

**MINISTRY OF EDUCATION 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**

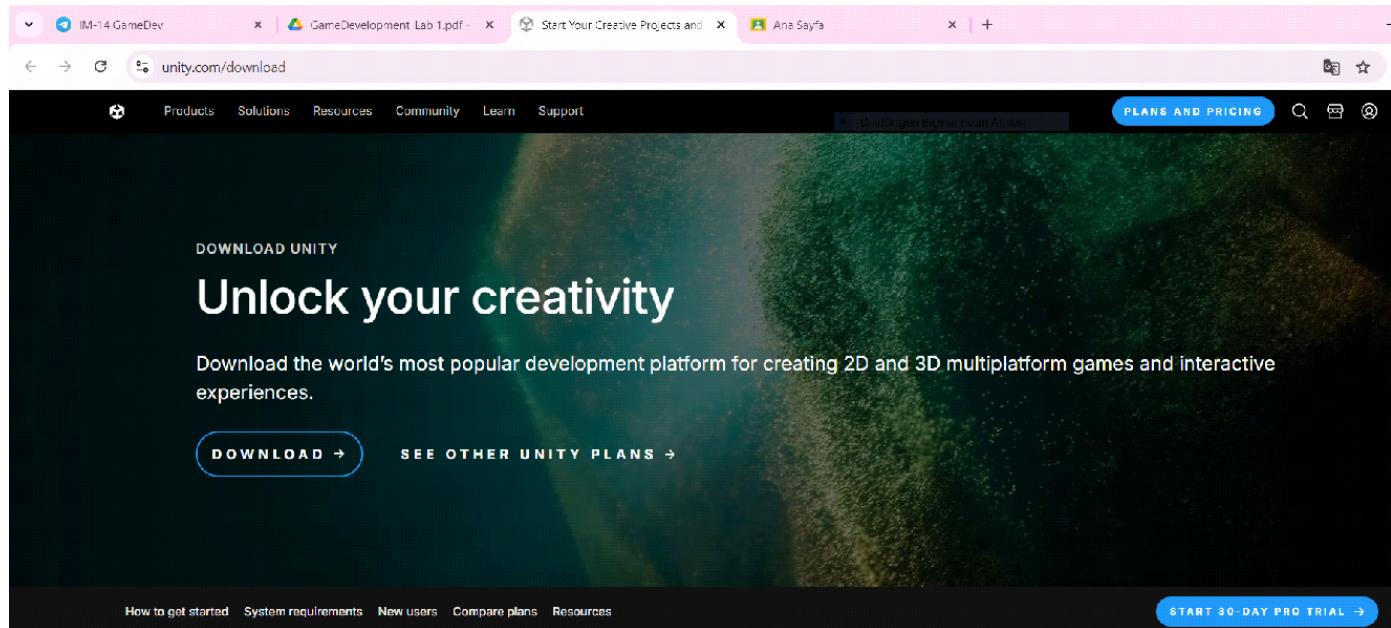
## 1. Tasks

- 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.



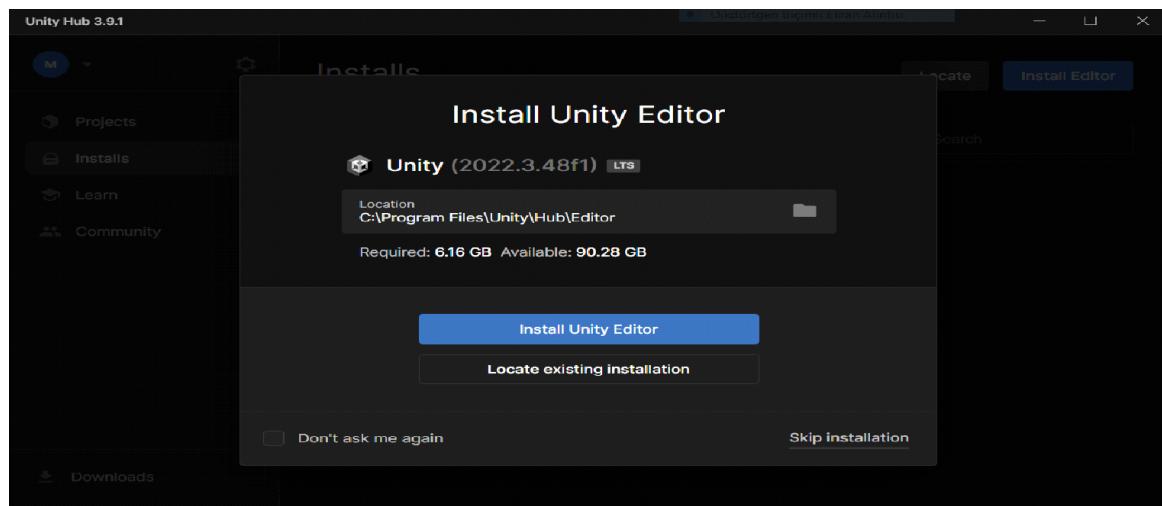
The screenshot shows the Unity website's 'Create with Unity in three steps' guide. It includes three sections: 1. Download the Unity Hub, 2. Choose your Unity version, and 3. Start your project. Each section has a brief description and a link to further information.

**1. Download the Unity Hub**  
Follow the instructions onscreen for guidance through the installation process and setup.  
[Download for Windows](#) [Download for Mac](#) [Instructions for Linux](#)

**2. Choose your Unity version**  
Install the latest version of Unity, an older release, or a beta featuring the latest in-development features.  
[Visit the download archive](#)

**3. Start your project**  
Begin creating from scratch, or pick a template to get your first project up and running quickly. Access tutorial videos designed to support creators, from beginners to experts.  
[Access our Pro Onboarding Guide](#)

- 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

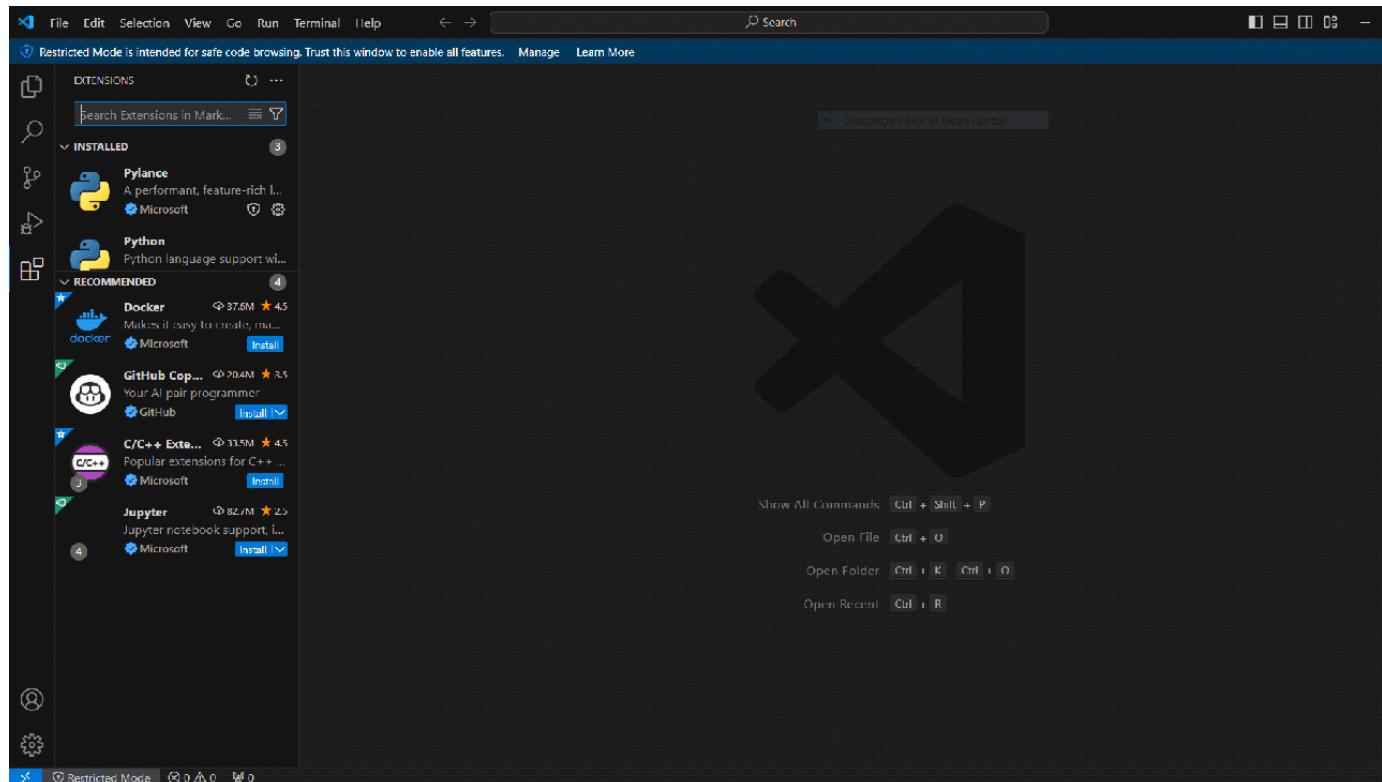


- Select Install Unity Editor to download and install the latest LTS version of the Unity Editor (for instance, I have installed Unity 2022.3.44f1). The download and installation of Unity Editor take some time. When the process is complete, this version of the Unity Editor will appear on the

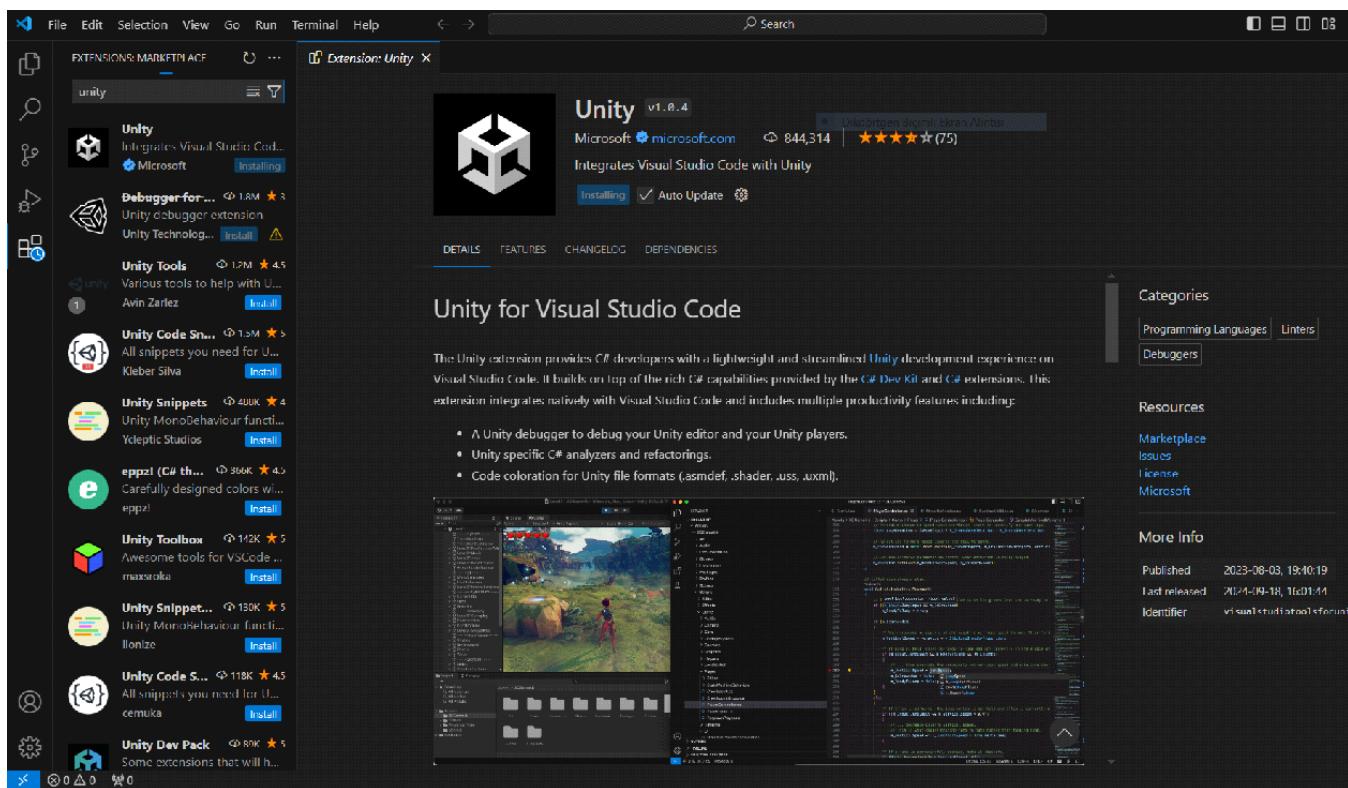
Installs tab of the Unity Hub.

## Install Visual Studio Code:

I have already vs code

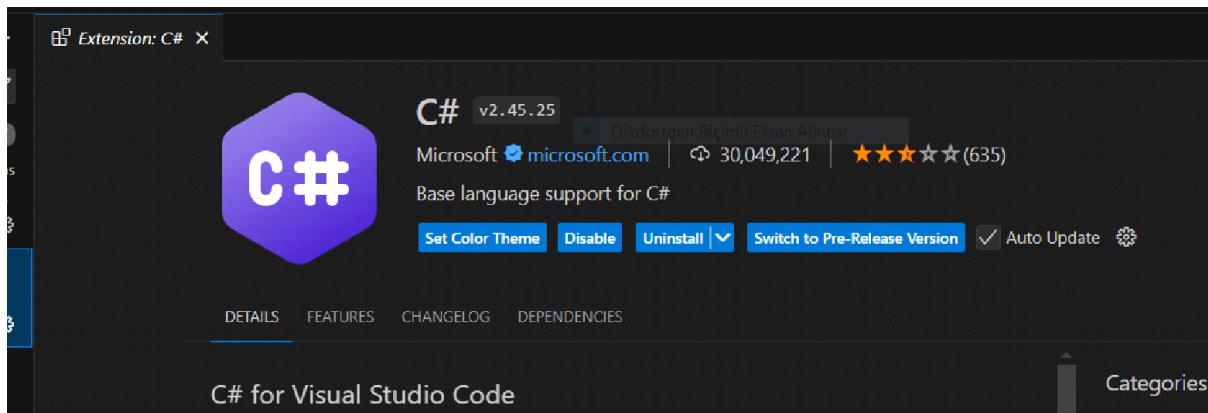


- Launch Visual Studio Code.
- In VS Code, open the Extensions view by clicking on the Extensions icon in the Activity Bar on the side of VS Code:



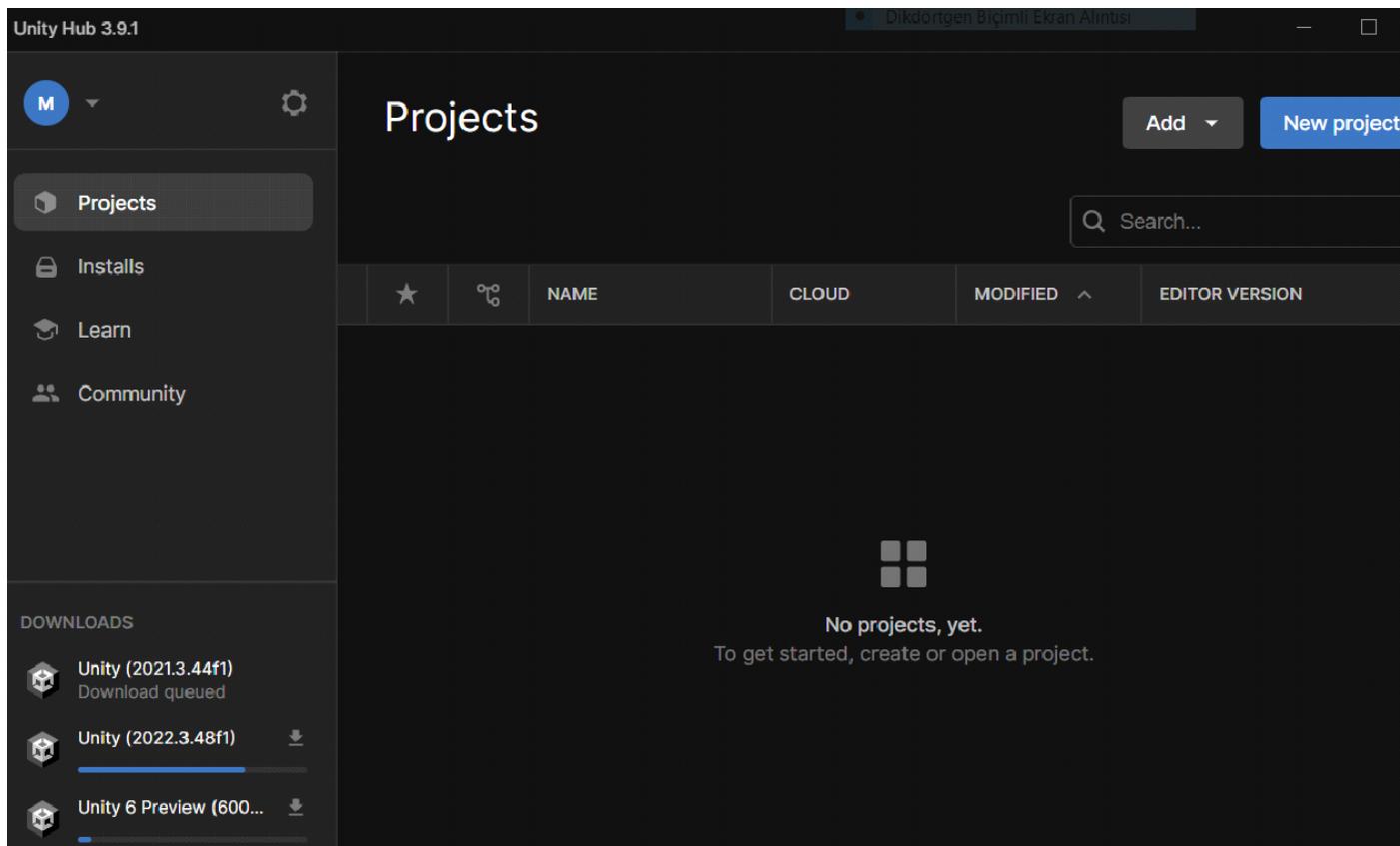
## Need to install next extensions:

- Unity
- C#

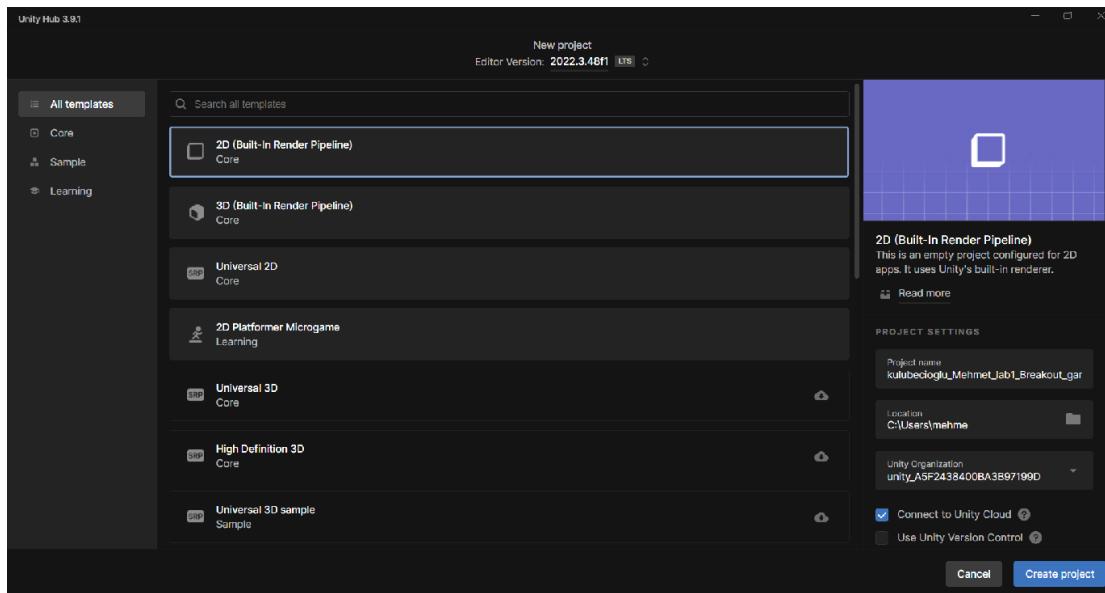


## Create new Unity project

- 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)

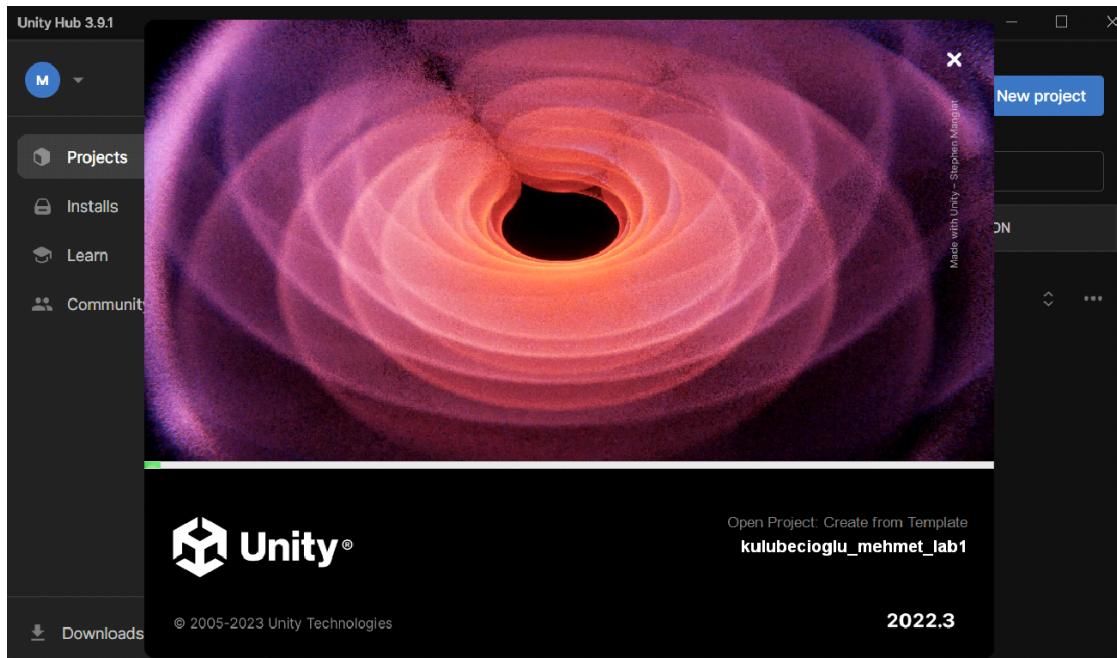


- Enter the name of your project (without space) – block 3
- Select a location for your project – block 4

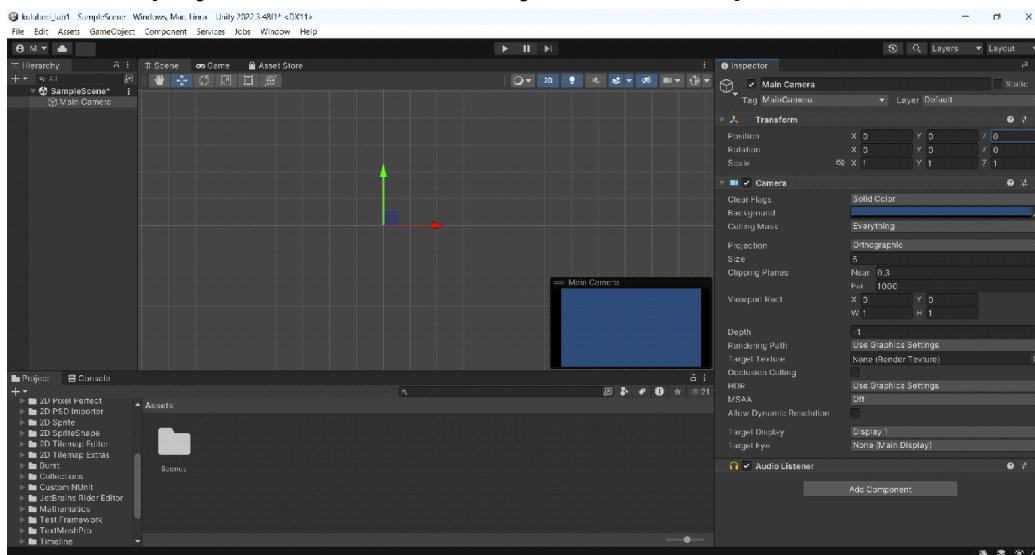
- Press button “Create project”.

