

**DOKUZ EYLÜL UNIVERSITY**  
**FACULTY OF ENGINEERING**  
**DEPARTMENT OF COMPUTER ENGINEERING**

**SPLASH ZONE**  
**2D MOBILE PLATFORMER GAME**

**DOĞUKAN BERK ÖZER**

6.7.2021

## TABLE OF CONTENTS

	Page
Cover Page.....	1
Table of Contents.....	2
Table of Figures.....	4
Acknowledgements.....	7
Abstract.....	7
<b>CHAPTER 1 INTRODUCTION.....</b>	<b>8</b>
1.1. Background Information	
1.2. Game Definition	
1.3. Motivation / Related Works	
1.4. Goal / Contribution.....	9
1.5. Project Scope	
1.6. Methodology / Tools / Libraries.....	10
1.6.1. Tools	
1.6.2. Libraries	
<b>CHAPTER 2 LITERATURE REVIEW / IMPLEMENTATION 1.....</b>	<b>12</b>
<b>CHAPTER 3 REQUIREMENT ENGINEERING.....</b>	<b>14</b>
3.1. General Description	
3.2. Specific Requirements.....	15
<b>CHAPTER 4 DESIGN.....</b>	<b>17</b>
4.1. Product Design	
4.2. System Features	
4.3. Dependencies.....	18

<b>CHAPTER 5 IMPLEMENTATION 2</b>	19
5.1. Beginning Settings	
5.2. Screen Compatibility	
5.3. Camera Movement	21
5.4. Infinite Background	22
5.5. Character Animations	23
5.6. Platforms	27
5.7. Object Pooling	30
5.8. Leg Tag & Jump Limit	31
5.9. Menu Scene	33
5.10. High Score Scene	36
5.11. Controllers	37
5.12. Scoring System	39
5.13. Endgame	40
5.14. Highest Score	42
5.15. Finishing Touch	44
 <b>CHAPTER 6 TEST / EXPERIMENTS</b>	 46
 <b>CHAPTER 7 CONCLUSION, FUTURE WORK</b>	 48
7.1. Conclusion	
7.2. Future Work	
 <b>REFERENCES</b>	 49

## TABLE OF FIGURES

	Page
Figure 1.3. Newzoo 2020 global games market report.....	9
Figure 1.6.2. Unity Asset Store.....	11
Figure 2.1. Design phase of the background.....	12
Figure 2.2. Coloring background	
Figure 2.3. Designing platform	
Figure 2.4. Water animation.....	13
Figure 2.5. Player movement script	
Figure 3.1. General overview.....	14
Figure 3.2. Activity diagram.....	16
Figure 5.1. Beginning settings of the project in Unity.....	19
Figure 5.2.1. Screen resolution	
Figure 5.2.2. Screen compatibility script.....	20
Figure 5.2.3. Result of the screen script	
Figure 5.3.1. Camera movement script.....	21
Figure 5.3.2. Deficiency of the background.....	22
Figure 5.4.1. Infinite background	
Figure 5.4.2. Background movement script.....	23
Figure 5.5.1. Idle, jump and walk animations	
Figure 5.5.2. Sprites of the character animations.....	24
Figure 5.5.3. Optimizing sprites for animation	
Figure 5.5.4. Aforementioned animations.....	25
Figure 5.5.5. Ordered animations	
Figure 5.5.6. Player movement script.....	26
Figure 5.5.7. Character components.....	27
Figure 5.6.1. Platform package from Asset Store	
Figure 5.6.2. Editing box collider 2D of the character.....	28
Figure 5.6.3. Prefabs	

Figure 5.6.4. Platforms script.....	29
Figure 5.7.1. Platform pooling script.....	30
Figure 5.7.2. Space button functions	
Figure 5.7.3. Character variables.....	31
Figure 5.8.1. Leg tag and synchronized movement	
Figure 5.8.2. Empty leg tag object.....	32
Figure 5.8.3. Jump limit script	
Figure 5.8.4. Jump limit resets itself.....	33
Figure 5.9.1. Menu scene	
Figure 5.9.2. Menu scene objects.....	34
Figure 5.9.3. Animations of the menu scene objects	
Figure 5.9.4. Menu control script.....	35
Figure 5.9.5. On click events	
Figure 5.9.6. Touch up inside music buttons	
Figure 5.10.1. High score scene.....	36
Figure 5.10.2. Back to main menu button	
Figure 5.11.1. Jump button UI.....	37
Figure 5.11.2. Joystick UI	
Figure 5.11.3. Joystick button script.....	38
Figure 5.11.4. Jumping control script	
Figure 5.12.1. Scoring script.....	39
Figure 5.12.2. UI elements	
Figure 5.13.1. Keep the character in camera script.....	40
Figure 5.13.2. End game objects	
Figure 5.13.3. End game collision control.....	41
Figure 5.13.4. End game panel	
Figure 5.13.5. Game UI and game over panel.....	42
Figure 5.14.1. Highest score control	
Figure 5.14.2. Score text element on endgame panel.....	43

Figure 5.14.3. Highest score get/set methods	
Figure 5.14.4. Highest score scene.....	44
Figure 5.15.1. Eng game issues	
Figure 5.15.2. Game objects to finish game.....	45

## **ABSTRACT**

As a project, it was a 2D hyper-casual mobile platformer game. In this documentation, it was tried to explain step by step, starting with the introduction to the project, continuing with the requirements analysis and design phase, and then the implementation phase of the project.

After the implementation phase was completed, tests were made on the game and the experiences gained were mentioned. Finally, the experiences gained from the graduation project were compiled and the improvements that could be made after this step of the game were mentioned.

## **CHAPTER 1**

### **INTRODUCTION**

#### **1.1. Background Information**

The Splash Zone is a fast-paced platformer where you control the main character to reach the finish line with maximum points by dodging obstacles and collecting curative effects. The game ideation approach in the Splash Zone is a combination of some mobile games and the fantasy-science fiction stories. It is also an improved version of a very famous and simple game.

The key game idea is reaching the finish line by extending the flight time and scoring as high as possible without crashing to the floor. There will be obstacles to dodge and buffs to collect on the flight path. The player instantly tries to make the right choices in order to score more. At the end of the game, they will be able to purchase skin packs and products that will provide an in-game advantage with the earned points from the store.

## **1.2. Game Definition**

The game starts three seconds after the player presses the start button. The main goal of the player is to reach the finish line with maximum points by dodging obstacles and collecting curative effects. The player must also dodge the obstacles and collect the buffs on the flight path.

Since the Splash Zone is a hyper-casual game, it must be playable with one hand and hence the character needs just screen touches to fly but picks up buffs automatically. The player only decides whether to fly or not.

## **1.3. Motivation/Related Works**

The mobile game market is expected to grow more than the PC and Console market in the coming years. The reasons for these are the low initial cost for mobile games, less cost to develop than PC and console games, and the addition of new players to the system with the hyper-casual game approach.



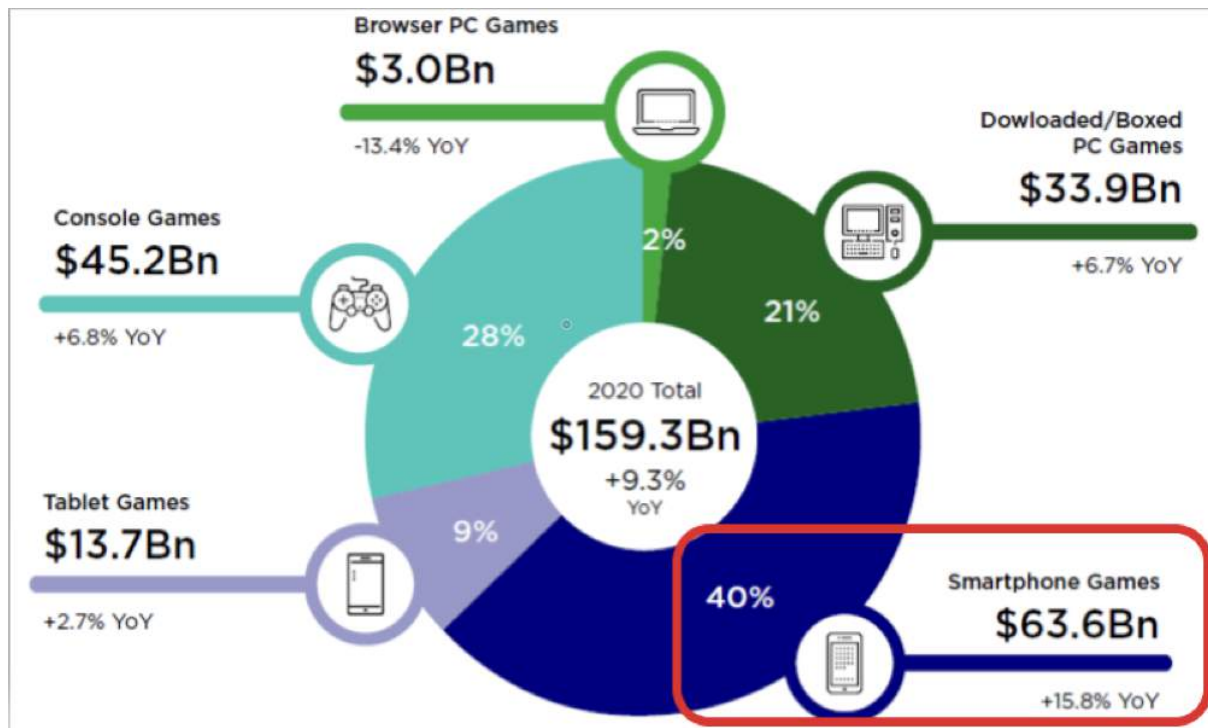


Figure 1.3. Newzoo 2020 global games market report [1].

- Mobile games are predicted to surpass the total PC and console game market.

#### 1.4. Goal/Contribution

Nowadays, people spend time in their homes, especially because of Covid-19. This situation made activities more popular that can be done at home. The games take place at the top of these activities. The reason for choosing mobile games is that smartphones are used more widely than PC and console platforms.

A mobile game was decided that people can easily play (hence hyper-casual) and make them happy with a sense of success. The main purpose underlying them is enabling a large community to play the game and to be successful.

#### 1.5. Project Scope

First of all, it is necessary to examine trending mobile games and analyze their good and bad sides. Then, it will evaluate the aspects that affect the player positively and blend them with the game idea. Necessary resource research will be done, libraries and tools will be analyzed that are deemed necessary and useful.

After the above steps, the design phase of the game is reached. In this section, the draft of the game is created. Game prototypes are created using physical and digital platforms. This is where the game takes its physical form.

The prototype stage is passed, now it's time to move on to the tools (the tool to be used in this project is Unity). In this section, a background is designed and created. As in this project, the background can be made up of more than one layer. While creating the background, attention should be paid to physics rules. If the physics rules are not taken into account, there may be situations such as the character falling from the map and disappearing.

After the background processes, the character part is handled. It is the object that the player can control. After the design, the character is placed on the background and the laws of physics come into play again. This section and the following are very important because most errors are encountered in these sections. The movement of character abilities and scales (such as running speed, jumping force), environmental factors (such as gravity) are carefully considered here. At the same time, their interactions with each other must be taken into account.

As the final stage of the design and implementation phase, the functional parts of the game are considered. The functional parts of this game are obstacles and strengthening agents on the map. These are obstacles that can lower or even kill the character's flight energy and on the other side, factors that increase the energy or speed up the character.

## **1.6. Tools/Libraries**

### **1.6.1. Tools**

More than 90% of all hyper-casual mobile games are created in Unity. Unity provides the bleeding-edge technology, monetization solutions, and live-ops services that I need to ensure success. Unity is not just a 3D platform; Unity is a complete platform for building beautiful and engaging 3D and 2D games. In fact, more 2D games are made with Unity than with any other game technology.

A comprehensive mobile-game solution, Unity's modular tools let me produce and deliver highly engaging 2D to players. And its powerful live-ops and monetization solutions ensure high visibility, rock-solid performance, and revenue growth. Unity enables me to deploy to all the major and emerging mobile operating systems, speed up the development process, optimize my game and achieve commercial success.

### 1.6.2. Libraries

Unity Asset Store : The Unity Asset Store is a growing library of Assets. Both Unity Technologies and members of the community create these Assets and publish them to the store. There are various types of Assets in the store ranging from textures, animations and models to entire Project examples, tutorials and Editor extensions [2].

Cinemachine : Cinemachine is a suite of tools for dynamic, smart cameras that let the best shots emerge based on scene composition and interaction, allowing you to tune, iterate, experiment and create camera behaviors in real-time.



Figure 1.6.2. Unity Asset Store

DOTween : Dotween is a fast, efficient, fully type-safe object-oriented animation engine for Unity, optimized for C# users, free and open-source, with tons of advanced features.

ProBuilder & ProGrids : ProGrids helps to build levels and greybox with ProBuilder by helping create proportional, clean geometry that we can easily resize and expand. ProGrids also provides excellent visual cues to help to extrude faces and edges evenly because the grid is in world space. Also can use the larger snap values to create the exterior walls, and then switch to smaller snap values to create insets or details.

## CHAPTER 2

### LITERATURE REVIEW / IMPLEMENTATION 1

Created the background and was placed on the zeroth layer.

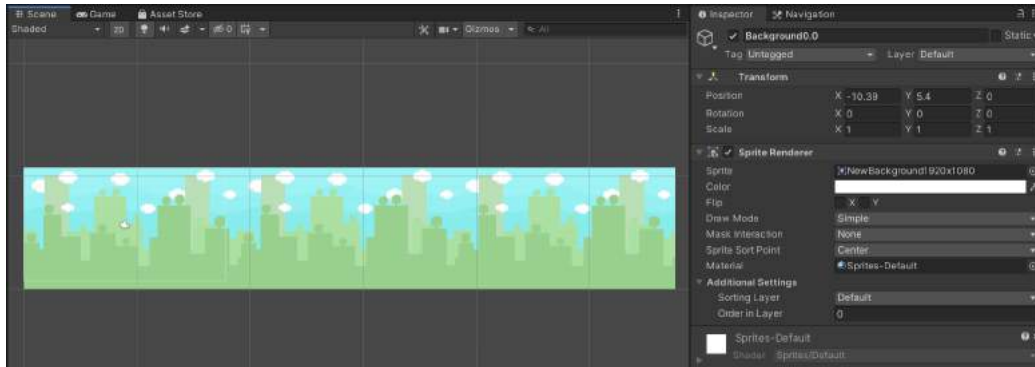


Figure 2.1. Background

Created the background to put in the first layer. And then, the process of coloring the backgrounds was done.

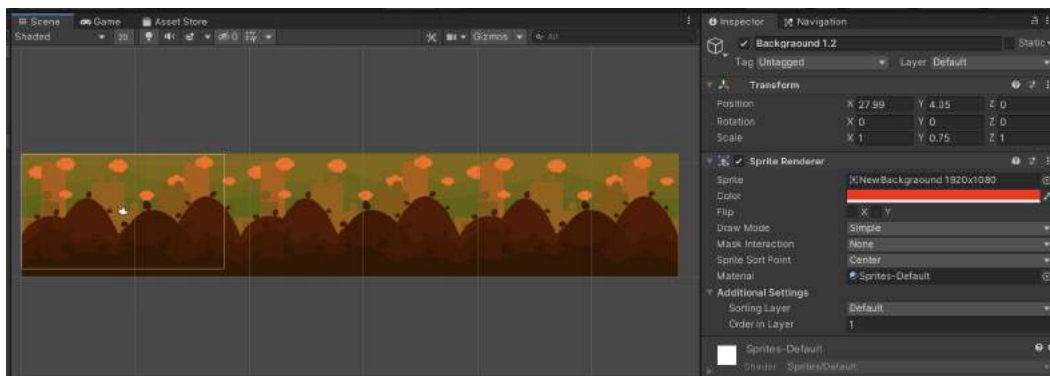


Figure 2.2. Coloring background

The platform was created into a second layer and physical properties of the platform were adjusted.

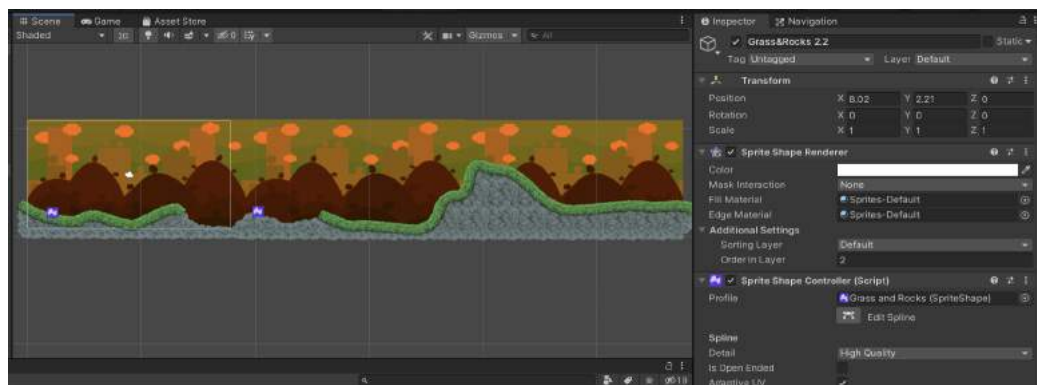


Figure 2.3. Designing platform

The watery zone was created and the water animation was successfully run.

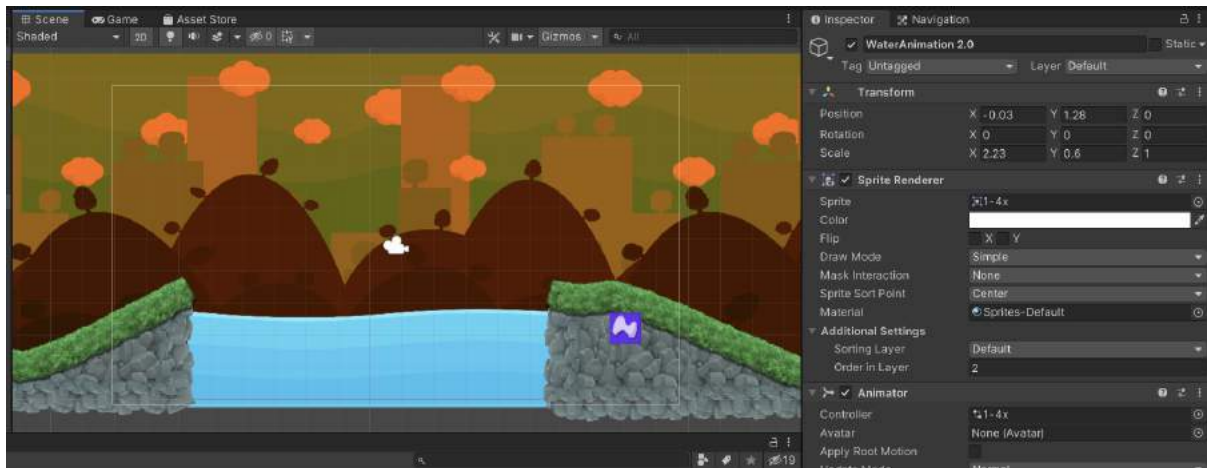


Figure 2.4. Water animation [3]

The character was created and related physical factors were placed partially on the character (such as gravity, running speed, jumping force—it will be flying later-).

```
PlayerMovement.cs
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class PlayerMovement : MonoBehaviour
6
7      public CharacterController2D controller;
8      public float runspeed = 40f;
9
10     float horizontalmove = 0f;
11
12     bool fly = false;
13
14     void Start()
15     {
16     }
17
18
19     void Update()
20     {
21         horizontalmove = Input.GetAxisRaw("Horizontal") * runspeed;
22
23         if(Input.GetButtonDown("Fly"))
24         {
25             fly = true;
26         }
27     }
28
29     void FixedUpdate()
30     {
31         controller.Move(horizontalmove * Time.fixedDeltaTime, false, fly);
32         fly = false;
33     }
34
35 }
```

Figure 2.5. Player movement script

## CHAPTER 3

### REQUIREMENT ENGINEERING

This chapter covers the requirements specification of Splash Zone. It includes general specification, specific requirements and analysis of models. Also includes changes management of the requirement specification.

