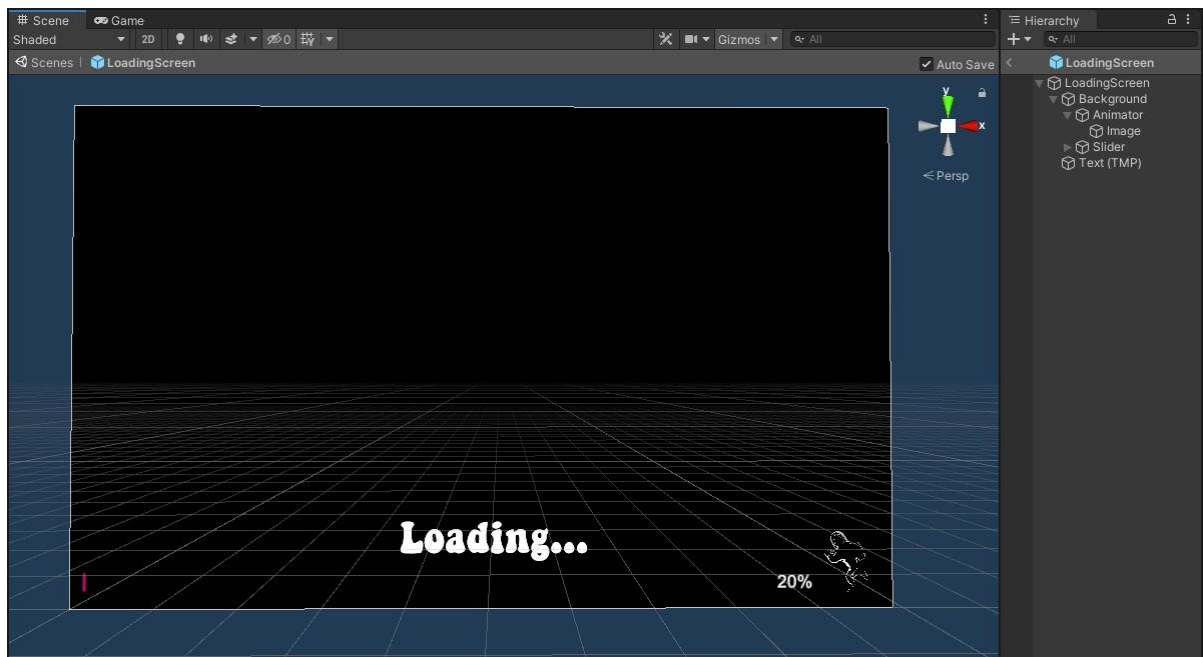


Sprint 01

Assignment 02 : Loading Screen

Step 01: Setup

- First, I created a new Canvas called *"LoadingScreen"* and added it in the *"Resources/UI"* folder as a prefab.
- Then I created an Empty GameObject as a child object into the LoadingScreen group and added an Image as BG.
- Then I repeated the above to create the Running Man Image.
- Then I added an *"Animator"* component to *"EmptyGameObject"*, and assigned it the *"RunningIcon"* Animation Controller.
- Also, I added Animator to LoadingScreen component too and assigned it *"LoadingScreen"* Controller and added the *"AnimationListener"* script, as it has the *"AnimtonEvents"* set up in it which we would access in a script further.
- After doing the above steps, the result:



- I also added a Slider and two Text components, to show the Progress of scene loading.
 - **Slider:** Shows the progress by sliding the red bar.
 - **Text:** Shows the progress from 0% to 100%
- And an Extra “Loading...” text with TextMeshPro.

Step 02: Script & Workflow

- The Scripts workflow is like this:
 - GameController > LevelController > UIController > LevelEndMenuController
- GameController is the main parent components which calls all the other script component.
- So in order to add a logic that would proceed to NextLevel, Restart Game and Exit to MainMenu, we need to add all the code in the GameController.
- And while doing this, we need to pass values & data from all the other dependent scripts.
- To do that, we need to create delegates to pass in the information through various scripts.
- First we declare them, and access them from other scripts, and then repeat the steps till we reach to GameController script.
- It would be the exact opposite workflow as we would pass values from;
 - LevelEndMenuController > UIController > LevelController > GameController

❖ LevelEndMenu.

```
Assets > Scripts > UI > LevelEnd > LevelEndMenuController.cs > LevelEndMenuController > LevelEndMenuController(LevelEndMenu data, int currentLevelID, LevelStatsController levelStatsController)

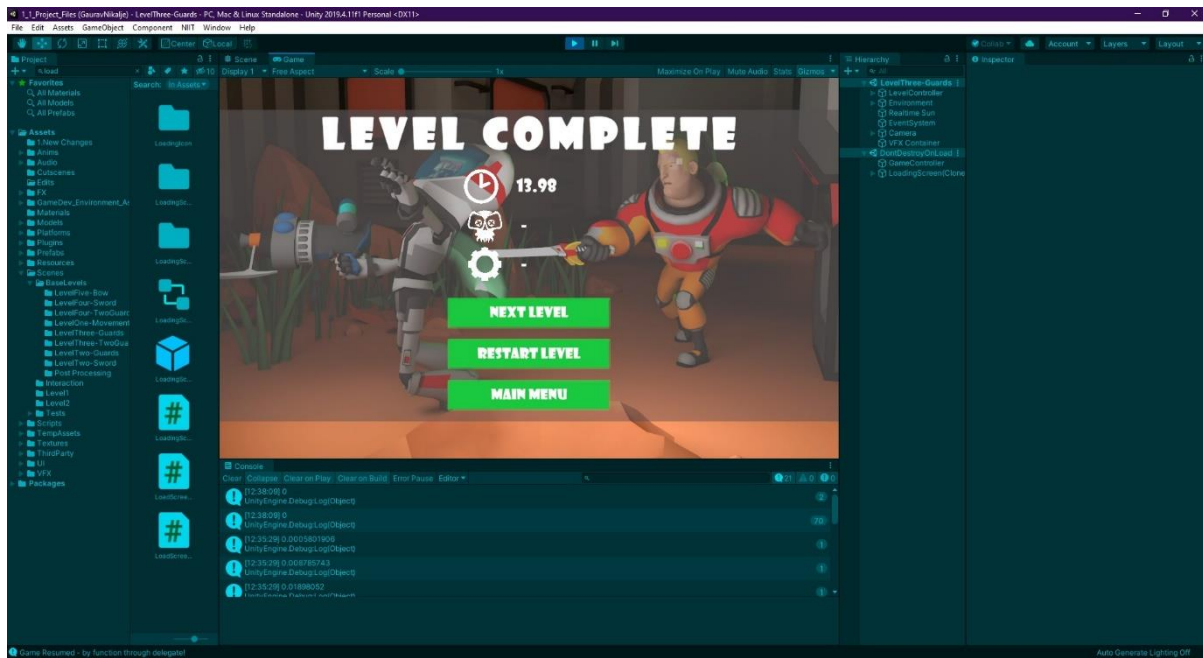
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using TMPro;
7 using UnityEngine;
8 using UnityEngine.SceneManagement;
9
10 2 references
11 public class LevelEndMenuController
12 {
13     11 references
14     private LevelEndMenu data;
15     2 references
16     public LevelStatsController levelStatsController;
17
18     2 references
19     public Action<Levels.Data> OnLevelLoadRequested = delegate { }; //Assignment - 02
20     2 references
21     public Action OnRetryRequested = delegate { }; //Assignment - 02
22     2 references
23     public Action OnExitRequested = delegate { }; //Assignment - 02
24
25     1 reference
26     public LevelEndMenuController(LevelEndMenu data, int currentLevelID, LevelStatsController levelStatsController) //Constructor
27     {
28         this.data = data;
29         this.levelStatsController = levelStatsController;
30         Hide();
31
32         GetNextLevel(currentLevelID);
33
34         data.retryButton.onClick.AddListener( () => //Assignment - 02 (for 'RestartButton' - Restarts)
35         {
36             OnRetryRequested(); //Invokes whenever & wherever it is called ( OnRetryRequested.Invoke(); )
37         });
38 }
```