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:**



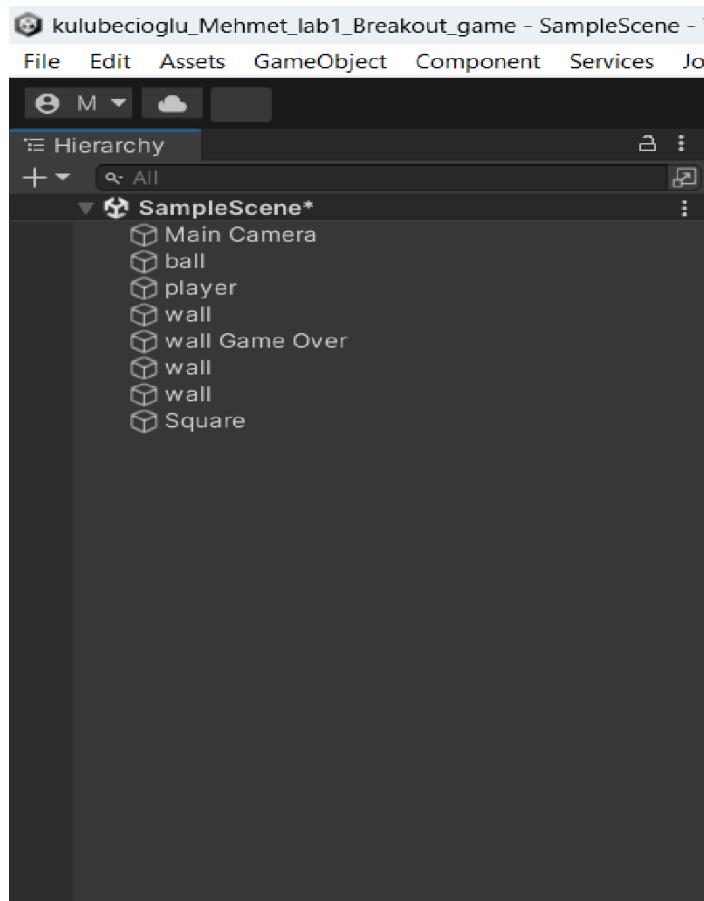
The project was created successfully.

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.

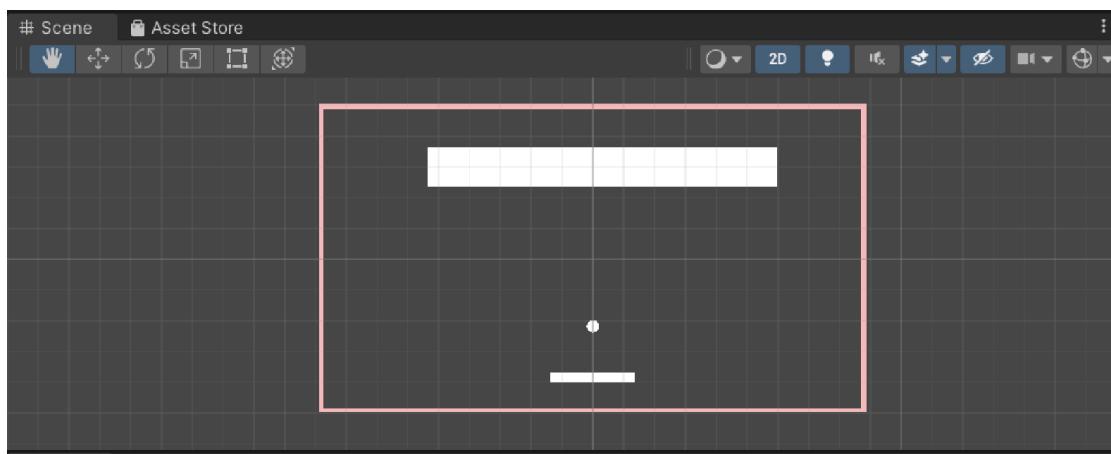
2. **Scene/Game View Window:** Here, I can edit my scenes and levels and test them in real-time.
3. **Project Window and Console Tab:**
  - **Project Window:** This section contains all the assets related to my project. 3D models, 2D textures, sound and music files, and anything else required for my 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.



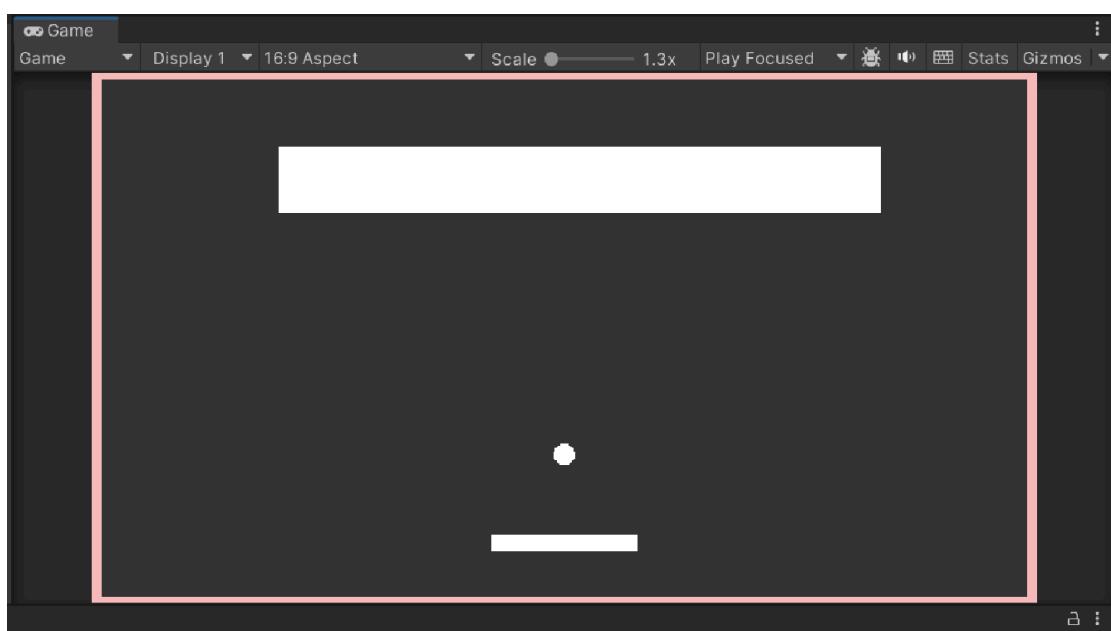
## Scene View

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.



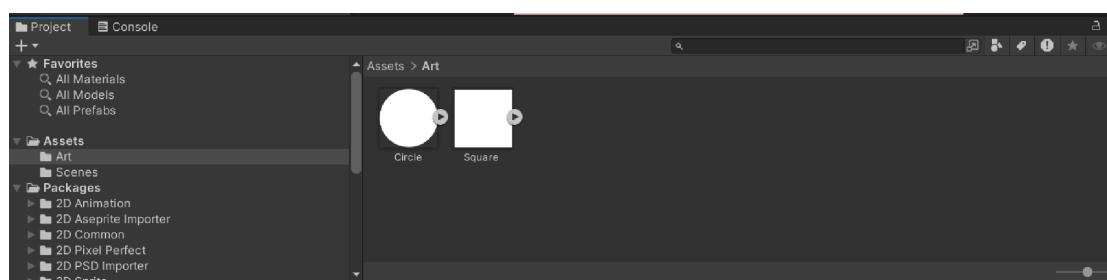
## Game View

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

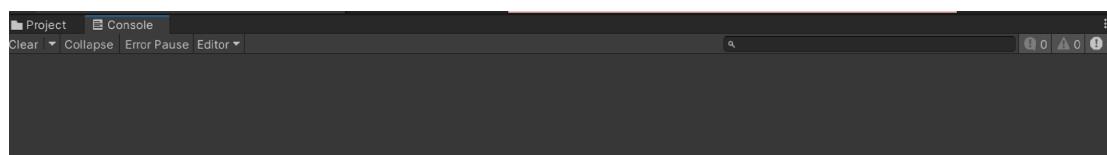
The Project Window gives me an overview of all the content in the Assets folder. It's helpful for me to create folders in the Project Window to organize items such as audio files, materials, models, textures, scenes, and scripts. Throughout the lifetime of my project, I'll spend a lot of time dragging 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.



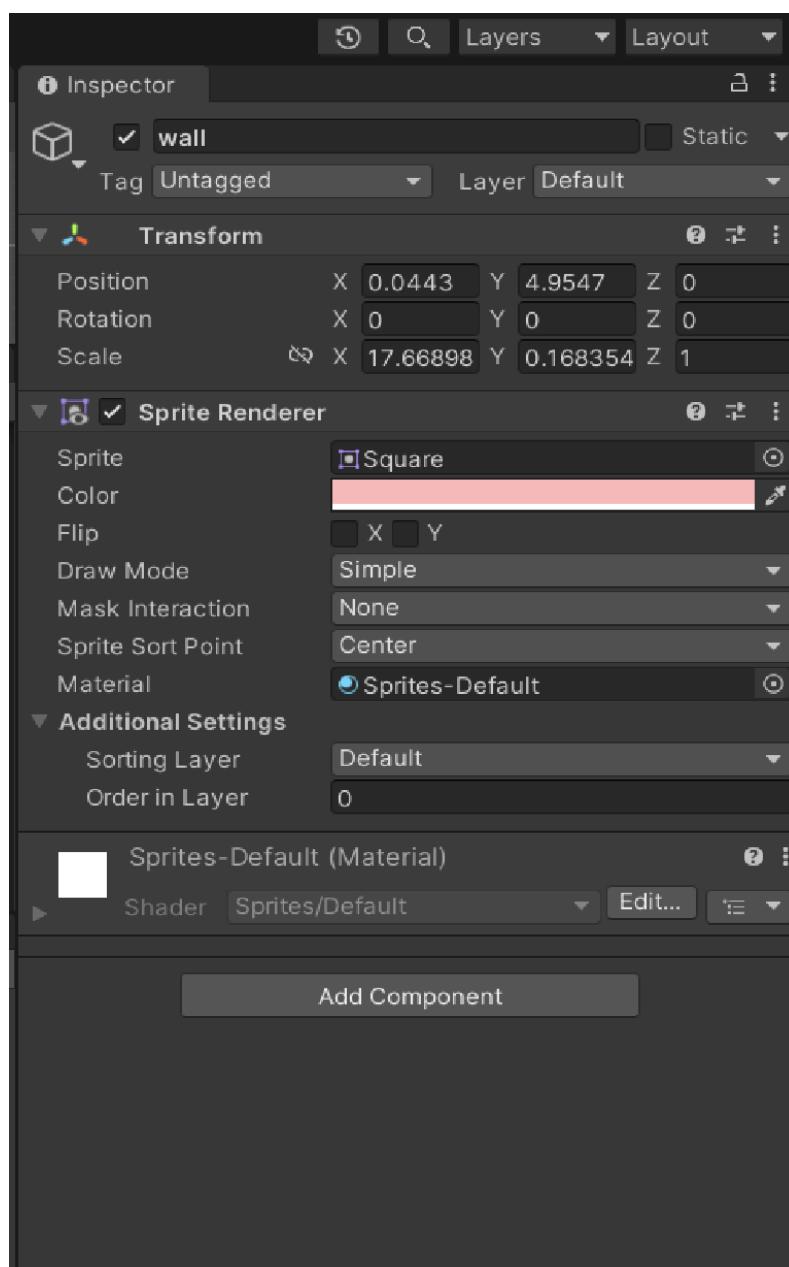
Sometimes you'll get an error message that occurs with every Unity frame update, and those messages will clog up your Console View in a hurry. In situations like this, it's helpful to hit the Collapse toggle button to collapse all identical error messages into a single message.



## 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.

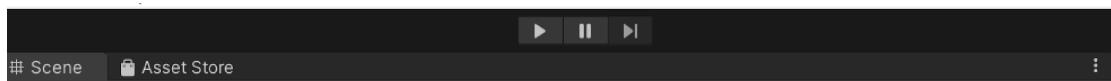


The six Transform tools, from left to right, are:

- **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.



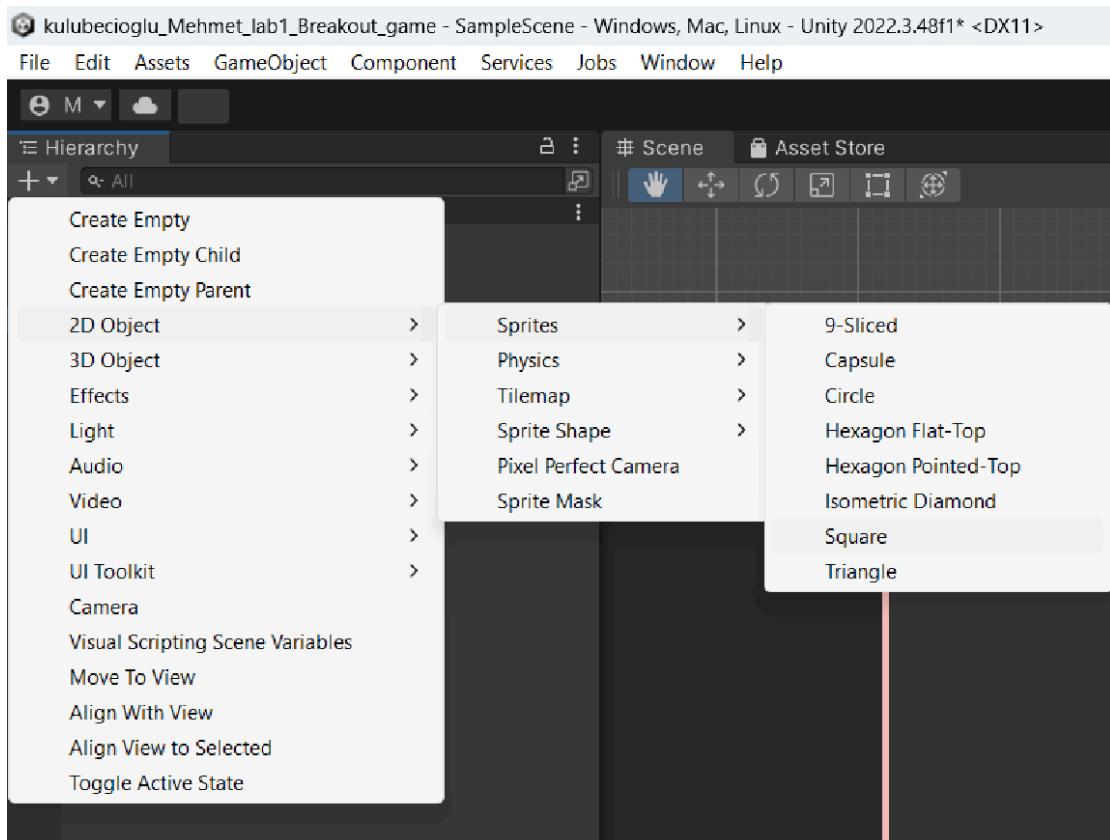
## Create the First Scene

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...

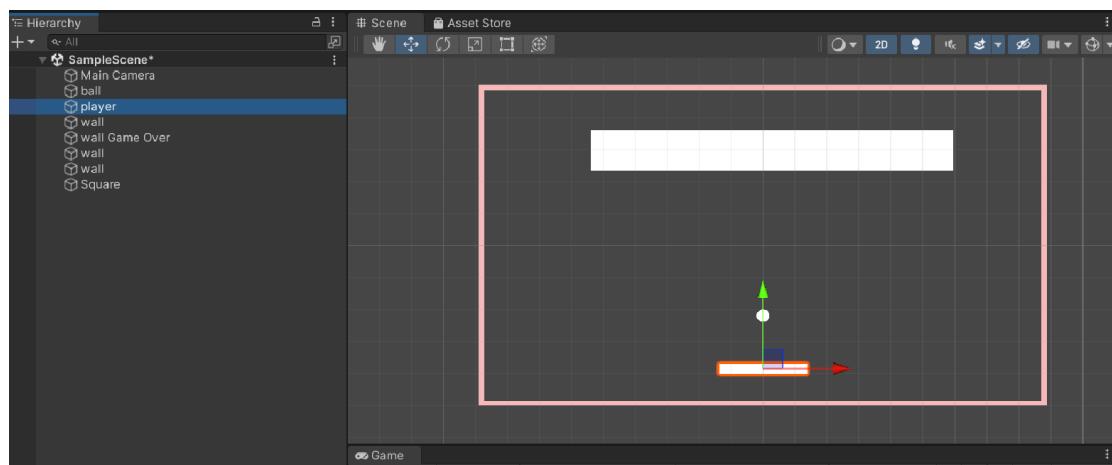
GameObjects serve as a sort of container, composed of many pieces of individually implemented functionalities. As I discussed in Chapter 2, GameObjects can even contain other GameObjects in parent–child relationships.

## 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."



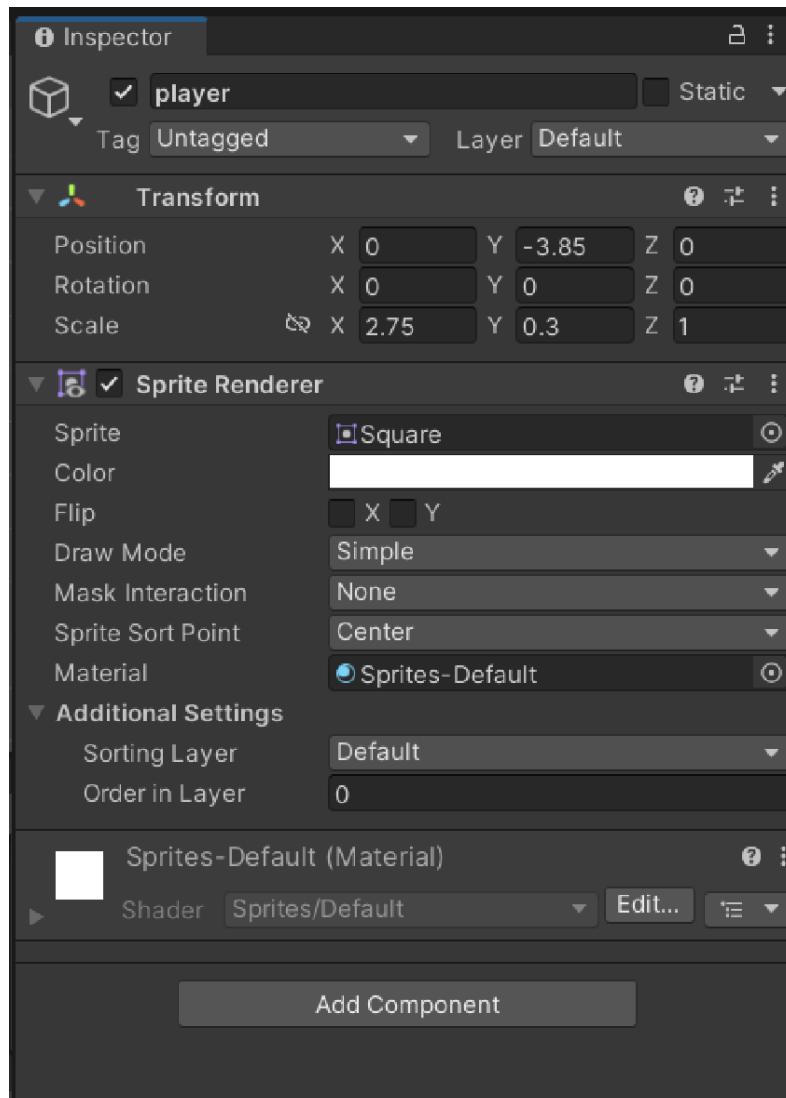
Selecting Sphere from this menu will place a 2D object “square” into the scene directly in the centre of the viewport.



## Inspector Panel

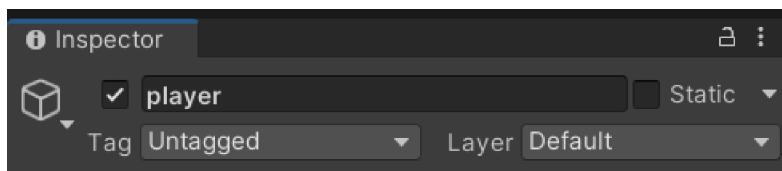
Select our new square object to see all its information listed in the

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.

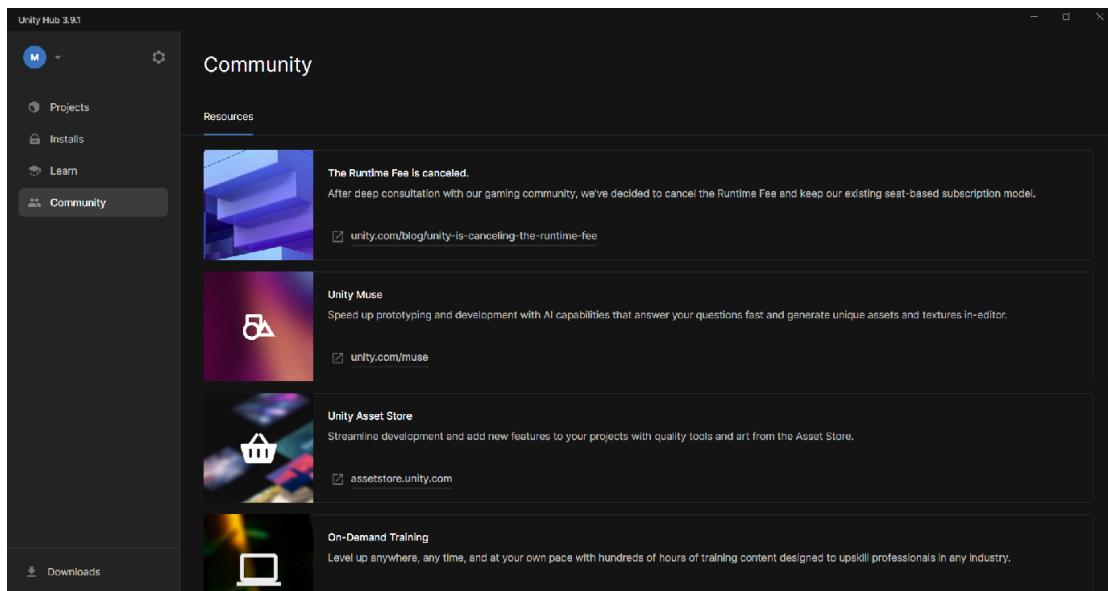


Following this top section, I see the Transform panel. This is an important section that shows where my object is in 3D space, as well as its rotation and scale information.

