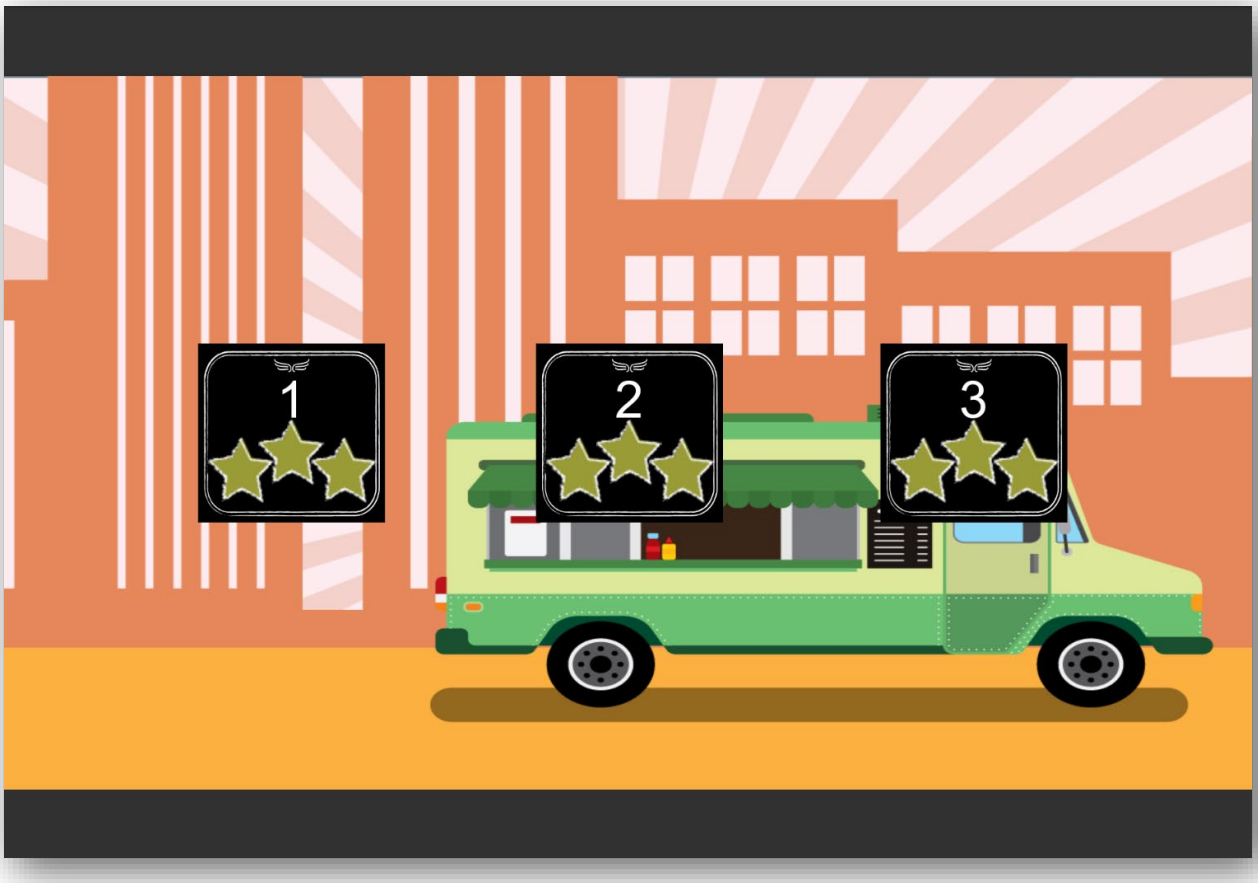
CODE NINJAS®

Silver Belt Ninja Guide
Activity 17: Food Frenzy
Part 2

Activity 17

Food Frenzy Part 2

Well done! Now it's time for the finishing touches on our final Silver Belt game! The interface isn't quite done yet. There needs to be something after the game has finished to either let the player **play again or continue**, in other words, the **Game Over** screen. These next steps create the **Game Over** object to do that. We'll also be creating a **Level Select** to let players choose what level they want to play.

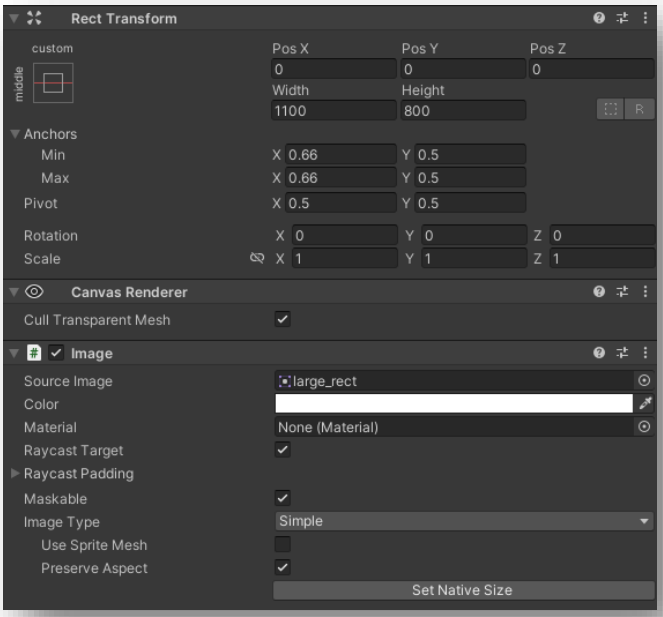


1 We'll continue using the same project from Part 1. Re-open this project if you have closed it, or use the Part 2 package.

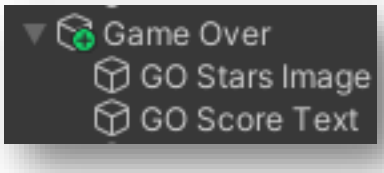
2 Inside the **Hierarchy**, add a **UI > Image** object. Rename this object "**Game Over**".

In the **Inspector**, find the **Image** component and set the **Source Image** to **large_rect**.

Keep the image centered and adjust the image so that it fills most of the screen. (We used a width of 1100 units and a height of 800 units).



