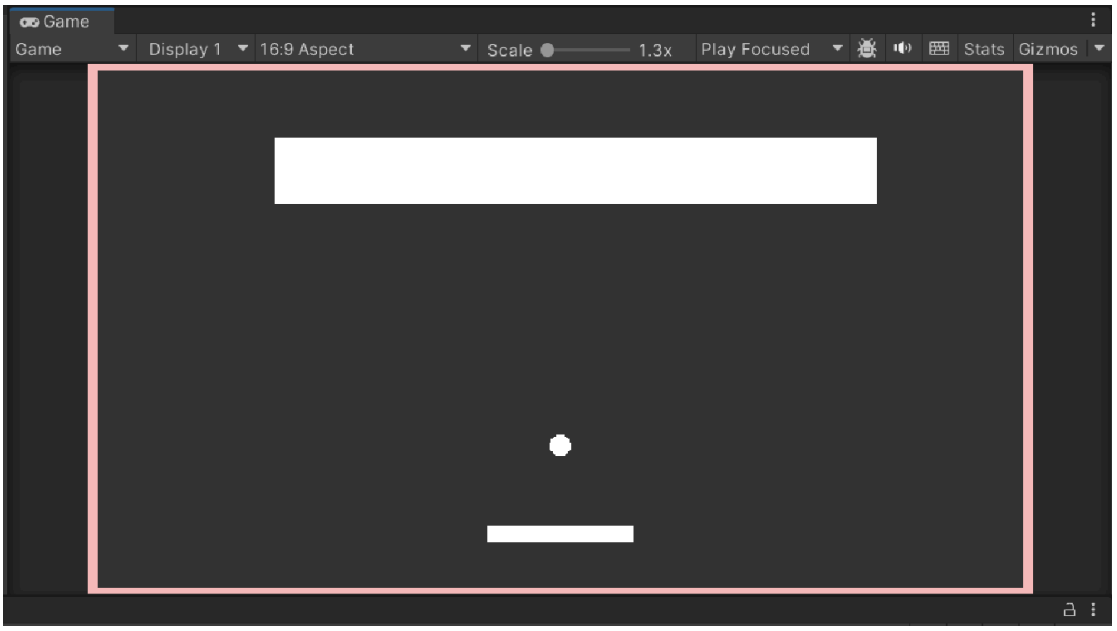
Scene View

I can think of scenes as the foundation of my Unity projects, so I'll have the Scene View open most of the time while I'm working in the Unity Editor. Everything that happens in my game takes place in a scene. The Scene View is where I'll construct my game and do most of my work with sprites and colliders. Scenes contain GameObjects, which hold all the functionality relevant to that scene. Just know that every object in my Unity scene is a GameObject.



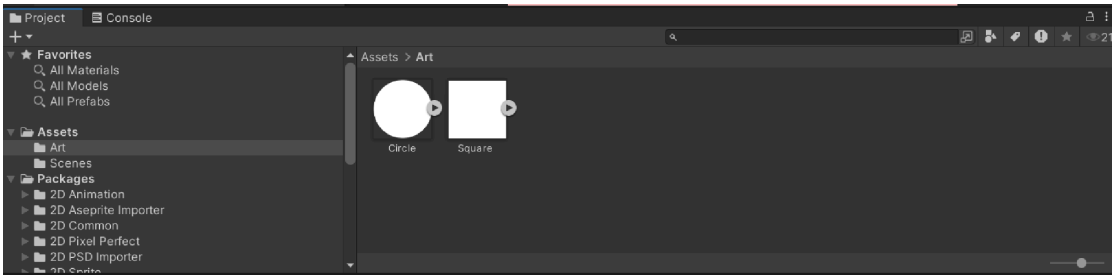
Game View

The Game View renders my game from the currently active camera's point of view. It's also where I'll view and play my actual game while I'm working on it in the Unity Editor. There are ways to build and run my game outside of the Unity Editor as well, such as a stand-alone application, in a web browser, or on a mobile phone, and I'll cover some of these platforms later in this book.



Project Window

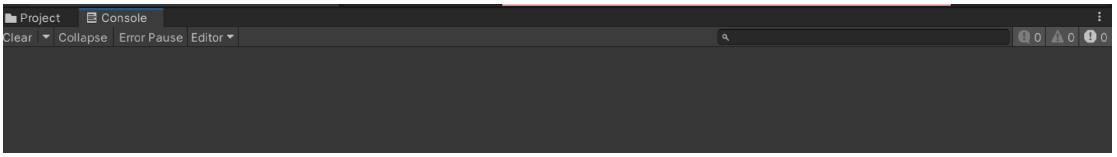
The Project Window gives me an overview of all the content in the Assets folder. It's helpful for me to create folders in the Project Window to organize items such as audio files, materials, models, textures, scenes, and scripts. Throughout the lifetime of my project, I'll spend a lot of time dragging and rearranging assets in folders and selecting those assets to view them in the Inspector Window. I should feel free to rearrange things in a way that makes logical sense to me and suits the way I like to work.



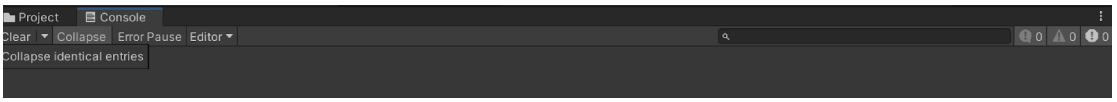
Console View

The Console View displays errors, warnings, and other output from my Unity application. There are C# scripting functions that I can use to output information to the Console View at runtime to aid in debugging. I can toggle the various forms of output on and off via the three buttons in the top-right of the window.

Console View.

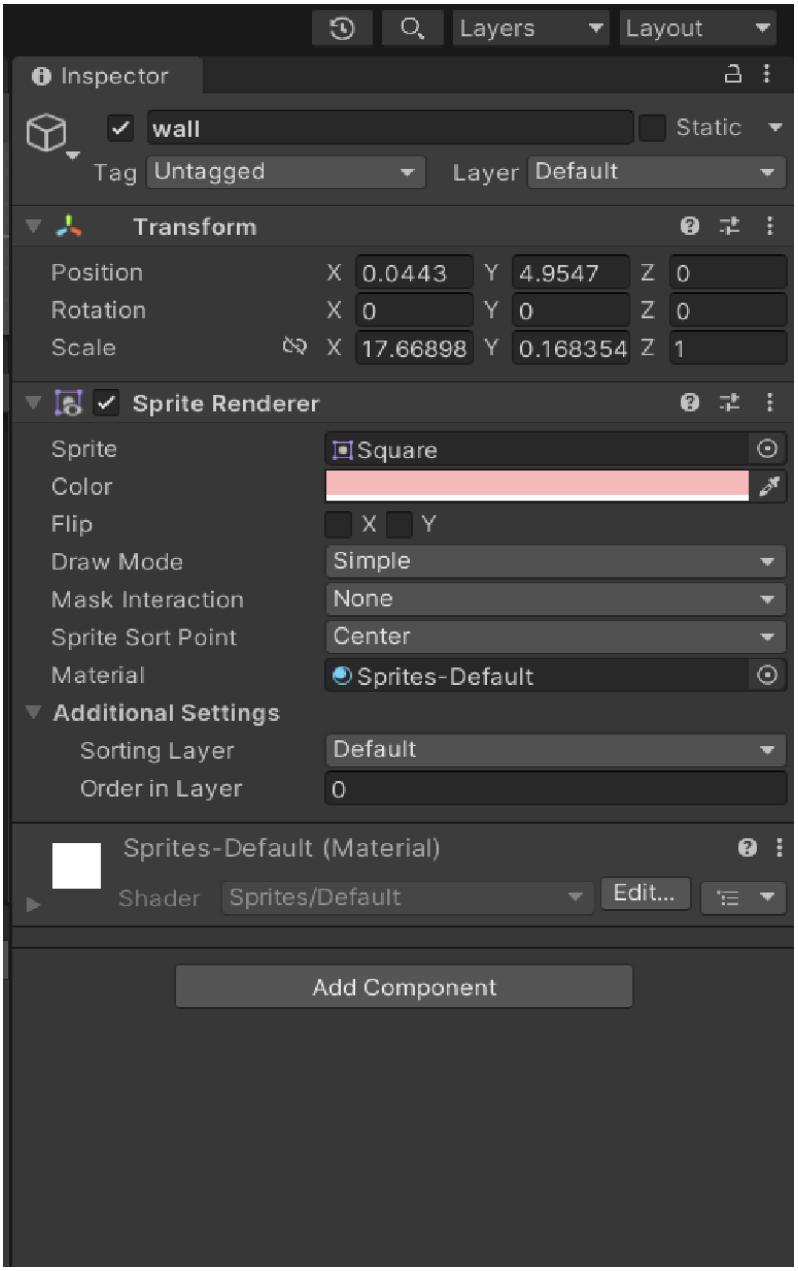


Sometimes you'll get an error message that occurs with every Unity frame update, and those messages will clog up your Console View in a hurry. In situations like this, it's helpful to hit the Collapse toggle button to collapse all identical error messages into a single message.



Inspector Window

The Inspector Window is one of the most useful and important windows in the Unity Editor; be sure to familiarize yourself with it. Scenes in Unity are made up of GameObjects, which consist of Components such as Scripts, Meshes, Colliders, and other elements. You can select a GameObject and use the Inspector to view and edit the attached Components and their respective properties. There are even techniques to create your own properties on GameObjects that can then be modified.
You can also use the Inspector to view and change properties on Prefabs, Cameras, Materials, and Assets as well. If an Asset is selected, such as an audio file, the Inspector will show details such as how the file was loaded, its imported size, and the compression ratio. Assets such as Material Maps will allow you to inspect the Rendering Mode and Shader.



The Transform Toolset

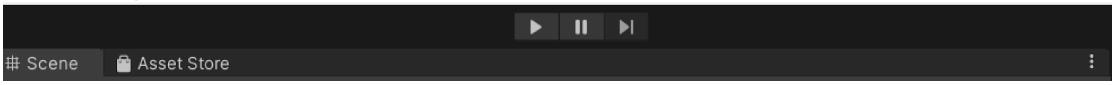
The Transform tools allow users to navigate the Scene View and interact with GameObjects.

The six Transform tools, from left to right, are:

- **Hand:** The Hand tool allows me to left-click and drag the mouse around the screen to pan around the view. I note that I won't be able to select any objects when the Hand Tool is selected.
- **Move:** By selecting the Move tool and then selecting a GameObject in either the Hierarchy or Scene View, I can move that object around the screen.
- **Rotate:** The Rotate tool allows me to rotate selected objects.
- **Scale:** The Scale tool lets me scale selected objects.
- **Rect:** The Rect tool allows for moving and resizing selected objects using 2D Handles, which will appear on the selected object.
- **Move, Rotate, or Scale Selected Objects:** This tool is a combination of the Move, Rotate, and Scale tools, consolidated into one set of Handles.

Play, Pause, and Step Controls

The Unity Editor has two modes: Play Mode and Edit Mode. When I press the Play button, provided there are no bugs preventing the game from building, the Unity Editor enters Play Mode and switches to the Game View.



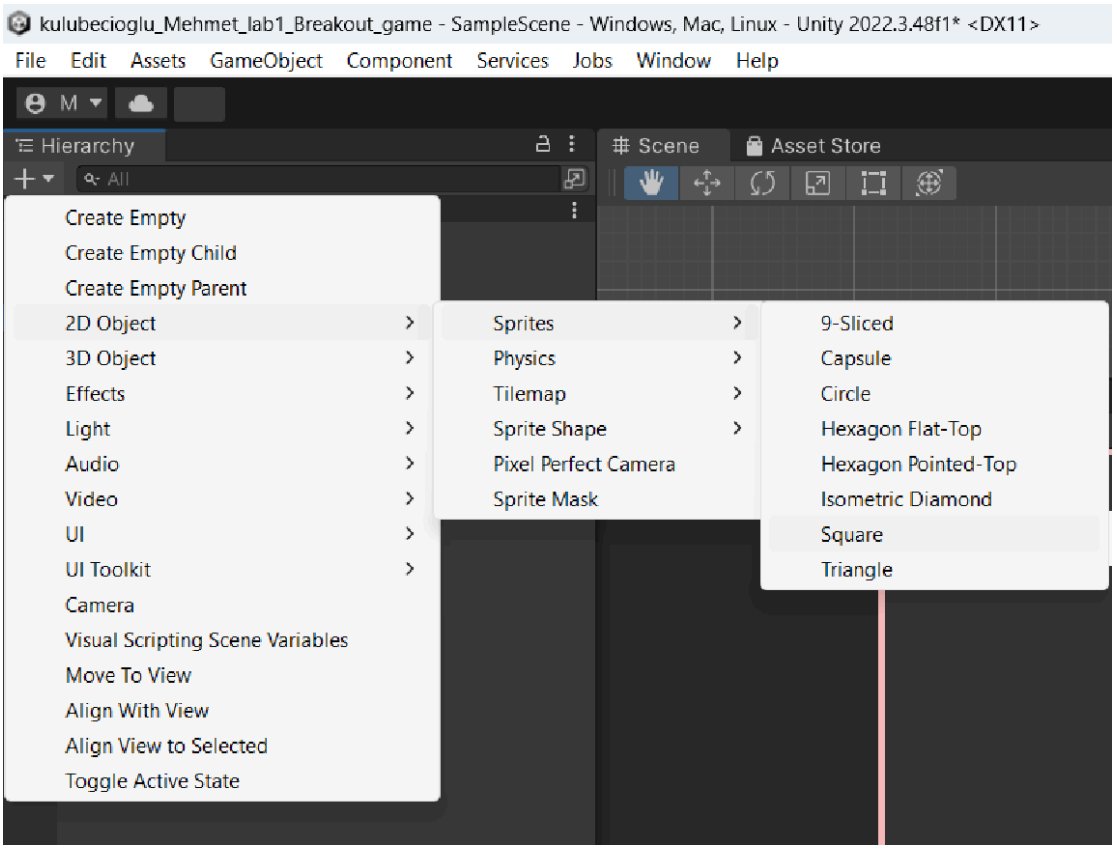
Create the First Scene

Games in Unity consist of Scenes, and everything in a Scene is called a GameObject. I'll encounter Scripts, Colliders, and other types of elements in my Unity adventures, and all of these are GameObjects. It's helpful for me to think of...

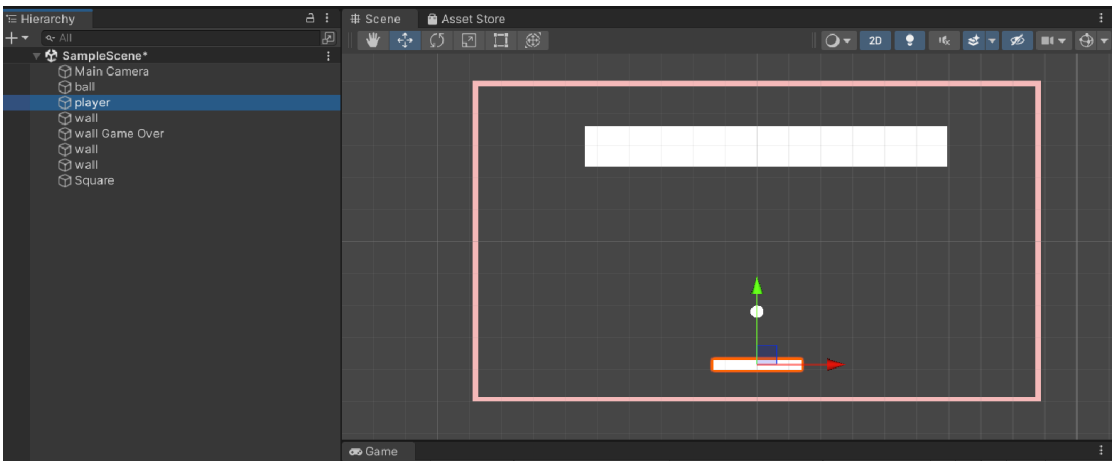
GameObjects serve as a sort of container, composed of many pieces of individually implemented functionalities. As I discussed in Chapter 2, GameObjects can even contain other GameObjects in parent–child relationships.

Placing Objects in the Scene

The most straightforward way for me to place an object into my scene is to use the right-click menu in the Hierarchy panel. I can bring up the menu, navigate down to 2D Object, and select, for example, "Square."

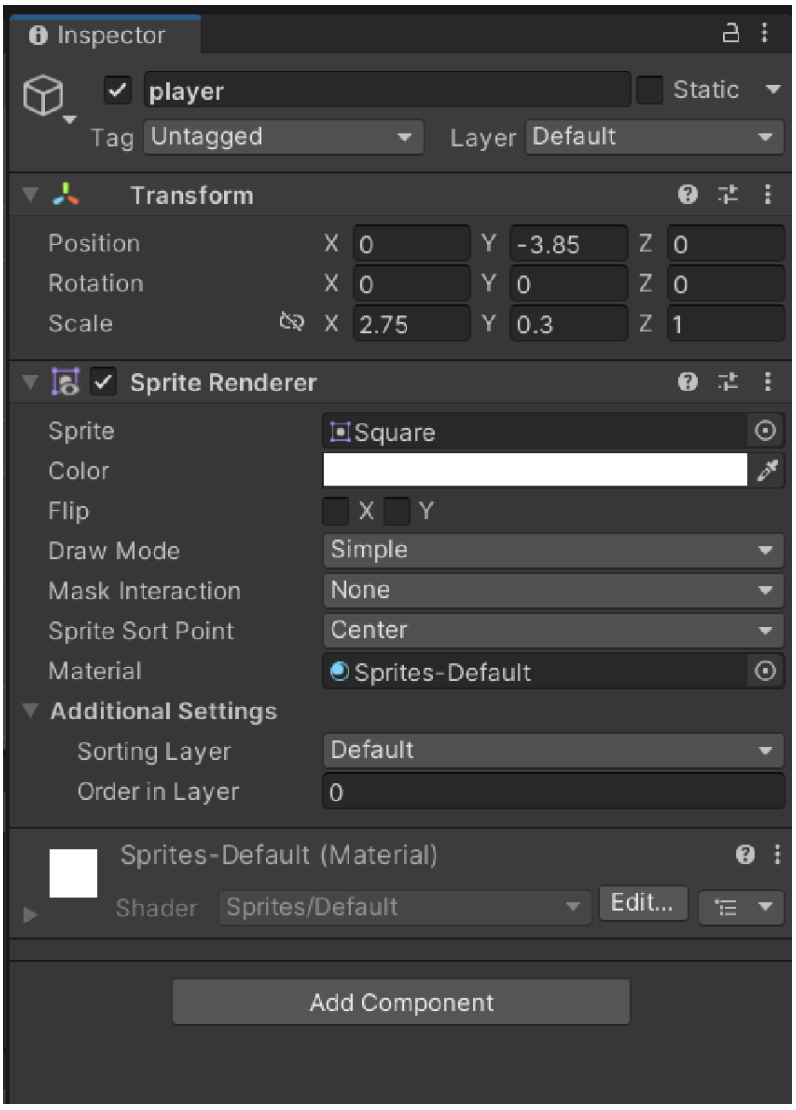


Selecting Sphere from this menu will place a 2D object “square” into the scene directly in the centre of the viewport.



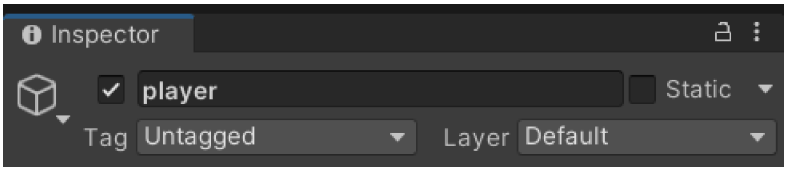
Inspector Panel
Select our new square object to see all its information listed in the

inspector panel:



At the top, I will see the Object Name. This is for internal use and will never be displayed to the player. I feel free to come up with a naming system to give my objects instantly identifiable names. For example, I'll name this one "MainCharacter." I can click on the existing name and enter "MainCharacter" as my new one.

To the left of the name is a small checkbox to enable or disable this object. I can click on it once to see the sphere disappear in the Scene View. I'll also notice that the object name turns grey in the Hierarchy panel. When I toggle it again, the sphere reappears, and its name is no longer greyed out.

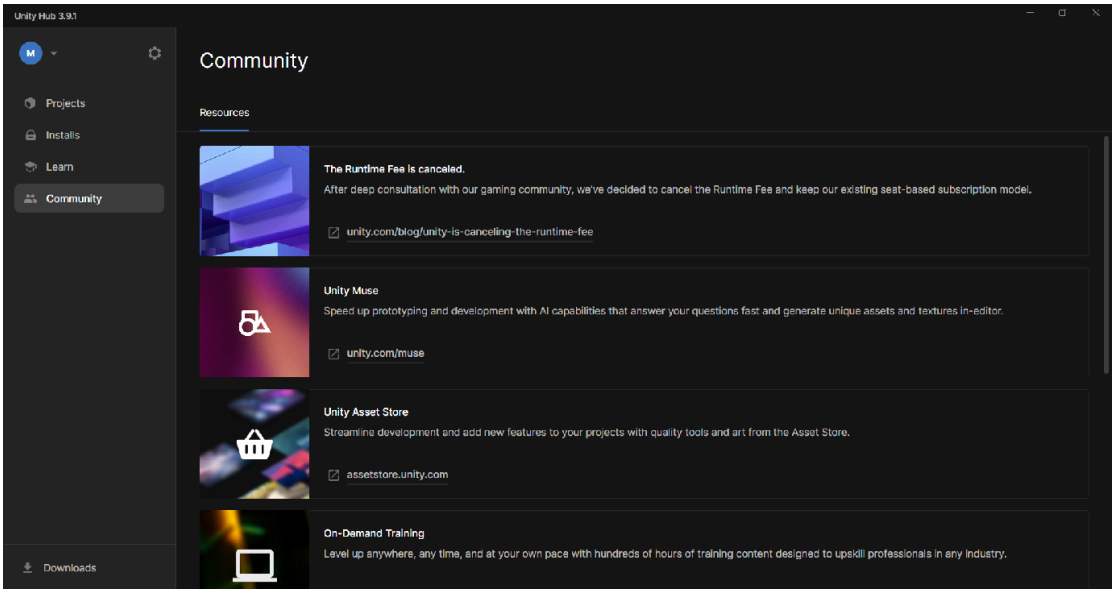


Following this top section, I see the Transform panel. This is an important section that shows where my object is in 3D space, as well as its rotation and scale information.

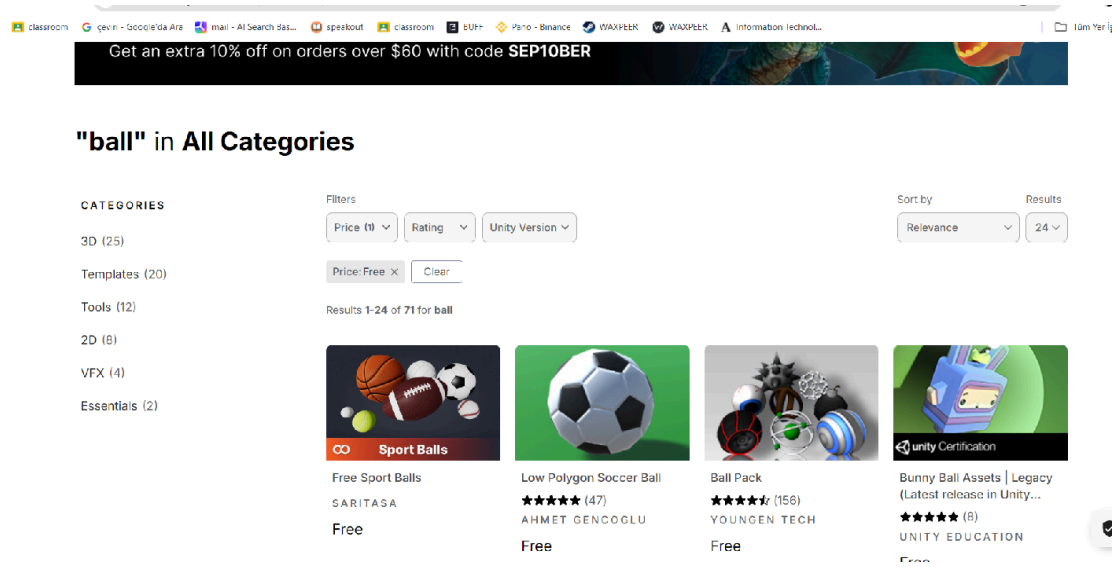
Apply Textures

Let's add some balls to my Scene. First, I need to create my PNG file in Photoshop or find and download an asset. Now, I'll download free assets.