### 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:**



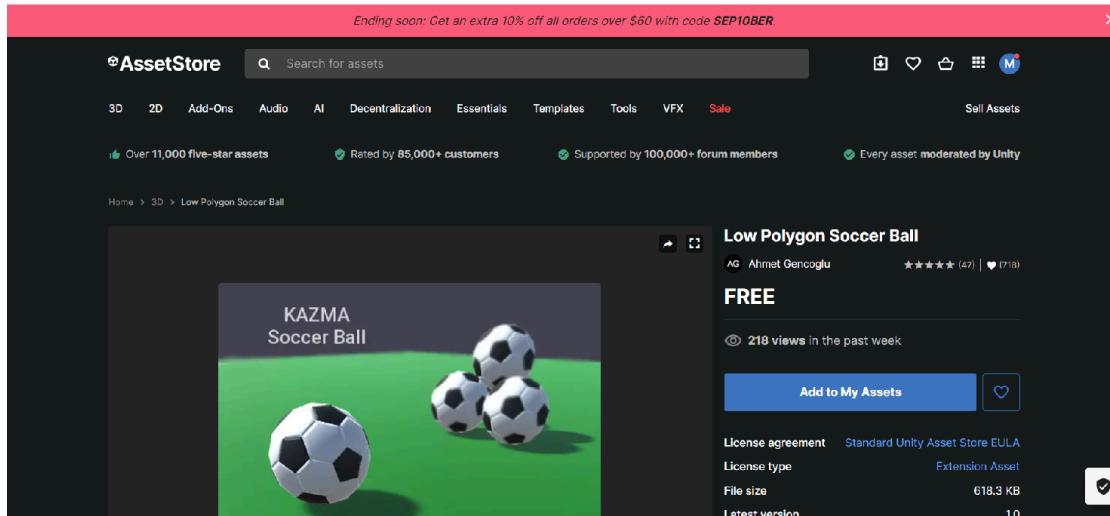
### "ball" in All Categories

CATEGORIES	Filters	Sort by	Results
3D (25)	Price (0)   Rating   Unity Version	Relevance	24
Templates (20)	Price: Free X   Clear		
Tools (12)			
2D (8)			
VFX (4)			
Essentials (2)			

Results 1-24 of 71 for ball

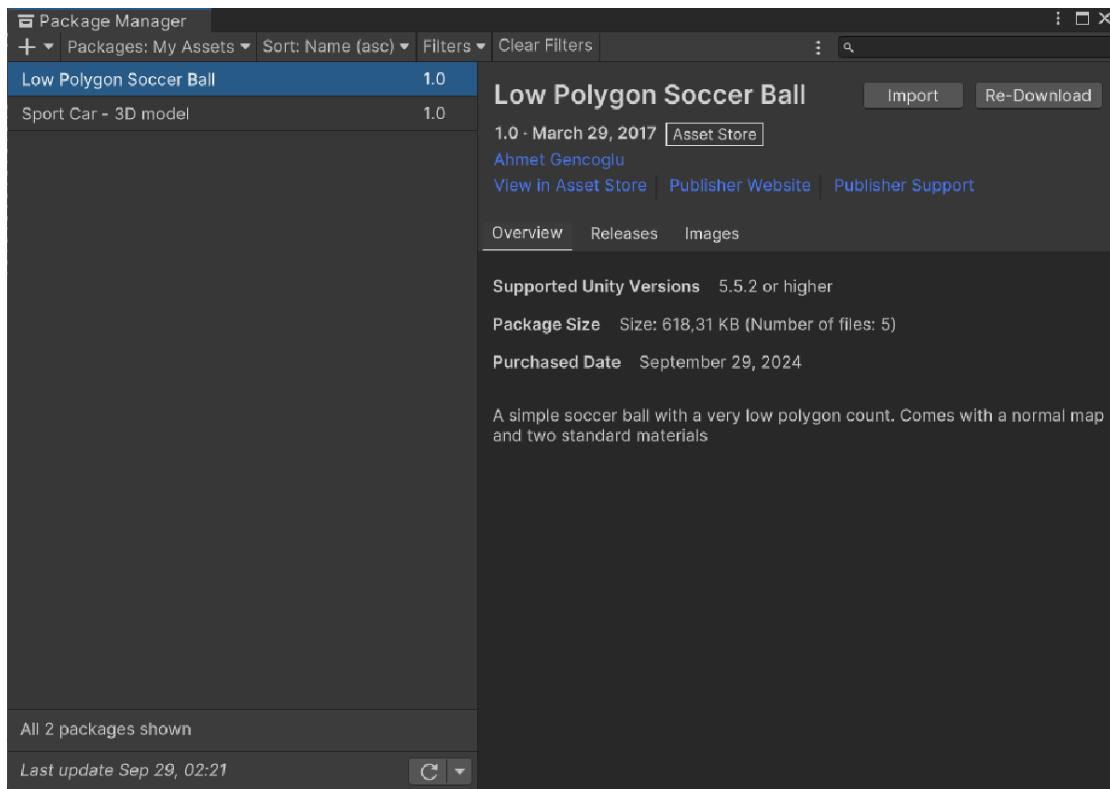
<b>Sport Balls</b> Free Sport Balls SARITASA Free	<b>Low Polygon Soccer Ball</b> ★★★★★ (47) AHMET GENCOGLU Free	<b>Ball Pack</b> ★★★★★ (156) YOUNGEN TECH Free	<b>Bunny Ball Assets   Legacy</b> (Latest release in Unity...) ★★★★★ (8) UNITY EDUCATION Free
--	--	---	---

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

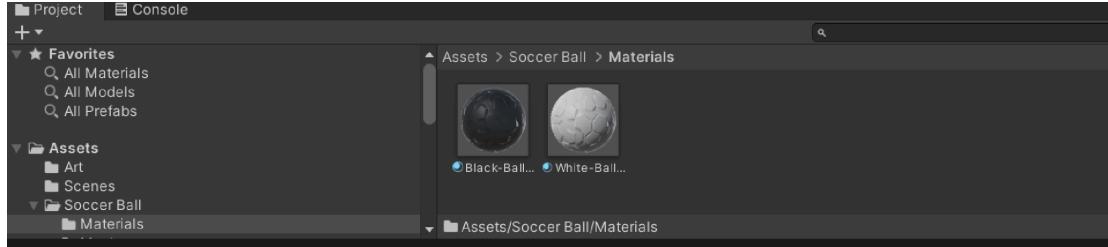
### “Package Manager”:



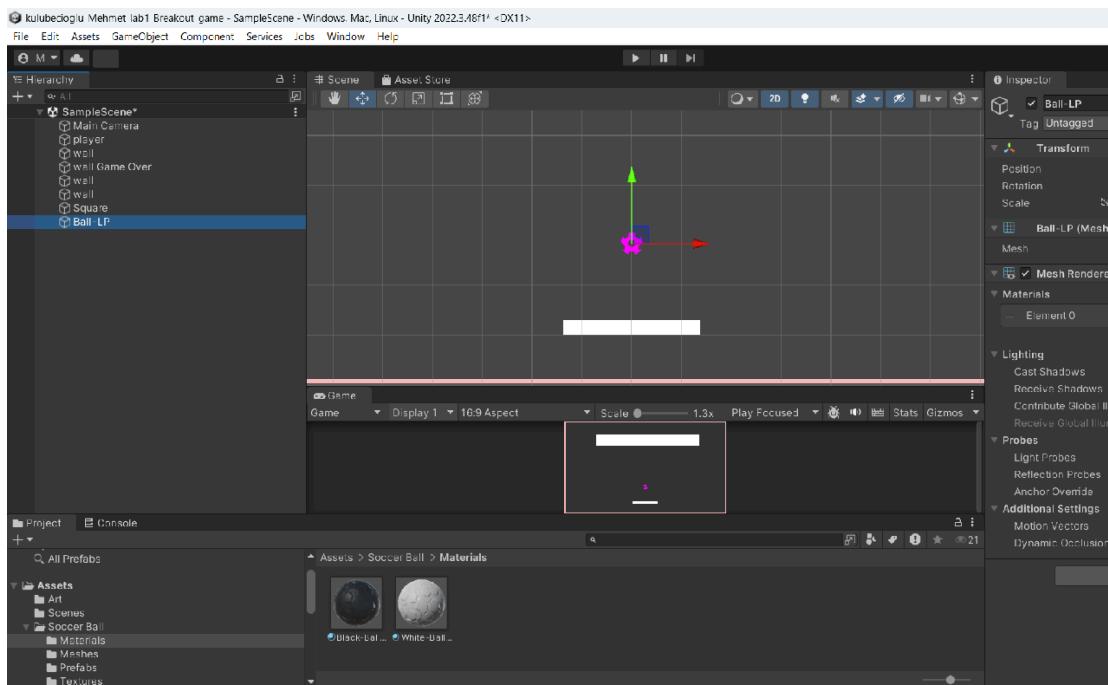
While the package downloads to my computer, a progress bar appears, and the Pause and Cancel buttons replace the Download button I clicked. When the download finishes, the Package Manager replaces Download with Re-Download, and an Import button appears next to Re-Download.

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

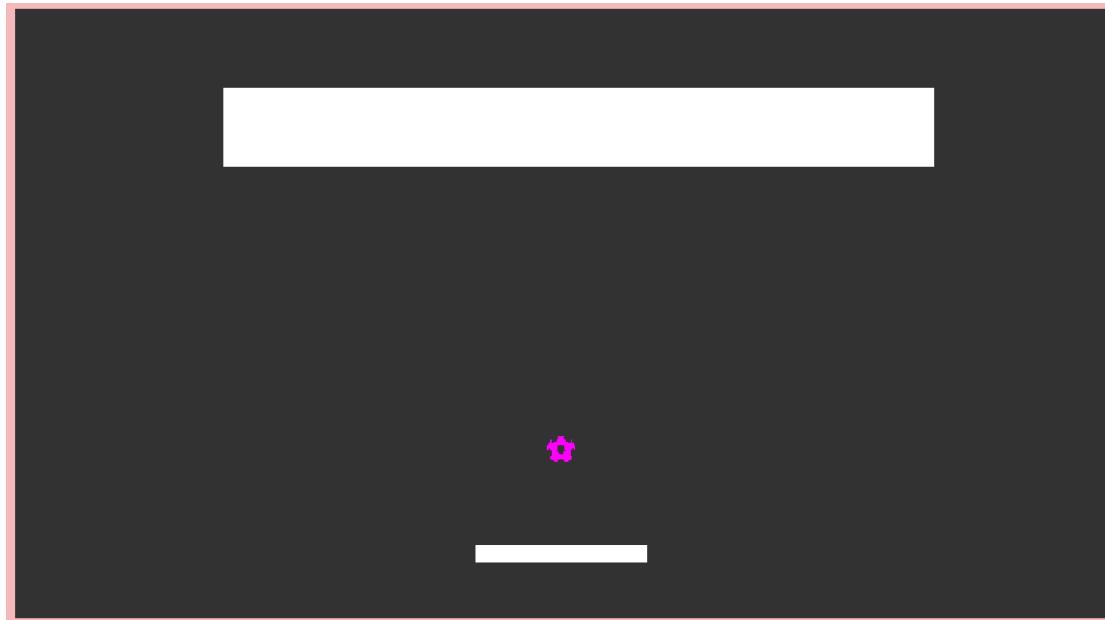
After importing, I will have a new collection in Assets called “Low Polygon Soccer Ball.”



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!**

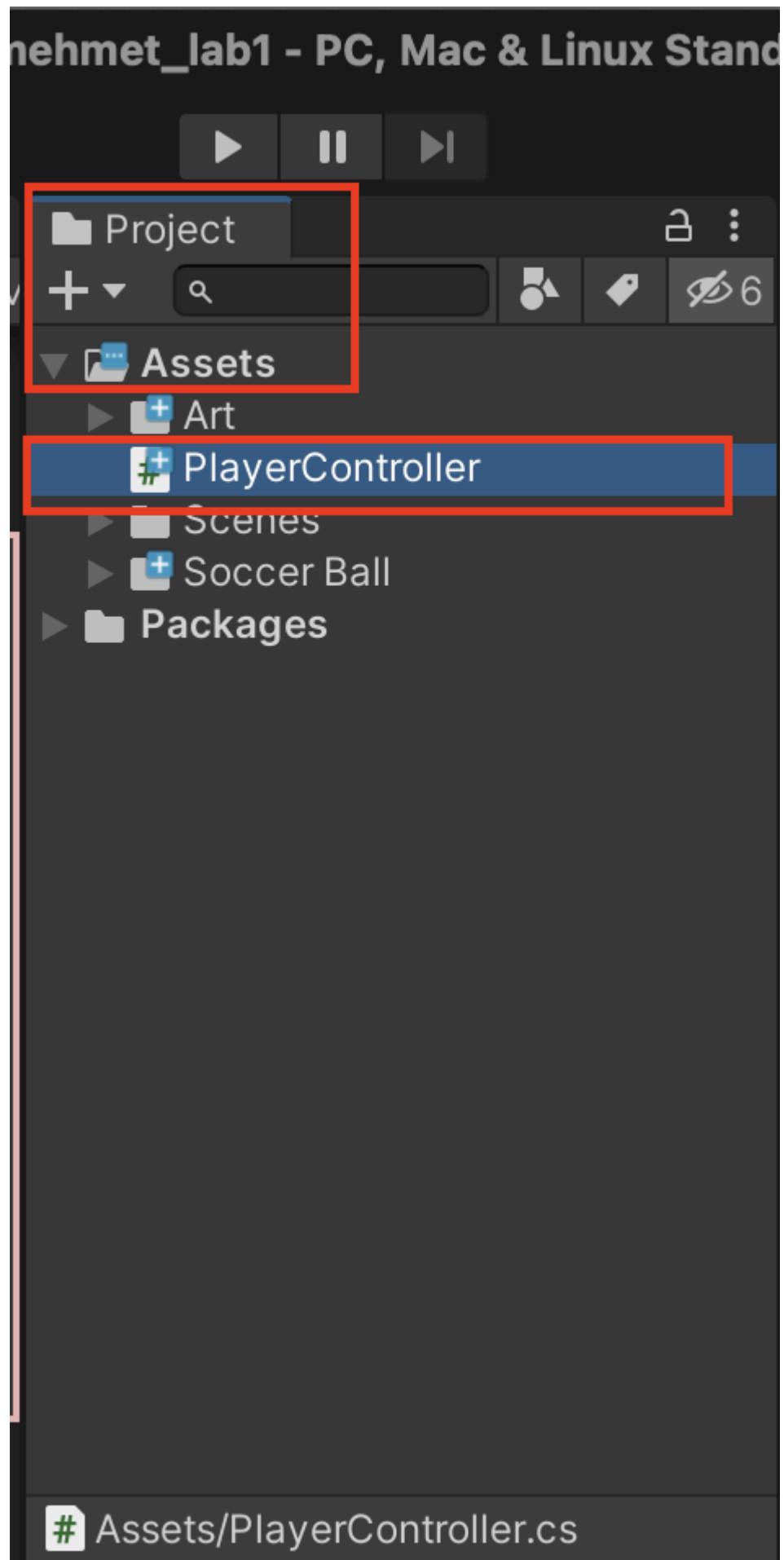
**Since my computer is broken, I will continue to improve my game with my Macbook from here on.**

### **My Next Steps:**

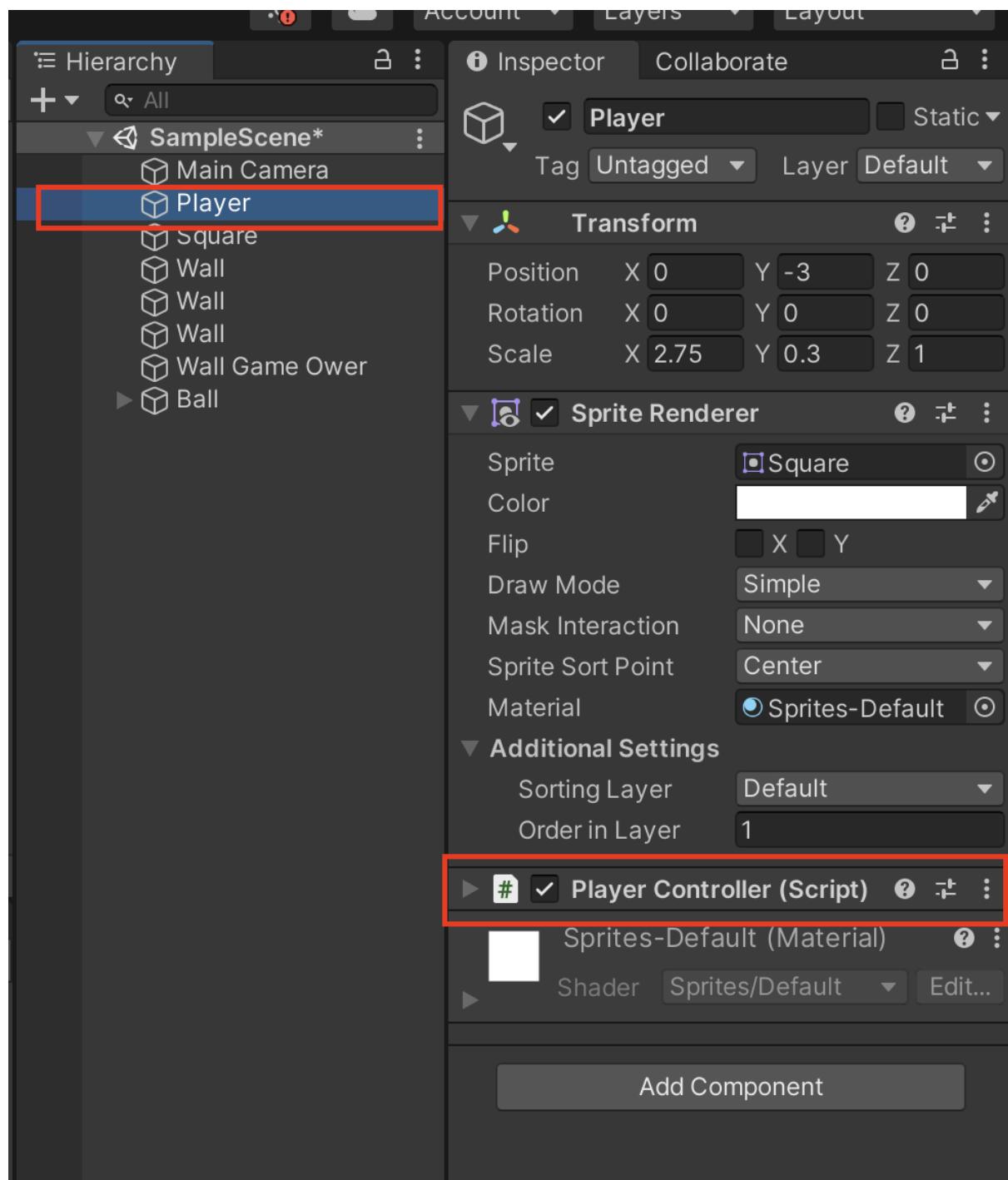
1. I will create an idea for my game.
2. I will find assets suitable for my game.
3. I will create a base Scene for my game.

### **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.



### 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.

The screenshot shows the Visual Studio Code interface with a dark theme. The main editor window displays the file `PlayerController.cs`. The code defines a `PlayerController` class that inherits from `MonoBehaviour`. It contains two methods: `Start()` and `Update()`. The `Start()` method is annotated with a comment stating it is called before the first frame update. The `Update()` method is annotated with a comment stating it is called once per frame. Below the editor, the bottom panel shows the `PROBLEMS`, `OUTPUT`, `DEBUG CONSOLE`, and `...`  tabs. The `OUTPUT` tab is active, displaying a message from `ms-dotnettools.csharp` indicating it tried to install .NET 8.0.8~arm64 but found an already installed version. It also shows the download of the .NET Runtime and the progress of downloading .NET version(s) 8.0.8~arm64.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PlayerController : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    ...    .NET Install Tool    v    ...    ^    x

ms-dotnettools.csharp tried to install .NET 8.0.8~arm64 but that install had already been requested. No downloads or changes were made.  
ms-dotnettools.csharp requested to download the .NET Runtime.  
Downloading .NET version(s) 8.0.8~arm64 .....

