User Interface (UI)

Sebastian Grygorczuk - STEM Institute at CCNY

Today's Agenda

We're going to be going to be going over Chapter 14 UI Elements

- Learn how to create UI that informs player of their status
- Use Buttons to take player input
- Save Data and Move it Between Scenes

Examples of UI

UI consists of everything that's not directly in the game world.

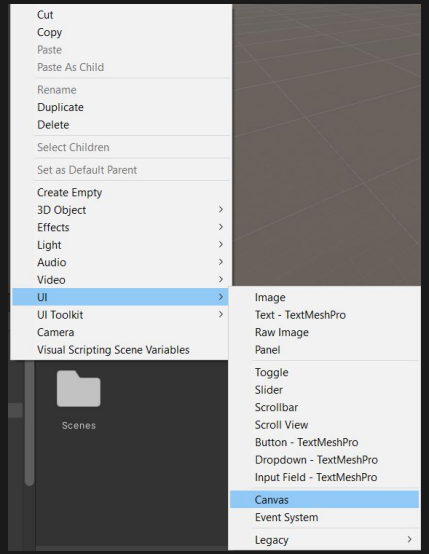
Character Inventory, Overlays that tells you your stats, menus that pause game, title screens, character dialogue section. All of those are UI elements that allow you to convey the general mechanics and ideas to the player.

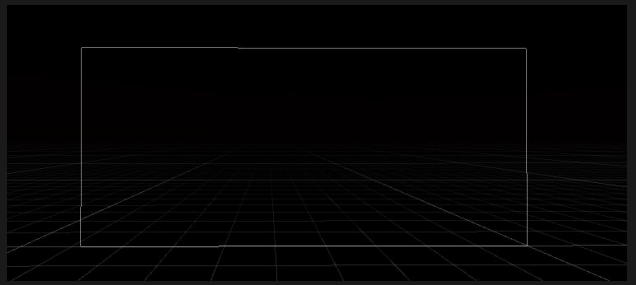
Canvas

To start creating you UI you have to create a Canvas. All UI Game Objects have to be stored under the Canvas Game Object otherwise they will not function.

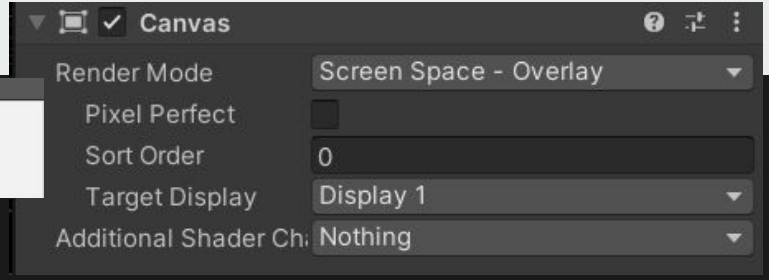
Canvas is the 2D space where the Objects are displayed and is directly linked to how the Camera views the UI elements.

When you create a Canvas you will notice that is is massive, it's because it emulating the size of the screen you are using.





Canvas Component

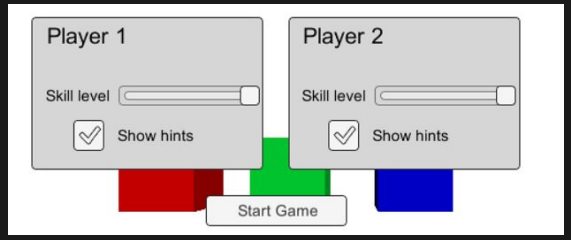


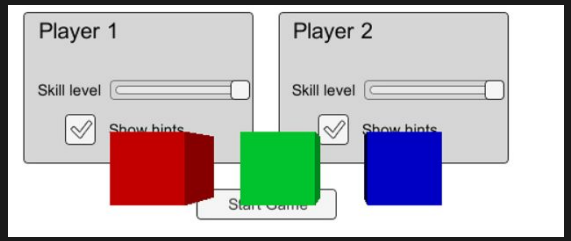
The Canvas component controls where in the Game World the UI should be placed. You have three Options. Screen Space -Overlay, Screen Space - Camera, and World Space.

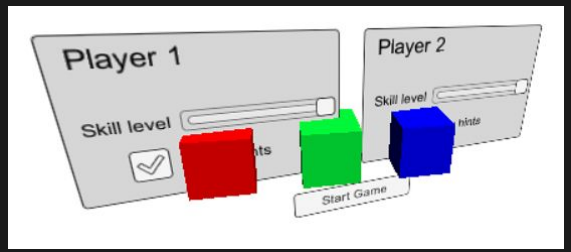
The Screen Space tells you that the UI element will always be attached to the Screen at the same position and size all the time.

Overlay will always keep the UI elements on top of the game while Camera will allow objects to clip into the UI if the Camera moves to them.

World Space on the other hand tells the UI that it exists away from the Camera and will be different size depending on the distance of the player to it.





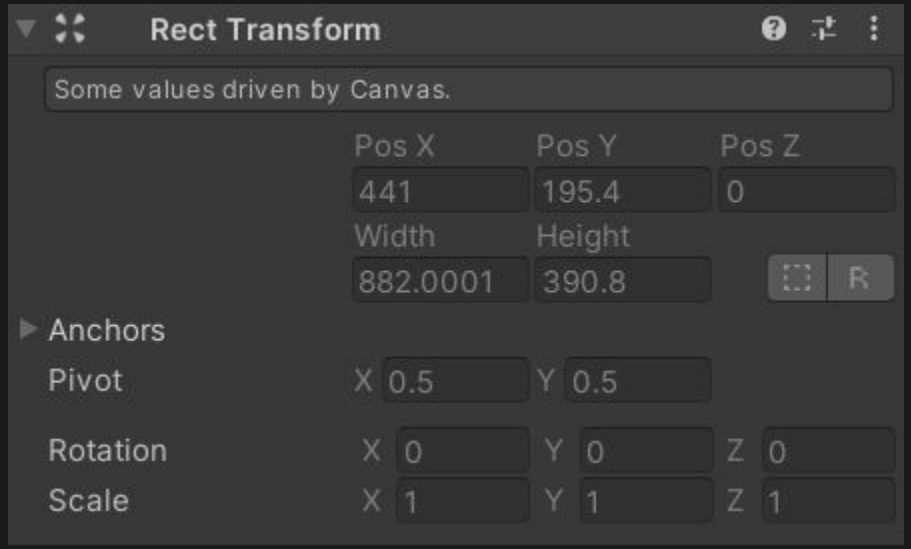


Sebastian Grygorczuk - STEM Institute at CCNY

Rect Transform

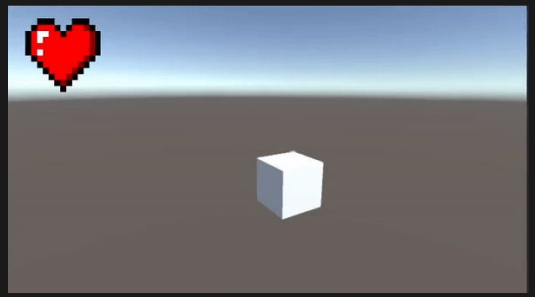
Rect Transform works similar to regular transform with the exception that when you have Overlay or Camera setting on you don't get to control it, it's decided by the camera position and the Resolution set in the Game View.

World Space however you get full control over as it behaves as a regular game object.



Canvas Scalar

Canvas Scalar determine how to UI elements are scaled you can do it in 3 ways, using pixels, by scale to a specific resolution or by using physical dimensions. Either one will let you scale the size of Images, Text and buttons that the Canvas holds.



Canvas Scaler
UI Scale Mode: Constant Pixel Size
Scale Factor: 1
Reference Pixels Per: 100

Canvas Scaler
UI Scale Mode: Scale With Screen Size
Reference Resolution X: 800 Y: 600
Screen Match Mode: Match Width Or Height
Match: 0
Width Height
Reference Pixels Per: 100

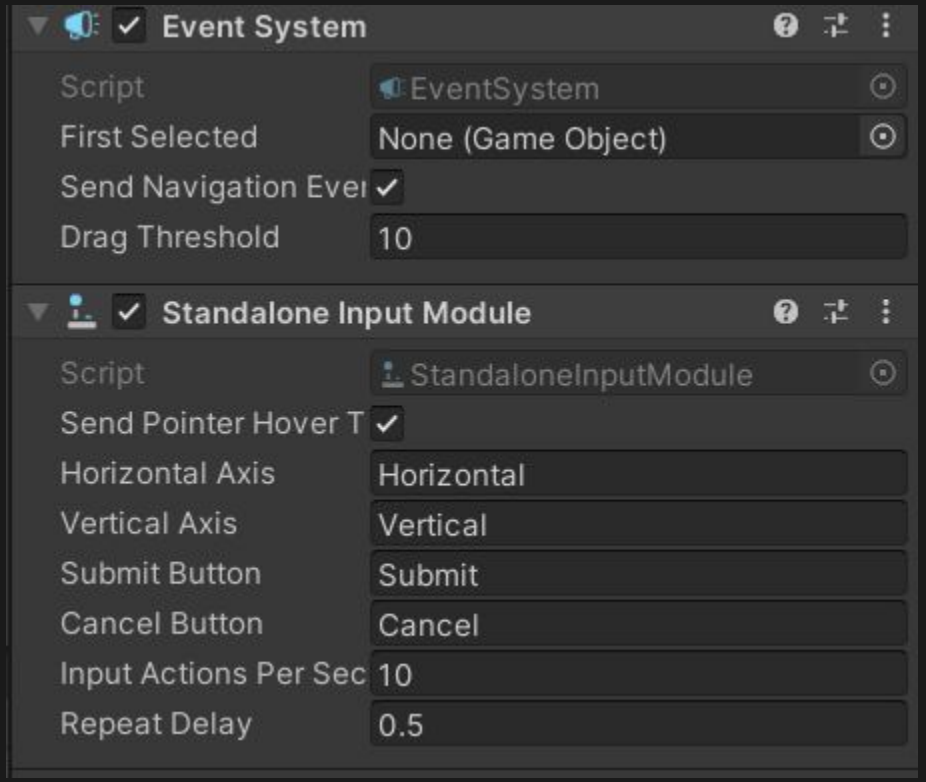
Canvas Scaler
UI Scale Mode: Constant Physical Size
Physical Unit: Points
Fallback Screen DPI: 96
Default Sprite DPI: 96
Reference Pixels Per: 100

Sebastian Grygorczuk - STEM Institute at CCNY

Event System

When you create a Canvas automatically the Event System Game Object will also be created.

The Event System Game Object allows the Canvas to take in user Input and perform actions based on them.



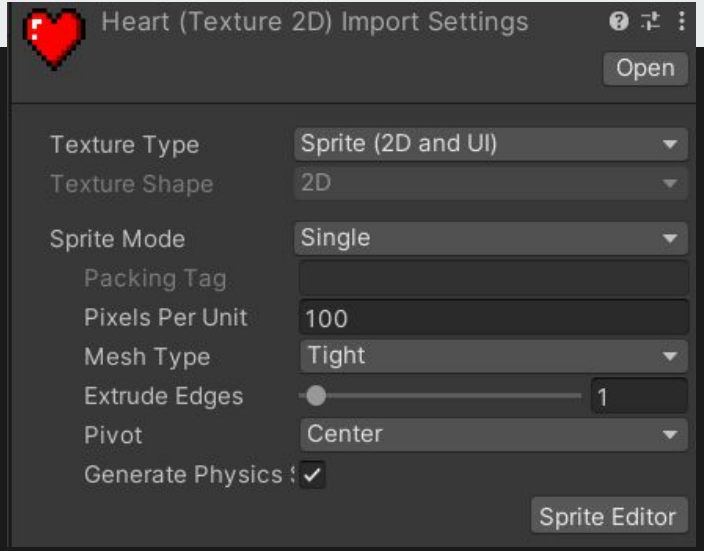
Images

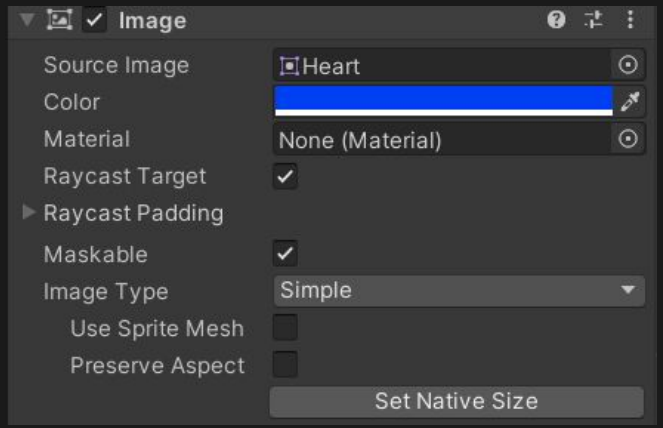
When placing a image down into the UI you first have to make sure that its Import Settings are set to Sprite (2D and UI) otherwise it will be imported as a Texture and won't show up on the Source Image List.