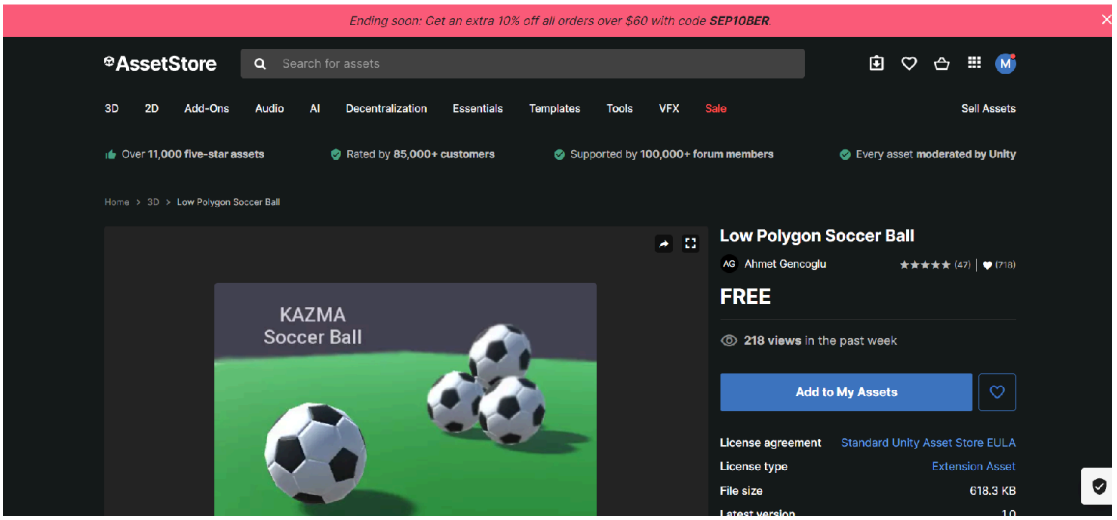
I can open the Unity Asset Store from Unity Hub, or I can click this link to open the Unity Asset Store.



In the search field, type ‘ball’ and press search.
On the next page select 2D in the left panel, select Price free in Filters, and
I will find a lot of free assets:

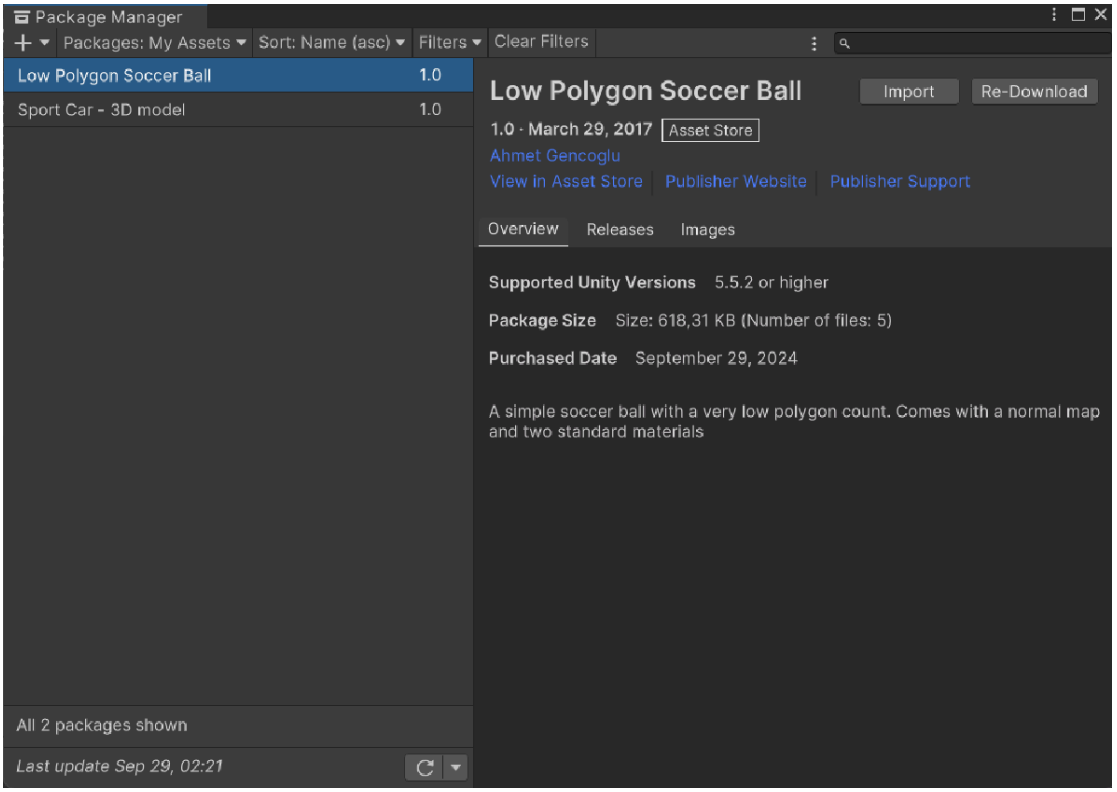


I select “Low Polygon Soccer Ball” and add it to My Assets.
After this you can see page like my:



Press the button “Open in Unity”, and in Unity Editor will see opened window

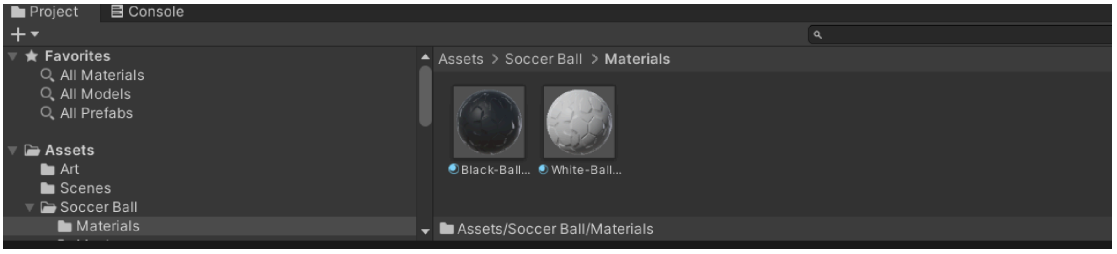
“Package Manager”:



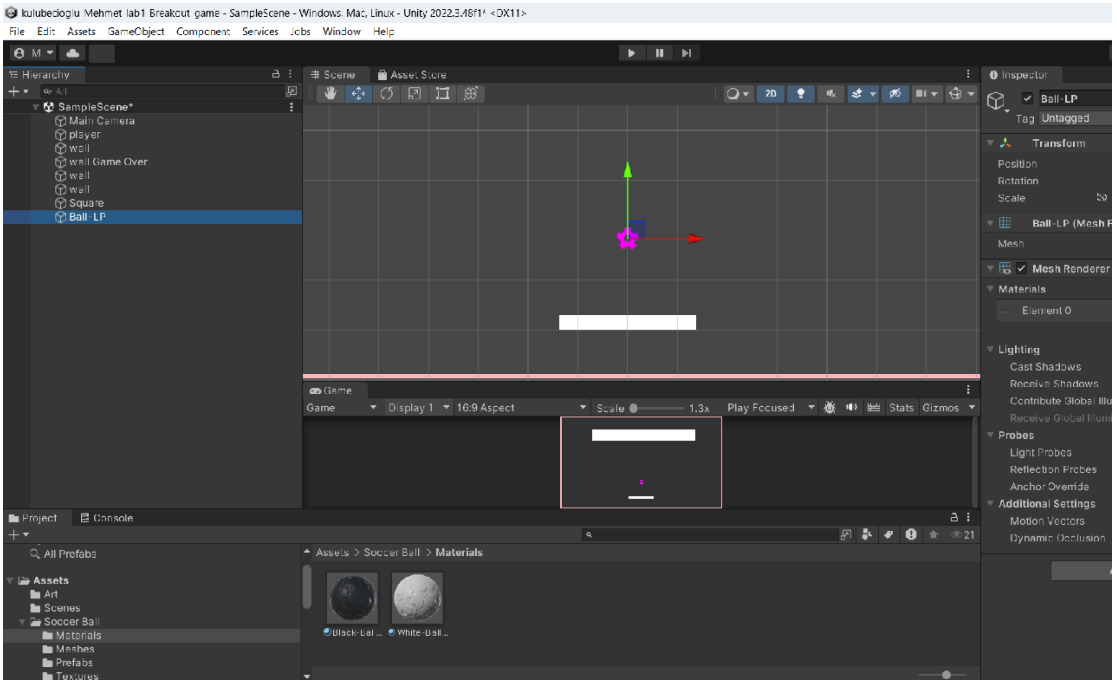
While the package downloads to my computer, a progress bar appears, and the Pause and Cancel buttons replace the Download button I clicked. When the download finishes, the Package Manager replaces Download with Re-Download, and an Import button appears next to Re-Download.

I can find more details about importing assets by following this link.

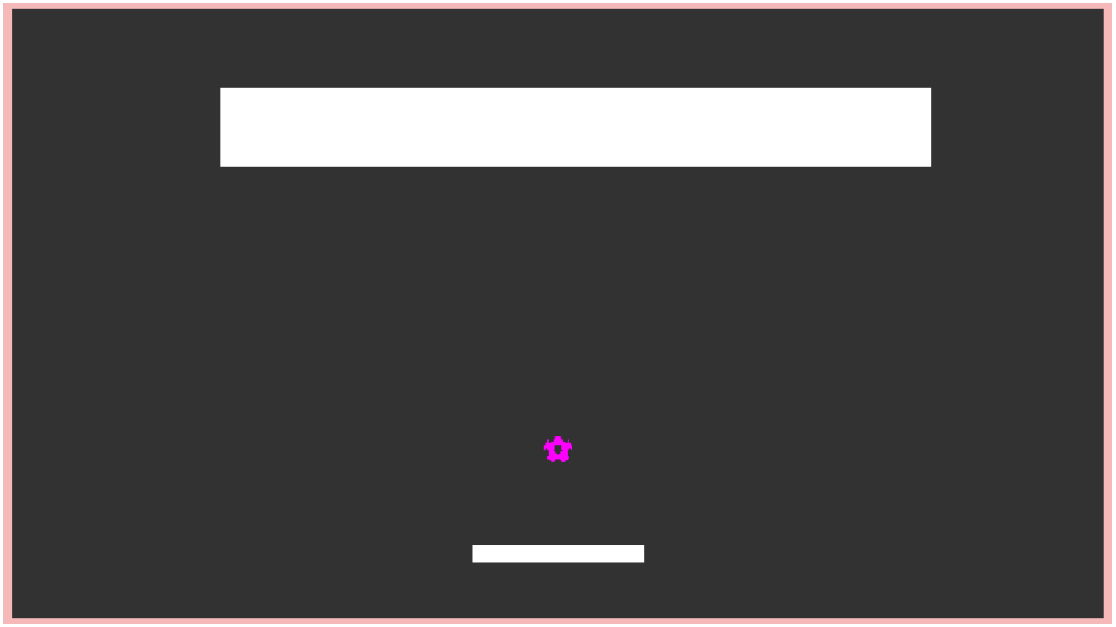
After importing, I will have a new collection in Assets called “Low Polygon Soccer Ball.”



I can drag-n-drop trees from assets to the Scene window. After that, it will be created new GameObject:



I can change the size of new objects if I need to. Next screenshot was created in the Game window (after press Run button):



important!

Since my computer is broken, I will continue to improve my game with my Macbook from here on.

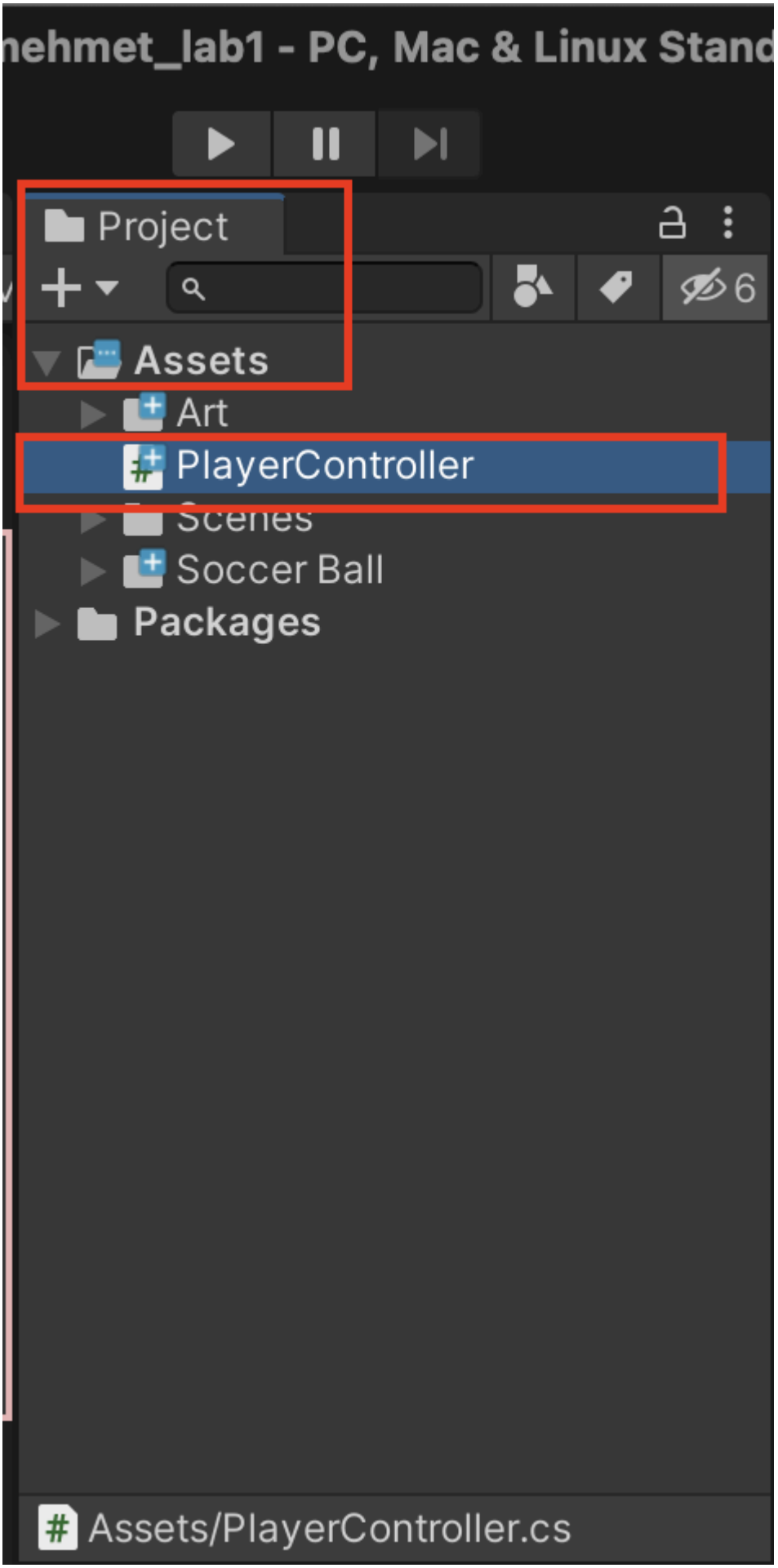
My Next Steps:

1. I will create an idea for my game.
2. I will find assets suitable for my game.
3. I will create a base Scene for my game.

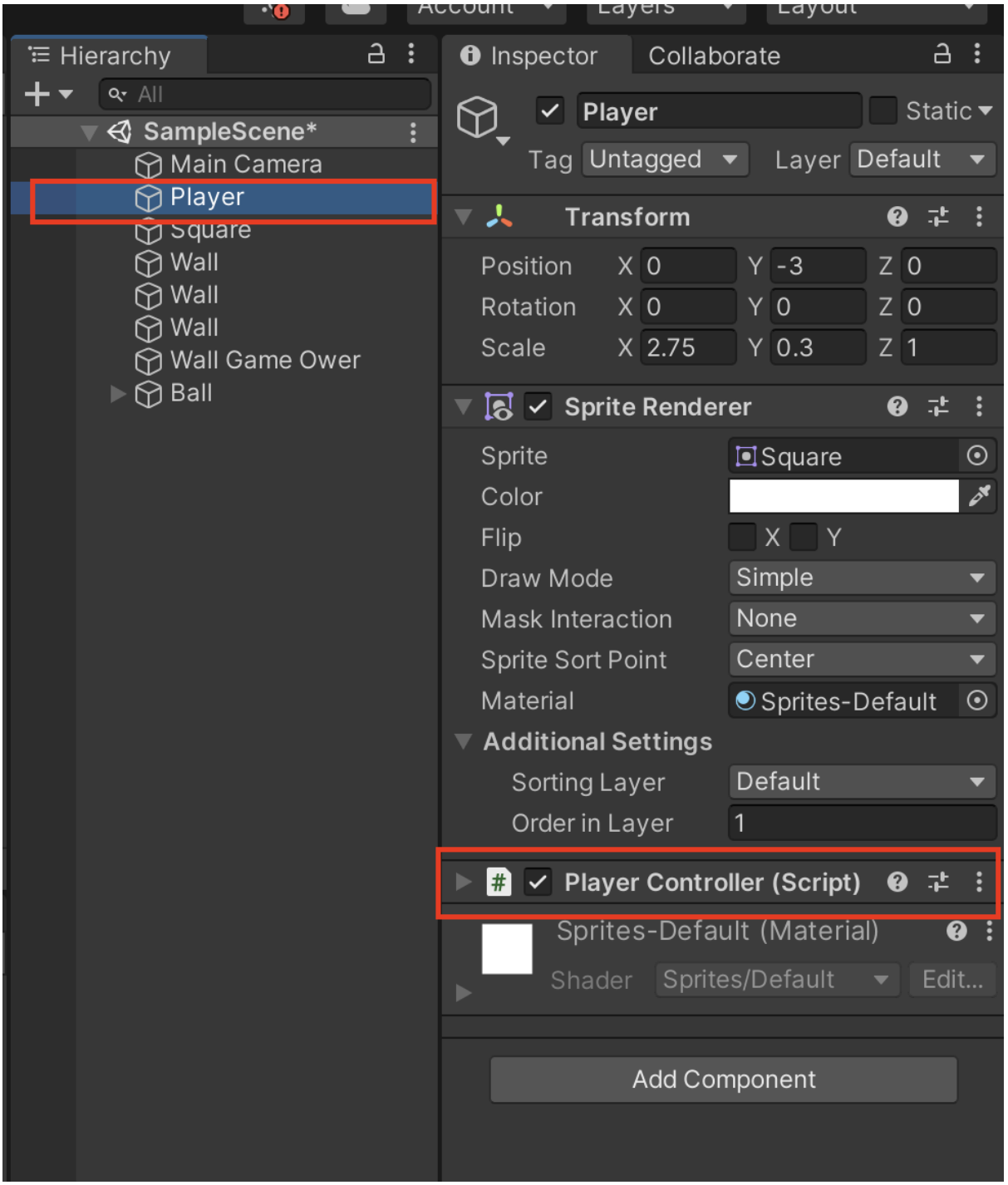
I Created a “PlayerScript”

- To create a script in Unity, I right-click in the "Assets" folder, select "Create," and then choose "C# Script." I give it a name and double-click to open it in the code editor, where I can add my code.

- After clicking the plus sign in the project section, we can click on the new script option.