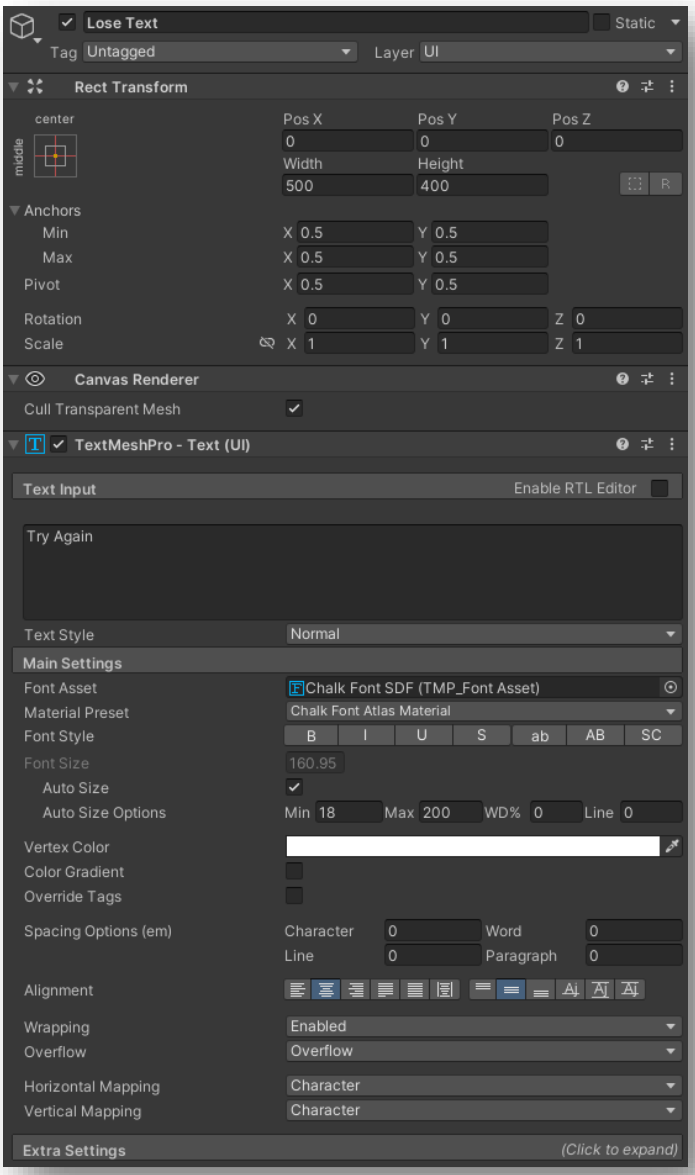
3 Make a **duplicate** of the **Score's Stars Image and Score Text** objects and place the duplicate inside **Game Over**. Rename the objects to **GO Stars Image** and **GO Score Text**. Adjust the new **Score** elements so that it is in the center of the **Game Over** object.



4 If the player loses, we won't show the stars or score, just a **message**. Inside the **Screen** object, create a **UI > Text - TextMeshPro** object and name it "**Lose Text**". Change the **color** of the text to white, the font to **Chalk**. **Resize** it so that it fits in the center of the **Screen** background.

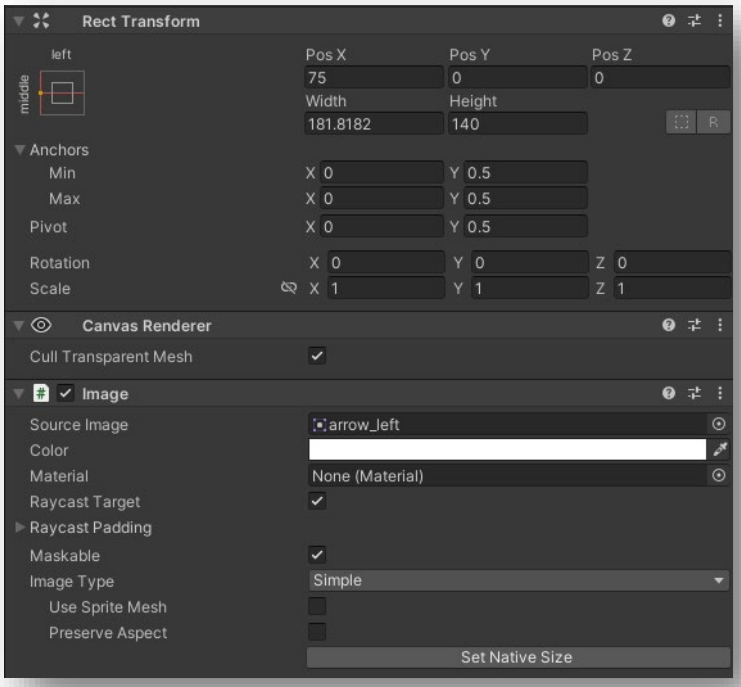
Do not worry if it is hard to see in front of the stars, we won't be showing the stars if the player loses. Change the **text** to "**Try Again**".


00
REMAINING MOVES
00
TARGET SCORE
00000
TRY
AGAIN
00000

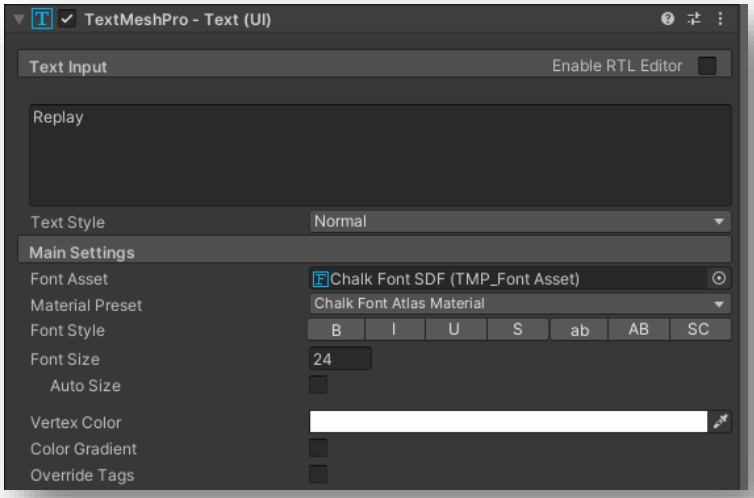

Lose Text
Static
Tag Untagged
Layer UI
Rect Transform
center
middle
Pos X
Pos Y
Pos Z
0
0
0
Width
Height
500
400
R
Anchors
Min
X 0.5
Y 0.5
Max
X 0.5
Y 0.5
Pivot
X 0.5
Y 0.5
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Canvas Renderer
Cull Transparent Mesh
TextMeshPro - Text (UI)
Text Input
Enable RTL Editor
Try Again
Text Style
Normal
Main Settings
Font Asset
Chalk Font SDF (TMP_Font Asset)
Material Preset
Chalk Font Atlas Material
Font Style
B
I
U
S
ab
AB
SC
Font Size
160.95
Auto Size
Auto Size Options
Min 18
Max 200
WD% 0
Line 0
Vertex Color
Color Gradient
Override Tags
Spacing Options (em)
Character 0
Word 0
Line 0
Paragraph 0
Alignment
Wrapping
Enabled
Overflow
Overflow
Horizontal Mapping
Character
Vertical Mapping
Character
Extra Settings
(Click to expand)

5 Our screen needs buttons to give the user choices on what to do next.

Add a **UI > Button - TextMeshPro** to the **Screen** object and name it "**Replay Button**". The **button** object comes with its own **sprite** and **text**, but we can change that. Find the **Image** component in the **Inspector** and replace the **Source Image** with **arrow_left** from the assets.