- First, you declare delegate variables.
- Then, you need to call that delegate when you press the button by using AddListener().
- The button data is called from the “LevelEndMenu” script, which stores all the UI elements of the LevelEnd Menu.

❖ LevelEndMenu.

```
C# GameController.cs 1 • C# LevelController.cs C# LevelEndMenuController.cs • C# LevelEndMenu.cs X
Assets > Scripts > UI > LevelEnd > C# LevelEndMenu.cs > LevelEndMenu
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using UnityEngine;
7  using UnityEngine.UI;
8  using TMPro;
9  ⚡

10 public class LevelEndMenu : MonoBehaviour
11 {
12     public TextMeshProUGUI message;
13     public Transform scoreContainer;
14     public TextMeshProUGUI score;
15     public TextMeshProUGUI enemiesKilled;
16     public TextMeshProUGUI pickupsCollected;
17     public Button nextLevelButton;
18     public Button retryButton;
19     public Button exitToLevelSelectButton;
20 }
21
```



❖ Code to get NextLevel.

```

GameController.cs  LevelController.cs  UIController.cs  LevelEndMenuController.cs
Assets > Scripts > UI > LevelEnd > LevelEndMenuController.cs > LevelEndMenuController > LevelEndMenuController(LevelEndMenu data, int currentLevelID,
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66

data.exitToLevelSelectButton.onClick.AddListener(() => //Assignment - 02 (for 'MainMenuButton' - MainMenu)
{
    OnExitRequested();
});

2 references
public void Show(string message, bool levelComplete)
{
    data.gameObject.SetActive(true);

    data.message.text = message;
    data.score.text = levelStatsController.LevelTime.Value.ToString("0.00");
    data.enemiesKilled.text = "-";
    data.pickupsCollected.text = "-";

    if (!levelComplete)
    {
        data.scoreContainer.gameObject.SetActive(false);
        data.nextLevelButton.gameObject.SetActive(false);
    }
}

1 reference
private void GetNextLevel(int currentLevelID) //Assignment - 02 (NextLevel - Code Logic)
{
    for(int i = 0; i < Levels.ALL.Count; i++)
    {
        if(Levels.ALL[i].ID == currentLevelID)
        {
            if((i + 1) < Levels.ALL.Count)
            {
                FoundNextLevel(Levels.ALL[i + 1]);
                return;
            }
        }
    }
}

```

```
GameController.cs | LevelController.cs | UIController.cs | LevelEndMenuController.cs
Assets > Scripts > UI > LevelEnd > LevelEndMenuController.cs > LevelEndMenuController > LevelEndMenuController(LevelEndMenu

58     {
59         if(Levels.ALL[i].ID == currentLevelID)
60         {
61             if((i + 1) < Levels.ALL.Count)
62             {
63                 FoundNextLevel(Levels.ALL[i + 1]);
64                 return;
65             }
66         }
67     }
68
69     HideNextLevelButton();
70 }
71
72 1 reference
73 public void FoundNextLevel(Levels.Data levelData) //Assignment - 02
74 {
75     data.nextLevelButton.onClick.AddListener(() =>
76     {
77         OnLevelLoadRequested(levelData);
78     });
79 }
80 1 reference
81 private void HideNextLevelButton() //Assignment - 02
82 {
83     data.nextLevelButton.gameObject.SetActive(false);
84 }
85 1 reference
86 public void Hide()
87 {
88     data.gameObject.SetActive(false);
89 }
90 }
```

❖ UI Controller.