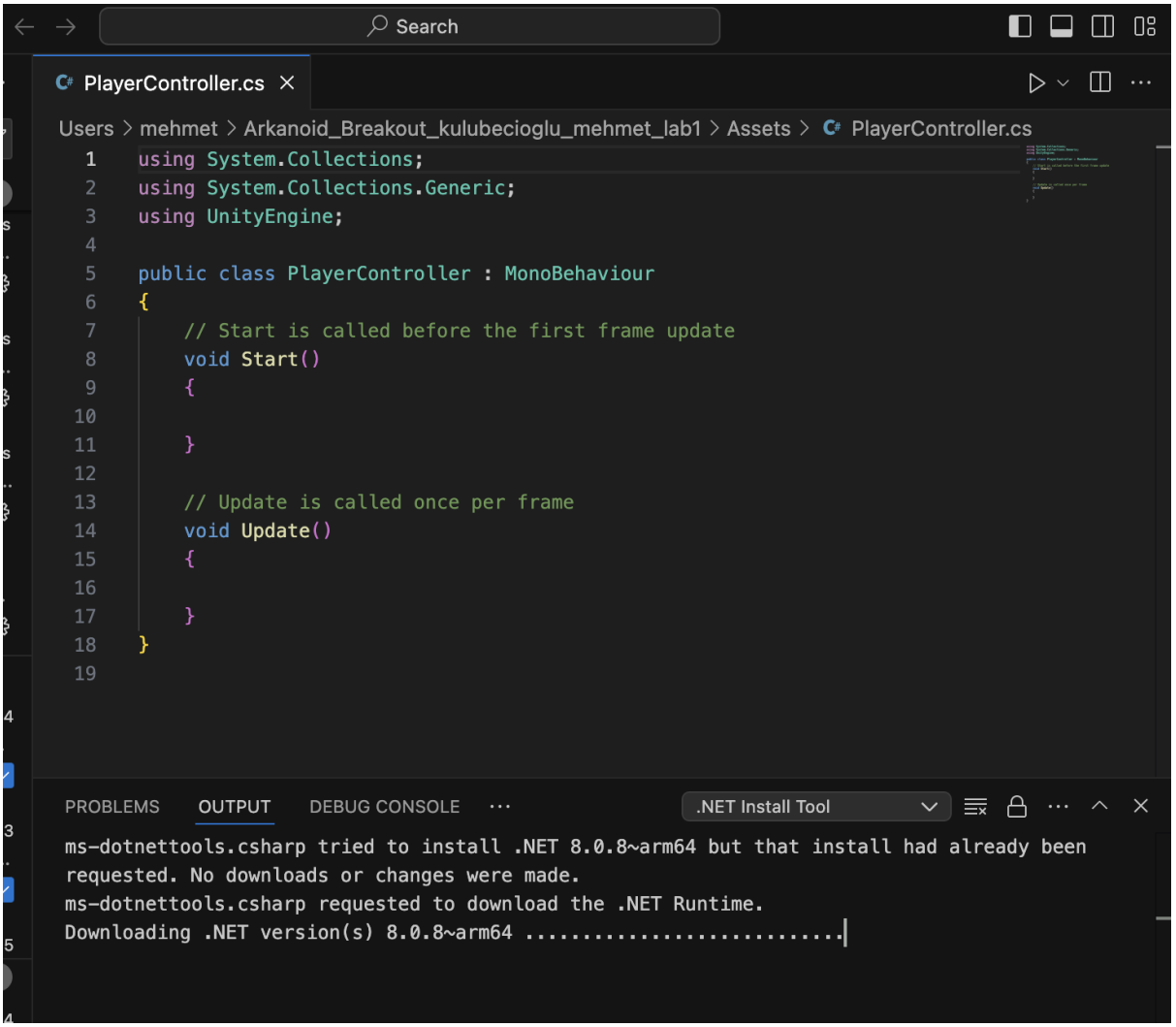

nehmet_lab1 - PC, Mac & Linux Stand
Project
6
Assets
Art
PlayerController
Scenes
Soccer Ball
Packages
Assets/PlayerController.cs

Adding playerScript to the player:

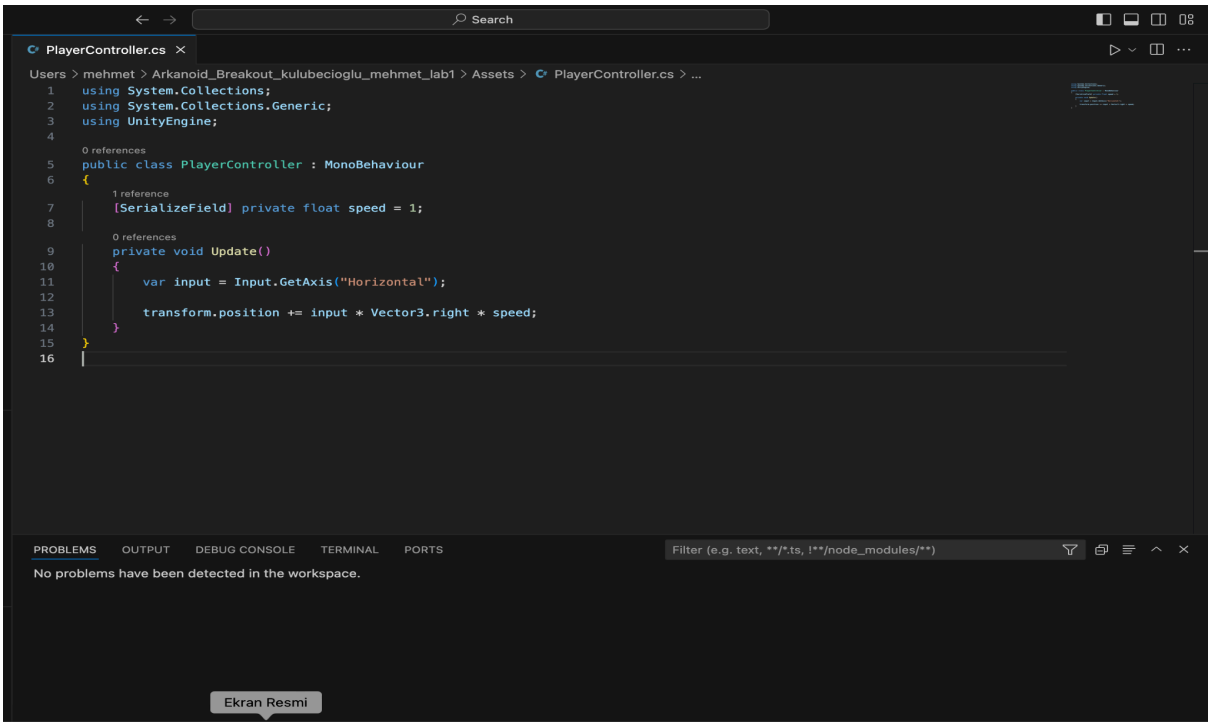


- **After setting the name of the script to PlayerController, I drag it into the player and it is added automatically.**

- **I open my newly created PlayerScript file.**

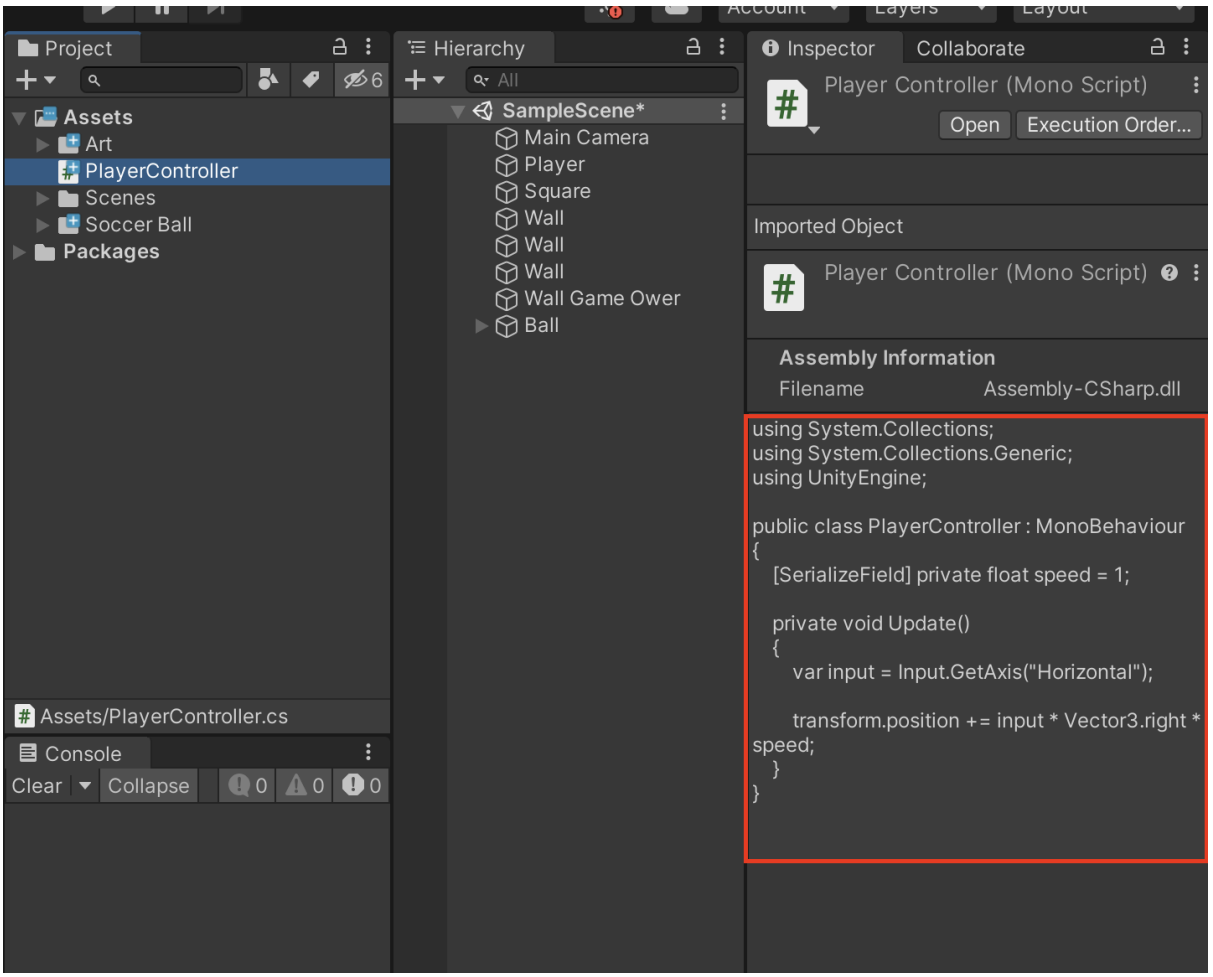


- **I wrote the following codes to move the player left and right.**



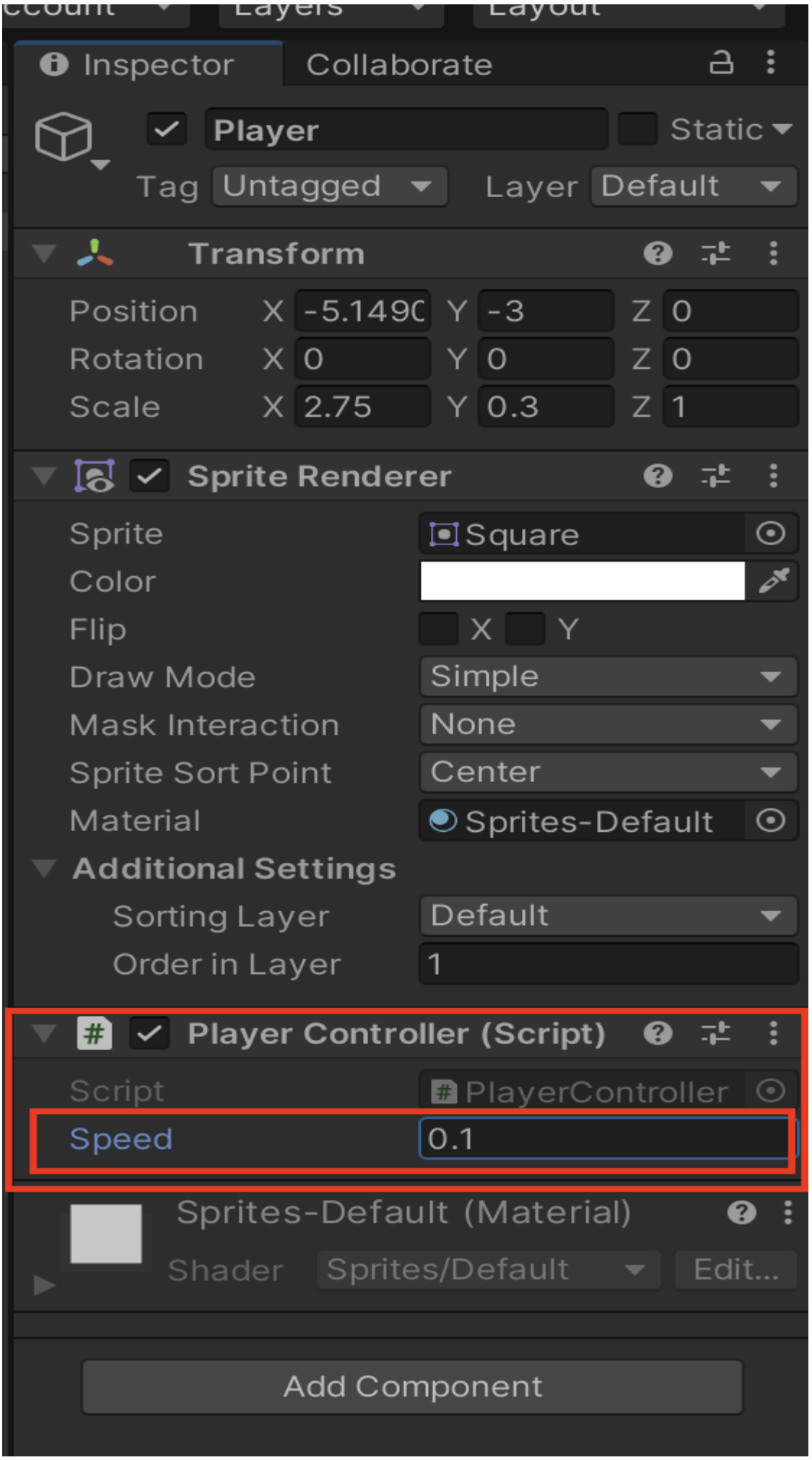
We can also see our script codes on Unity:

- **For this, after clicking on PlayerScript, we should look at the Inspecor section. I give an example in the image below.**

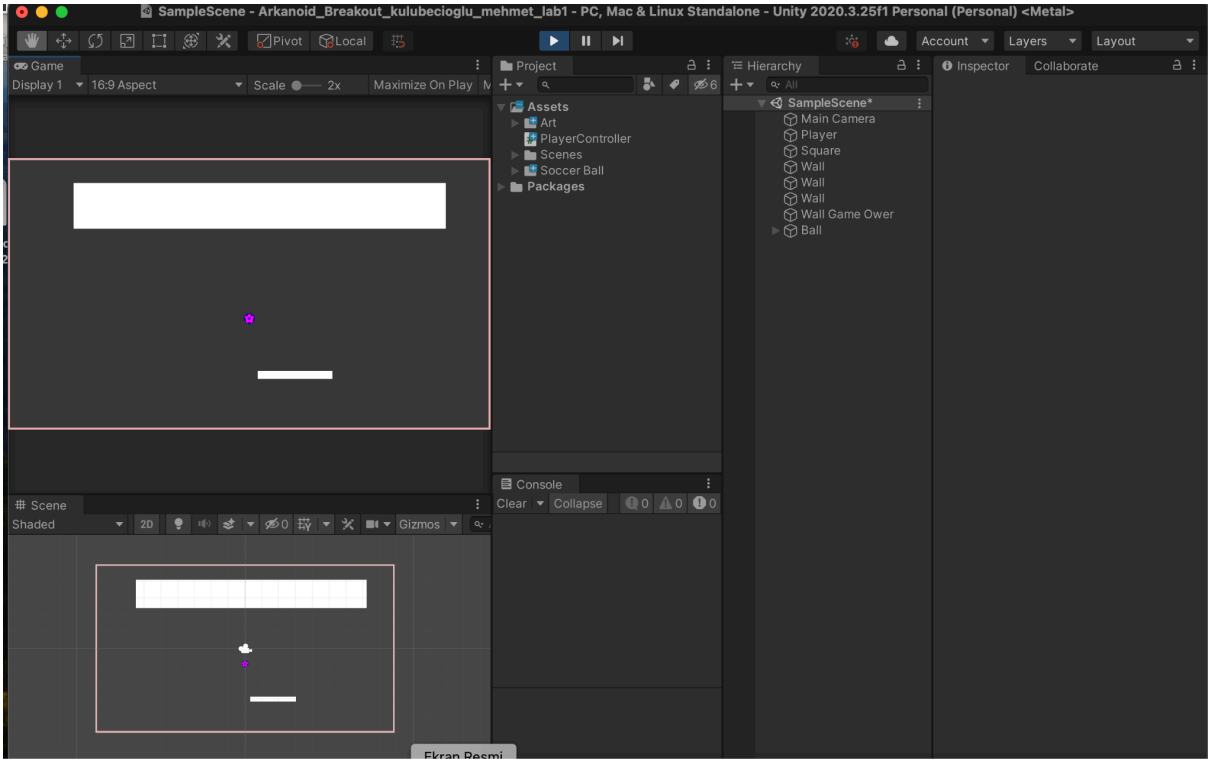


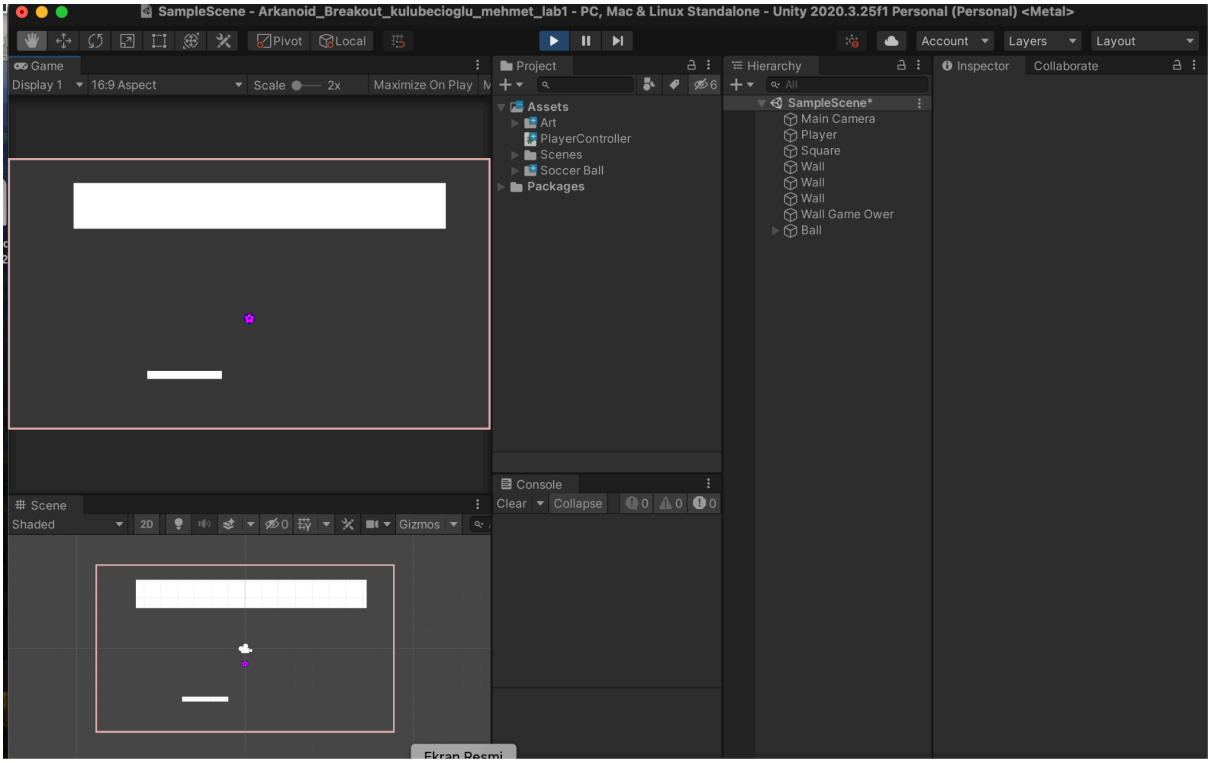
Adjusting player speed:

- **To adjust the player speed, we must enter a value into the "Speed" option under the PlayerControl script. I give an example in the photo below.**


Inspector
Collaborate
Player
Static
Tag
Untagged
Layer
Default
Transform
Position
X -5.1490
Y -3
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 2.75
Y 0.3
Z 1
Sprite Renderer
Sprite
Square
Color
Flip
X
Y
Draw Mode
Simple
Mask Interaction
None
Sprite Sort Point
Center
Material
Sprites-Default
Additional Settings
Sorting Layer
Default
Order in Layer
1
Player Controller (Script)
Script
PlayerController
Speed
0.1
Sprites-Default (Material)
Shader
Sprites/Default
Edit...
Add Component

- **Now let's check if the player moves or not**



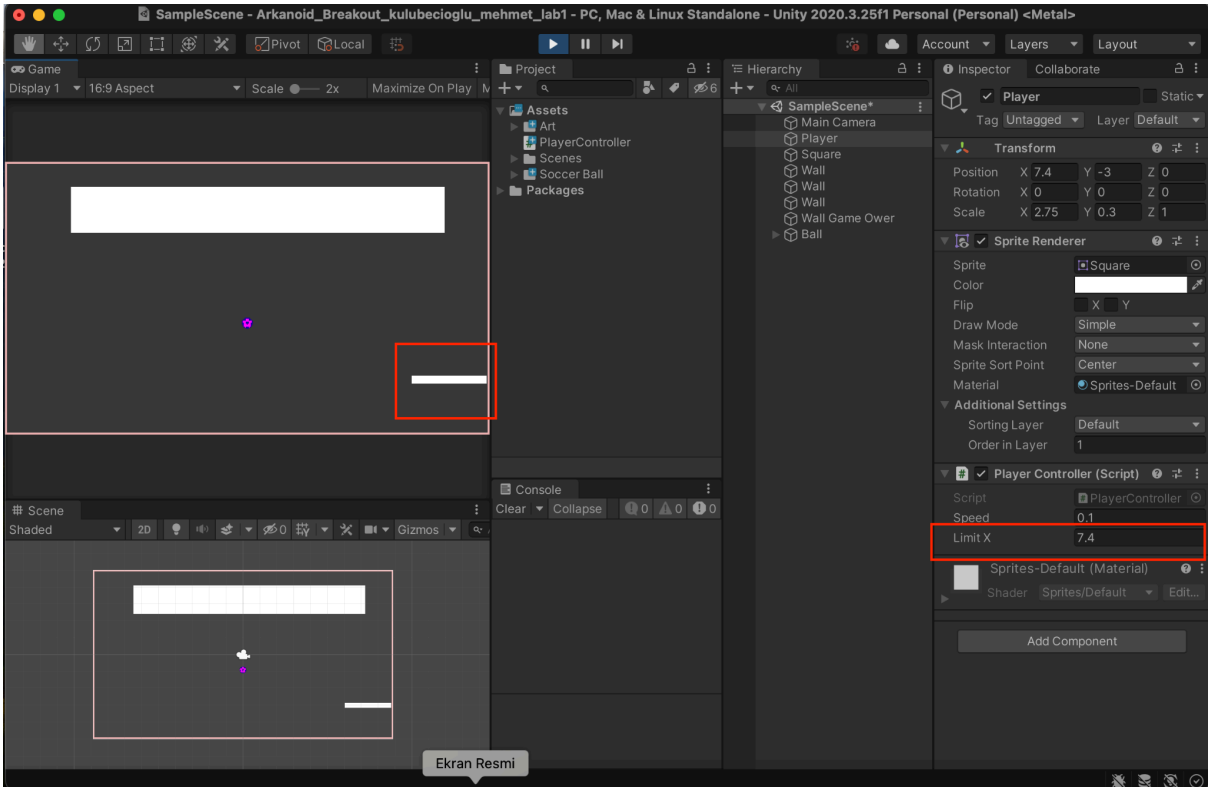


- **Yes my player moves left and right**

Now I am adding the codes below that enable my player to move left and right.
I also use limitx so that my player does not go out of the frame.

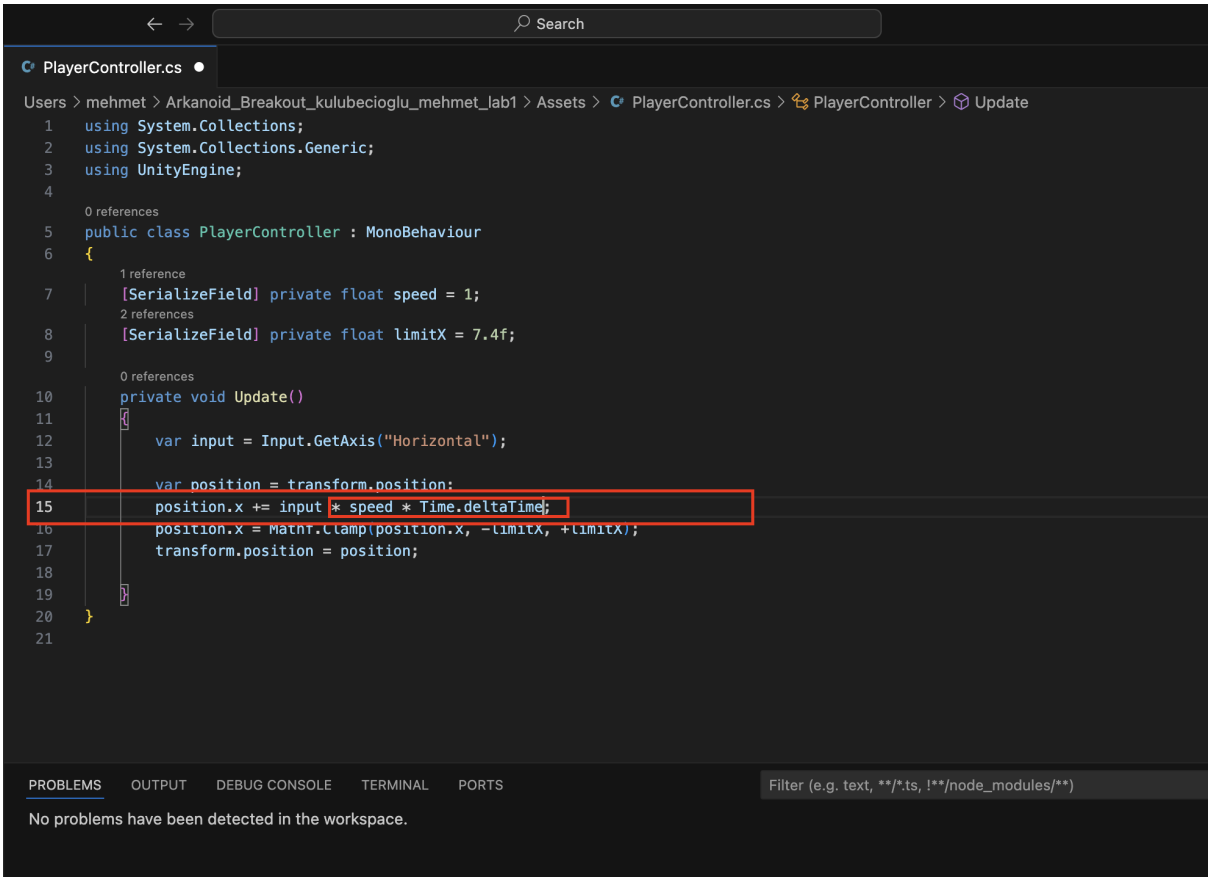
```
PlayerController.cs
Users > mehmet > Arkanoid_Breakout_kulubecioglu_mehmet_lab1 > Assets > PlayerController.cs > PlayerController > Update
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    0 references
5   public class PlayerController : MonoBehaviour
6   {
        1 reference
7       [SerializeField] private float speed = 1;
        2 references
8       [SerializeField] private float limitX = 7.4f;
9
        0 references
10      private void Update()
11      {
12          var input = Input.GetAxis("Horizontal");
13
14          var position = transform.position;
15          position.x += input * speed;
16          position.x = Mathf.Clamp(position.x, -limitX, +limitX);
17          transform.position = position;
18
19      }
20  }
21
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
Filter (e.g. text, **/*.ts, !**/node_modules/**)
```

Now, to check whether this code works, I open Unity and start the game.