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

The screenshot shows the Visual Studio Code interface with a dark theme. The main editor window displays the file `PlayerController.cs`. The code now includes a private float variable `speed` with a value of 1, marked with the `[SerializeField]` attribute. The `Update()` method has been modified to add the product of `input * Vector3.right * speed` to the transform's position, effectively moving the player horizontally. The bottom panel shows the `PROBLEMS`, `OUTPUT`, `DEBUG CONSOLE`, `TERMINAL`, and `PORTS` tabs. The `OUTPUT` tab is active, showing a message indicating no problems were detected in the workspace. A watermark "Ekran Resmi" is visible at the bottom of the screen.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PlayerController : MonoBehaviour
6 {
7     [SerializeField] private float speed = 1;
8
9     private void Update()
10    {
11        var input = Input.GetAxis("Horizontal");
12
13        transform.position += input * Vector3.right * speed;
14    }
15 }
```

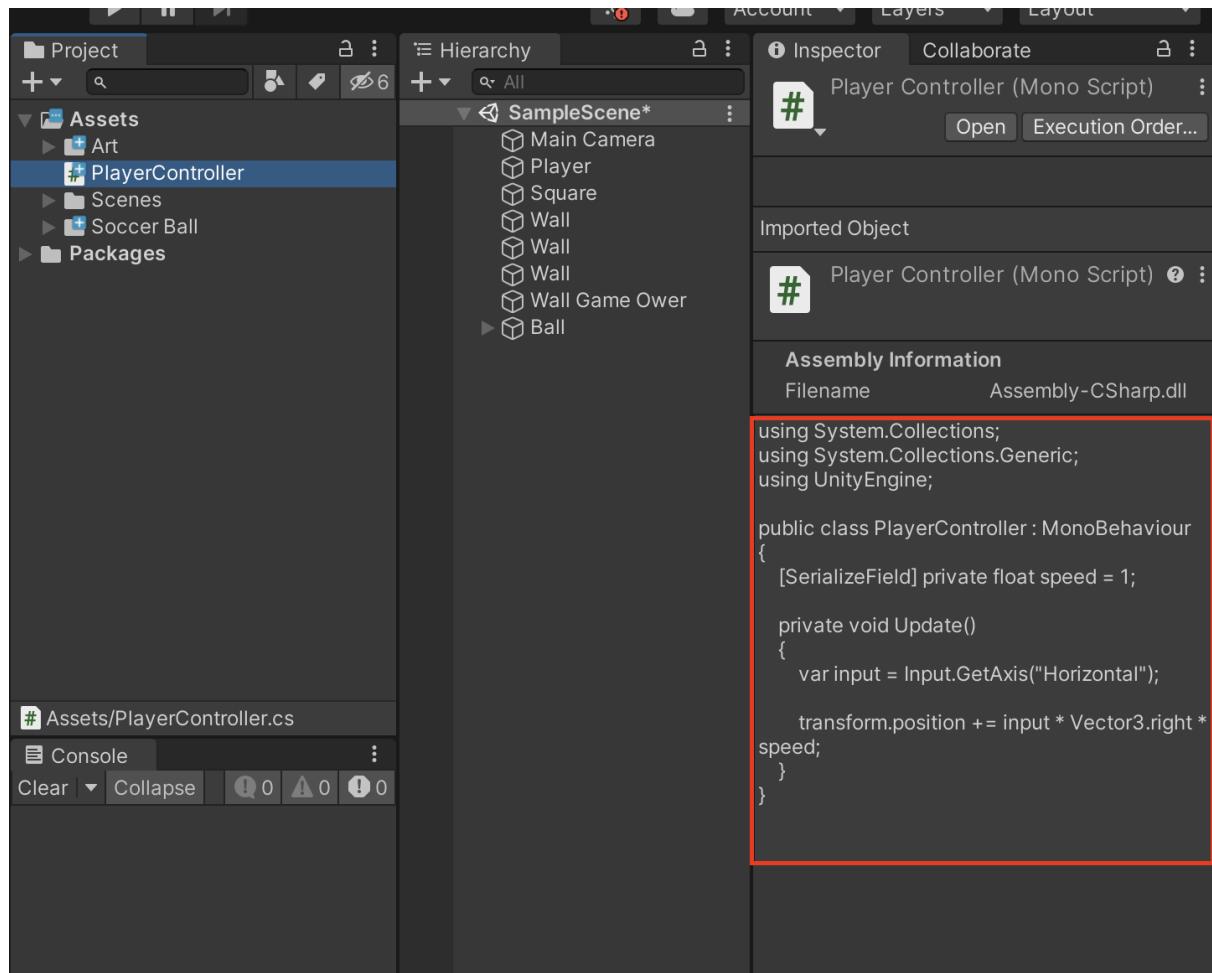
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    Filter (e.g. text, \*\*/\*.ts, \*\*/\*node\_modules/\*\*\*)    ...    ^    x

No problems have been detected in the workspace.

Ekran Resmi

We can also see our script codes on Unity:

- For this, after clicking on PlayerScript, we should look at the Inspector 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.

ccount Layers Layout

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)

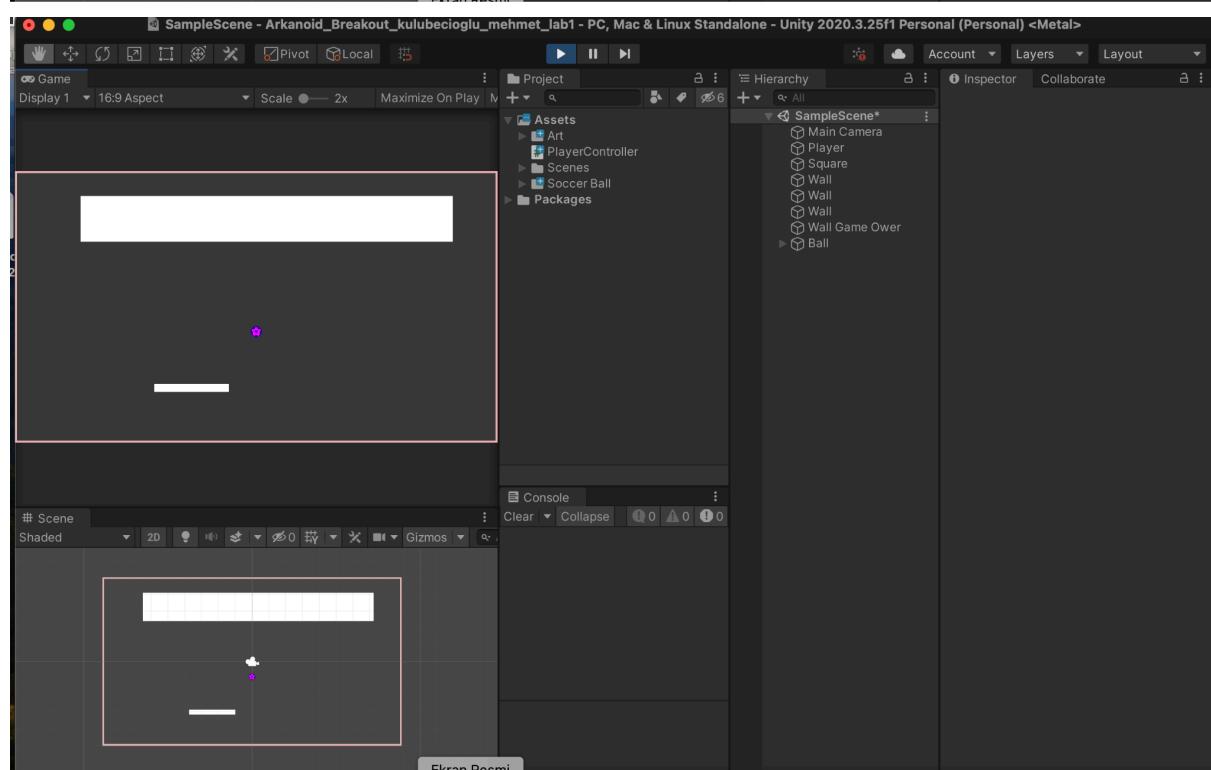
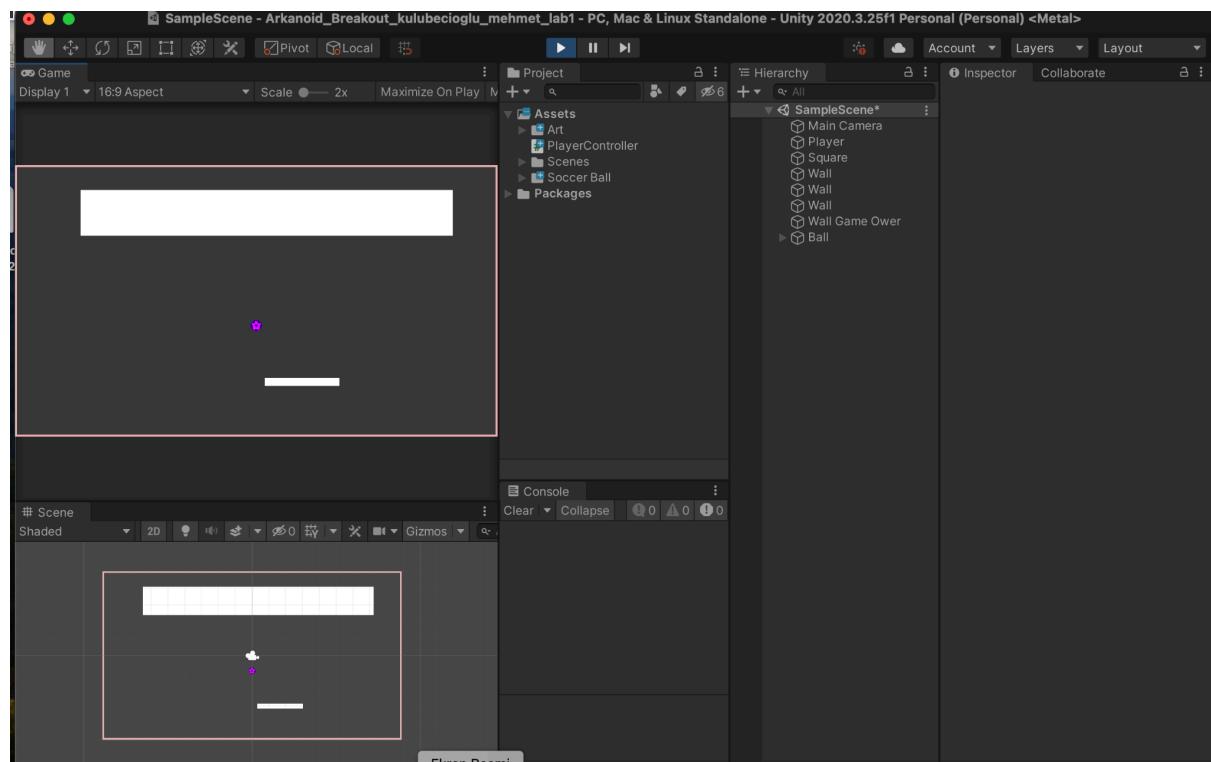
Script # PlayerController   
Speed 0.1

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.

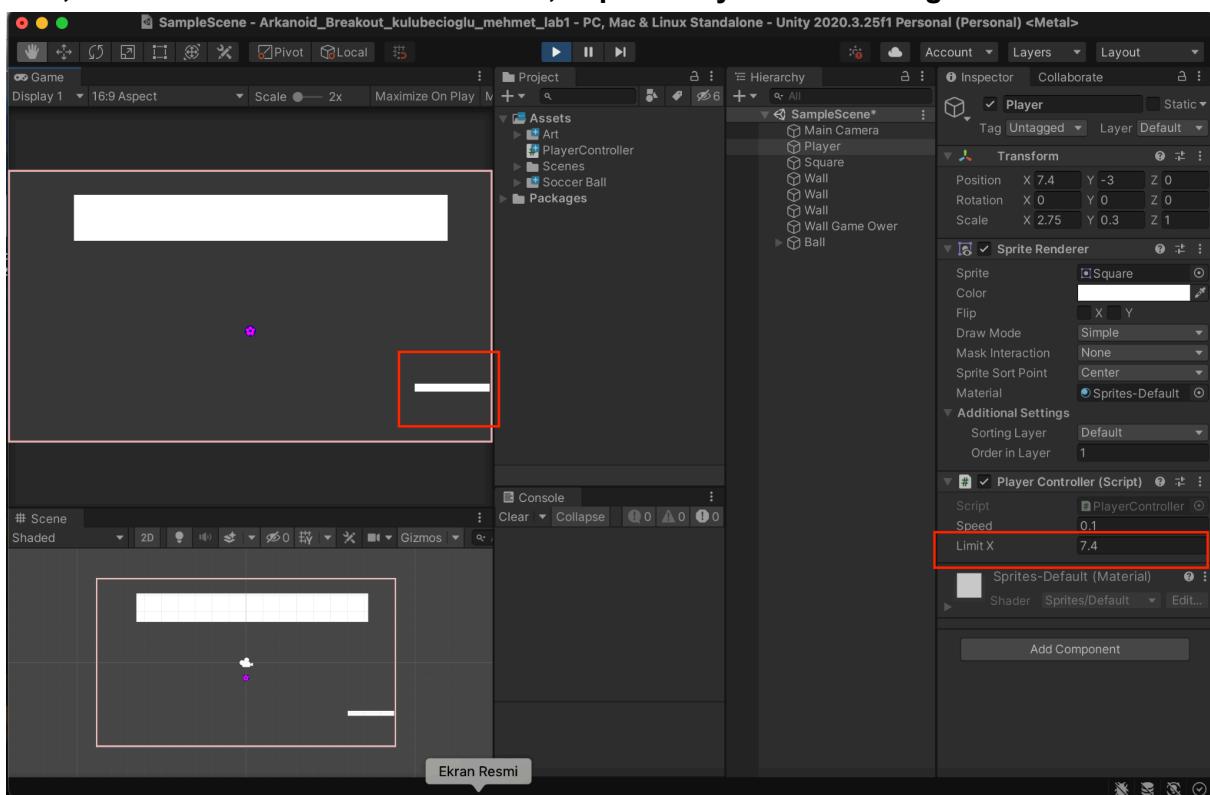
```

C# PlayerController.cs •
Users > mehmet > Arkanoid_Breakout_kulubecioglu_mehmet_lab1 > Assets > C# PlayerController.cs > PlayerController > Update
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class PlayerController : MonoBehaviour
6  {
7      [SerializeField] private float speed = 1;
8      [SerializeField] private float limitX = 7.4f;
9
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.

The screenshot shows a code editor interface with a dark theme. At the top, there's a navigation bar with back/forward buttons and a search bar. Below it, a breadcrumb trail shows the file path: Users > mehmet > Arkanoid\_Breakout\_kulubeciooglu\_mehmet\_lab1 > Assets > PlayerController.cs. To the right of the path are icons for a file, a person, and update.

The main area displays the C# code for `PlayerController.cs`:

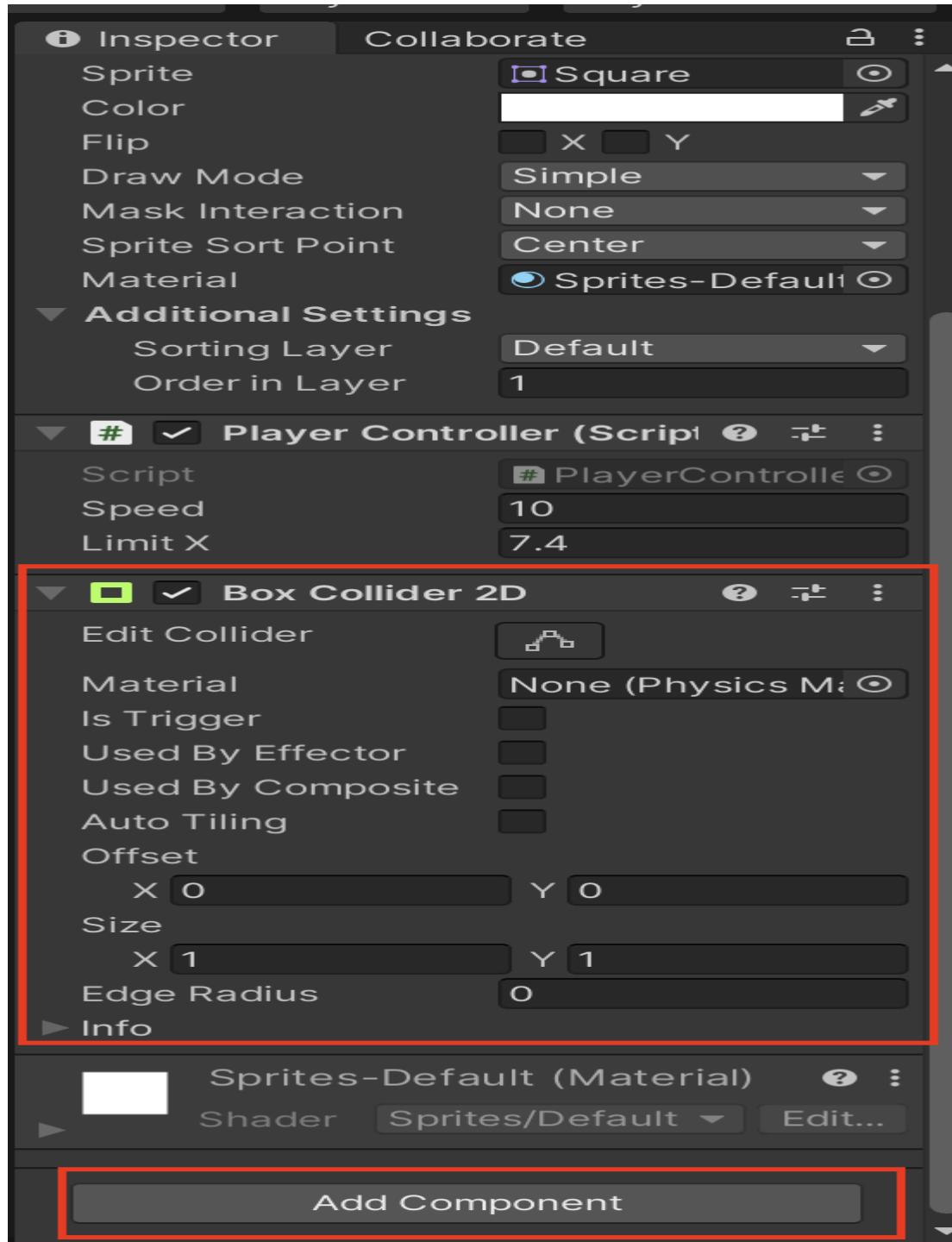
```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class PlayerController : MonoBehaviour
6  {
7      [SerializeField] private float speed = 1;
8      [SerializeField] private float limitX = 7.4f;
9
10     private void Update()
11     {
12         var input = Input.GetAxis("Horizontal");
13
14         var position = transform.position;
15         position.x += input * speed * Time.deltaTime;
16         position.x = Mathf.Clamp(position.x, -limitX, +limitX);
17         transform.position = position;
18     }
19 }
20
21
```

Line 15, which contains the assignment `position.x += input * speed * Time.deltaTime;`, is highlighted with a red rectangular selection.

At the bottom of the editor, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The PROBLEMS tab is currently selected. A message below the tabs says "No problems have been detected in the workspace." To the right, there's a filter input field with placeholder text "Filter (e.g. text, \*\*/\*ts, !\*\*/node\_modules/\*\*)".

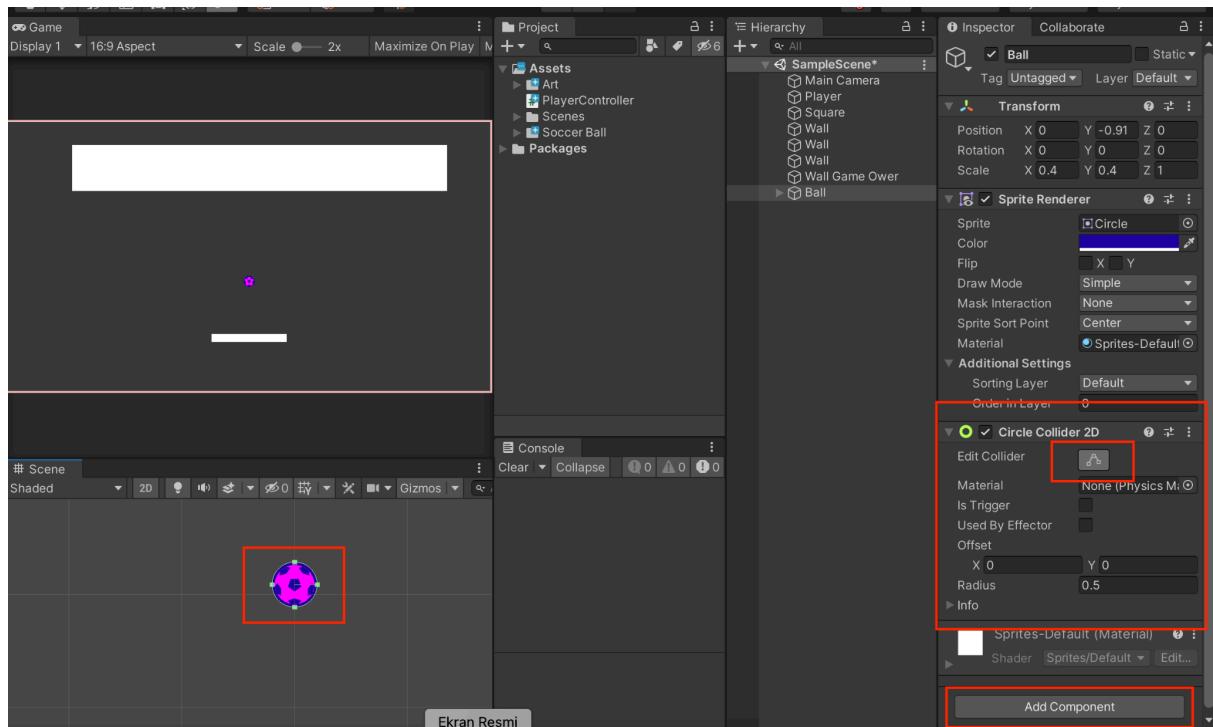
# Basic Character Movement & Physics

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

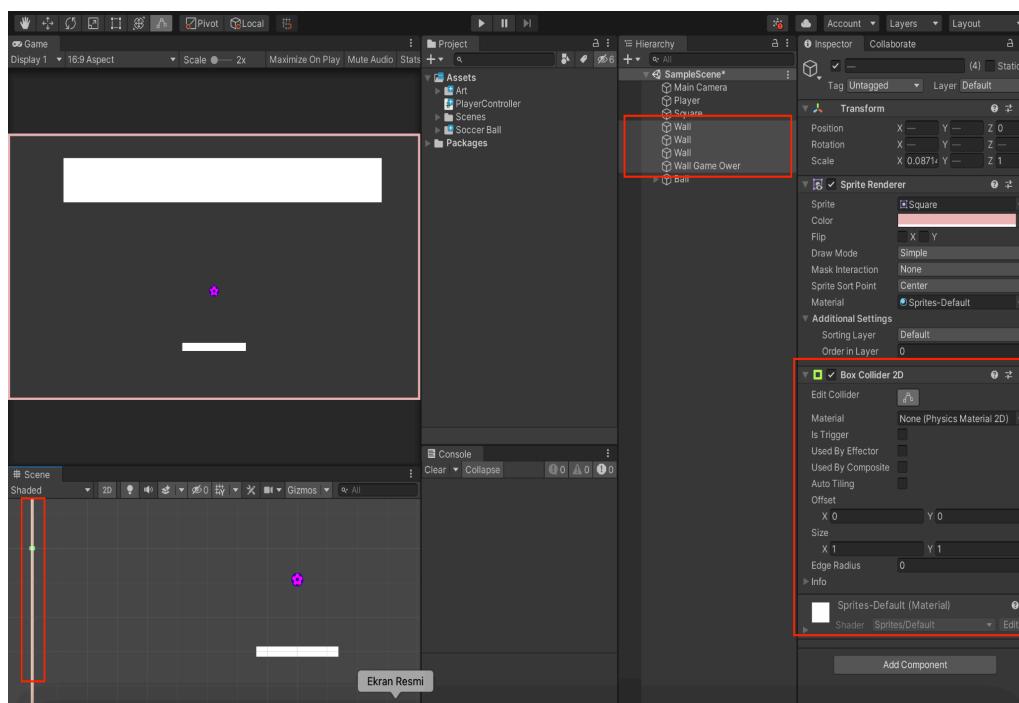


# Interactions and Object Collection

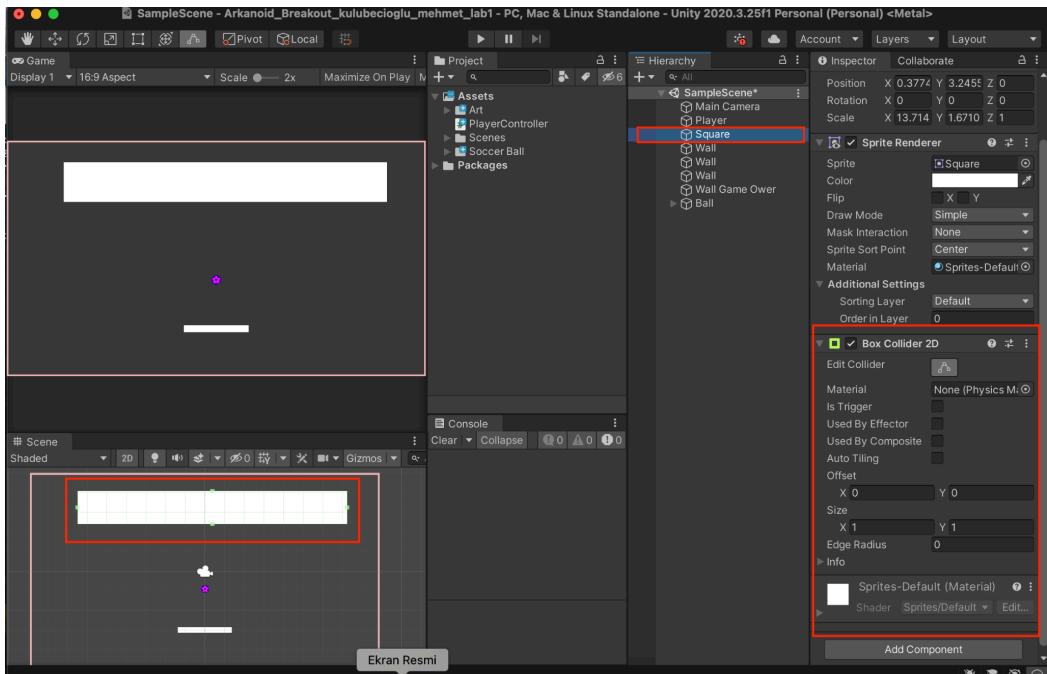
- adding circle collider to ball for collision



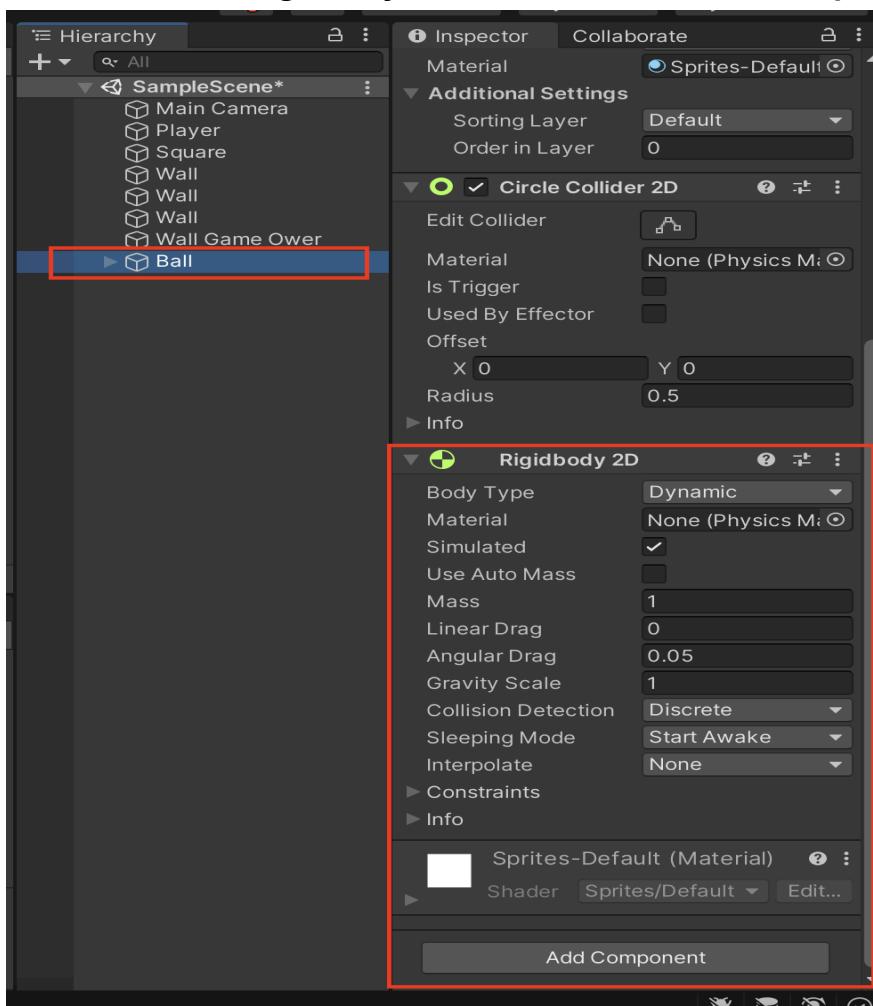
- adding Box Collider to walls for collision



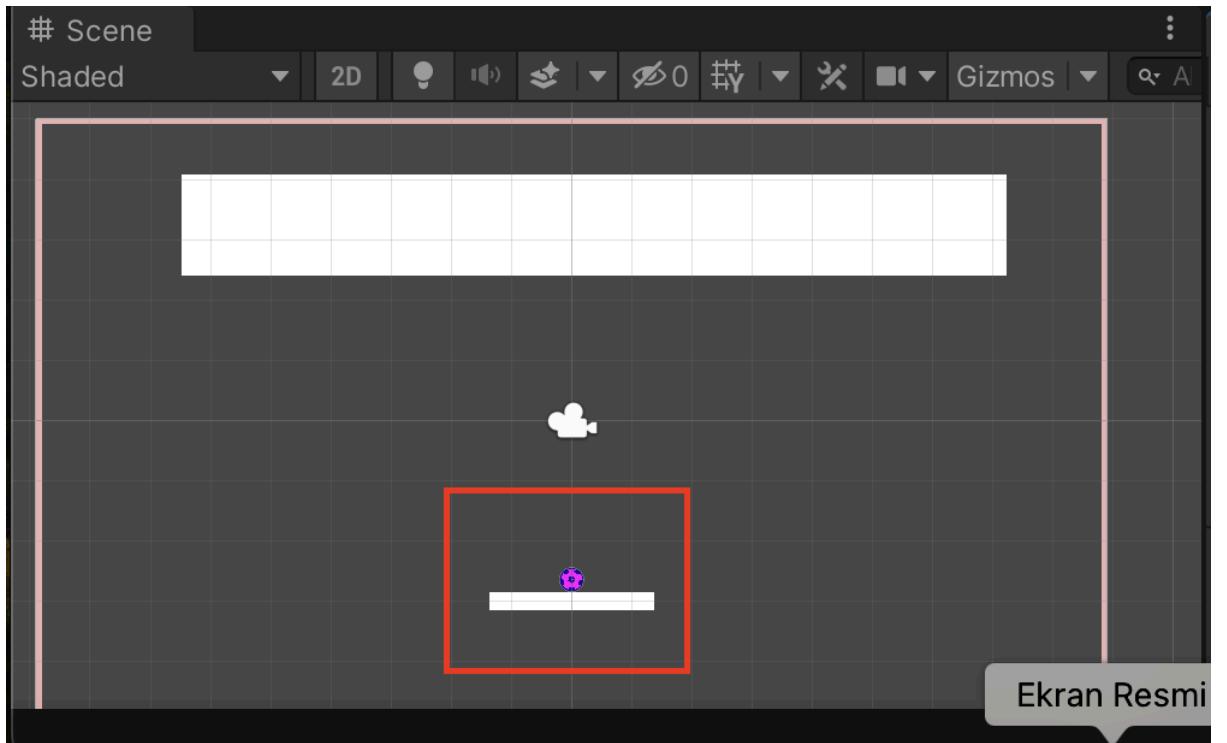
- Now I added box collider to the obstacles for collision.