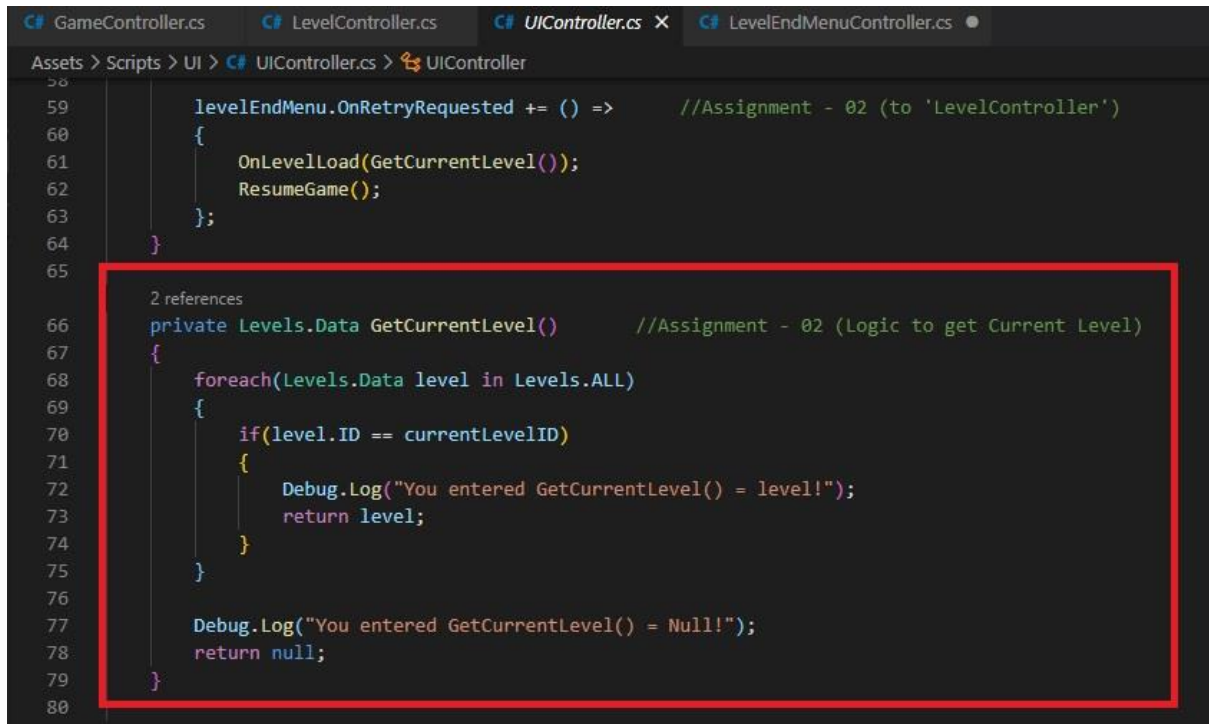
- We repeat the same steps of declaring a delegate variable, calling delegate function from the child script (in this case LevelEndMenuController) and passing the values.

```
GameController.cs x LevelController.cs UIController.cs x LevelEndMenuController.cs
Assets > Scripts > UI > UIController.cs > UIController
37
38 pauseMenuController.OnPause += () => { ShowPause(); Debug.Log("Using OnClick Button Event!"); };
39
40
41 pauseMenuController.OnRestart += () => //Assignment - 02 (for 'RestartButton' - to Restart)
42 {
43     OnLevelLoad(GetCurrentLevel());
44     ResumeGame();
45     Debug.Log("Game Restarted - by PauseMenu!");
46 };
47
48 levelEndMenu.OnLevelLoadRequested += (level) => //Assignment - 02 (to 'LevelController')
49 {
50     OnLevelLoad(level);
51     ResumeGame();
52 };
53
54 levelEndMenu.OnExitRequested += () => //Assignment - 02 (to 'LevelController')
55 {
56     OnExit();
57     ResumeGame();
58 };
59
60 levelEndMenu.OnRetryRequested += () => //Assignment - 02 (to 'LevelController')
61 {
62     OnLevelLoad(GetCurrentLevel());
63     ResumeGame();
64 };
65 }
```

```
GameController.cs x LevelController.cs UIController.cs x LevelEndMenuController.cs
Assets > Scripts > UI > UIController.cs > UIController
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5
6 public class UIController
7 {
8     private HUDController hudController;
9     private LevelEndMenuController levelEndMenu;
10    private Player player;
11    private TimeController timeController; //Assignment - 01
12    private LevelIntroUIController levelIntroController;
13    private int currentLevelID;
14    private PauseMenuController pauseMenuController;
15    public Action<Levels.Data> OnLevelLoad = delegate { }; //Assignment - 02 (Conatins <Levels.Data> bcoz to load level)
16    public Action OnExit = delegate { }; //Assignment - 02
17
18    public UIController(Player player, //Constructor
19        Transform cameraTransform, int currentLevelID, TimeController timeControl, LevelStatsController levelStatsController) //Assignment - 01
20    {
21        this.player = player;
22        this.currentLevelID = currentLevelID;
23        this.timeController = timeControl; //Assignment - 01
24
25        hudController = cameraTransform.Find("HUDCanvas/HUD").GetComponent<HUDController>();
26        hudController.SetupHUDController(player);
27    }
```

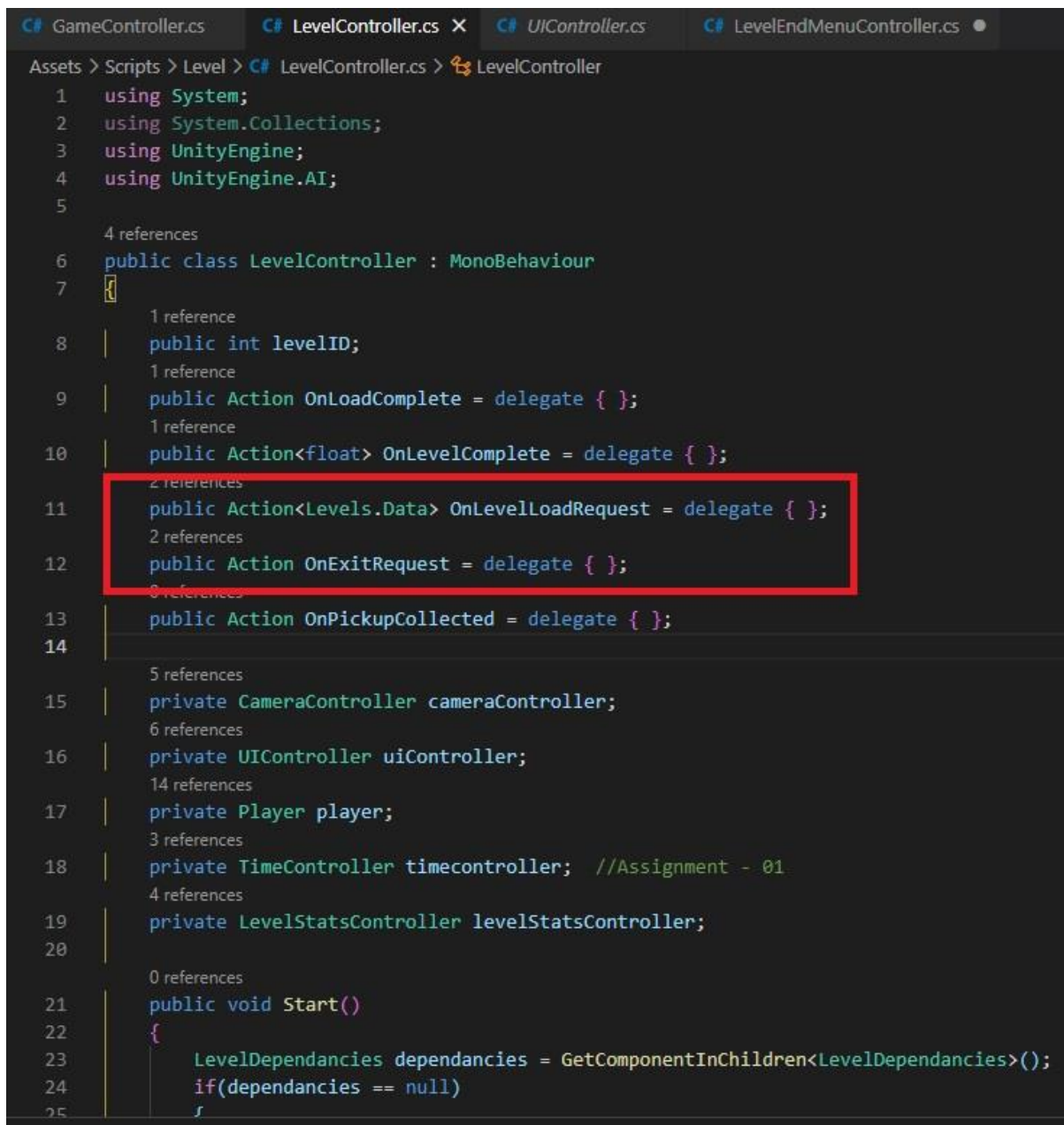

❖ Code to get CurrentLevel.