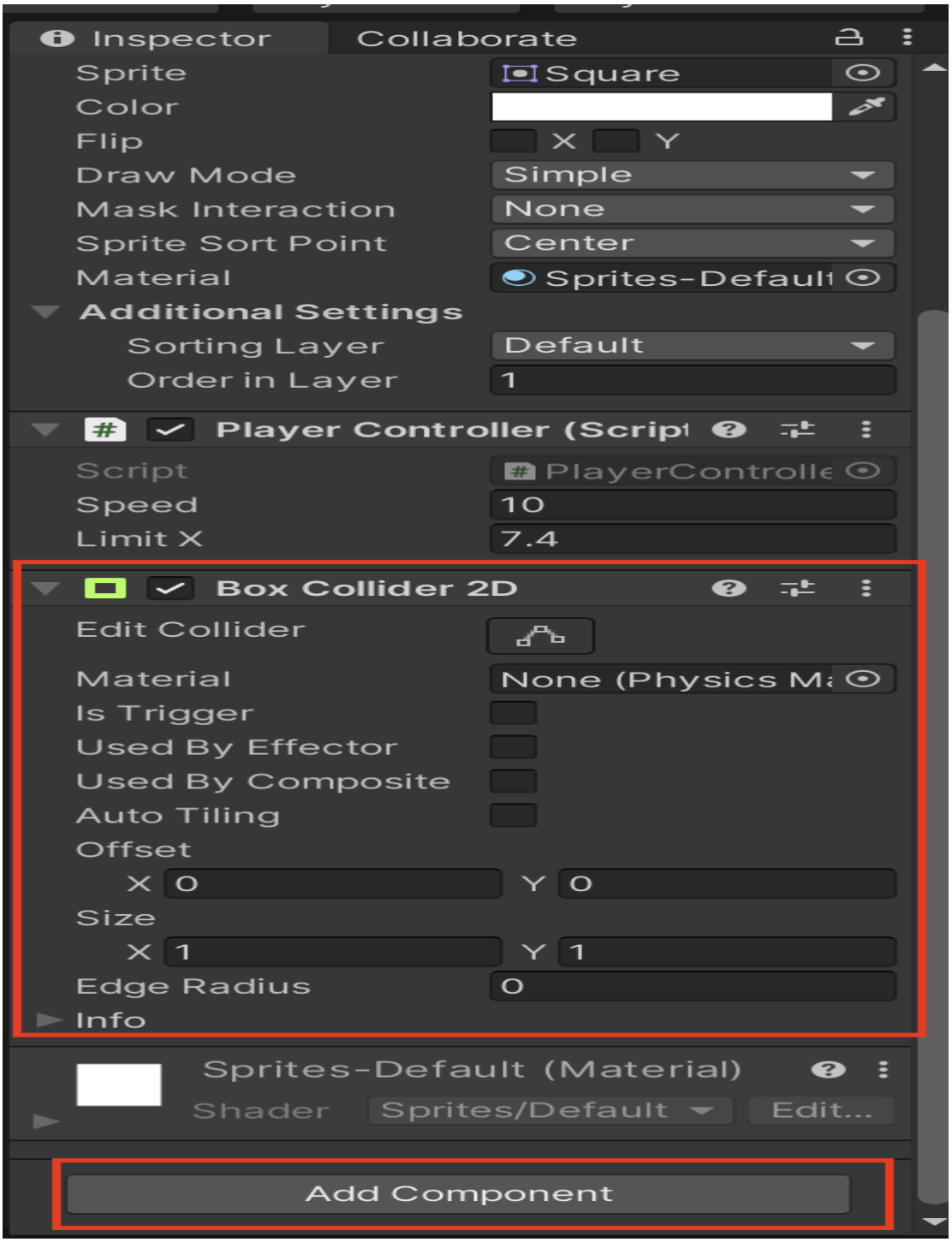
Here I see that my player does not go out of frame, so my code works correctly.

- **Now I am writing the code below to ensure that the speed of my player is the same on every computer.**


Search
PlayerController.cs
Users > mehmet > Arkanoid_Breakout_kulubecioglu_mehmet_lab1 > Assets > PlayerController.cs > PlayerController > Update
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
0 references
public class PlayerController : MonoBehaviour
{
1 reference
[SerializeField] private float speed = 1;
2 references
[SerializeField] private float limitX = 7.4f;
0 references
private void Update()
{
var input = Input.GetAxis("Horizontal");
var position = transform.position;
position.x += input * speed * Time.deltaTime;
position.x = Mathf.Clamp(position.x, -limitX, +limitX);
transform.position = position;
}
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, **/*.ts, !**/node_modules/**)
No problems have been detected in the workspace.

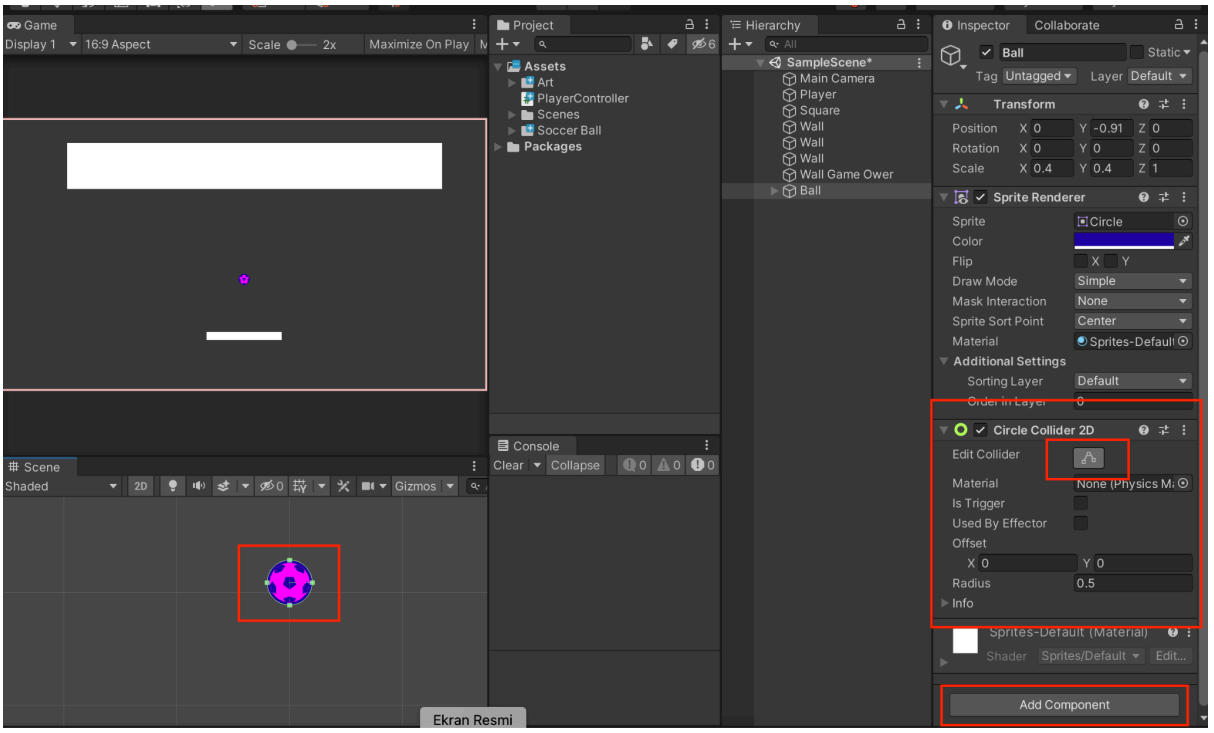
Basic Character Movement & Physics

Now I will add “Box Collider” for the ball to hit

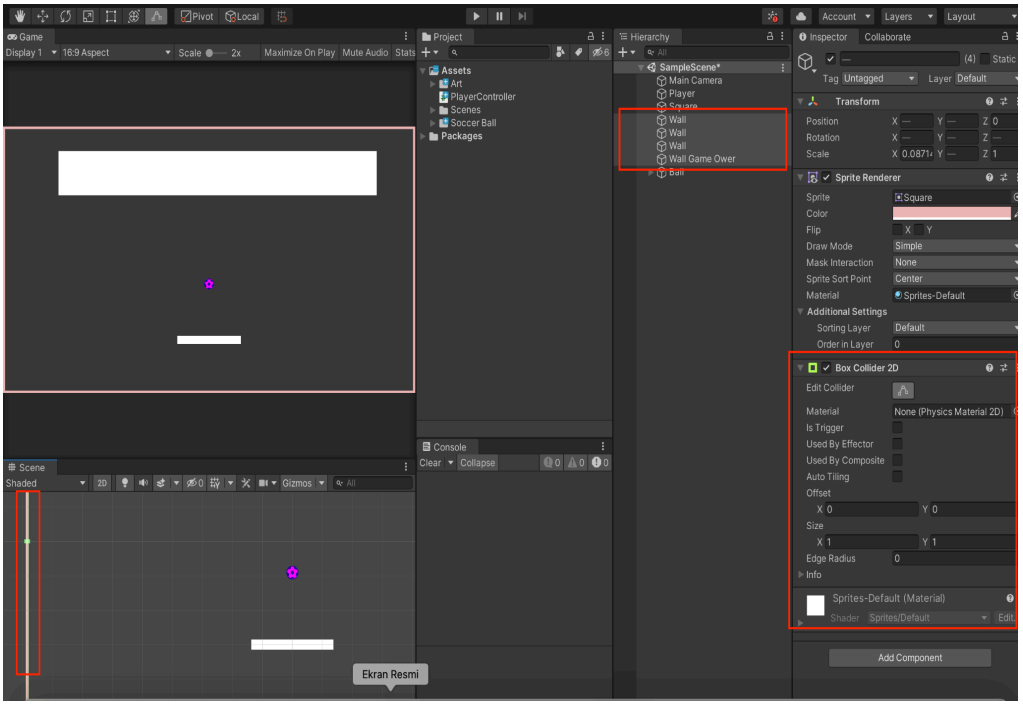


Interactions and Object Collection

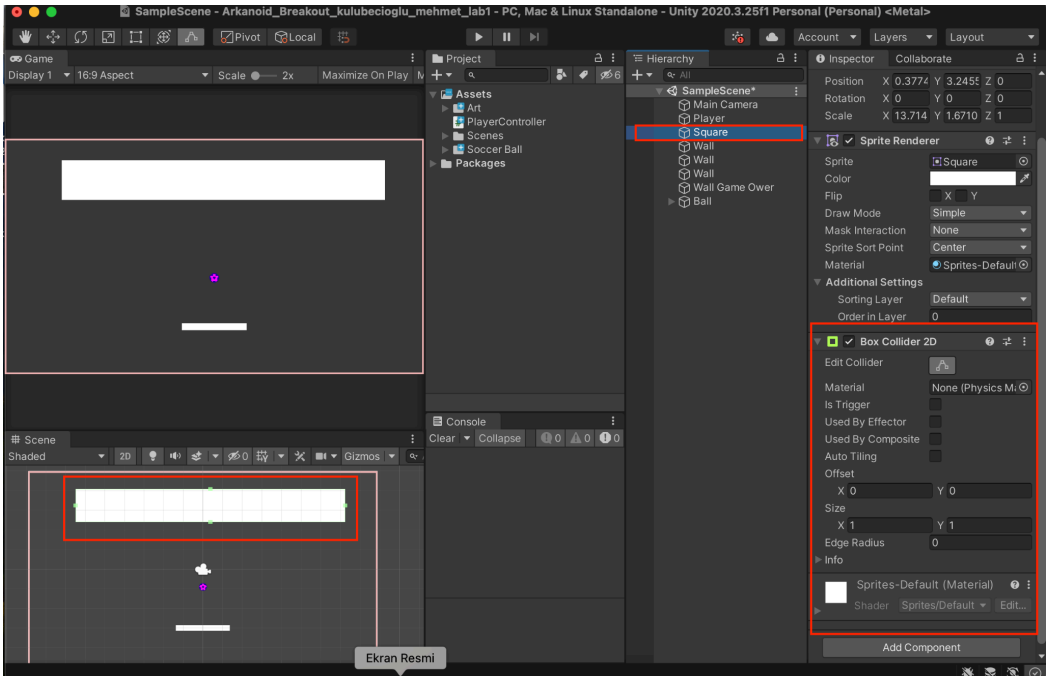
- **adding circle collider to ball for collision**



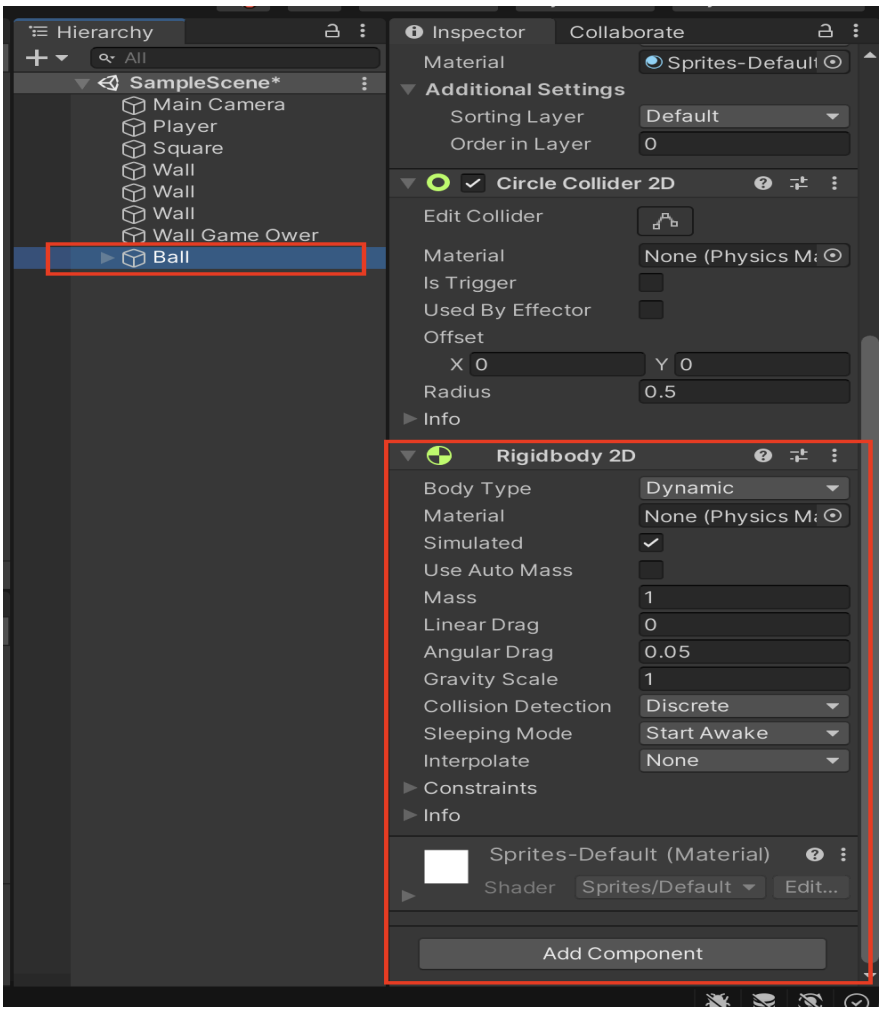
- **adding Box Collider to walls for collision**



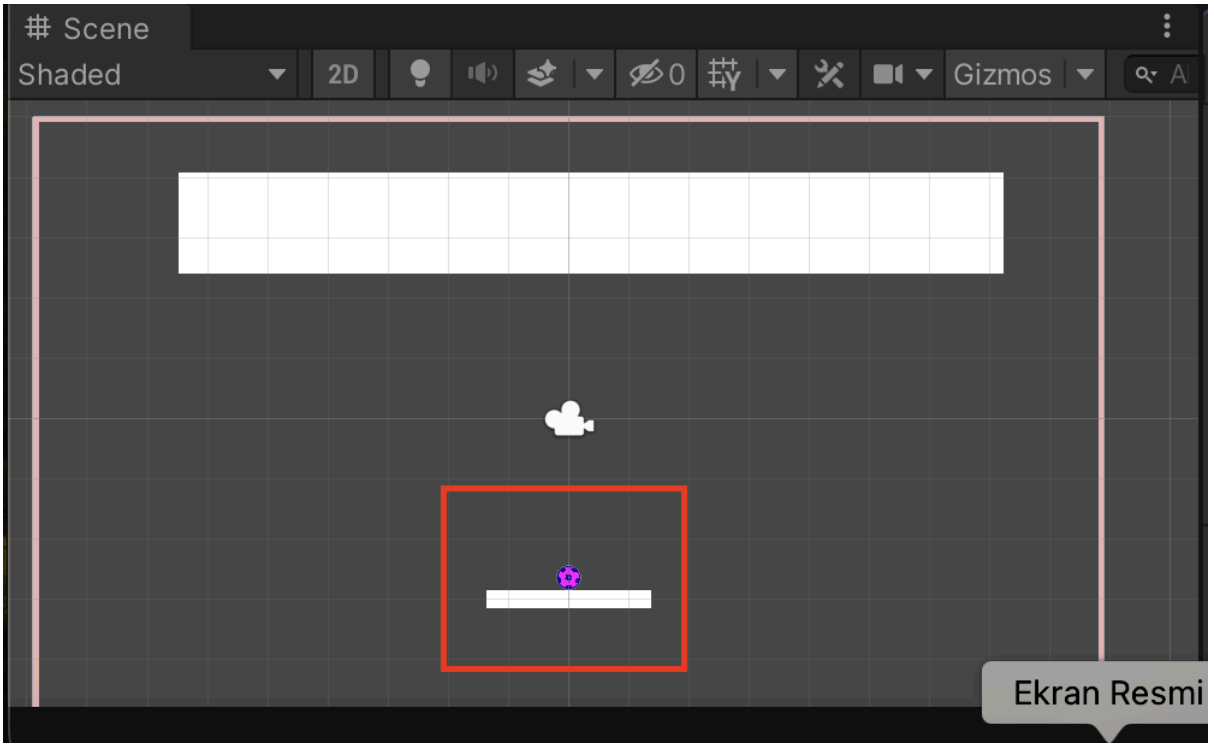
- **Now I added box collider to the obstacles for collision.**



- **now i add a new rigidbody to the ball for the ball to drop**



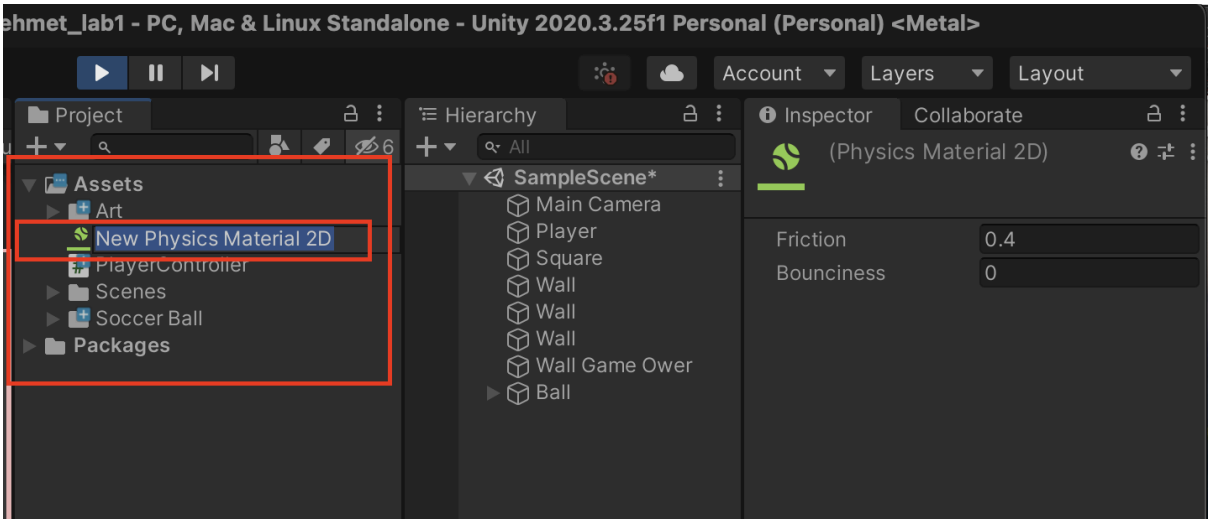
- **Now let's check whether the ball dropped by starting the project**



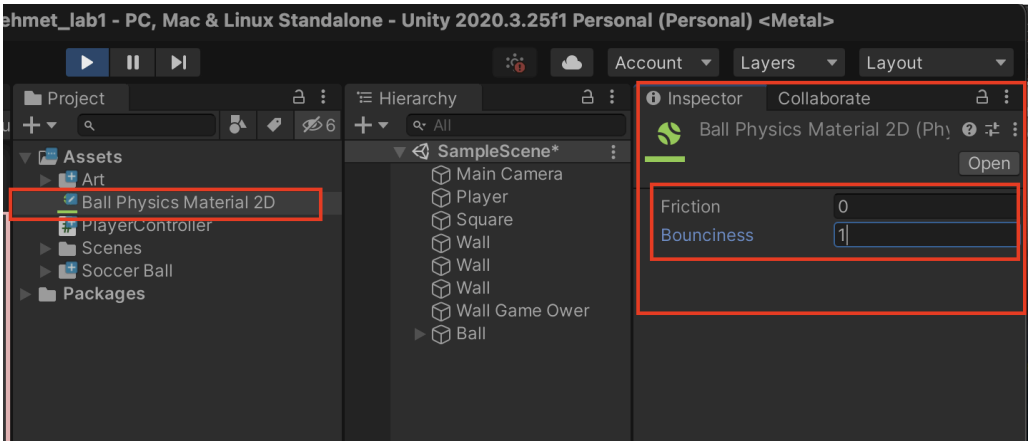
- **I started the project and observed that the ball dropped, so we have no problems so far.**