#### 3.1 General Description

Developing a mobile game is a significant challenge. It requires considerable investments in terms of effort, client involvement and knowledge about the client market.

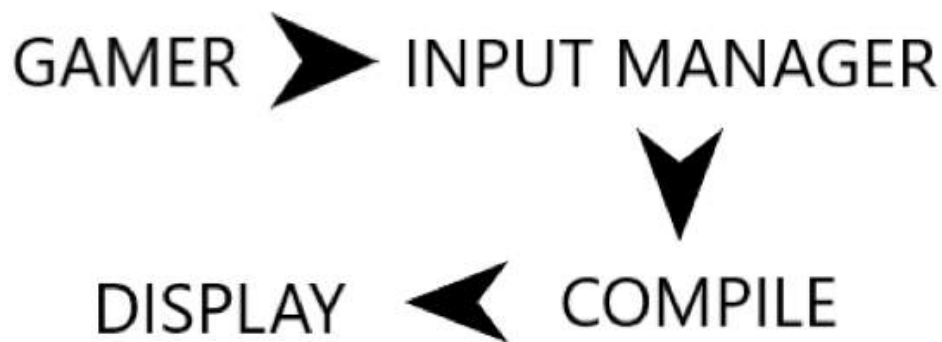


Figure 3.1. General overview

Gamers interact with the system by giving input and the system gives these inputs to script, if any change occurs, objects are sent to renders to display changes.

Requirements :

- User friendly efficient system.
- Easy to operate.
- Measured coding and professional thinking.
- Max. high definition.
- Min. hardware requirements. But these must be relevant for the game.
- Design system with efficient manner.
- Easy to update.
- Maximum high regulation with minimum hardware.

Splash Zone is a 2D hyper-casual multi-platform game which is supported by IOS and Android smartphones.

After running the game, the UI of the game will appear on the screen. It is used to explain all aspects of experience with the system. Then, the gamer can select start from the main menu and turn sound settings. Gamers also interact with pause, restart and exit by pressing.

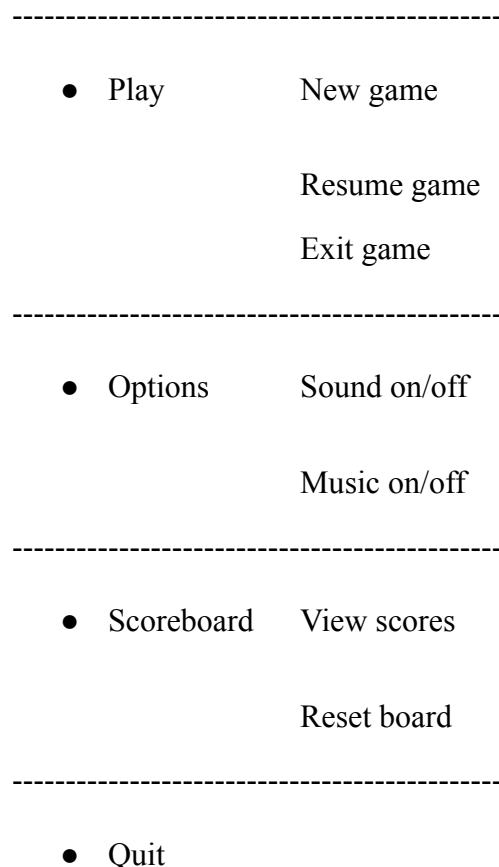
### 3.2 Specific Requirements

Every game must have a menu so it can be user friendly enough and gamer can easily fulfill their needs.

Splash Zone will be a 2d mobile platformer gaming application designed specially for IOS and Android platform and it's functional on both smartphones and tablets. Gaming app data will be stored locally on the game engine elements.

There is only one user at unit time in the game. So, a gamer is only one who communicates with the system through playing games and the gamer can be any person.

#### Use Case Diagram



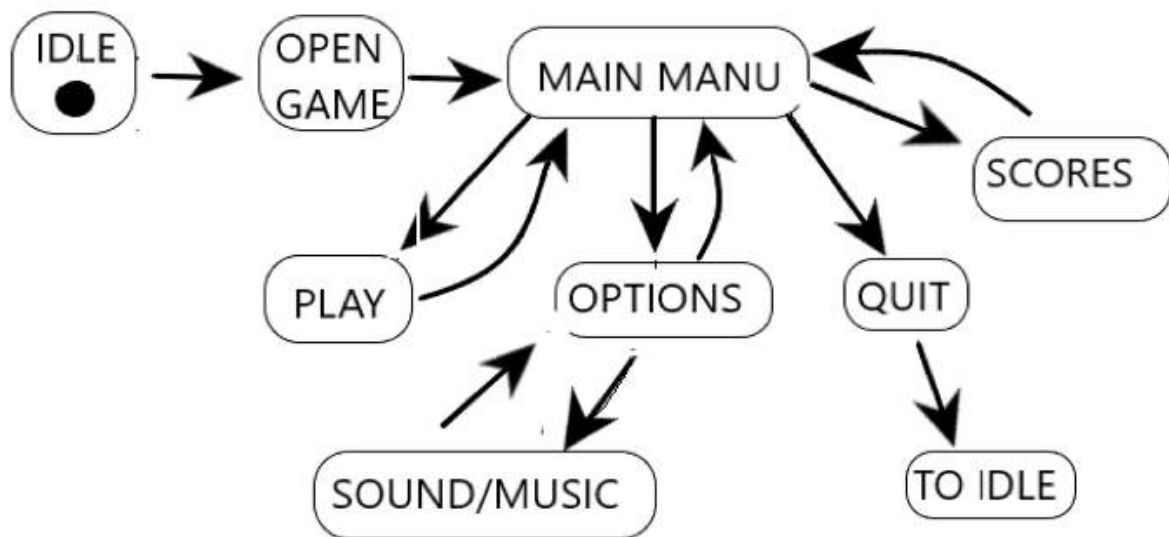


Figure 3.2. Activity diagram



## **CHAPTER 4**

### **DESIGN**

#### **4.1 Product Design**

- User Experience : To avoid unnecessary product features, must simplify design documentation. UX design is any aspect of user experience with a given system including UI, graphics, industrial design and physical interaction.

UE design fully encompasses traditional human-computer interaction design and extends it by addressing all aspects of product or service by users.

- Backend Programming : This is code supporting that responsible for DB access, business logic etc. In simple terms, app frontend is what you see but app backend is an app engine that the user does not see.

For efficient implementation, increasing user acceptance are both very important in software.

#### **4.2 System Features of Splash Zone**

- Camera is added to the game platform in the game scene. The player needs to be very careful so that the character may crash into obstacles which may cause the game to end.

Camera must consistently update the position of the character and if flying energy runs out, the game is finished.

- Title screen is the screen the player will see every time upon playing the game. Through the interface, players can choose to start a new game or adjust options.

It must load and appear every time after the game is launched. If players quit the game, they must be returned to the title screen. If players press exit, the game will end and return to the regular interface of the phone. If players complete the game, the game will end before returning to the title screen.

- Pause menu must appear anytime during game play. It also allows the player to navigate between game play and the title screen.

Return button must continue the game without any change to character or state from the pause action moment.

- Option menu is accessible on the title screen to allow the player to configure controls. The screen isn't essential for game-play.

Sound/music will be enabled when "on" selected and disabled when "off" selected.

- Flying is the main function in the game. The player must reach the finish line, preserving time and energy. If the energy runs out, it crashes to the ground and the game is over.
- Rocket is a buff in the game. If the player can reach the rocket, the player will go more than they would at the same time.
- Wind can be buff or ill effect. If it blows from the front, the character's flight speed decreases for a certain period of time, but if it blows from behind, the character's flight speed increases.
- Obstacles are ill to character. During the game, obstacles must be avoided. Contact with obstacles can reduce the energy of flight or even crash and end the game.
- Finish line is the finishing place of the game. The player needs to arrive at the line before energy ends.
- Timer will be in the scene for keeping track of time the player takes to the finish level. This is used to calculate the score of a player in a section. When less time is spent, more score is earned.

Timer starts automatically at game begin and it stops after player finishes the game.

### **4.3 Dependencies**

Final destination of the game operation will be IOS and Android mobile devices and Unity will be responsible for both.

Unity includes many built-in components and these will expedite the process of the game development immensely. These components are physics engine, collision detection and handling, input recognition, object creation and manipulation, model attachment etc.

Build setting of Unity simplifies the process of transferring the game to IOS-Android devices. After complete game development or during any step to test, can select the OS and build the game to one of the devices.

## CHAPTER 5

### IMPLEMENTATION 2

#### 5.1. Beginning Settings

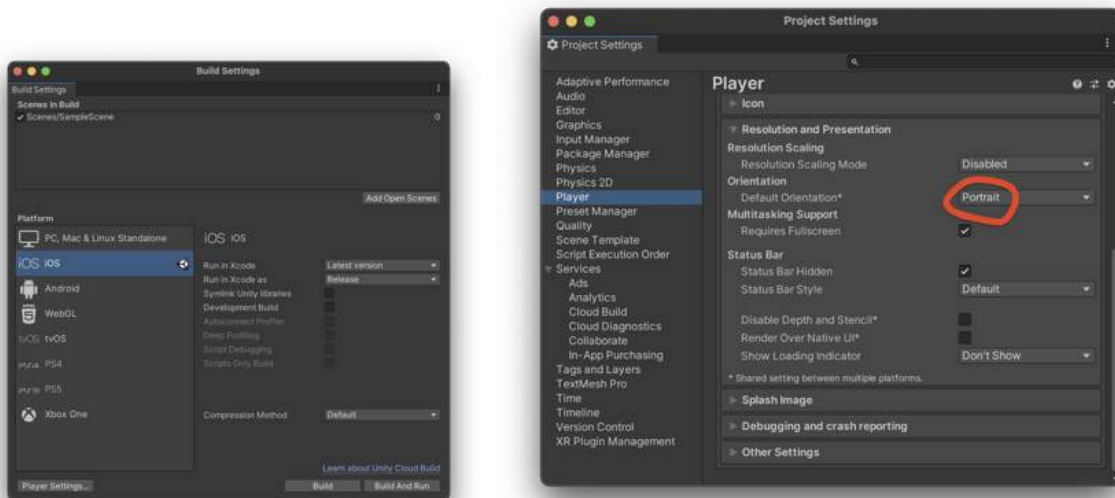


Figure 5.1. Beginning settings of the project in Unity

After creating the game project, iOS is chosen as a platform from the build settings window. And then in the project settings, the portrait is chosen as default orientation. Because the mobile game will be played on a vertical screen.

#### 5.2. Screen Compatibility

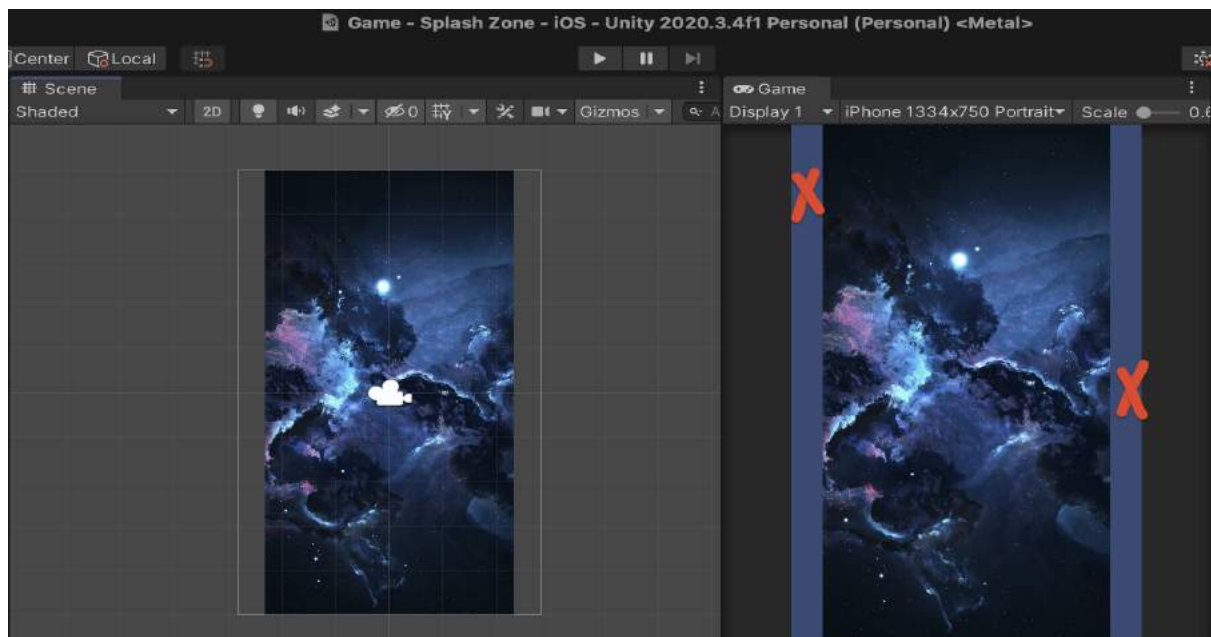


Figure 5.2.1. Screen resolution

As seen in the screenshot above, when the screen resolution changes, the spaces in the background are automatically filled with blue space. The following script was created to make the background compatible with all screen resolutions.

```
FullScreen.cs
Seçim yok
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class FullScreen : MonoBehaviour
6  {
7      // Start is called before the first frame update
8      void Start()
9      {
10         SpriteRenderer spriteRenderer = GetComponent<SpriteRenderer>(); // to get reference
11         Vector2 tempScale = transform.localScale; // reference of scale value
12
13         float spriteWidth = spriteRenderer.size.x; // length of the sprite on x axis
14         float screenHeight = Camera.main.orthographicSize * 2.0f; // height of the screen
15         float screenWidth = screenHeight / Screen.height * Screen.width; // width of the screen
16
17         tempScale.x = screenWidth / spriteWidth; // to apply scaling process
18         transform.localScale = tempScale;
19     }
20
21     // Update is called once per frame
22     void Update()
23     {
24     }
25 }
26
27
```

Figure 5.2.2. Screen compatibility script

SpriteRenderer renders a sprite for 2D graphics [4]. Vector2 is a representation of 2D vectors and points. This structure is used to update the 2D position of the background. By the way, the Start function is called once before the first frame. But the Update function is called once per frame in the game. After adding the script as a component to the background object, the screen is compatible with all screen resolutions.

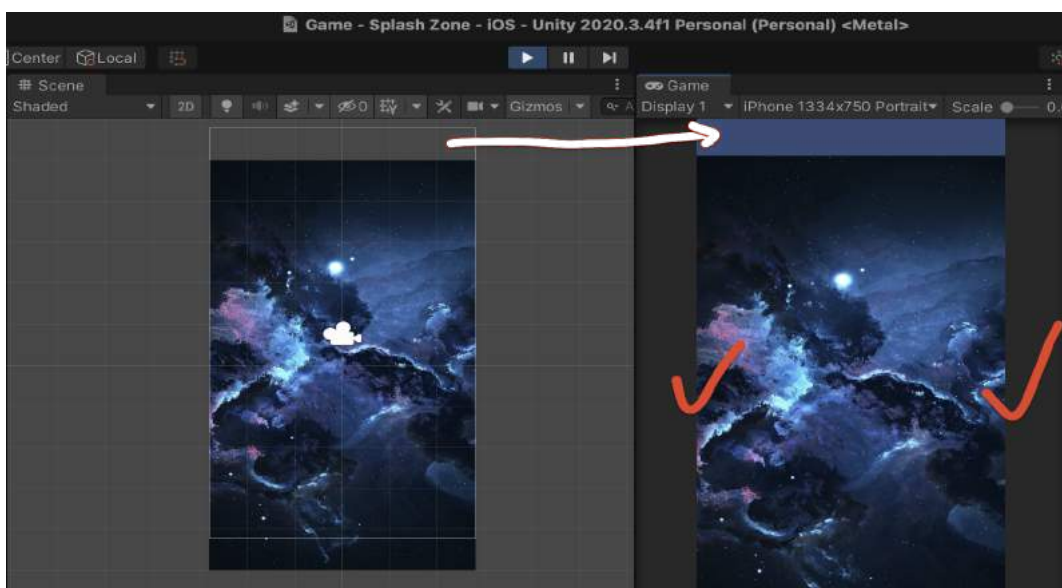
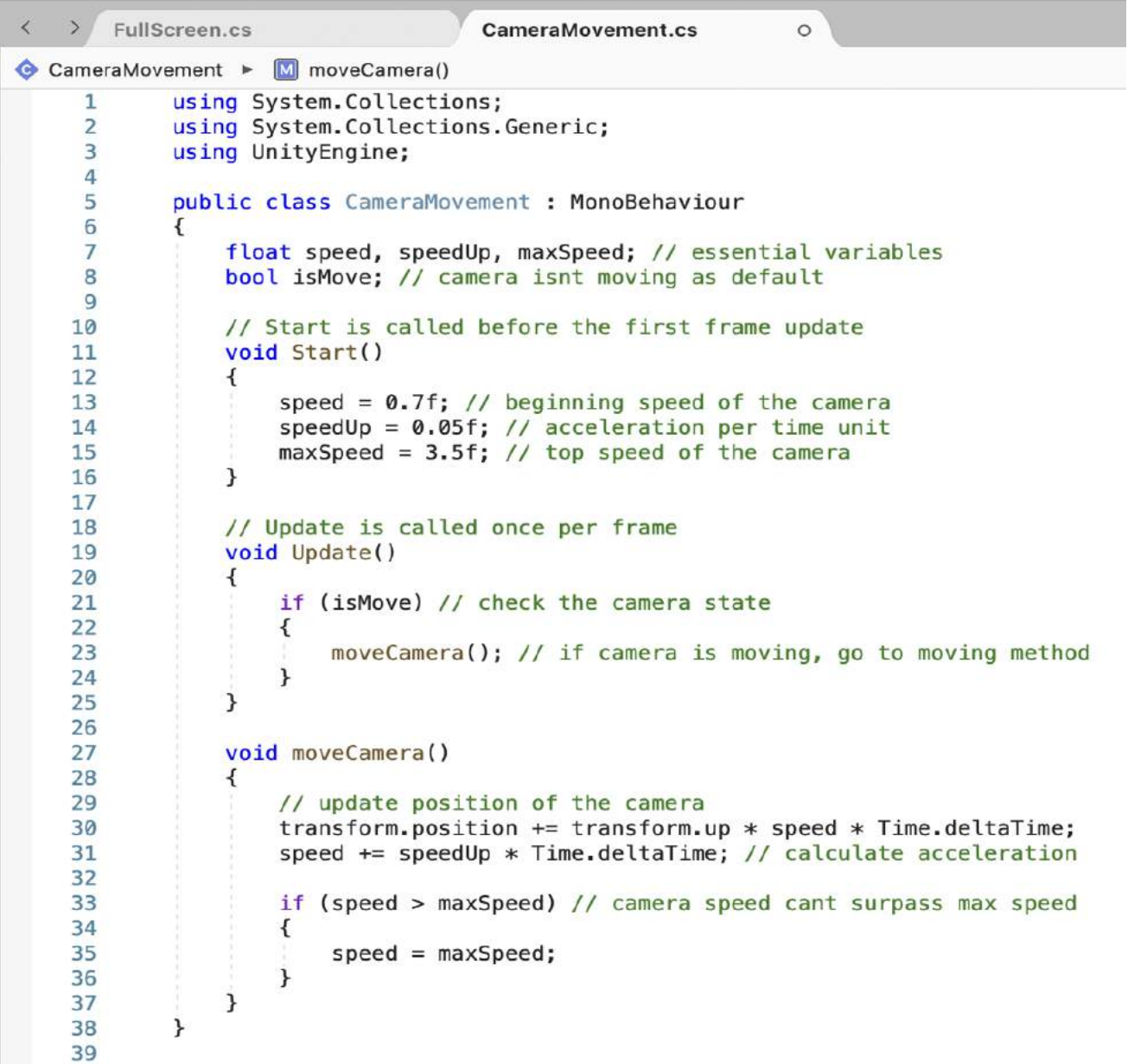


Figure 5.2.3. Result of the screen script

### 5.3. Camera Movement

It's time for the camera movement. The script below allows the camera to accelerate upward in the Y axis continuously [5]. The camera has a certain initial speed. As time progresses, the movement of the camera in the y-axis accelerates to increase the difficulty level. However, this speed should not exceed a certain limit. Otherwise, it will be impossible for the player to continue after a certain point.