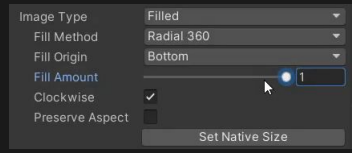
The Image shares a lot of common aspect as the Sprite Renderer.

And has one more Image Type, in addition to sliced, simple, tiled now we also have field which can control how much of the image is visible.

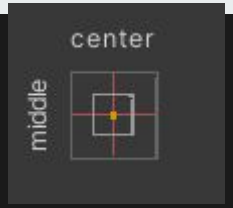
This is very useful for showing player health, stamina or other stats.

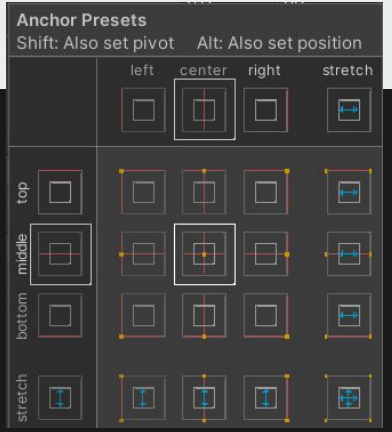






Anchors

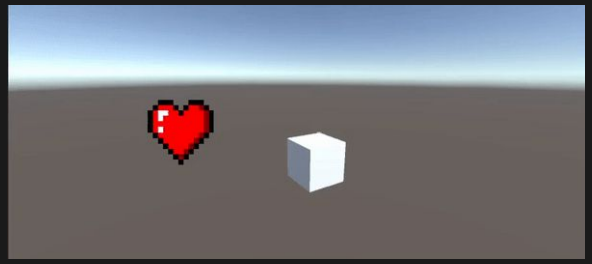


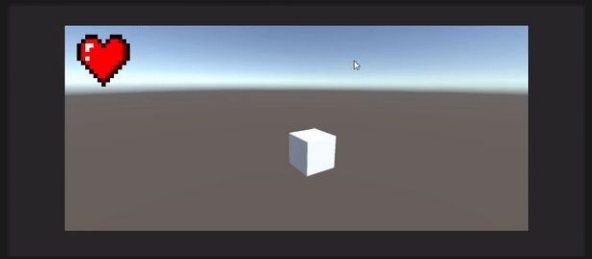


Inside the Rect Transform for the Image you may have found this little Menu Item.

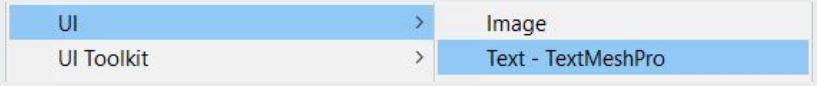
This is the Anchor it determines how the UI element should behave when. All UI elements come in with the Center behavior enabled meaning that the item will more or less stay in the same position of the screen regardless of the size of the screen.

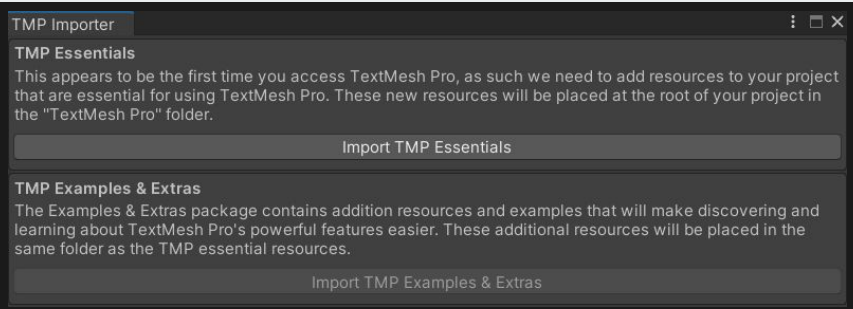
As you can see that's not very useful, it's best to attach an anchor to a corner and then the item will always move in reference to that corner.

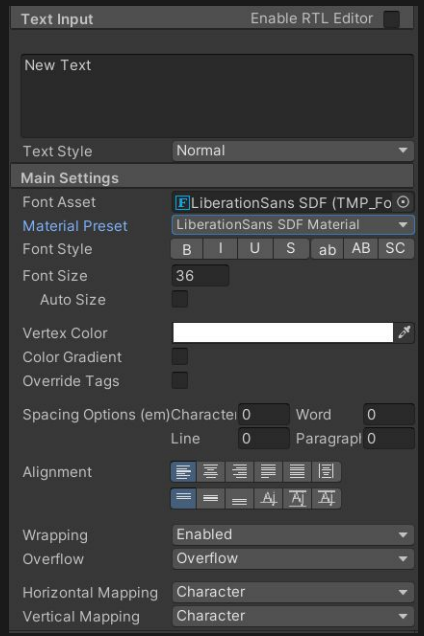




Text







Text is an important part of the game to let the player know what to do.

When you first create a Text you will get a Popup to install TextMesh Pro Essential, click yes and that will bring in basic fonts to be drawn and will create a folder with the font game assets.

Once all of that is in you will be met with the Text Input and you will have the big text box where you can type in what will show up with many settings that you should recognize from word processing softwares.

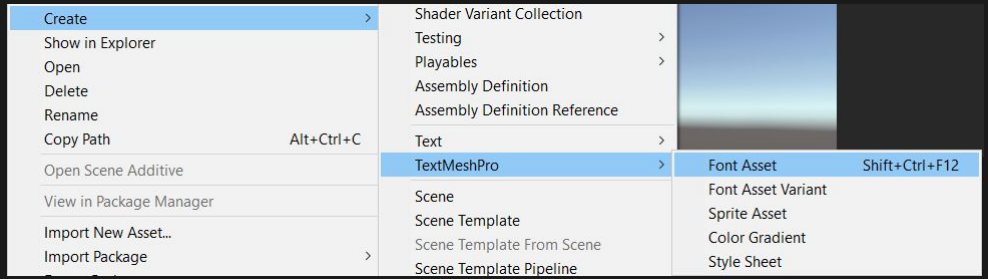
Sebastian Grygorczuk - STEM Institute at CCNY

Fonts

https://www.1001freefonts.com/

When you install the Mesh Pro Text you will only be met with one Font type, if you want your text to look different you will have to download it from a website like 1001 free fonts.

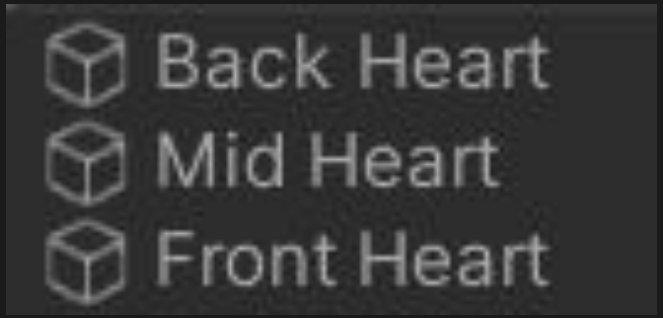
Once you have it download and imported the font you want you then have to process it into a Pro Mesh Text before you can apply it to a Text Game Object.



Layering Object on Canvas

Unlike 3D where the Z axis or 2D where Layer Order controls what is draw closer to the camera the order of the UI elements inside the Hierarchy determines the draw order.

Game Object that are higher in the child list will be draw in the back while the Game Object that are lower on the child list will be drawn in the font.

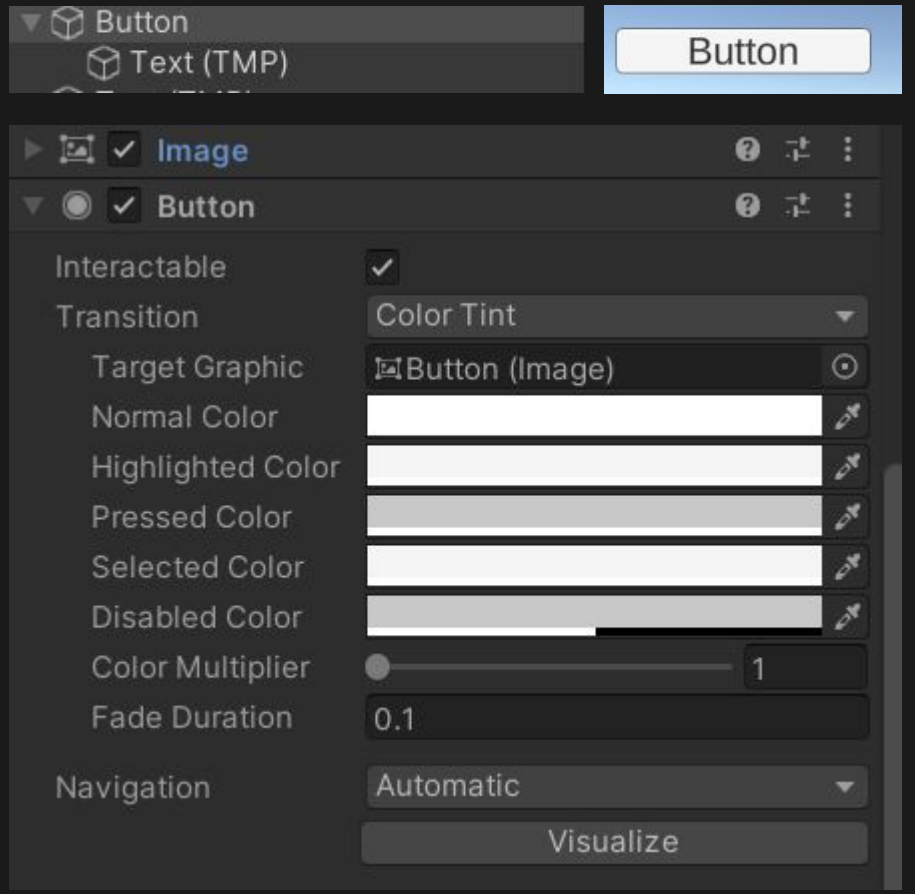


Buttons

Creating a button will also create a Text Game Object that will display the text on the button.

The button also comes with an Image Component where you can replace the image of the button.

Finally the button component gives you the ability to click it how the image should behave after interacting with it.



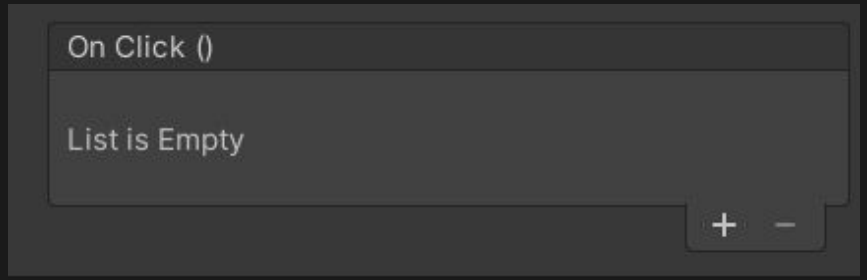
Connect Button Action To Text

To be able to connect a script to a button you will use this OnClick section bellow the color selection.

First you will need your script, we're going to make this button show an number of times you've click on something.

So here we have a script will require that you add a new using TMPro, so that we can connect to the Text which is TextMesh Pro GUI class.