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



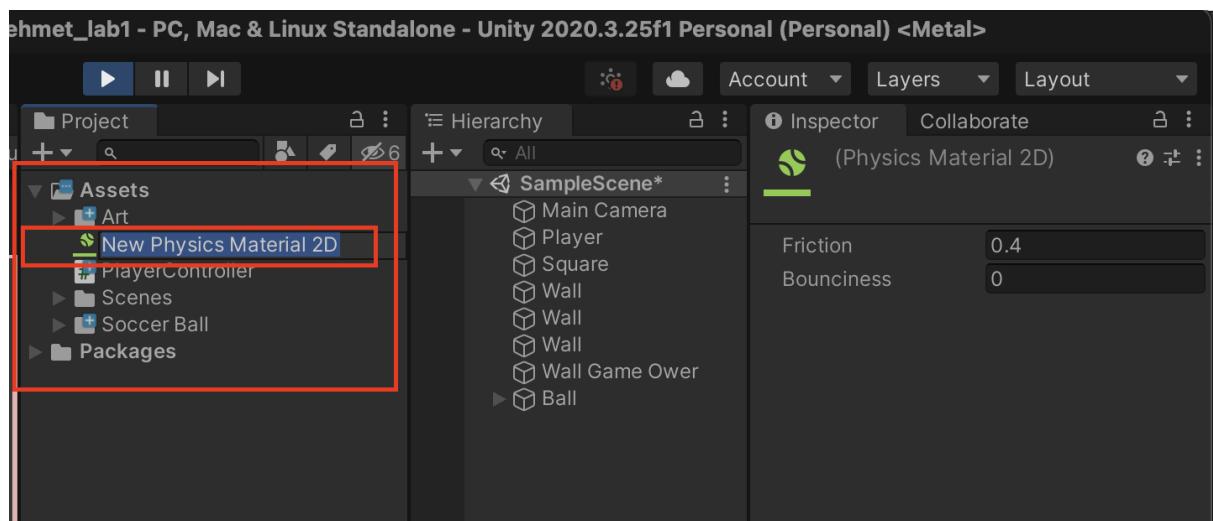
- Now let's check whether the ball dropped by starting the project



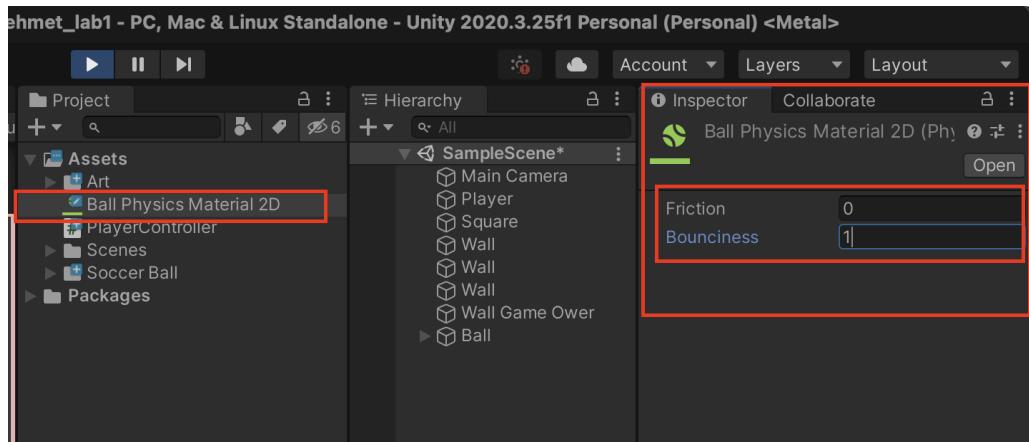
- I started the project and observed that the ball dropped, so we have no problems so far.

### Adding physics material to give physical properties to the ball:

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

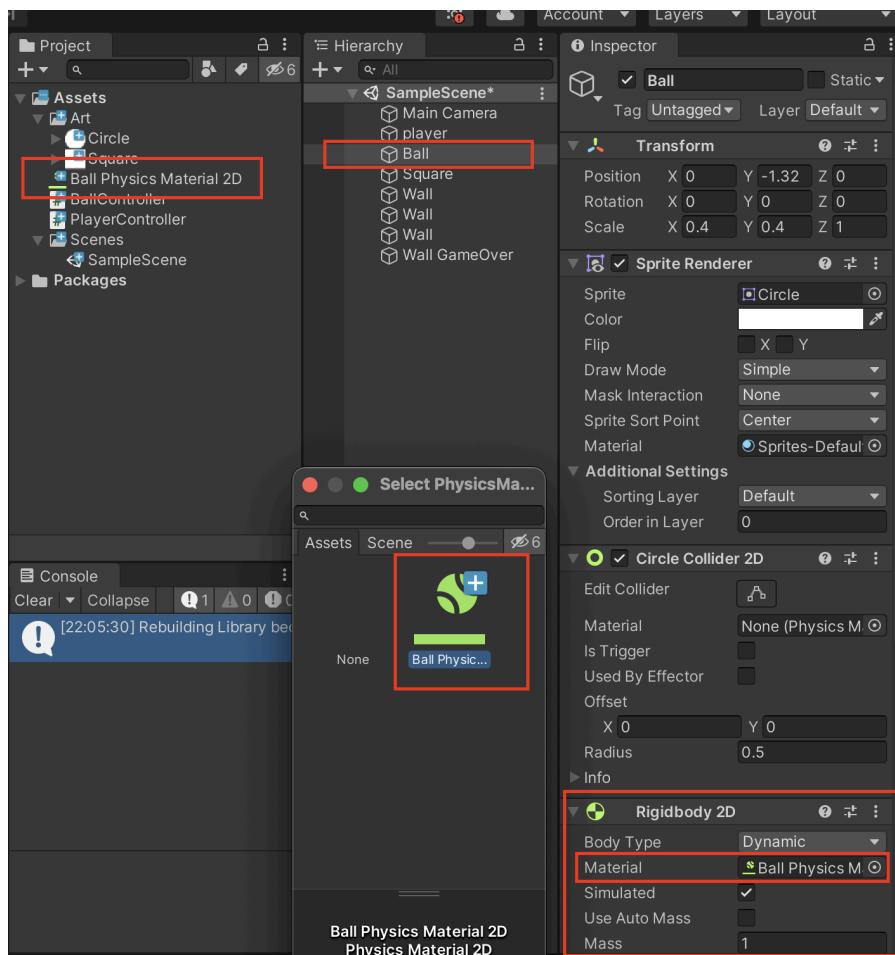


- Add friction and bounce values

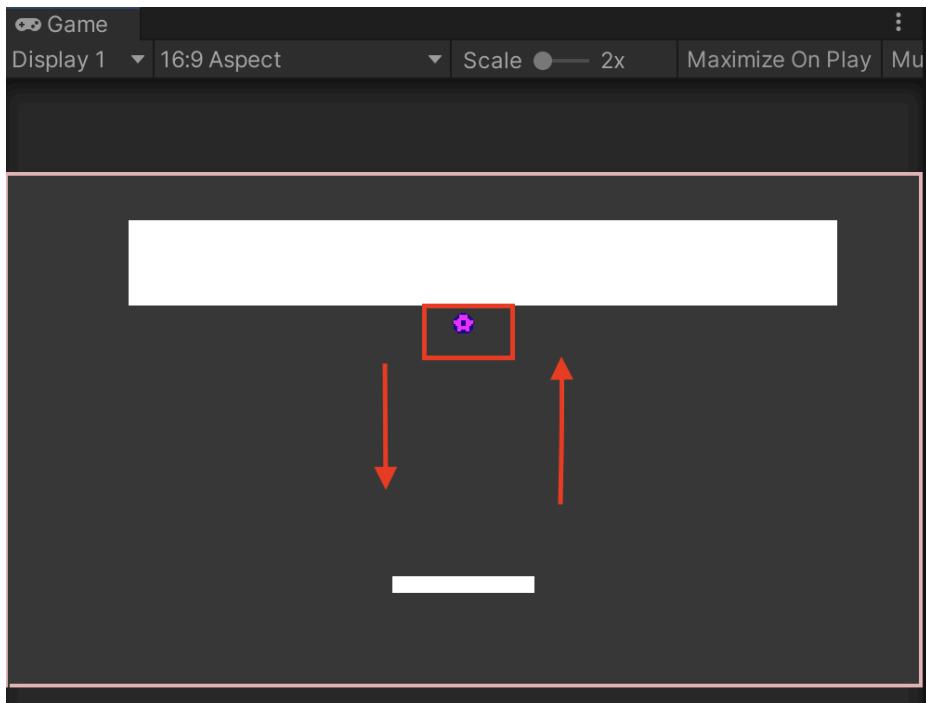


**Adding the physics materials we created to the ball:**

- 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:

```

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

C# PlayerController.cs C# BallController.cs X
Users > mehmet > Kulubecioglu_mehmet_1 > Assets > C# BallController.cs
1  using UnityEngine;
2
3  public class BallController : MonoBehaviour
4  {
5      private Rigidbody2D _rigidbody2D;
6
7      [SerializeField] private float speed = 1f;
8
9      private void Awake()
10     {
11         _rigidbody2D = GetComponent<Rigidbody2D>();
12     }
13
14     private void Start()
15     {
16         var force = Vector2.down;
17         _rigidbody2D.AddForce(force * speed);
18     }
19 }

```

The image shows the Unity Editor's code editor window. The title bar says 'Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More'. The tab bar shows 'PlayerController.cs' and 'BallController.cs X'. The code editor displays the 'BallController.cs' script. The script uses the Unity Engine and defines a 'BallController' class that inherits from 'MonoBehaviour'. It contains a private 'Rigidbody2D' variable named '\_rigidbody2D'. A public float variable 'speed' is set to 1f with the attribute '[SerializeField]'. The 'Awake' method initializes '\_rigidbody2D' to the component of the same type on the current object. The 'Start' method sets a downward force vector and adds it to the rigidbody's forces.

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

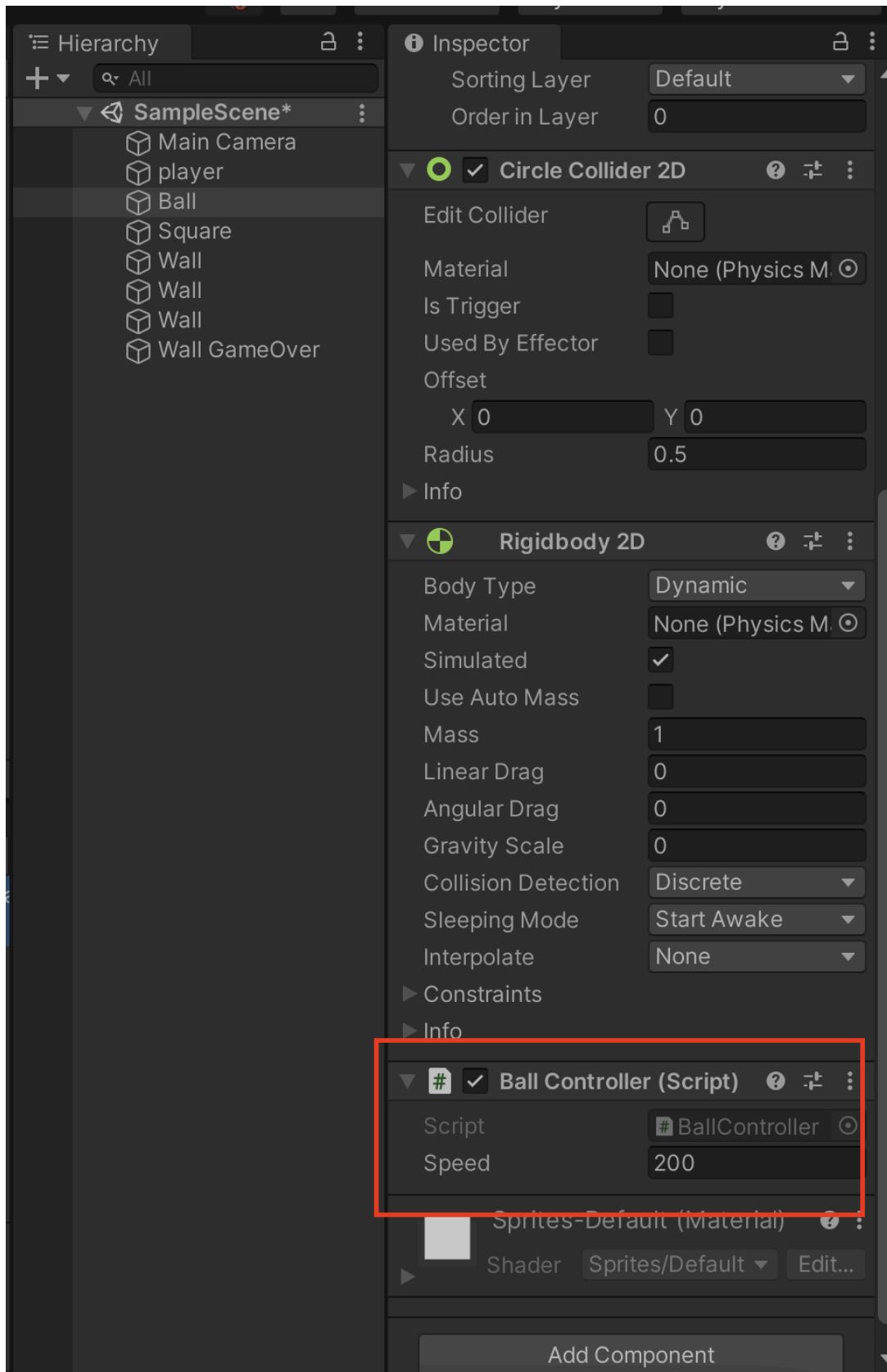
The image shows the Unity Inspector window with two components selected:

- Rigidbody 2D**:
  - Body Type: Dynamic
  - Material: None (Physics M.)
  - Simulated: checked
  - Use Auto Mass: unchecked
  - Mass: 1
  - Linear Drag: 0
  - Angular Drag: 0
  - Gravity Scale: 0
  - Collision Detection: Discrete
  - Sleeping Mode: Start Awake
  - Interpolate: None
- Ball Controller (Script)**:
  - Script: # BallController
  - Speed: 1

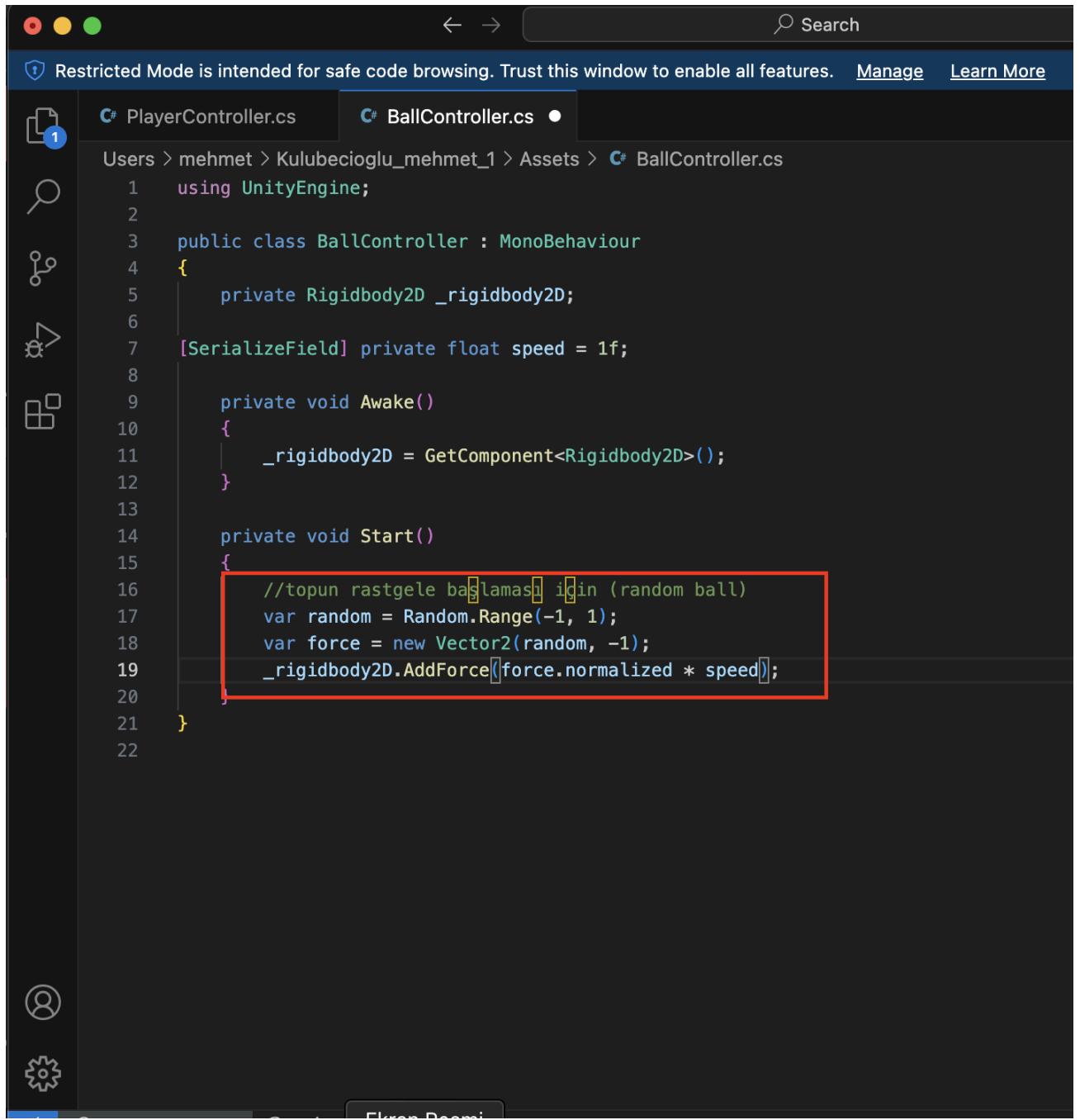
A red box highlights the Rigidbody 2D component settings.

## Adjusting the speed of the ball:

- I entered the speed as 200

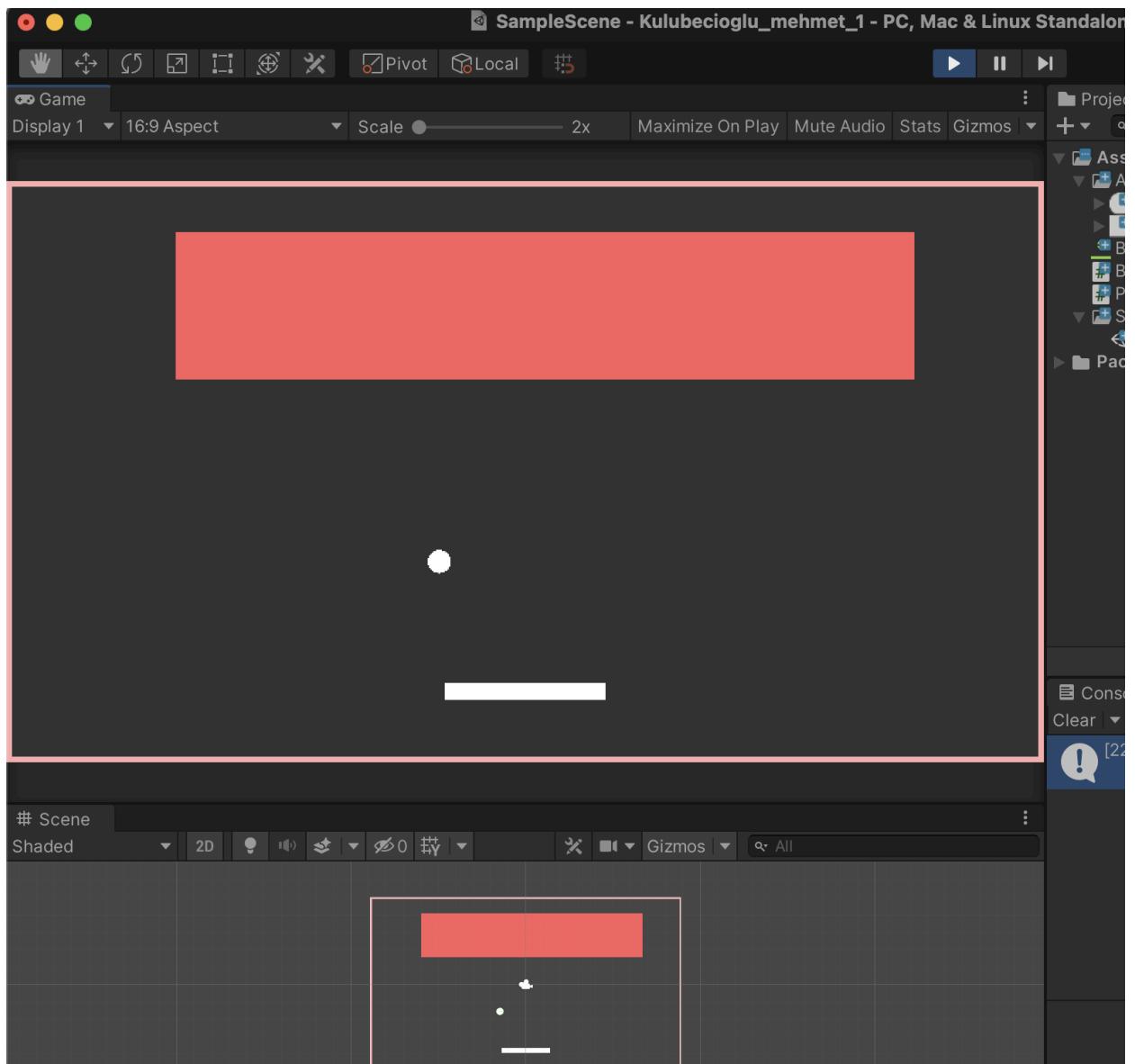


## Starting the ball randomly to the right or left:



```
1  using UnityEngine;
2
3  public class BallController : MonoBehaviour
4  {
5      private Rigidbody2D _rigidbody2D;
6
7      [SerializeField] private float speed = 1f;
8
9      private void Awake()
10     {
11         _rigidbody2D = GetComponent<Rigidbody2D>();
12     }
13
14     private void Start()
15     {
16         //topun rastgele baslasmas\u0111 i\u0111in (random ball)
17         var random = Random.Range(-1, 1);
18         var force = new Vector2(random, -1);
19         _rigidbody2D.AddForce(force.normalized * speed);
20     }
21 }
22
```

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.