The image is a screenshot of a code editor window. At the top, there are two tabs: 'FullScreen.cs' and 'CameraMovement.cs'. The 'CameraMovement.cs' tab is active. Below the tabs, the script name 'CameraMovement' is shown with a dropdown arrow, and a method 'moveCamera()' is selected. The script code is as follows:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class CameraMovement : MonoBehaviour
6  {
7      float speed, speedUp, maxSpeed; // essential variables
8      bool isMove; // camera isnt moving as default
9
10     // Start is called before the first frame update
11     void Start()
12     {
13         speed = 0.7f; // beginning speed of the camera
14         speedUp = 0.05f; // acceleration per time unit
15         maxSpeed = 3.5f; // top speed of the camera
16     }
17
18     // Update is called once per frame
19     void Update()
20     {
21         if (isMove) // check the camera state
22         {
23             moveCamera(); // if camera is moving, go to moving method
24         }
25     }
26
27     void moveCamera()
28     {
29         // update position of the camera
30         transform.position += transform.up * speed * Time.deltaTime;
31         speed += speedUp * Time.deltaTime; // calculate acceleration
32
33         if (speed > maxSpeed) // camera speed cant surpass max speed
34         {
35             speed = maxSpeed;
36         }
37     }
38 }
39
```

Figure 5.3.1. Camera movement script

The camera object is working correctly. Next up, as can be seen from the screenshot below, although the camera moves, the background is fixed and not synchronized with the camera. Next is the infinite background task.

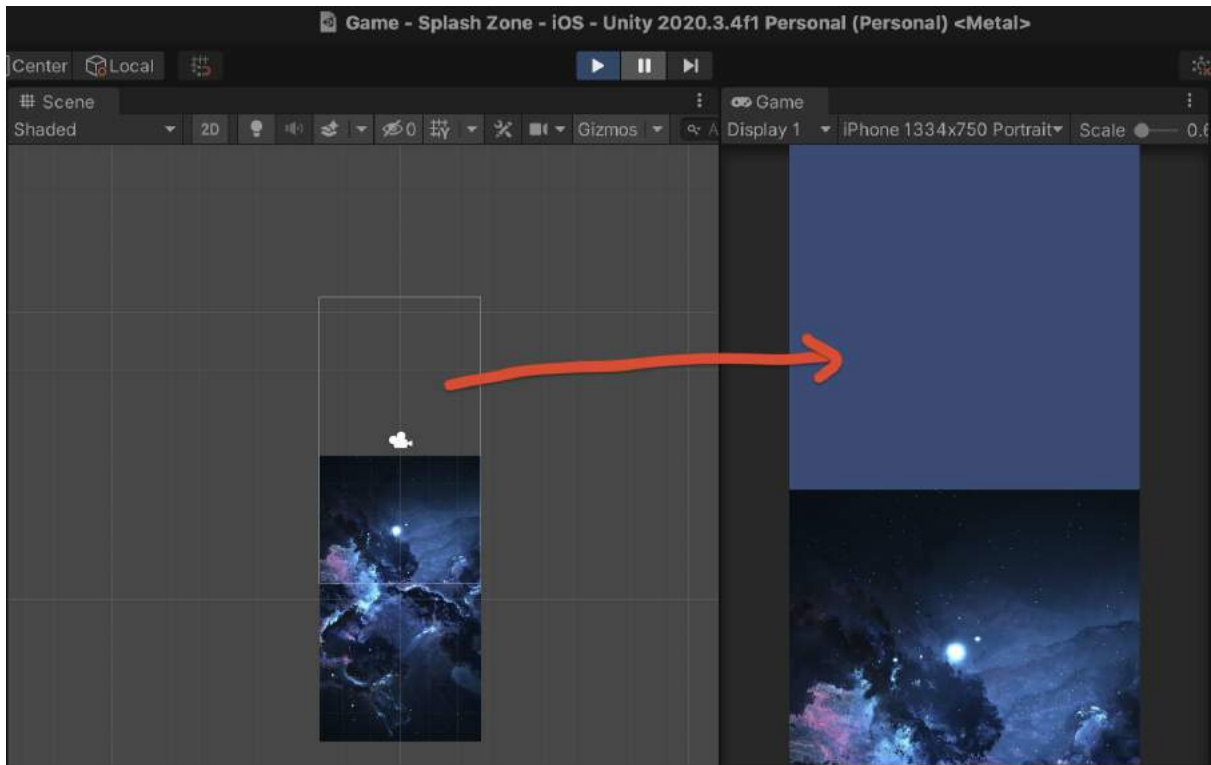


Figure 5.3.2. Deficiency of the background

#### 5.4. Infinite Background

Two background objects are needed to create an infinite background. Backgrounds are placed upside down. As soon as the background below is out of the camera, the background below is placed on top of the other background. This process continues continuously and thus an infinite background is created.

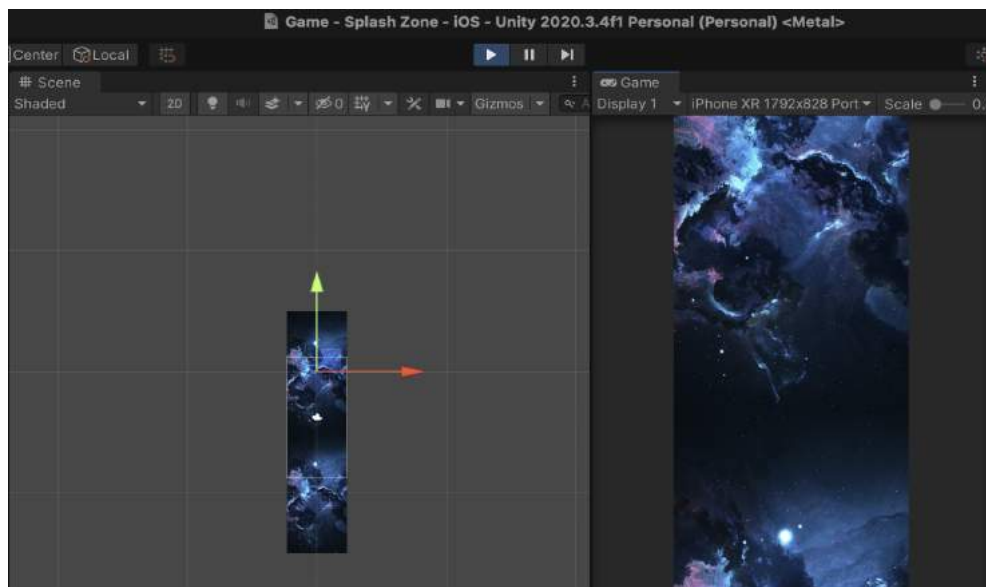
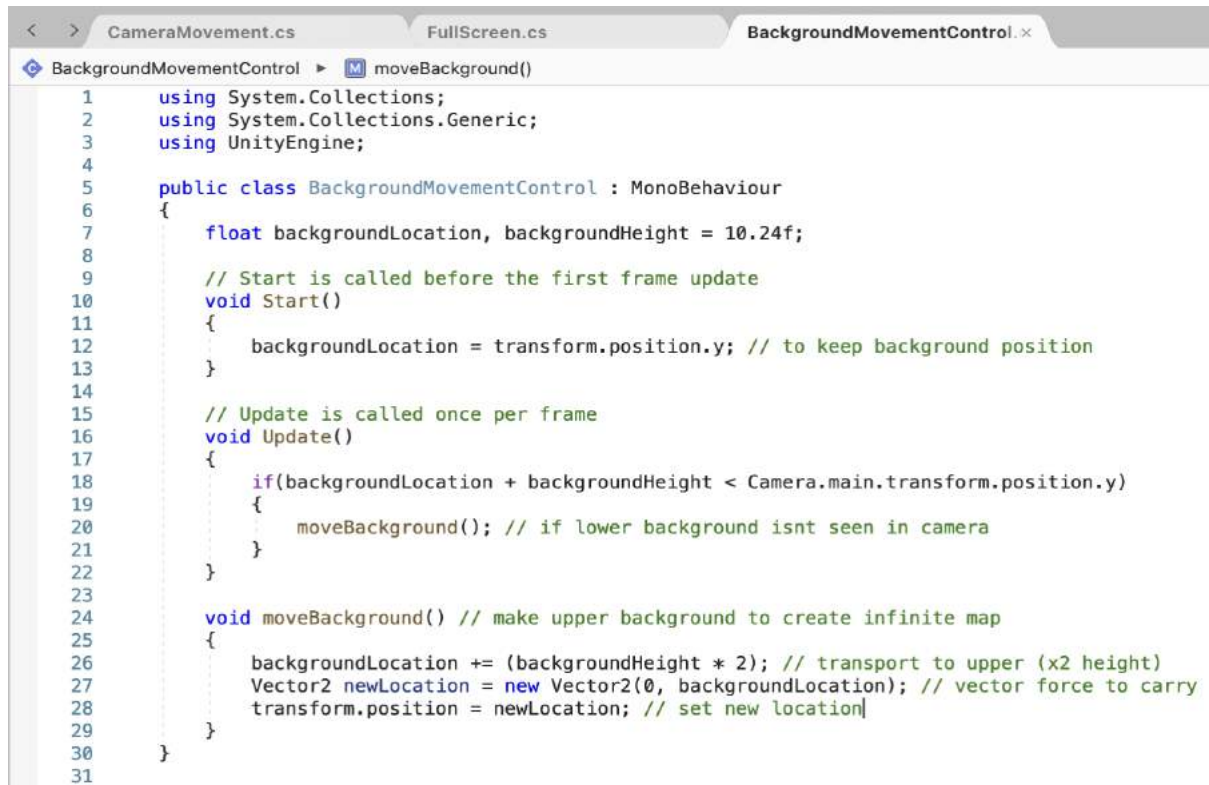


Figure 5.4.1. Infinite background





```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class BackgroundMovementControl : MonoBehaviour
6  {
7      float backgroundLocation, backgroundHeight = 10.24f;
8
9      // Start is called before the first frame update
10     void Start()
11     {
12         backgroundLocation = transform.position.y; // to keep background position
13     }
14
15     // Update is called once per frame
16     void Update()
17     {
18         if(backgroundLocation + backgroundHeight < Camera.main.transform.position.y)
19         {
20             moveBackground(); // if lower background isnt seen in camera
21         }
22     }
23
24     void moveBackground() // make upper background to create infinite map
25     {
26         backgroundLocation += (backgroundHeight * 2); // transport to upper (x2 height)
27         Vector2 newLocation = new Vector2(0, backgroundLocation); // vector force to carry
28         transform.position = newLocation; // set new location
29     }
30 }
31

```

Figure 5.4.2. Background movement script

As can be seen in the related script, the length of the background image is 10.24. In the start method, the location of the background is taken and the background function works if the condition is satisfied after the camera control is done. It is moved up at the y minus twice the height of the background. So the background below is now the new top background.

## 5.5. Character Animations

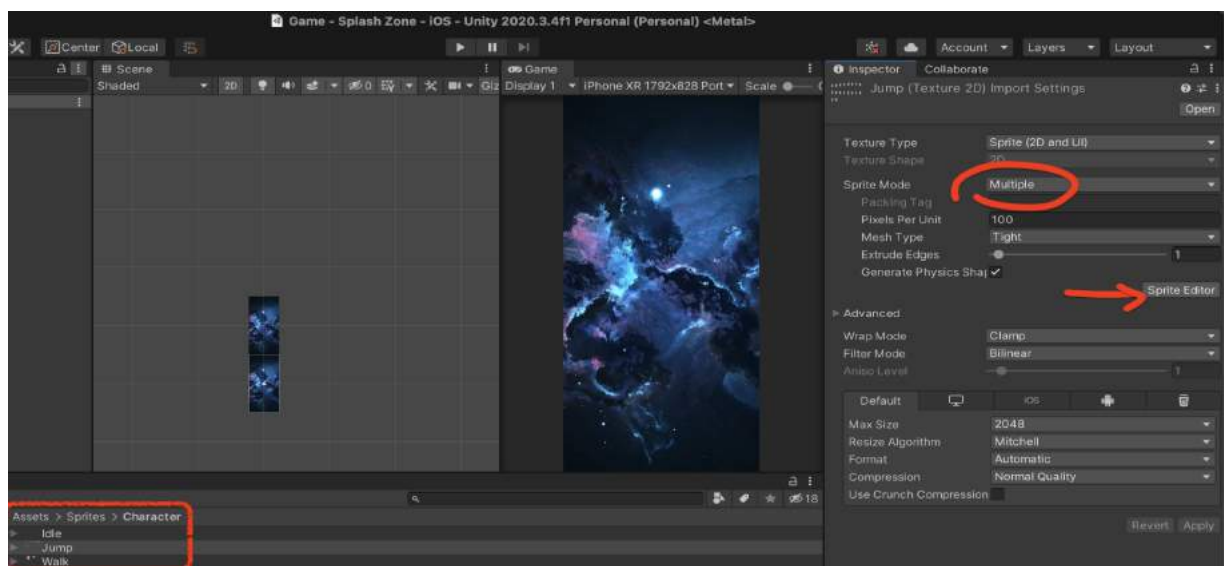


Figure 5.5.1. Idle, jump and walk animations

As seen in the screenshot above, the character has 3 animations: idle, walk and jump. After changing the sprite mode to multiple, the animation images were parsed with the sprite editor.

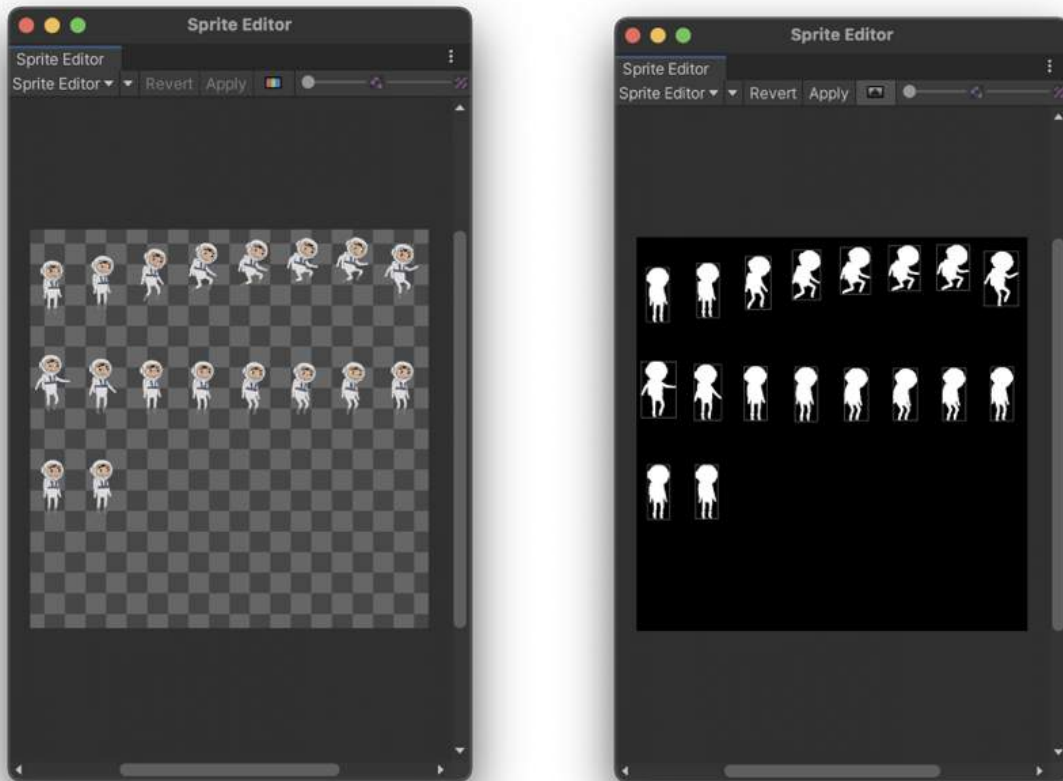


Figure 5.5.2. Sprites of the character animations

When run after the above process, the animation came up very quickly and didn't look very visually pleasing. This issue was also fixed by increasing the display time of each frame in the animation.



Figure 5.5.3. Optimizing sprites for animation



The next step is to use the animator window to switch and order the aforementioned animations.

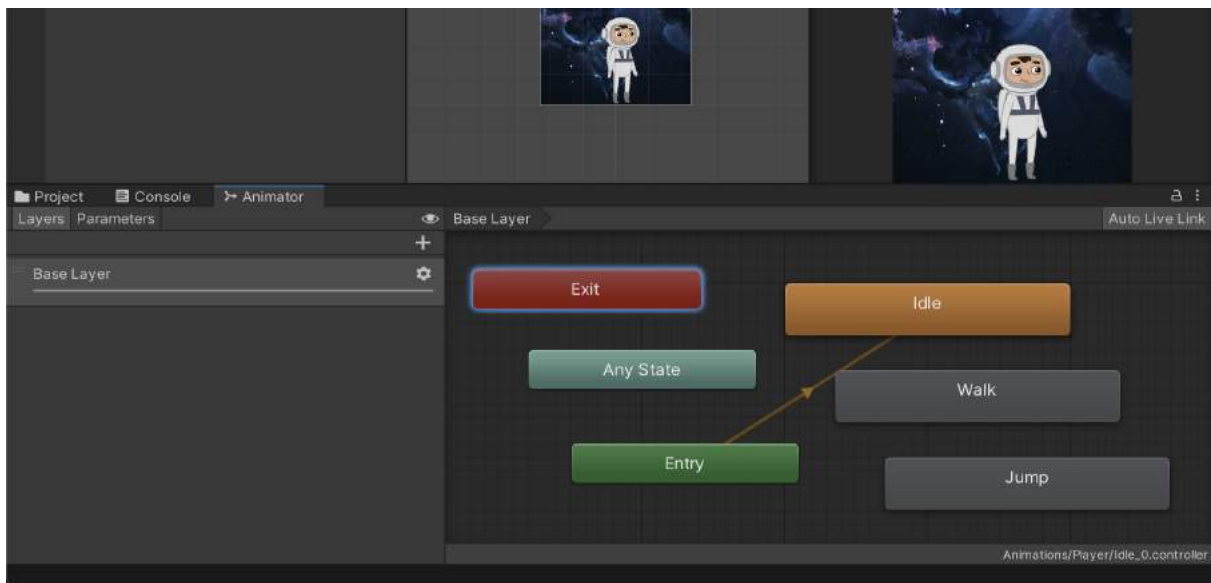


Figure 5.5.4. Aforementioned animations

In the base layer, there are entry, any state and exit state as default. In this game, bidirectional links were established between the three animations mentioned.

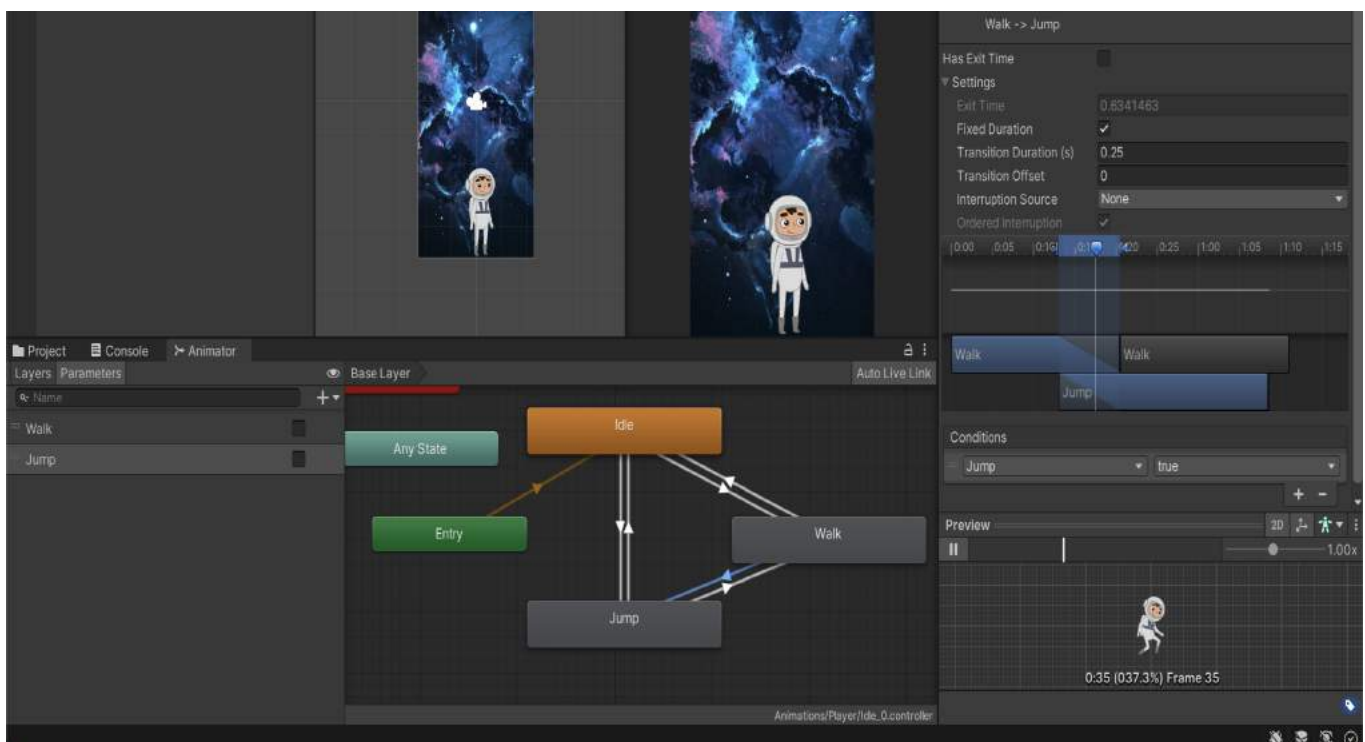


Figure 5.5.5. Ordered animations

Player movement was done correctly with the script below. Mathf functions were used to perform operations [6]. The necessary explanations were expressed with comments.