Select the **text** object that is inside the **Replay Button** object. Change the **text color** to white and the **text** to "**Replay**". Resize the arrow so that the text can be seen, then position it on the left side of the screen.

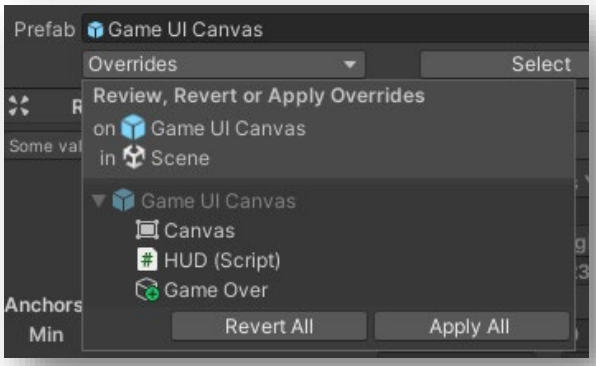

TextMeshPro - Text (UI)
Text Input
Enable RTL Editor
Replay
Text Style
Normal
Main Settings
Font Asset
Chalk Font SDF (TMP_Font Asset)
Material Preset
Chalk Font Atlas Material
Font Style
B
I
U
S
ab
AB
SC
Font Size
24
Auto Size
Vertex Color
Color Gradient
Override Tags


00
REMAINING MOVES
00
TARGET SCORE
00000
TRY
AGAIN
00000
REPLAY

6 Duplicate the **Replay Button** object and change the name to "**Done Button**". Replace the **Source Image** with **arrow_right** and change the **text** to "**Done**". Move this button to the right side of the **Screen** object as shown below.

7 Update the **Game UI Canvas** prefab so that the changes are applied to all scenes. You can do this by pressing the "**Override**" button at the top-right with the **Game UI Canvas** selected.

You will see a list of changes that have been made to the prefab. Make sure they're correct and click **Apply All**.

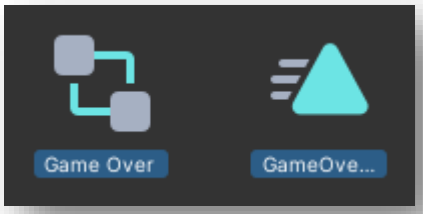


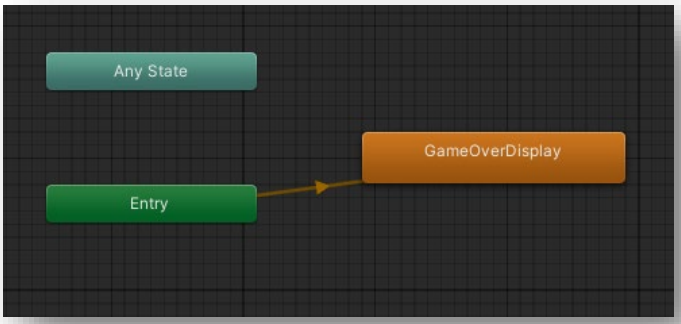
Next, we will have a little fun with how this object appears.

8 We could just have the game over screen show up, but where's the fun in that?

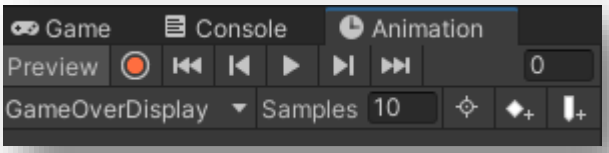
In the **Animations** folder, create a new animation by right-clicking on an empty space in the folder, and going to **Create > Animation**. Call it "**GameOverDisplay**". Add this component to the **GameOver** object in the Hierarchy by dragging it into the **Inspector**.

9 Open the **GameOver** controller. This is set to play the animation immediately.

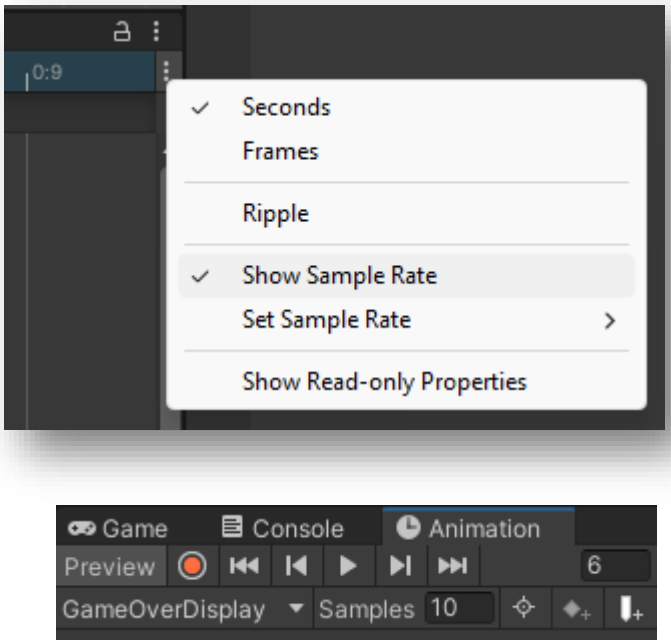




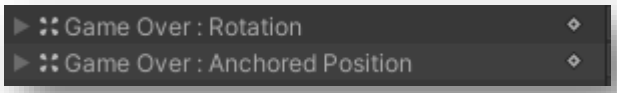
10 Now that the animator is set up, let's make our animation. In the **Animations** folder, double-click on **GameOverDisplay** to open the animation window. With the window open, confirm that you're editing the **GameOverDisplay** animation by selecting the **GameOverObject** in the **Hierarchy**.



11 First, we'll change the sample rate of the animation. Click the three dots in the top right of the animation window and enable the sample rate. Change the sample rate to 10.

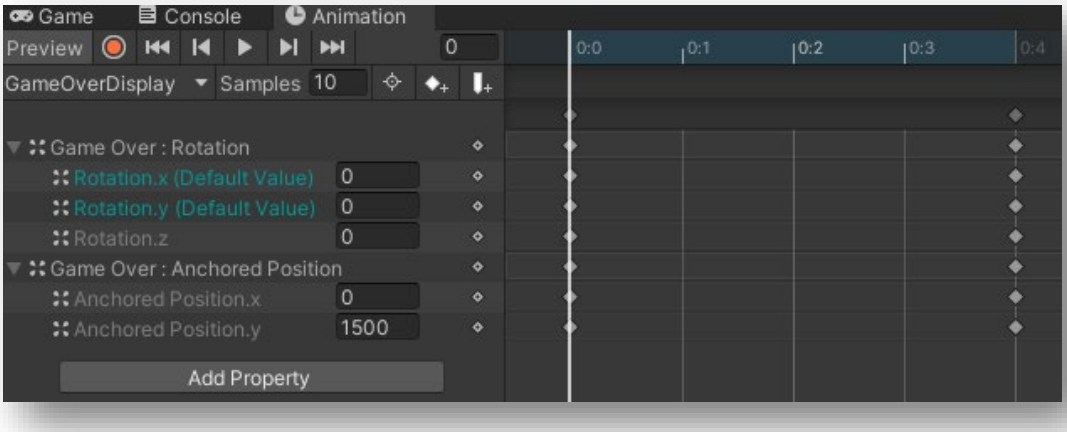


12 Click on **Add Property** and select **Screen > Rect Transforms > Rotation**. Repeat these steps to add **Screen > Rect Transforms > Anchored Position**.