- We simply pass in a “Level.Data” variable – ‘level’, & check if it matches with the currentLevelID.
- And if it does, we return the level.



```
59 levelEndMenu.OnRetryRequested += () => //Assignment - 02 (to 'LevelController')
60 {
61     OnLevelLoad(GetCurrentLevel());
62     ResumeGame();
63 };
64 }
65
66 2 references
67 private Levels.Data GetCurrentLevel() //Assignment - 02 (Logic to get Current Level)
68 {
69     foreach(Levels.Data level in Levels.ALL)
70     {
71         if(level.ID == currentLevelID)
72         {
73             Debug.Log("You entered GetCurrentLevel() = level!");
74             return level;
75         }
76     }
77     Debug.Log("You entered GetCurrentLevel() = Null!");
78     return null;
79 }
80
```


❖ Level Controller.



```
1  using System;
2  using System.Collections;
3  using UnityEngine;
4  using UnityEngine.AI;
5
6  4 references
7  public class LevelController : MonoBehaviour
8  {
9      1 reference
10     public int levelID;
11     1 reference
12     public Action OnLoadComplete = delegate { };
13     1 reference
14     public Action<float> OnLevelComplete = delegate { };
15     2 references
16     public Action<Levels.Data> OnLevelLoadRequest = delegate { };
17     2 references
18     public Action OnExitRequest = delegate { };
19     0 references
20     public Action OnPickupCollected = delegate { };
21
22     5 references
23     private CameraController cameraController;
24     6 references
25     private UIController uiController;
26     14 references
27     private Player player;
28     3 references
29     private TimeController timecontroller; //Assignment - 01
30     4 references
31     private LevelStatsController levelStatsController;
32
33     0 references
34     public void Start()
35     {
36         LevelDependancies dependancies = GetComponentInChildren<LevelDependancies>();
37         if(dependancies == null)
38         {
39             //
40         }
41     }
42 }
```

```
GameController.cs | LevelController.cs | UIController.cs | LevelEndMenuController.cs
Assets > Scripts > Level > LevelController.cs > LevelController

48
49     levelStatsController = new LevelStatsController();
50     timecontroller = new TimeController(); //Concept: Hiding //Assignment - 01
51
52     HUDController hudController = cameraController.MainCameraTransform.Find("HUDCanvas/HUD").GetComponent<HUDController>();
53     player.Controller.OnPlayerDamageTaken += (currentHealth) => hudController.UpdatePlayerHealth(currentHealth);
54
55     uiController = new UIController(player, cameraController.MainCameraTransform,
56         levelID, timecontroller, levelStatsController);
57
58     uiController.OnLevelLoad += (level) => //Assignment - 02 (to 'GameController')
59     {
60         OnLevelLoadRequest(level);
61     };
62
63     uiController.OnExit += () => //Assignment - 02 (to 'GameController')
64     {
65         OnExitRequest();
66     };
67
68     _ = new LevelVFXController(dependencies.vfxLibrary, player.Controller);
69
70     OnLoadComplete();
71 }
72
73 0 references
74 public void OnLevelLoadResume()
75 {
76     timecontroller?.StartTime();
77     Debug.Log("OnLevelLoadResume Called!");
78 }
79
80 1 reference
81 private void EndLevel(EndLevelInteraction endLevel)
82 {
83     player.Broadcaster.EnableActions(ControlType.None);
84     CompleteLevel();
85 }
```

❖ Game Controller.

- Here we finally call all the delegate functions and put them into action.
- We first set an “IF” condition, to check and proceed further only if the levels in the Level Select Menu are unlocked.
- And if so, then we pass the level using the “Scene Path” of the scene while calling it from a Coroutine Function. (I used the coroutine function as I had to call the Slider & the Text Progress animations each time we load scenes)

```
GameController.cs x LevelController.cs UIController.cs LevelEndMenuController.cs
Assets > Scripts > GameController.cs > GameController

59 1 reference
60 private LevelSelectData CreateLevelData()
61 {
62     LevelSelectData levelSelectData = new LevelSelectData();
63     bool previousLevelCompleted = true;
64
65     foreach (Levels.Data level in Levels.All)
66     {
67         LevelSelectButtonData buttonData = new LevelSelectButtonData();
68
69         buttonData.name = level.name;
70
71         SaveData.Level levelData = saveData.levels.FirstOrDefault(x => x.ID == level.ID);
72
73         if (levelData != null)
74         {
75             buttonData.score = levelData.score;
76         }
77         else if (previousLevelCompleted)
78         {
79             previousLevelCompleted = false;
80         }
81         if(buttonData.locked == false) //Assignment - 02 (Unlocks Levels in 'Menu')
82         {
83             buttonData.OnClicked += () => StartCoroutine(OnSceneLoadRequested(level));
84         }
85         else
86         {
87             buttonData.locked = true;
88         }
89
90         levelSelectData.buttons.Add(buttonData);
91     }
92
93     return levelSelectData;
94 }
```

- Code that loads requested scene.

```
2 references
IEnumerator OnSceneLoadRequested(Levels.Data levelData) //Assignment - 02 (Loads a level using its 'Path')
{
    loadingScreenController.Show();
    AsyncOperation asyncOP = SceneManager.LoadSceneAsync(levelData.scenePath);
    asyncOP.allowSceneActivation = false;
}
```