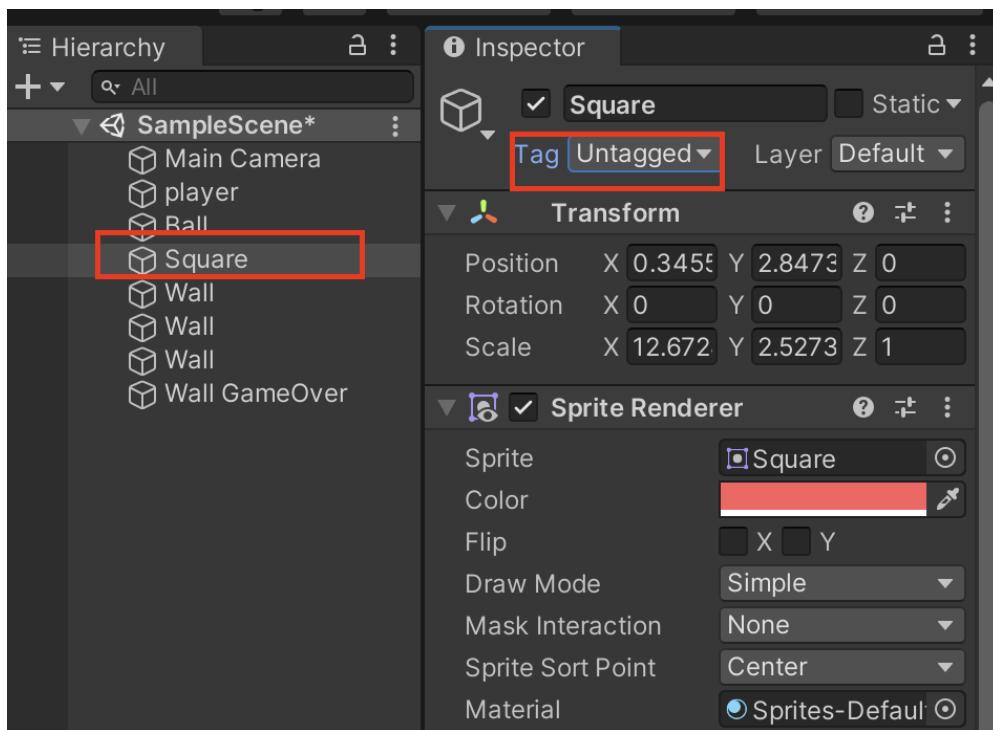
## Interactions

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

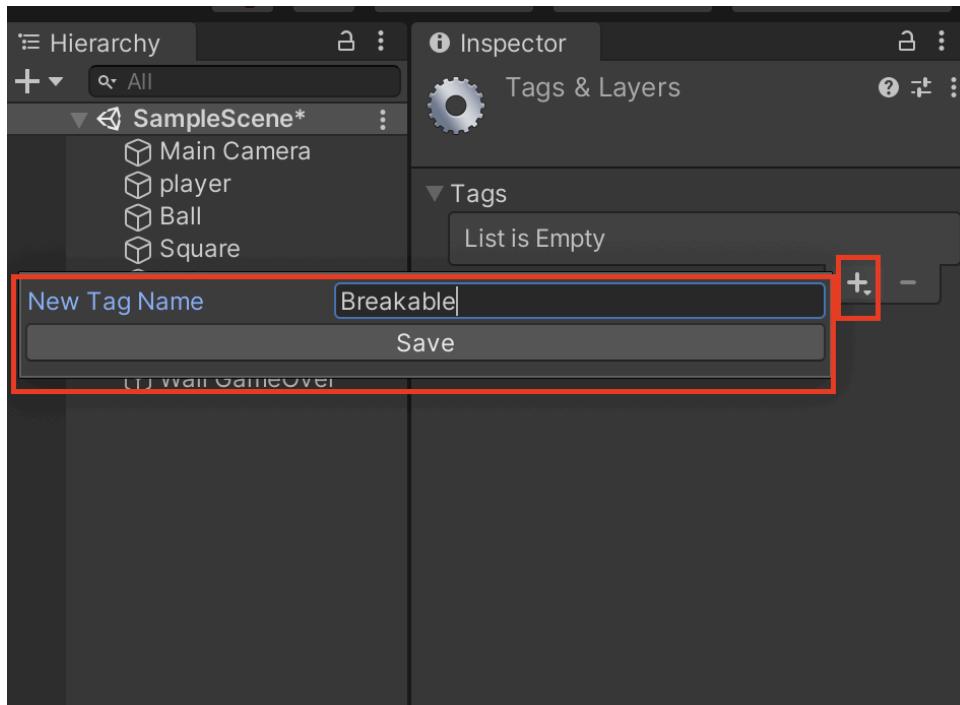
```
using UnityEngine;
public class BallController : MonoBehaviour
{
    private Rigidbody2D _rigidbody2D;
    [SerializeField] private float speed = 1f;
    private void Awake()
    {
        _rigidbody2D = GetComponent<Rigidbody2D>();
    }
    private void Start()
    {
        var random = Random.Range(-1, 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);
        }
    }
}
```

## breaking blocks:

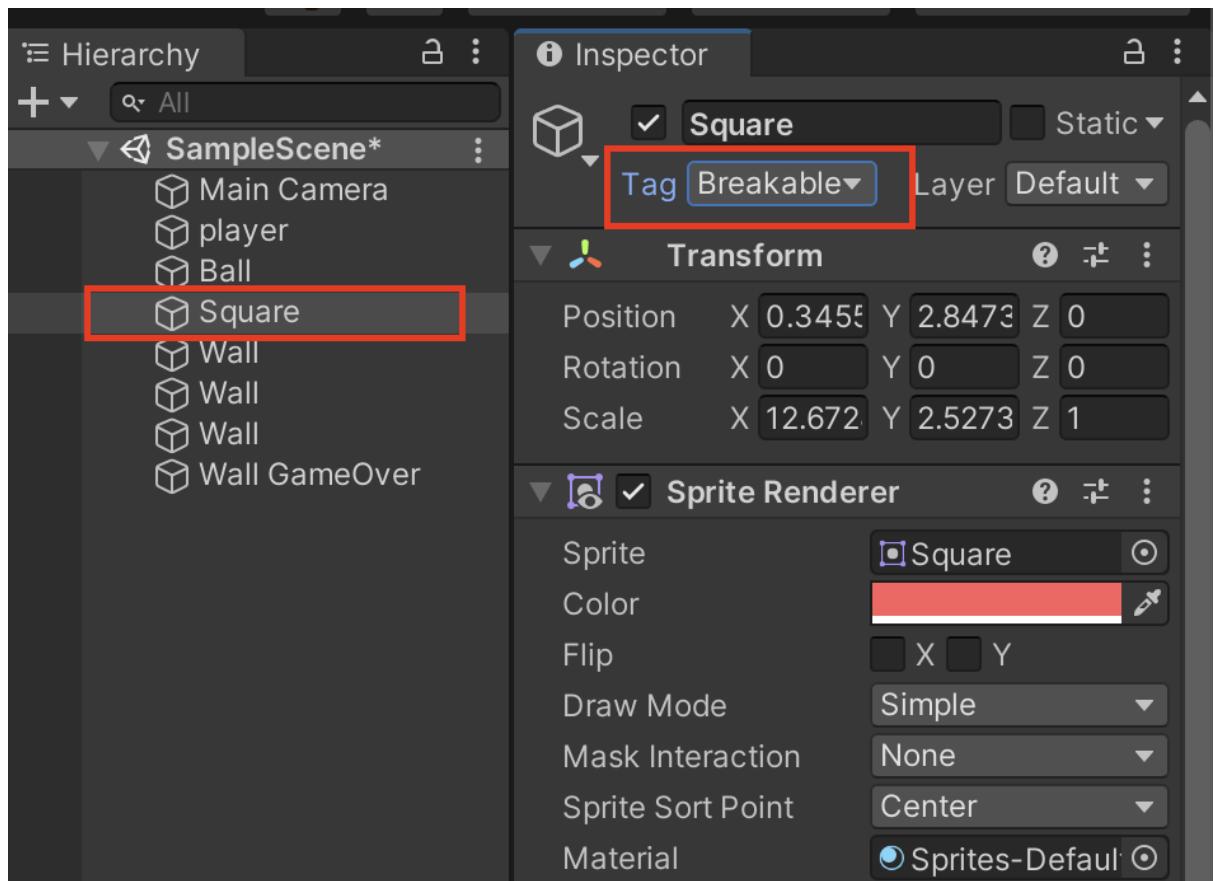
- Now I go into Square to break the blocks and make the necessary settings in the inspector section



## Creating breakable:

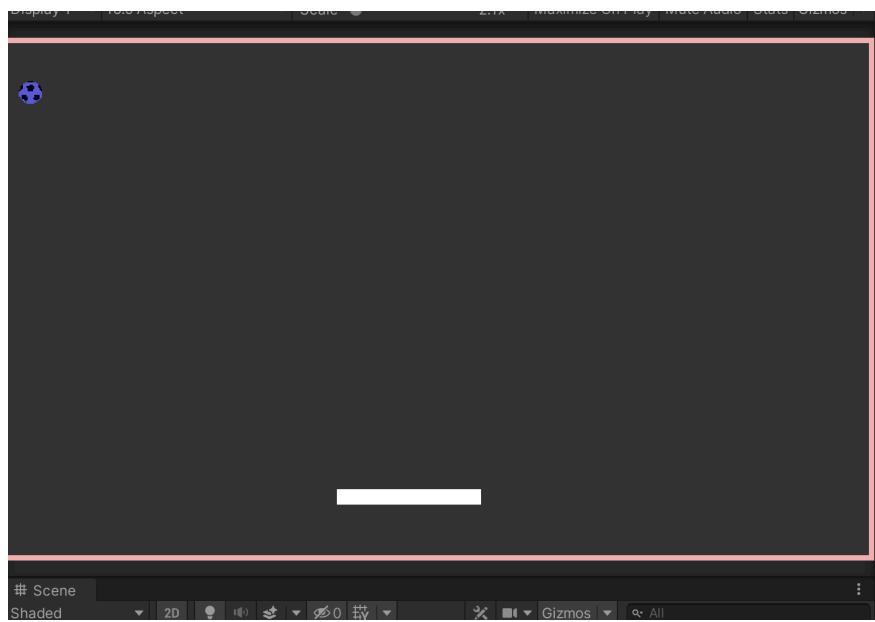


- Now I will transfer breakable to the object



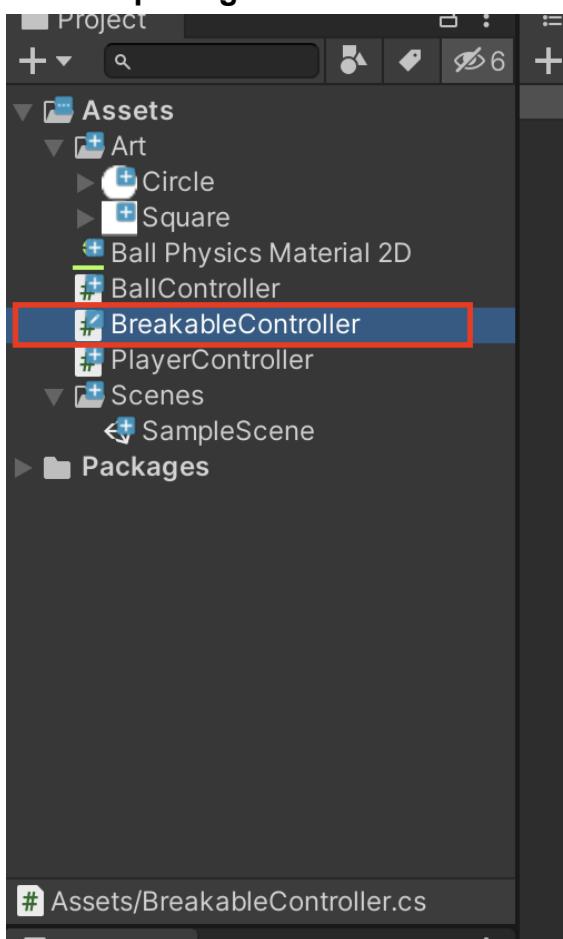
I usually add things by drag and drop.

- 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.

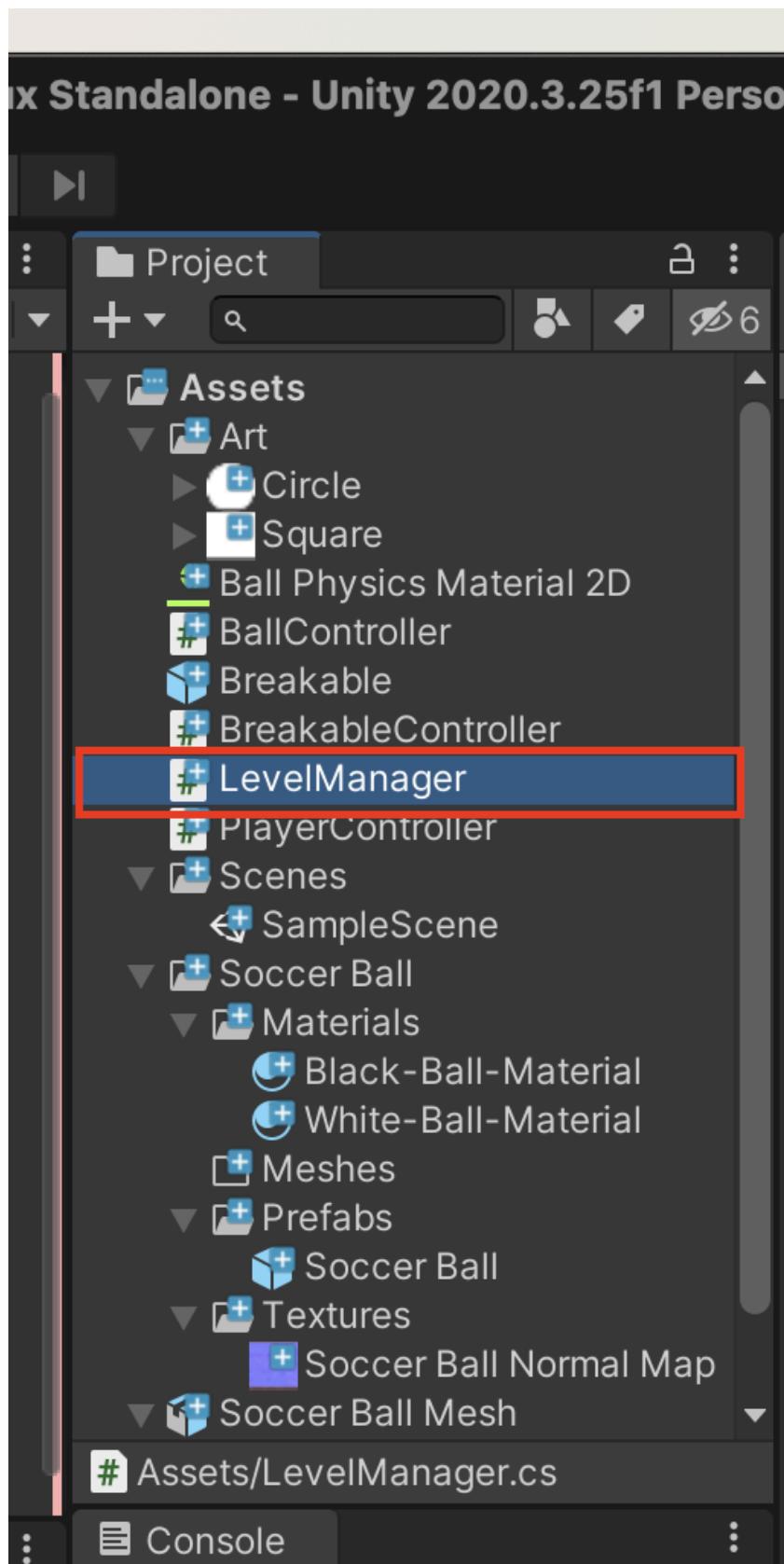


- I ran the game and when the ball hit the breakable, the breakable broke and the ball bounced back.

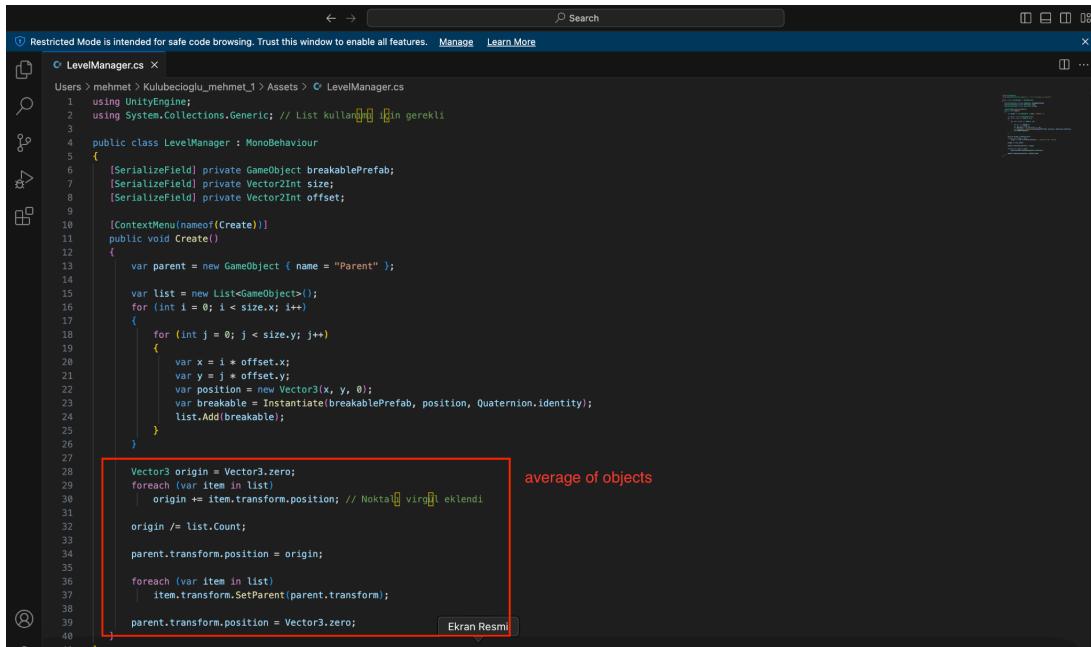
- opening new Breakable controller script:



- Opening new LevelManager controller script:



- Arranging objects and centering objects LevelManager code content:



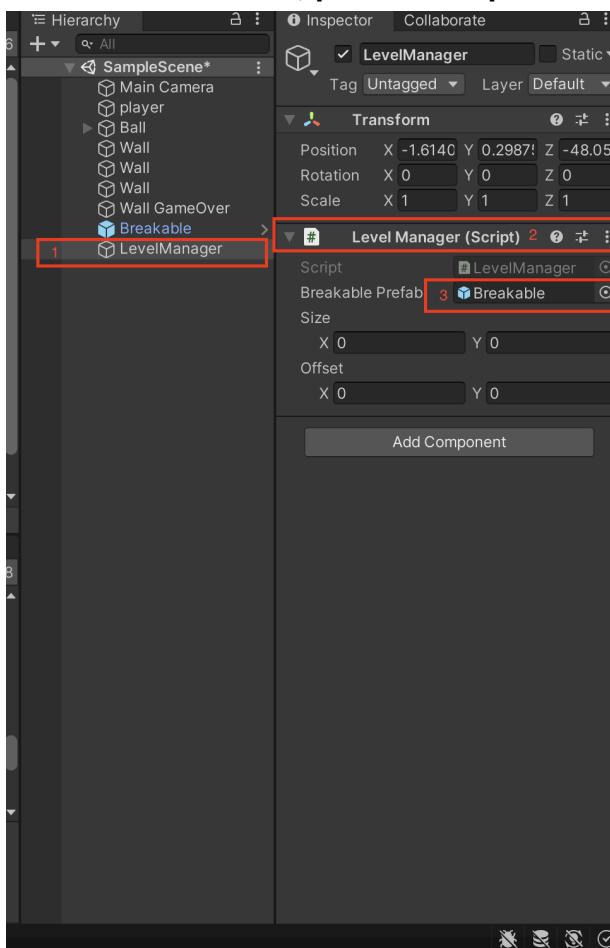
```

1 Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
2 LevelManager.cs
3
4 using UnityEngine;
5 using System.Collections.Generic; // List kullanmak için gereklidir
6
7 public class LevelManager : MonoBehaviour
8 {
9     [SerializeField] private GameObject breakablePrefab;
10    [SerializeField] private Vector2Int size;
11    [SerializeField] private Vector2Int offset;
12
13    [ContextMenu(nameof(Create))]
14    public void Create()
15    {
16        var parent = new GameObject { name = "Parent" };
17
18        var list = new List<GameObject>();
19        for (int i = 0; i < size.x; i++)
20        {
21            for (int j = 0; j < size.y; j++)
22            {
23                var x = i * offset.x;
24                var y = j * offset.y;
25                var position = new Vector3(x, y, 0);
26                var breakable = Instantiate(breakablePrefab, position, Quaternion.identity);
27                list.Add(breakable);
28            }
29        }
30
31        Vector3 origin = Vector3.zero;
32        foreach (var item in list)
33        {
34            origin += item.transform.position; // Nokta listesi ekleniyor
35        }
36        origin /= list.Count;
37
38        parent.transform.position = origin;
39
40        foreach (var item in list)
41        {
42            item.transform.SetParent(parent.transform);
43        }
44        parent.transform.position = Vector3.zero;
45    }
46 }

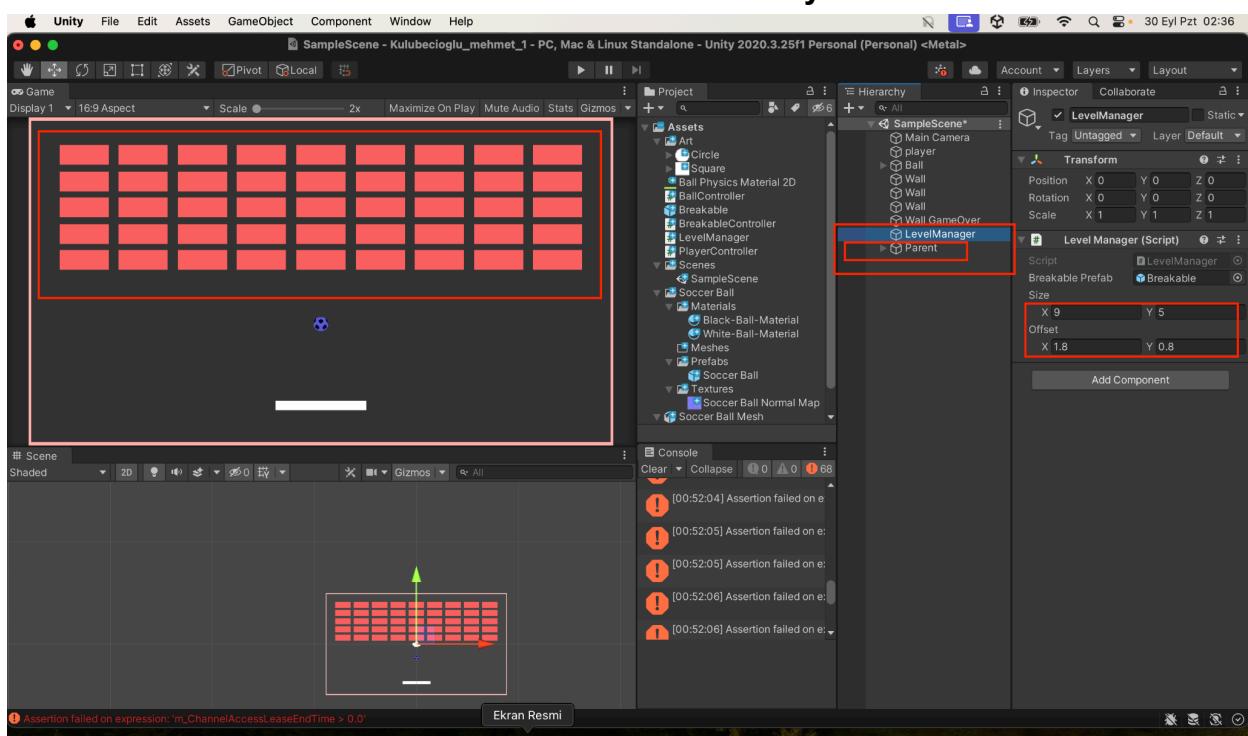
```

