CODE NINJAS®

Silver Belt Ninja Guide
Activity 14: Amazing Ninja Worlds Part 3

Activity 14

Amazing Ninja Worlds – Part 3

In this final part of building your Super Ninja World platform game, you'll add a start screen and a menu. The start screen and menu will make it feel like a completed game!

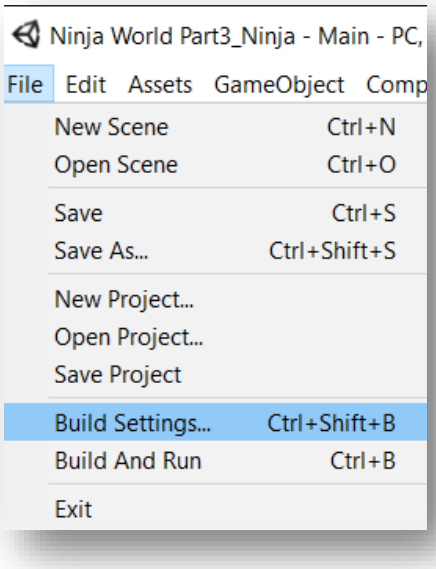
1 You should continue with the project that you had completed in the previous activity.

2 Now that you're comfortable moving the player around the environment and between scenes, we can make this into a complete game by adding a start screen and a menu.

3 Remember using the Build Settings in World of Color? We are now going to use it to manage even more scenes.

Click on the **File** tab and click on **Build Settings**.

Reminder: The main role of the Build Settings is to export your project so that others can play it.

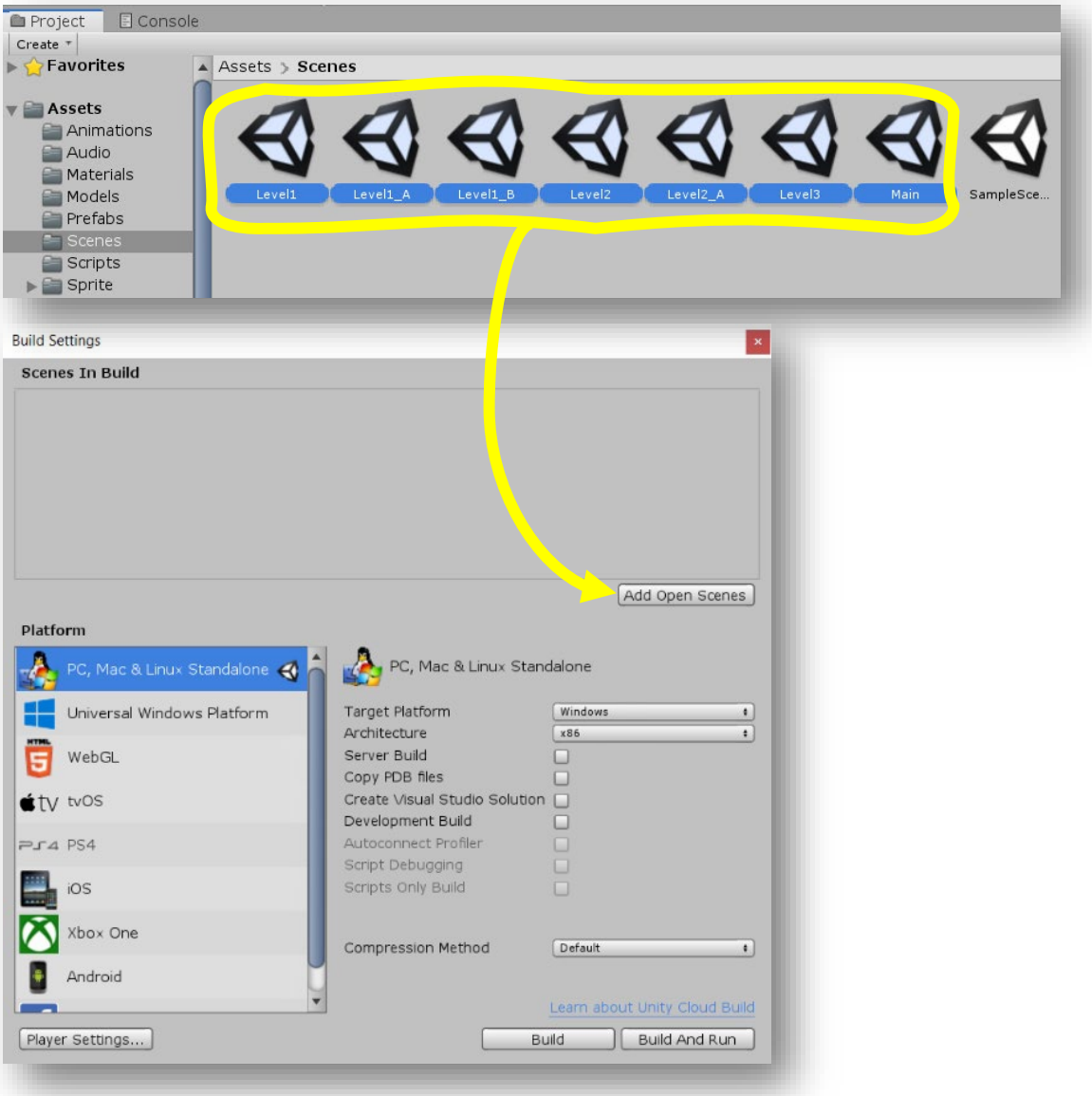


When you have multiple scenes like in this game, all scenes need to be in the Scenes in Build section of the Build Settings. If not, Unity won't be able to load any of the scenes by name.