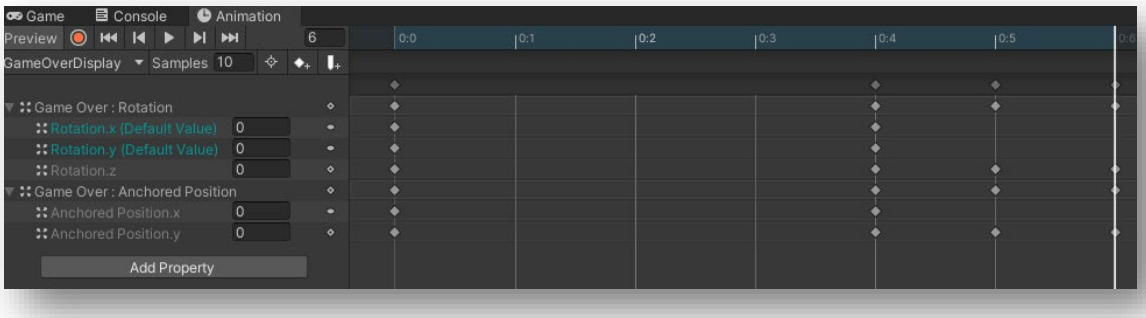


13 Move the last key frame to 0:4 by highlighting and dragging the dots to the column. Expand the properties by clicking on the triangle next to each of the properties.

14 We want to start with the **Game Over** object beyond the top of the canvas. Make sure the scrubber is at the start, then click on the **Anchored Position.y** property and give it a positive number, like 1500.



15 Let's give the screen a little shake when it plays. Move the playback line to the **fifth** frame, then set **Rotation.z** to 10 and Position.y slightly higher, like 100. Then at the **sixth** frame, set the **Rotation.z** to 0.



16 Move the playback line back and forth (this is known as "scrubbing") to see the animation. Make sure the **Preview** button is on in the **Animation** window.

17 Select the **Game UI Can**vas in the **Hierarchy** and **apply the Overrides** to the prefab so that the animation is the same for all scenes. The next step is writing the script that controls this animation.

18 In the **Scripts** folder, create a **new C# script** and give it the name "**GameOver**". Add this script to the **GameOver** object in the **Hierarchy**. Open the script.

19 To start, this Script needs some additional **directives** to handle everything. At the top of the script, add:

```
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using TMPro;
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using TMPro;
```

20 For the script to work with the objects, we'll need variables for the **text**, **image** and **button** objects:

```
public Image starsImage;
public TMP_Text scoreText;
public TMP_Text loseText;

public Button replayButton;
public Button doneButton;
```

```
public class GameOver : MonoBehaviour
{
    public Image starsImage;
    public TMP_Text scoreText;
    public TMP_Text loseText;

    public Button replayButton;
    public Button doneButton;
```

21 Next, because there is only one Game Over screen, this is a great opportunity for another singleton! Create a variable for the static instance.

```
public static GameOver instance;
```

22 When the game starts, we want all objects to be hidden. We also need to assign the singleton's instance. In the `Start()` function, add this:

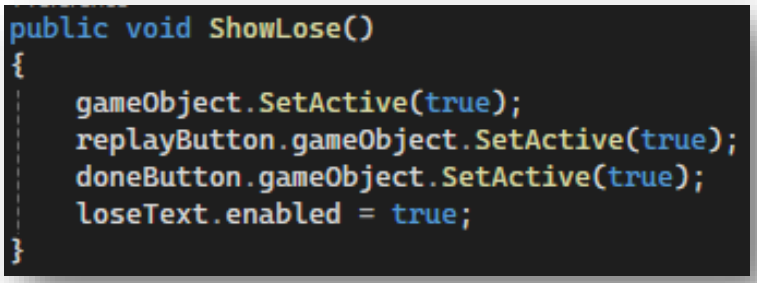
```
if (!instance)
{
    instance = this;
    gameObject.SetActive(false);
    starsImage.enabled = false;
    scoreText.enabled = false;
    loseText.enabled = false;
    replayButton.gameObject.SetActive(false);
    doneButton.gameObject.SetActive(false);
}
```

```
void Start()
{
    if (!instance)
    {
        instance = this;
        gameObject.SetActive(false);
        starsImage.enabled = false;
        scoreText.enabled = false;
        loseText.enabled = false;
        replayButton.gameObject.SetActive(false);
        doneButton.gameObject.SetActive(false);
    }
}
```

23

Next, we need a **function** for when the player doesn't get enough points to earn a star. In this function, we'll show the screen and the lose text, then hide the score object before playing the animation:

```
public void ShowLose()
{
    gameObject.SetActive(true);
    replayButton.gameObject.SetActive(true);
    doneButton.gameObject.SetActive(true);
    loseText.enabled = true;
}
```



24

We will need to do the same for when the player gets one or more stars. In this case, we'll hide the lose text and show the score object.

When this function is triggered, we'll need parameters for the score and the number of stars. However, we won't show the score immediately (we'll explain why in a moment).

Our `ShowWin` function will look like this:

```
public void ShowWin(int score, int starCount)
{
    gameObject.SetActive(true);
    starsImage.enabled = true;
    scoreText.text = score.ToString();
}
```

```
public void ShowWin(int score, int starCount)
{
    gameObject.SetActive(true);
    starsImage.enabled = true;
    scoreText.text = score.ToString();
}
```

25

At the moment, the object isn't showing the score or the stars. We want them to appear one at a time with a pause. Normally, C# does everything at once. If we want a pause, we will need a **coroutine**. In the `ShowWin()` function, add this at the end:

```
StartCoroutine(ShowWinCoroutine(starCount));
```

```
public void ShowWin(int score, int starCount)
{
    gameObject.SetActive(true);
    starsImage.enabled = true;
    scoreText.text = score.ToString();

    StartCoroutine(ShowWinCoroutine(starCount));
}
```

`ShowWinCoroutine()` doesn't exist yet, so let's make it.