- Calling the delegates and getting the passed value and passing it to MainMenu & NextLevel events.

```

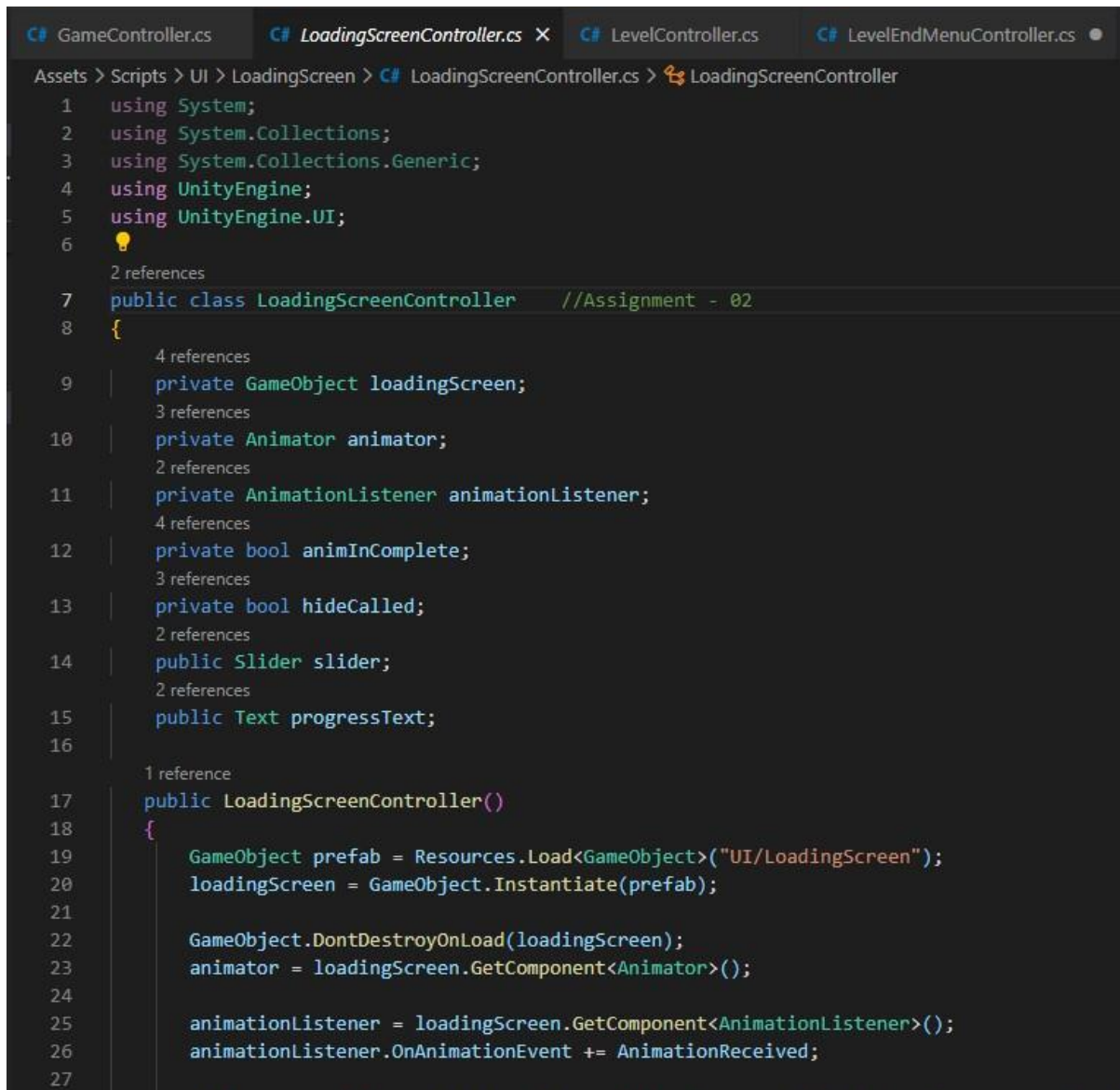
126
127 private void LoadMainMenu() //Assignment - 02
128 {
129     SceneManager.LoadScene("Menu");
130 }
131
132 1 reference
133 private void OnLevelLoaded(Scene scene)
134 {
135     LevelController levelController = FindObjectOfType<LevelController>();
136     if (levelController != null)
137     {
138         levelController.OnLevelLoadRequest += (level) => //Assignment - 02 (Passes a parameter to Load a Level)
139         {
140             StartCoroutine(OnSceneLoadRequested(level));
141         };
142
143         levelController.OnExitRequest += () => //Assignment - 02
144         {
145             LoadMainMenu();
146         };
147
148         Debug.Log("Level loaded");
149     }
150 }
151

```

❖ Load Screen.

- After finishing with assigning the events when the buttons are pressed, we need to load the “LoadScreen” that we just created to cover up the transition between two levels or scenes.
- So first, I create a “LoadingScreen” script.
- Then I wrote a code to Load the LoadScreen prefab using the “Resources.Load” and instantiated and set it to “DontDestroyOnLoad” so that it would be available in every scene or level.
- Then I added a reference for the AnimationListener script as it contains the animation event on the timelines of the “AnimIn” & “AnimOut” animations.
- Then I assigned a function called “AnimationReceived” to the animationListener reference variable.

❖ LoadingScreenController.



```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5 using UnityEngine.UI;
6
7 public class LoadingScreenController //Assignment - 02
8 {
9     private GameObject loadingScreen;
10    private Animator animator;
11    private AnimationListener animationListener;
12    private bool animIncomplete;
13    private bool hideCalled;
14    public Slider slider;
15    public Text progressText;
16
17    public LoadingScreenController()
18    {
19        GameObject prefab = Resources.Load<GameObject>("UI/LoadingScreen");
20        loadingScreen = GameObject.Instantiate(prefab);
21
22        GameObject.DontDestroyOnLoad(loadingScreen);
23        animator = loadingScreen.GetComponent<Animator>();
24
25        animationListener = loadingScreen.GetComponent<AnimationListener>();
26        animationListener.OnAnimationEvent += AnimationReceived;
27    }
28 }
```

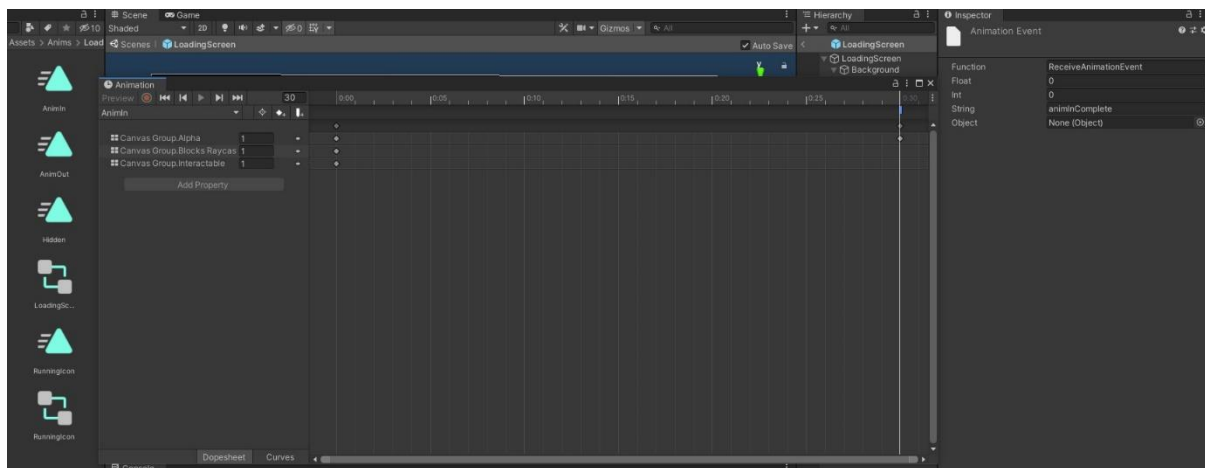
- Then I find or get the Slider & Text components using the “*FindObjectOfType*” function and assigned it to their respective variables.
 - **FindObjectOfType**: Used to find components on a GameObject.
- Note: I created public variables for the Slider & Text components as we would further access them in the “*GameController*” script.

```

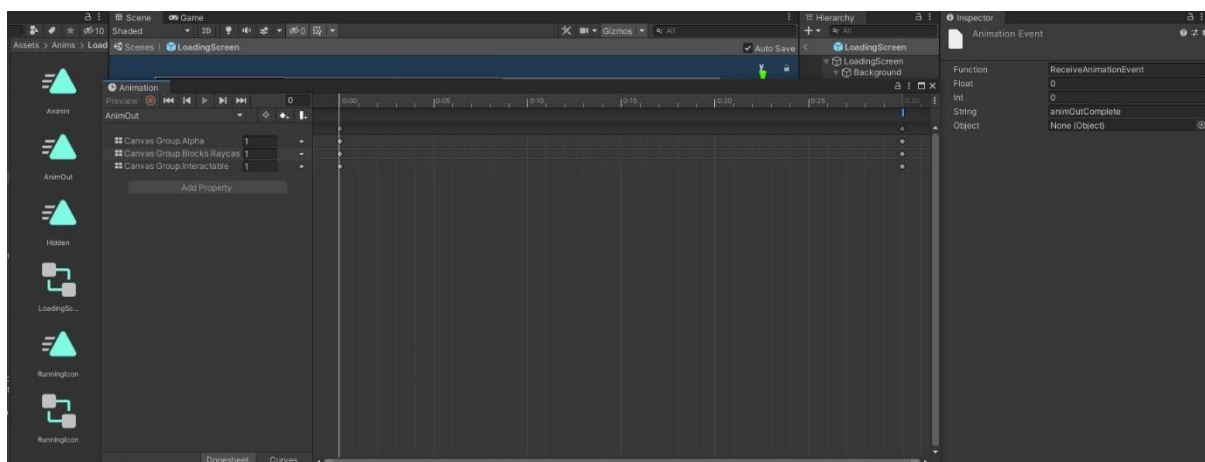
21
22     GameObject.DontDestroyOnLoad(loadingScreen);
23     animator = loadingScreen.GetComponent<Animator>();
24
25     animationListener = loadingScreen.GetComponent<AnimationListener>();
26     animationListener.OnAnimationEvent += AnimationReceived;
27
28     slider = GameObject.FindObjectOfType<Slider>(); //FindObjectOfType - Used to find Components/Objects (Use when GetComponent doesn't work!)
29     progressText = GameObject.FindObjectOfType<Text>();
30
31 }
32