Pro Tip: Use the Scene's Name

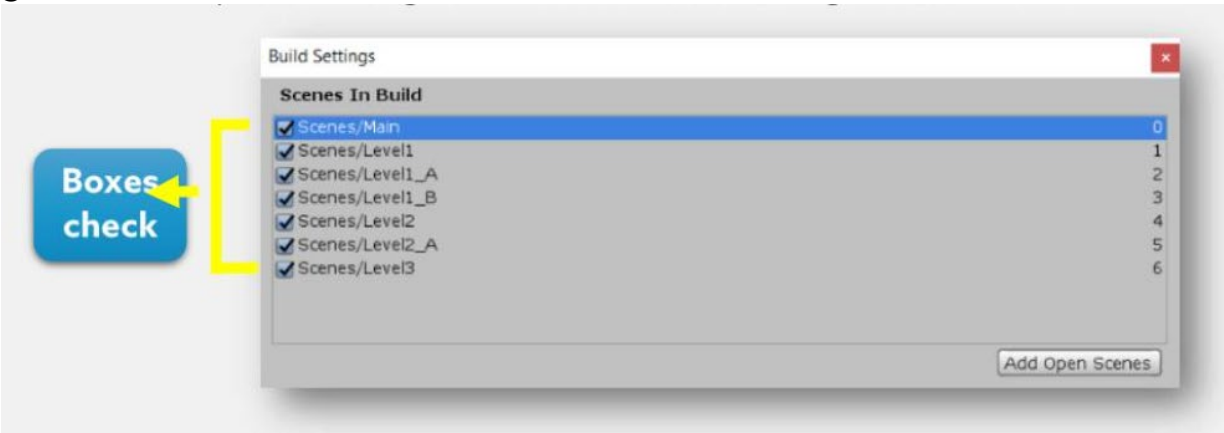
This game has more than five scenes and it's possible that there will be more. If scenes get added (or removed), their order might change. What was once scene number 1 might get moved to 3 or anywhere else. By referring to the scenes by name, you'll always get the scene you want.

4 With **Build Settings** still open, select all the scenes in the **Project** window (except for SampleScene) and drag them into the **Scenes in**

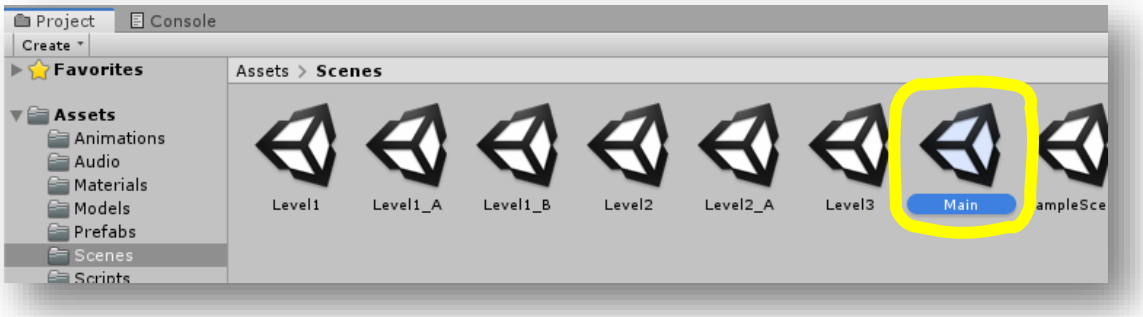


Build box in Build Settings.

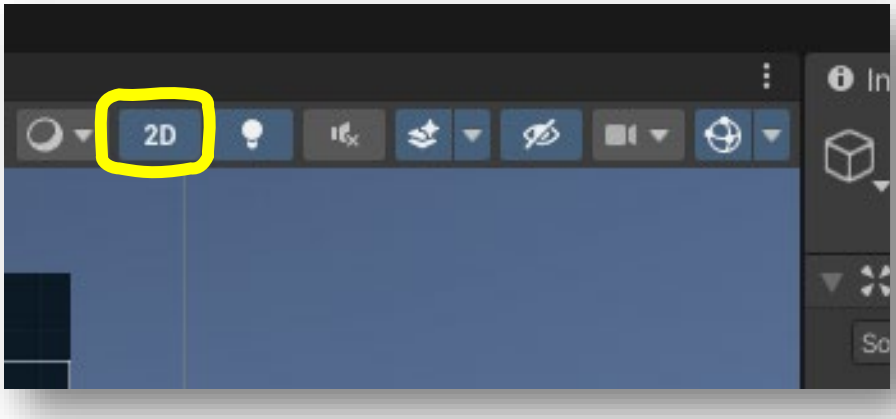
5 Make sure that all their boxes are checked. Highlight the **Main** scene and drag it to the top so that it is scene 0. Whatever scene is at the top of the list gets loaded first when the game is started after the game is built.