```
5 public class PlayerMove : MonoBehaviour
6 {
7     Animator animator; // required components
8     Rigidbody2D rb2d;
9     Vector2 velocity;
10
11     [SerializeField] // to interfere via editor
12     float speed = default; // player speed
13     [SerializeField]
14     float speedup = default; // player acceleration
15     [SerializeField]
16     float slowdown = default; // to reduce player speed
17
18     // Start is called before the first frame update
19     void Start()
20     {
21         animator = GetComponent<Animator>(); // induction of the components
22         rb2d = GetComponent<Rigidbody2D>();
23     }
24
25     // Update is called once per frame
26     void Update()
27     {
28         KeyboardControl();
29     }
30
31     void KeyboardControl()
32     {
33         float movementInput = Input.GetAxisRaw("Horizontal"); // horizontal movements
34         Vector2 scale = transform.localScale; // to fix animation
35
36         if(movementInput > 0) // right arrow or 'D' to move right
37         {
38             velocity.x = Mathf.MoveTowards(velocity.x, movementInput * speed, speedup * Time.deltaTime);
39             // Mathf is a collection of common math functions
40             animator.SetBool("Walk", true); // walk animation
41             scale.x = 0.7f; // to fix animation when start to move right
42         }
43         else if(movementInput < 0) // left arrow or 'A' to move left
44         {
45             velocity.x = Mathf.MoveTowards(velocity.x, movementInput * speed, speedup * Time.deltaTime);
46             animator.SetBool("Walk", true);
47             scale.x = -0.7f; // to fix animation when start to move left
48         } else // when no input
49         {
50             velocity.x = Mathf.MoveTowards(velocity.x, 0, slowdown * Time.deltaTime); // the character will stop
51             animator.SetBool("Walk", false); // walk animation is over
52         }
53
54         transform.localScale = scale; // to update fixed walking animation
55         transform.Translate(velocity * Time.deltaTime); // updating the character speed
56     }
57 }
```

Figure 5.5.6. Player movement script

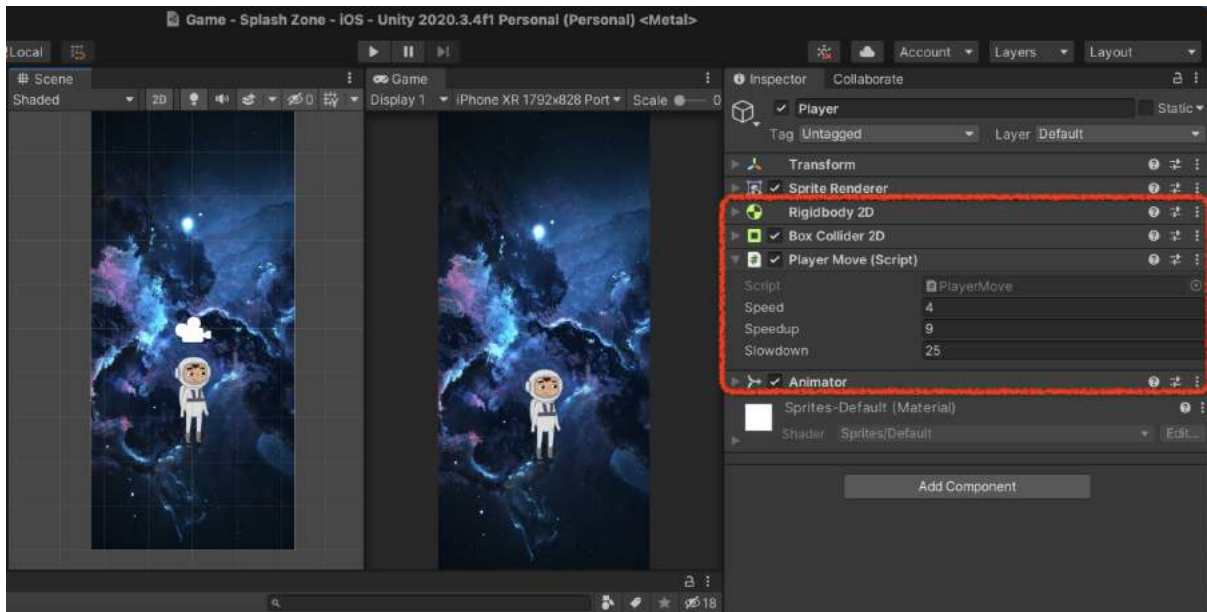


Figure 5.5.7. Character components

The components in the player object are as in the screenshot above. Transform allows us to manage parameters such as the size of the object and its position in the scene. As said before, sprite renderer renders sprites for 2D graphics. Rigidbody 2D ensures that the laws of physics (such as gravity, friction) are applied and controlled to the object. Collider allows objects to interact with each other when they come in contact with each other. The reason for using a box collider in a player object is due to the smooth shape of the object. As you can imagine, Animator also allows animations to be visualized and edited.

## 5.6. Platforms

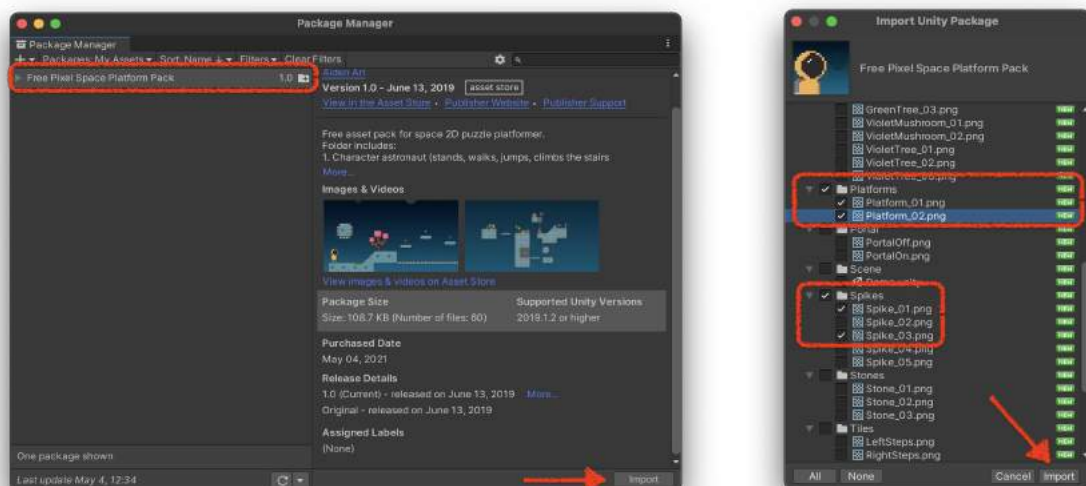


Figure 5.6.1. Platform package from Asset Store

At this stage, an asset prepared for free use was used from the Unity Asset Store. Named as Pixel Space Platform Pack, and then the platforms and spike sprites we need from these asset packs were imported into the game [7].

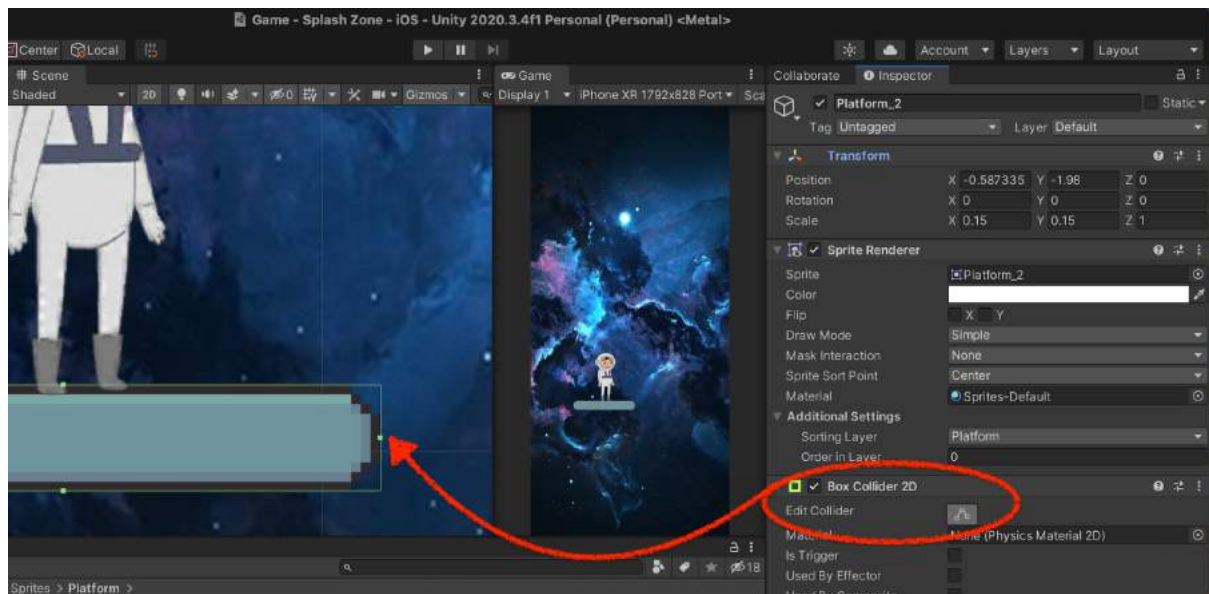


Figure 5.6.2. Editing box collider 2D of the character

The edges of the collider have been smoothed as seen in the screenshot above to create a 2D collider that matches the shapes of the platforms.

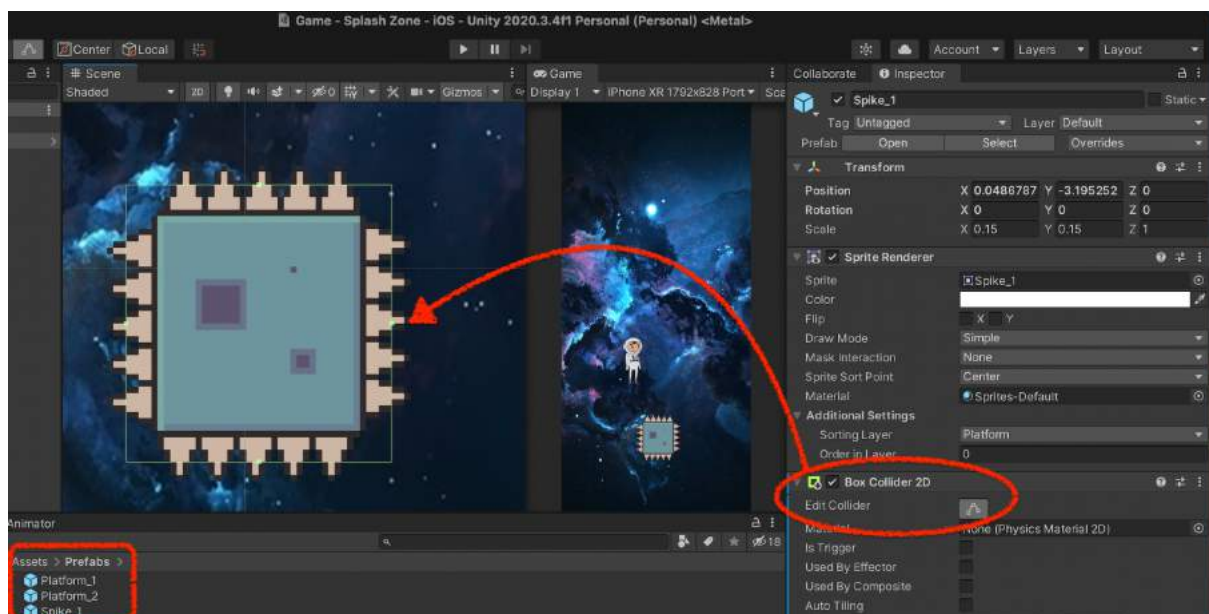


Figure 5.6.3. Prefabs

The same process done for platforms was done for deadly platforms. And since the same types of these platforms will be used many times, the platforms have been prefabs.



Platforms have BoxCollider2D to manage collisions, a boolean variable isMove that keeps track of whether they are moving or not and randomSpeed to make the speed of each platform different from each other. In addition, platforms have the Movement method to manage their movements from other classes. Here, the movements of the platforms are provided by the Unity Mathf library's PingPong method [8].

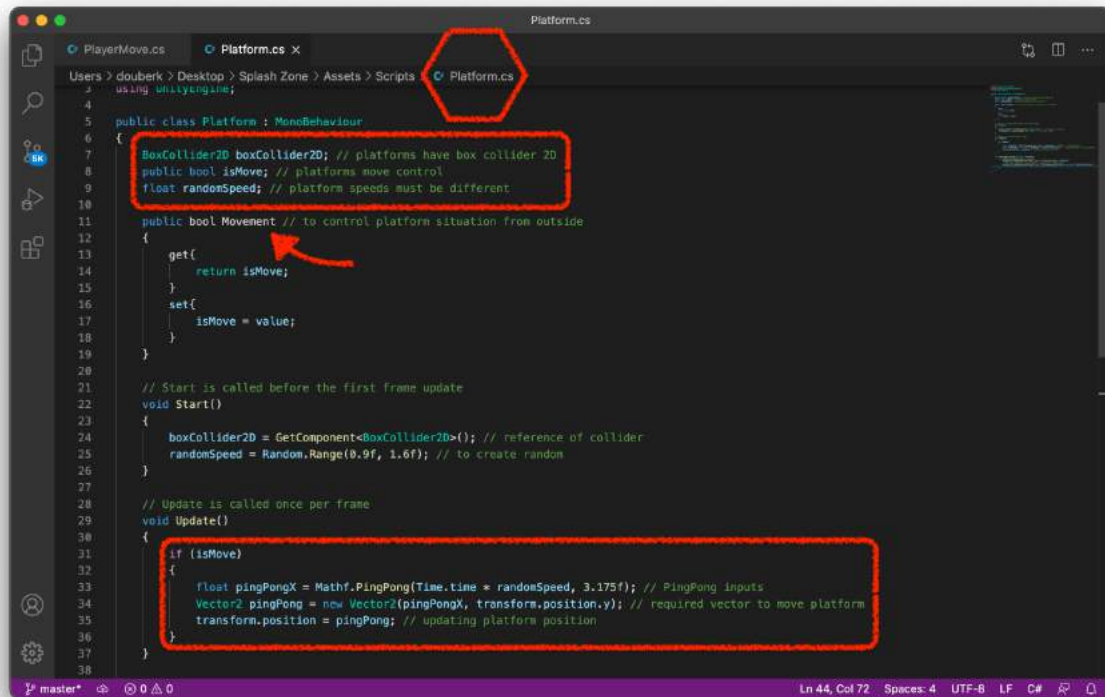
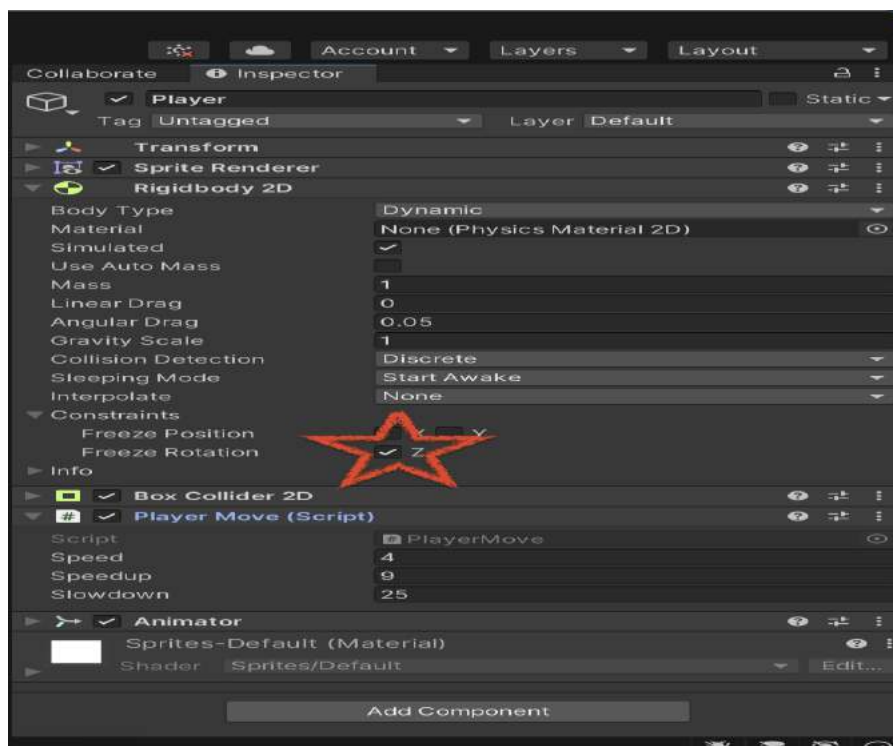


Figure 5.6.4. Platforms script



When the surface of the collider in the platform object that is in contact with the character is not flat and the colliders of these two objects interact, the character turns to the right or left and falls. In order to avoid this situation, the rotation of the character object, that is the feature of rotating around itself, has been turned off as in the screenshot.

## 5.7. Object Pooling

Continuously destroying and re-creating objects reduces game performance. Platforms such as the background and camera in this game also have to go forever. Using the Object Pool design, moving pre-created platforms made the platforms exist forever [9].

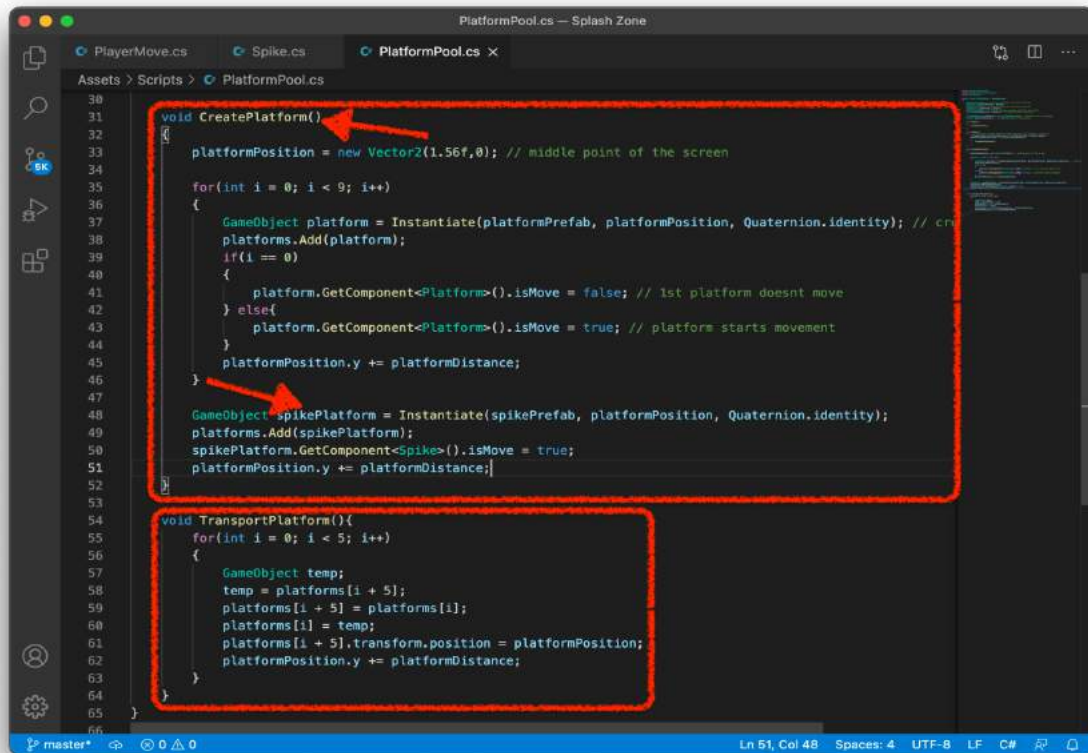


Figure 5.7.1. Platform pooling script

As seen in the script above, the method creates 3 different types of platforms and keeps them in a list. In addition, the first platform which the character will form, is created motionless. After this process, the ten platforms created are moved up as long as the game continues in synchronization with the movement of the camera.

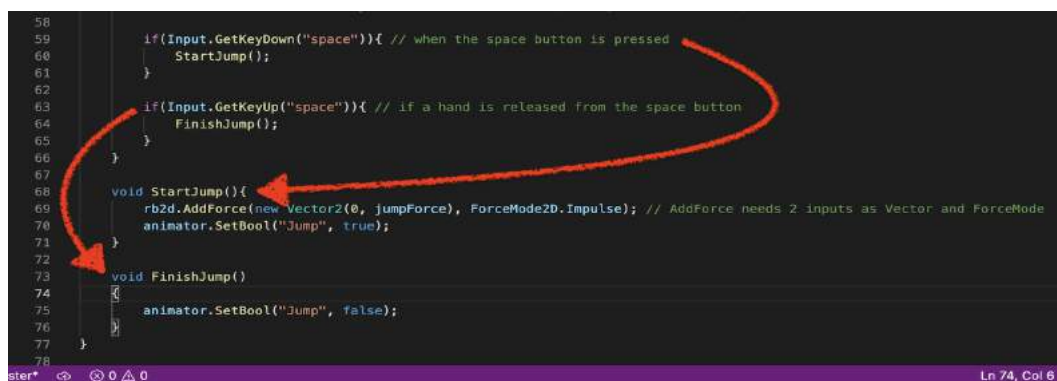


Figure 5.7.2. Space button functions

Currently, character movements are provided from the keyboard. In these operations, jumping is started and ended by using the GetKeyUp/GetKeyDown methods for the space key. 2D vector is used for character jumping and Rigidbody2D is used for physical factors. It is also managed with an animator object for animations that occur during this time [10].

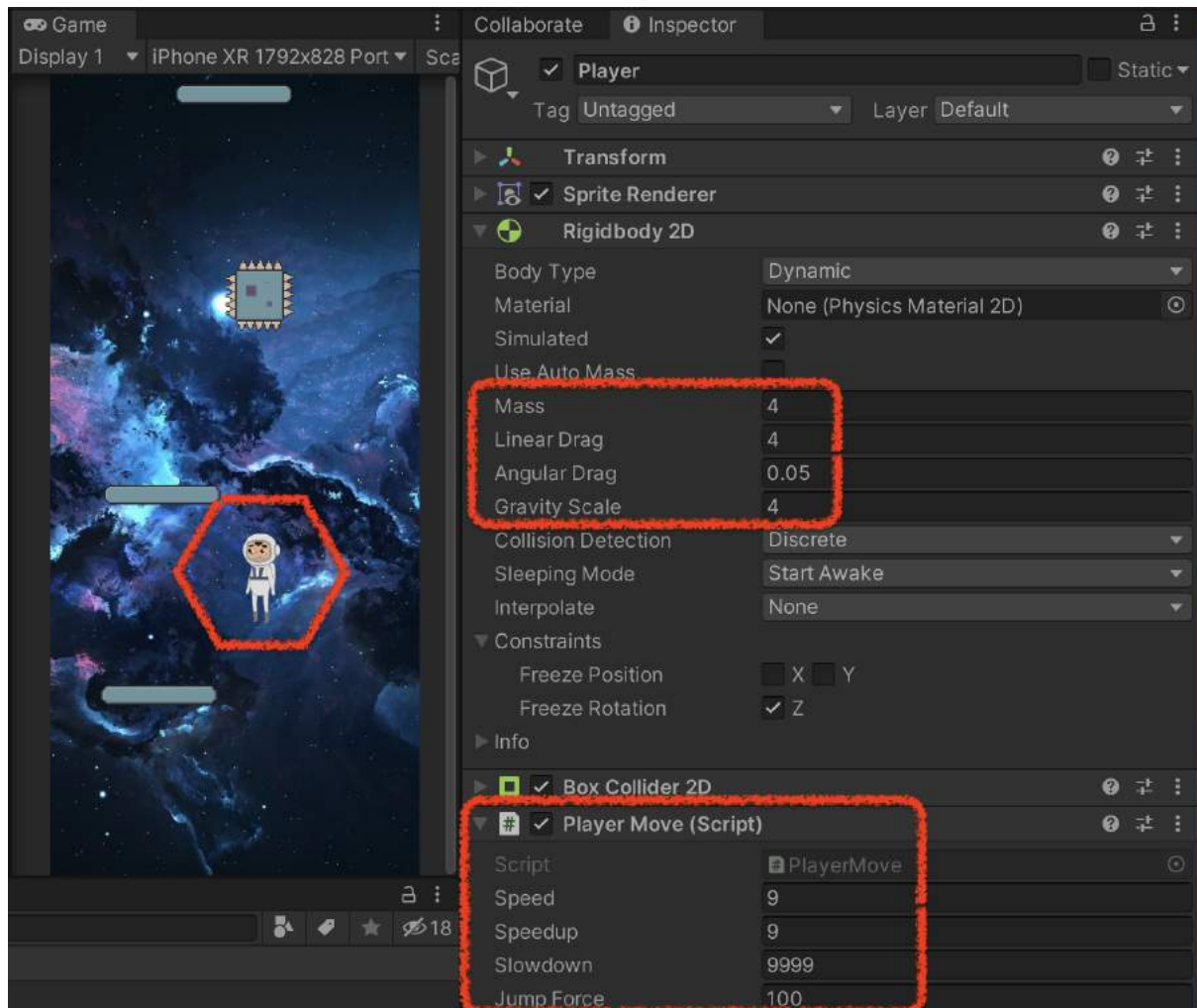


Figure 5.7.3. Character variables

As seen above, different types of platforms and movement of the character are provided. The character's mobility and the values of the physical factors affecting them have been optimized to make the game more playable.

## 5.8. Legs & Jump Limit