Adding physics material to give physical properties to the ball:

- **We can open it by clicking add physics Material 2D in the project window.**

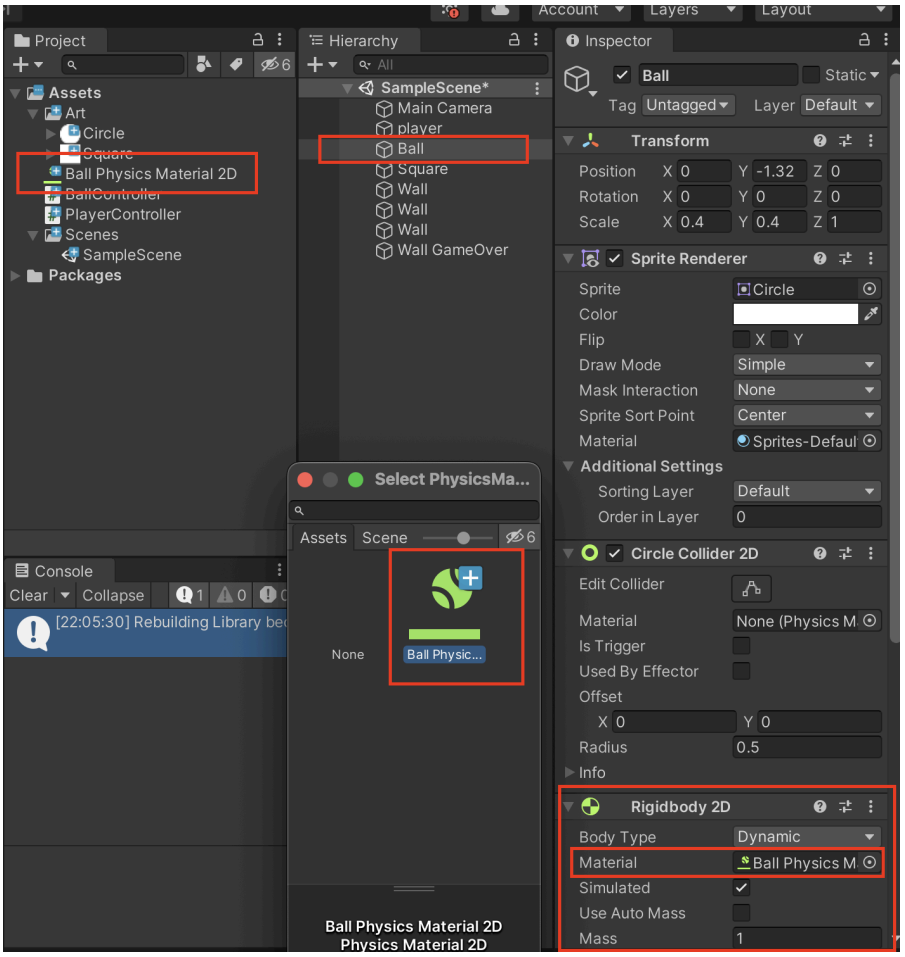


- **Add friction and bounce values**

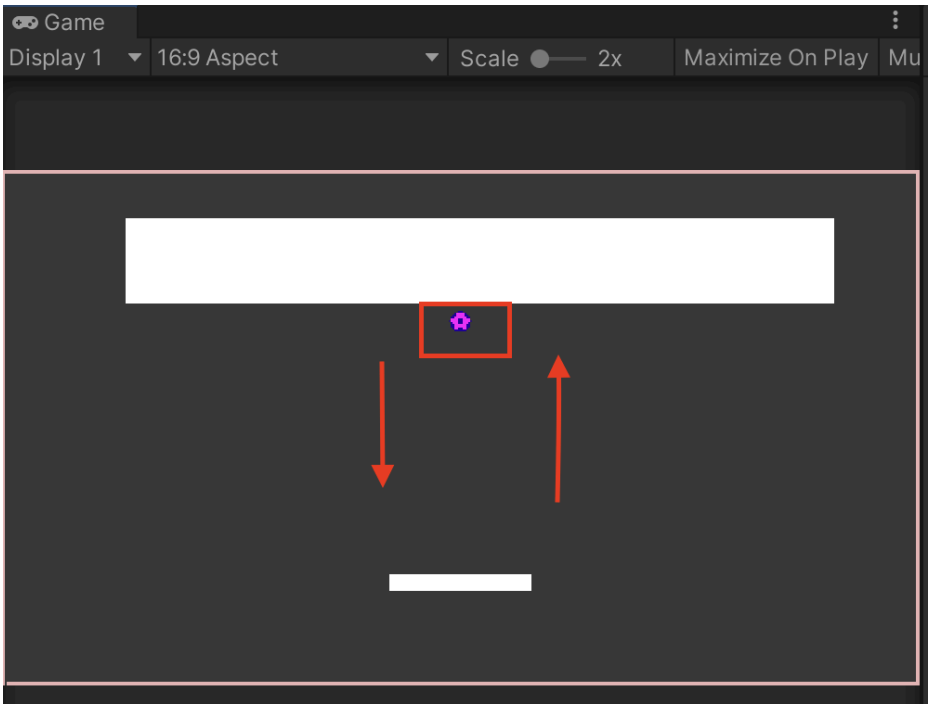


Adding the physics materials we created to the ball:

- **After clicking on the ball, I add it to the Material section in the Rigidbody under the Inspector section.**

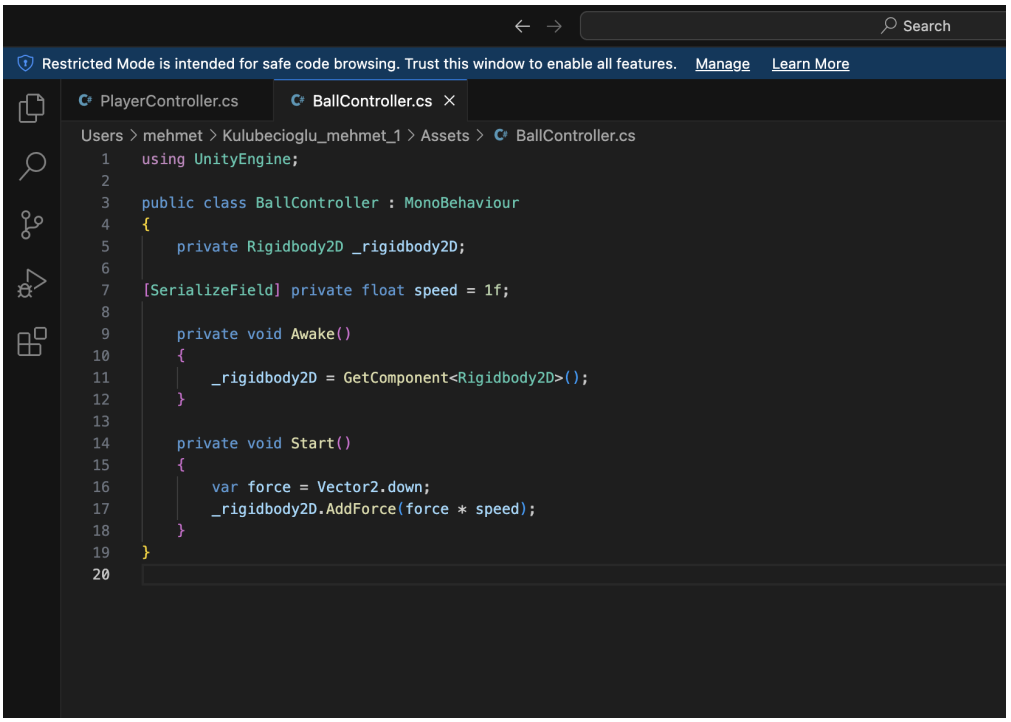


- **After the process, I will start the game and check whether the ball bounces or not.**

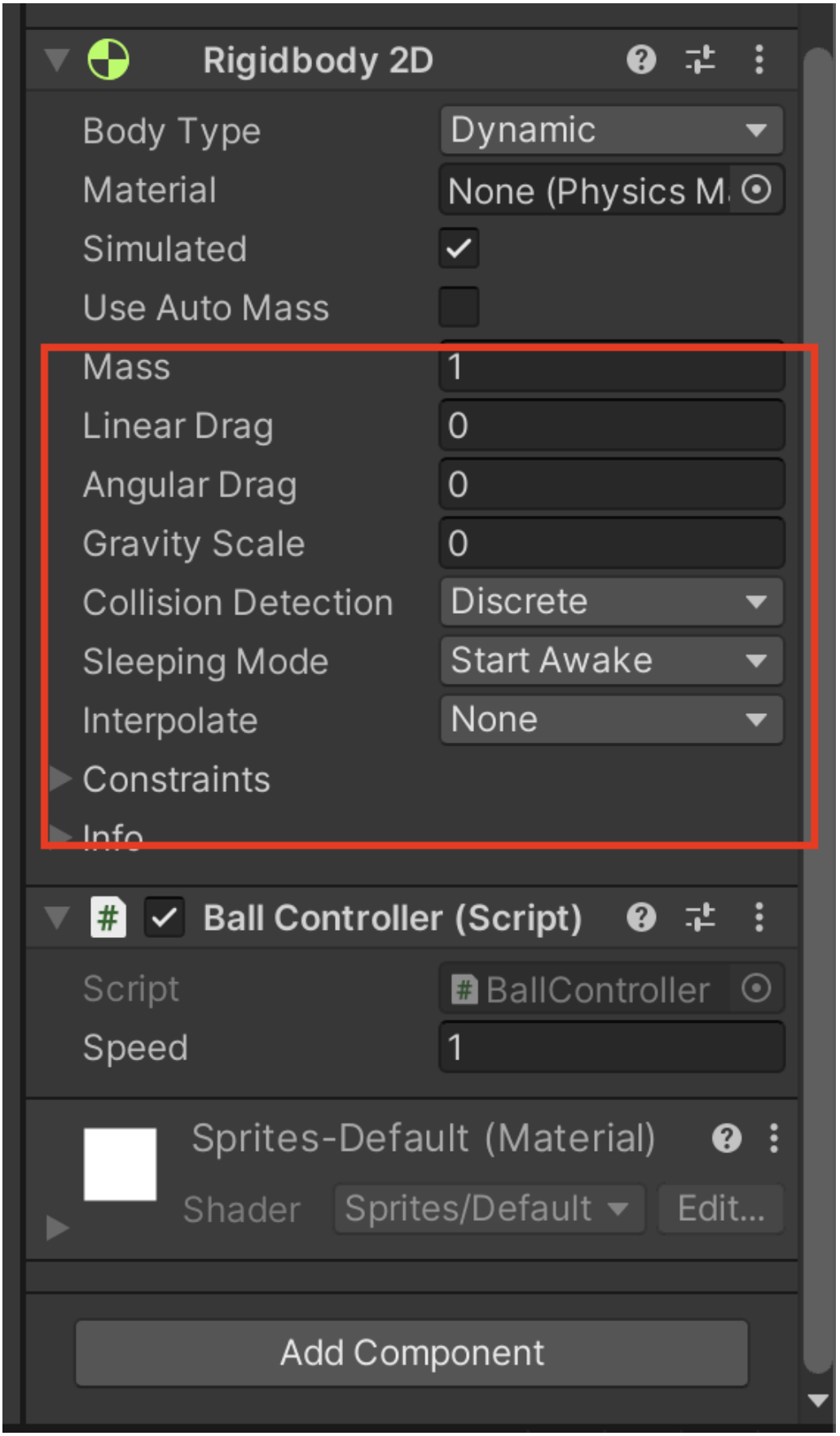


After starting the game, the project worked as I wanted and the ball started to move up and down.

BallController script:

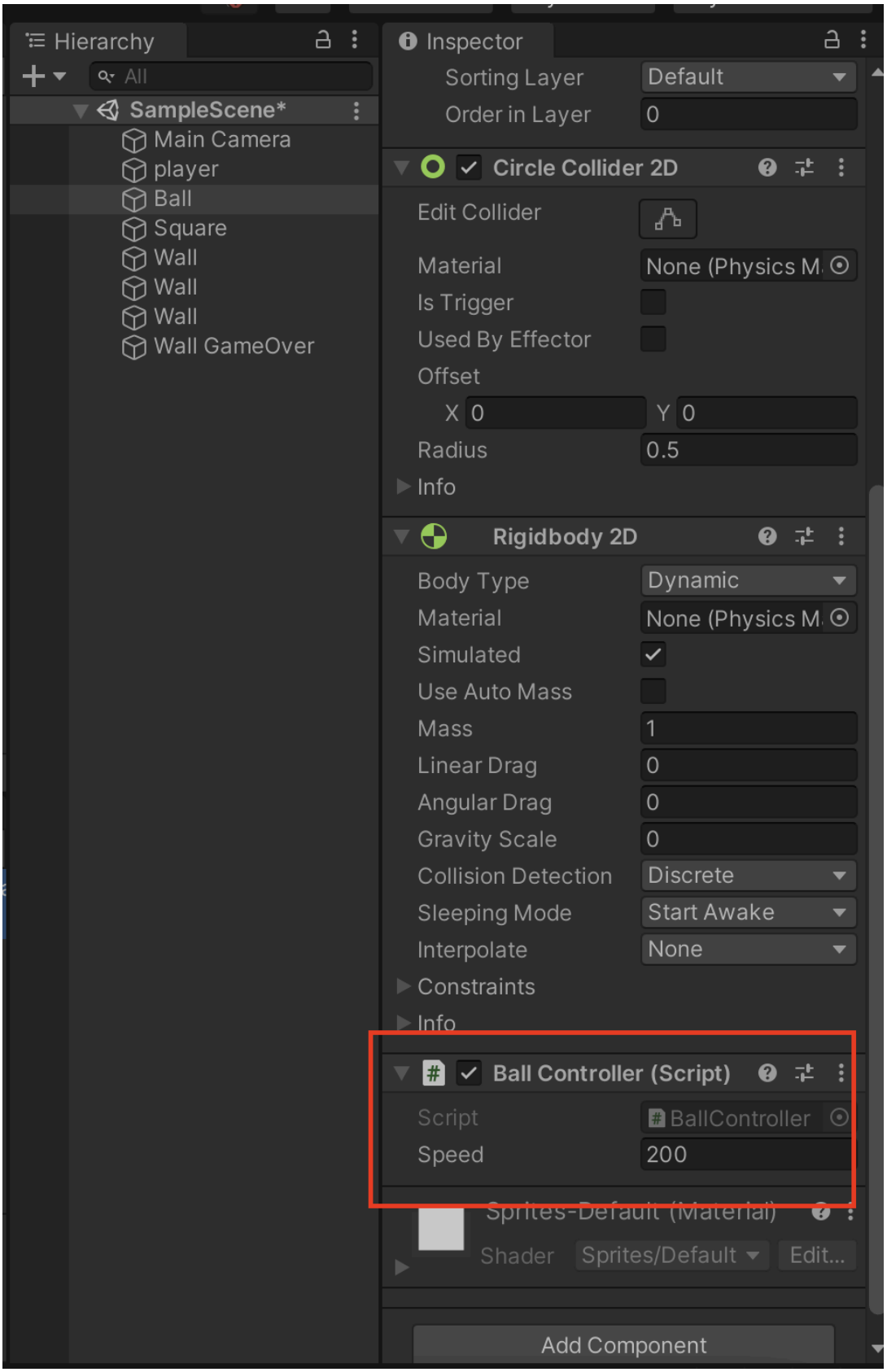


- **adjusting physics rules such as rubbing weight on the ball:**

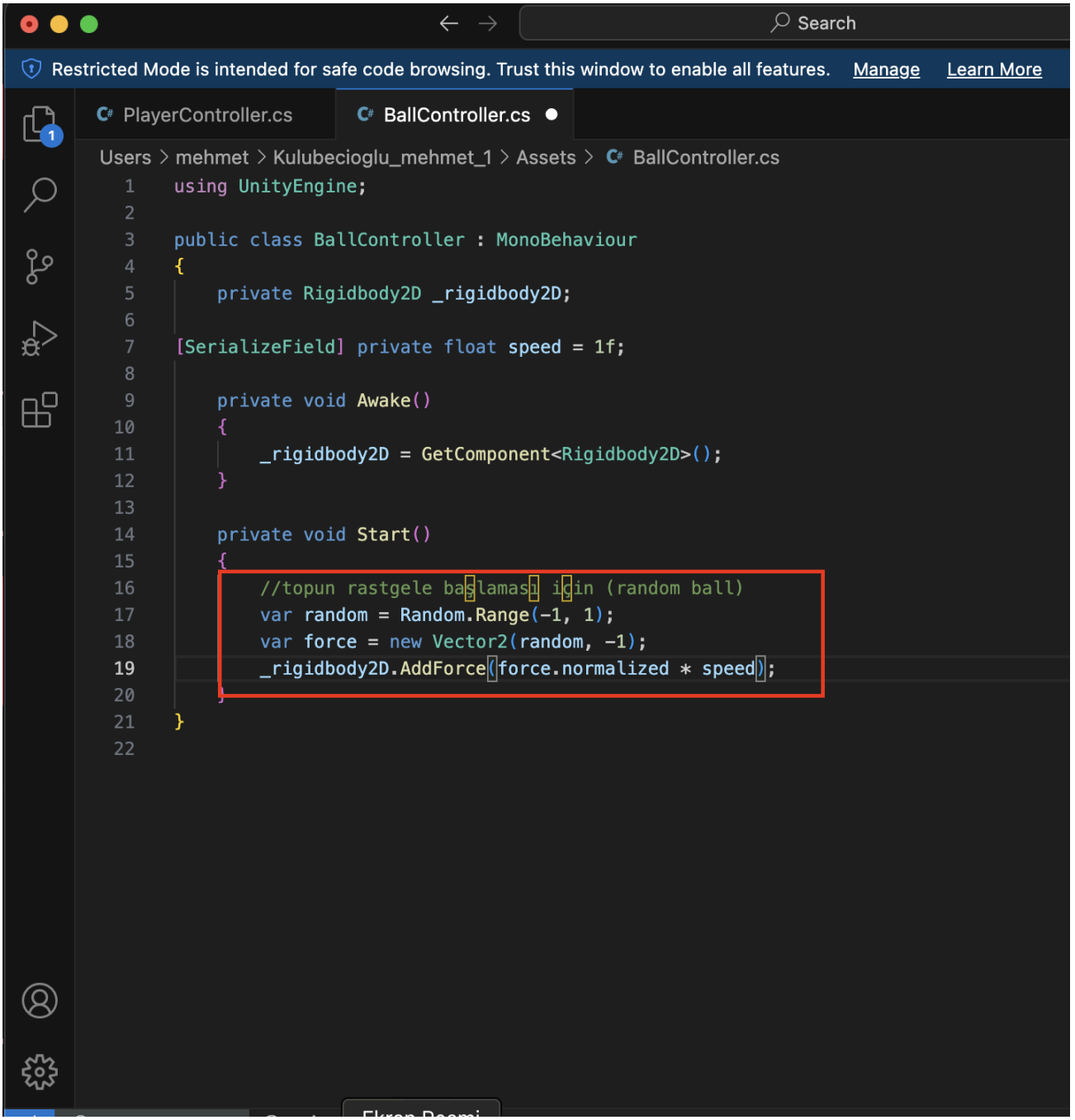


Adjusting the speed of the ball:

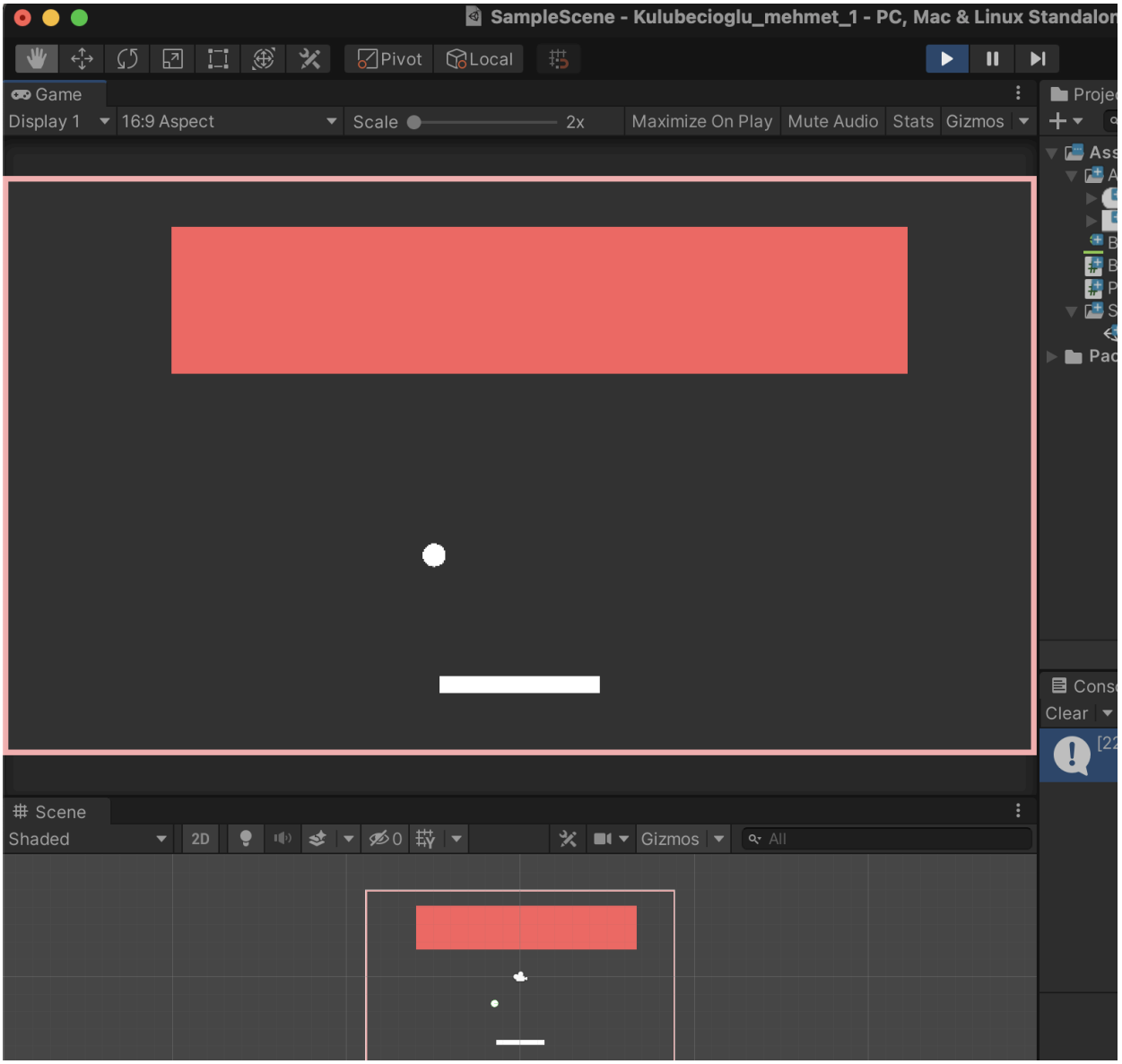
- **I entered the speed as 200**



Starting the ball randomly to the right or left:



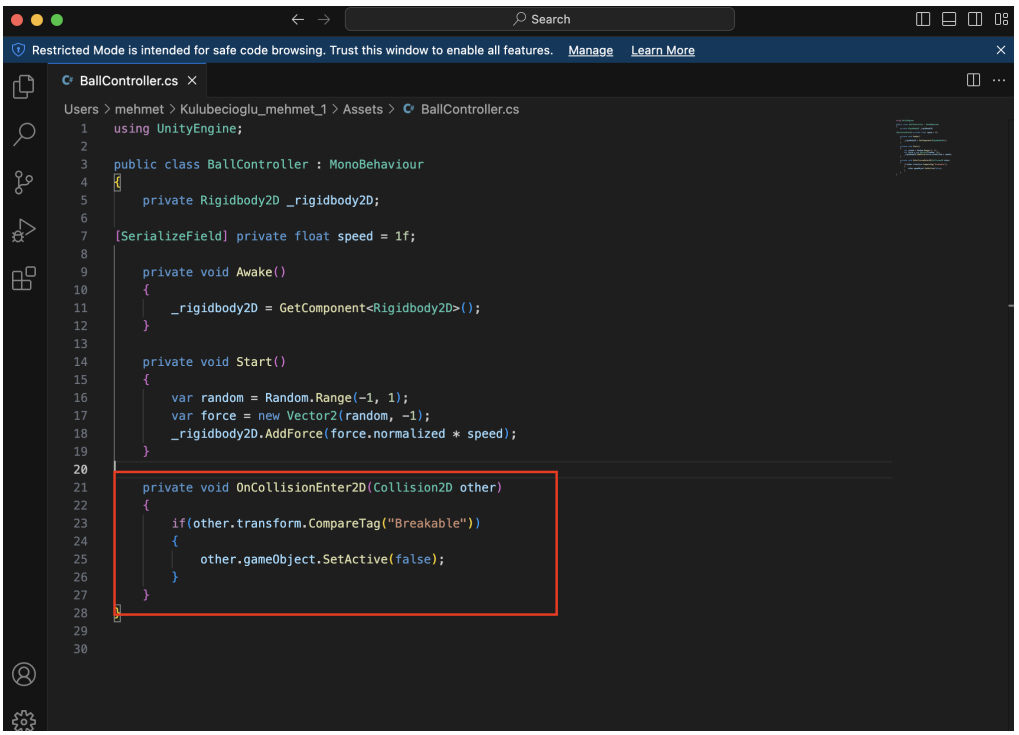
Now I start the game to check if the code works as I want:



Yes, my ball is not going straight down but to the right or left, so my code worked as I wanted.