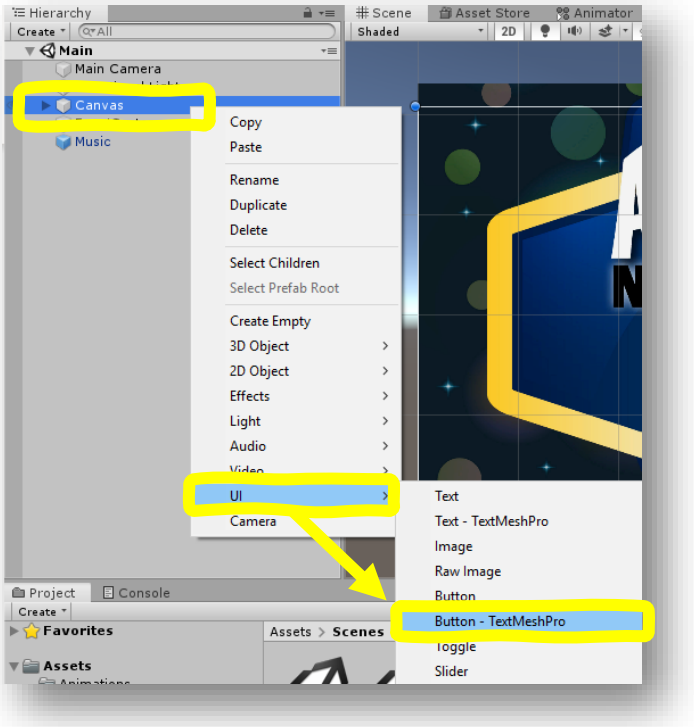
6 Now we can set up the main menu where the player can pick which scene they want to play. Close the Build Settings menu and load the **Main** scene by selecting it from the **Scenes** folder in the **Project** window.



Since the menu is 2D, make sure the **Scene** window is in **2D** mode.



7 The menu canvas object has been set up with an image, but there are no interactive buttons. We need to add buttons that send the player to the right scenes.



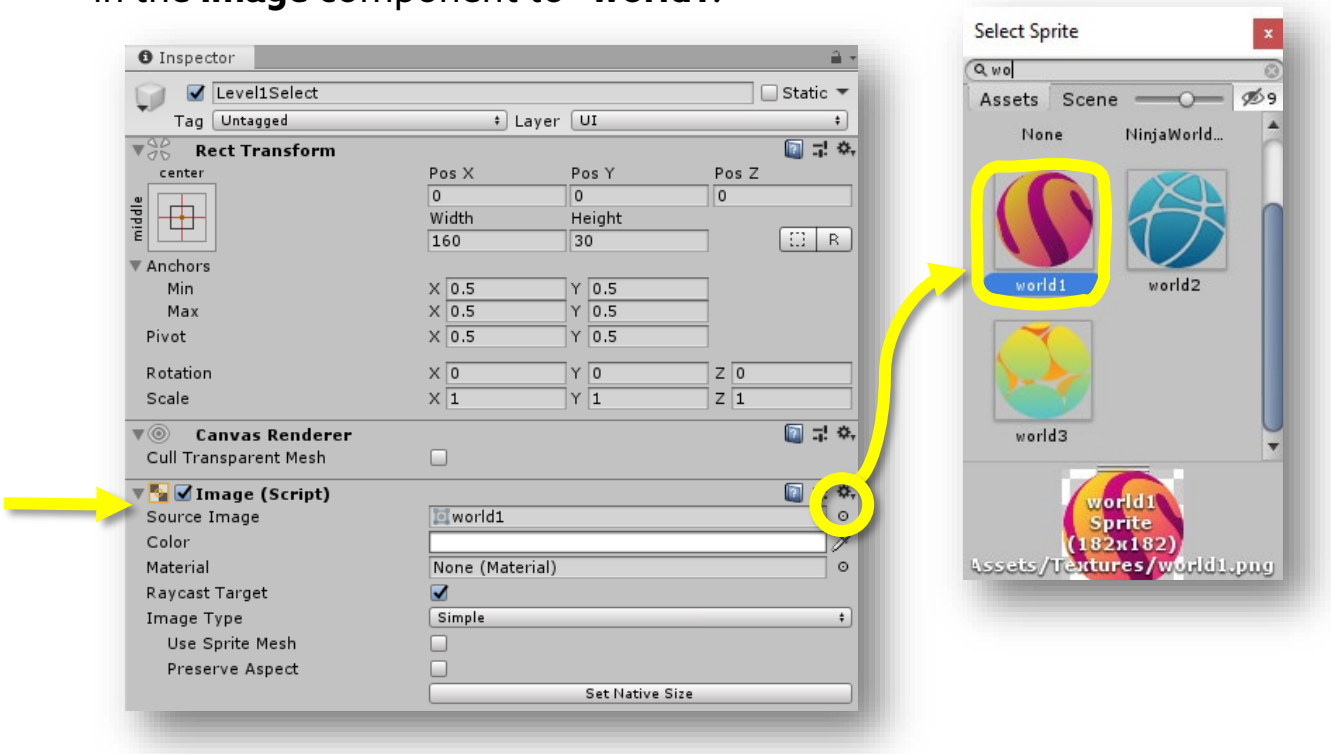
With the **Canvas** selected, add a new GameObject from **UI/Button – Text Mesh Pro**.