```

38
39 void OnTriggerEnter2D(Collider2D collider){
40     if(collider.gameObject.tag == "Legs"){
41         GameObject.FindGameObjectWithTag("Player").transform.parent = transform;

```

Figure 5.8.1. Leg tag and synchronized movement

Now the character can stand still on the moving platform, but there was a problem in this case. While the player only needs to move with the platform when his feet touch the platform, for example when even his head touches the platform, the character moves with the platform. To prevent this situation, a new child component is added to the character object, a new tag is given to the created object and its collider is arranged [11].

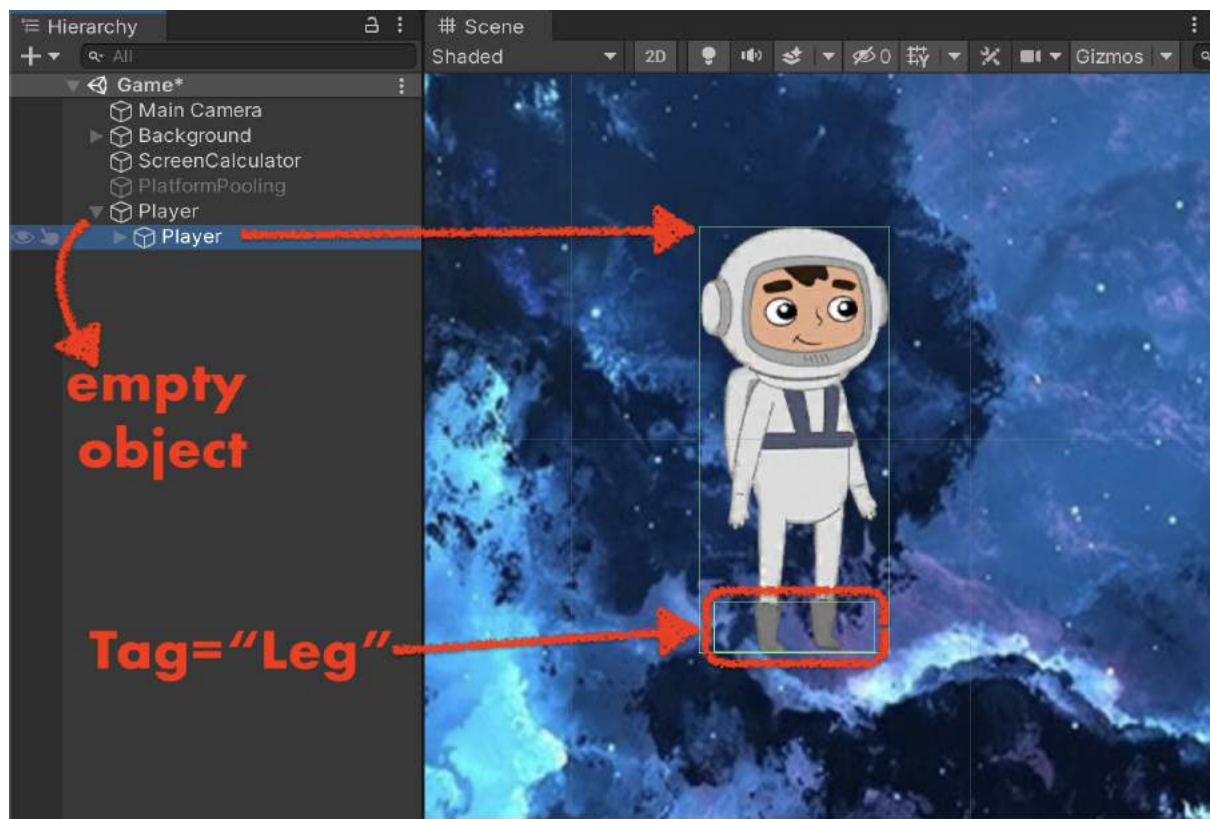


Figure 5.8.2. Empty leg tag object

The gameplay changes by setting a limit on the jump function of the character in the game.

```

71
72 void StartJump(){
73     if(jumpCounter < jumpLimit)
74     {
75         rb2d.AddForce(new Vector2(0, jumpForce), ForceMode2D.Impulse);
76         animator.SetBool("Jump", true);
77     }
78 }
79
80 void FinishJump()
81 {
82     animator.SetBool("Jump", false);
83     jumpCounter++;
84 }
85
86 public void ResetJumpCounter()
87 {
88     jumpCounter = 0;
89 }
90 }

```

Figure 5.8.3. Jump limit script



Before the jump function works, it is checked whether the character has exceeded the jump limit. If not, the jump function works. Each time a character's jump ends, he moves one step closer to his jump limit. When the character touches any platform, the jump count is reset and the character can jump many times [12].