26

A **coroutine** acts a lot like a **function**, but it needs a **special syntax** to work. Any **coroutine** needs to be called with **IEnumerator**:

```
private IEnumerator ShowWinCoroutine(int starCount){}
```

```
56    private IEnumerator ShowWinCoroutine(int starCount)
57    {
58
59    }
```

27

Then we'll set up a loop to show the earned stars one by one:

```
for (int i = 0; i <= starCount; i++)
{
    yield return new WaitForSeconds(0.5f);
    starsImage.sprite = HUD.instance.stars[i];
}
```

```
private IEnumerator ShowWinCoroutine(int starCount)
{
    for (int i = 0; i <= starCount; i++)
    {
        yield return new WaitForSeconds(0.5f);
        starsImage.sprite = HUD.instance.stars[i];
    }
```

Notice the WaitForSeconds(0.5f) is causing a 0.5 second delay between stars, and that the images are coming from the HUD's star array!

28

After each star is shown, we'll use the code below to display the score and buttons:

```
scoreText.enabled = true;
replayButton.gameObject.SetActive(true);
doneButton.gameObject.SetActive(true);
```

```
private IEnumerator ShowWinCoroutine(int starCount)
{
    for (int i = 0; i <= starCount; i++)
    {
        yield return new WaitForSeconds(0.5f);
        starsImage.sprite = HUD.instance.stars[i];
    }

    scoreText.enabled = true;
    replayButton.gameObject.SetActive(true);
    doneButton.gameObject.SetActive(true);
}
```

29 Finally, we need functions for when the **Replay** and **Done** buttons are clicked. If the player clicks **Replay**, we'll reload the current scene by using `SceneManager`:

```
public void OnReplayClicked()
{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildI
ndex);
}
```

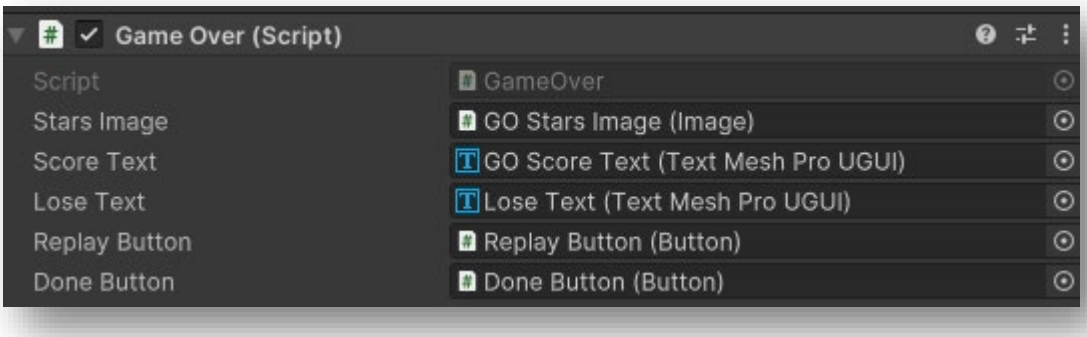
```
public void OnReplayClicked()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
}
```

30 We want to do the same for the **Done** button, but that scene hasn't been created yet. We'll leave that blank for now:

```
public void OnDoneClicked() { }
```

```
83      public void OnDoneClicked()
84      {
85
86      }
```

31 **Save** this script and go back to Unity. Select the **GameOver** object to link the objects and images to the variables:



32 We could manually assign the clicked events in the inspector, but it might be easier to do this using code. In the **Start** method, add these two lines to add the **OnReplayClicked** and **OnDoneClicked** methods to the **OnClicked** event for each button.

```
replayButton.onClick.AddListener(OnReplayClicked);
doneButton.onClick.AddListener(OnDoneClicked);
```

```
if (!instance)
{
    instance = this;
    gameObject.SetActive(false);
    starsImage.enabled = false;
    scoreText.enabled = false;
    loseText.enabled = false;
    replayButton.gameObject.SetActive(false);
    doneButton.gameObject.SetActive(false);
    replayButton.onClick.AddListener(OnReplayClicked);
    doneButton.onClick.AddListener(OnDoneClicked);
}
```

33 Once again, select the **Game UI Canvas** in the **Hierarchy** and apply the override to the prefab.

34 The **Replay** and **Done** buttons may be connected, but the game over functions still need to be called. Open the **HUD** script.

35 We already have the functions for `OnGameWin()` and `OnGameLose()` in the script. Now we need to use them to call the correct function in the **GameOver** script. In the `OnGameWin()` function, replace the `isGameOver` code with this code:

```
GameOver.instance.ShowWin(score, starIndex);
```

```
public void OnGameWin(int score)
{
    GameOver.instance.ShowWin(score, starIndex);
}
```

36 We'll make a similar change in the `OnGameLose()` function:

```
GameOver.instance.ShowLose();
```

```
public void OnGameLose()
{
    GameOver.instance.ShowLose();
}
```

Unlike `ShowWin()`, which takes parameters, `ShowLose()` doesn't need any.

37 **Save** the script and go back to Unity. Now **test** the game to make sure that the **GameOver** screen shows up at the end. Is something wrong?

38 The screen is behind the game! This is because the layer order of the canvas is behind the game objects. Add this line to OnGameWin and OnGameLose in the HUD script to move the UI to the front.

```
GetComponent<Canvas>().sortingOrder = 3;
```

```
public void OnGameWin(int score)
{
    GetComponent<Canvas>().sortingOrder = 3;
    GameOver.instance.ShowWin(score, starIndex);
}

1 reference
public void OnGameLose()
{
    GetComponent<Canvas>().sortingOrder = 3;
    GameOver.instance.ShowLose();
}
```

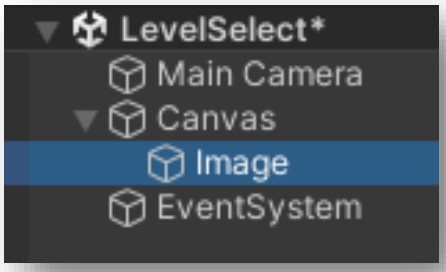
39 The settings for the star scores and other parameters for each level have already been placed in the script attached to the **Level** object in each scene. If you think the game is too easy or too hard, you can modify these numbers.

You might have noticed that the **Done** button doesn't do anything.

40 To complete the game, we need a menu to start from and return to after each game is finished. This menu will let the player choose any level they want to play.

Go to the **Scenes** folder, and right-click to create a scene. Name it "**LevelSelect**".

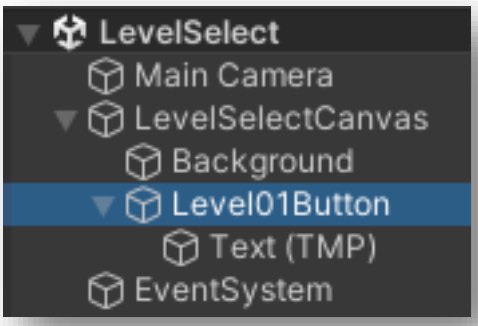
41 Use **UI > Image** to add an image to the scene. When you do that, Unity automatically adds a canvas and an event system.