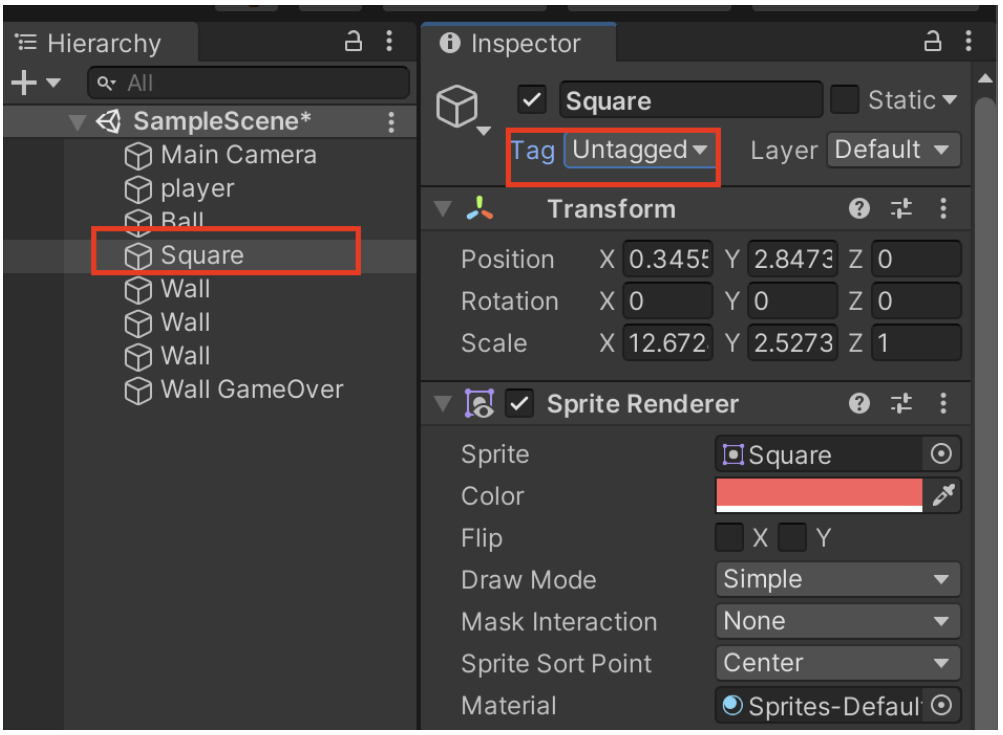
Interactions

- **Now I want the blocks hit by the balls to disappear, for this I open the BallController script and write the necessary codes.**

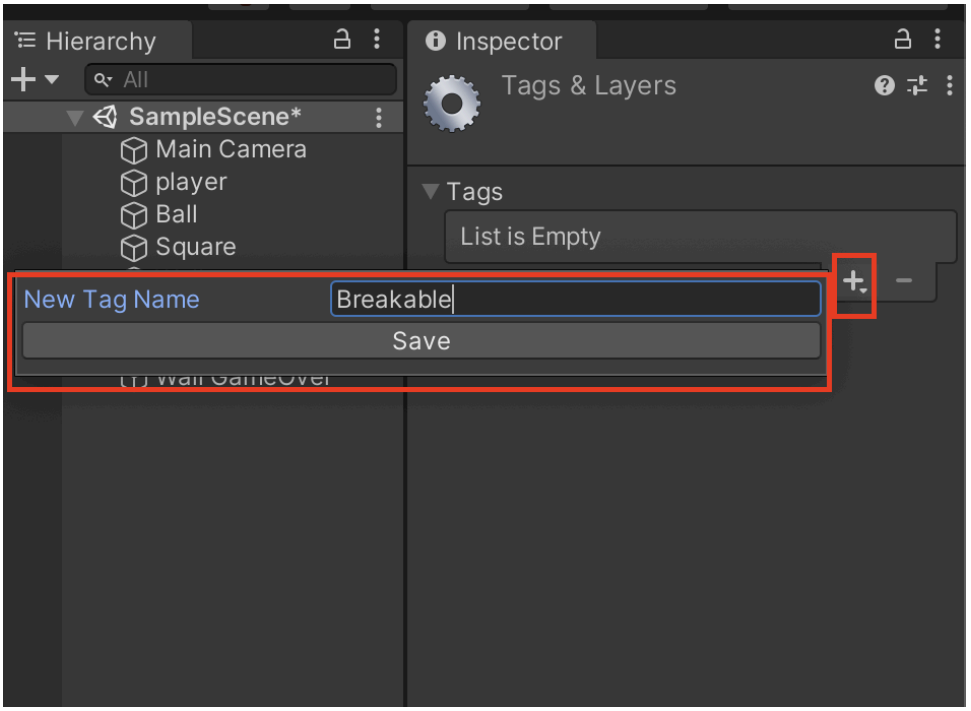


breaking blocks:

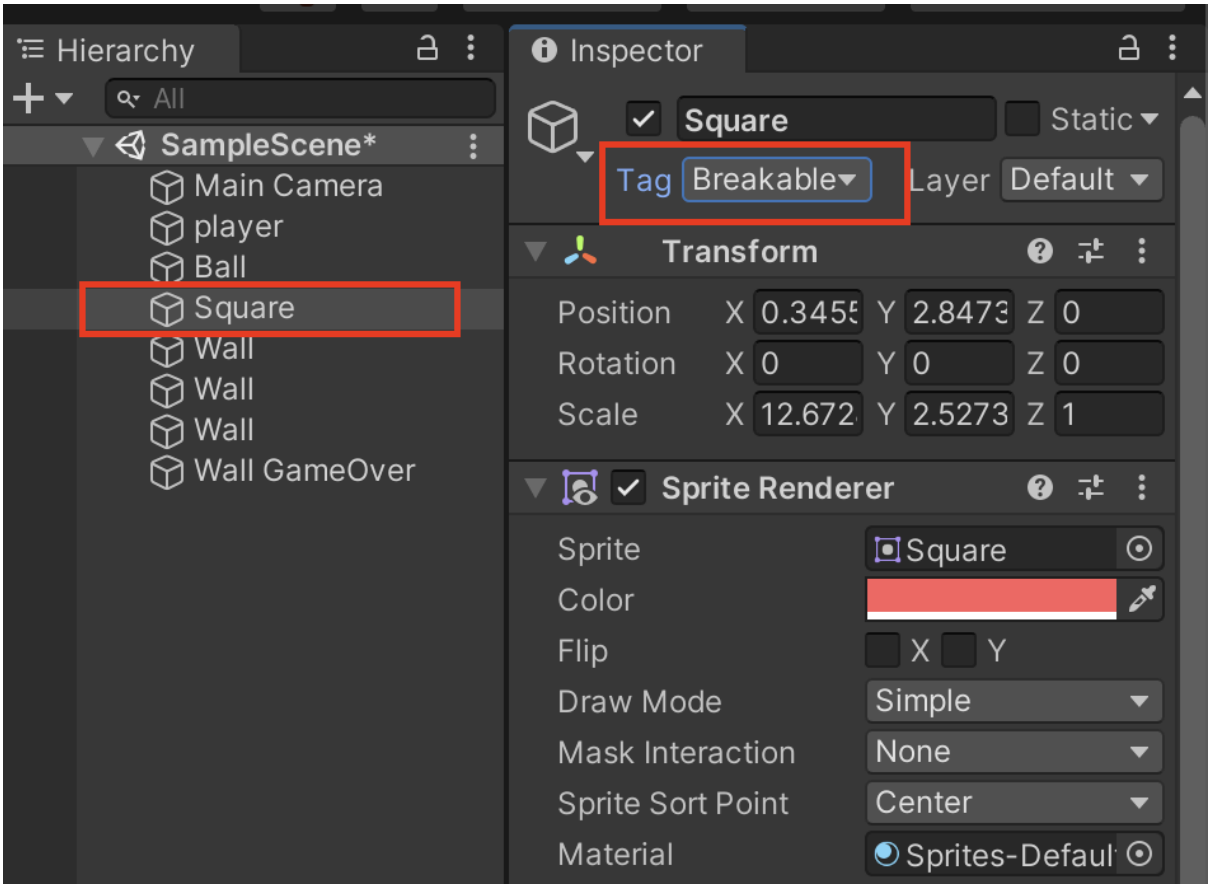
- **Now I go into Square to break the blocks and make the necessary settings in the inspector section**



Creating breakable:

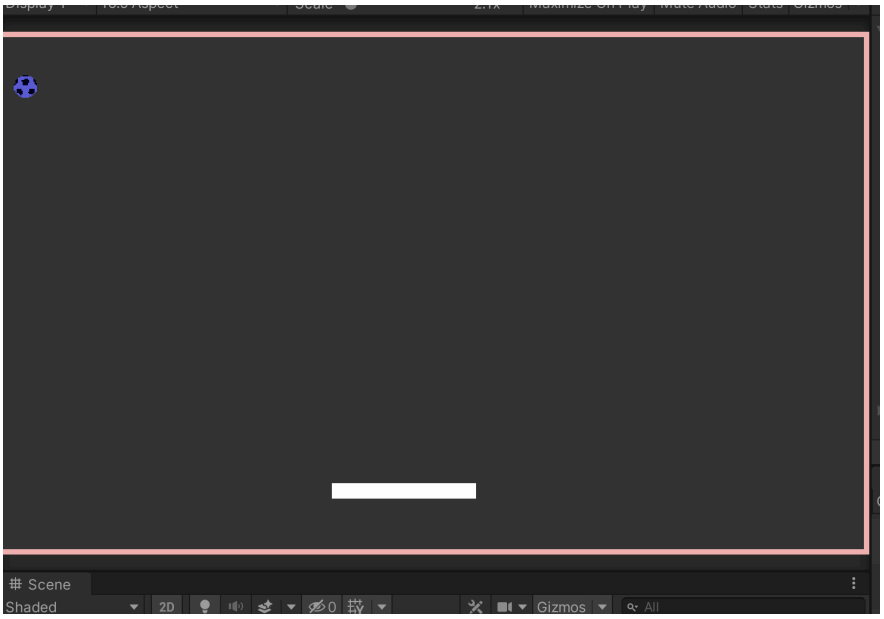


- **Now I will transfer breakable to the object**



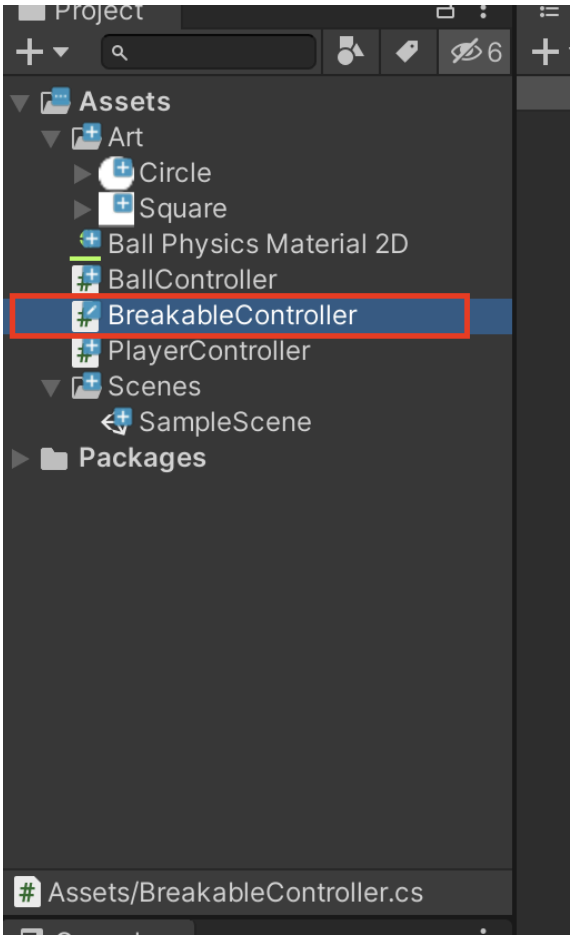
I usually add things by drag and drop.

- **Now, at the end of all of this, when the ball touches the Breakable, the breakable should disappear and after the ball hits the breakable, the ball should bounce back.**

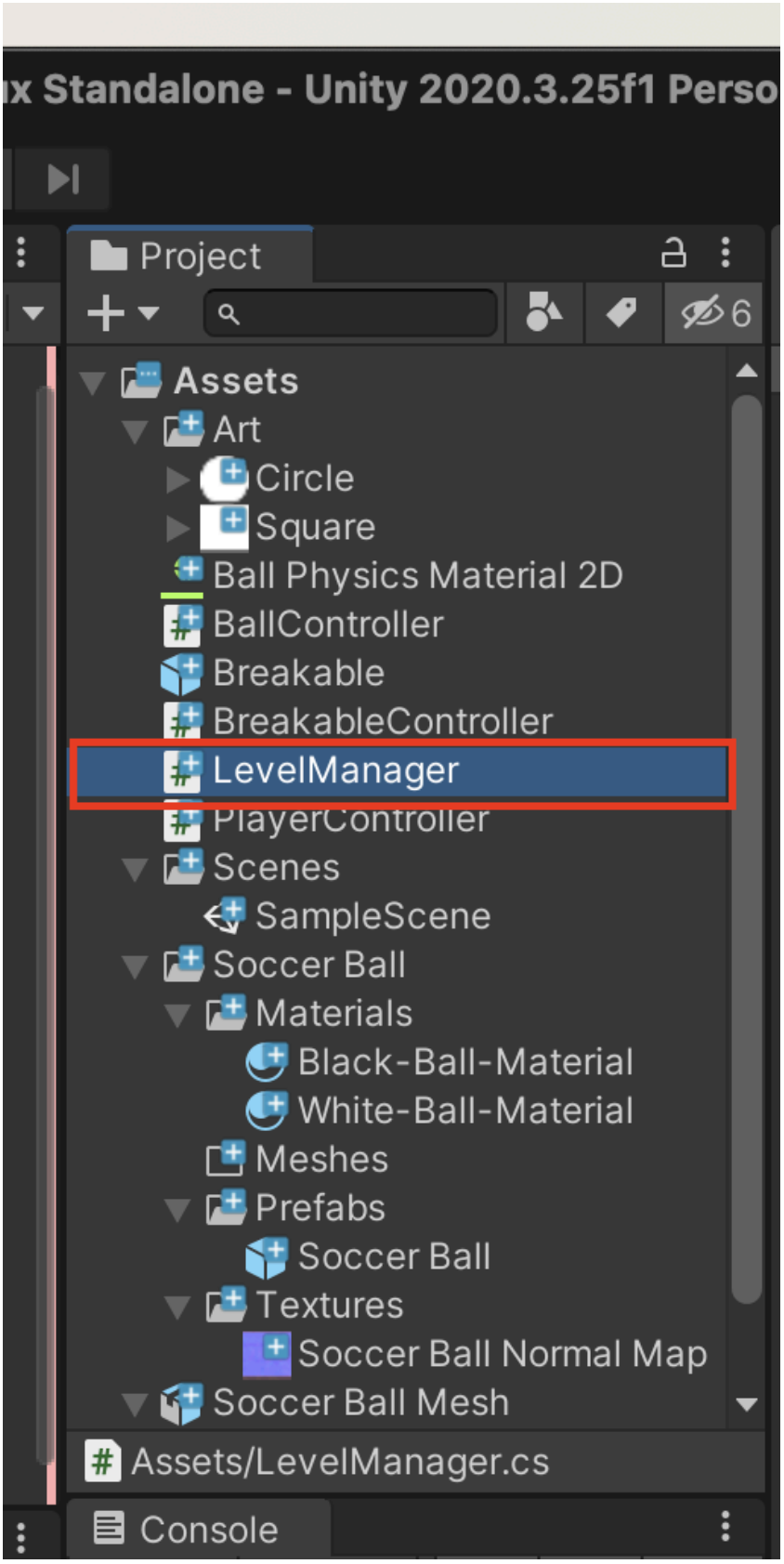


- **I ran the game and when the ball hit the breakable, the breakable broke and the ball bounced back.**

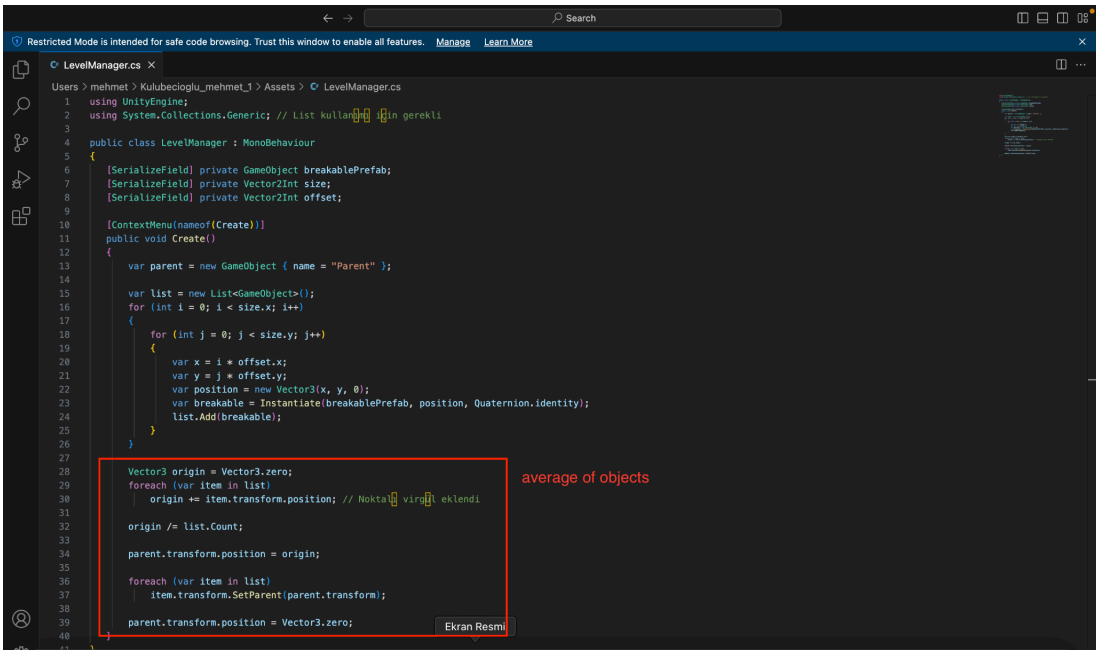
    - **opening new Breakable controller script:**



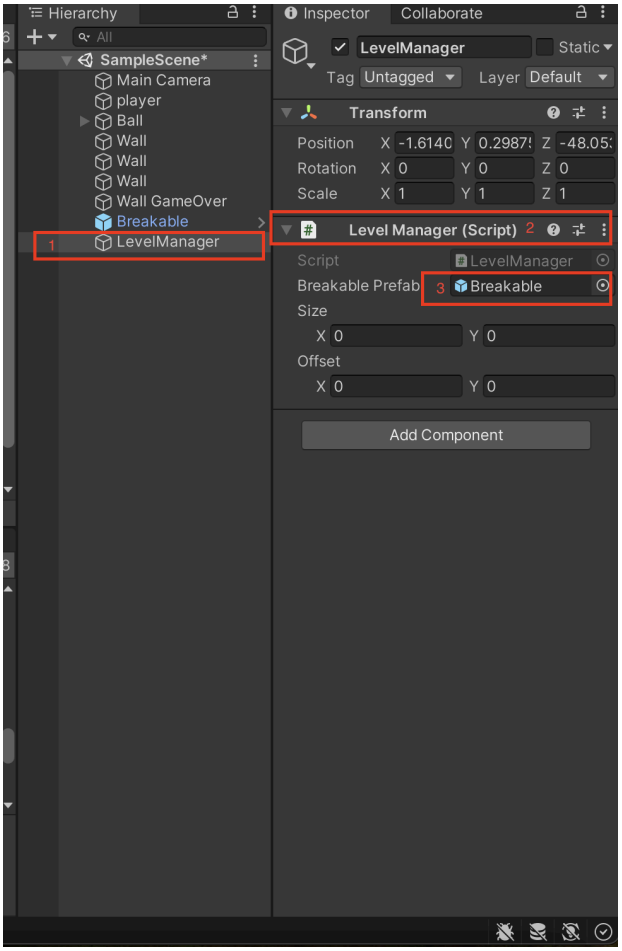
- **Opening new LevelManager controller script:**