Then we just update the button counter each time it's clicked and set the text equal to that new updated value.




```
//You have to be using TMPro to connect
//To the GUI elements
using TMPro;
```

```
//Counts how many times the button has been pressed
private int _buttonCounter = 0;
//Connect to the Text On Screen that will update
public TextMeshProUGUI textMesh;

//Increments the button Counter and update the text shown in the
//Text box
0 references
public void UpdateText()
{
    _buttonCounter++;
    textMesh.text = "Score: " + _buttonCounter;
}
```

Sebastian Grygorczuk - STEM Institute at CCNY

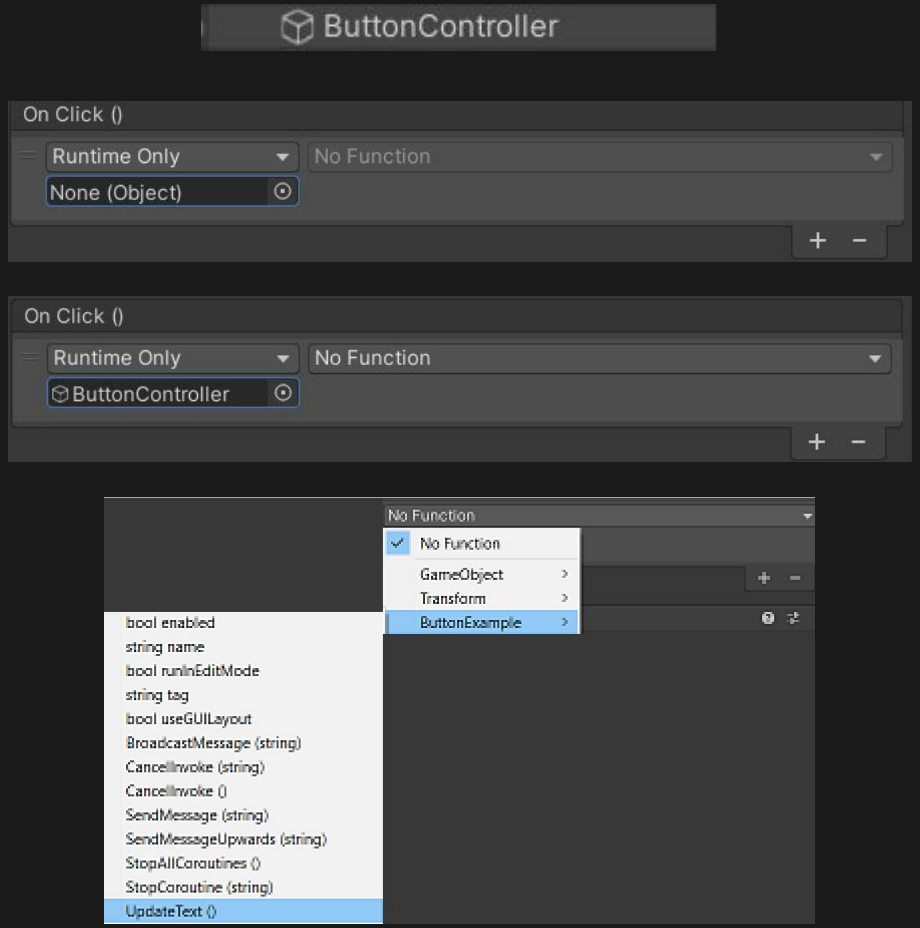
Connect Button Action To Text

To connect the script you will have to have a game object in the heichy that has the script connected to it, in this case Button Controller.

Then you can click on the plus button which will unlock a space to add an object into.

You will drag the game object into that space.

Finally you will be able to select the script component and pull the Update Text Method we created.



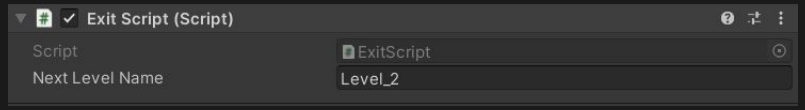
Sebastian Grygorczuk - STEM Institute at CCNY

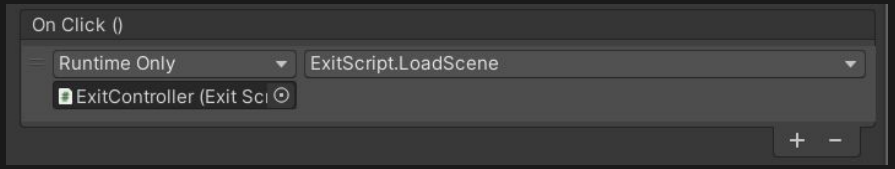
Move Between Scenes

You can move between scene you'll have to set up a using UnityEngine.SceneManagement;

Once you have that you'll be able to call a SceneManager class and using the LoadScene you can pass in a string with the name of the level you want to go to.

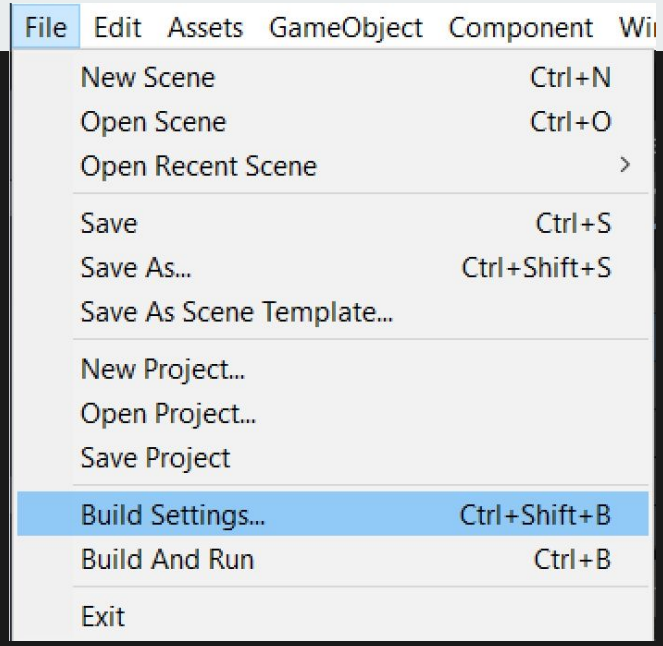
Following the example of the button we can set up movement between scene in the same fashion.

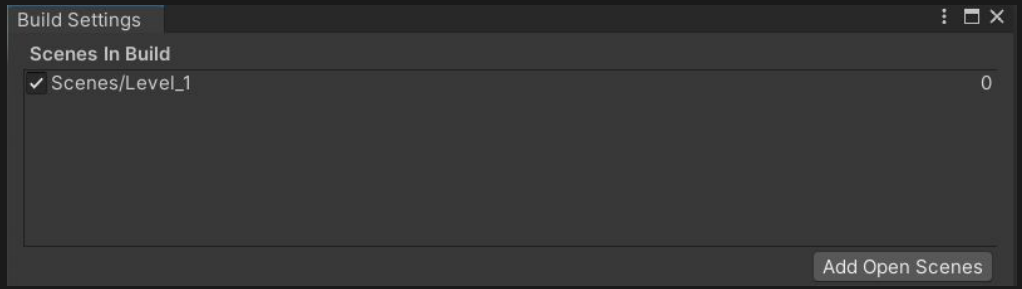




Build Settings

For you to be able to move between scene the scene have to be added in the Scene in Build in the Build Setting inside the File Drop down Menu



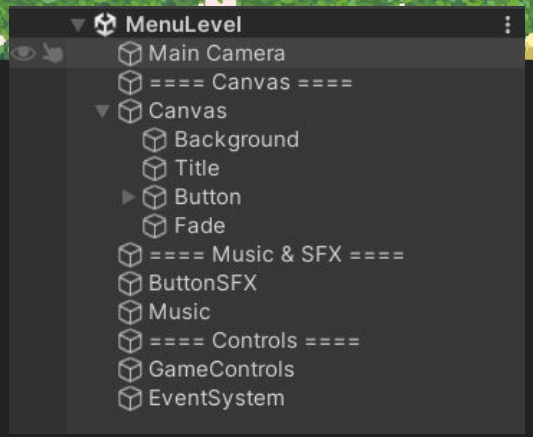


Main Menu

Starting with Main Menu

First, we will create our MenuScene. We'll have three important sections here:

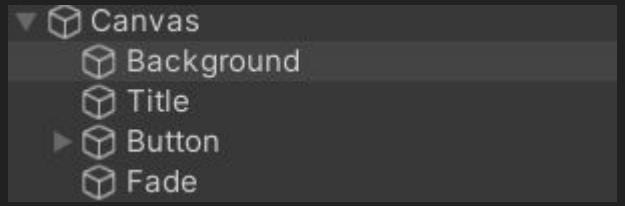
1. Canvas
2. Music
3. Game Controls

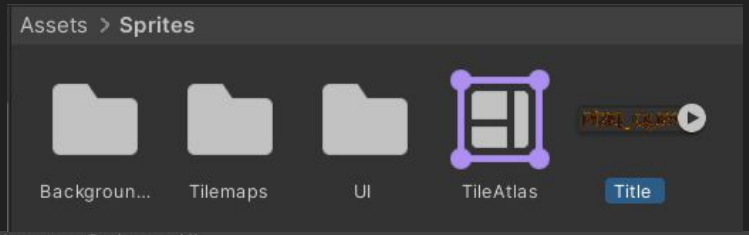


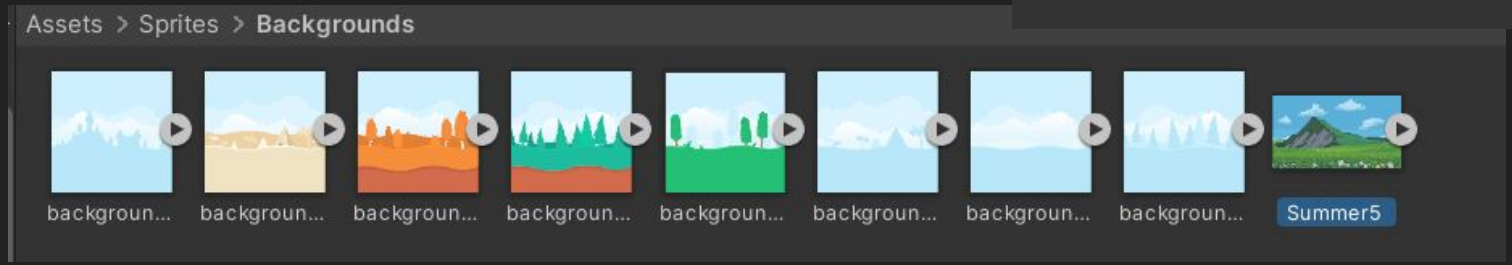
Canvas

The "Canvas" is broken down into four objects. Two of them are images (Background, Title), one is a button, and one is a panel that we'll use to fade to black when exiting the screen. That will be provided to you.

You will find all the images in the "Sprites" folder. The background and title are ready for use, but for the button image, you will have to cut it out.