We're using Text Mesh Pro instead of an ordinary button since we want to include some special text with the button.

Change the name of the object to "**Level1Select**".

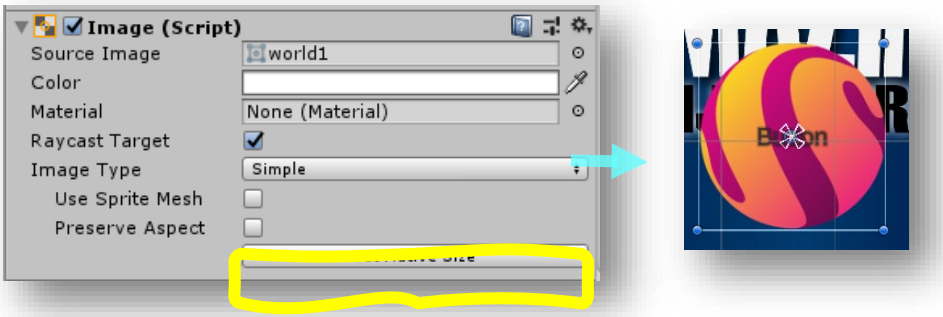
8

In the **Inspector** for the **Level1Select** button, change the **Source Image** in the **Image** component to "**world1**."



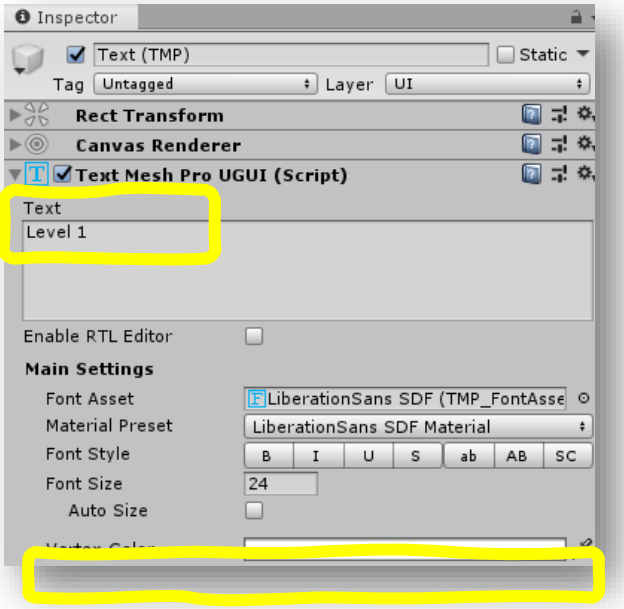
9

You may notice that the button looks stretched. It looks like an oval rather than a circle. This means that it has the wrong aspect ratio. To fix this, in the **Inspector**, click **Set Native Size**.



10 Resize and move the button object so that it fits well under the game title. There will be three buttons total, so move this button to the left side of the menu as shown.

11 The button comes with a text object attached. Select it. In the **Inspector**, change the **Text** to "**Level 1**" and change the **Vertex Color** to white.



12

Move the text object so that it is in the lower area of the button as shown:

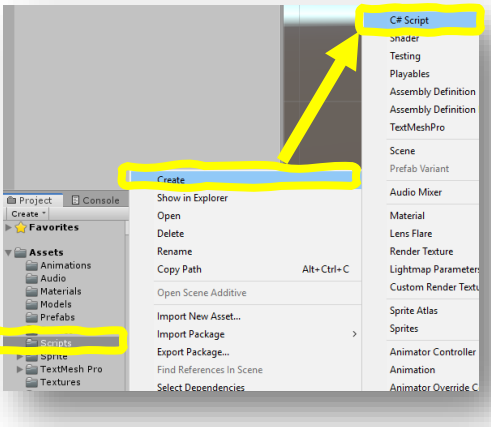
Master Your Skill

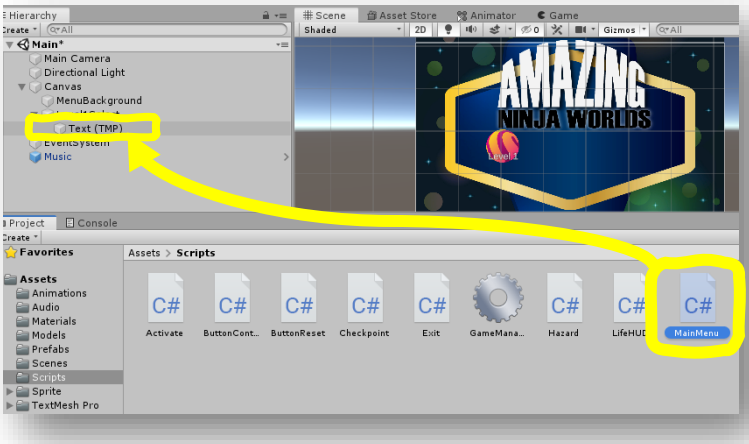
Text Mesh Pro (or "TMP") has lots of options to modify the look of the text. You can add things like a drop shadow and so on. Feel free to take a moment to experiment with the settings!