42 Rename the **Canvas** to **LevelSelectCanvas** and the image to **Background**. Set up the Canvas and the background similar to the Canvas from the other scenes. This includes:

- Changing the Canvas' **Render Mode** to **Screen Space – Camera**
- Changing the Cavas Scaler's **UI Scale Mode** to **Scale With Screen Size** and **Reference Resolution** to **1920x1080**
- The background image to **FoodBG001**
- Preserve the **Aspect Ratio** of the background image
- Add an **Aspect Ratio Fitter** to the background
- Set the Apect Ratio Fitter's **mode** to **Envelope**, and **ratio** to **1.41**

43 A menu needs buttons. In the **Canvas**, add a **UI > Button - TextMeshPro** to add a button and name it "**Level01Button**".

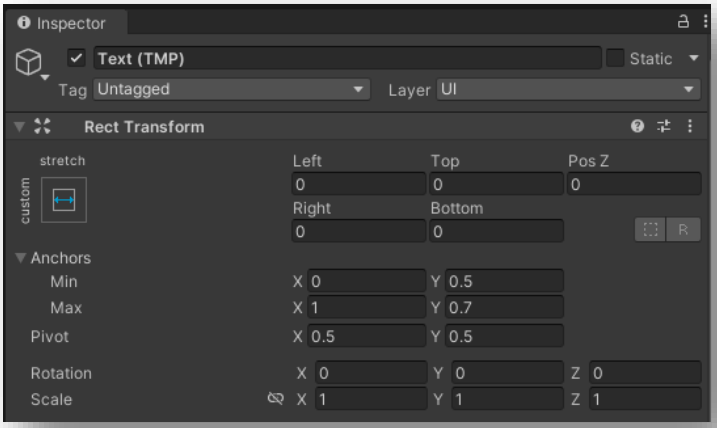


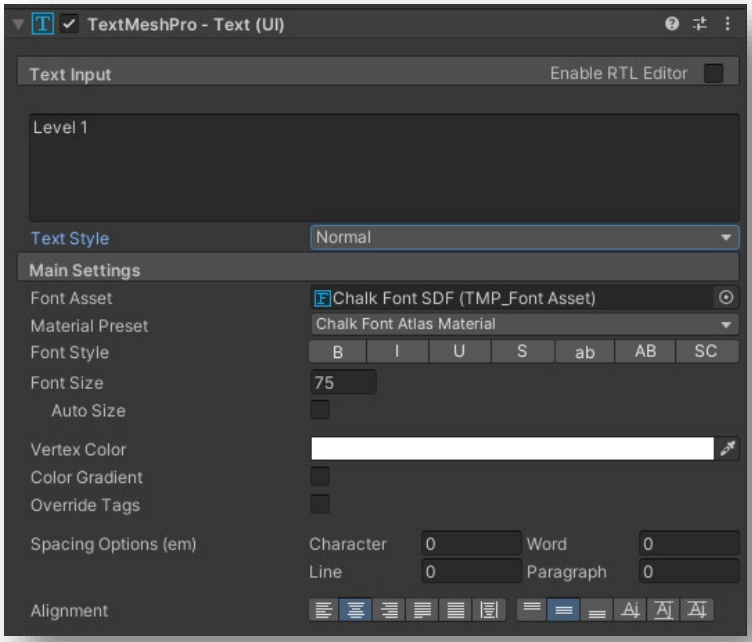
44

Change the **Source Image** of the button to **small_square**, preserve the aspect ratio, and set the width and height to 500.

Change the button **Text** to “Level 1”, change the **color** to white, the font to Chalk, and make it large enough to fill the upper half of the button. A font size of 75 should work.

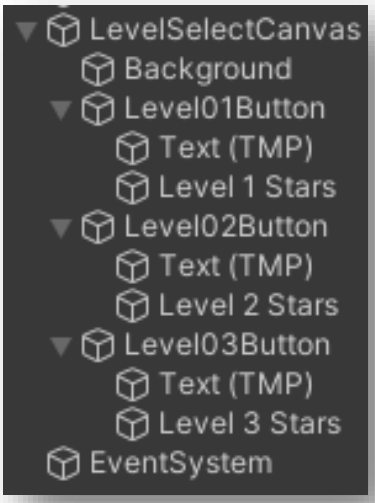
If the number disappears, adjust the size of the object rectangle until the number is visible.



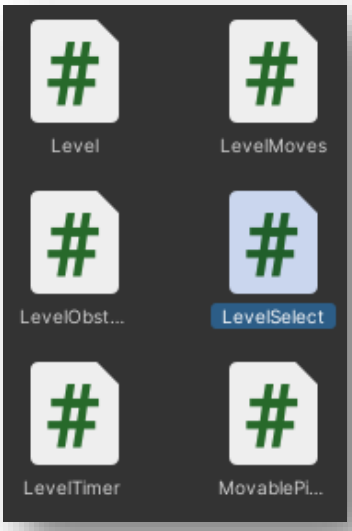

TextMeshPro - Text (UI)
Text Input
Enable RTL Editor
Level 1
Text Style
Normal
Main Settings
Font Asset
Chalk Font SDF (TMP_Font Asset)
Material Preset
Chalk Font Atlas Material
Font Style
B
I
U
S
ab
AB
SC
Font Size
75
Auto Size
Vertex Color
Color Gradient
Override Tags
Spacing Options (em)
Character
0
Word
0
Line
0
Paragraph
0
Alignment

45 In the button object, add **UI > Image**. Change the **Source Image** to **stars_0** and name it "**Level 1 Stars**". Adjust the image so that it fits under the number text.

46 Currently this game has three levels so we will need three buttons. Select **Level01Button** and **duplicate** it twice. Move **Level01Button** to the left and the new buttons to the right of **Level01Button** and rename them as "**Level02Button**" and "**Level03Button**". Change the **button text** for each of the buttons to the appropriate number, adjusting the size of the **Text** rectangle as needed.



47 To make the **LevelSelect** scene work, we're going to need a script. In the **Scripts** folder, create a **new C# script** and name it "**LevelSelect**". Attach this script to our **canvas** object by dragging it onto the canvas in the **Hierarchy**.