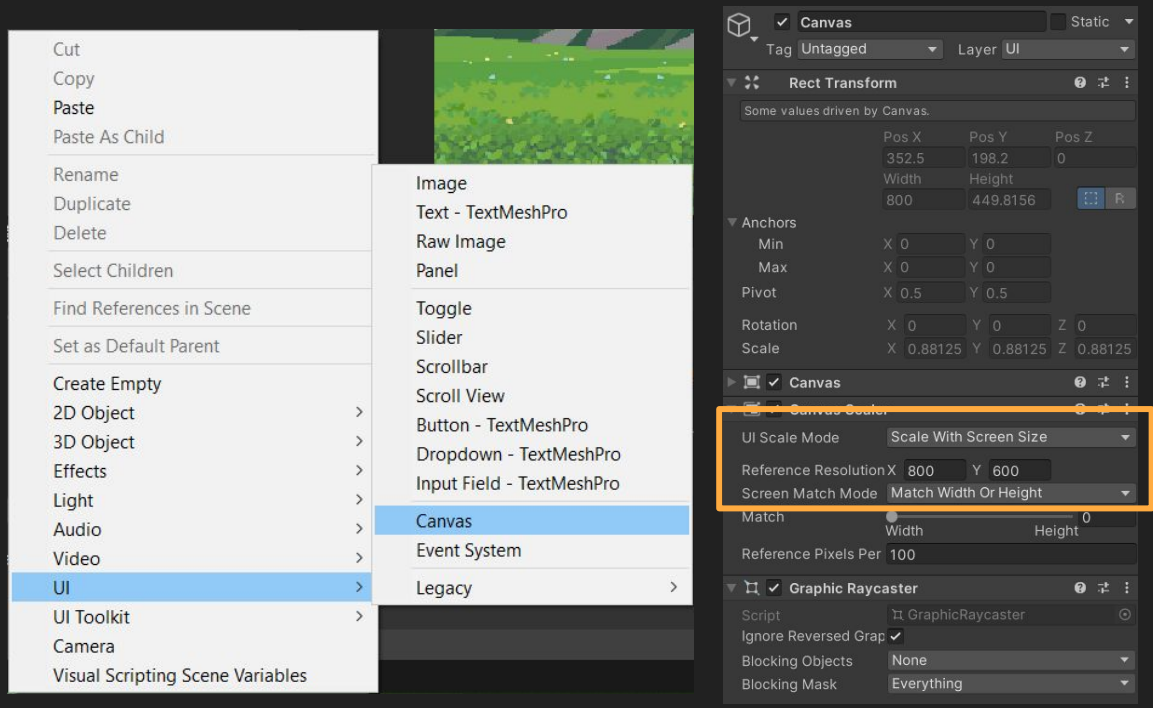






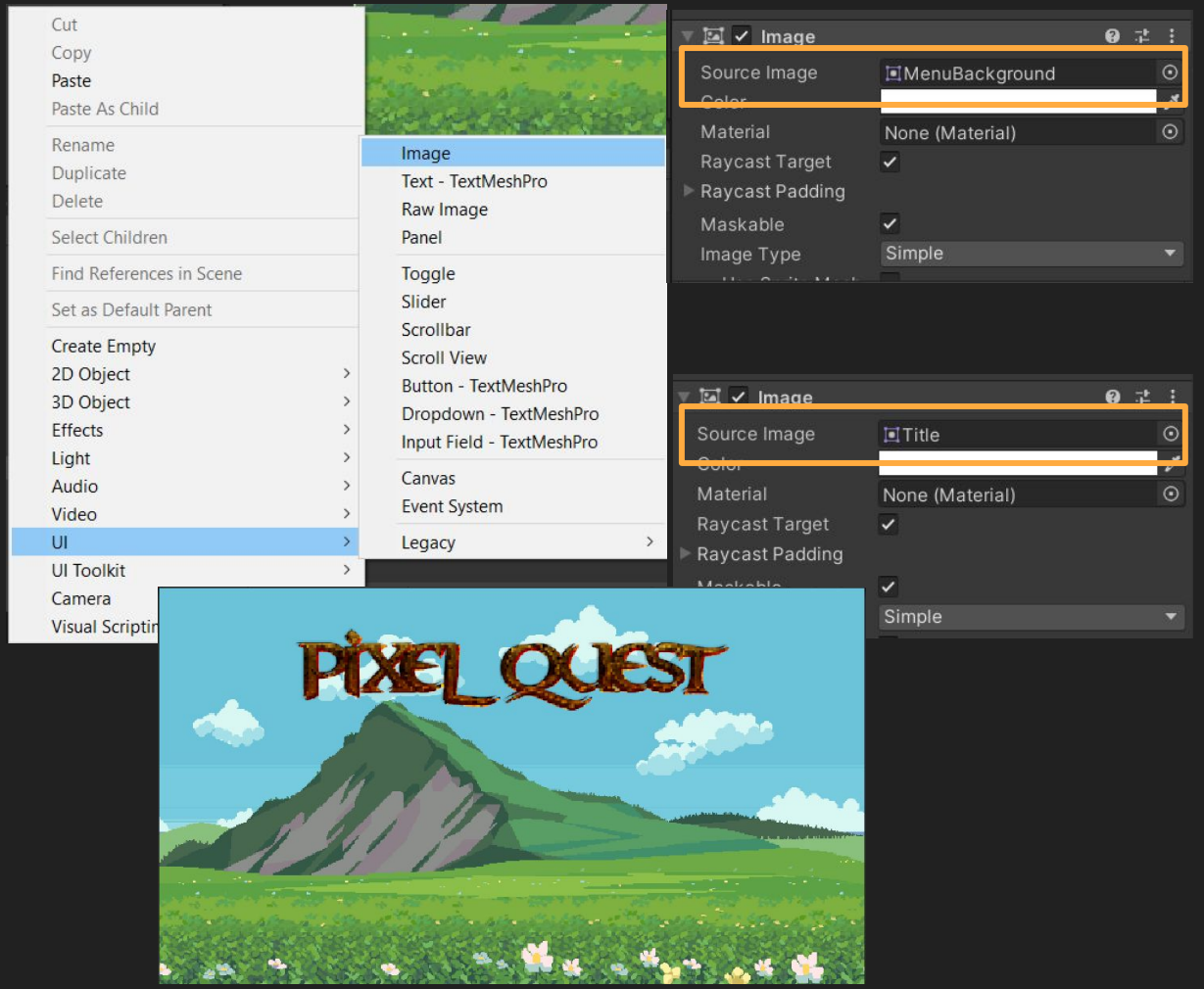
Canvas

Start by creating a canvas in the hierarchy. Make sure to change the settings to scale mode "Scale with Screen Size."



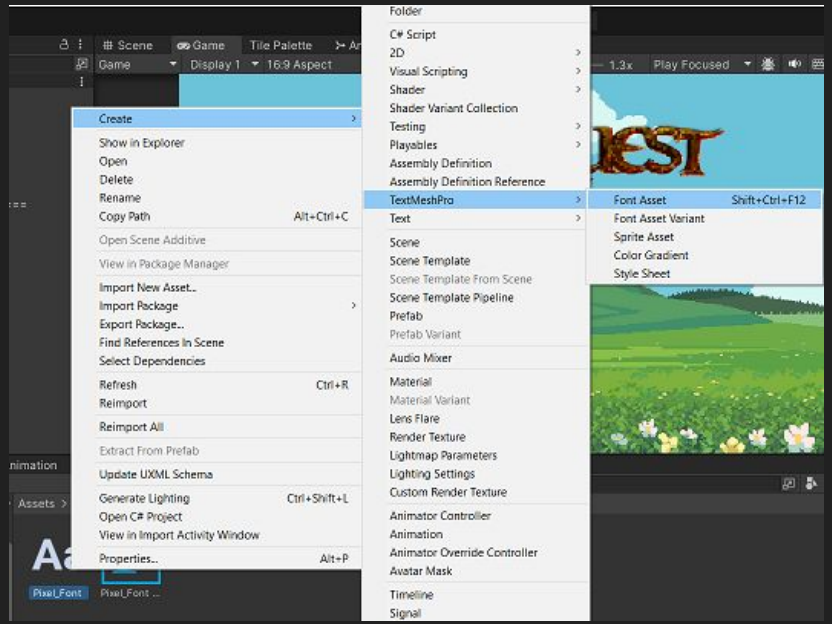
Background and Title

Create two Image game objects that are parented under Canvas. Set the Source Image to be "MenuBackground" for one and "Title" for the other, ensuring that the Background game object is placed on top, as the order matters in how the objects appear.



Button Font

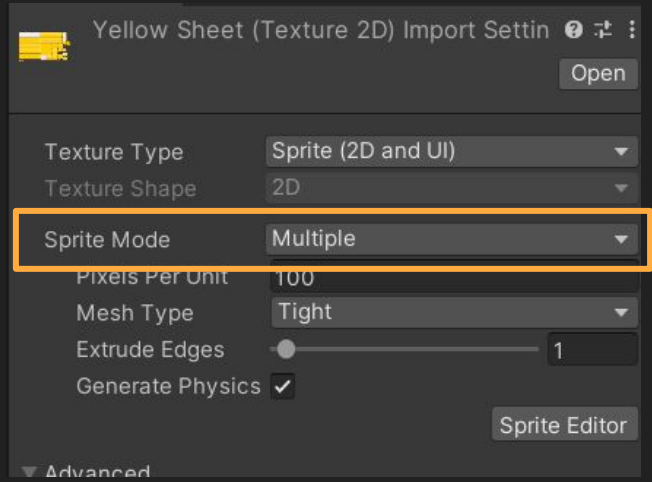
Before we create a button, we need to set up our assets. Start by going to the Font Folder, right-click on "Pixel Font," and create a TextMeshPro Font Asset.



Button Image

We're going to cut out one of the images from the Yellow Sheet UI Sprite Sheet.

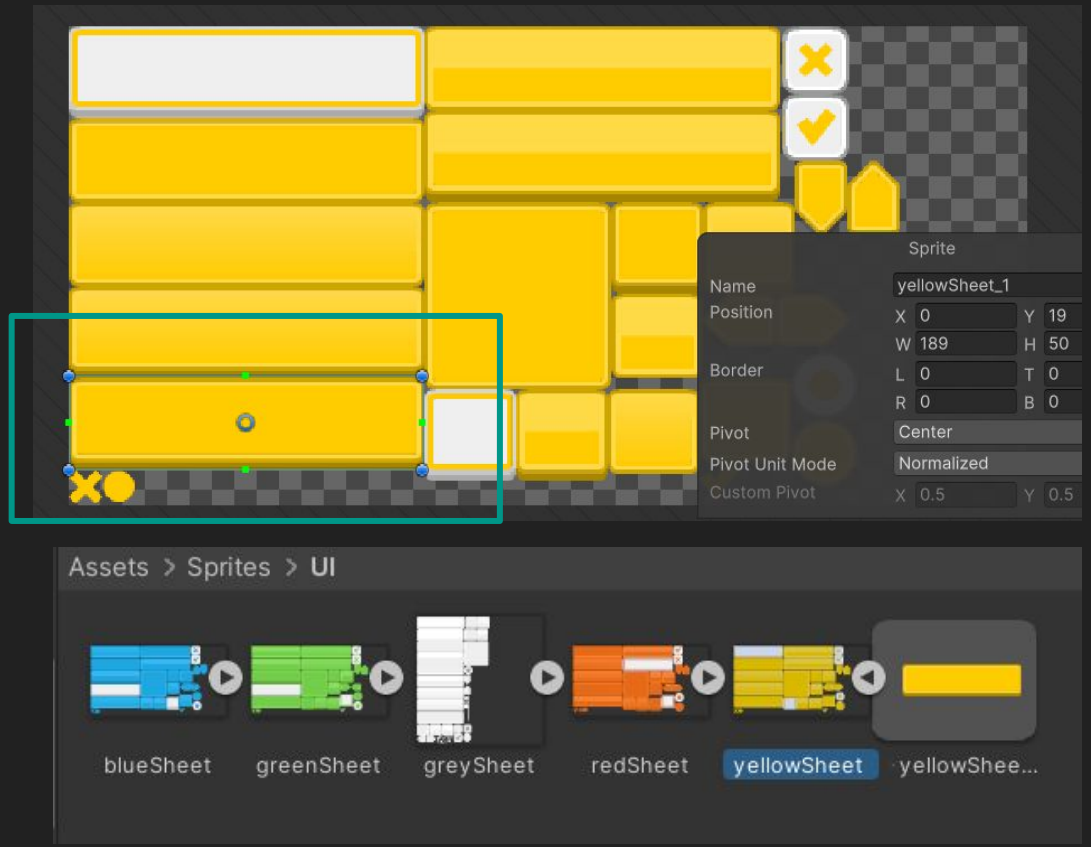
First, select it and set Sprite Mode to be multiple. With that done, we can go into the Sprite Editor.



Button Image Sprite Sheet

Hold down the left mouse button and drag to create a rubber band. Encase the bottom-left button in the band.