13

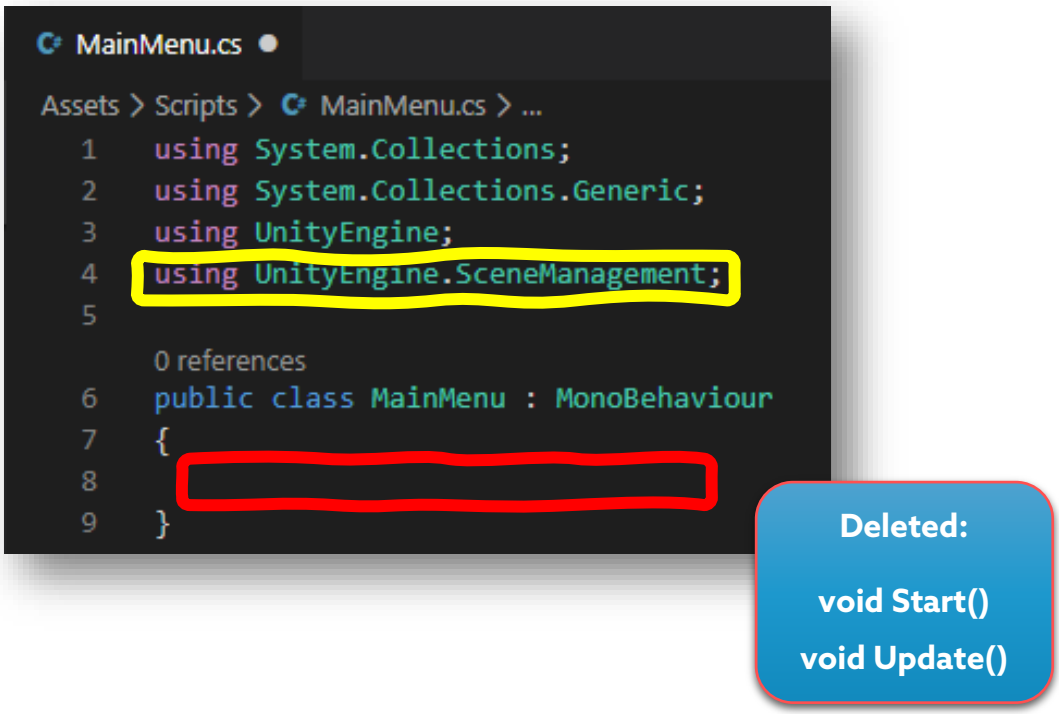
We need a script to control what happens when the user clicks on the button. Create a **C# script** in the **Scripts** folder and name it "**MainMenu**". Drag the script component onto the **Level1Select** game object.

IMPORTANT – As you might have guessed, we will be duplicating the Level1Select button for Level2 and Level3, but do not do so until instructed to do it. Complete Level1Select button first in order to save yourself some time and frustration.

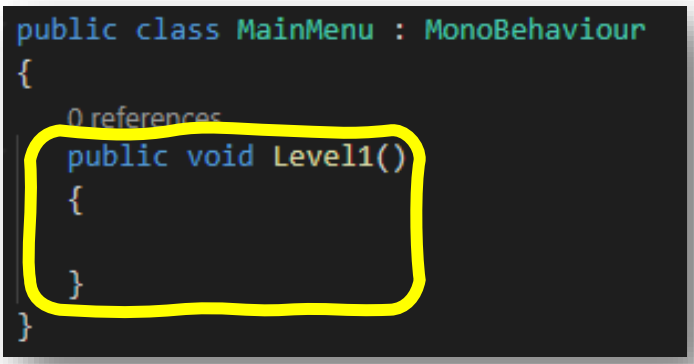




14 Open the **MainMenu** script. We won't be using the `Start` or `Update` functions, so delete those. This script will not need any variables, but you will need to add the scene management directive at the top: `UnityEngine.SceneManagement`



15 Create a new function as follows:



16

Inside the function, we will use the **SceneManager** to change scenes:

```
public void Level1()
{
    SceneManager.LoadScene("Level1");
}
```

Careful!

Check your spelling. Make sure it matches the scene name exactly. As always,

LoadScene does exactly what it says – it replaces the current scene with the scene you've chosen. When **LoadScene** is used, the new scene behaves exactly like it would when you start it the first time.

17

When we load a new game- a new level 1- we need to think about what happens to the other variables, such as the lives left.

How Does Ninja World Handle Lives Left?

The game uses a built-in Unity class called PlayerPrefs to track the number of lives left. The lives are saved as "LIVES_LEFT" in PlayerPrefs. We use this so the number of lives left can be carried over between levels. For example, if you have two lives left at the end of level 1, you will start level 2 with two lives.