48 This script is going to be used to load scenes, so we need to add `using UnityEngine.SceneManagement;` to the directives at the top:

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

49 Just like the buttons in the Game UI, we just need a function to call when `On Button Click ()` is activated. Add a function called `LevelSelectButtonPressed()` and give it a **string** parameter "`levelName`":

```
public void LevelSelectButtonPressed(string levelName) { }
```

```
public void LevelSelectButtonPressed(string levelName)
{

}
```

50 Inside this function is where we will use the **SceneManager** to load a scene:

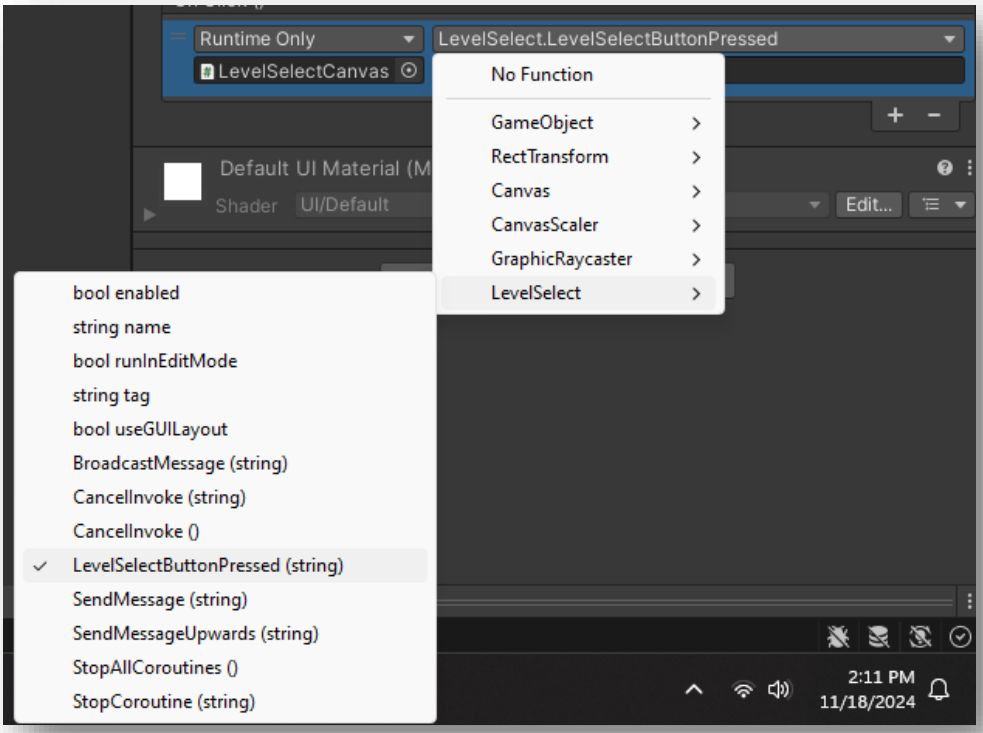
```
SceneManager.LoadScene(levelName);
```

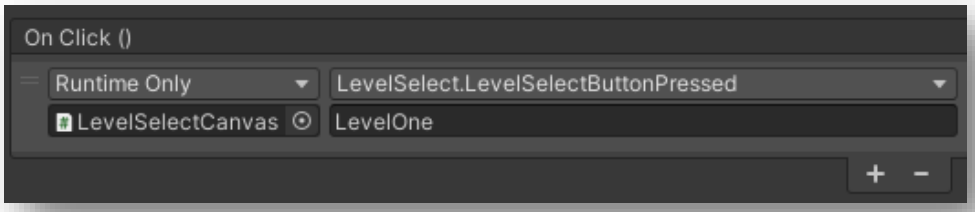
```
public void LevelSelectButtonPressed(string levelName)
{
    SceneManager.LoadScene(levelName);
}
```

51 **Save** the script and go back to Unity.

52 Select the **Level01Button** in the **Hierarchy**.

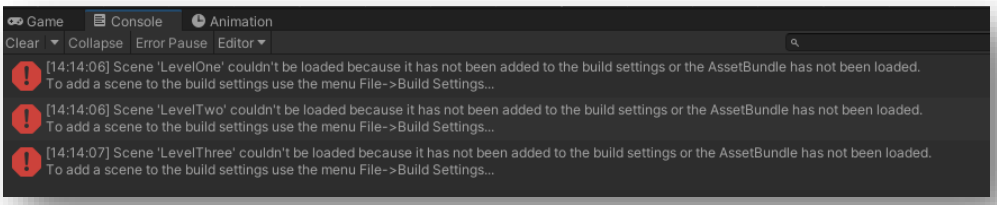
53 Add the **LevelSelectCanvas** Object and have it call the `LevelSelectButtonPressed()` function. Since this function requires a string parameter, we will need to specify the name of the first scene exactly as it appears in the Scenes folder: **LevelOne**.





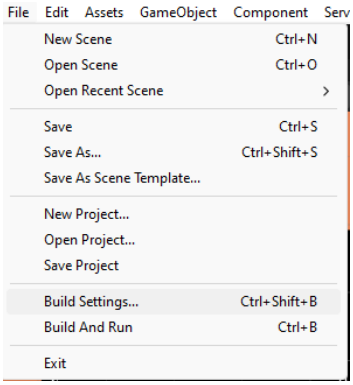
54 Repeat these steps for the second and third buttons.

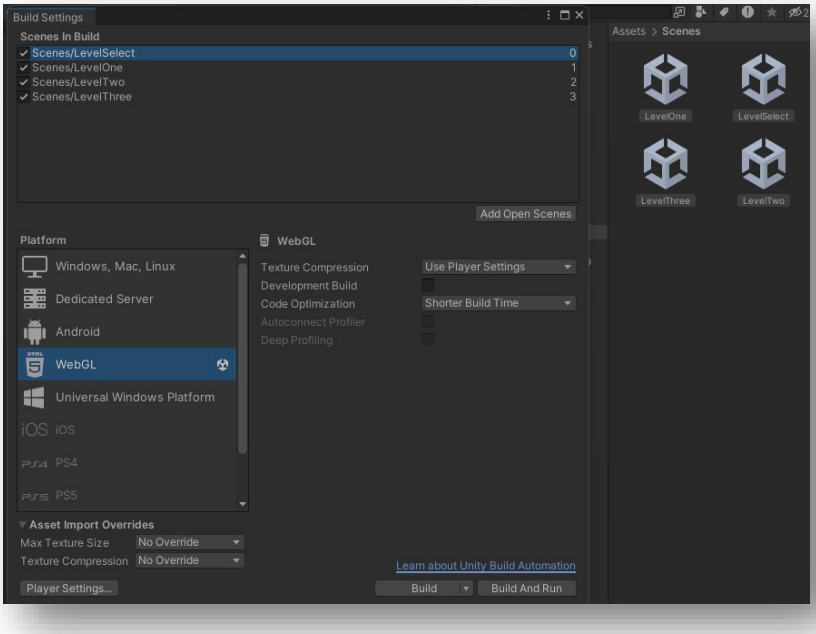
55 Try **playing** the game. If you click on a **button**, it will give you an **error**.



What's up with that?

Unity needs to have the **levels defined** in the **Build Settings**. Click on **File > Build Settings**. Drag the **three level scenes** and the **LevelSelect** scene from the **Scenes** folder into the "**Scenes In Build**" window. **Rearrange** the scenes so that **LevelSelect** is first and then you can close the window.





56 Now we can get to the levels. How do we get back?

Open the **GameOver** script. Inside the `OnDoneClicked()` function, add:

```
SceneManager.LoadScene("LevelSelect");
```

```
83      public void OnDoneClicked()
84      {
85          SceneManager.LoadScene("LevelSelect");
86      }
```

57 **Save** the script. Now, when you play the **LevelSelect** scene, you can load any level and get back to the **LevelSelect** scene when you're finished.