average of objects

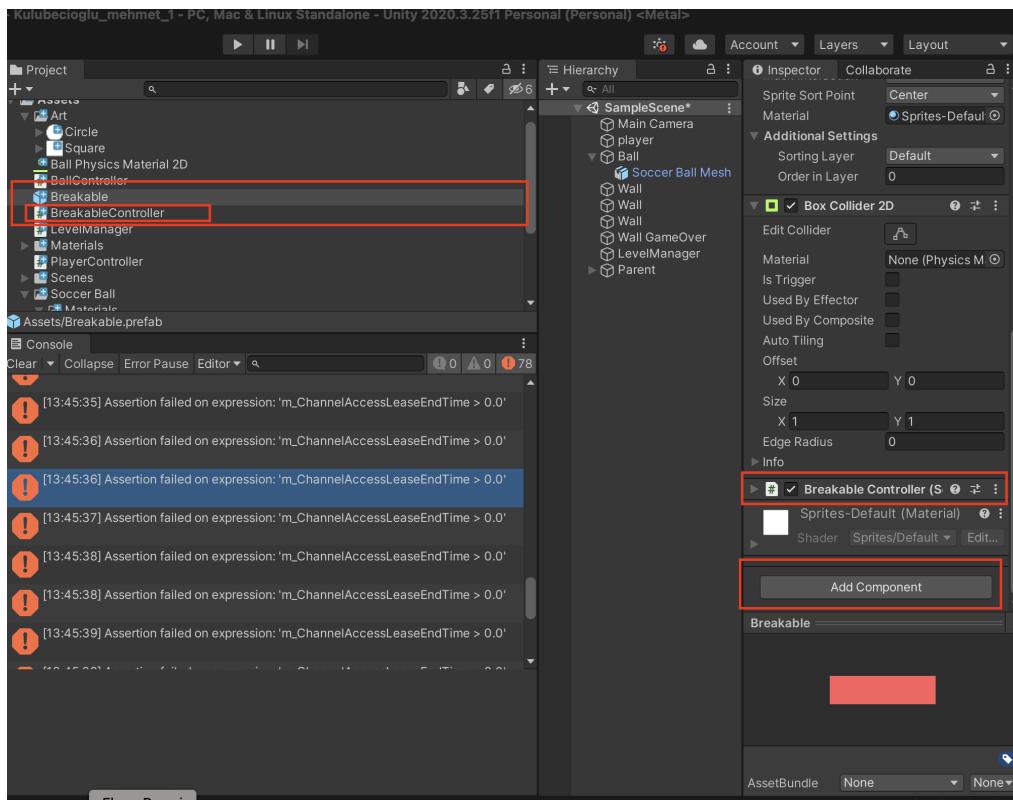
- I created a new one, put the script in it and put the breakout in it:



- 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

```

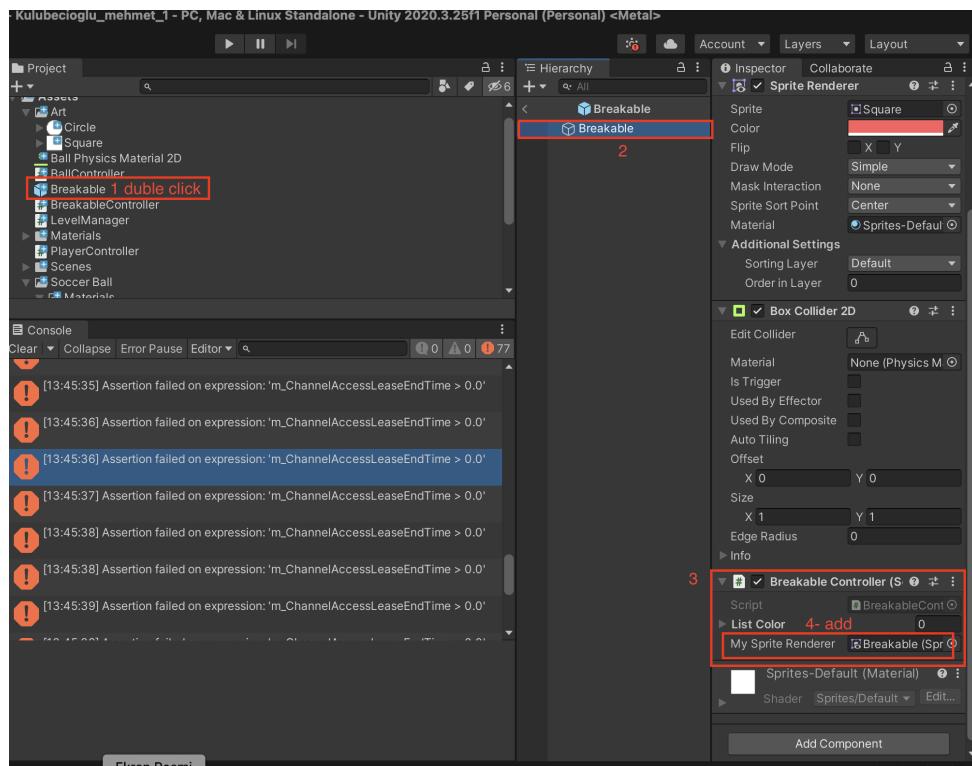
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

Users > mehmet > Kulubeciooglu_mehmet_1 > Assets > BreakableController.cs

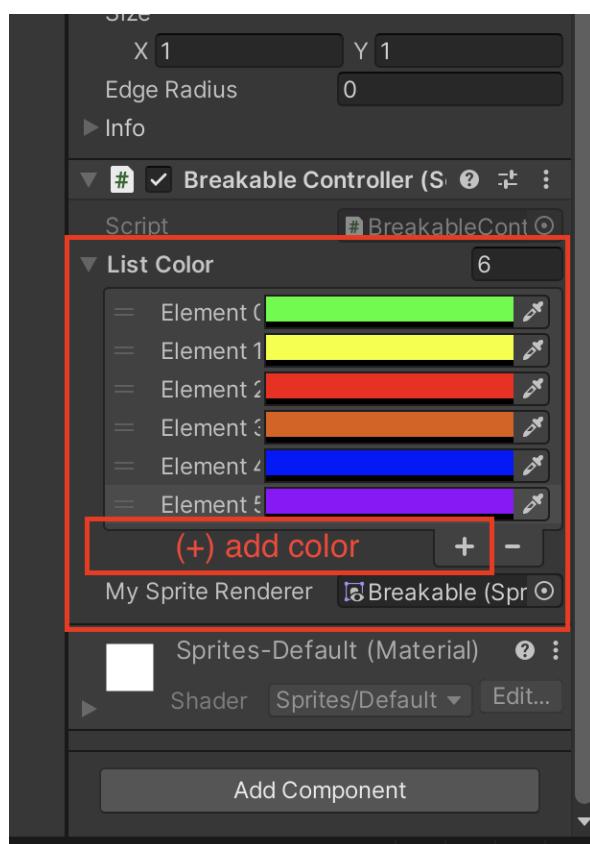
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class BreakableController : MonoBehaviour
6  {
7      [SerializeField] private List<Color> listColor;
8      [SerializeField] private SpriteRenderer mySpriteRenderer;
9
10
11     private void Start()
12     {
13         var index = Random.Range(0, listColor.Count);
14         var color = listColor[index];
15         mySpriteRenderer.color = color;
16     }
17 }
18 
```

add random color

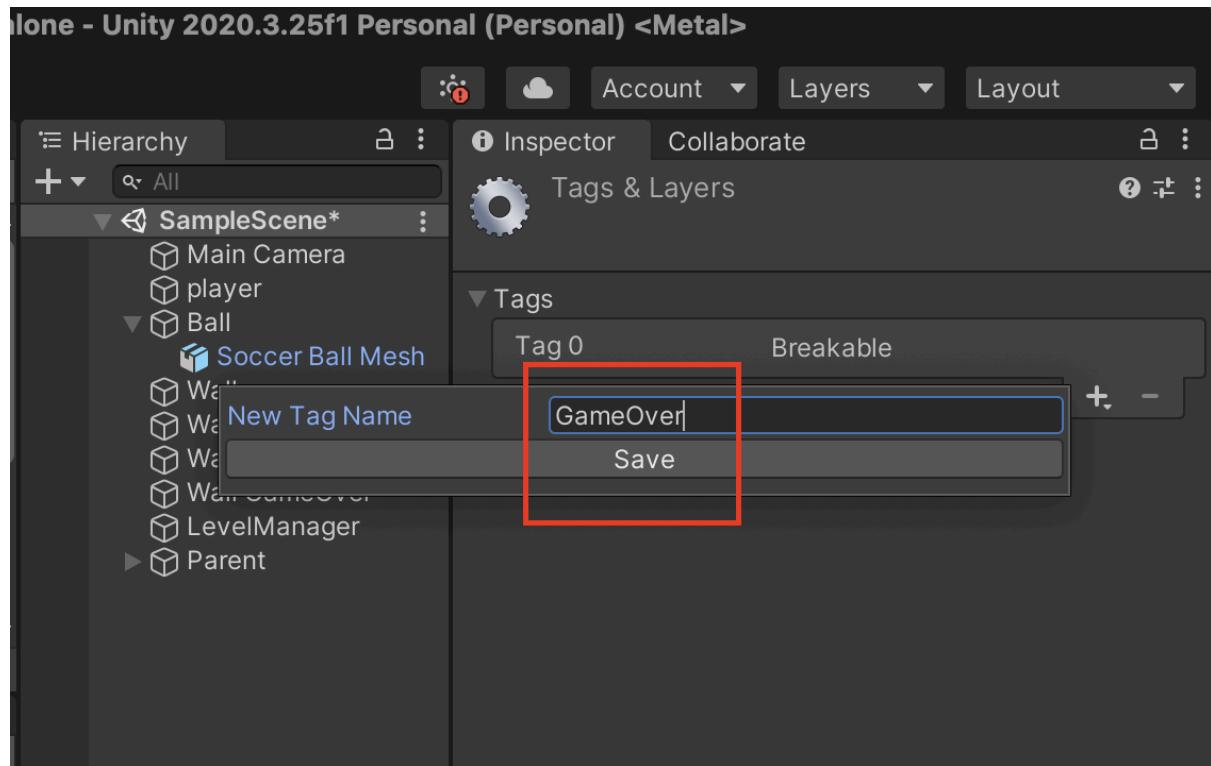
- I added the breakable by dragging it to the list color.



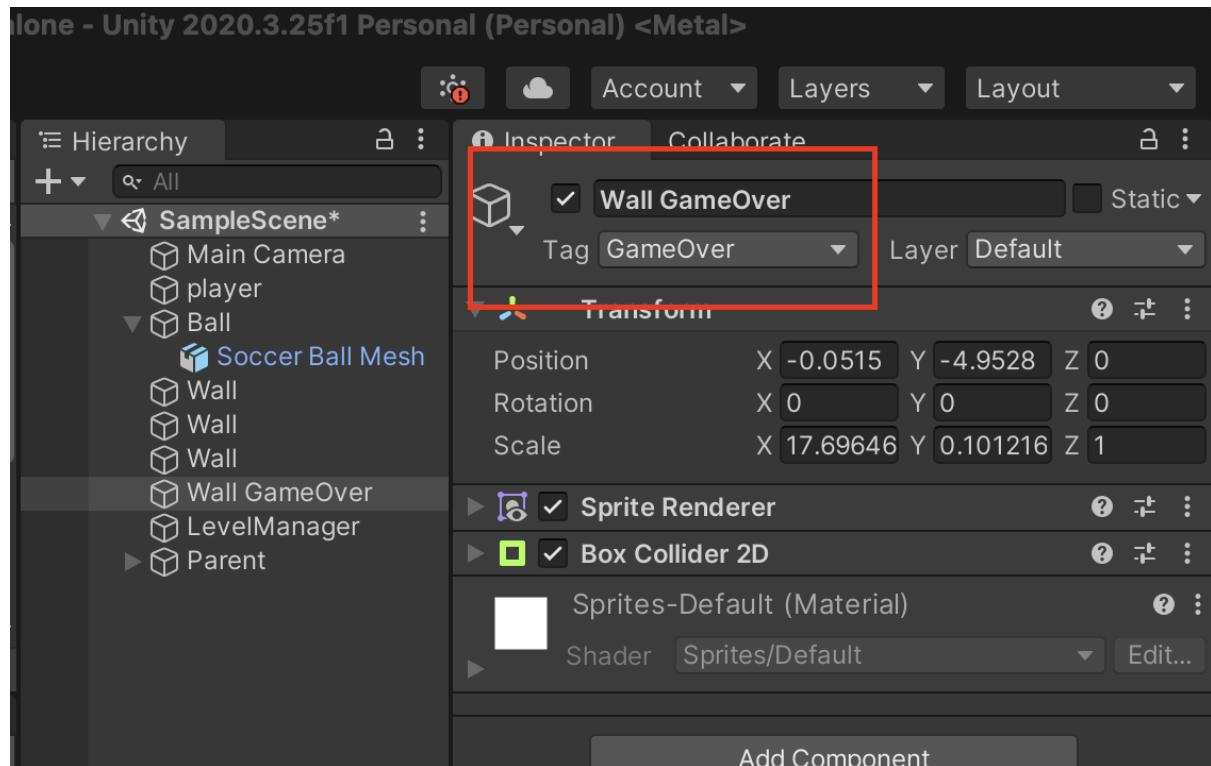
- I add 5 different color options for the code to place breakables in random colors and save it:



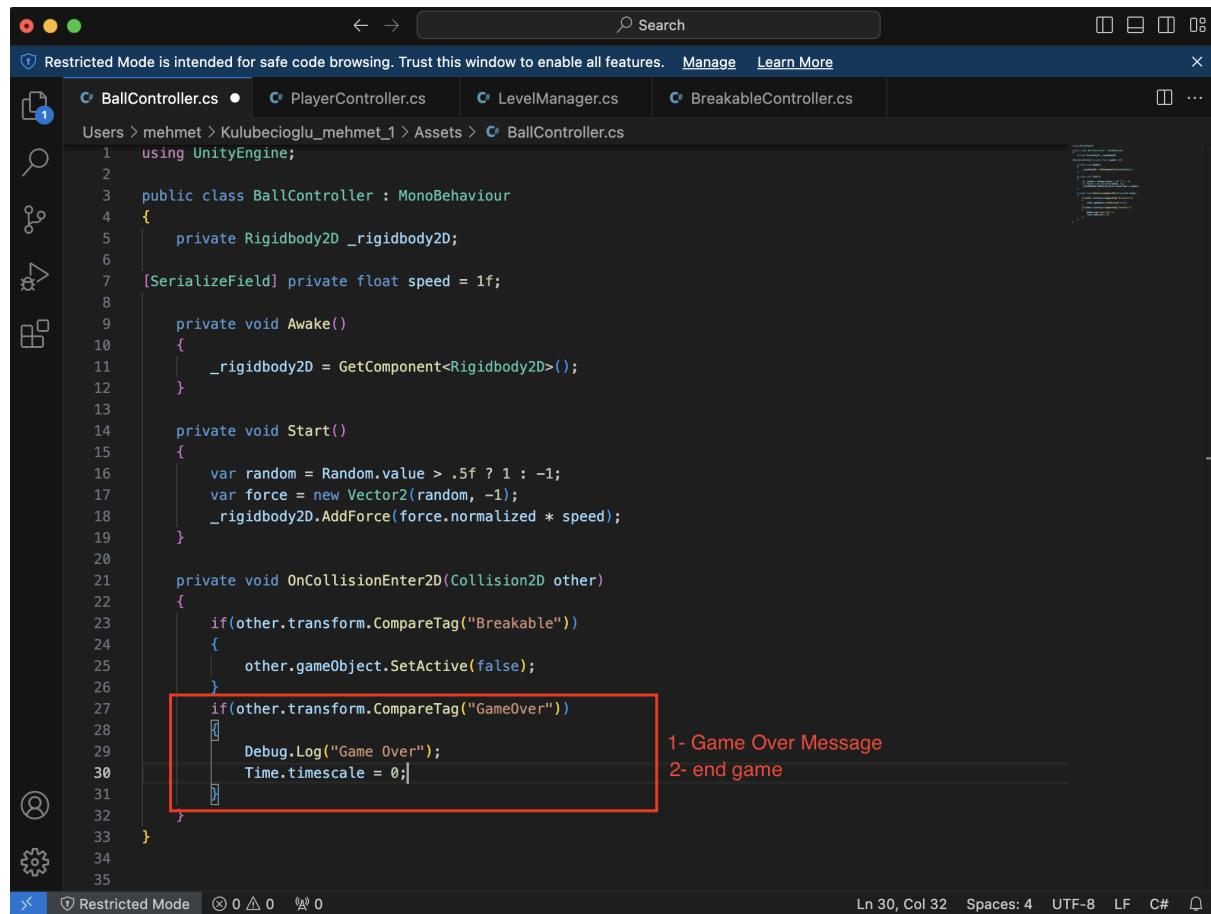
- Adding a game over tag so that the game ends when it hits the wall below:



- 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:

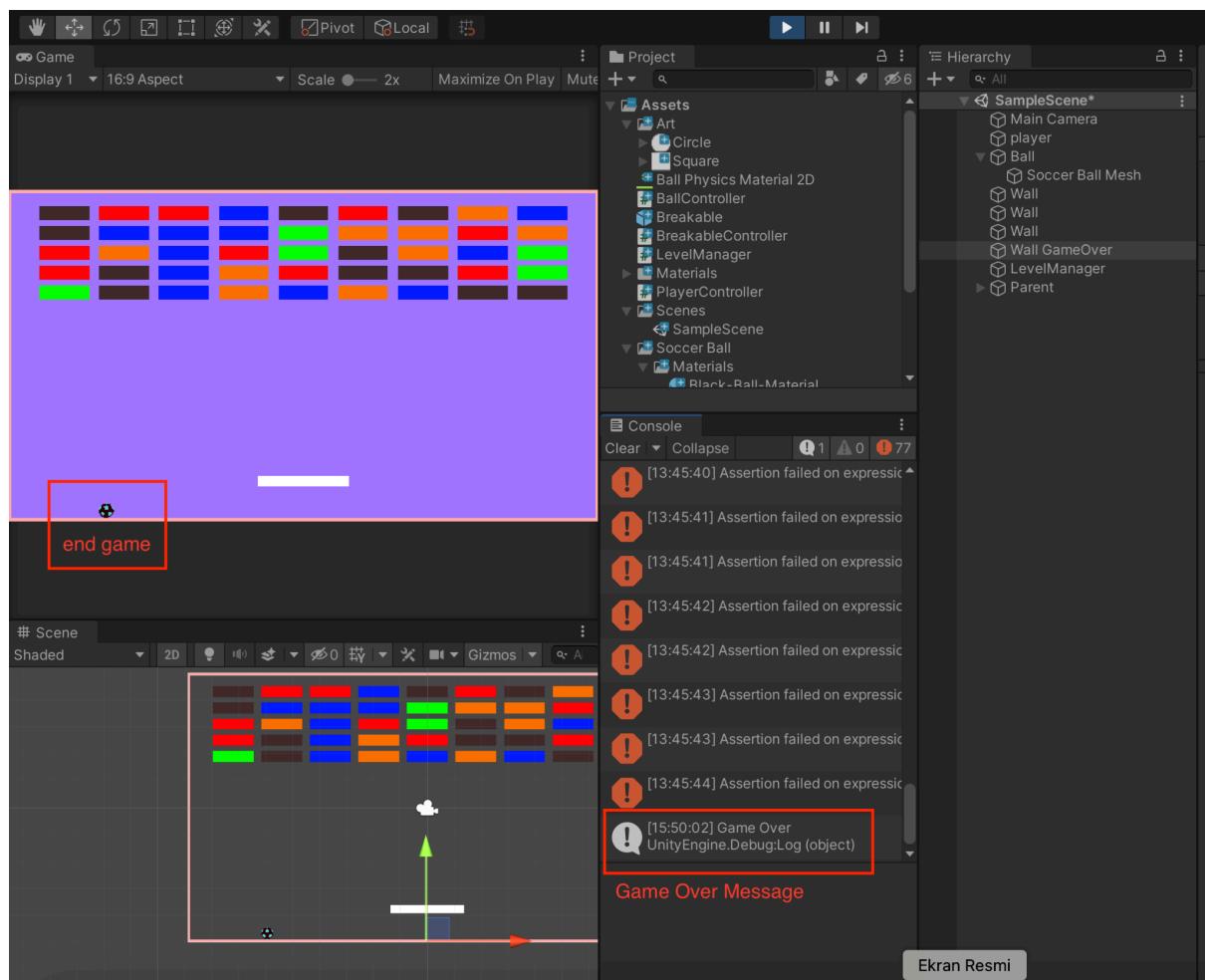


```

1  using UnityEngine;
2
3  public class BallController : MonoBehaviour
4  {
5      private Rigidbody2D _rigidbody2D;
6
7      [SerializeField] private float speed = 1f;
8
9      private void Awake()
10     {
11         _rigidbody2D = GetComponent<Rigidbody2D>();
12     }
13
14     private void Start()
15     {
16         var random = Random.value > .5f ? 1 : -1;
17         var force = new Vector2(random, -1);
18         _rigidbody2D.AddForce(force.normalized * speed);
19     }
20
21     private void OnCollisionEnter2D(Collision2D other)
22     {
23         if(other.transform.CompareTag("Breakable"))
24         {
25             other.gameObject.SetActive(false);
26         }
27         if(other.transform.CompareTag("GameOver"))
28         {
29             Debug.Log("Game Over");
30             Time.timeScale = 0;
31         }
32     }
33
34
35 }
```