When the player starts a new game, we want to make sure that the player has the maximum lives, not the lives left. To do this, use PlayerPrefs to delete the "LIVES_LEFT" key so that the game starts fresh with the maximum 3 lives. Add the following line:

```
public void Level1()
{
    SceneManager.LoadScene("Level1");
    PlayerPrefs.DeleteKey("LIVES_LEFT");
}
```

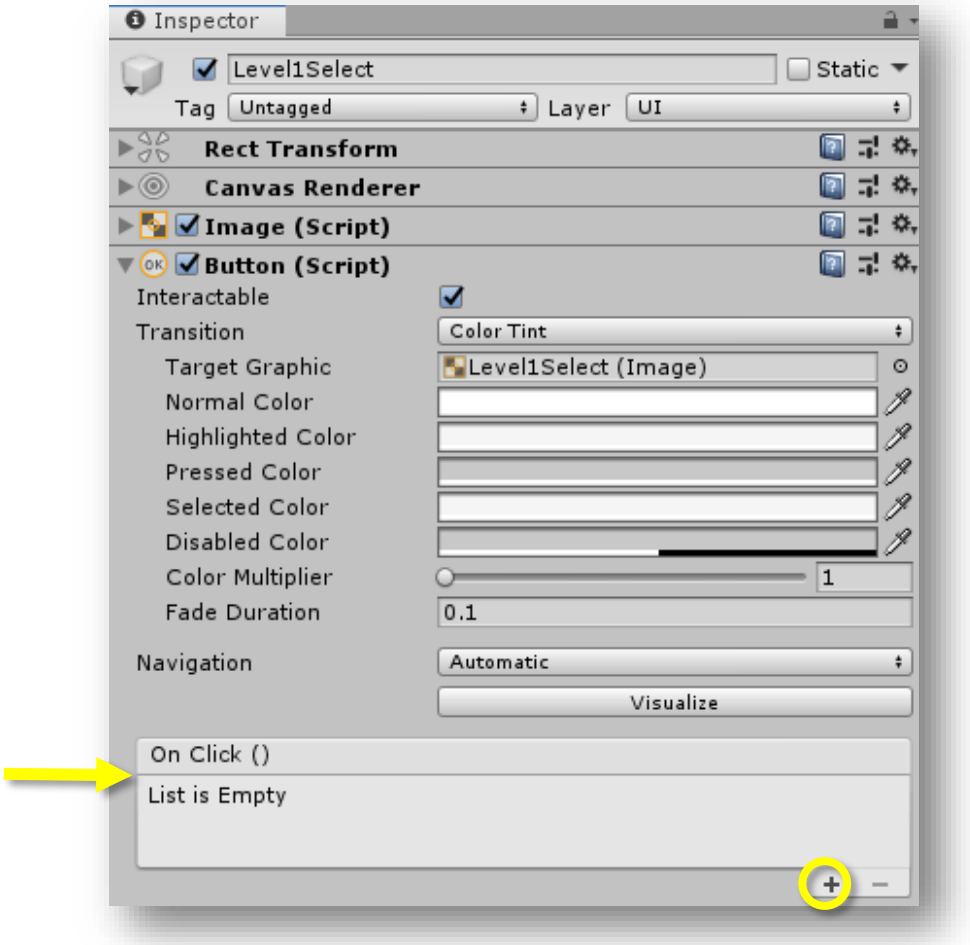
`DeleteKey()` removes the variable completely so that there is no stored value for "LIVES_LEFT" when beginning the game.

18 **Save** your script.

19 Back in Unity, make sure the **Level1Select** object is selected. We will now set up the button component, so the button actually works.

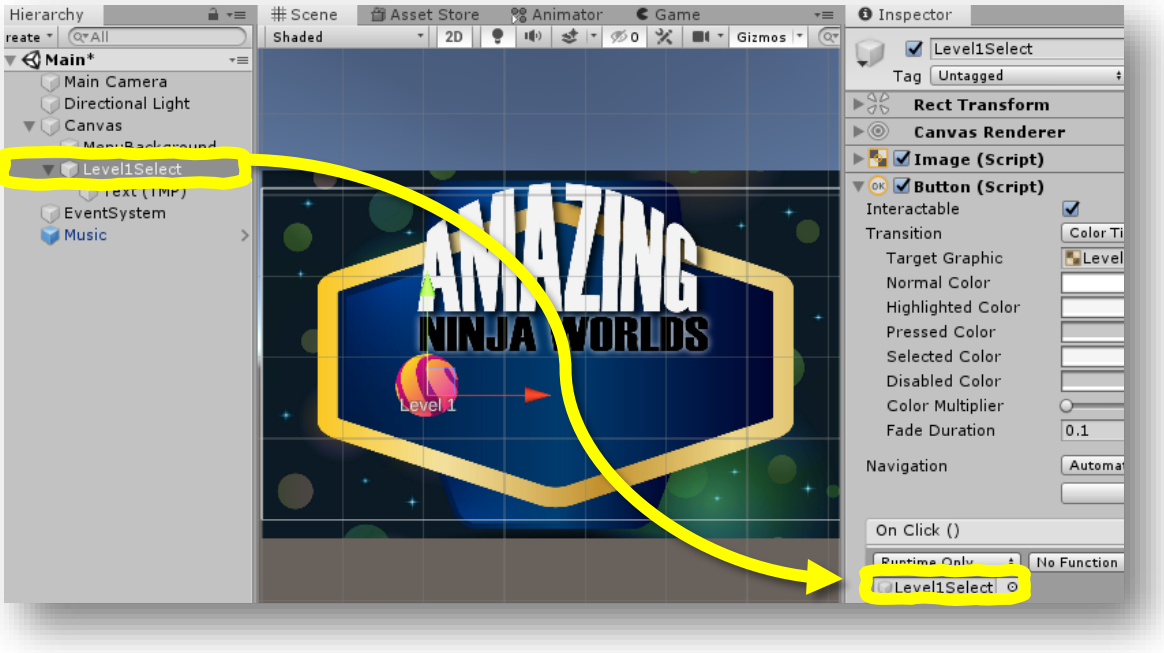
Scroll down in the **Inspector** to the **Button** component. At the bottom of the component is an empty list for events to run when "**On Click ()**" happens. Click on the **+** symbol to add an event to the list.