```

38
39 void OnTriggerEnter2D(Collider2D collider){
40     if(collider.gameObject.tag == "Legs"){
41         GameObject.FindGameObjectWithTag("Player").transform.parent = transform;
42         // when legs component is on a platform, jump counter resets itself
43         GameObject.FindGameObjectWithTag("Player").GetComponentInChildren<PlayerMove>().ResetJumpCounter();
44         // ResetJumpCounter method is script of the child component
45     }
46 }
47

```

Figure 5.8.4. Jump limit resets itself

## 5.9. Menu Scene

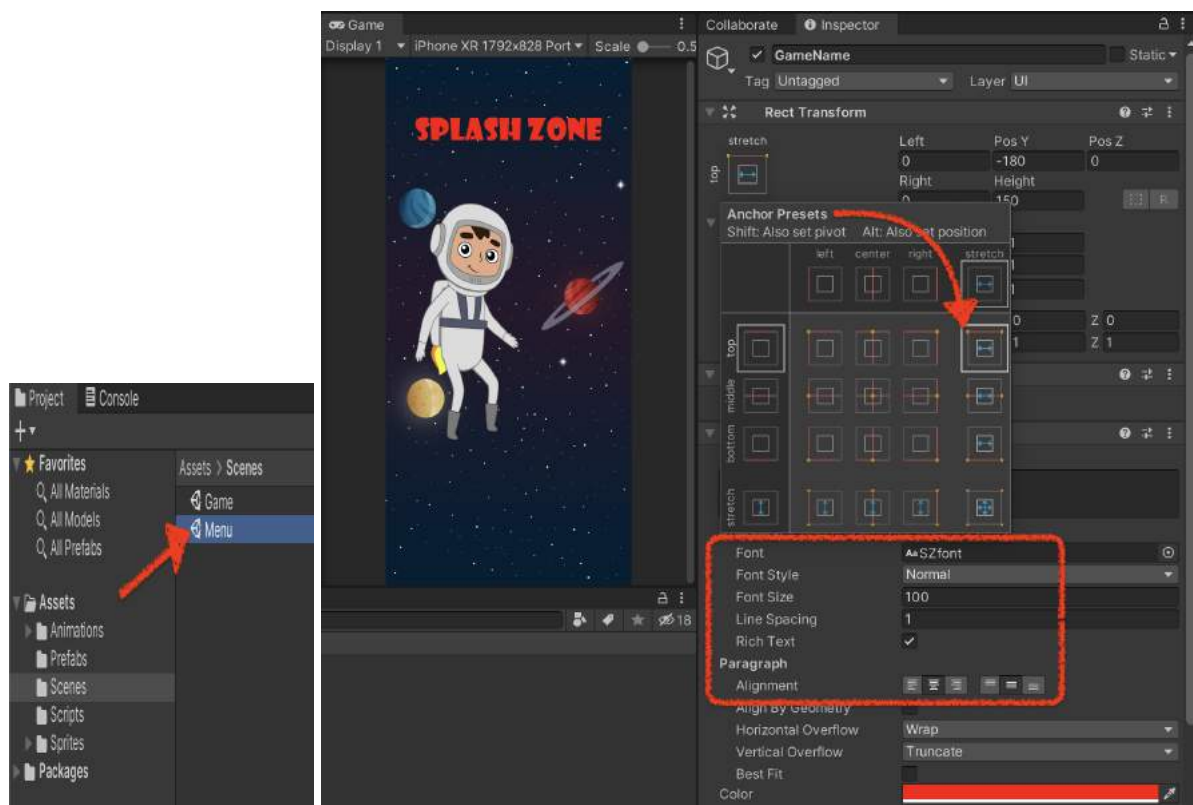


Figure 5.9.1. Menu scene

In this section, after the game scene as the first, the menu scene is provided with transition functions to other scenes. After the menu scene is created, the background is added and Anchor Presets are used to set its position. Then, adjustments were made to the text of the game name. With these settings, the font type, font size and positioning of the text on the canvas were determined.

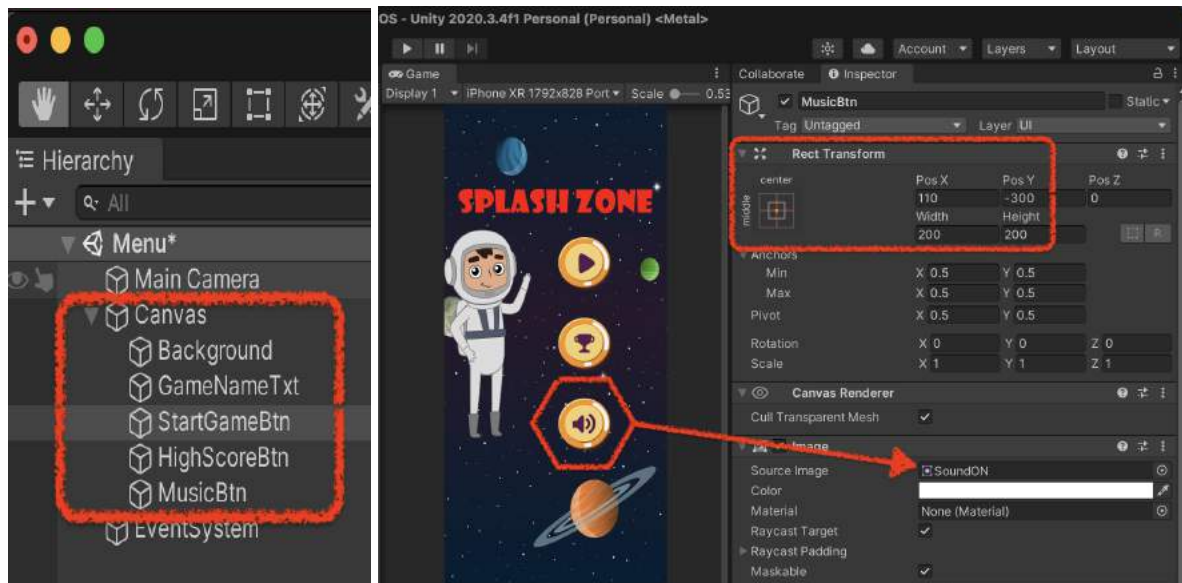


Figure 5.9.2. Menu scene objects

Required button objects have been created on the canvas. These will perform the functions of starting the game, displaying high scores and turning the game sound on and off. The positions and sizes of the buttons have been adjusted and the images in the downloaded asset store package as the source are selected [13].

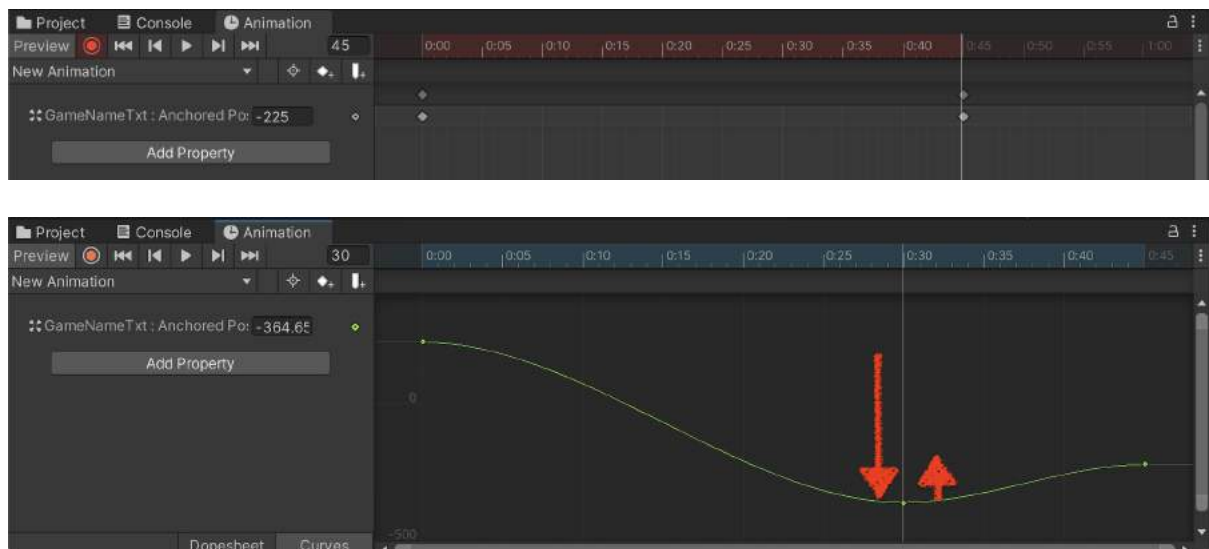


Figure 5.9.3. Animations of the menu scene objects

Here, animations have been created for the objects in the menu scene to come to the screen in an animated way. The start/end time of the animation and the position of the object during the animation have been adjusted. Then, the "Curves" window was opened and the time period of this animation was determined [14].

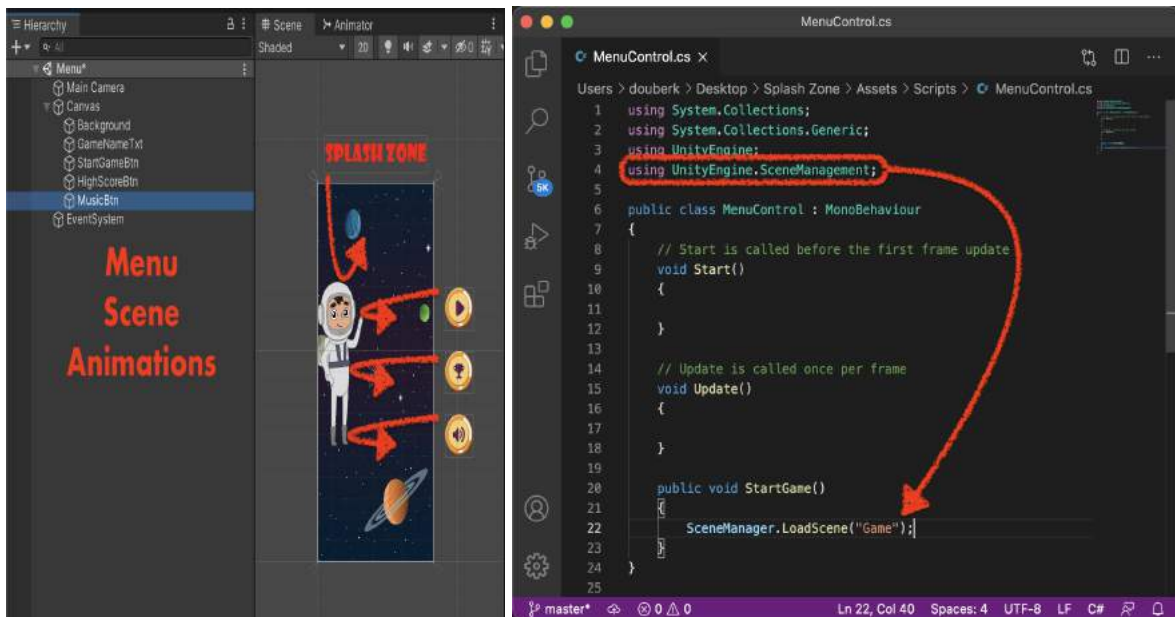


Figure 5.9.4. Menu control script

At this stage, a "Menu Control" script has been created and the unity engine's scene management library has been added to this script [15]. Thus, the mentioned script was added to the buttons as a component and when the button was pressed, the transition to the required scene was completed.

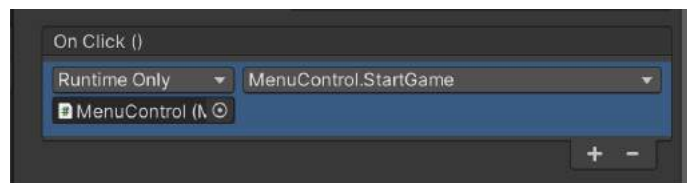


Figure 5.9.5. On click events

The visual change of the volume button according to whether the game music is on or off has been completed. The sprites of the sound button were kept in a list and after the necessary components were provided to the "Menu Control" script, the sprite change of the button was successfully provided.

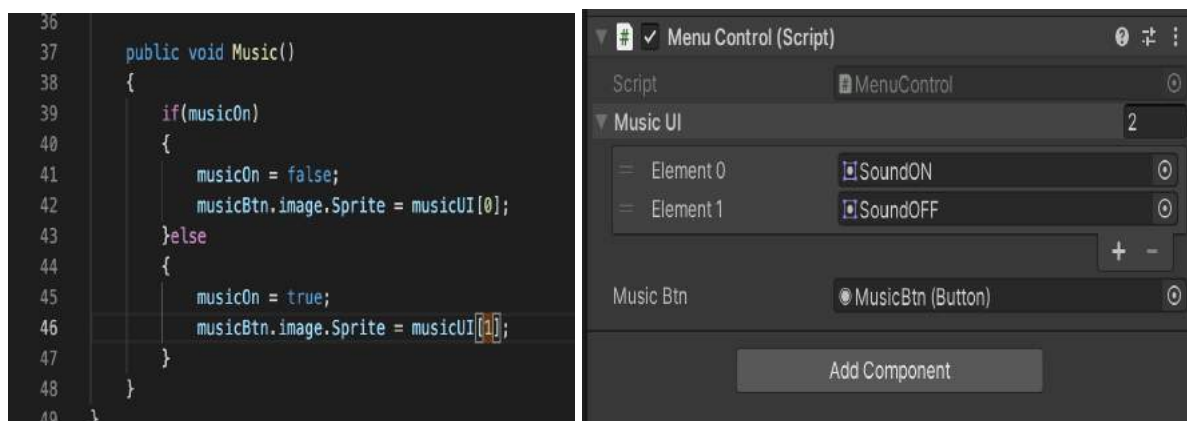


Figure 5.9.6. Touch up inside music buttons

## 5.10. High Score Scene

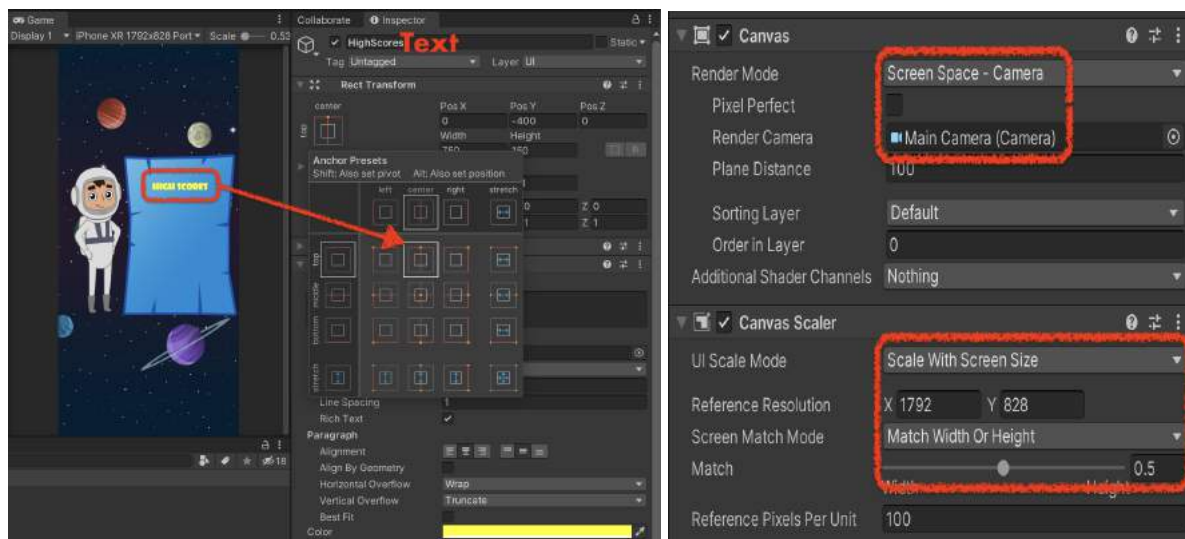


Figure 5.10.1. High score scene

High scores, which is the 3rd scene, have been added to the game. The text on it is aligned to coincide with the middle and top of the canvas. Then some changes were made to make the high score scene appear proportionally on all device resolutions [16]. Changed render mode to camera and scale mode to screen scales. The reference resolution variables were manipulated.

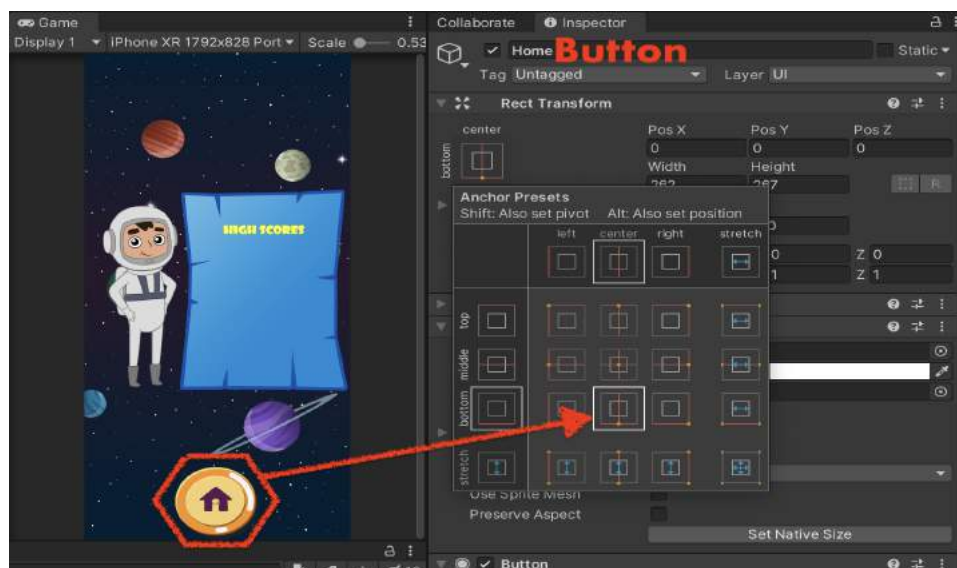


Figure 5.10.2. Back to main menu button

After the scene creation, a button has been added to return to the main menu from the high score scene and the transition between these scenes is completed with the "MenuControl" script.



## 5.11. Controllers for Mobile Devices

During the development of the game, the game could be played from the keyboard. Now it's up to the task of adding the necessary objects to the game scene so that it can be played on mobile devices.

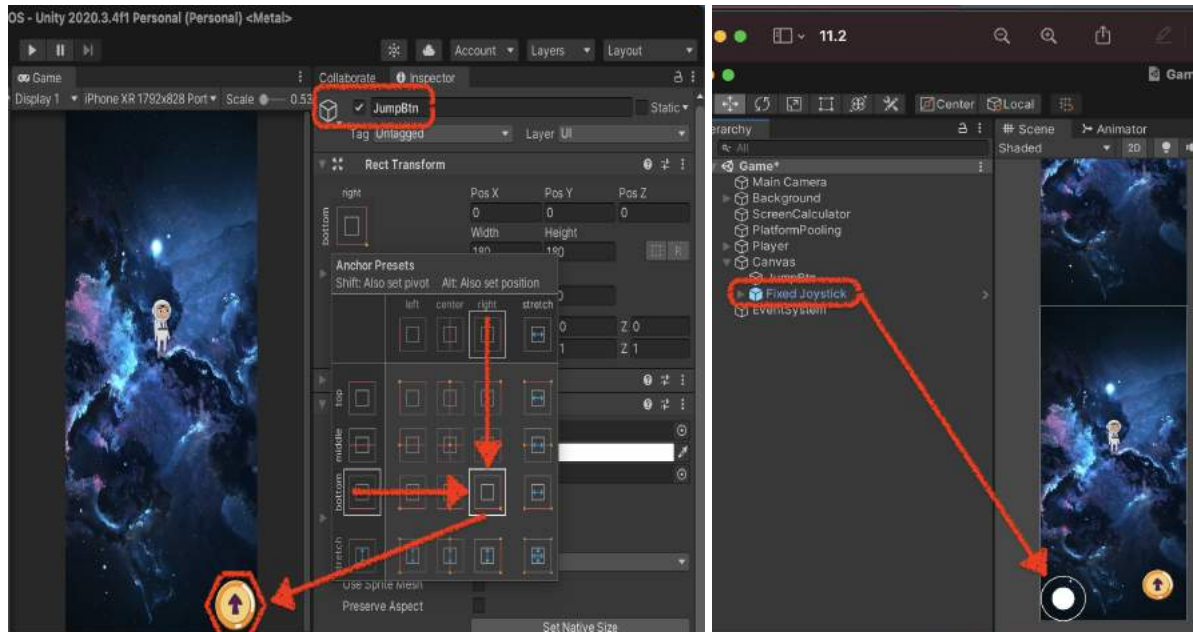


Figure 5.11.1. Jump button UI

First, a button to make the character jump was added to the lower right corner of the game scene. After the necessary adjustments of the button, a joystick package has been added to the game from the Asset Store to enable the character to move left and right [17].

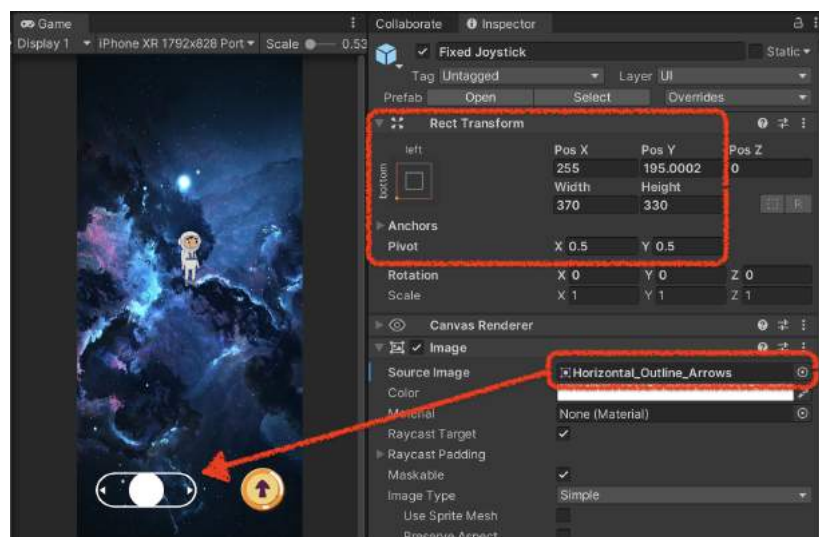
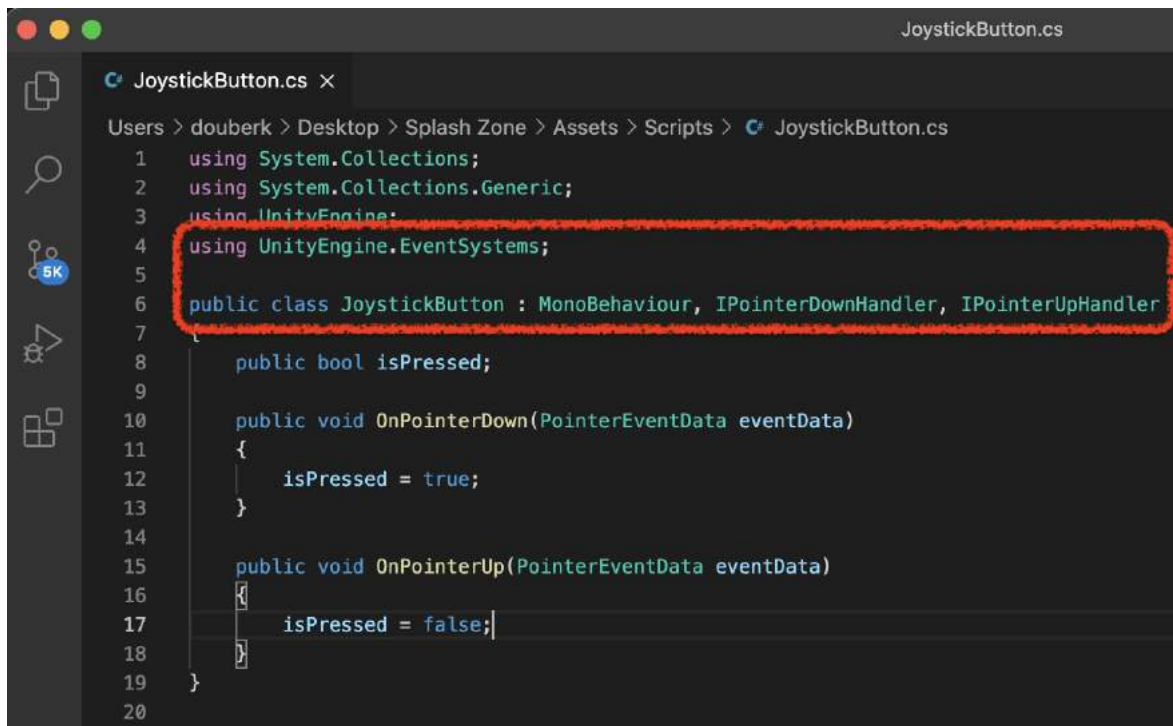


Figure 5.11.2. Joystick UI

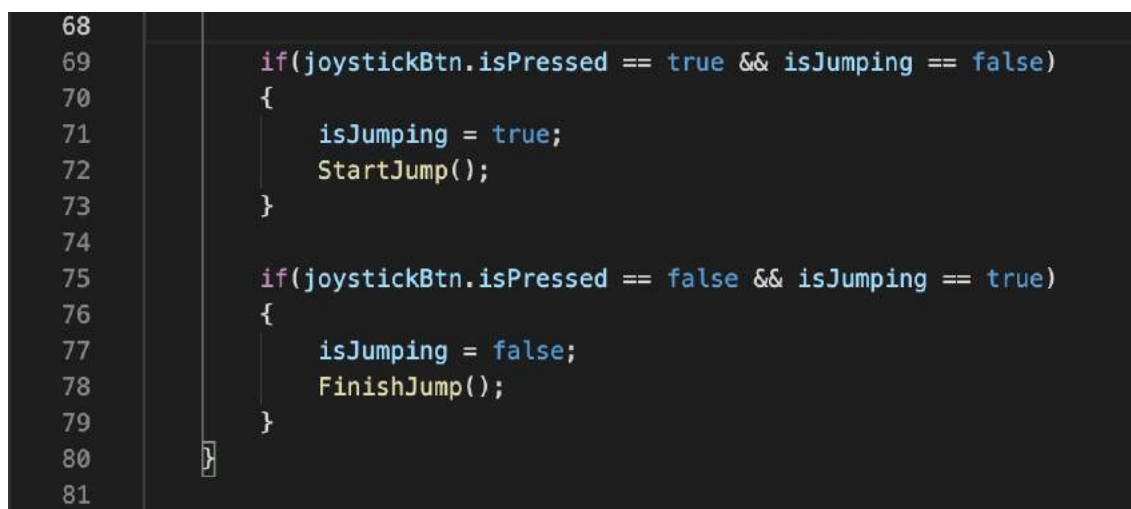
After the necessary operations for the joystick view were completed, a Joystick object was created and managed in the “PlayerMovement” script.