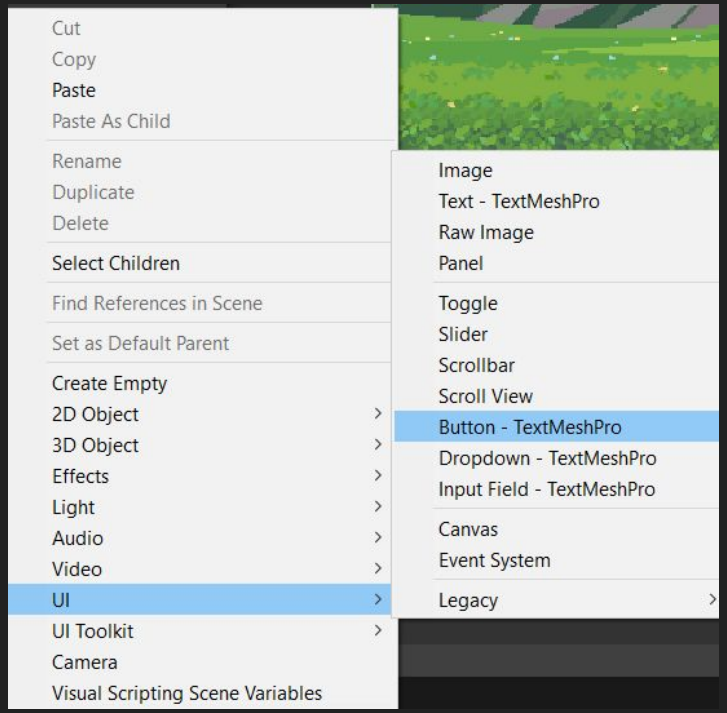
Once you've correctly contained it, make sure you can apply the changes. You'll see that the sprite sheet now has that button cut out.

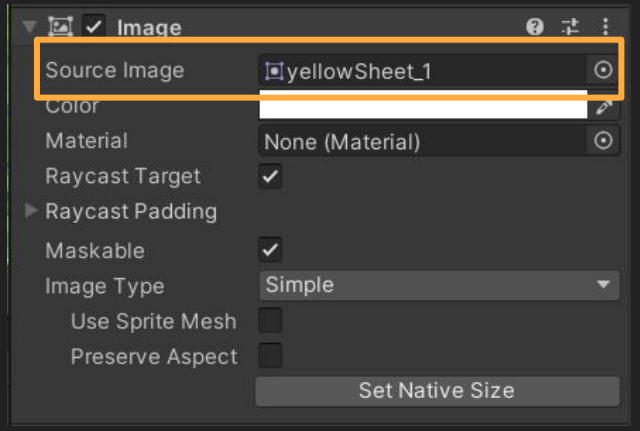


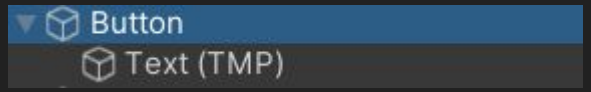
Button

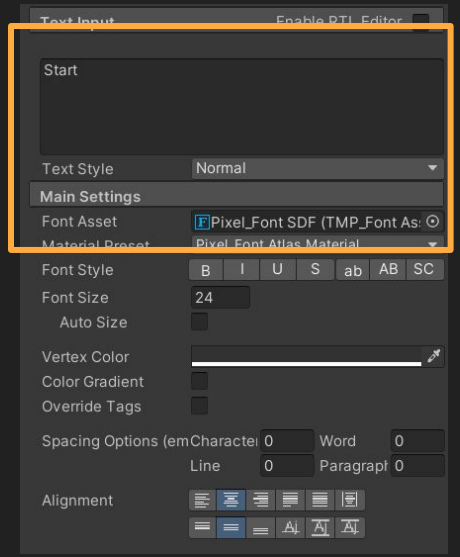
Once you have all of the game assets preformatted, we're going to create the button.

When you create it, you will have two objects: "Button," in which you will change the Source Image to your newly cut-out yellow button image, and "TextMeshPro," where you'll change the text to "Start" and the Font Asset to our new Font Asset.







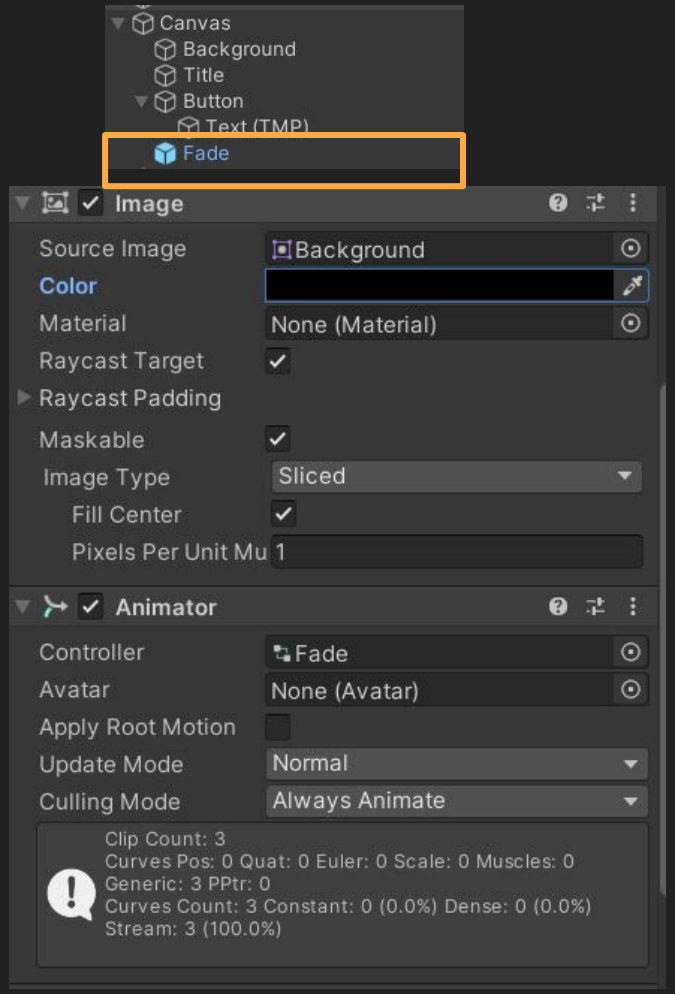


Fade

You will have one game object that is already created for you: "Fade," a Prefab that will allow us to achieve a smoother transition between levels and when the player dies.

You will notice that this object has an Image with an alpha of 0, making it see-through.

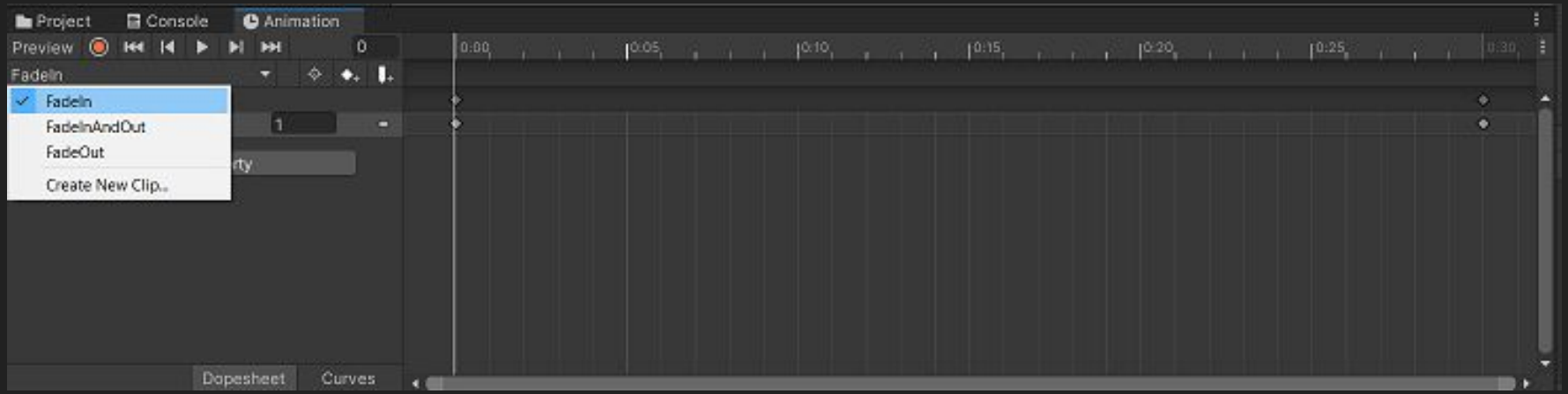
We will use the Animator in conjunction with the Fade Animation Controller to animate the image, transitioning it from 0 to 1 alpha and back in different situations. Drag the Prefab under the Canvas so that its effects are visible in the game.



Fade

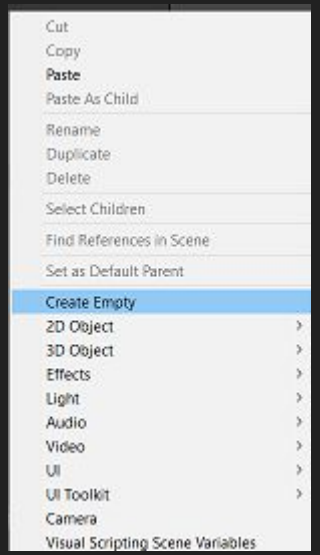
You will notice that this object has three animations: FadeIn, FadeOut, and FadeInAndOut. FadeIn goes from black to see-through, FadeOut does the opposite; these are used for loading between levels.

FadeInAndOut is for when the player dies and needs to be moved between checkpoints.



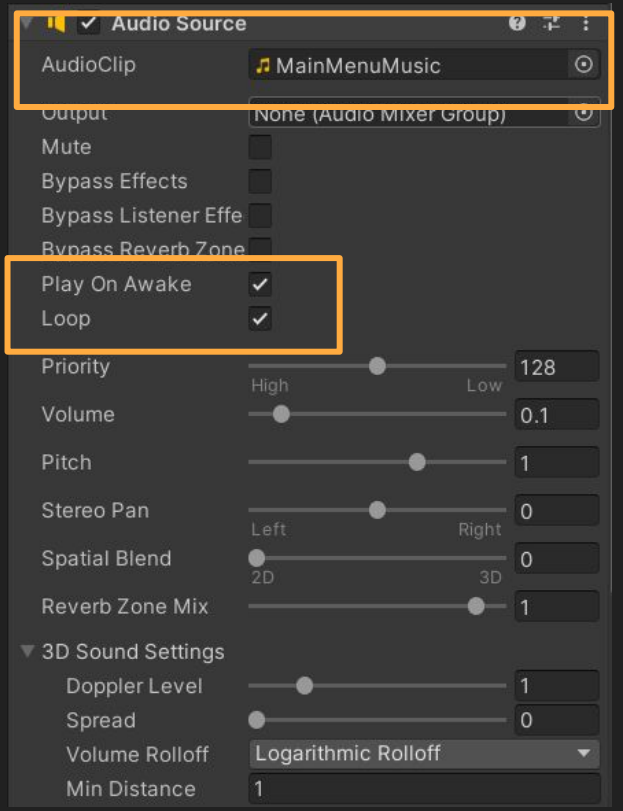
Music and SFX

aCreate two empty game objects, and we're going to fill them with Audio Sources. Name them "Music" and "Button SFX."



Music

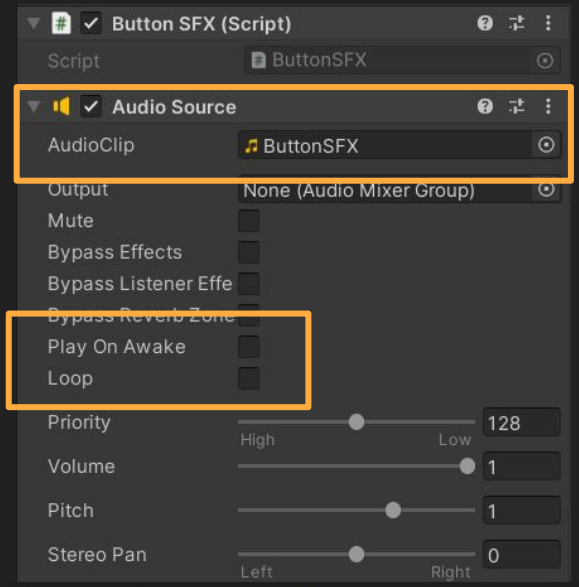
For "Music," add an Audio Source, use the "MainMenuMusic," and make sure that both "PlayOnAwake" and "Loop" are checked so that the music starts playing from the start of the game and continues in a loop.