The first selector is "**Runtime Only**". Since we only want this to work when the game is actually being run, leave this as it is.



20

Beneath the **Runtime Only** drop-down, is a slot for the object. Drag the **Level1Select** object from the **Hierarchy** into this **box**.

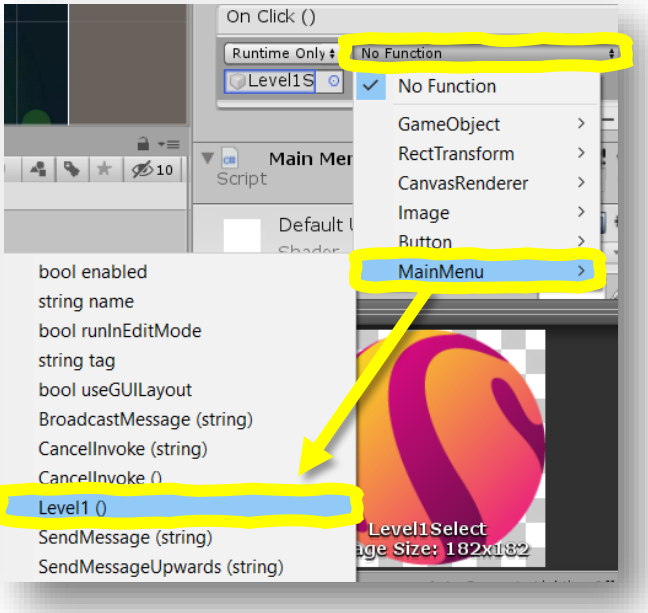


Why does Unity insist on referring to an object that the component is already attached to? Because sometimes you may want to have a button affect an unrelated object – so that functionality is there.

21

Finally, click on the selector that says, "**No Function**." In the drop down, select **MainMenu** (the script you just wrote) and select **Level1()** from the options.

This tells Unity: when click on the Level1Select button, run the Level1() function in the MainMenu script.



22 Play the scene. Test the button to see if Level1 is loaded on click.

Is something missing? How do we get back to the main menu? We'll address that in a minute, but first, let's set up the buttons for the remaining levels.

Stop your game.

23 Instead of creating 3 methods, one for each level, since they will each be doing the same thing we can use a method with a parameter!

24 Open the **MainMenu** script. Change the method from Level1 to LevelChange.

```
public void LevelChange()
{
    SceneManager.LoadScene("Level1");
    PlayerPrefs.DeleteKey("LIVES_LEFT");
}
```

25 Inside the parentheses, define a string parameter called "level".

```
public void LevelChange(string level)
{
    SceneManager.LoadScene("Level1");
    PlayerPrefs.DeleteKey("LIVES_LEFT");
}
```

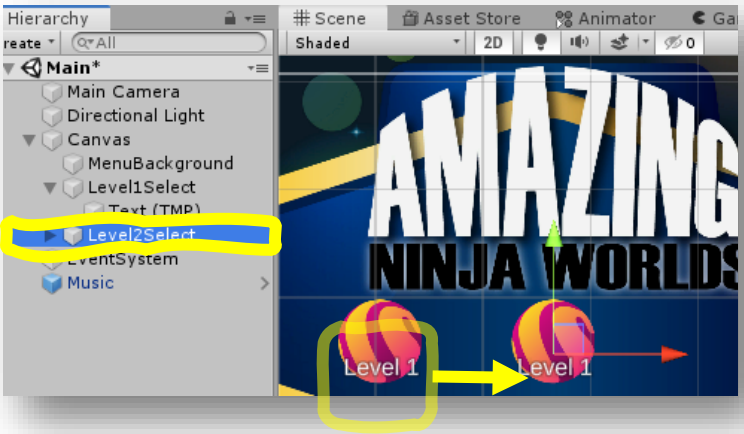
26 Replace the "Level1" string inside the LoadScene method with our new parameter.

```
public void LevelChange(string level)
{
    SceneManager.LoadScene(level);
    PlayerPrefs.DeleteKey("LIVES_LEFT");
}
```

27 Back in Unity, select **Level1Select** and change the method to our new **LevelChange** method. In the box below, type in the name of the scene we are going to switch to ("Level1").