```

- Next I wrote some code to Hide & UnHide the LoadScreen GameObject and Play the AnimIn & AnimOut animation.
- The code checks in for some Boolean values, as we do not want the transition (FadeIn-FadeOut) of the LoadScreen to be interrupted in between without completing the animation.



AnimIn



AnimOut


```
GameController.cs LoadingScreenController.cs X LevelController.cs LevelEndMenuController.cs
Assets > Scripts > UI > LoadingScreen > LoadingScreenController.cs > LoadingScreenController

1 reference
33 public void Hide()
34 {
35     hideCalled = true;
36     AttemptHide();
37 }
38

1 reference
39 public void Show()
40 {
41     hideCalled = false;
42     animInComplete = false;
43     animator.SetBool("show", true);
44 }
45

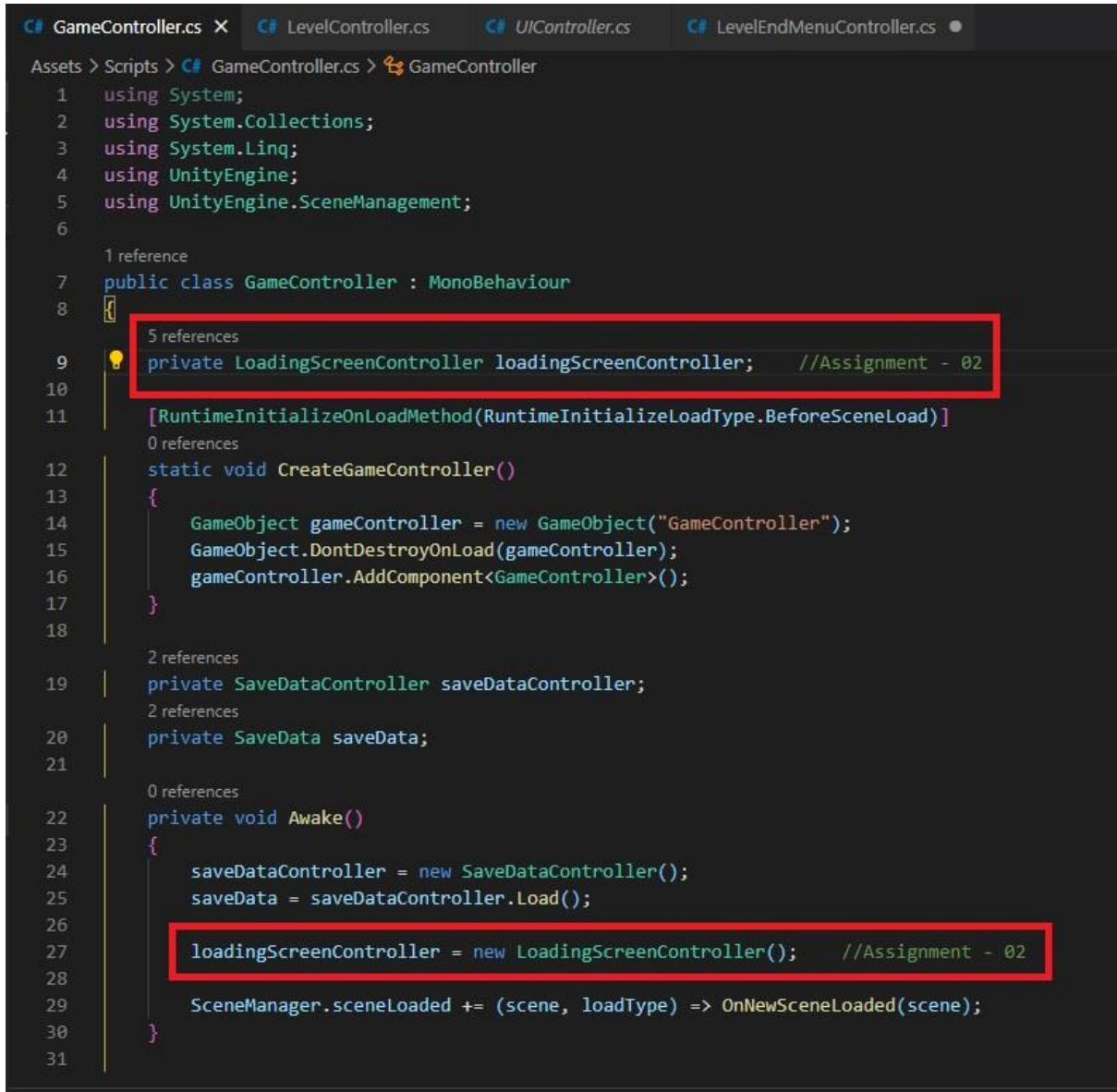
1 reference
46 private void AnimationReceived(string payload)
47 {
48     if(payload == "animInComplete")
49     {
50         animInComplete = true;
51         AttemptHide();
52     }
53     else if(payload == "animOutComplete")
54     {
55         animInComplete = false;
56     }
57 }
58

2 references
59 private void AttemptHide()
60 {
61     if(animInComplete && hideCalled)
62     {
63         animator.SetBool("show", false);
64     }
65 }
66
```

LoadingScreenController

❖ Back to Game Controller.