Button SFX

For the "Button SFX," also add an Audio Source, use the AudioClip "ButtonSFX," and make sure that both "Play On Awake" and "Loop" are turned off.

Additionally, add a script for the button. The script will make the button play when triggered. We will attach it to the button in a moment. Take a look at the script to understand how it works.

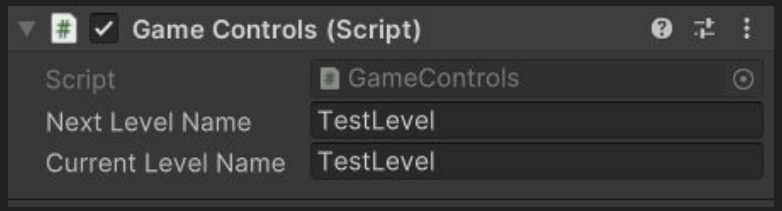



```
Unity Script (1 asset reference) | 0 references
public class ButtonSFX : MonoBehaviour
{
    private AudioSource buttonSFX;

    Unity Message | 0 references
    private void Start()
    {
        buttonSFX = GetComponent<AudioSource>();
    }

    0 references
    public void PlayButtonSFX()
    {
        buttonSFX.Play();
    }
}
```

Game Control

"Game Controls" will be an empty object that holds a script enabling you to load between scenes. In this script, there are two public variables: one for reloading if the player dies and another for loading into the next scene when transitioning.

The script is ready for you, but feel free to take a look at how it works.

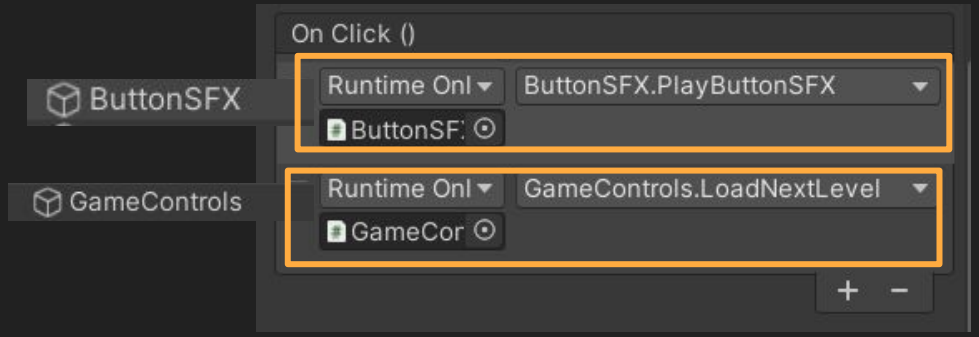



```
public class GameControls : MonoBehaviour
{

    public string nextLevelName = "TestLevel";
    public string currentLevelName = "TestLevel";
    private GameObject _fadeObject;
    private Animator _animator;

    Unity Message | 0 references
    private void Start()
    {
        _fadeObject = GameObject.Find("Fade");
        _fadeObject.SetActive(false);
        _animator = _fadeObject.GetComponent<Animator>();
    }

    0 references
    public void LoadNextLevel()
    {
        StartCoroutine(WaitForFade(nextLevelName));
    }

    0 references
    public void ReloadLevel()
    {
        StartCoroutine(WaitForFade(currentLevelName));
    }

    0 references
    public void LoadMenu()
    {
        StartCoroutine(WaitForFade("MenuLevel"));
    }

    3 references
    private IEnumerator WaitForFade(string levelName)
    {
        _fadeObject.SetActive(true);
        _animator.Play("FadeOut");
        yield return new WaitForSeconds(0.5f);
        SceneManager.LoadScene(levelName);
    }
}
```

Button Functionality

The last part of creating the Main Menu is making the button work.

Go back to the Button in the canvas, scroll down until you get to OnClick, click the plus button twice, and connect the "ButtonSFX" game object and the "Game Controller" object. From there, you will add the functions "PlayButtonSFX" and "LoadNextLevel."



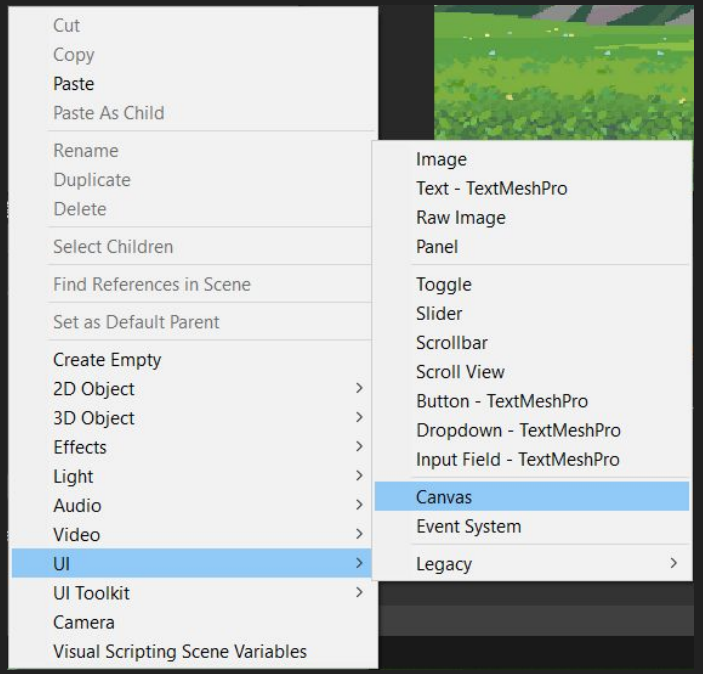
Level UI

Canvas

There are four UI elements that we will want in our canvas:

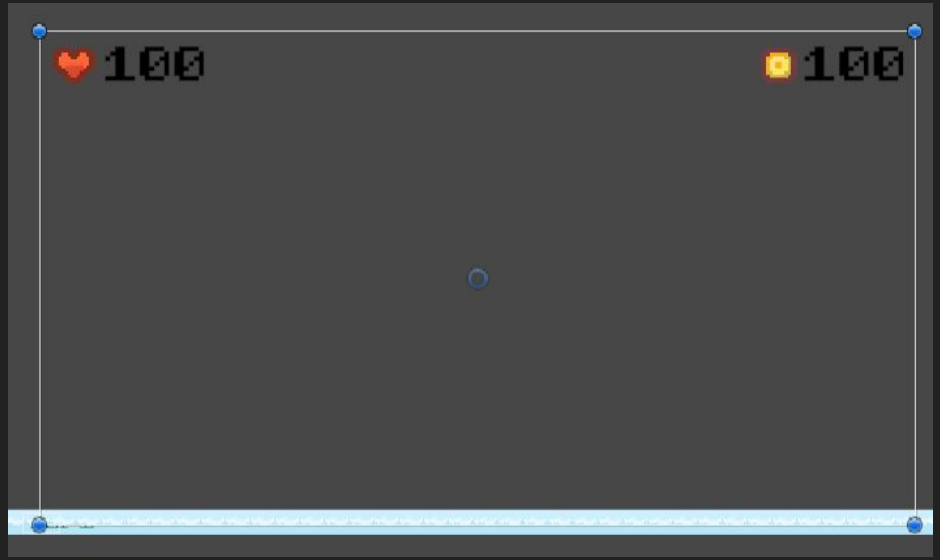
1. Game Over UI
2. Level UI
3. Victory UI
4. Fade

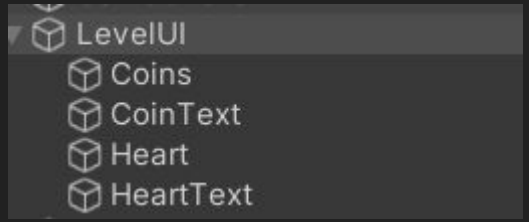
Like before, we will use Fade for transitions. The rest of them, we will construct from the ground up.



Level UI

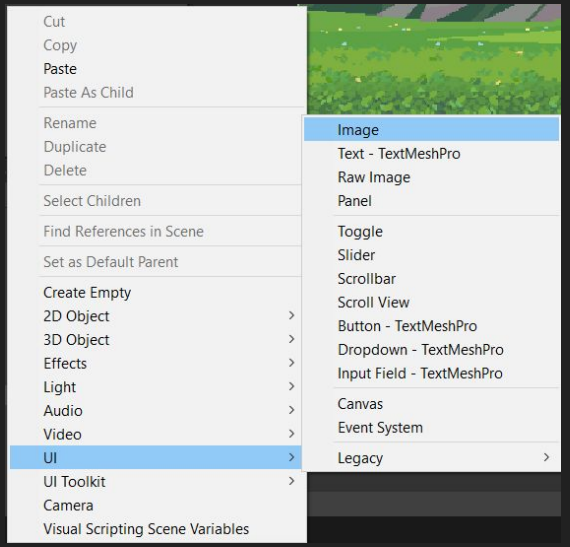
We will start with the Level UI; this is what the player will see when playing the game. We will have two Images and two TextMeshProGUI elements in it.

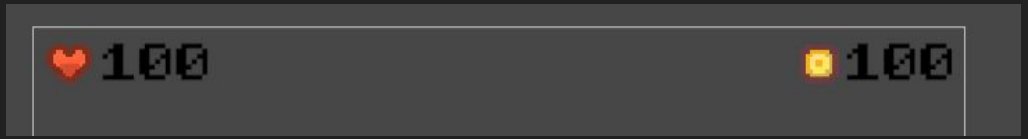


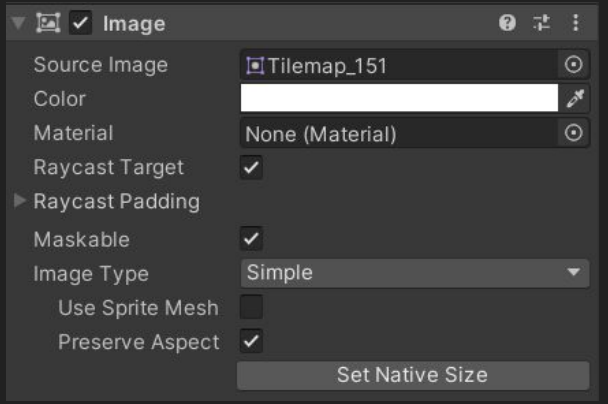


Images

Let’s start with the images; we will create two images, one for the heart and one for the coin, so the player knows what the numbers are associated with.





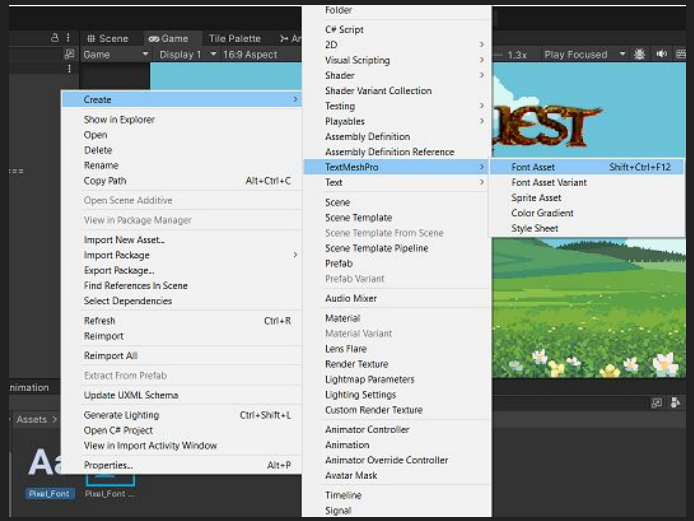


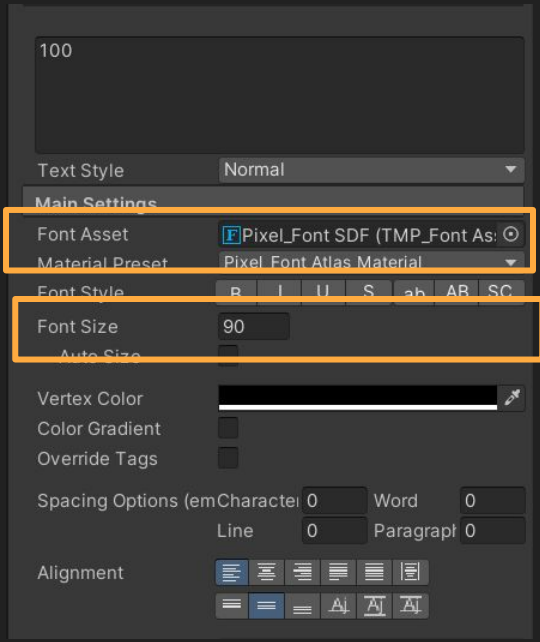
TextMeshProGUI

Next, create two TextMeshProGUI game objects. Make sure to name them CoinText and HeartText; the code looks for those specific names.