58 We can save the player's progress and show it to the player when they play again.

To start, open the **HUD** script. We will need to use the `UnityEngine.SceneManagement` directive to keep track of the player's progress from each scene. Add this to the top of the script:

```
using UnityEngine.SceneManagement;
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using UnityEngine.SceneManagement;
```

59 There is no need to check the saved progress if the player loses. Only if they win. Add this `if` statement to the `OnGameWin()` function:

```
if (starIndex >
PlayerPrefs.GetInt(SceneManager.GetActiveScene().name, 0))
{
     PlayerPrefs.SetInt(SceneManager.GetActiveScene().name,
starIndex);
}
```

```
public void OnGameWin(int score)
{
    GetComponent<Canvas>().sortingOrder = 3;
    GameOver.instance.ShowWin(score, starIndex);

    if (starIndex > PlayerPrefs.GetInt(SceneManager.GetActiveScene().name, 0))
    {
        PlayerPrefs.SetInt(SceneManager.GetActiveScene().name, starIndex);
    }
}
```

If the **starIndex** value is **greater than** what was **previously saved** for the **current scene**, we will **save the current value of starIndex** to the **PlayerPrefs** associated with the **name of the current scene**.

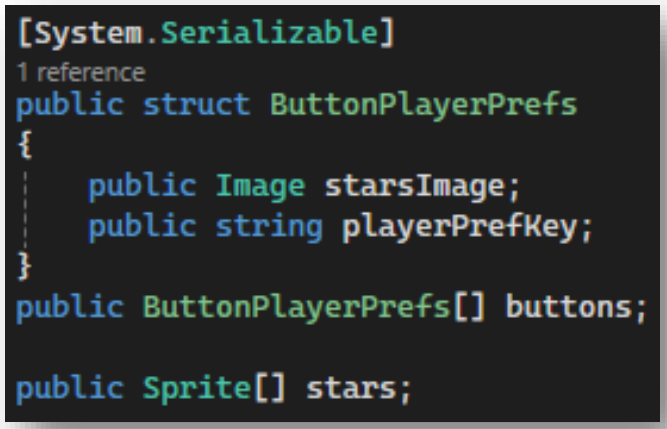
It helps to think of **PlayerPrefs** as just another variable, but one that gets remembered between scenes. The name of the current scene becomes the **key**, or variable name, for this.

60 Now to use this information in our **LevelSelect** scene. Open the **LevelSelect** script. We will be changing the stars image using code, so import the UI library at the top of the script.

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

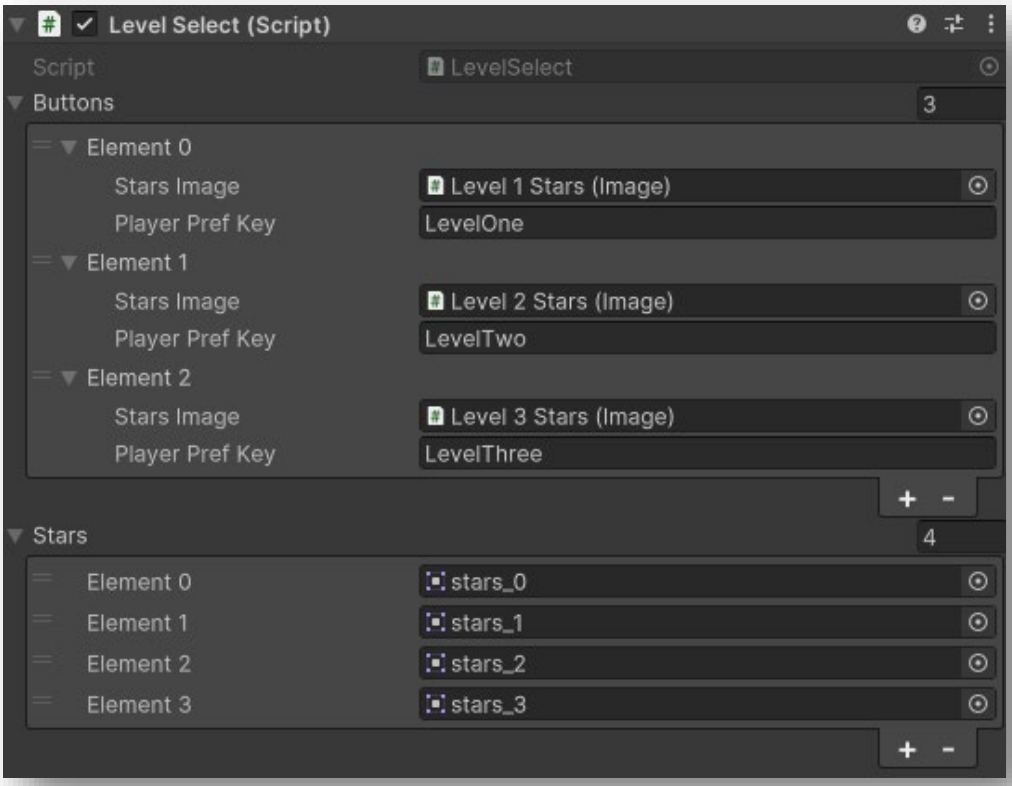
61 Our script needs to know the **PlayerPrefs** associated with each scene. To do this, we'll use a **struct**. **Structs** are like mini classes, and are great for storing information. In the **class**, set up the **struct** and **variables** for the button array and star array like this:

```
[System.Serializable]
public struct ButtonPlayerPrefs
{
     public Image starsImage;
     public string playerPrefKey;
}
public ButtonPlayerPrefs[] buttons;
public Sprite[] stars;
```



62 **Save** the script and in Unity, select the **LevelSelectCanvas**.

Find the **LevelSelect** script component in the **Inspector** and add each of the buttons and the names of the associated scenes to the array:



63

Save this and go back to the **LevelSelect** script. In the `Start()` function, we will add a foreach loop.

Start by looping through each button in the array.

```
foreach (ButtonPlayerPrefs button in buttons) { }
```

Inside that loop, set a new variable `score` to the value found in the **playerPrefKey** at that button.

```
int score = PlayerPrefs.GetInt(button.playerPrefKey, 0);
```

Below that, set the button image's sprite to the corresponding stars image.

```
button.starsImage.sprite = stars[score];
```

```
void Start()
{
    foreach (ButtonPlayerPrefs button in buttons)
    {
        int score = PlayerPrefs.GetInt(button.playerPrefKey, 0);
        button.starsImage.sprite = stars[score];
    }
}
```

64

Save your work and play the game. Try to get at least one star in one of the levels and click on **Done**. You will see that the level button now shows that number of stars that you earned. Feel free to create additional levels with more challenging stars to earn!