- Then after finishing with the “LoadingScreenController” script, I added a reference for it in the “GameController” script and created an instance of it.



```
Assets > Scripts > GameController.cs > GameController
1  using System;
2  using System.Collections;
3  using System.Linq;
4  using UnityEngine;
5  using UnityEngine.SceneManagement;
6
7  1 reference
8  public class GameController : MonoBehaviour
9
10     5 references
11     private LoadingScreenController loadingScreenController; //Assignment - 02
12
13     [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]
14     0 references
15     static void CreateGameController()
16     {
17         GameObject gameController = new GameObject("GameController");
18         GameObject.DontDestroyOnLoad(gameController);
19         gameController.AddComponent<GameController>();
20     }
21
22     2 references
23     private SaveDataController saveDataController;
24     2 references
25     private SaveData saveData;
26
27     0 references
28     private void Awake()
29     {
30         saveDataController = new SaveDataController();
31         saveData = saveDataController.Load();
32
33         loadingScreenController = new LoadingScreenController(); //Assignment - 02
34
35         SceneManager.sceneLoaded += (scene, loadType) => OnNewSceneLoaded(scene);
36     }
37
38 }
```

- Then I added some code in the “OnSceneLoadRequested()” function which enables and disables the LoadScreen by accessing the “Hide()” and “Show()” function from the “LoadingScreenController” script.
- In that, I added a code that converts the Slider Bar’s value from 0 to 1, instead of 0 to 0.9f.
- I did that by using the “Mathf.Clamp1()” method which converts the values ranged between 0 to 1.

- Then I assigned it to a progress float, which is then further assigned to the slider's value (slider.value).
- Then I did same with the Text that would show the Level Loading progress in percentage(%).
- I simply multiplied the progress value with 100 to achieve the percentage value & added a "%" string in the end.

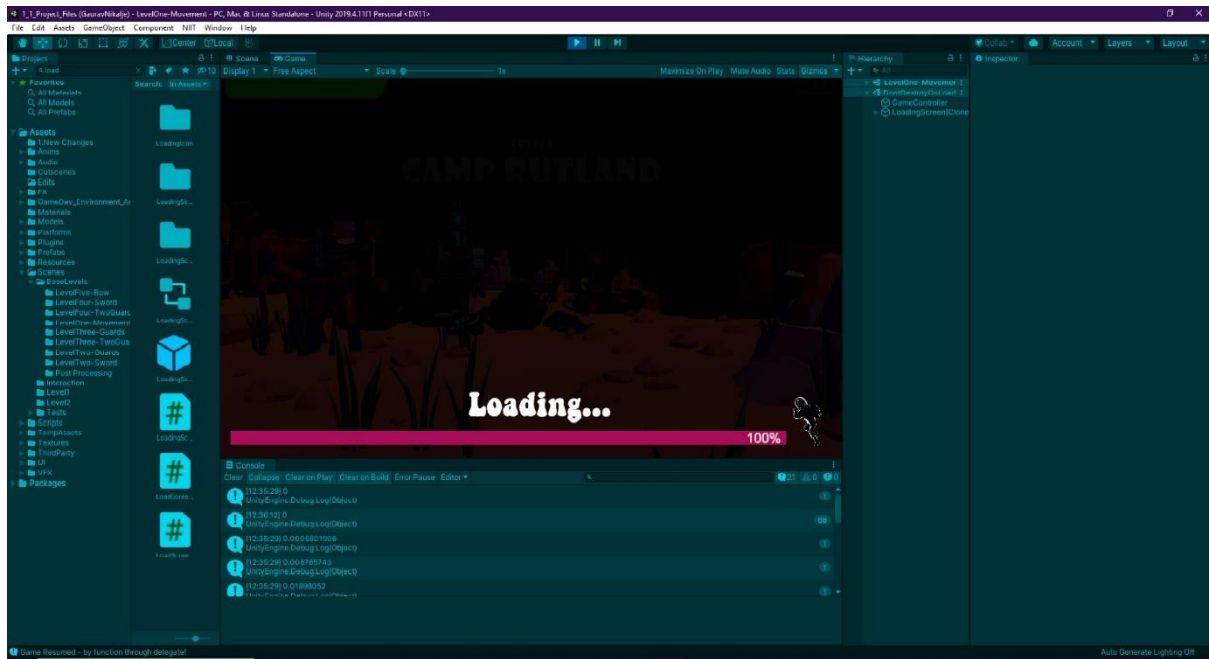
```

C# GameController.cs X C# LevelController.cs C# UIController.cs C# LevelEndMenuController.cs
Assets > Scripts > C# GameController.cs > GameController

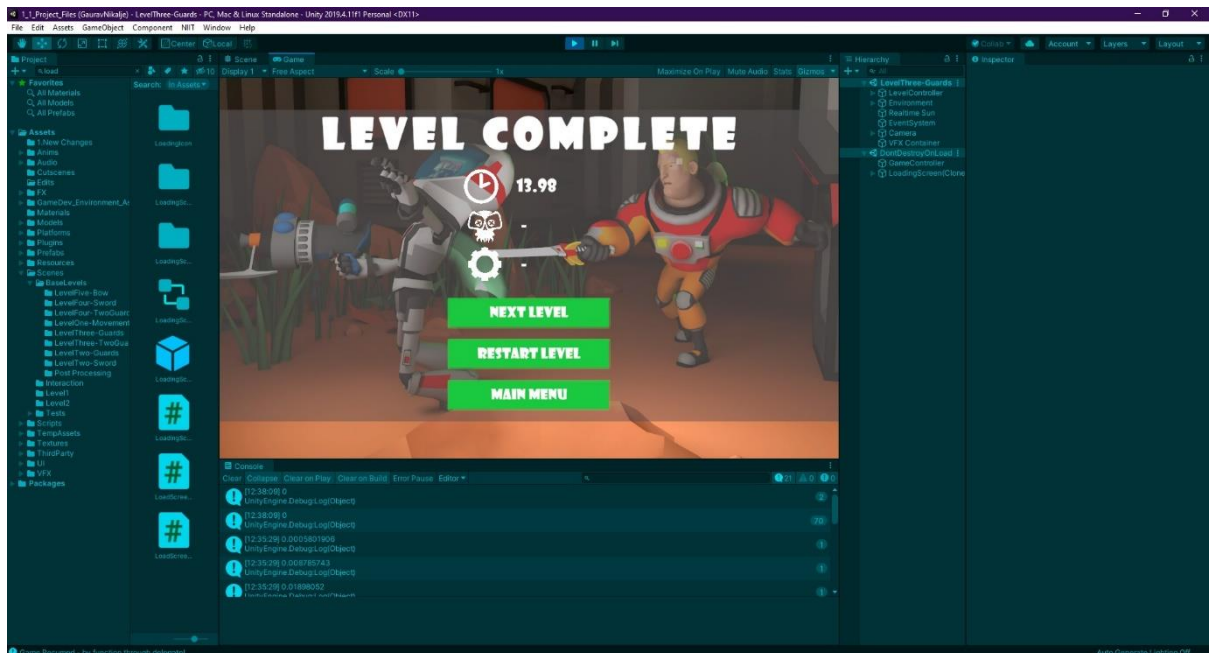
2 references
96 IEnumerator OnSceneLoadRequested(Levels.Data levelData) //Assignment - 02 (Loads a level using its 'Path')
97 {
98     loadingScreenController.Show();
99     AsyncOperation asynOP = SceneManager.LoadSceneAsync(levelData.scenePath);
100     asynOP.allowSceneActivation = false;
101
102     /* asynOP.completed += (op) =>          //op - bcoz asynOP does not take 0 argument
103     {
104         loadingScreenController.Hide();
105     }; */
106
107     while(!asynOP.isDone) //
108     {
109         //if(loadingScreenController != null)
110         //{
111             float progress = Mathf.Clamp01(asynOP.progress / 0.9f);
112             loadingScreenController.slider.value = progress;
113             loadingScreenController.progressText.text = progress * 100 + "%";
114             Debug.Log(progress);
115             asynOP.allowSceneActivation = true;
116         //}
117         yield return null;
118     }
119
120     loadingScreenController.Hide();
121
122     //LevelController levelController = new LevelController();
123     //levelController?.OnLevelLoadResume(); //Time.timeScale = 1;
124     Time.timeScale = 1;
125 }
126
11 reference
127 private void LoadMainMenu() //Assignment - 02
128 {
129     SceneManager.LoadScene("Menu");
130 }
131