You can set the number in them to be anything; it will be changed using code.

Attach the Font Asset we created before and set the alignment to be left-bound, with the font size being 90.

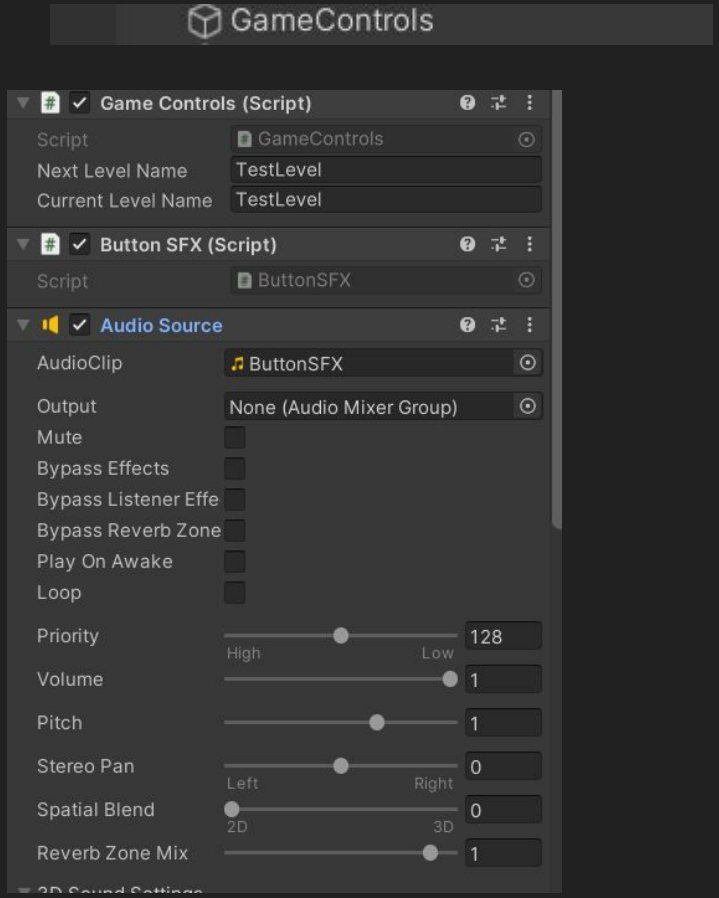




Game Controller

Create an empty object and call it "Game Controller." We will use this for functionality between levels.

Add the "Game Controls" script and "Button SFX" script alongside an Audio source with the ButtonSFX AudioClip. We will use these in the buttons for the UI.



Game Over

The Game Over UI will appear whenever the player loses all of their hearts.

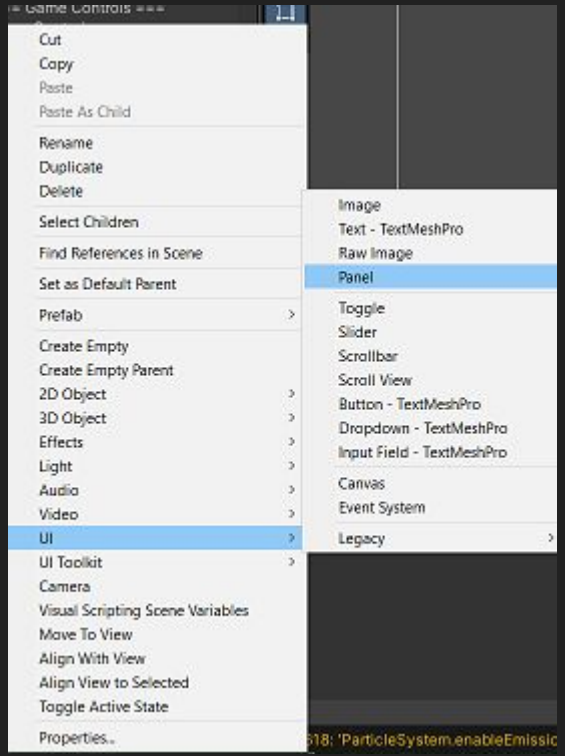
It will have two buttons: one to quit back to the main menu and another to reload this level. Visually, it will have a TextMeshProGUI and an image in the middle to give it some flare.

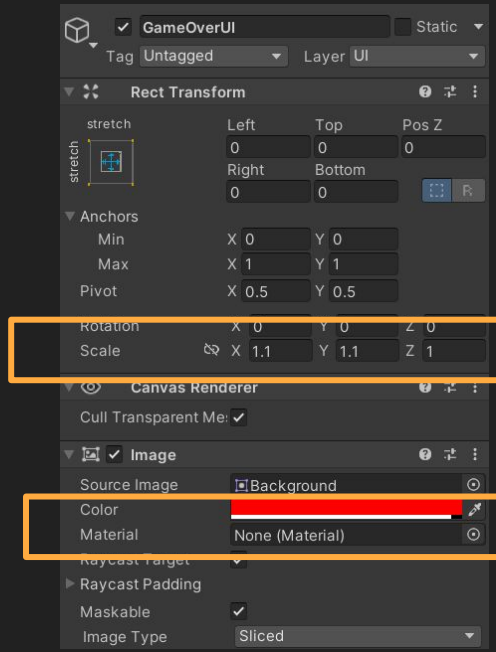
Game Over

Start by creating a Panel; it will create an Image object that covers up most of the screen. Make sure to call that object "GameOverUI"; the code will look for that name.

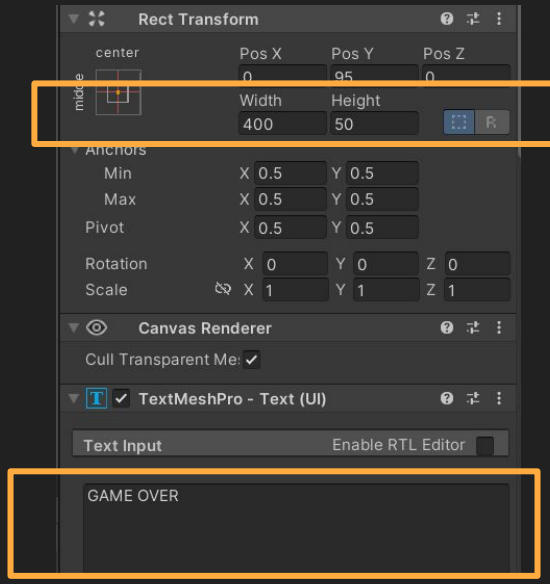
In the image settings, make it almost fully red, with some alpha to allow the player to slightly see the level.

And set the scale to be 1.1 to make sure it covers all of the screen.



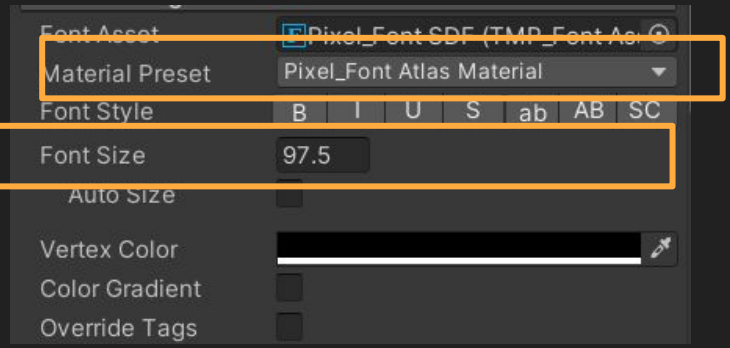


TextMeshProGUI



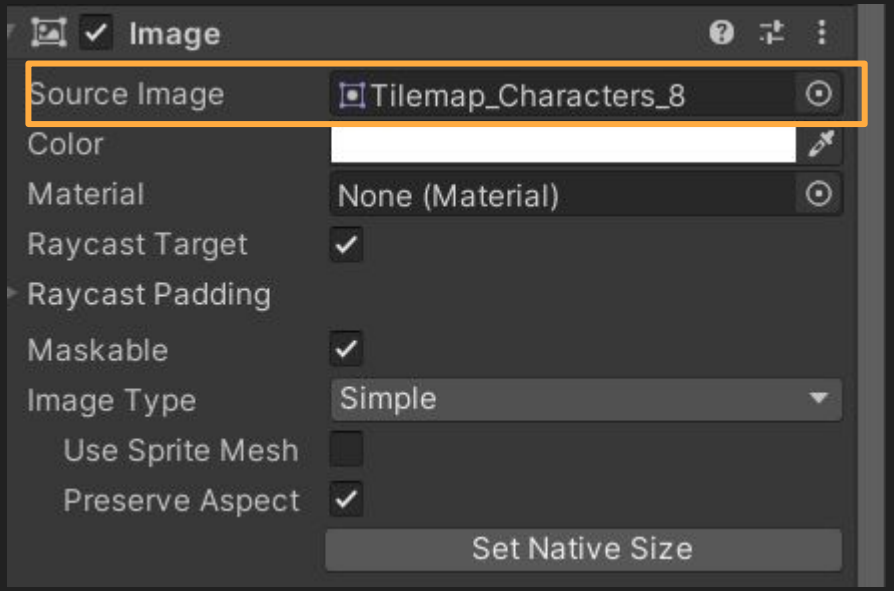
Create a TextMeshProGUI game object, make the width 400 so the larger text fits.

Make the text say "GAME OVER," and attach the Font Asset we created while setting the font size to be around 98.



Image

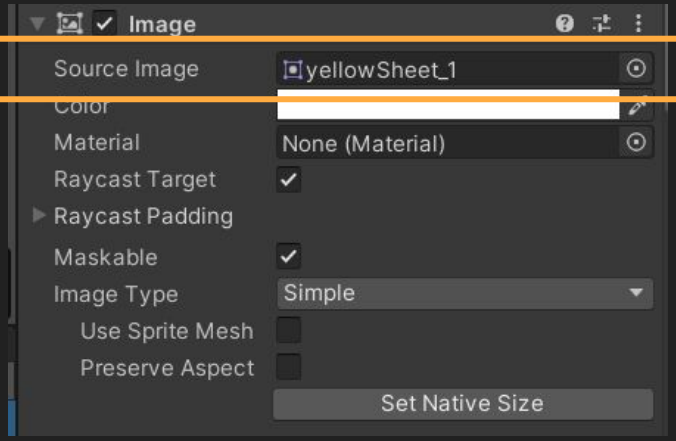
Create an image and place the Tilemap_Character_8. This will add some flare to the screen.

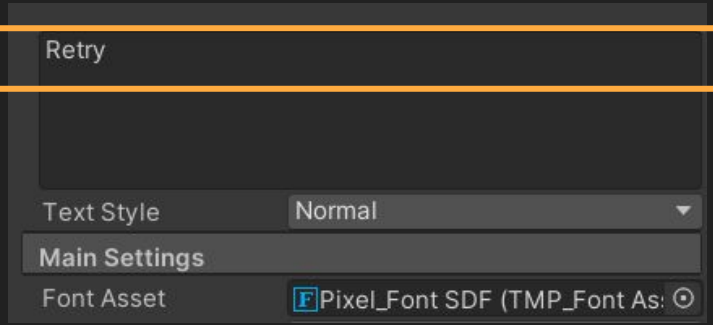


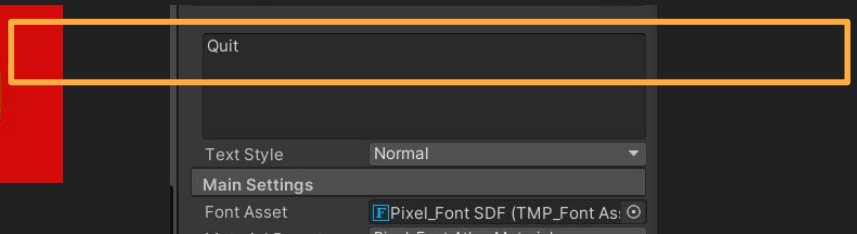
Buttons

Create two buttons; make sure that they have the yellow image attached to them like we did in the main menu.

And set the text to be "Retry" on the right and "Quit" on the left.



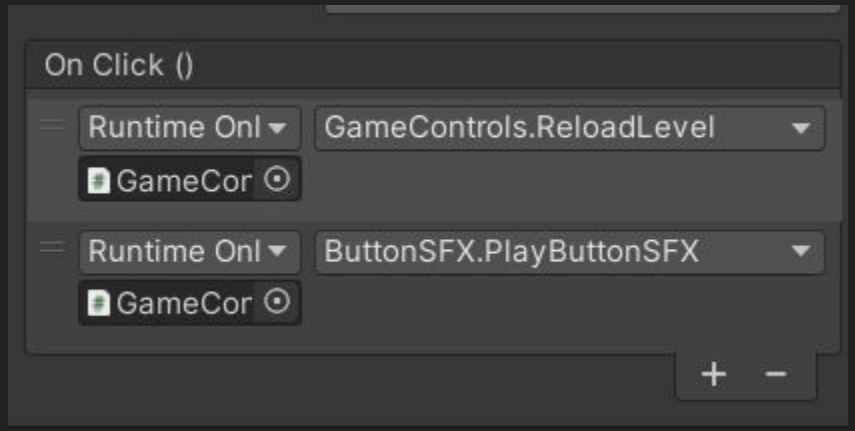


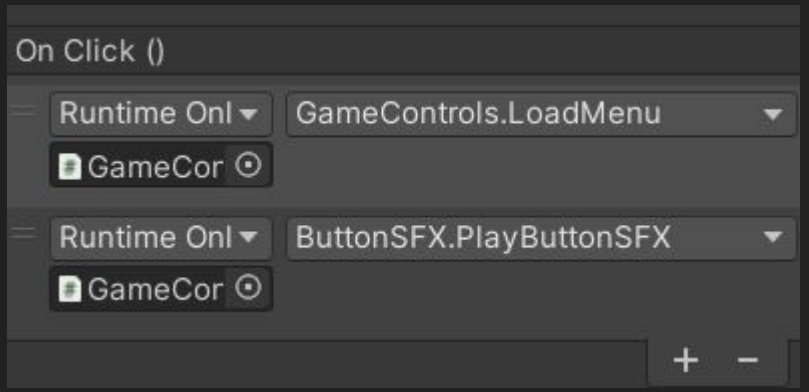


Button Functionality

In both buttons, add two function reactions. Both will use the code that's connected to the GameController game objects.

The Retry Button will use the "ReloadLevel" function, while the Quit Button will use the "LoadMenu" function. Both will have a "PlayButtonSFX" method connected to it.





Victory UI

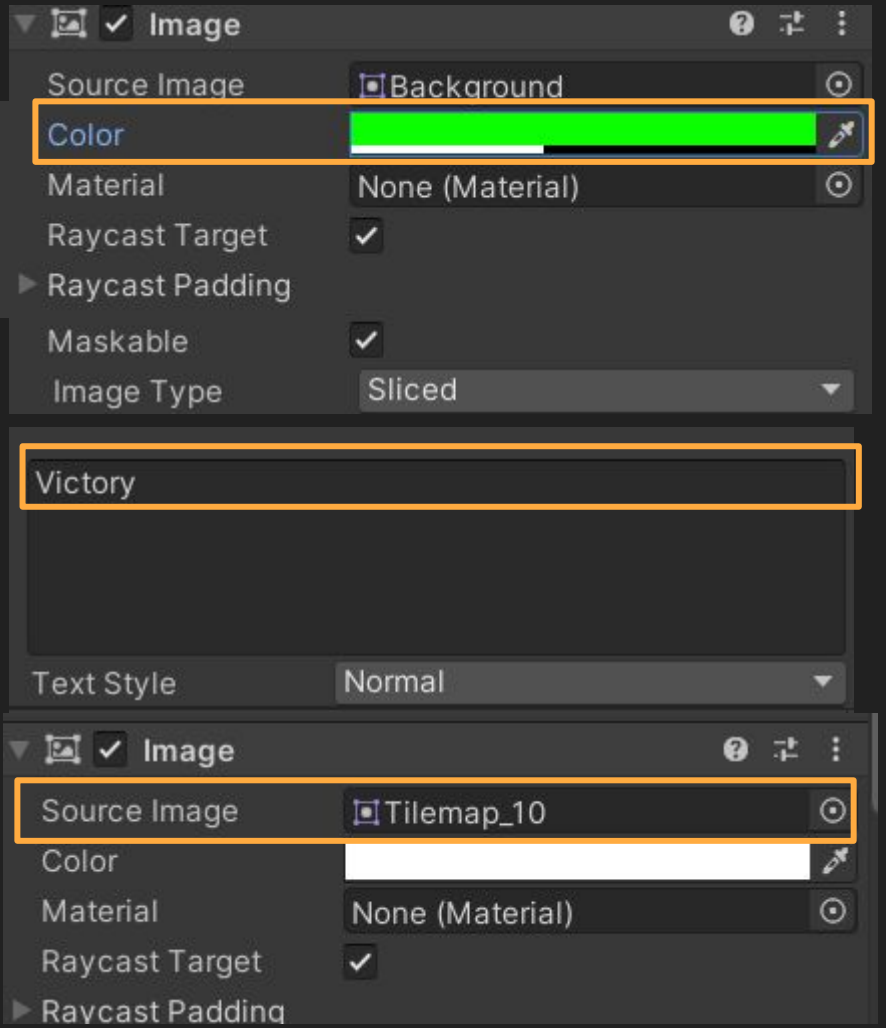
The Victory UI we are creating is basically the same as the Game Over UI, so we'll duplicate the Death UI and tweak it to work the way we want it to.

Victory UI

Make sure to name the parent "VictoryUI"; the code will look for that name.

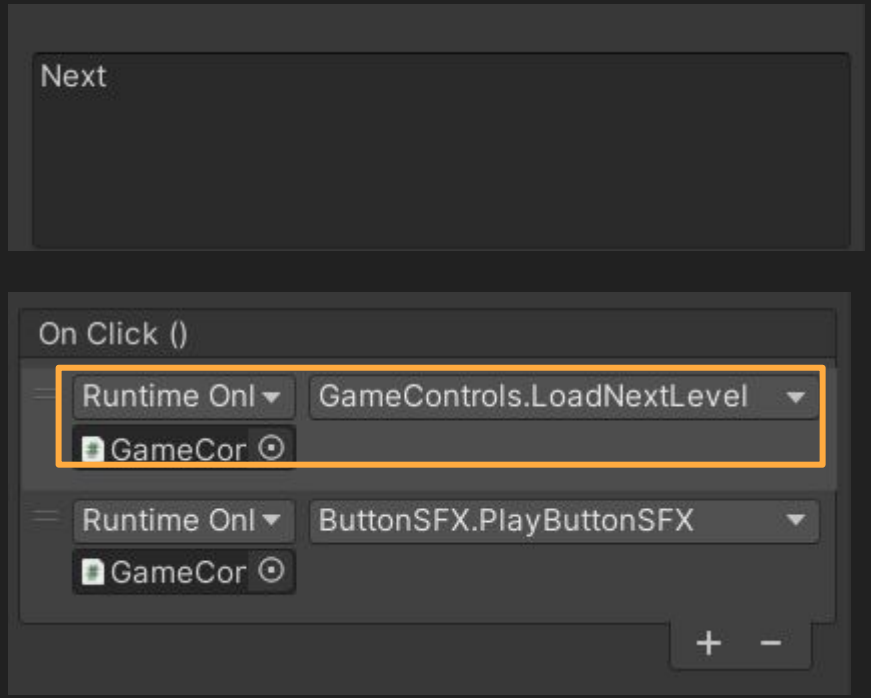
Set the color of the background image to be green.

Change the "Game Over" text to "Victory." Also, change the image in the center to a cube with an exclamation mark.



Victory Button

For the buttons, we will only change the Retry Button to the Next Level button, change the text to say "Next." And in the On Click area, change the "ReloadLevel" function to "LoadNextLevel."



Fade

Like we did in the menu, add the Fade prefab; it will fade in and out to make the gameplay feel smoother.

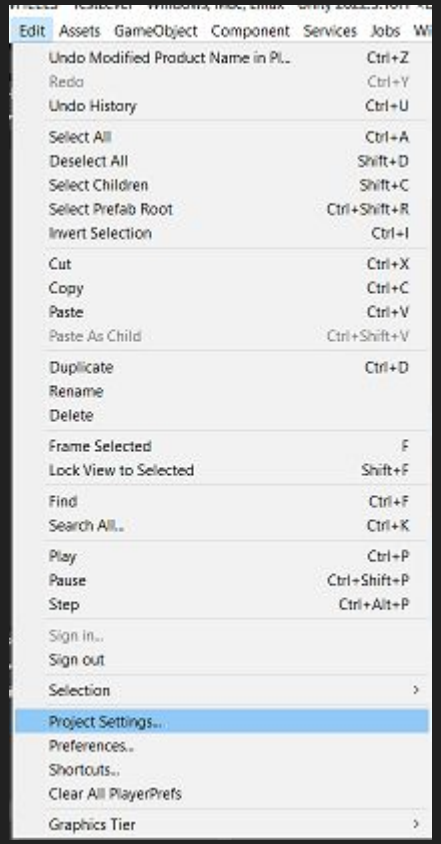
Projects Settings & Building the Game

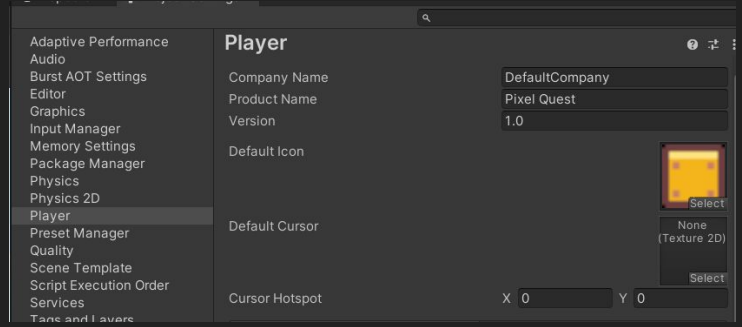
Project Settings

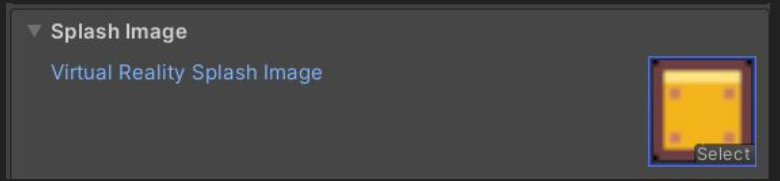
Open up the project Settings; here, you can go to the Player tab.

You can change the name of your game, set an icon that will be seen with the exe.

Set the Fullscreen Mode (I recommend Windowed), and splash image of your logo.







Game Build

Once you're happy with the game, you can open up the Build Settings window.

Here, make sure all of the scenes are added. You want the Menu Scene on top of the list, as it's the one the game will open with, and then you can click build.

That will create an exe file you can bring to any PC and let you play your game.