28 Now for the other buttons. Select **Level1Select** and duplicate it using **ctrl+D**. Rename this button "**Level2Select**."

Move it to the right so it is in the middle under the title.



29 We will go through the same steps to update the image for level 2's button's as we did for level 1. In the **Inspector**, in the **Image** component, change the following items:

1) Change the **Source Image** to "**world2**".
2) If the image looks a little strange, click on "**Select Native Size**."
3) Change **Text(TMIP)**'s **Text** component to "**Level2**."

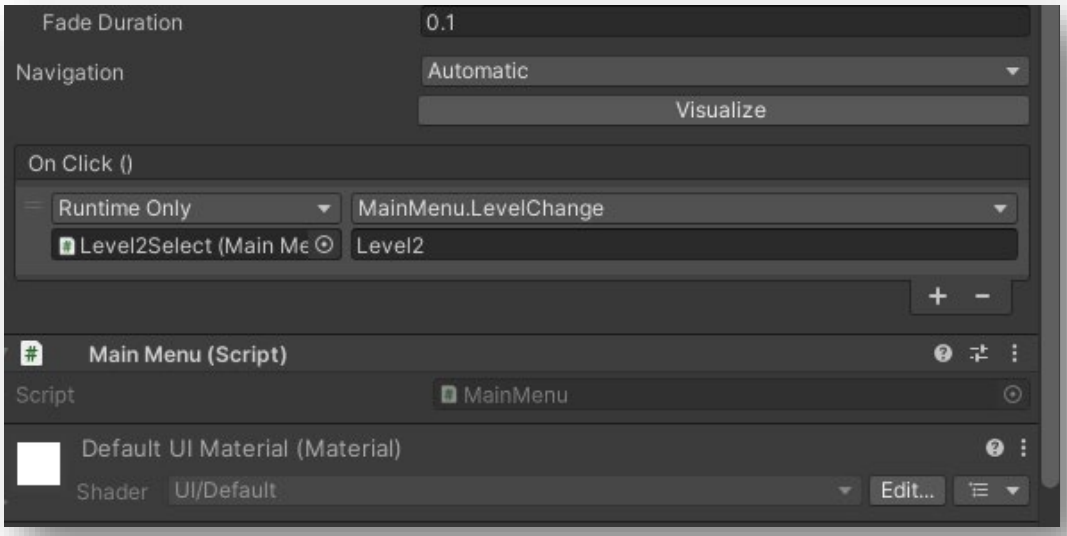
Your Level2Select button should look like this:

30

For the **Button** component, make the following changes:

1) Change the object in the **On Click ()** list to **Level2Select**.
2) Change **No Function** to **MainMenu/LevelChange()**.
3) Type in "Level2" as the parameter.

Your Level2Select OnClick () should look as follows:



31

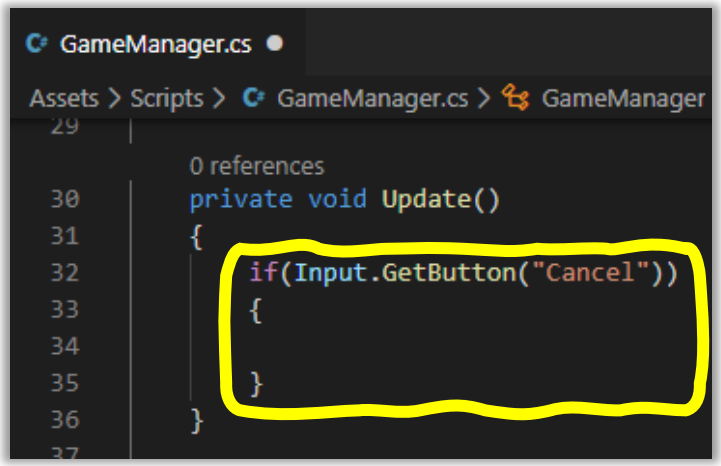
Repeat the steps to make a **Level3Select** button, changing the number 2 to 3 in all the same places.

Your Main scene should look as follows:

32 Play the game and see what pressing the buttons does.

33 Now all we need is some way to get back to the main menu from anywhere in the game. There is one script that is in every level: **GameManager**. Find it in your **Project** window to open it.

34 In the `private void Update()` function, add a conditional to check if the cancel button is being pressed:



In Edit > Project Settings, the cancel button is linked to the esc key.

35 Inside the conditional, we'll use the SceneManager to load the main menu:

```
private void Update()
{
    if(Input.GetButton("Cancel"))
    {
        SceneManager.LoadScene(0)
    }
}
```

You can also use the scene's name, "Main," but loading 0 will always take you to the opening scene because it's the first scene in the build.

Save and close your script.

36 **Play** the game. Make sure you can load the right level with the right button and that you can press esc to go back to the main menu at any time.