```
JoystickButton.cs
Users > douberk > Desktop > Splash Zone > Assets > Scripts > JoystickButton.cs
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.EventSystems;
5
6 public class JoystickButton : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
7 {
8     public bool isPressed;
9
10    public void OnPointerDown(PointerEventData eventData)
11    {
12        isPressed = true;
13    }
14
15    public void OnPointerUp(PointerEventData eventData)
16    {
17        isPressed = false;
18    }
19 }
20
```

Figure 5.11.3. Joystick button script

Afterwards, a special script was created for the joystick object. Unity engine's library named "EventSystems" has been added to this script. The above interfaces were used to capture both the time when the jump button on the screen was pressed and when it was released. These "IPointerDownHandler" and "IPointerUpHandler" interfaces are used to control whether the joystick object is pressed [18].



```
68
69     if(joystickBtn.isPressed == true && isJumping == false)
70     {
71         isJumping = true;
72         StartJump();
73     }
74
75     if(joystickBtn.isPressed == false && isJumping == true)
76     {
77         isJumping = false;
78         FinishJump();
79     }
80
81
```

Figure 5.11.4. Jumping control script

After the use of the necessary interfaces, the joystick object has been adjusted to start and stop jumping, as is the control from the keyboard. And finally, with a boolean variable, the possibility of pressing the joystick while pressed again has been eliminated.

## 5.12. Scoring System

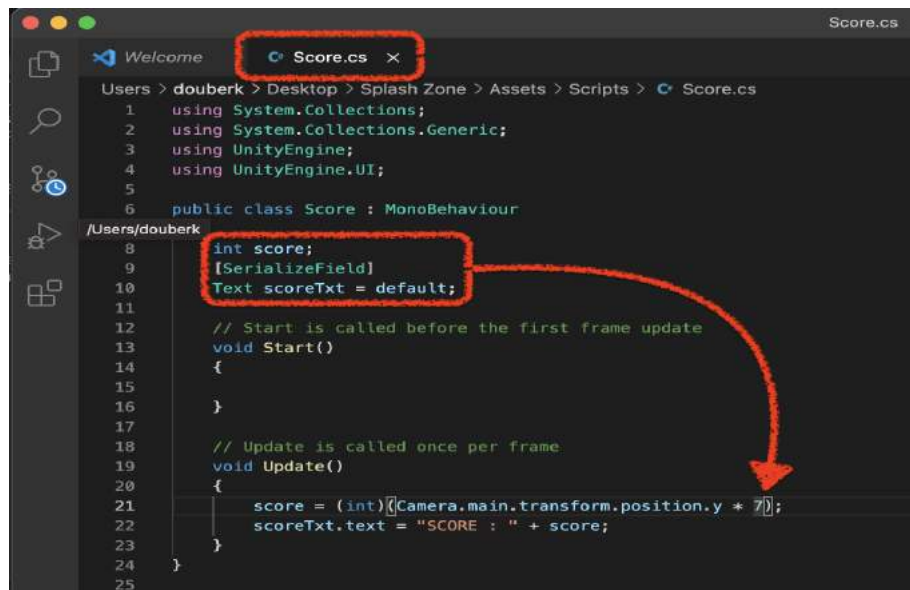


Figure 5.12.1. Scoring script

It was provided to earn points as the player continues to play. Here, the player's score is directly proportional to how high he actually climbed. That's why the score calculation system was designed in sync with the upward movement of the camera.

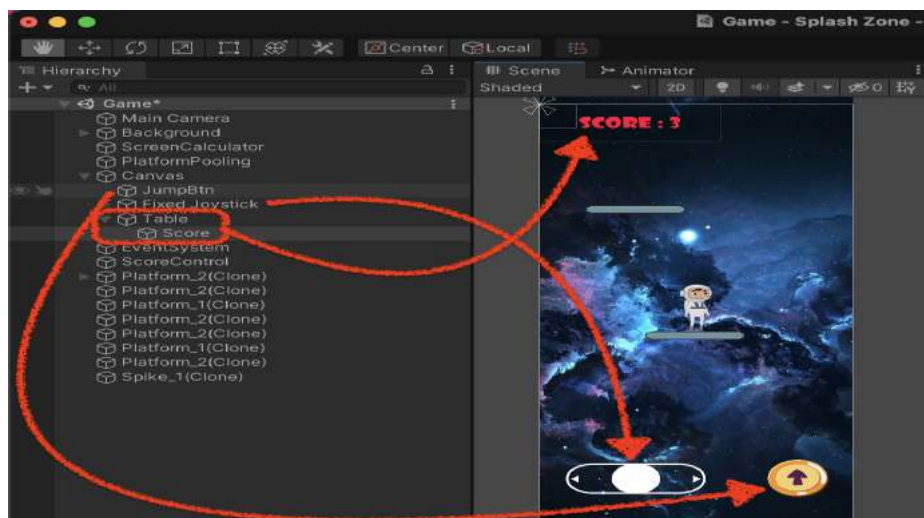


Figure 5.12.2. UI elements

Thus as a result, the character's jump button, the joystick graphical interface that allows the character to move to the right and left, and a score system that calculates and displays the score the character will win the game were created and fixed on the canvas.

### 5.13. End Game

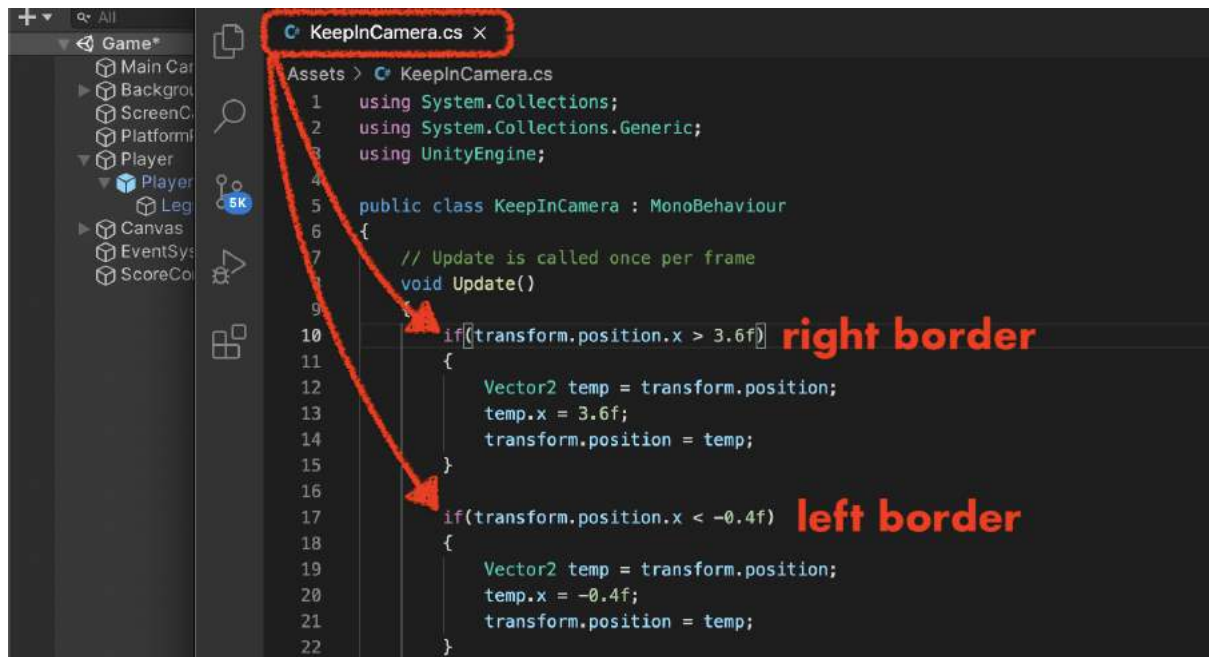


Figure 5.13.1. Keep the character in camera script

Due to the gameplay of the game, the character object should not leave the camera except in the case of death. The script above was used to ensure this situation. The vector size applied to the character may cause the object to protrude from the left or right border of the screen. In this case, the position of the character object is synchronized to the positions of the left/right borders of the camera.

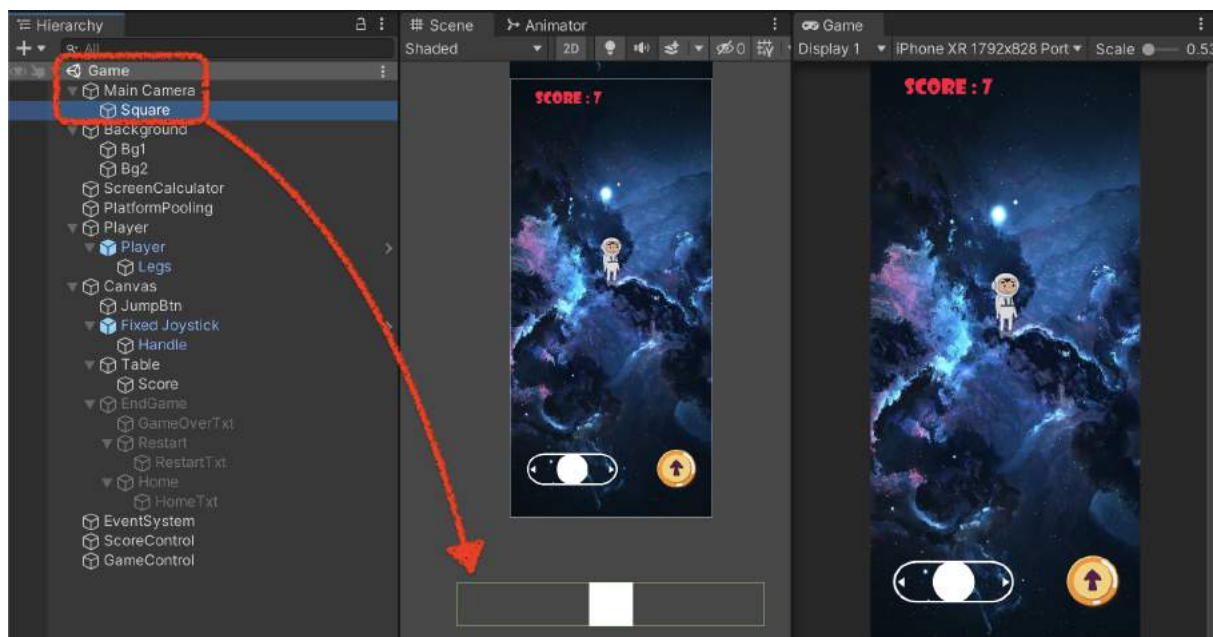


Figure 5.13.2. End game objects



There are two possible reasons why the character dies. The first is that the character stays below the lower limit of the camera. As seen in the screenshot above, this is achieved with the child object added to the main camera object. This object moves synchronously with the main camera object.

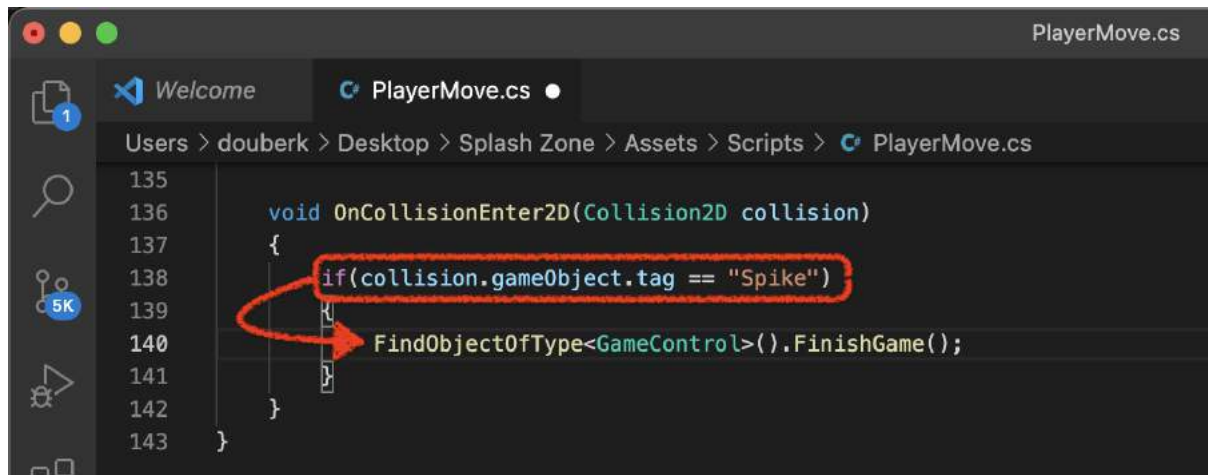


Figure 5.13.3. End game collision control

The second reason for the character to die is when it comes into contact with a "Spike" type platform. This situation is controlled by the tag logic in the OnCollisionEnter2D method seen above.

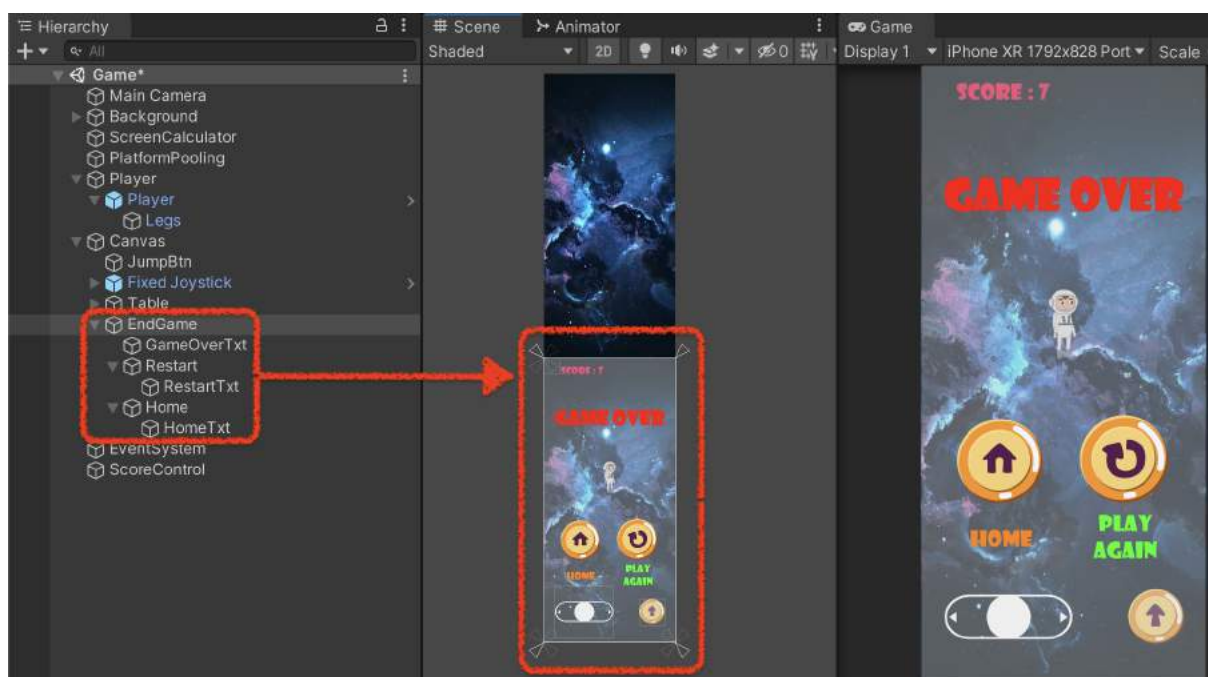


Figure 5.13.4. End game panel

An object named "EngGame" has been added to the canvas where the jump button and joystick object are located. In this object, there are game over text, play again and return to the main menu buttons. In addition, this object becomes visible only in the death states of the character.

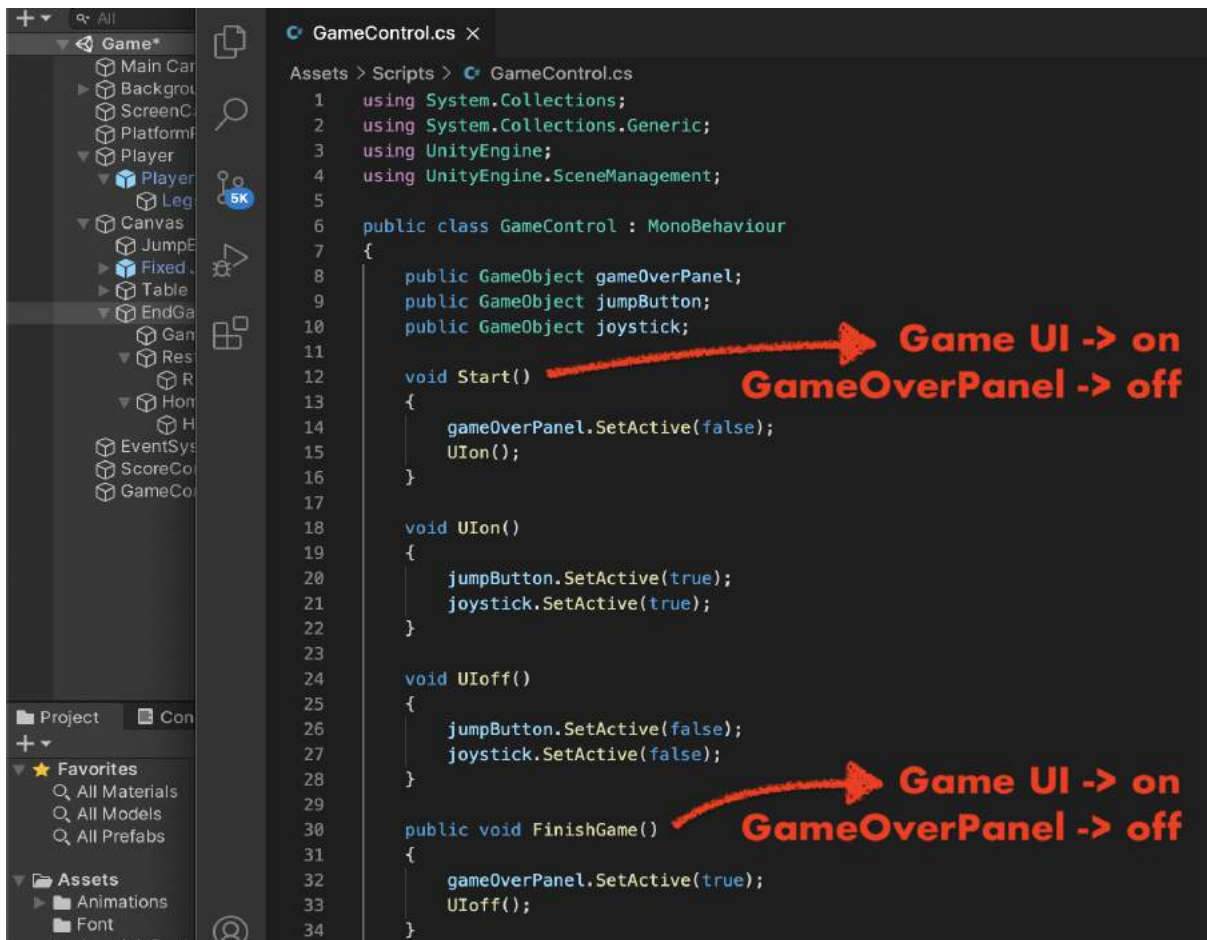


Figure 5.13.5. Game UI and game over panel

As seen in the "GameControl" script above, game control interfaces become visible when the game is started. These are the joystick and jump button that provide movement. When the game is over, these objects are made invisible and then the "gameOverPanel" is shown to the player.

## 5.14. Highest Score



Figure 5.14.1. Highest score control

The next task is to keep the highest score ever earned and show it to the player. As seen in the score script, when a new score is obtained, it is compared with the highest score. And if the new score is higher, it takes its place as the highest score. Then it is shown to the player with a text on the "gameOverPanel".

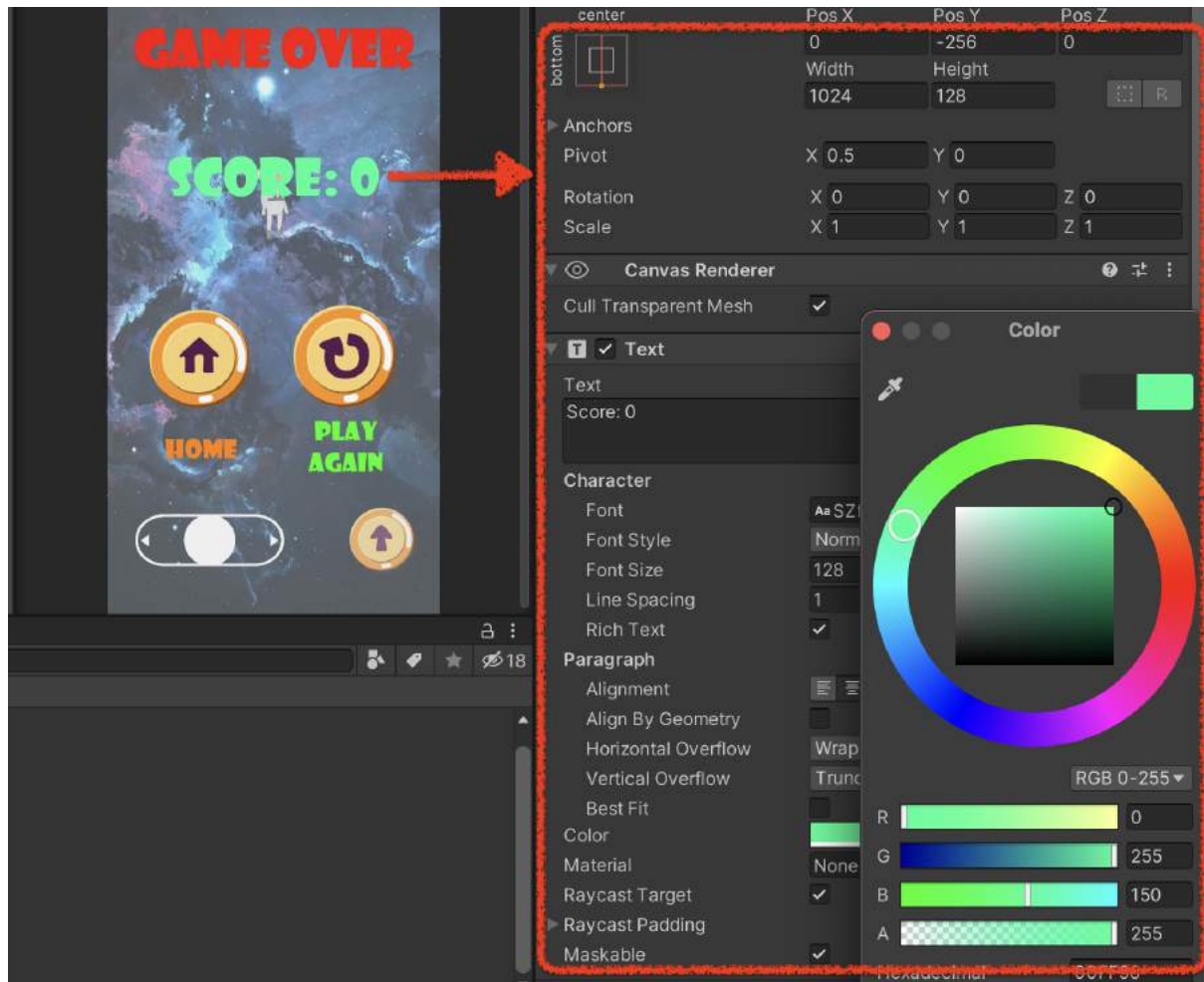


Figure 5.14.2. Score text element on end game panel

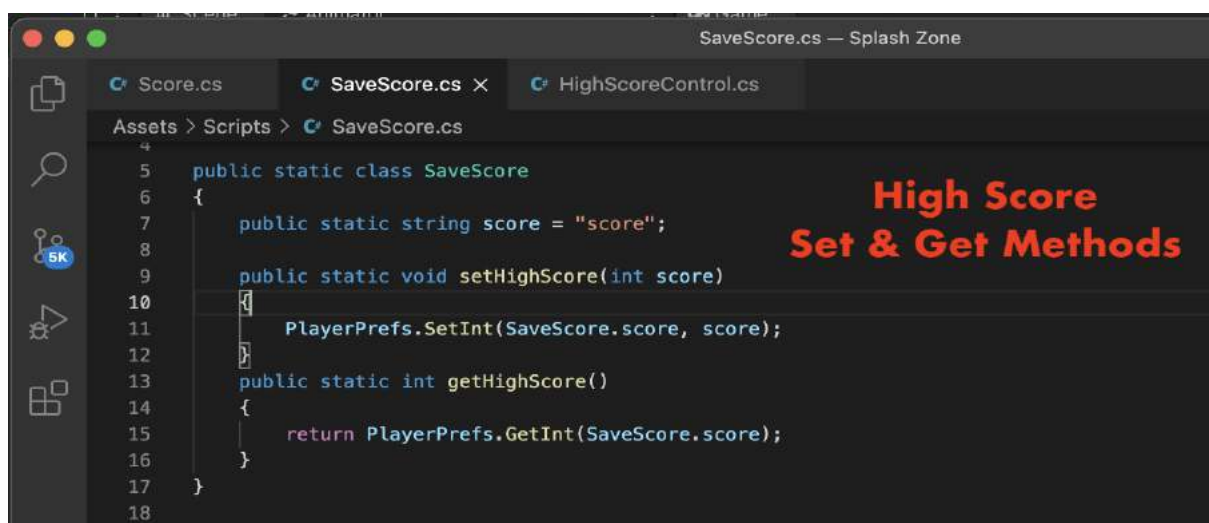


Figure 5.14.3. Highest score get/set methods

The get method was used to show the highest score, and the set method to determine a new highest score. This script was created as static so that these methods can be called from different blocks. In this case, it has become mandatory for the get and set methods to be static.

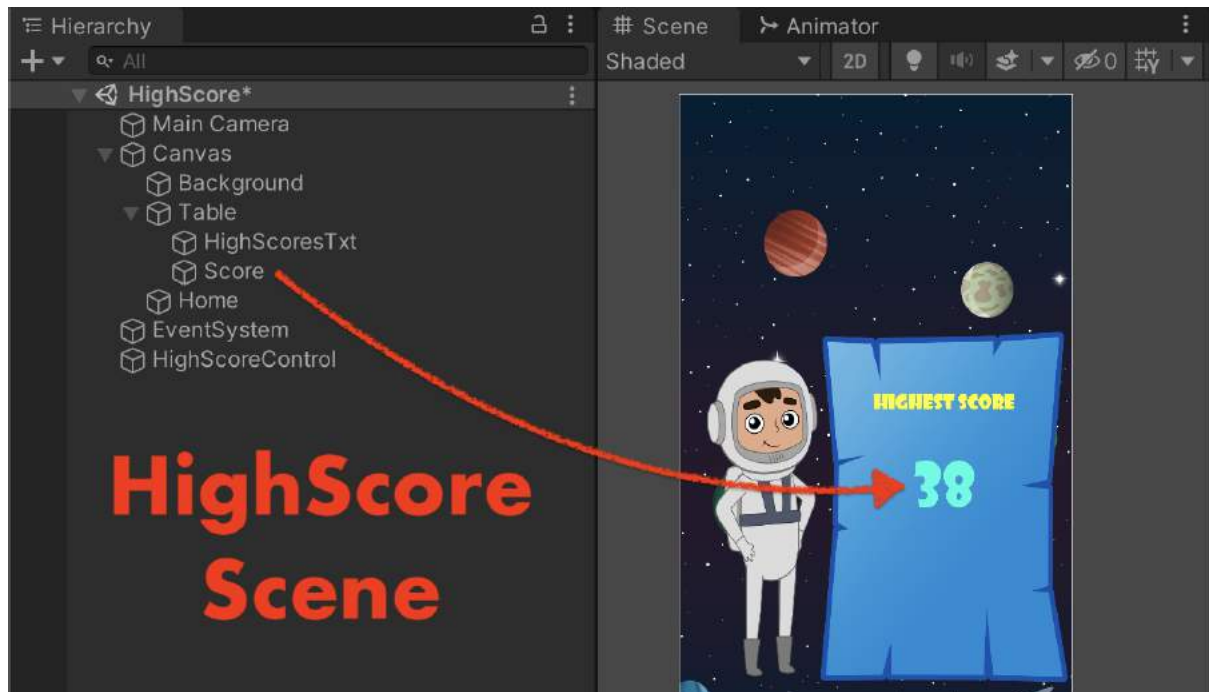


Figure 5.14.4. Highest score scene

As can be seen in the screenshot above, the "HighScore" scene shows the highest score achieved so far.

## 5.15. Finishing Touch



Figure 5.15.1. Eng game issues



The "gameOverPanel" becomes visible when the character dies and the game ends. The score obtained by the player is found in this panel. Camera motion is terminated and character movement interfaces disappear from the scene.

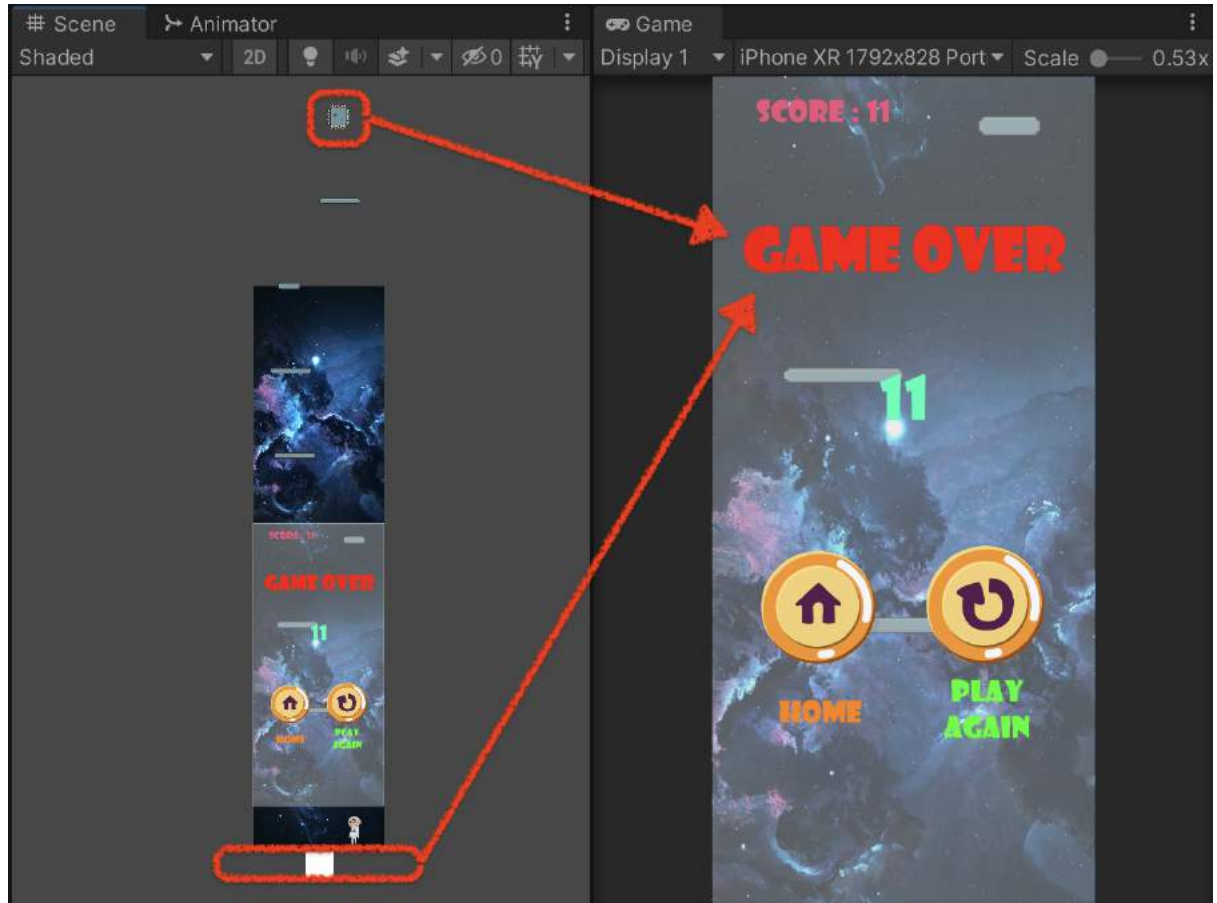


Figure 5.15.2. Game objects to finish game

## **CHAPTER 6**

### **EXPERIMENTS**

- The images to be displayed in the game objects used in two-dimensional games are called sprites. When an object is assigned a sprite, the Sprite Renderer component is automatically added. While working on this project, I experienced how to manage sprite textures using Sprite Renderer.
- Using two-dimensional vectors, experienced how to manage the position, offset, etc. of a point or object in a two-dimensional space.
- Learned to change the position, rotation and scaling of any object in the scene using the Transform component in Unity.
- Experimented with how to separate multiple sprites in a composite using the Sprite Editor window.