```

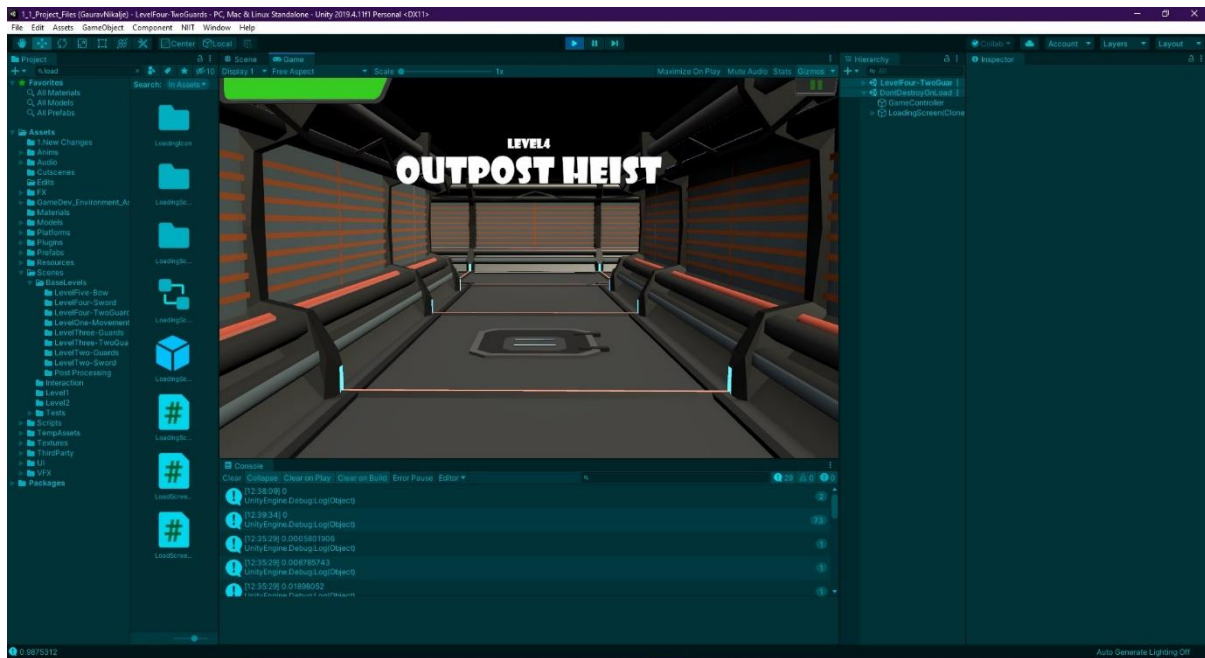
❖ Final Result



Loading Screen



Level Complete Menu



Next Level Loaded

Step 03: What have I learnt

- **What are synchronous and asynchronous operations. What are the benefits and drawbacks of each?**
 - There are two ways of Loading a scene in Unity
 - *Synchronous*
 - *Asynchronous*
 - **Synchronous operation:**
 - It is the simplest way of loading scenes.
 - Synchronous means that Unity will close the First Scene and then load the Next Scene.
 - Although this sometimes end up freezing or crashing the game, as the unloading of the First Scene leaves the Game entirely with No Scene or Empty Scene. And if there is some problem with the Next Scene like there are some components missing in that scene, or Unity fails to load them somehow, then the Game crashes or freezes.
 - **SceneManager.LoadScene(sceneName)**

- **Asynchronous operation:**
 - They are useful to track progress of loading scenes.
 - Asynchronous means that Unity will load the Next Scene, but also keep the First Scene available for some time, until the entire Next Scene is loaded.
 - It has a callback, "*AsyncOperation*" which is used to store the scene load function.
 - It is helpful as this information can be used to create progress bars and know how much time will it take to load Next Scene.
 - It also has some of its own properties.
 - ***isDone***: Checks whether the loading process is done or not (i.e., completed or not).
 - ***completed***: Used to pass logic after the scene is finished loading.
 - ***progress***: Is used to track the progress of scene loading.
 - ***allowSceneActivation***: Passes a Boolean True or False value.
 - **SceneManager.LoadSceneAsync(sceneName)**

- **Why are loading screens used in games?**

- Loading Screens are used in Games to cover the loading mess that is being executed in the background while the game switches from one scene/level to another.
- Because the Player should not necessarily need to see what is happening behind the game.
- Loading a new scene or level takes a specific amount of time, depending on the computer.
- To keep a track of the amount of time required to load next level, there is a concept of Progress Bars in Games which show the progress of the level load in Percentages as well as Bars.
- And also sometimes "SlideShows" of Game Images and "Game Hints" are used in Loading Screens.

- **Give a scenario where using multiple additive scenes could be useful?**

- *“Additive Scene Loading”* is a method where a New Scene is loaded or added in an existing scene.
- It is used to load multiple scenes or levels at a time.
- It is useful in OpenWorld Games or in games where there is a Huge Map for the Player to explore and activities or tasks to perform.
- In that case, Additive Scene Loading can be used to load different scenes, as the Player reaches or moves towards particular directions.
- There is also a method to UnLoad scenes, where you can unload multiple scenes, so as to save memory.
- Examples in Games where this concept might have been used are GTA Series, Assassin’s Creed Series, Fallout, etc...
- There is a concept in Unity called *“Occlusion Culling”* which disables the GameObjects or does not render them when they are not seen by the camera.
- This method can be used in Unity while creating a Open World Game.

-----**THE END**-----