1- Game Over Message  
2- end game

- 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:

```

1  using UnityEngine;
2  using System.Collections.Generic; // List kullanmak için gerekli
3
4  public class LevelManager : MonoBehaviour
5  {
6      [SerializeField] private GameObject breakablePrefab;
7      [SerializeField] private Vector2Int size;
8      [SerializeField] private Vector2 offset;
9
10     [SerializeField] private List<GameObject> breakables;
11
12     public bool CheckWin()
13     {
14         foreach (var item in breakables)
15             if (!item.activeSelf)
16                 return false;
17
18         return true;
19     }
20
21     [ContextMenu(nameof(Create))]
22     public void Create()
23     {
24         var parent = new GameObject { name = "Parent" };
25
26         var list = new List<GameObject>();
27         for (int i = 0; i < size.x; i++)
28         {
29             for (int j = 0; j < size.y; j++)
30             {
31                 var x = i * offset.x;
32                 var y = j * offset.y;
33                 var position = new Vector3(x, y, 0);
34                 var breakable = Instantiate(breakablePrefab, position, Quaternion.identity);
35                 list.Add(breakable);
36             }
37         }
38
39         Vector3 origin = Vector3.zero;
40         foreach (var item in list)
41             origin += item.transform.position; // Mekanik visuel ekleniyor

```

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

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. [Manage](#) [Learn More](#)

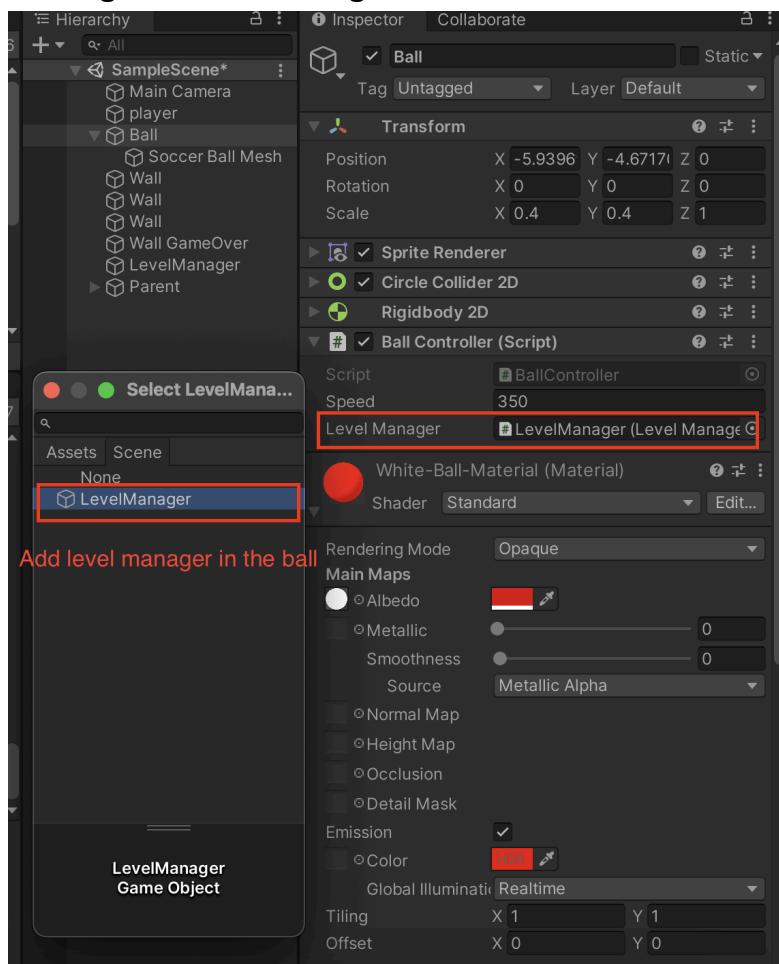
```

C BallController.cs • C PlayerController.cs • C LevelManager.cs • C BreakableController.cs
Users > mehmet > Kulubecioglu_mehmet_1 > Assets > C BallController.cs
1 using UnityEngine;
2
3 public class BallController : MonoBehaviour
4 {
5     private Rigidbody2D _rigidbody2D;
6
7     [SerializeField] private float speed = 1f;
8     [SerializeField] private LevelManager levelManager; add level manager
9
10    private void Awake()
11    {
12        _rigidbody2D = GetComponent<Rigidbody2D>();
13    }
14
15    private void Start()
16    {
17        var random = Random.value > .5f ? 1 : -1;
18        var force = new Vector2(random, -1);
19        _rigidbody2D.AddForce(force.normalized * speed);
20    }
21
22    private void OnCollisionEnter2D(Collision2D other)
23    {
24        if(other.transform.CompareTag("Breakable"))
25        {
26            other.gameObject.SetActive(false);
27            if (levelManager.CheckWin())
28            {
29                Debug.Log("You Win!");
30                Time.timeScale = 0; stopping time when blocks run out
31            }
32        }
33        if(other.transform.CompareTag("GameOver"))
34        {
35            Debug.Log("Game Over");
36            Time.timeScale = 0;
37        }
38    }
39 }
40

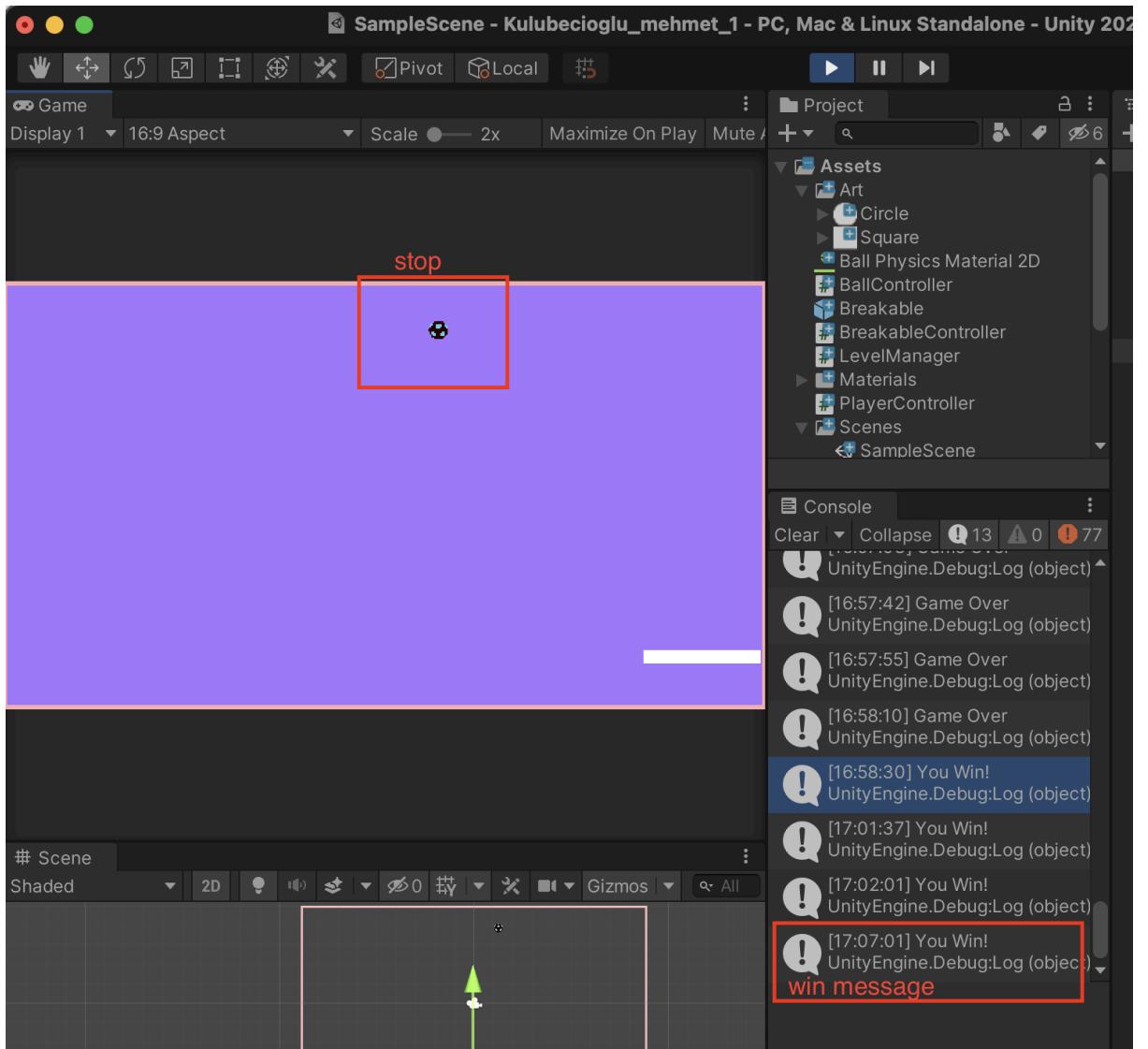
```

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.

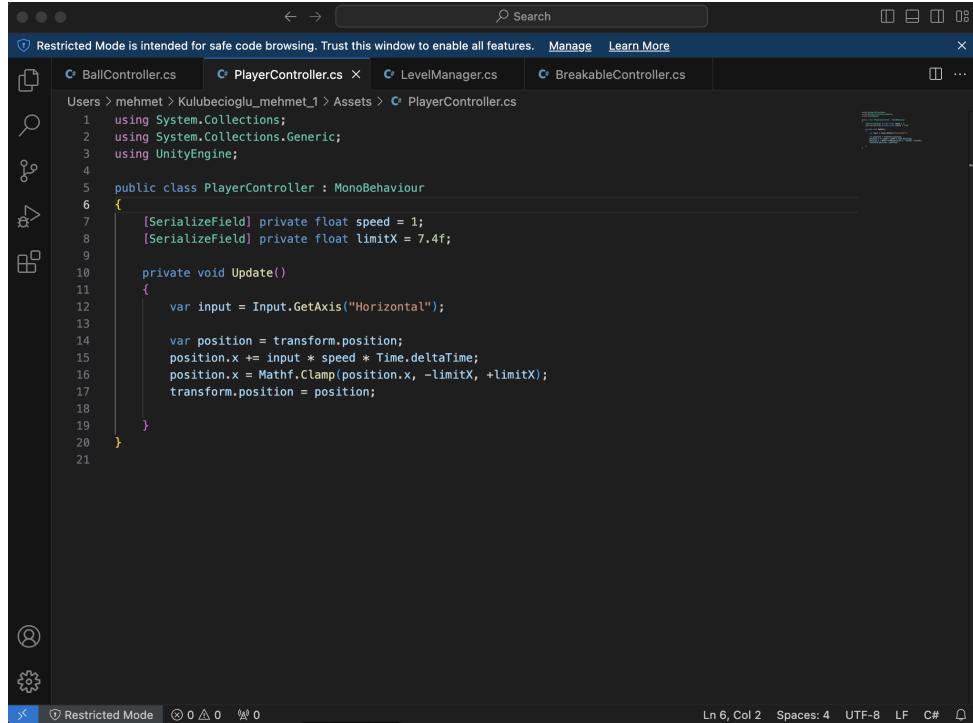


## Collision:

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.**

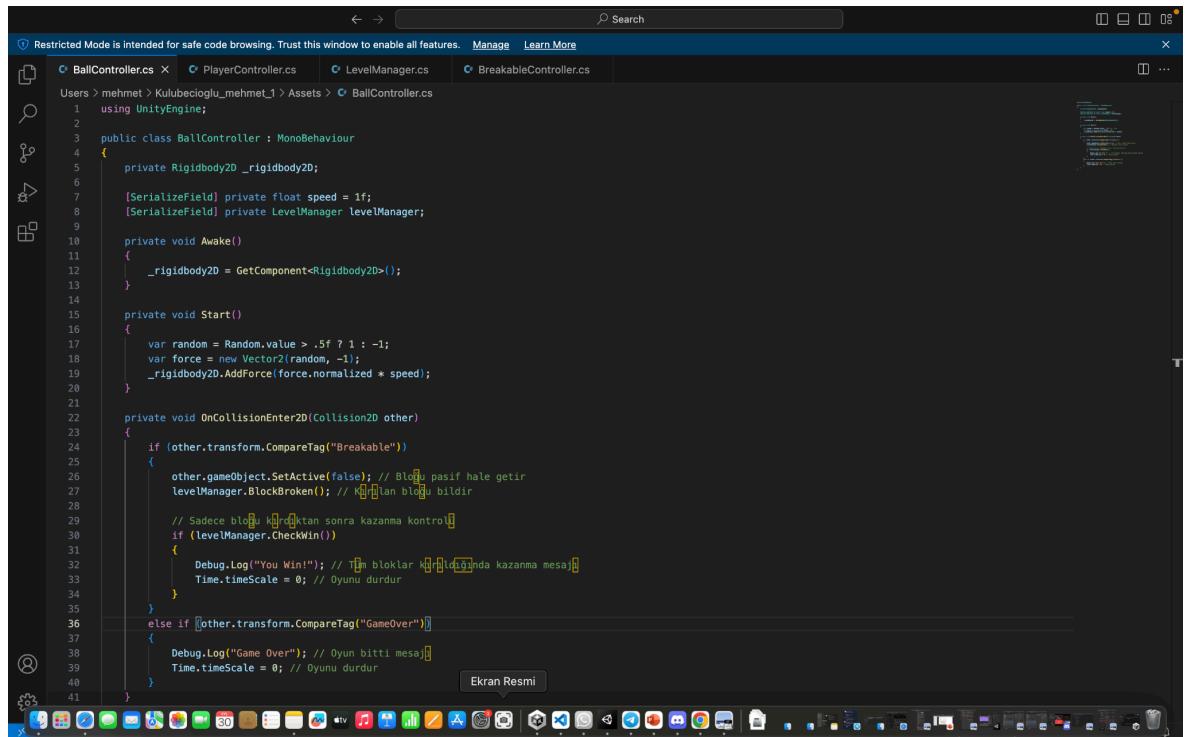
## PlayerController:



The screenshot shows the Unity Editor's code editor window with the PlayerController.cs script open. The script is a MonoBehaviour that moves a game object based on player input. It includes a private float speed and a private float limitX. The Update() method checks for horizontal input and moves the transform along the x-axis at a constant speed, while ensuring it stays within the limitX boundaries.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PlayerController : MonoBehaviour
6 {
7     [SerializeField] private float speed = 1;
8     [SerializeField] private float limitX = 7.4f;
9
10    private void Update()
11    {
12        var input = Input.GetAxis("Horizontal");
13
14        var position = transform.position;
15        position.x += input * speed * Time.deltaTime;
16        position.x = Mathf.Clamp(position.x, -limitX, +limitX);
17        transform.position = position;
18    }
19
20 }
21
```

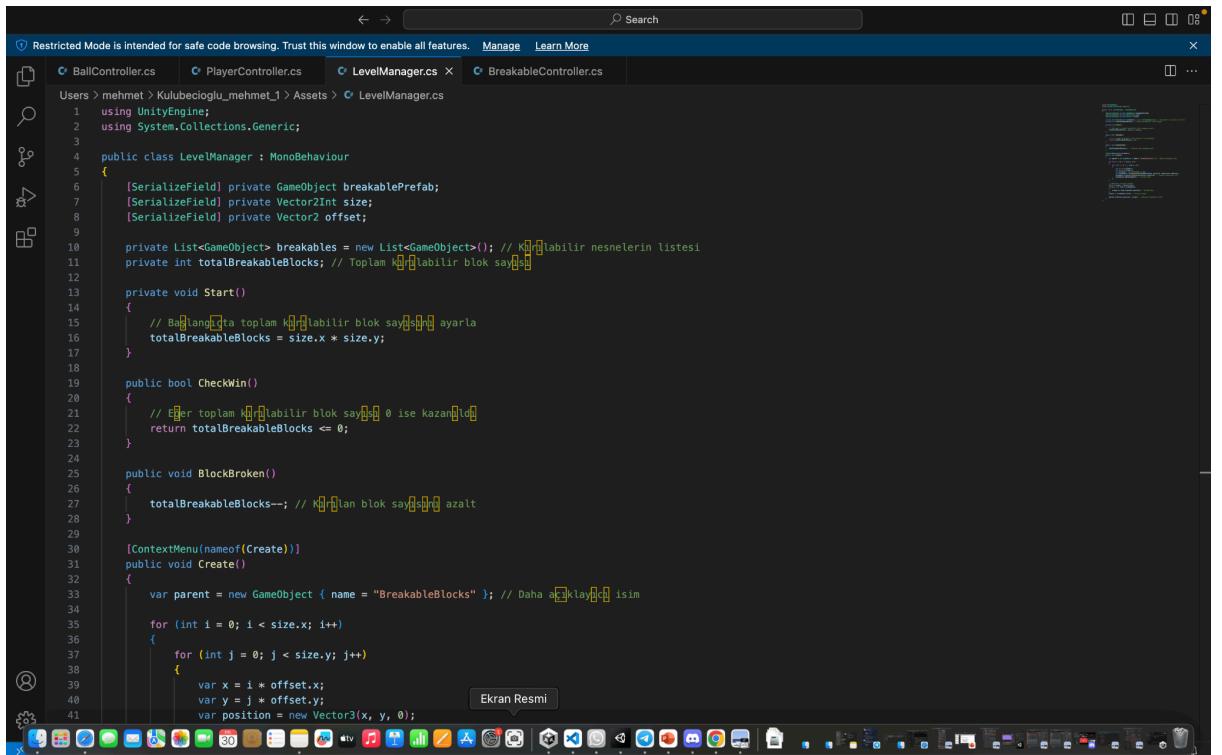
## BallController:



The screenshot shows the Unity Editor's code editor window with the BallController.cs script open. This script is a MonoBehaviour that controls a ball's movement and handles collisions. It uses a Rigidbody2D component to move the ball and checks for collisions with other objects. If a collision occurs with a "Breakable" object, it deactivates the object and increments a score. If the ball collides with a "GameOver" object, the game ends.

```
1 using UnityEngine;
2
3 public class BallController : MonoBehaviour
4 {
5     private Rigidbody2D _rigidbody2D;
6
7     [SerializeField] private float speed = 1f;
8     [SerializeField] private LevelManager levelManager;
9
10    private void Awake()
11    {
12        _rigidbody2D = GetComponent<Rigidbody2D>();
13    }
14
15    private void Start()
16    {
17        var random = Random.value > .5f ? 1 : -1;
18        var force = new Vector2(random, -1);
19        _rigidbody2D.AddForce(force.normalized * speed);
20    }
21
22    private void OnCollisionEnter2D(Collision2D other)
23    {
24        if (other.transform.CompareTag("Breakable"))
25        {
26            other.gameObject.SetActive(false); // Blok pasif hale getir
27            levelManager.BlockBroken(); // Blok打破 bildir
28
29            // Sadece blok打破tan sonra kazanma kontrolü
30            if (levelManager.CheckWin())
31            {
32                Debug.Log("You Win!"); // You Win! mesajı
33                Time.timeScale = 0; // Oyunu durdur
34            }
35        }
36        else if (other.transform.CompareTag("GameOver"))
37        {
38            Debug.Log("Game Over"); // Oyun bitti mesajı
39            Time.timeScale = 0; // Oyunu durdur
40        }
41    }
42}
```

## LevelManager:

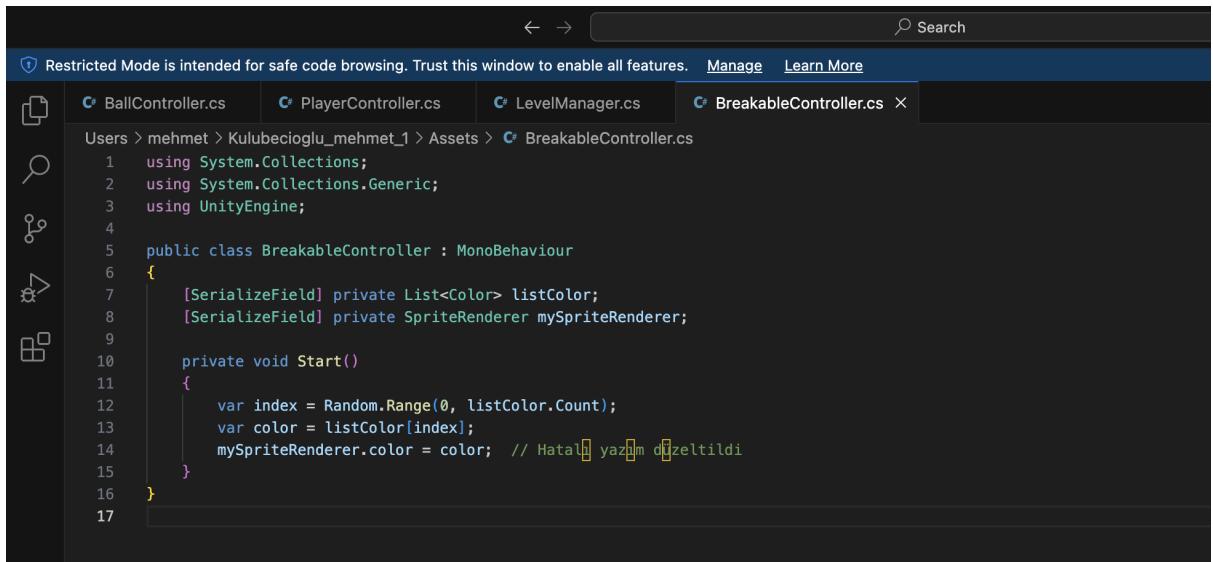


The screenshot shows the Unity Editor's code browser window. The current file is LevelManager.cs. The code defines a LevelManager class that inherits from MonoBehaviour. It contains fields for a breakable prefab, size, and offset. A list of breakables is initialized in the Start() method. The CheckWin() method returns true if all breakables have been destroyed. The BlockBroken() method decreases the totalBreakableBlocks count. The Create() method creates a grid of breakable blocks at the specified size and offset.

```
1 using UnityEngine;
2 using System.Collections.Generic;
3
4 public class LevelManager : MonoBehaviour
5 {
6     [SerializeField] private GameObject breakablePrefab;
7     [SerializeField] private Vector2Int size;
8     [SerializeField] private Vector2 offset;
9
10    private List<GameObject> breakables = new List<GameObject>(); // Kırılabilir nesnelerin listesi
11    private int totalBreakableBlocks; // Toplam kırılabilir blok sayısı
12
13    private void Start()
14    {
15        // Başlangıçta toplam kırılabilir blok sayısı 0 ayarla
16        totalBreakableBlocks = size.x * size.y;
17    }
18
19    public bool CheckWin()
20    {
21        // Eğer toplam kırılabilir blok sayısı 0 ise kazanıldı
22        return totalBreakableBlocks <= 0;
23    }
24
25    public void BlockBroken()
26    {
27        totalBreakableBlocks--;
28    }
29
30    [ContextMenu(nameof(Create))]
31    public void Create()
32    {
33        var parent = new GameObject { name = "BreakableBlocks" }; // Daha önceki kaydedilmiş isim
34
35        for (int i = 0; i < size.x; i++)
36        {
37            for (int j = 0; j < size.y; j++)
38            {
39                var x = i * offset.x;
40                var y = j * offset.y;
41                var position = new Vector3(x, y, 0);
42
43                var breakable = Instantiate(breakablePrefab, position, Quaternion.identity);
44                breakable.transform.SetParent(parent.transform); // Nesneyi ebeveynine ata
45                breakables.Add(breakable); // Listeye ekle
46            }
47
48            // Ebeveyinin ortasını hesapla
49            Vector3 origin = Vector3.zero;
50            foreach (var item in breakables)
51            {
52                origin += item.transform.position; // Ortaya ekle
53            }
54            origin /= breakables.Count; // Ortayı hesapla
55
56            parent.transform.position = origin; // Ebeveyinin konumunu ortala
57        }
58    }
59}
```

```
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
```

## BreakableController:



The screenshot shows a code editor interface for a Unity project. At the top, there are tabs for BallController.cs, PlayerController.cs, LevelManager.cs, and BreakableController.cs (which is currently selected). Below the tabs, a breadcrumb navigation path shows the file's location: Users > mehmet > Kulubecioglu\_mehmet\_1 > Assets > BreakableController.cs. The main area displays the C# code for the BreakableController class:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class BreakableController : MonoBehaviour
6  {
7      [SerializeField] private List<Color> listColor;
8      [SerializeField] private SpriteRenderer mySpriteRenderer;
9
10     private void Start()
11     {
12         var index = Random.Range(0, listColor.Count);
13         var color = listColor[index];
14         mySpriteRenderer.color = color; // Hatalı yazdırma düzeltildi
15     }
16 }
17
```

The code includes a note in the Start() method: "Hatalı yazdırma düzeltildi". The interface has a dark theme with light-colored text and icons.