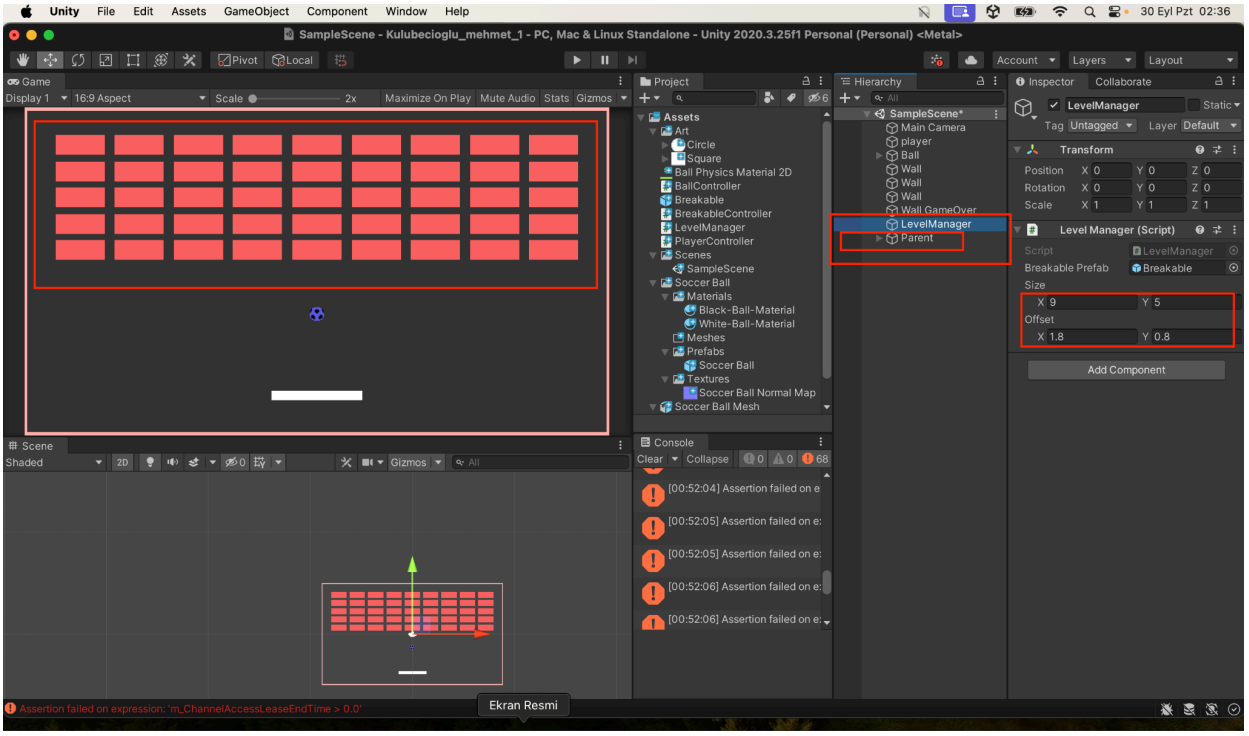
- **Arranging objects and centering objects LevelManager code content:**



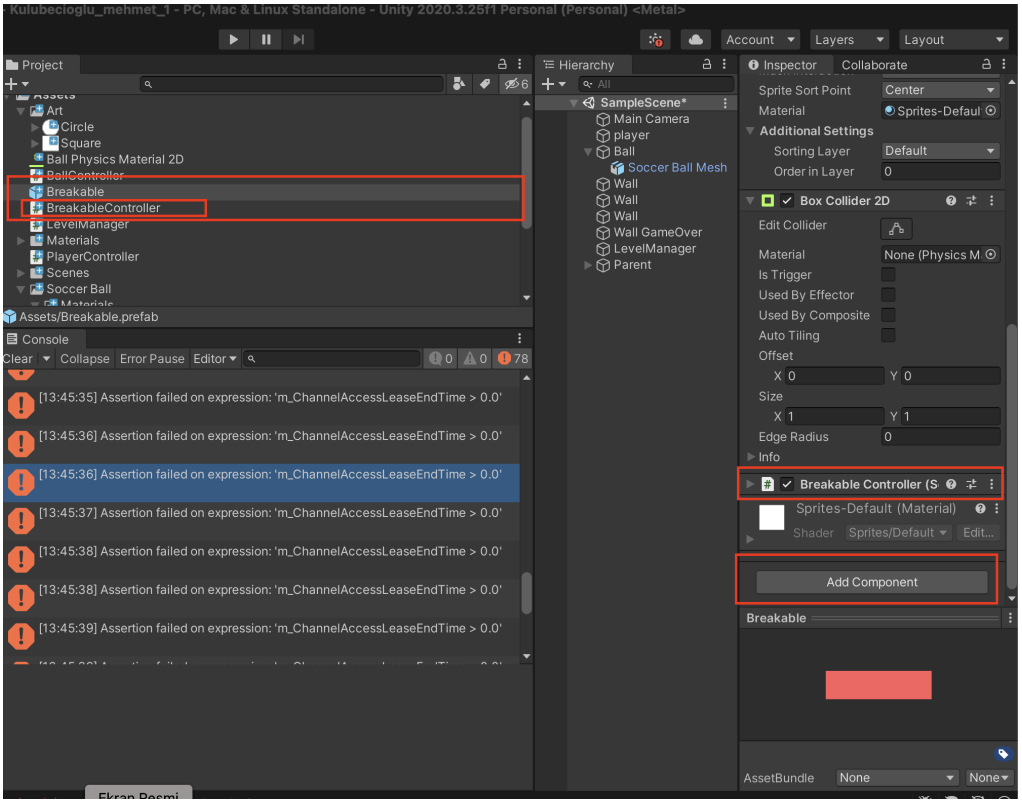
- **I created a new one, put the script in it and put the breakout in it:**



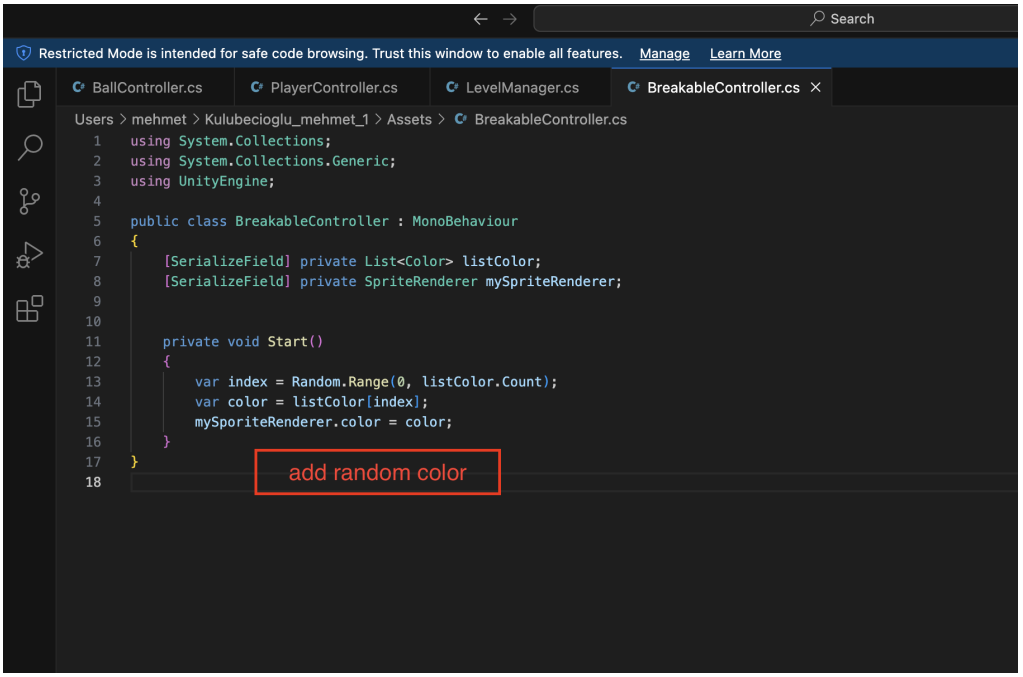
- **In order for the breakouts we wrote the code to be created automatically, I will click on the three dots next to the script and press the create button and the breakouts will be created automatically.**
- **Now let's check the breakouts created by the code**



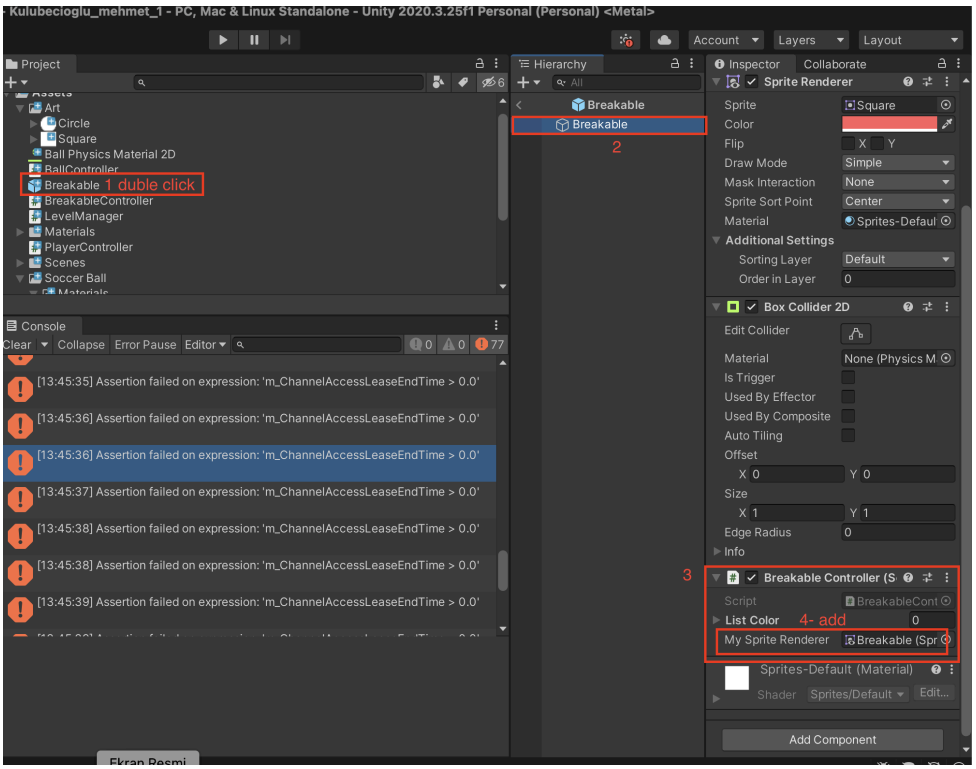
- **Adding BreakableController to Breakable:**



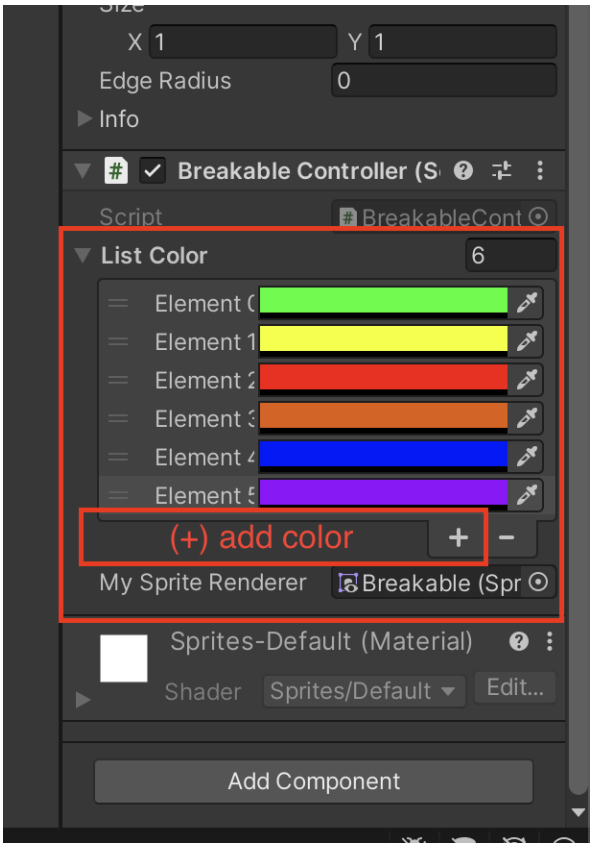
- **I wrote the necessary codes below to automatically generate random colors for the blocks**



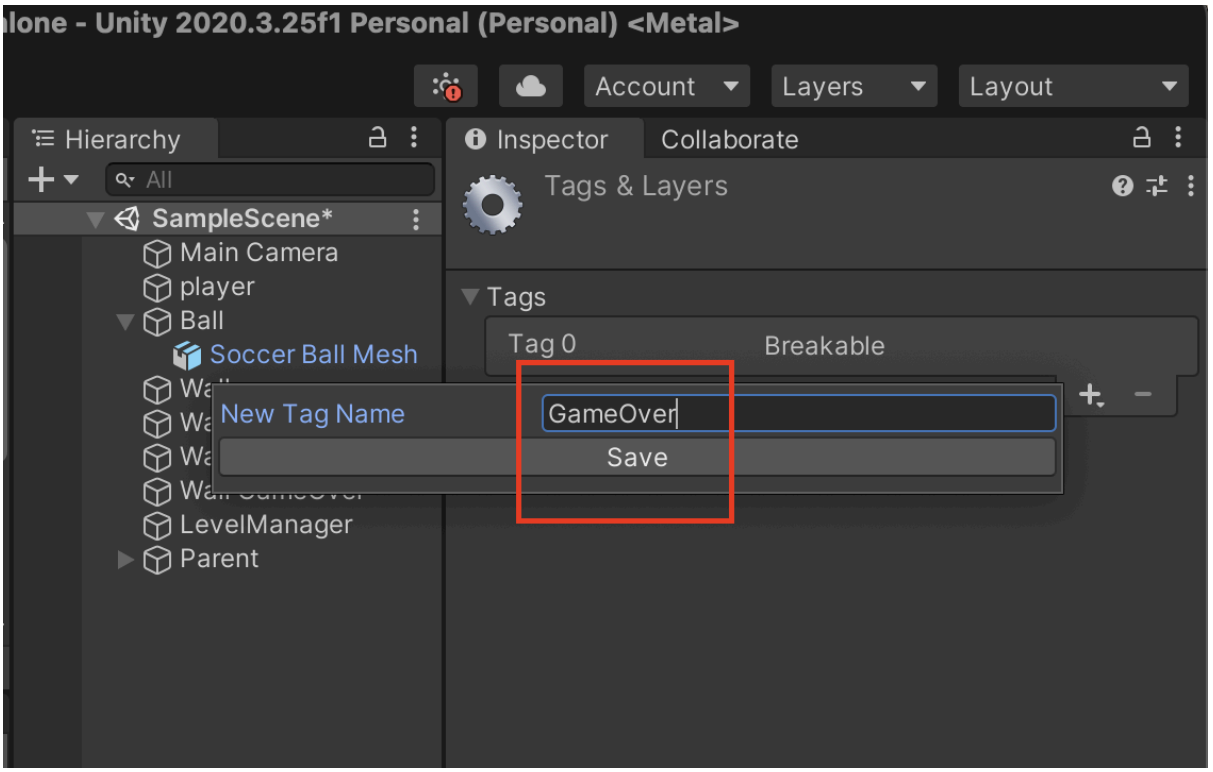
- **I added the breakable by dragging it to the list color.**



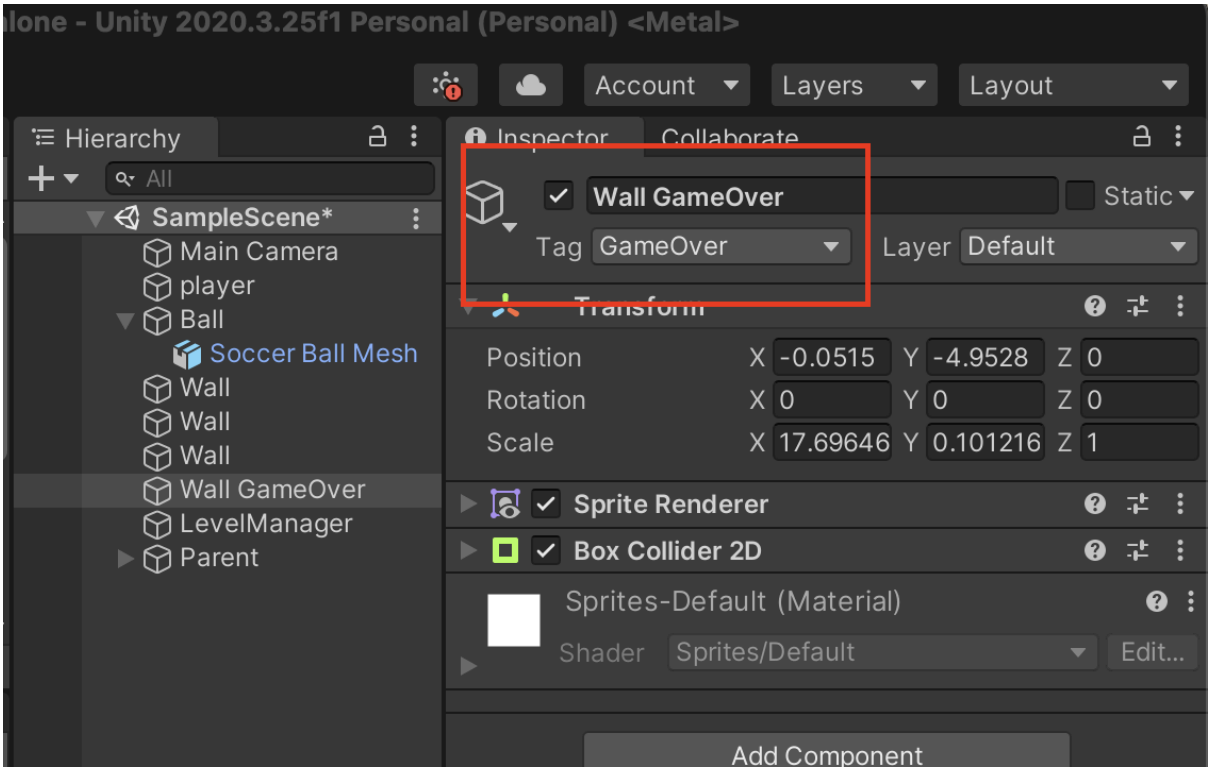
- **I add 5 different color options for the code to place breakables in random colors and save it:**



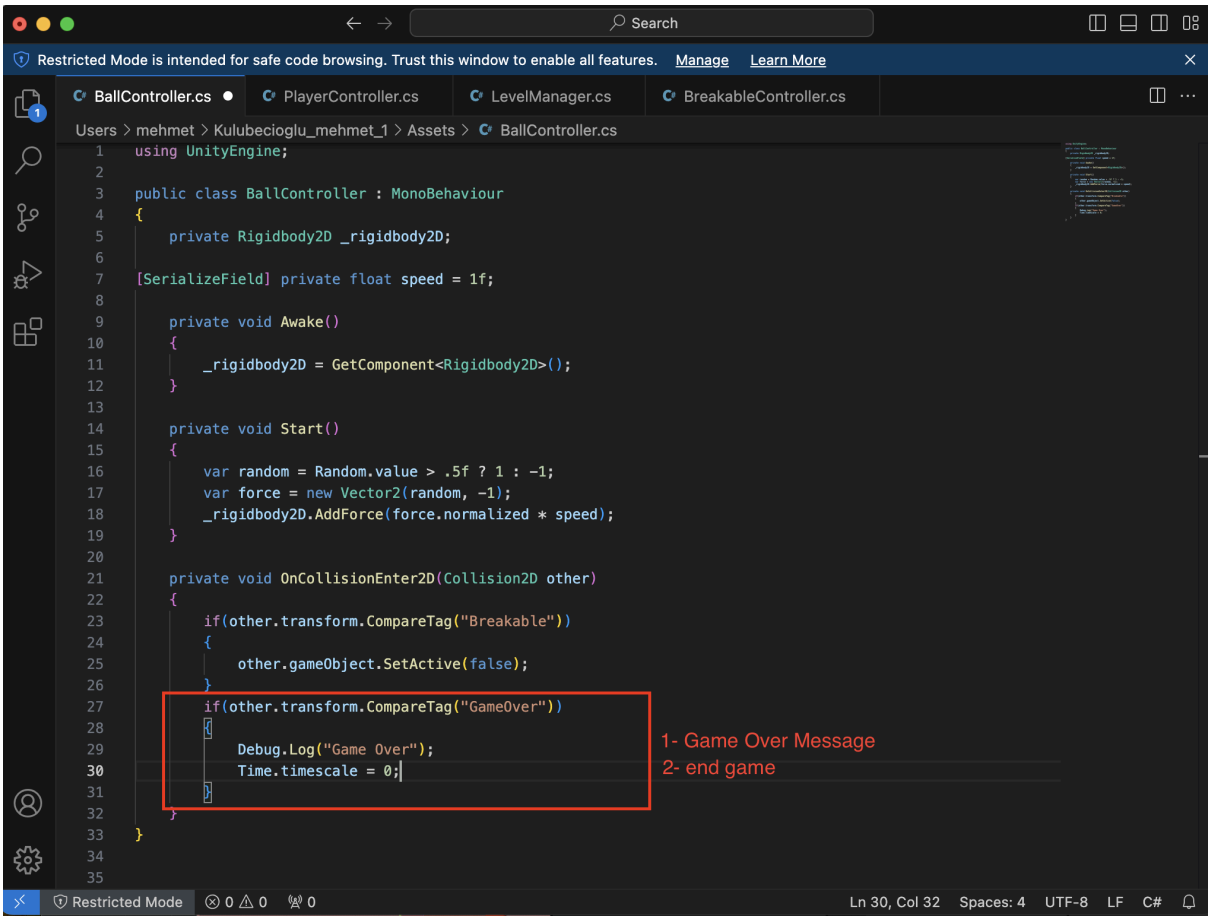
- **Adding a game over tag so that the game ends when it hits the wall below:**



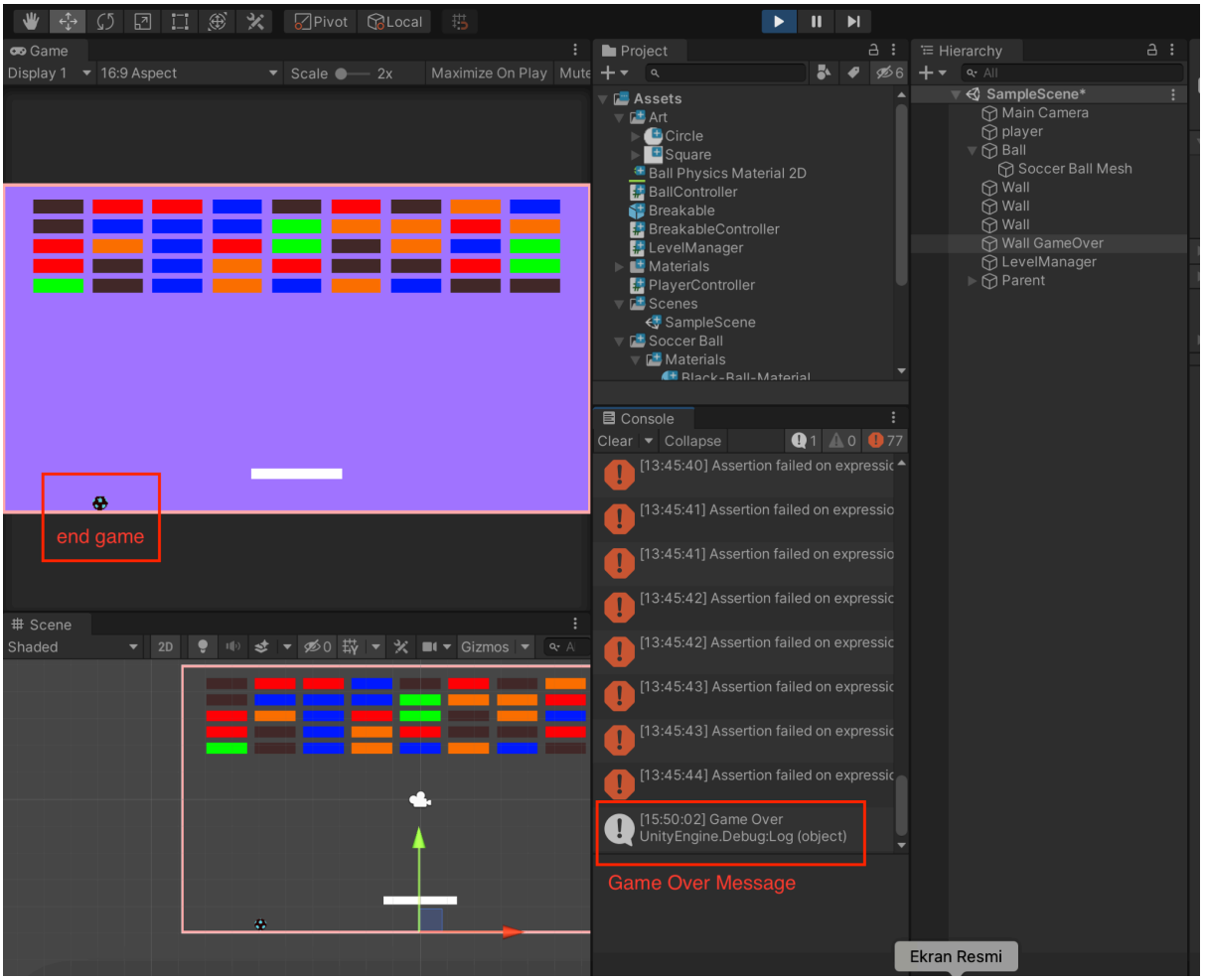
- **Enable gameover tag:**



- **Now, after this step, if the ball hits the wall below, the game ends, game over is printed on the console and the game stops. The necessary codes for this are:**