- Learned to create and run animations of the objects in the scene by using the animation feature in Unity. Also able to change the parameters of any animation with the functions in this window.
- It was learned that Rigidbody2D provides the physical agents for two-dimensional sprites. And by playing with the variables in this component, the in-game physical factors have been changed and the gameplay has become more immersive.
- It has been learned that the Mathf library contains many mathematical functions ready-made, and by using this library, a few necessary operations in the game have been completed.
- It has been experienced that two-dimensional colliders are needed to control whether the objects in the scene are in contact with each other and to trigger the desired actions as a result. In short, this component helps define the physical shape of the object.



- By using the PingPong method in the Mathf library, the platforms in the game hit the right and left frames and bounce back.
- The working logic of the object pooling design model is grasped. Creating and destroying objects that are used many times in the game decreases performance. Therefore, the objects created with this design model are thrown into a pool and recalled to be used again when needed, instead of being destroyed. Platforms in the project, which can go on forever as long as the game continues, are managed in this way.
- Due to the constant increase in the number of objects in the game, it becomes difficult to manage these objects. Therefore, there is a need to group game objects in order to manage them more easily. Tag feature in Unity is used to manage objects in bulk. In this project, a separate tag was defined for the foot of the character. Thanks to this tag, some bugs have been fixed in case the character moves with the platform while on the platform.
- By using anchor presets, visual objects in a canvas are scaled proportionally to the canvas, even at different resolutions.
- With the SceneManagement feature of the Unity engine, transitions between the menu, game and high score scenes in the project were settled.
- By changing the render mode and render camera in the Canvas game object, an optimized and accurate image was obtained at different resolutions of all devices. The scaling mode in Canvas scaler is updated according to the screen size and different resolutions.

## **CHAPTER 7**

### **CONCLUSION, FUTURE WORK**

#### **7.1. Conclusion**

As a result, I have largely provided the two-dimensional mobile platform game that I created during the analysis and design stages. During this month-long journey, I have gained a lot of knowledge and experience from the mobile game industry. In the implementation phase, I had great experiences in all stages of creating a two-dimensional mobile game using physics and mathematics. I was absolutely delighted with this very enjoyable and challenging process and I had many wonderful experiences.

#### **7.2. Future Work**

- ☐ When certain score thresholds are passed, the game can be stopped and animations can be added. When the game continues again, the theme can be changed and different platform types can be added. At the same time, different movement options can be offered to the character, such as the ultimate.
- ☐ The player's jump limit can be visualized with a bar above the game scene.
- ☐ Apart from the background music, different sounds can be added when the player jumps or dies.
- ☐ A return button to the main menu can be placed in the game scene.
- ☐ Different difficulty levels can be added, and the colors and theme of each difficulty level may vary depending on the level. In addition, at different difficulty levels, the character's movement abilities can be changed.

## REFERENCES

[1] Newzoo 2020 Global Games Market Report (2020).  
[newzoo.com/insights/trend-reports](https://newzoo.com/insights/trend-reports)

[2] Unity Documentation (2019). [Asset Store](#)

[3] Water Animation Video  
[youtube.com/doguberko](https://youtube.com/doguberko)

Game Design Progress Report [Progress Report | CISC 226: Game Design](#)

Game Scene Background [Galaxy Wallpaper Xr](#)

[4] Sprite Renderer  
[How to change a Sprite from a script in Unity \(with examples\)](#)  
[Scripting API: SpriteRenderer](#)  
<https://learn.unity.com/tutorial/introduction-to-the-sprite-renderer#>

[5] Time.deltaTime  
[medium.com/Time.deltaTime](https://medium.com/Time.deltaTime)  
[gamedevbeginner.com/timer-in-unity](https://gamedevbeginner.com/timer-in-unity)  
[Unity - Time.deltaTime](#)  
Character & Menu Scene Sprites [drive.google.com/file](https://drive.google.com/file)

[6] Mathf  
[kodlib.com/unity-matematik-metotlari](https://kodlib.com/unity-matematik-metotlari)  
[Scripting API: Mathf](#)  
[karadotgames.com/unity-mathf](https://karadotgames.com/unity-mathf)

[7] Pixel Space Platform & Spike Pack  
[Pixel Space Platform Pack](#)

[8] PingPong Method  
[c# Unity Mathf.PingPong not working](#)  
[Mathf.PingPong](#)

[9] Object Pooling Design  
[Pool Objects in Unity – GameDev Academy](#)

## [1.Intro to Object Pooling](#)

### [10] Animator

#### [Using Animator Controllers and Triggers](#)


[docs.unity3d.com/Animator.html](https://docs.unity3d.com/Animator.html)

### [11] Tags in Unity

[medium.com/@osmananilozcان/unity](https://medium.com/@osmananilozcان/unity)

#### [GameObject.tag](#)

### [12] GetComponentInChildren

 [36.Unity C# Scripting Tutorial-GetComponentInChildren Function In Unity C#](#)

### [13] Graphical UI

#### [Game GUI Buttons | 2D Icons](#)

### [14] Curves Window - Animator

 [Animation Curves are Awesome!!!](#)

#### [Using Animation Curves](#)

#### [1.Working with Animation Curves](#)

### [15] SceneManager

#### [SceneManager.SceneManager.LoadScene](#)

 [Scene Manager in Unity \(Unity Tutorial\)](#)

#### [How to load a new Scene in Unity \(with a Loading Screen\)](#)

### [16] Canvas

[ogreniyoruz.net/unity-ui-dersleri](https://ogreniyoruz.net/unity-ui-dersleri)

### [17] Joystick Pack

[assetstore.unity.com/input-management/107631#content](https://assetstore.unity.com/input-management/107631#content)

### [18] PointerHandler in Unity

[stackoverflow.com/ipointerdownhandler-approach](https://stackoverflow.com/ipointerdownhandler-approach)

#### [IPointerUpHandler](#)