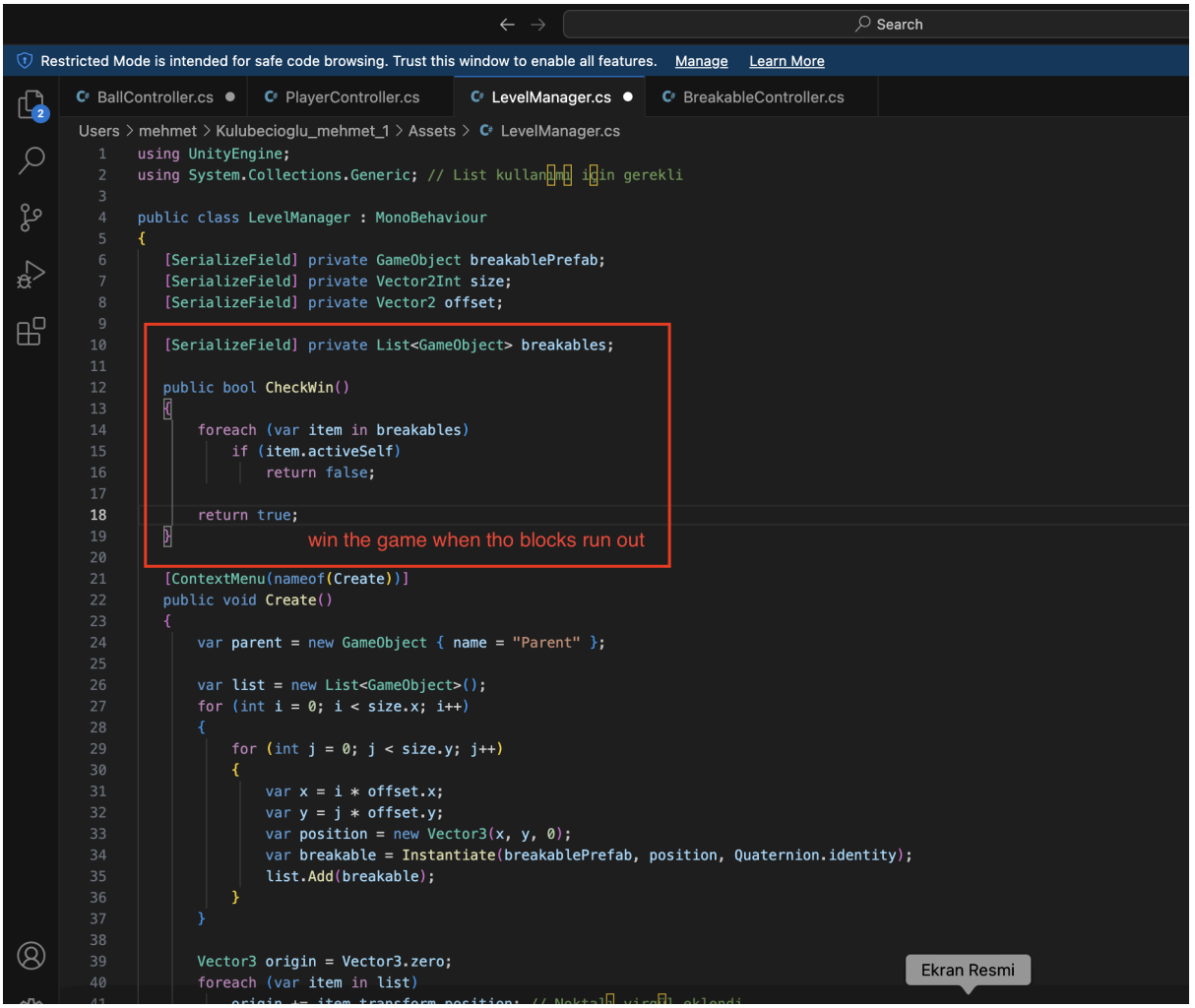


- **Now I will start the game and when the ball falls to the ground, I will print out the game over and test to see if the game ends:**



My code worked as I wanted, the game stopped when the ball touched the ground and the game over output was printed.

- **I added the LevelLanager code to win the game when the blocks are finished:**




```
using UnityEngine;
using System.Collections.Generic; // List kullanımı için gerekli

public class LevelManager : MonoBehaviour
{
    [SerializeField] private GameObject breakablePrefab;
    [SerializeField] private Vector2Int size;
    [SerializeField] private Vector2 offset;

    [SerializeField] private List<GameObject> breakables;

    public bool CheckWin()
    {
        foreach (var item in breakables)
            if (item.activeSelf)
                return false;

        return true;
    }
    [ContextMenu(nameof(Create))]
    public void Create()
    {
        var parent = new GameObject { name = "Parent" };

        var list = new List<GameObject>();
        for (int i = 0; i < size.x; i++)
        {
            for (int j = 0; j < size.y; j++)
            {
                var x = i * offset.x;
                var y = j * offset.y;
                var position = new Vector3(x, y, 0);
                var breakable = Instantiate(breakablePrefab, position, Quaternion.identity);
                list.Add(breakable);
            }
        }

        Vector3 origin = Vector3.zero;
        foreach (var item in list)
```

- **(BallController) codes I added to stop the game when the blocks are finished and get "you win" output:**

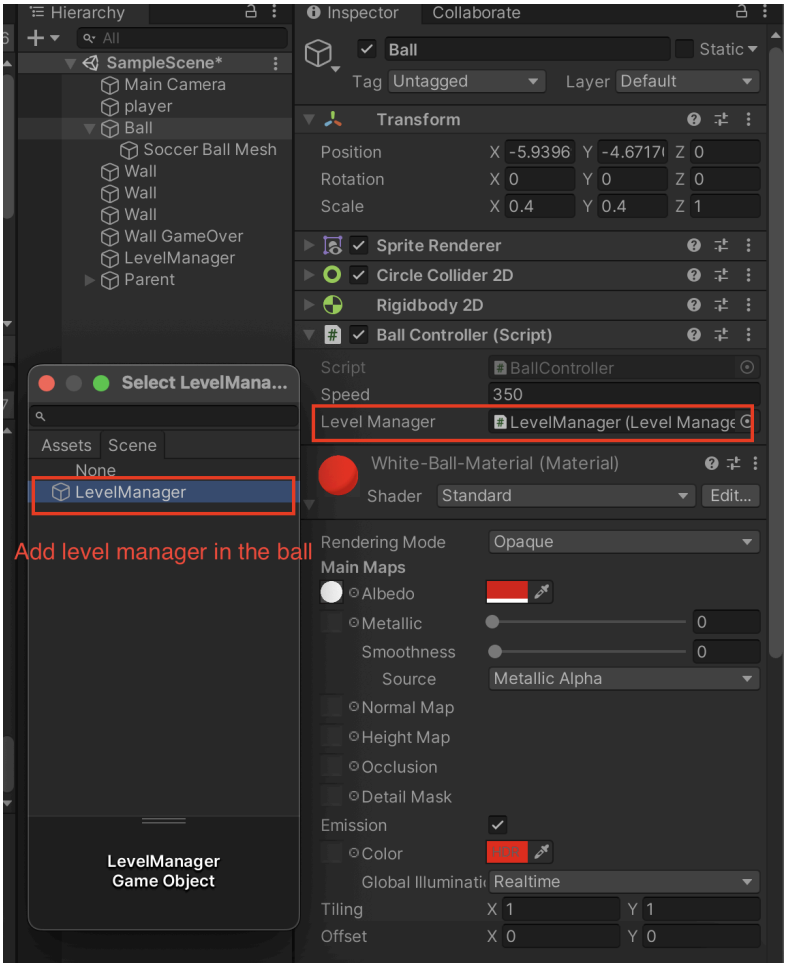
```
using UnityEngine;

public class BallController : MonoBehaviour
{
    private Rigidbody2D _rigidbody2D;

    [SerializeField] private float speed = 1f;
    [SerializeField] private LevelManager levelManager;

    private void Awake()
    {
        _rigidbody2D = GetComponent<Rigidbody2D>();
    }

    private void Start()
    {
        var random = Random.value > .5f ? 1 : -1;
        var force = new Vector2(random, -1);
        _rigidbody2D.AddForce(force.normalized * speed);
    }

    private void OnCollisionEnter2D(Collision2D other)
    {
        if(other.transform.CompareTag("Breakable"))
        {
            other.gameObject.SetActive(false);
            if (levelManager.CheckWin())
            {
                Debug.Log("You Win!");
                Time.timeScale = 0;
            }
        }
        if(other.transform.CompareTag("GameOver"))
        {
            Debug.Log("Game Over");
            Time.timeScale = 0;
        }
    }
}
```

add level manager

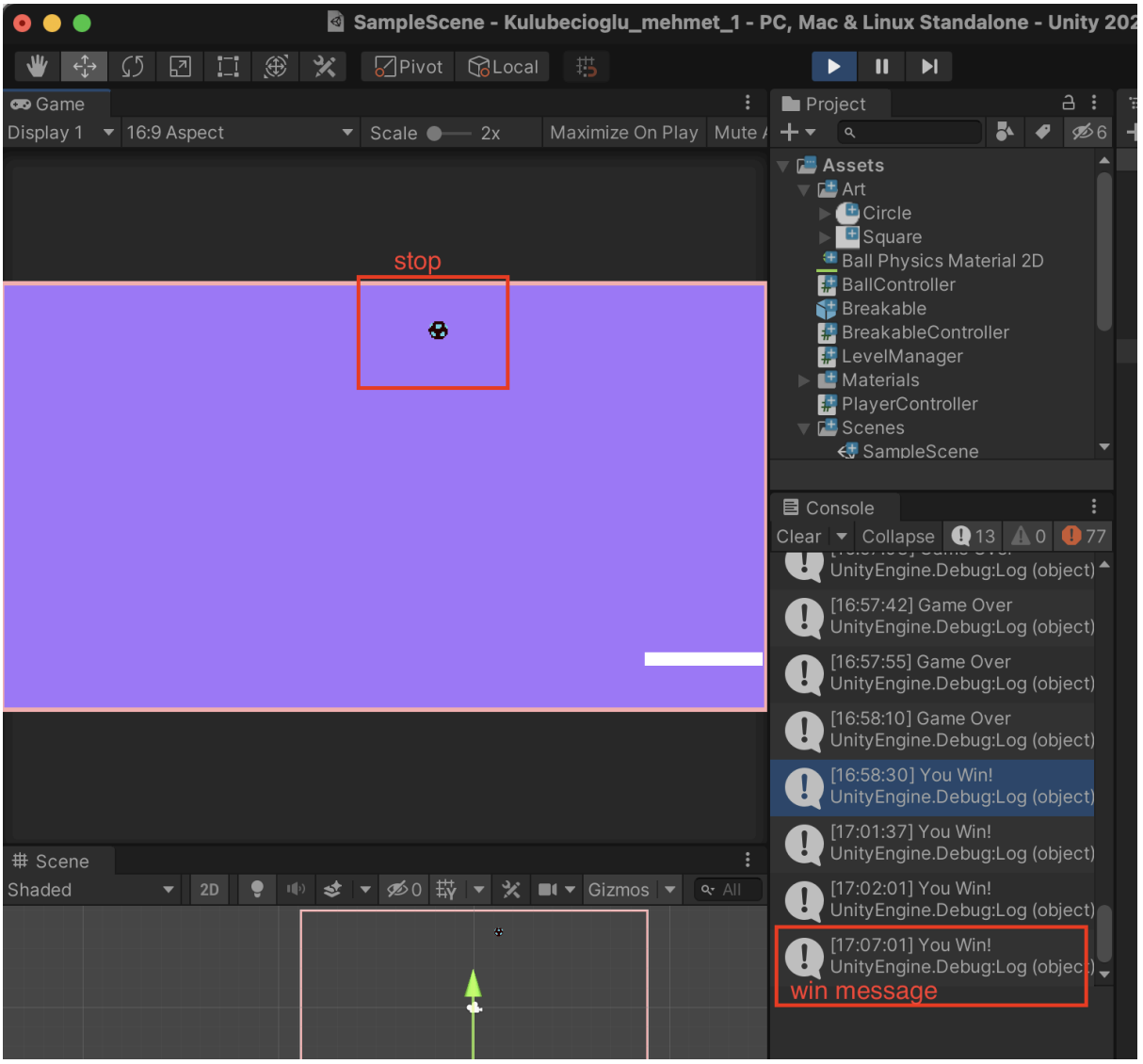
stoping time when blocks run out

Ekran Resmi

- **Placing the level manager inside the ball:**



- **Now, finally, I check whether the "you win" message will appear when the blocks are finished and the game will stop.**

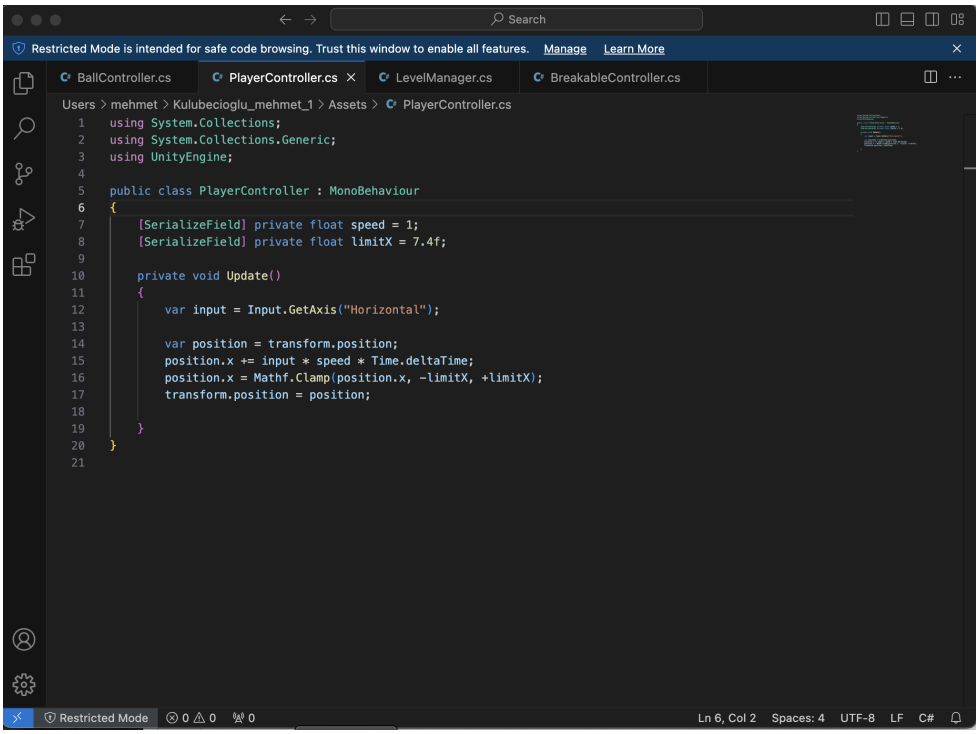


Collision:

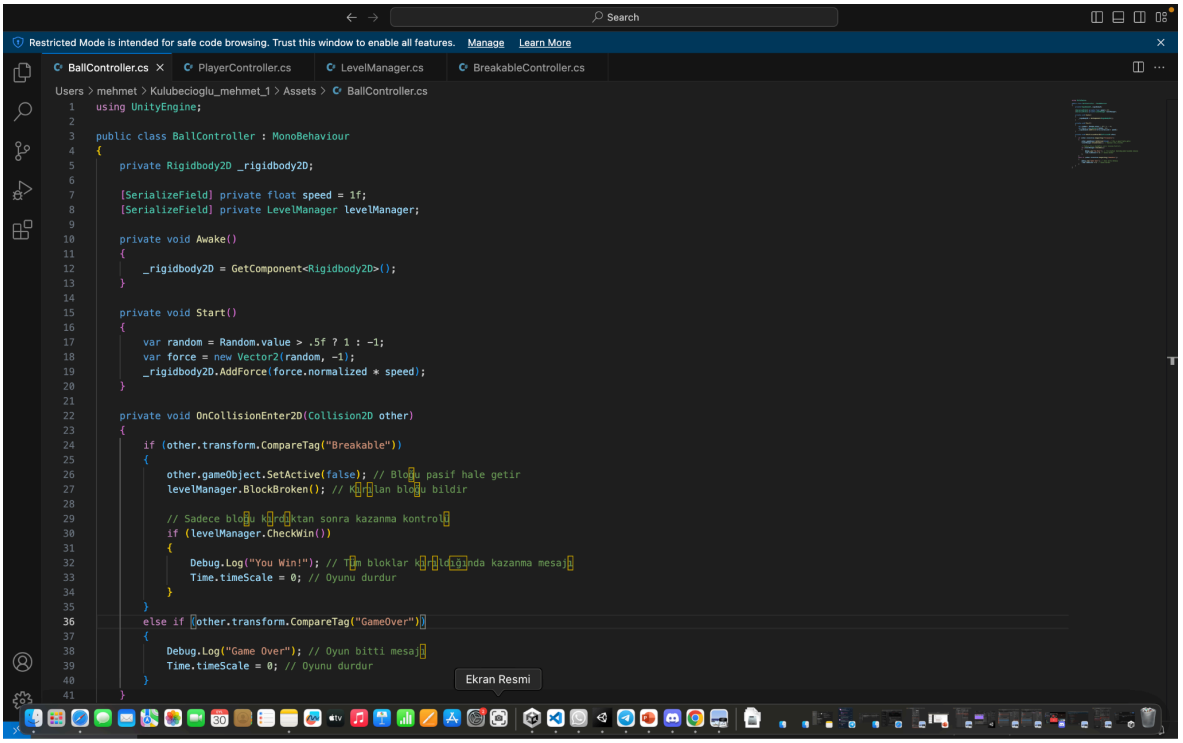
In this assignment, I learned how to develop games using Unity and how to download and use objects from the assets store. In addition, I learned to synchronize the scripts of the codes I wrote in Visual Studio Code with the objects in Unity.

Additionally, I am adding all the codes I wrote below.

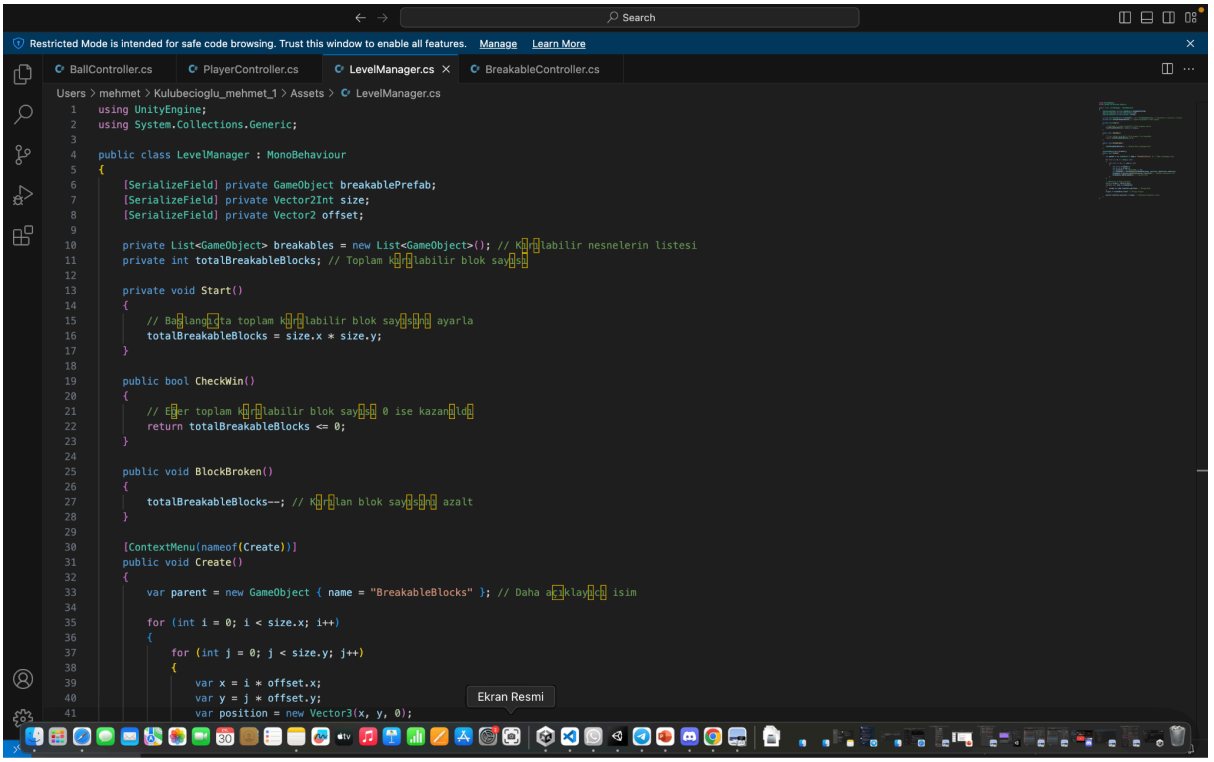
PlayerContrtoller:



BallController:



LevelManager:




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

public class LevelManager : MonoBehaviour
{
    [SerializeField] private GameObject breakablePrefab;
    [SerializeField] private Vector2Int size;
    [SerializeField] private Vector2 offset;

    private List<GameObject> breakables = new List<GameObject>(); // Kırılabilir nesnelerin listesi
    private int totalBreakableBlocks; // Toplam kırılabilir blok sayısı

    private void Start()
    {
        // Başlangıçta toplam kırılabilir blok sayısını ayarla
        totalBreakableBlocks = size.x * size.y;
    }

    public bool CheckWin()
    {
        // Eğer toplam kırılabilir blok sayısı 0 ise kazanıldı
        return totalBreakableBlocks <= 0;
    }

    public void BlockBroken()
    {
        totalBreakableBlocks--; // Kırılan blok sayısını azalt
    }

    [ContextMenu(nameof(Create))]
    public void Create()
    {
        var parent = new GameObject { name = "BreakableBlocks" }; // Daha açıklayıcı isim

        for (int i = 0; i < size.x; i++)
        {
            for (int j = 0; j < size.y; j++)
            {
                var x = i * offset.x;
                var y = j * offset.y;
                var position = new Vector3(x, y, 0);
                var breakable = Instantiate(breakablePrefab, position, Quaternion.identity);
                breakable.transform.SetParent(parent.transform); // Nesneyi ebeveynine ata
                breakables.Add(breakable); // Listeye ekle
            }
        }

        // Ebeveynin ortasını hesapla
        Vector3 origin = Vector3.zero;
        foreach (var item in breakables)
        {
            origin += item.transform.position; // Ortaya ekle
        }
        origin /= breakables.Count; // Ortayı hesapla

        parent.transform.position = origin; // Ebeveynin konumunu ortala
    }
}
```

BreakableController: