Lab
Pixel Quest

Lab Goals

In this lab we will be putting together a game from scratch.

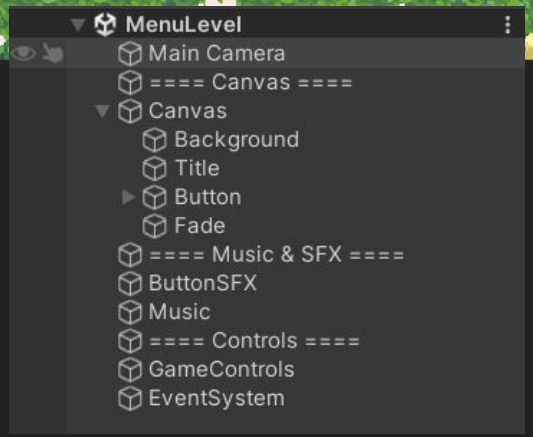
We will create a menu screen, and a level where you play a 2D platformer.

Main Menu

Starting with Main Menu

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

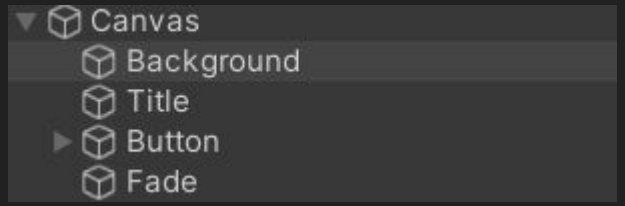
1. Canvas
2. Music
3. Game Controls

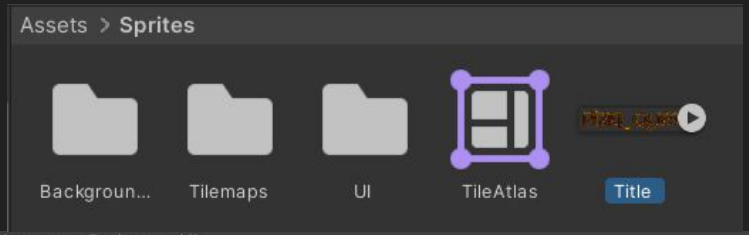


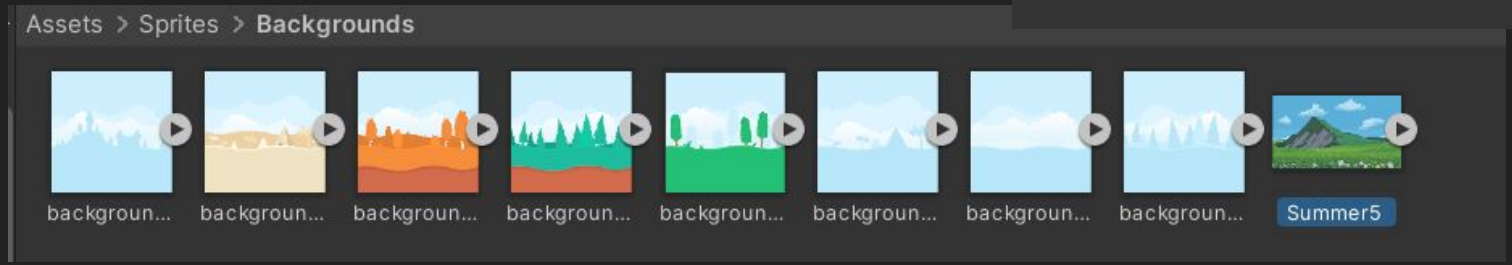
Canvas

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.

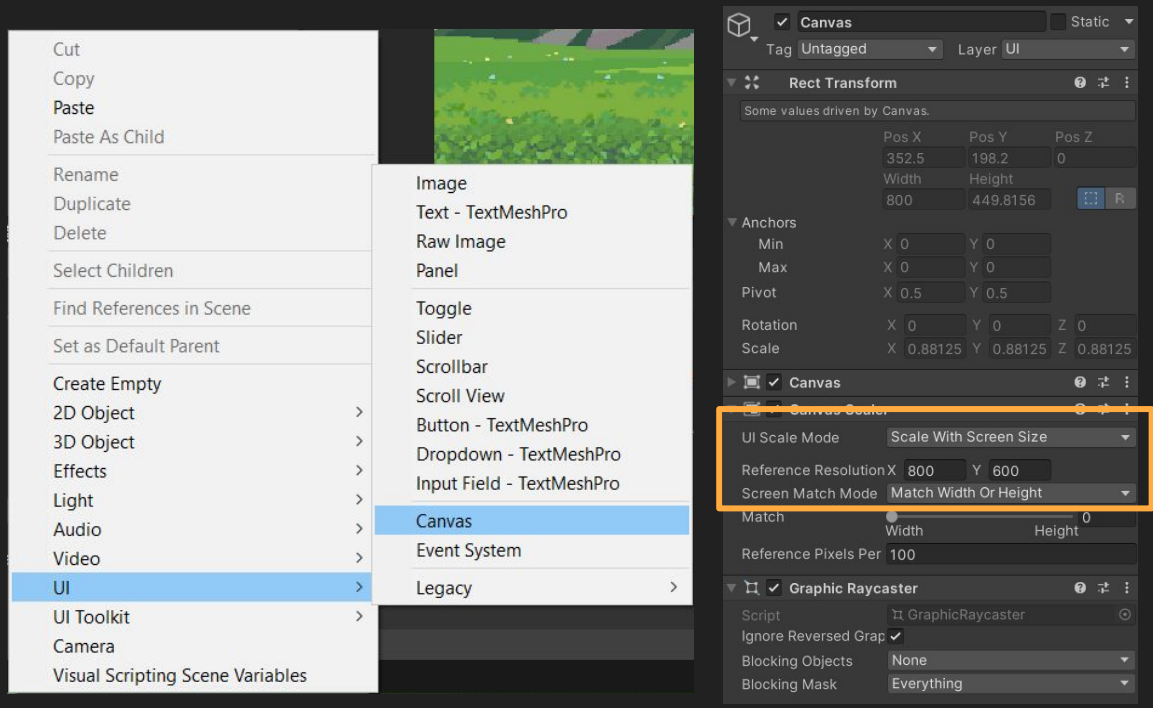






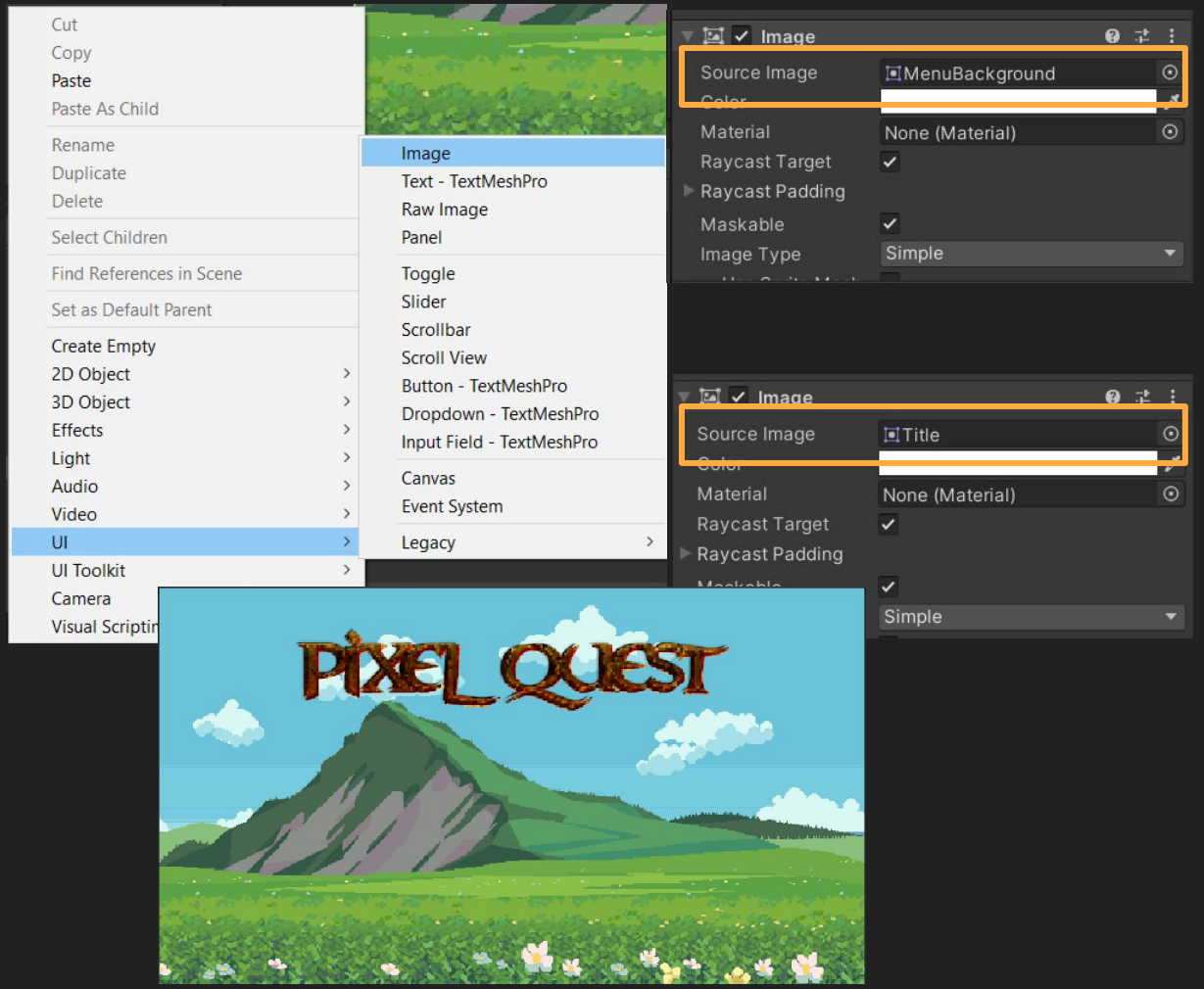
Canvas

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



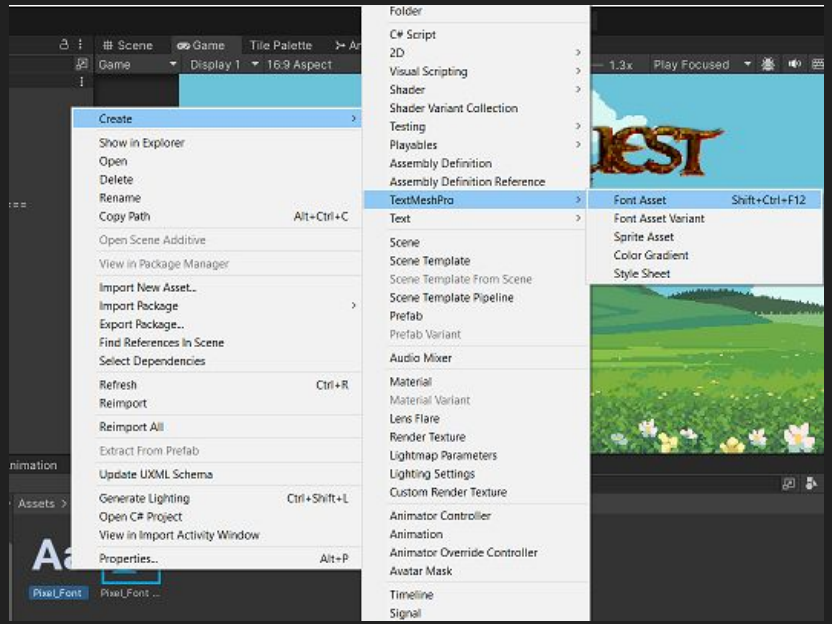
Background and Title

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

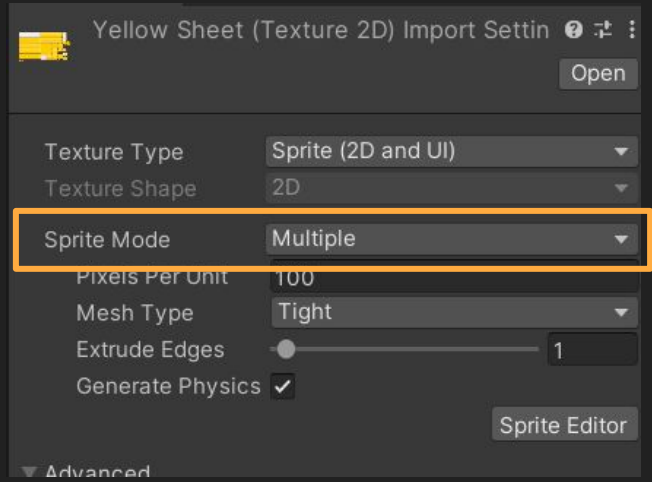
Before we create a button, we need to set up our assets. Start by going to the Font Folder, right-click on "Pixel Font," and create a TextMeshPro Font Asset.



Button Image

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

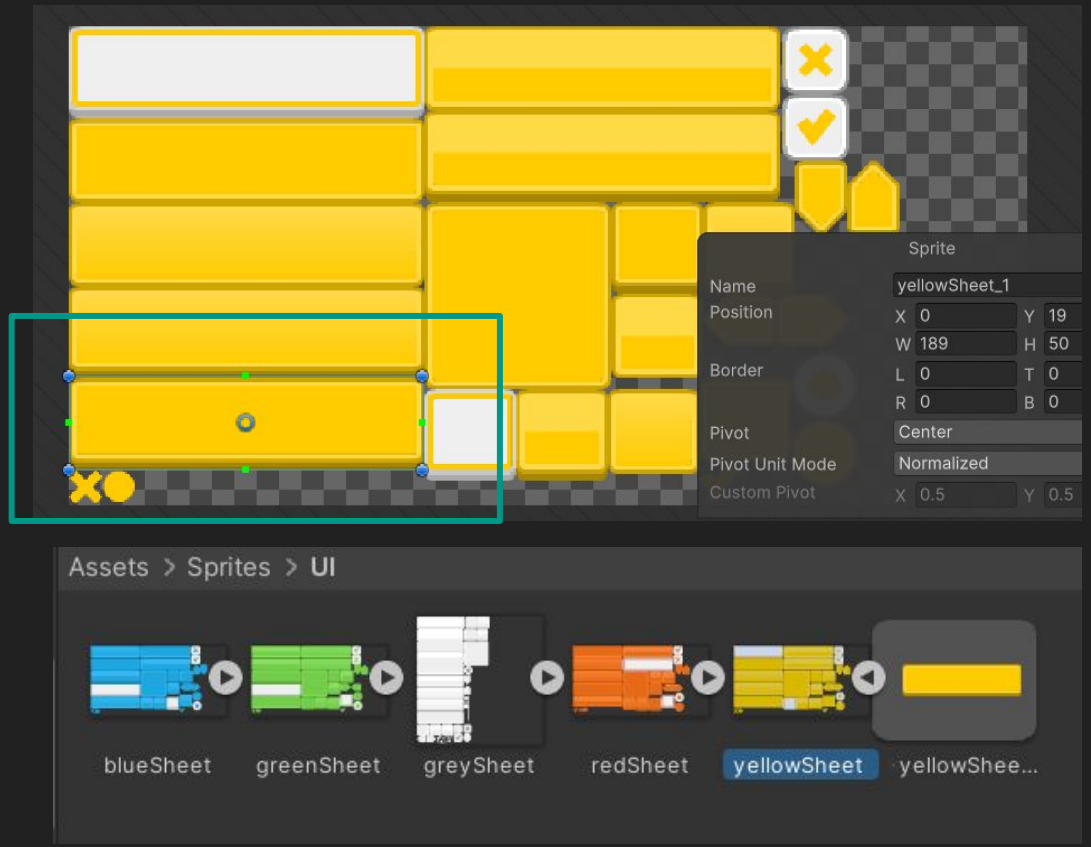
First, select it and set Sprite Mode to be multiple. With that done, we can go into the Sprite Editor.



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.

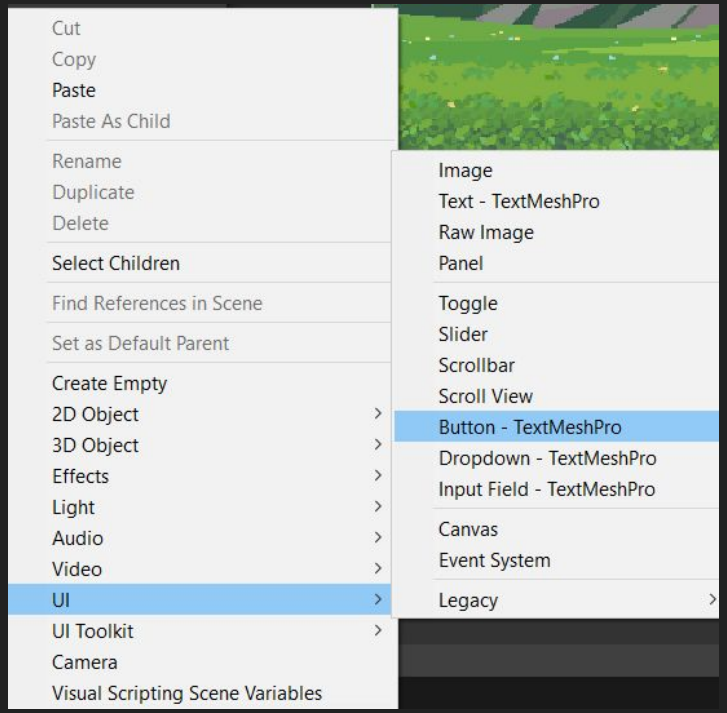
Once you've correctly contained it, make sure you can apply the changes. You'll see that the sprite sheet now has that button cut out.

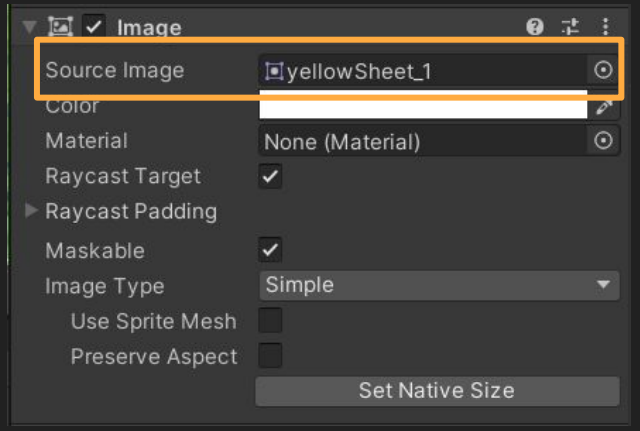


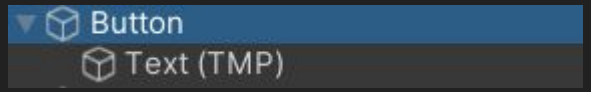
Button

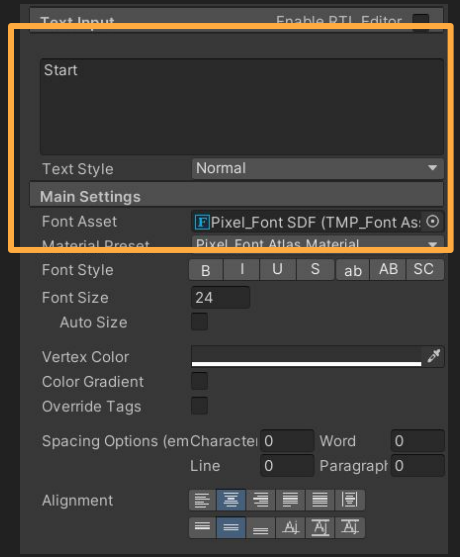
Once you have all of the game assets preformatted, we're going to create the button.

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.







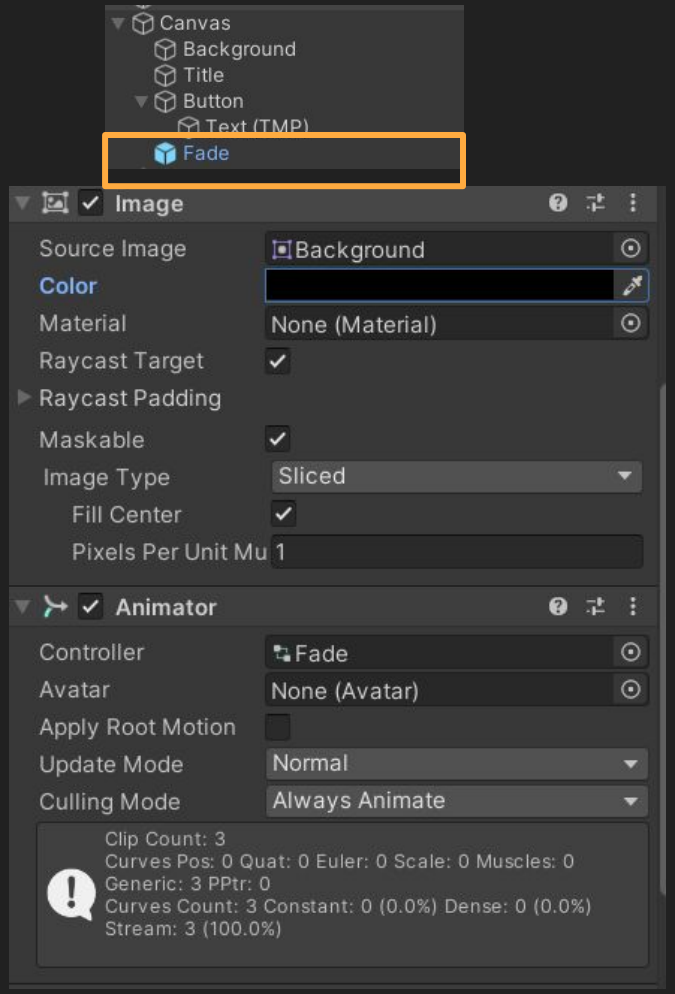


Fade

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.

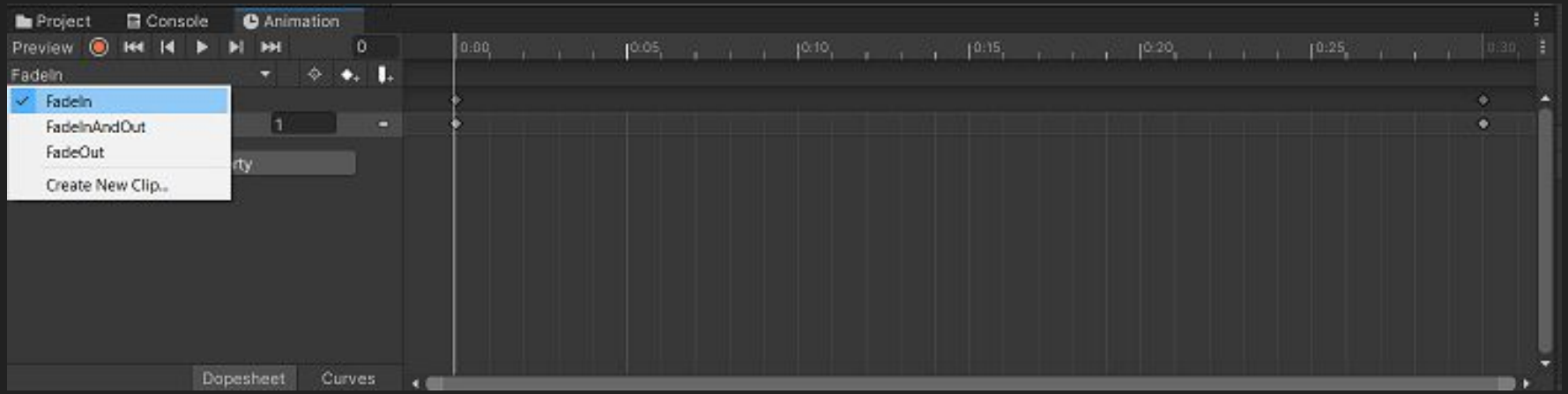
We will use the Animator in conjunction with the Fade Animation Controller to animate the image, transitioning it from 0 to 1 alpha and back in different situations. Drag the Prefab under the Canvas so that its effects are visible in the game.



Fade

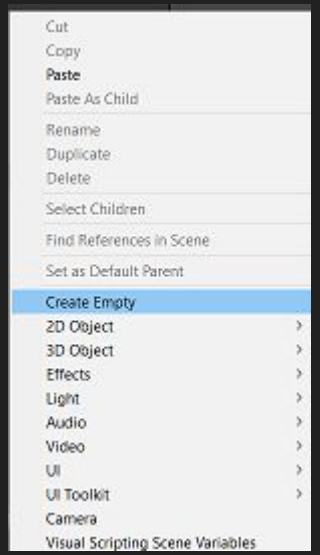
You will notice that this object has three animations: FadeIn, FadeOut, and FadeInAndOut. FadeIn goes from black to see-through, FadeOut does the opposite; these are used for loading between levels.

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



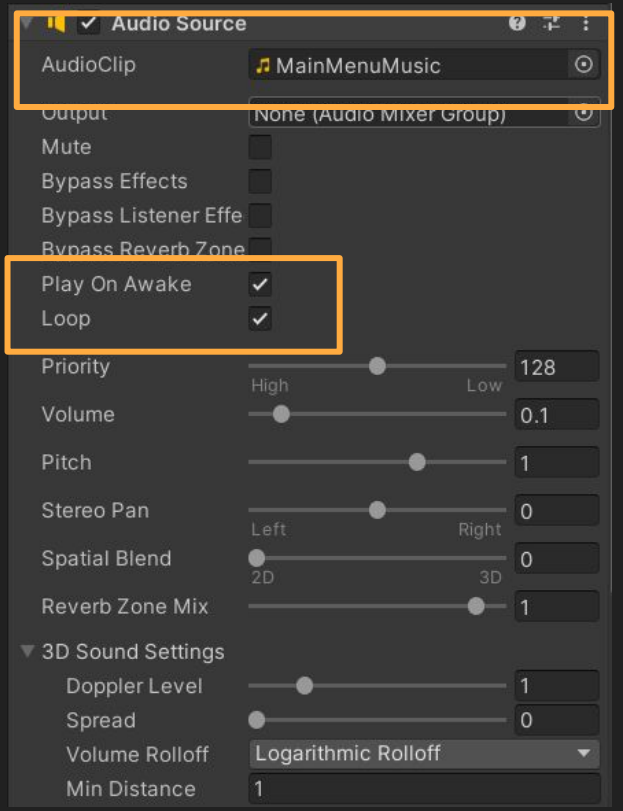
Music and SFX

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



Music

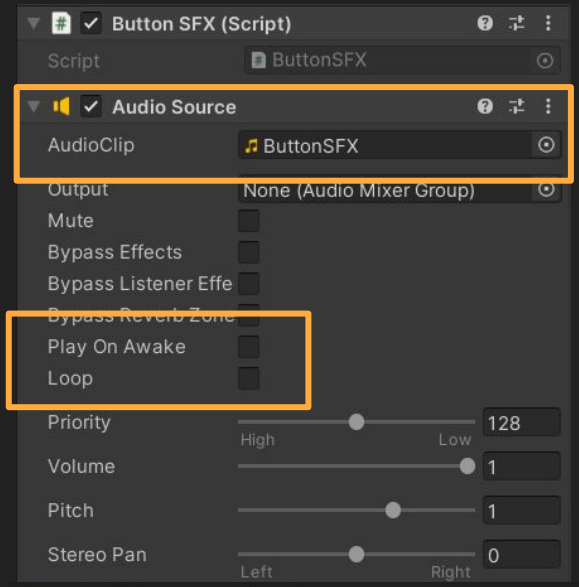
For "Music," add an Audio Source, use the "MainMenuMusic," and make sure that both "PlayOnAwake" and "Loop" are checked so that the music starts playing from the start of the game and continues in a loop.



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.

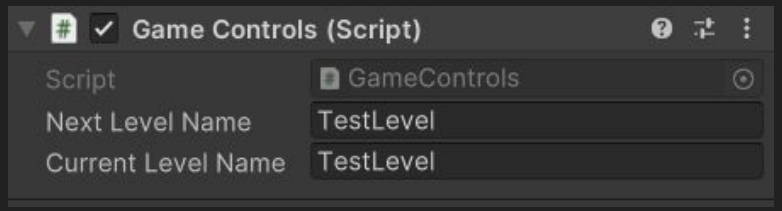



```
Unity Script (1 asset reference) | 0 references
public class ButtonSFX : MonoBehaviour
{
    private AudioSource buttonSFX;

    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.

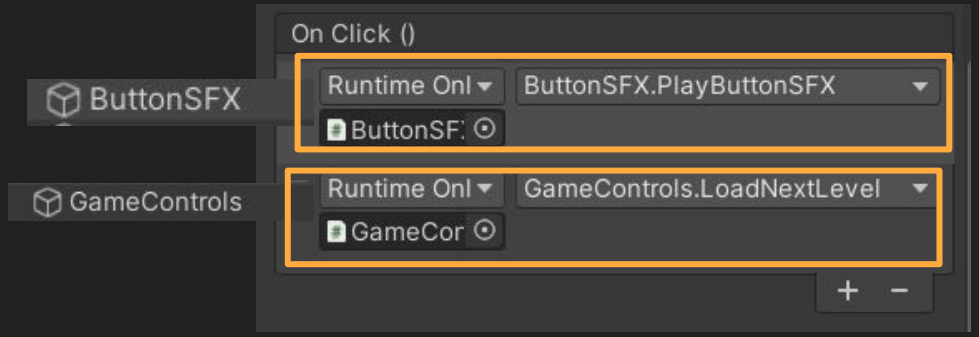



```
public class GameControls : MonoBehaviour
{

    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."

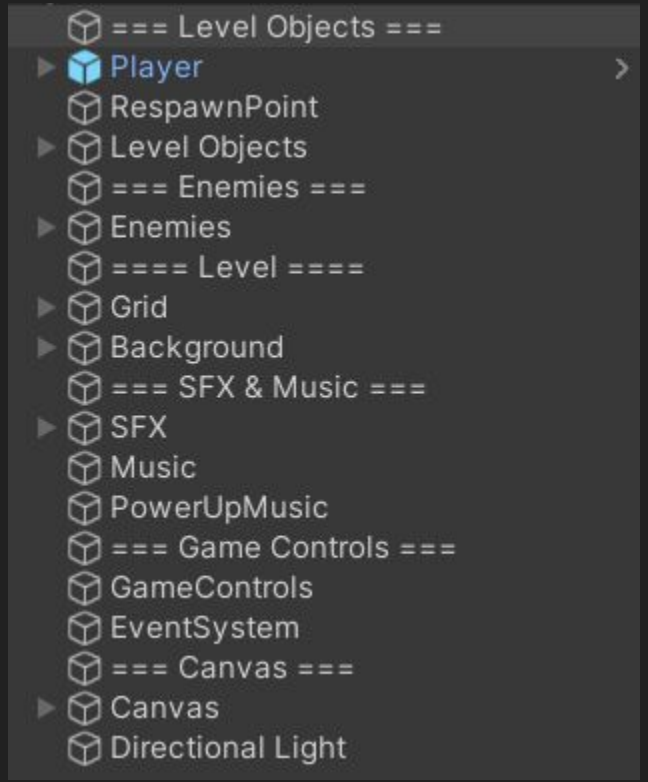


Level

Making the level

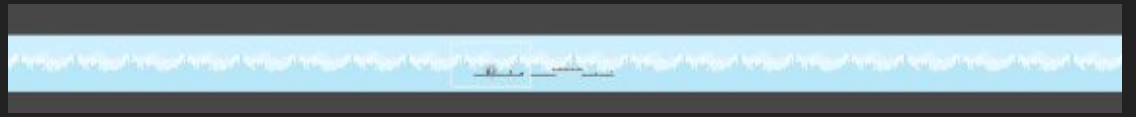
The level is much more complicated but we will break this down into five sections.

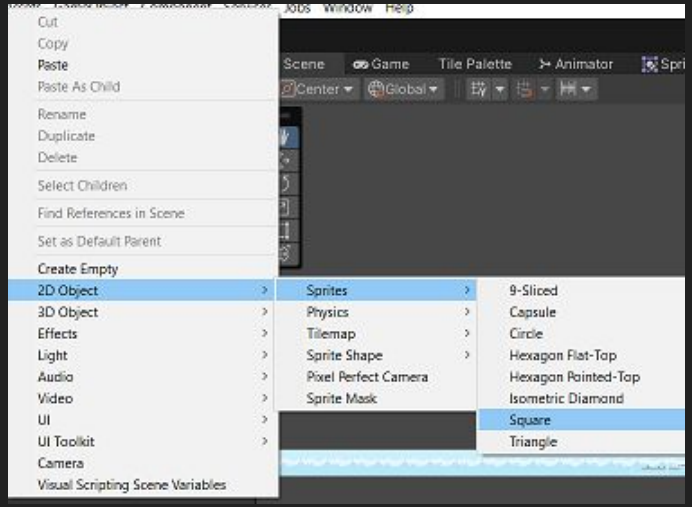
1) Level
2) Player
3) UI
4) Level Objects
5) Music and SFX

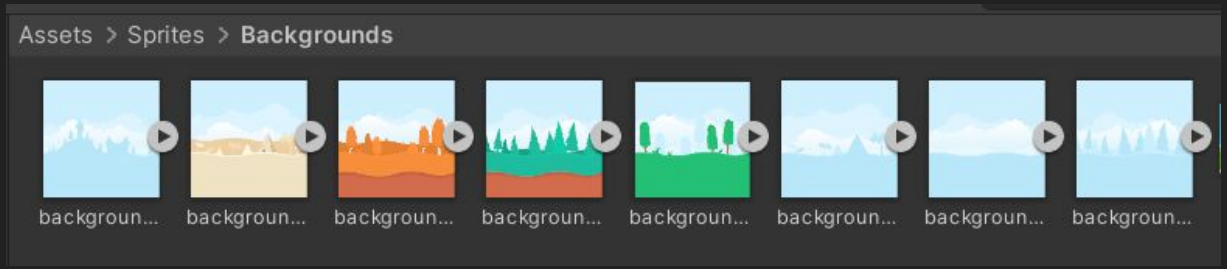


Level Background

We are going to take one of the background images, preferably the faded ones, and tile them across the scene so we have a never-ending background. Start by creating a 2D Sprite Square.

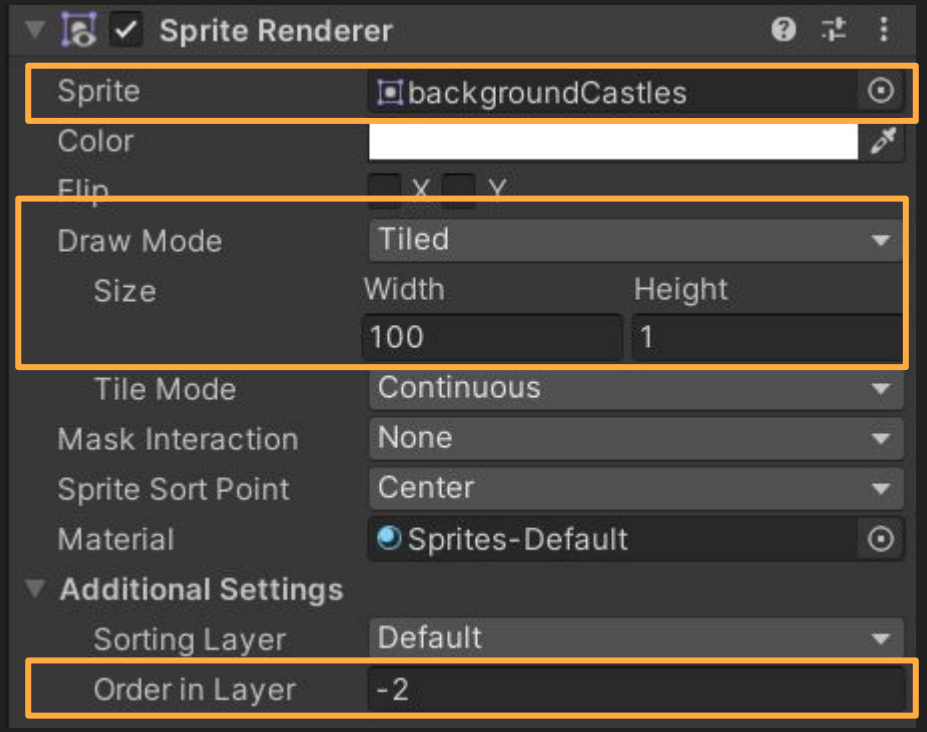






Level Background

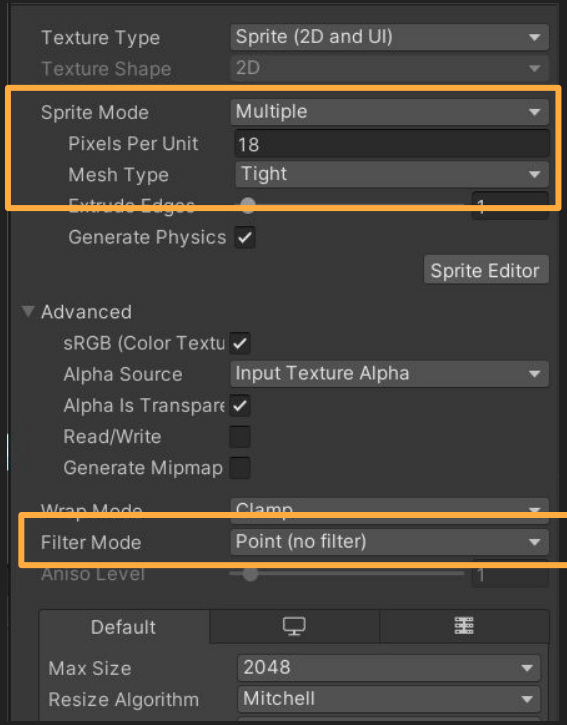
Set the Draw Mode to be tiled, and then make the Width large, like 100. This will tile the image so it's a long level. Also, make sure that the order in layers is low so that it's behind everything else, such as -2.

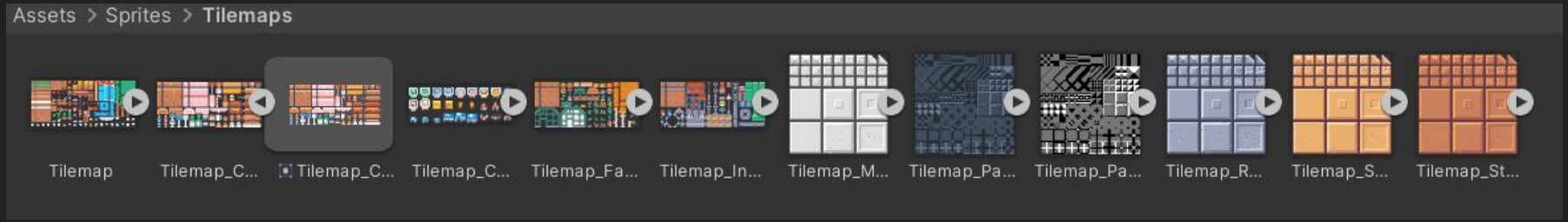


Level Tilemaps

Before we can start making parts of the level, we will have to format the image. In the Tilemaps folder, look for the Tilemap sprite. You will want to set the Sprite Mode to Multiple, the Pixels Per Unit to be 18, and the Filter Mode to be Point.

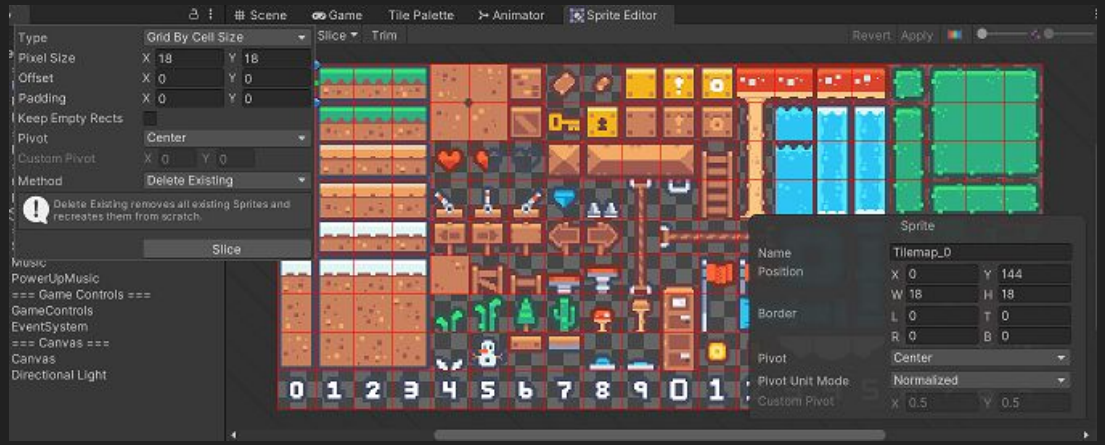
Once all of that is ready, go into the Sprite Editor.





Sprite Editor

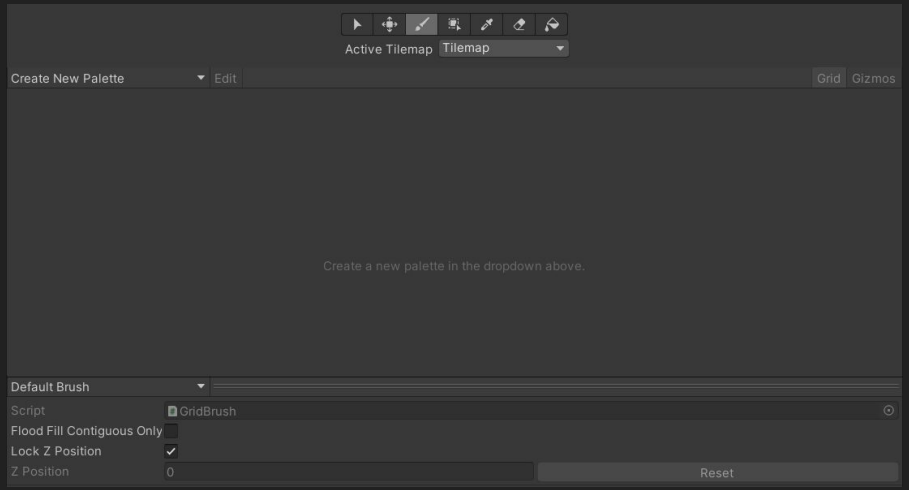
Inside the Sprite Editor, click on "Slice," set it to be Grid By Cell Size, and set the cell size to 18x18. Slice and make sure to apply the changes, and you should have your image cut up and ready to be made into a palette.

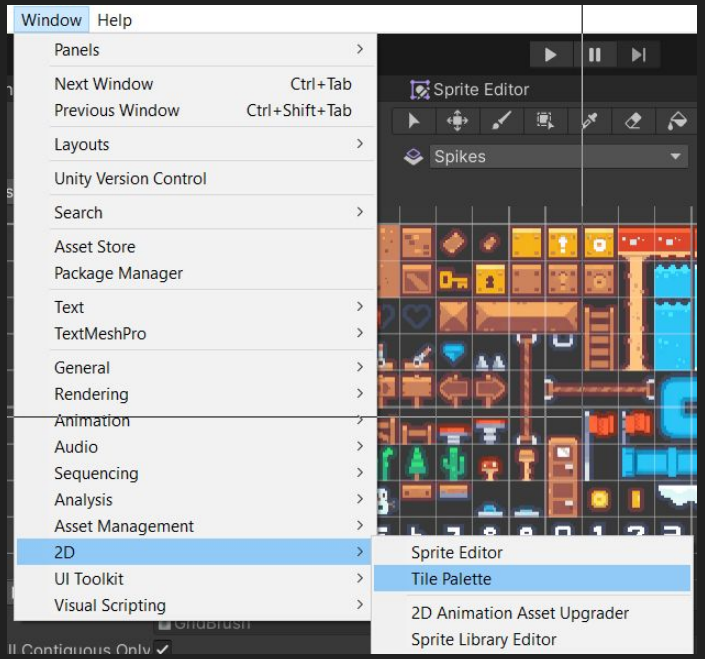


Tile Palette

Go to Windows and open up the Tile Palette.

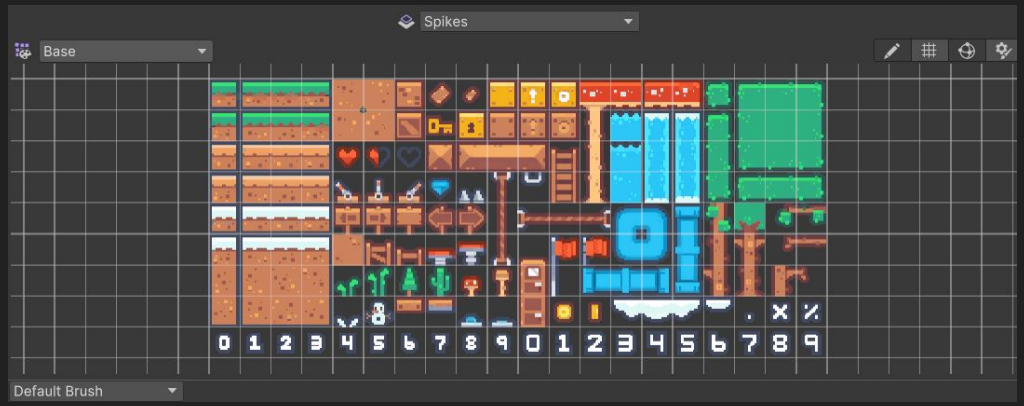
Once you've got it open, create a new Palette and save it to the Palette folder.





Tile Palette

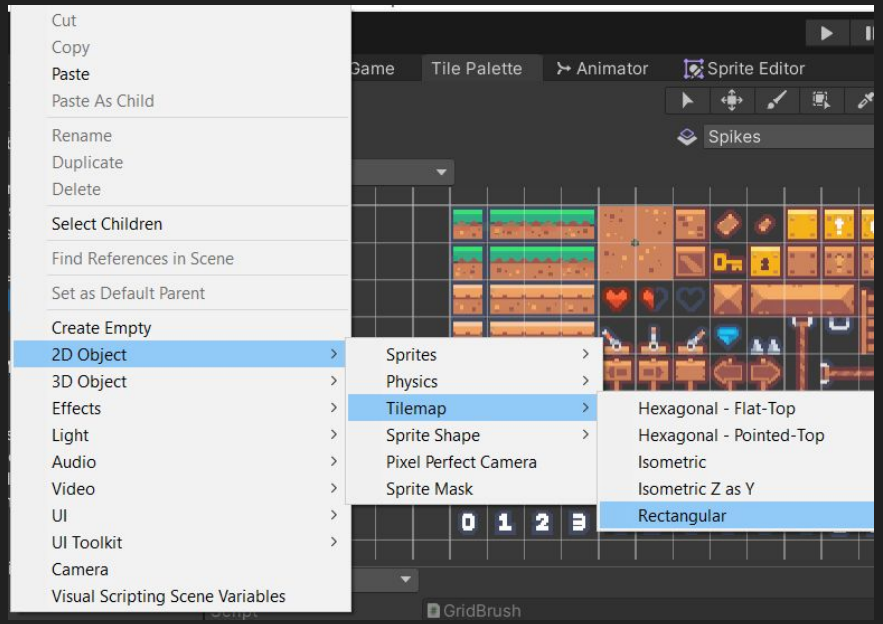
Once you've created the tile palette, drag the tilemap image into the palette, and save the tiles to the Tiles folder.

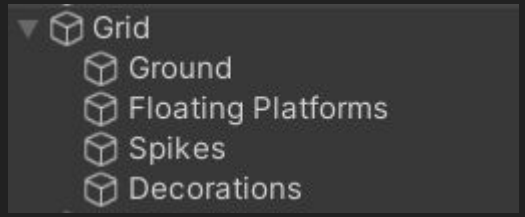


Grids & Tilemaps

Go back to the scene and create four Rectangle Tilemaps.

Call them Ground, Floating Platforms, Spikes, and Decorations. We will go through each one and discuss how they work.

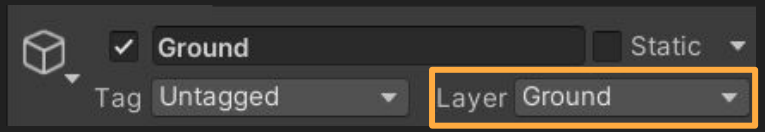


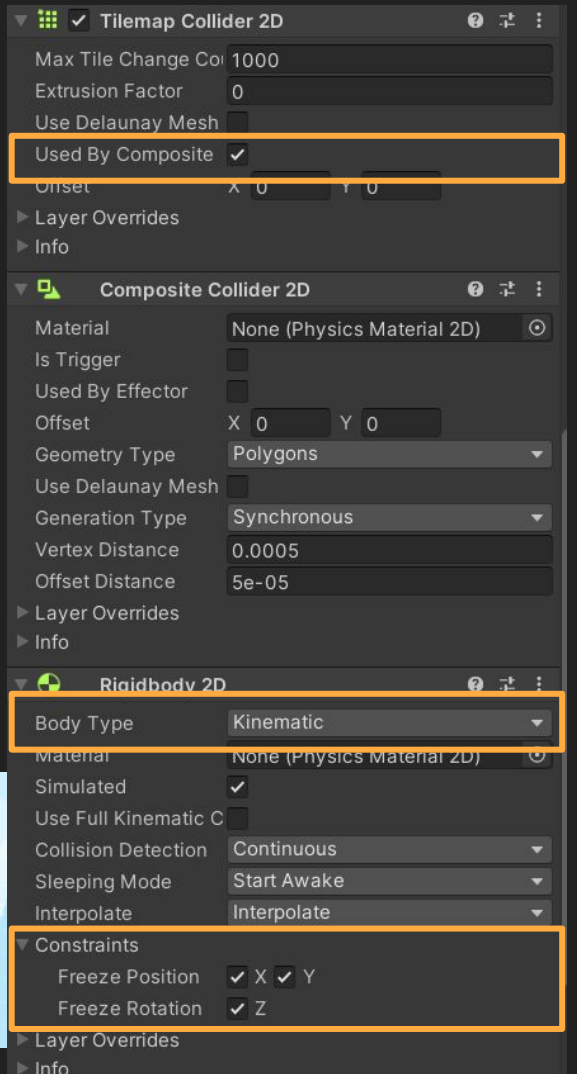


Ground TileMap

The "Ground" will be any surface the player walks on. Therefore, you will need a Tilemap Collider.

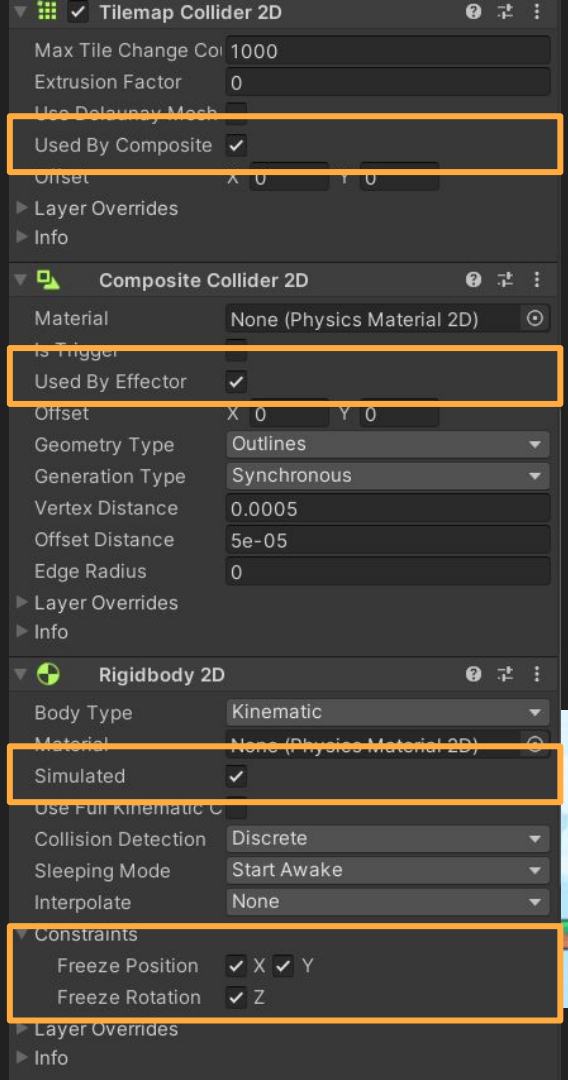
However, the Tilemap Collider creates tiny colliders around each tile, which the player can get stuck on. To address this, add a Composite Collider that combines them into bigger colliders. To make this work, you also need a Rigidbody 2D set to Kinematic Body Type. Lastly, set the Layer to Ground. This will allow the jumping code in the player to work correctly.

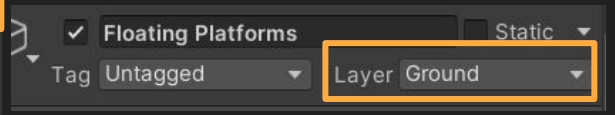


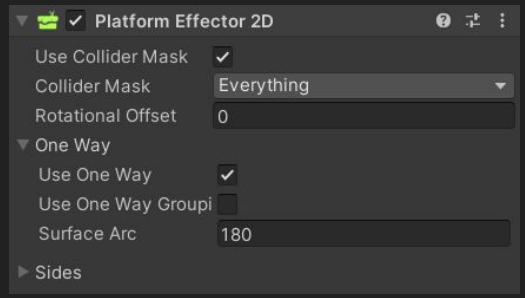


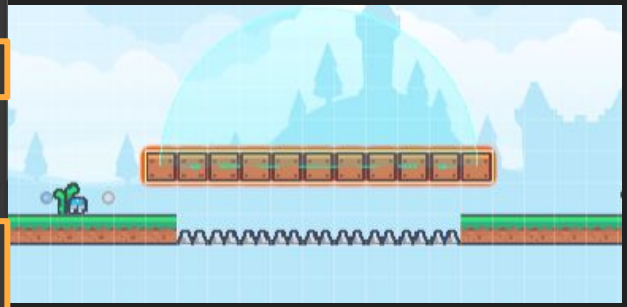
Floating Platform

The "Floating Platform" will have many of the same concepts as the Ground, but with the addition of a Platform Effector. This component will allow the player to jump through the platform from below.





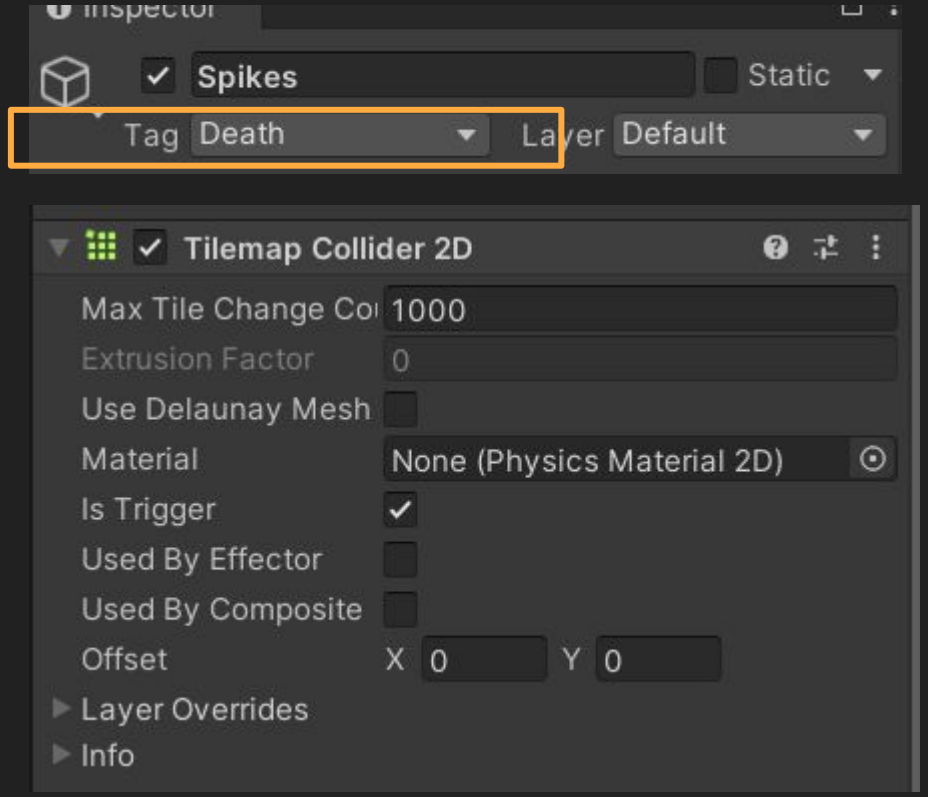


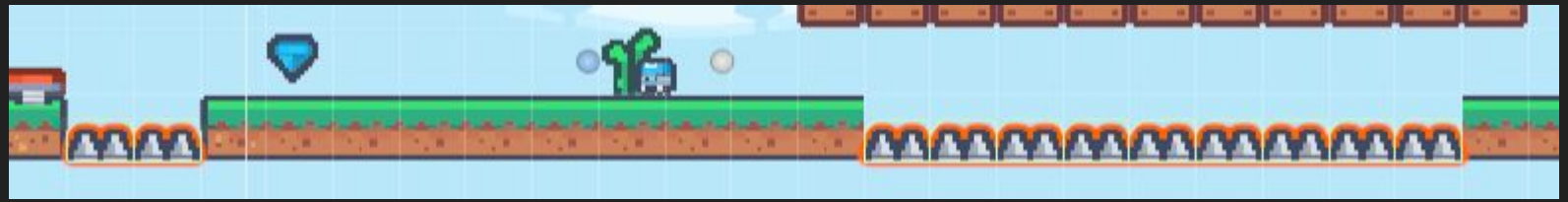


Spikes

For "Spikes," it's much simpler.

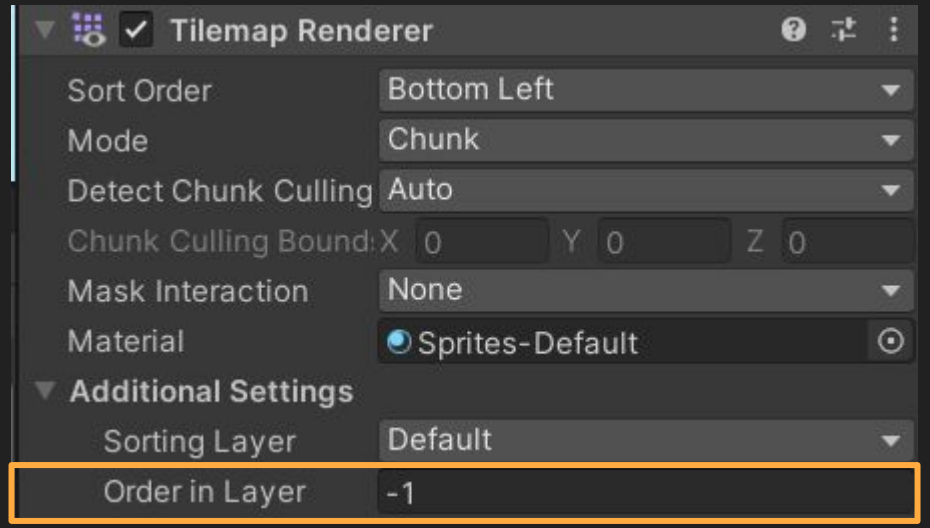
Add a Tilemap Collider and set it to trigger so that if the player touches it, they die. Also, make sure the Tag is set to "Death" for the code to work.

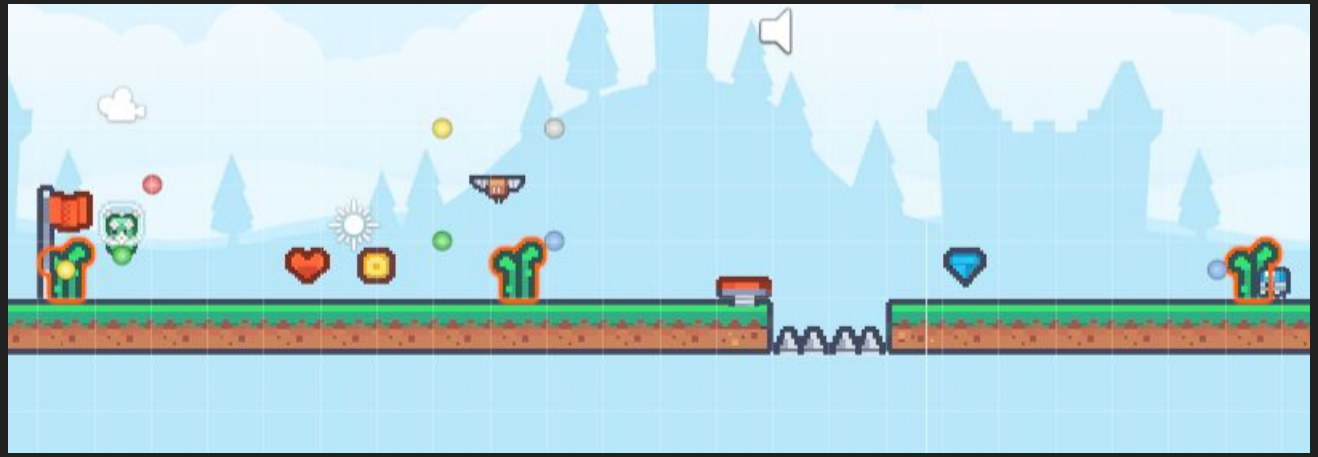




Decorations

The "Decorations" tilemap is the simplest, used to make the level prettier. Make sure the order in the layer is set to -1 in the Tilemap Renderer.





Player

Player

We will now create a player. The player is composed of five game objects:

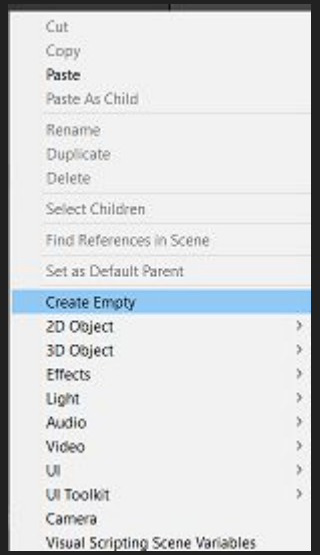
1. Parent
2. Sprite
3. Main Camera
4. Ground Check
5. Particle Effect

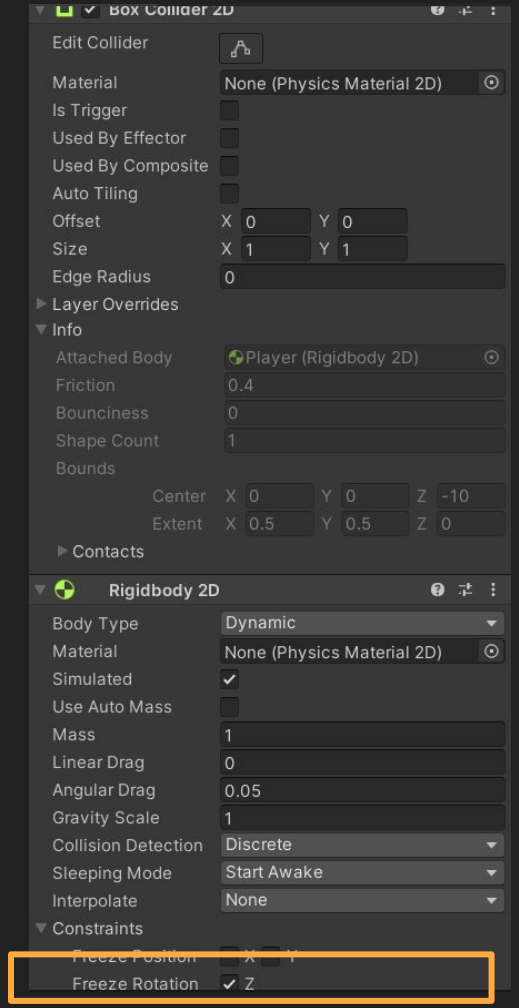
In addition to those, we will create an empty game object called RespawnPoint that the player will be sent back to if they die.

Player Object

Let's start by creating an empty game object and call it "Player." We're going to connect a Box Collider and Rigidbody.

Make sure the Rigidbody has Frozen Z rotation.



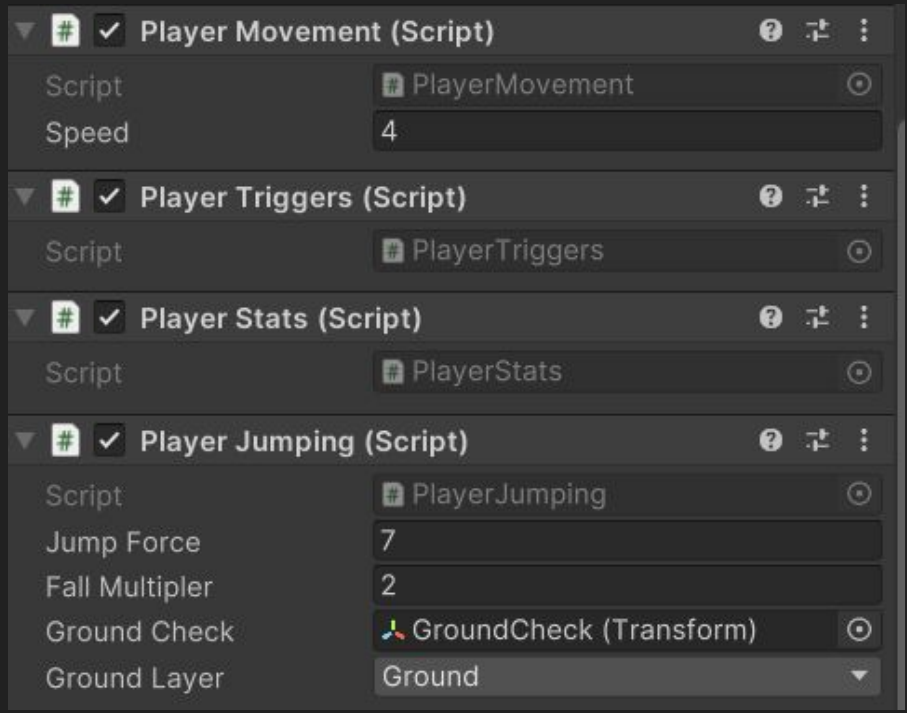


Scripts

We're going to add several scripts to the "Player" game object.

We're going to add "Player Movement," "Player Trigger," "Player Stats," and "Player Jumping."

We'll fill out these scripts as we continue making the level.

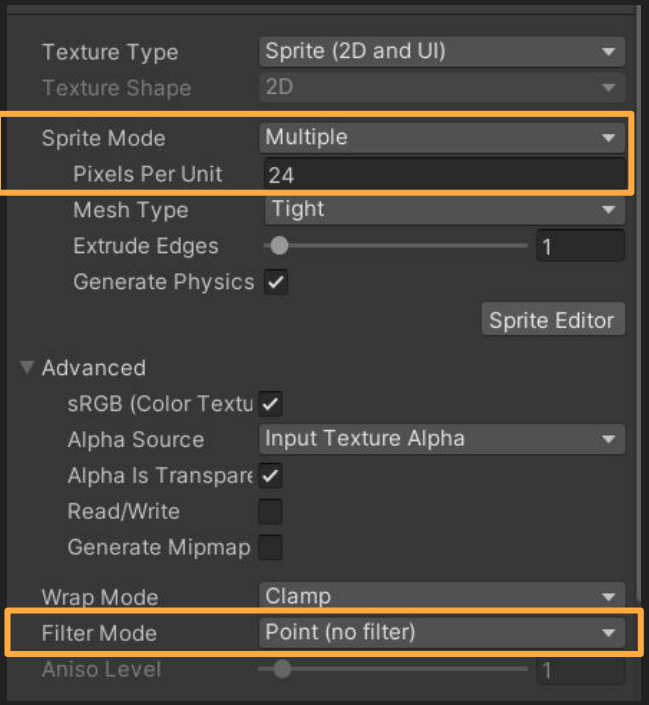


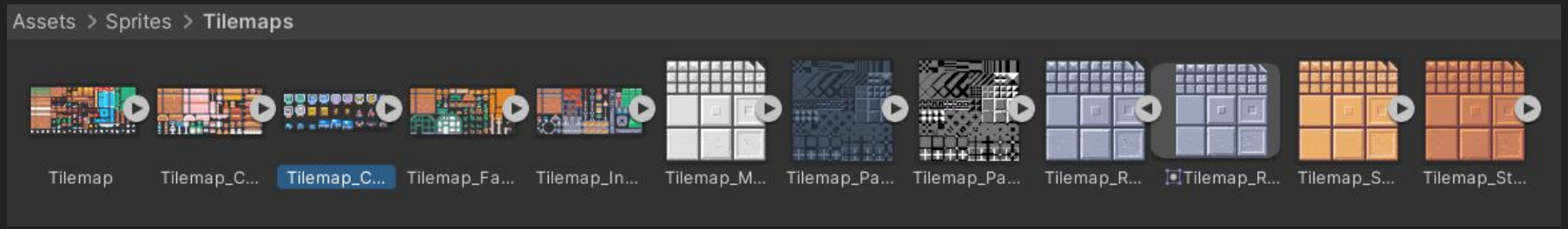
Tilemap

Before we can give the player a sprite, we have to break down the spritesheet.

Like before, go to the Tilemap folder, look at Tilemap_Character, and set Sprite Mode to Multiple, Pixels Per Unit to 24, and Filter Mode to Point.

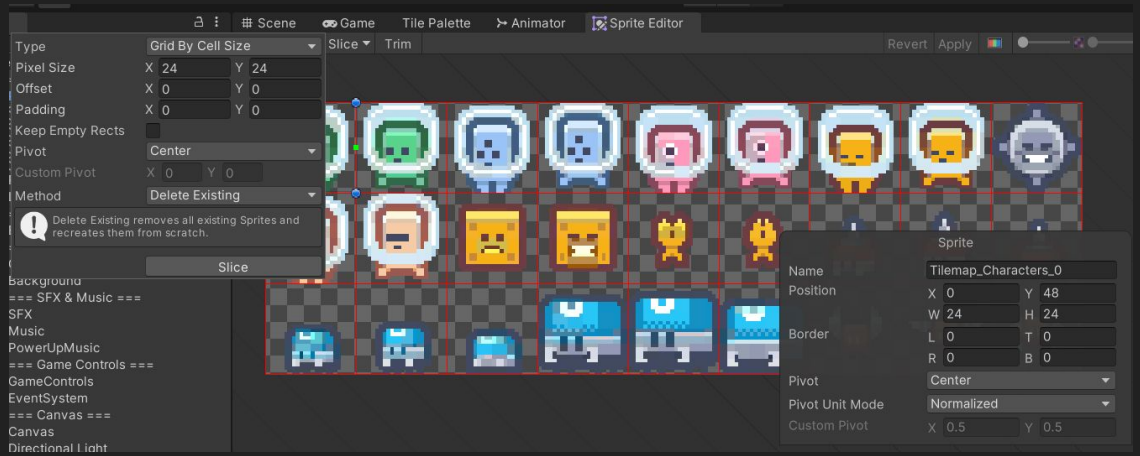
After it's all set up, open up Sprite Editor.





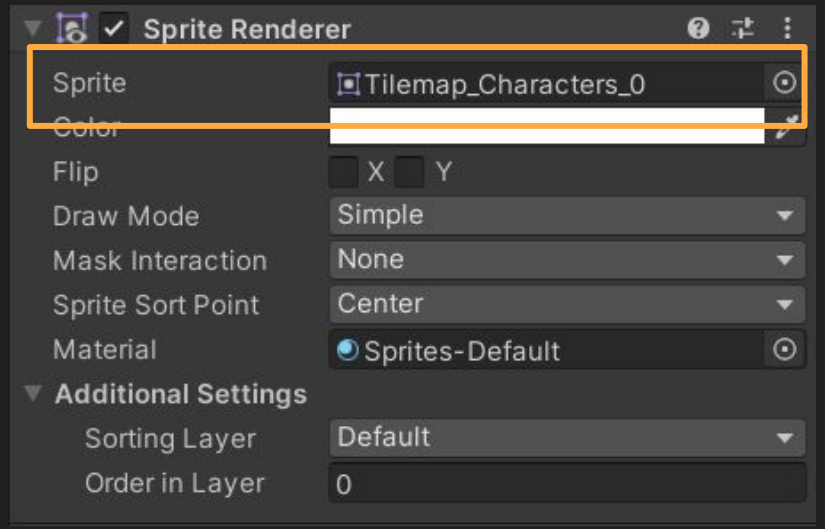
Sprite Editor

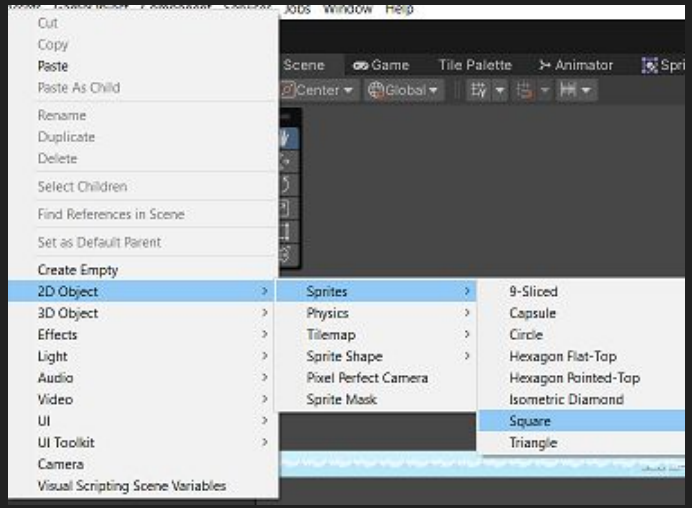
We’re going to slice it, go to Grid By Cell Size, with the size being 24x24, slice it, and remember to apply the changes.



Sprite

Next, create a child that has a Sprite Renderer component. Set the Sprite in the Sprite Renderer to be the green guy.



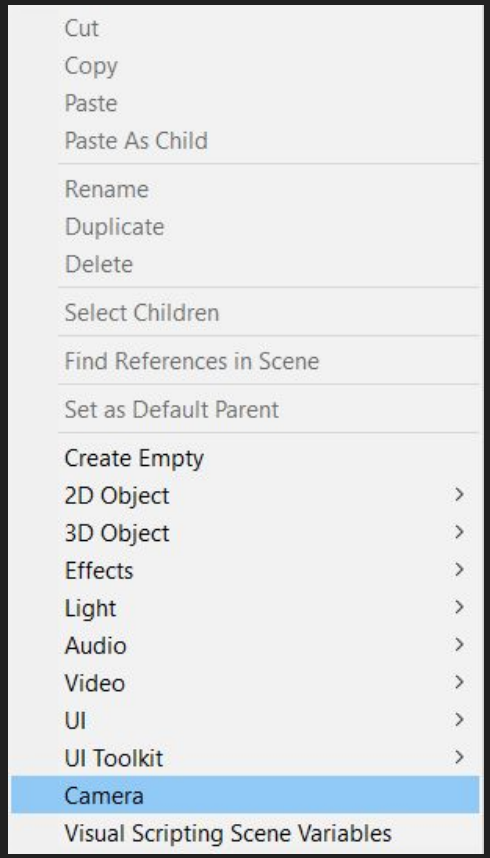


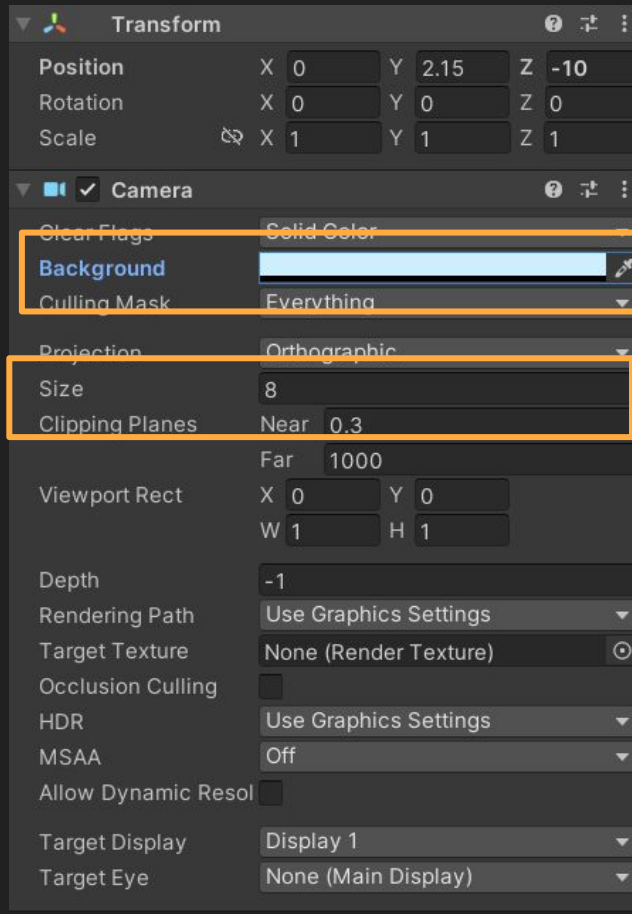
Camera

Add a camera to the Player by creating a camera object. Inside the camera, you want to set the Y position a little above so the player is a bit lower on the screen.

Ensure that the Size is large so the player has a good view of the level we're creating.

Also, make sure the background color is the same as the background image, so if the player jumps past the background top, it looks like the same color.

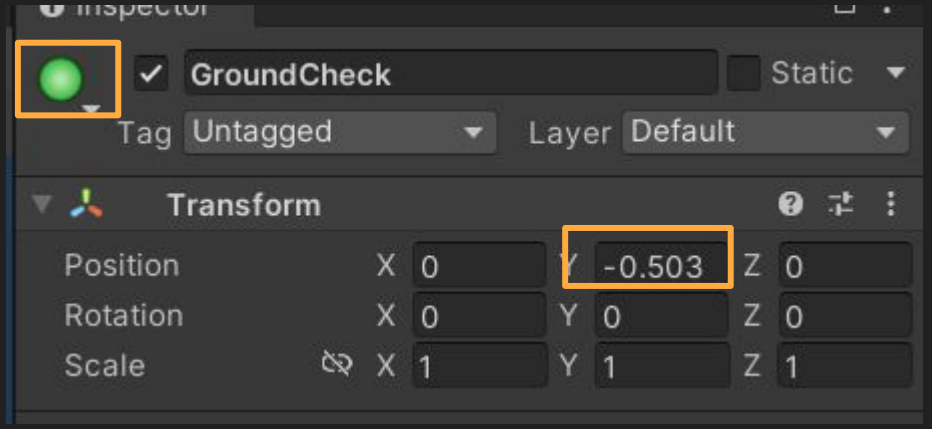


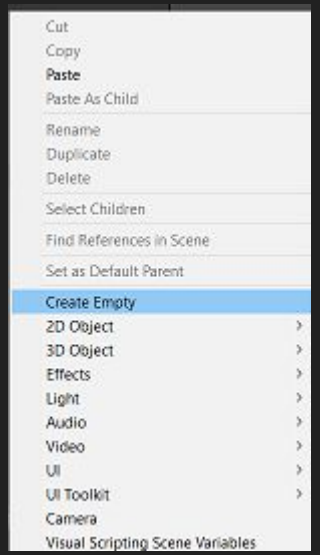


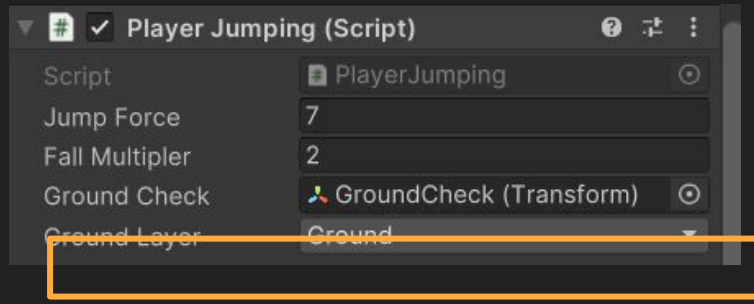
Ground Check

Create an object called "GroundCheck" and add an icon to it so you can see where it is.

Move it to the feet of the player, and this will act as a jumping collision box created through code. Make sure you connect this object to the script in the player and set the ground layer to "Ground."

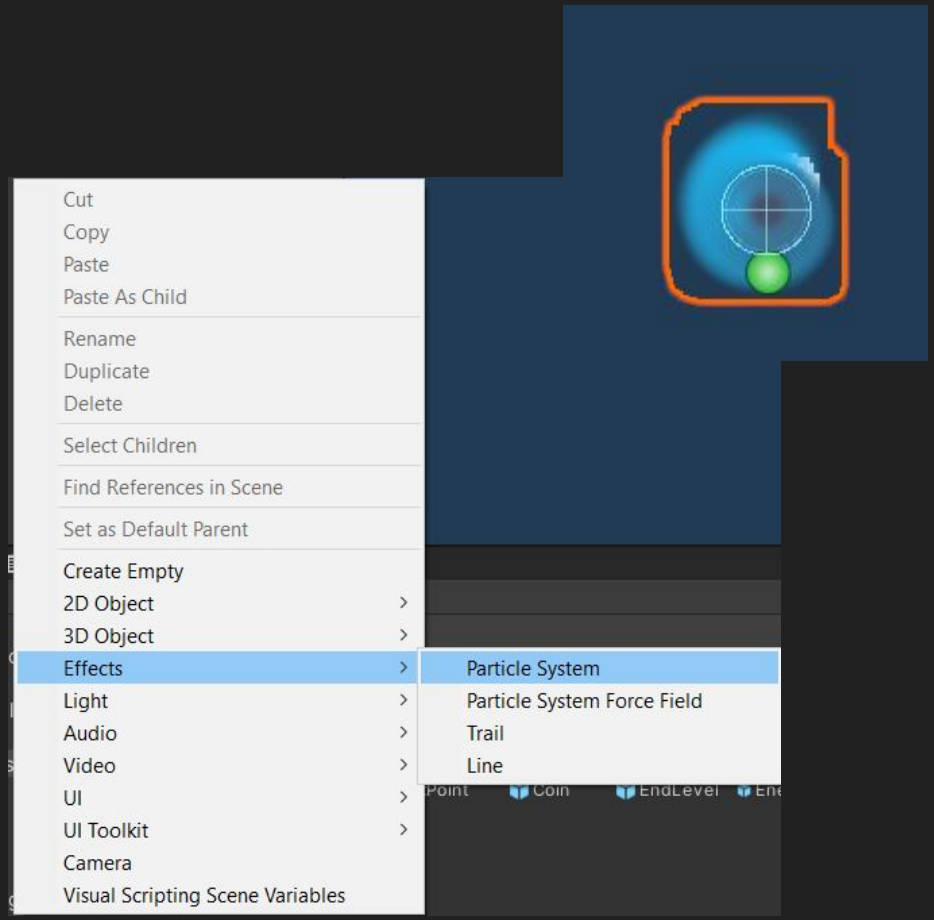






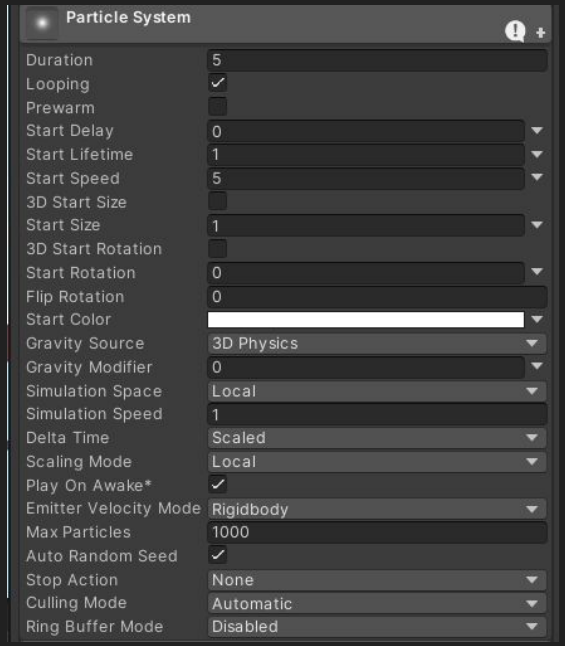
Particle Effect

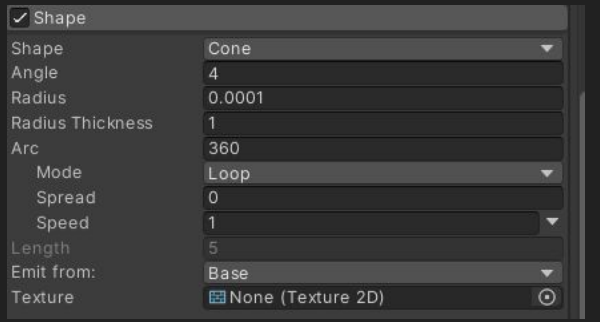
Next, create a Particle System. We'll be using this to show that the player has picked up a power-up and is invincible to enemies.

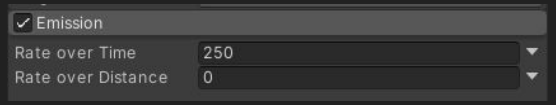


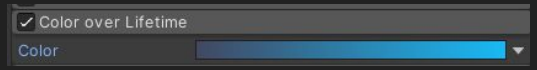
Particle Effect

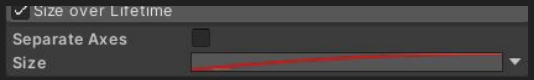
For the particle follow these values to make the particle system make the spinning effect.





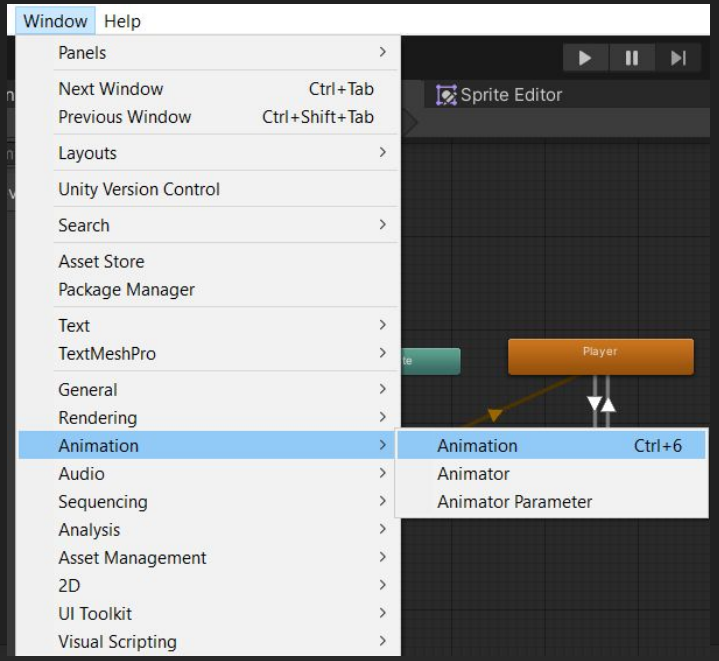


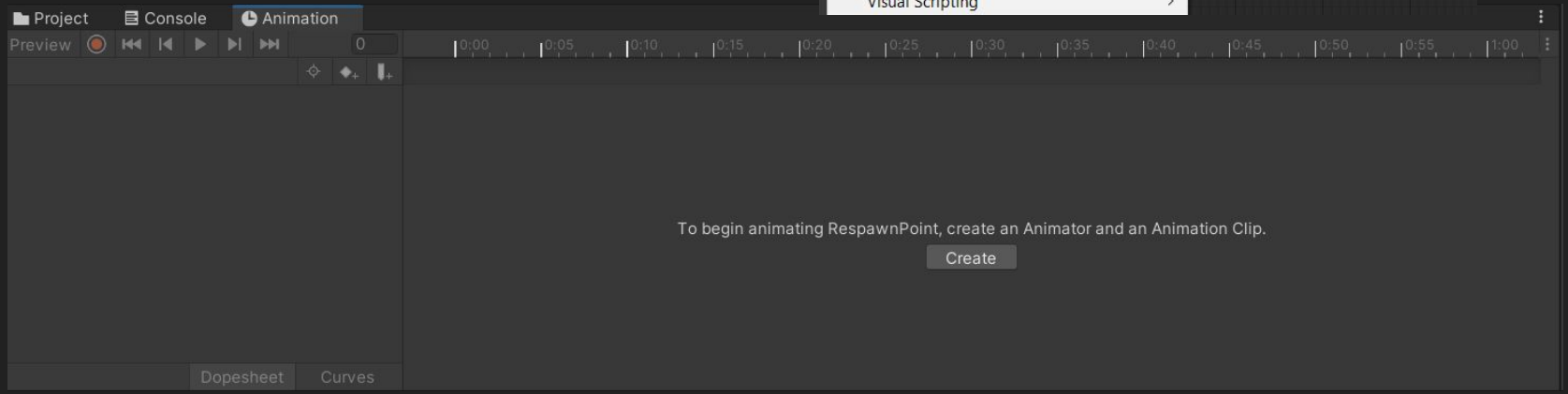




Animation

Open up the animation and select the Player Game Object. The animation window will say there is no animator connected to it, so we're going to create a new one.

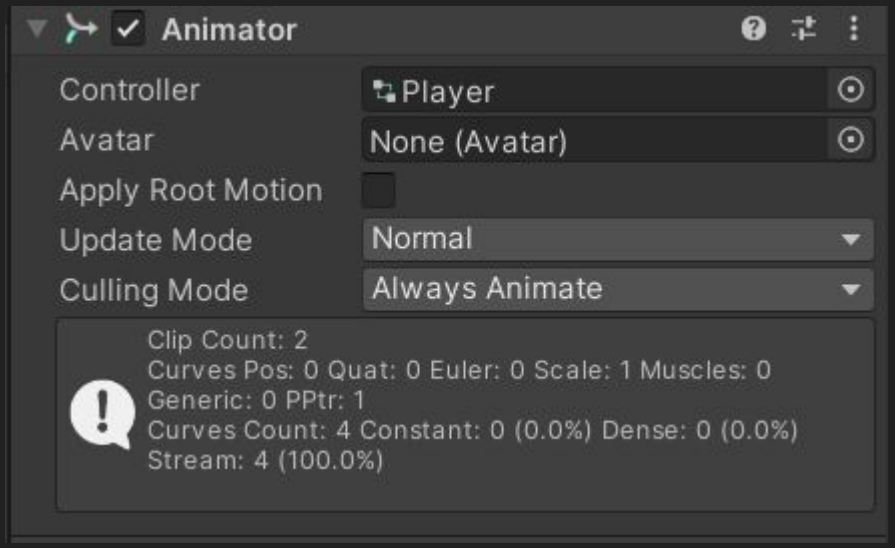


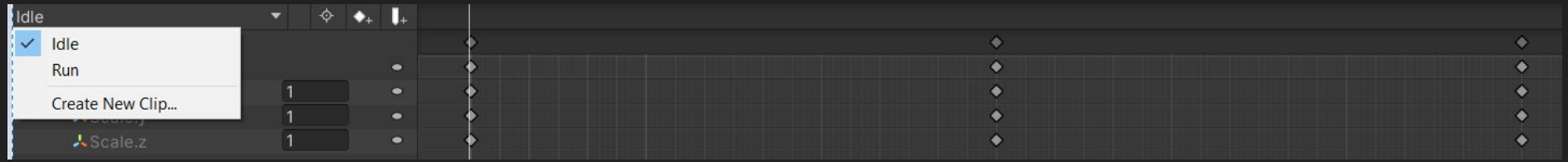


Animations

We’re going to give our player two animation clips. First, "Idle," where the body grows and shrinks as they stand still.

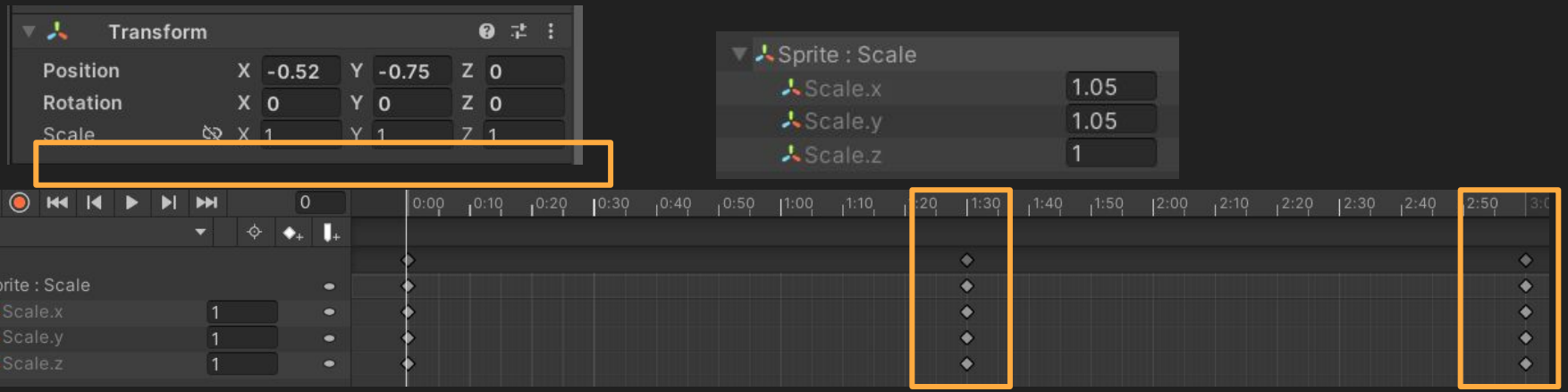
Second, "Run," where the sprite is switched to appear that the player is moving.





Idle

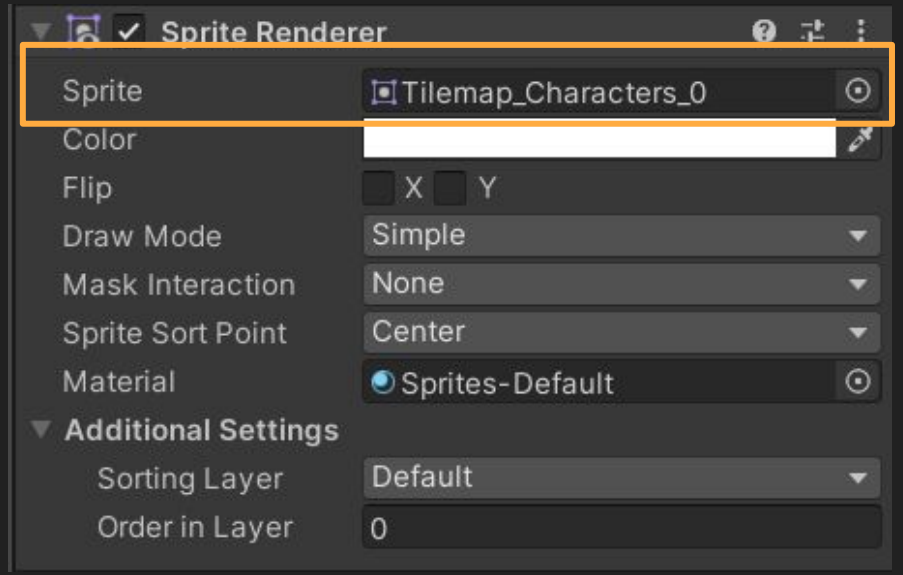
For the idle animation, we're going to have it run from 0 to 3 seconds. Click the red button to start recording the animation. Move to 1:30 on the time scale and change the x and y scales to be 1.05, then move to 3 seconds on the time scale and change the scale back to 1,1,1. This will create a basic animation that makes it look like the character is breathing.

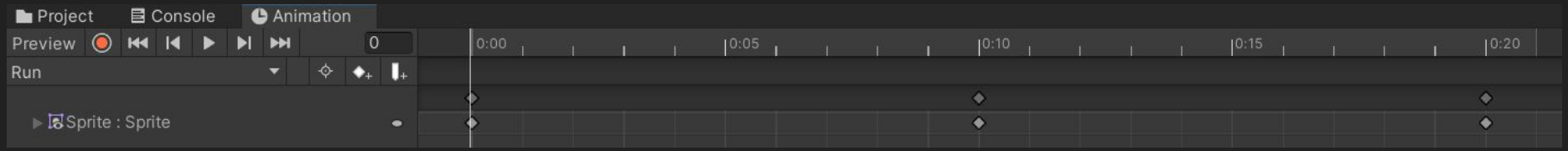


Run

To create a new animation, click on "Idle" and select new animation. Call it "Run," and it's going to last 20 seconds. Click the red record button, and go to the Sprite child game object.

Change at 10 seconds, change the sprite from Tilemap_Character_0 to Tilemap_Character_1, and at 20 seconds, change it back.

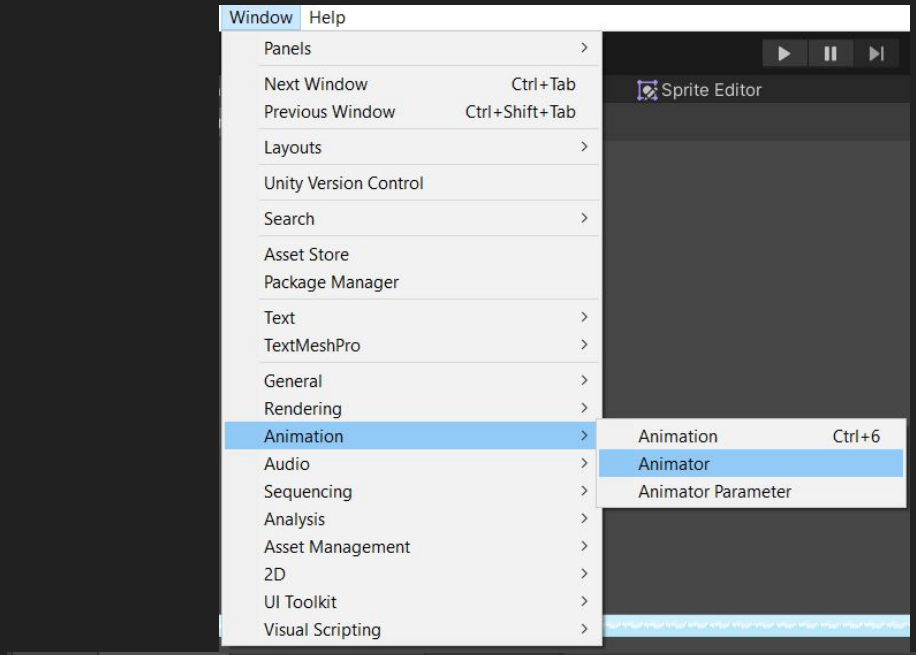


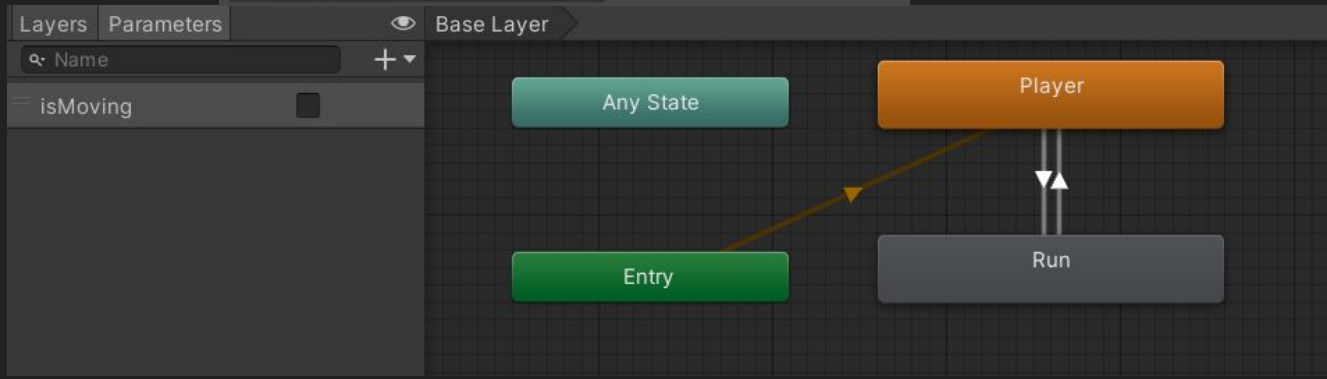


Animator

Open up the animator window. Here we're going to connect the two animation clips.

You will see that one of them is orange; that's the default animation that will play when the game is started.



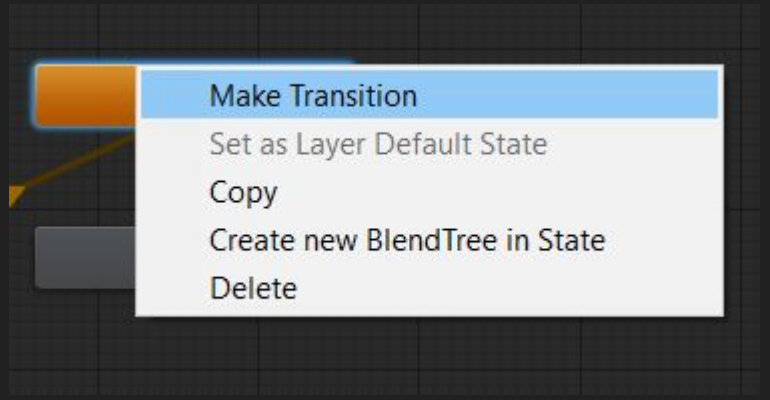


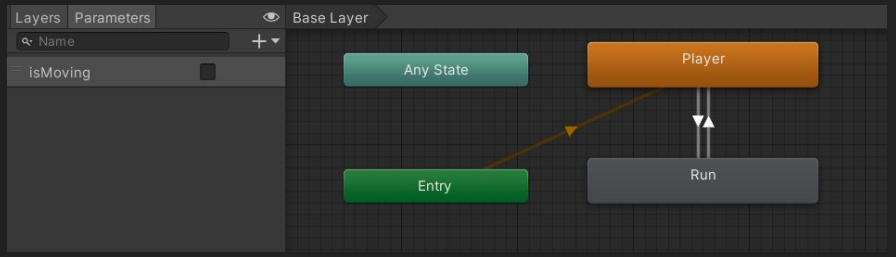
Animator

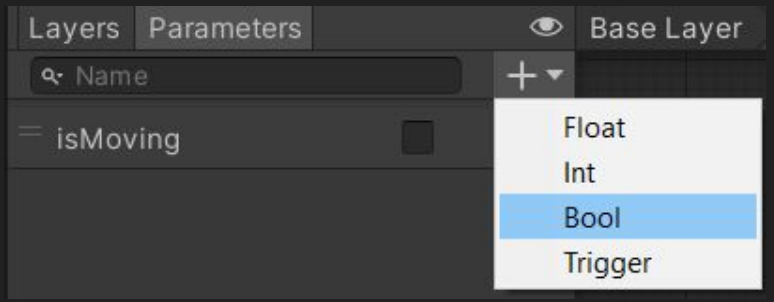
Right-click on one of the clips and create a transition. When you click on "Make Transition," you will get an arrow pointing at the other clip.

Once you've done that to one, do it to the other one as well, so you have an arrow moving up and down.

Then click on the "Parameter" Tab, click the plus and create an "isMoving" Bool variable. That's what we use in the code to change between the animations.





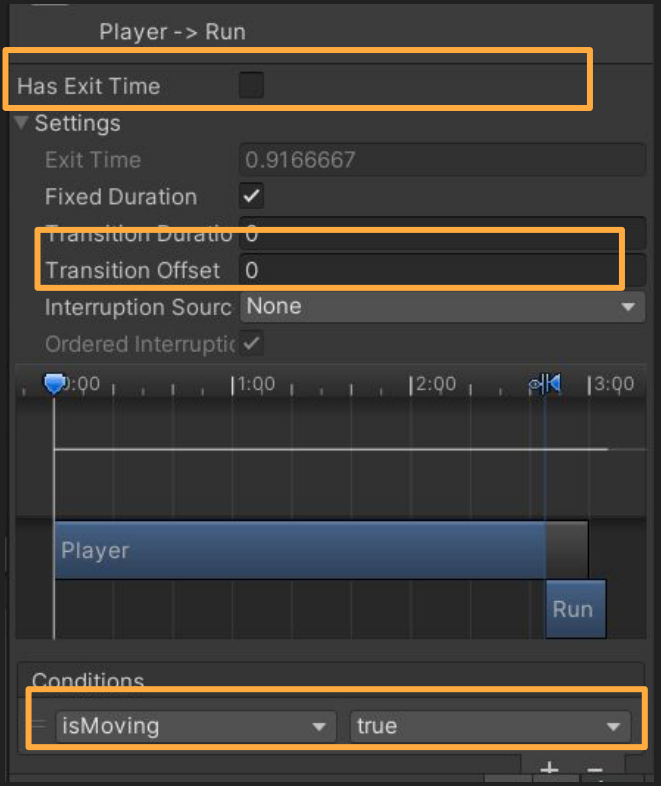


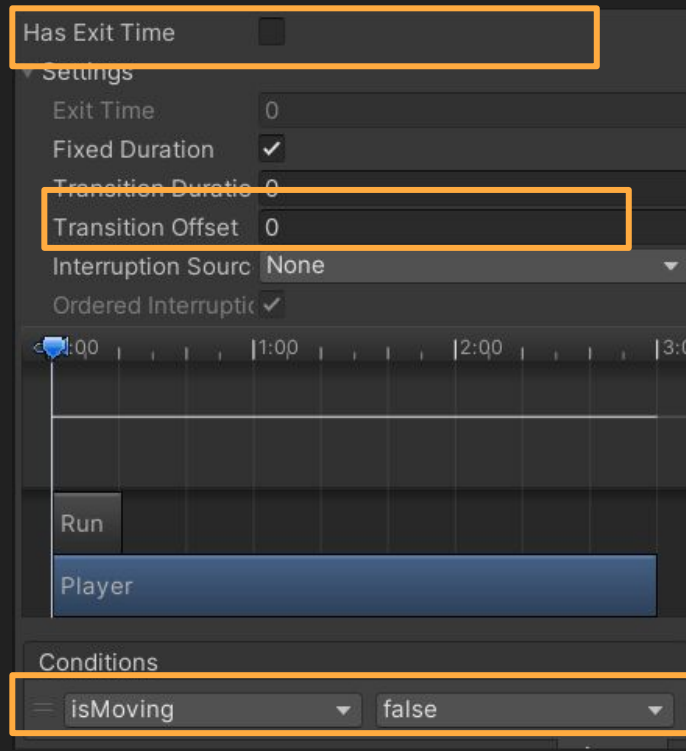
Transitions

Lastly, we will edit the transitions. By clicking on the arrows, you will see details in the inspector.

First, you want to turn off the Exit time and change Transition duration to 0.

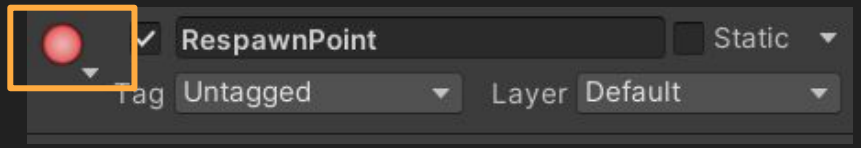
This will make it so you animate immediately when the player clicks. Second, you want to add a condition "isMoving." It should be true from idle to run and false from run to idle.

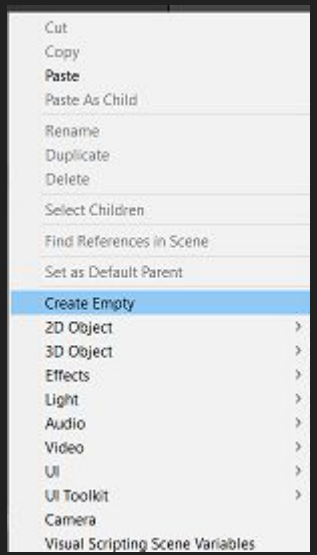




Respawn Point

Create an empty Game Object, call it "RespawnPoint," and set the icon to be a red circle. This will allow the code to respawn the player at that position if they touch something bad.





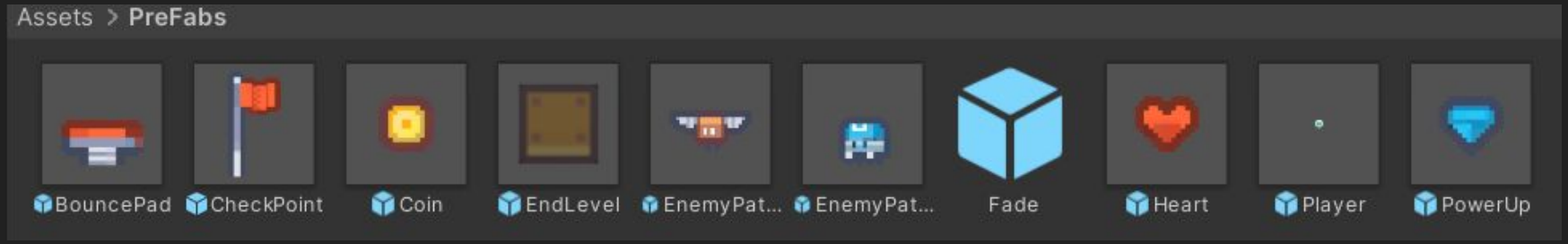
Level Objects

Level Objects

There are several objects that we will create to make the levels more interesting:

1. Coins
2. Heart Pickups
3. Power-Up Items
4. Checkpoints
5. Bounce Pads
6. End Level Goals

In addition to that, you will be provided with two enemies to populate your game.

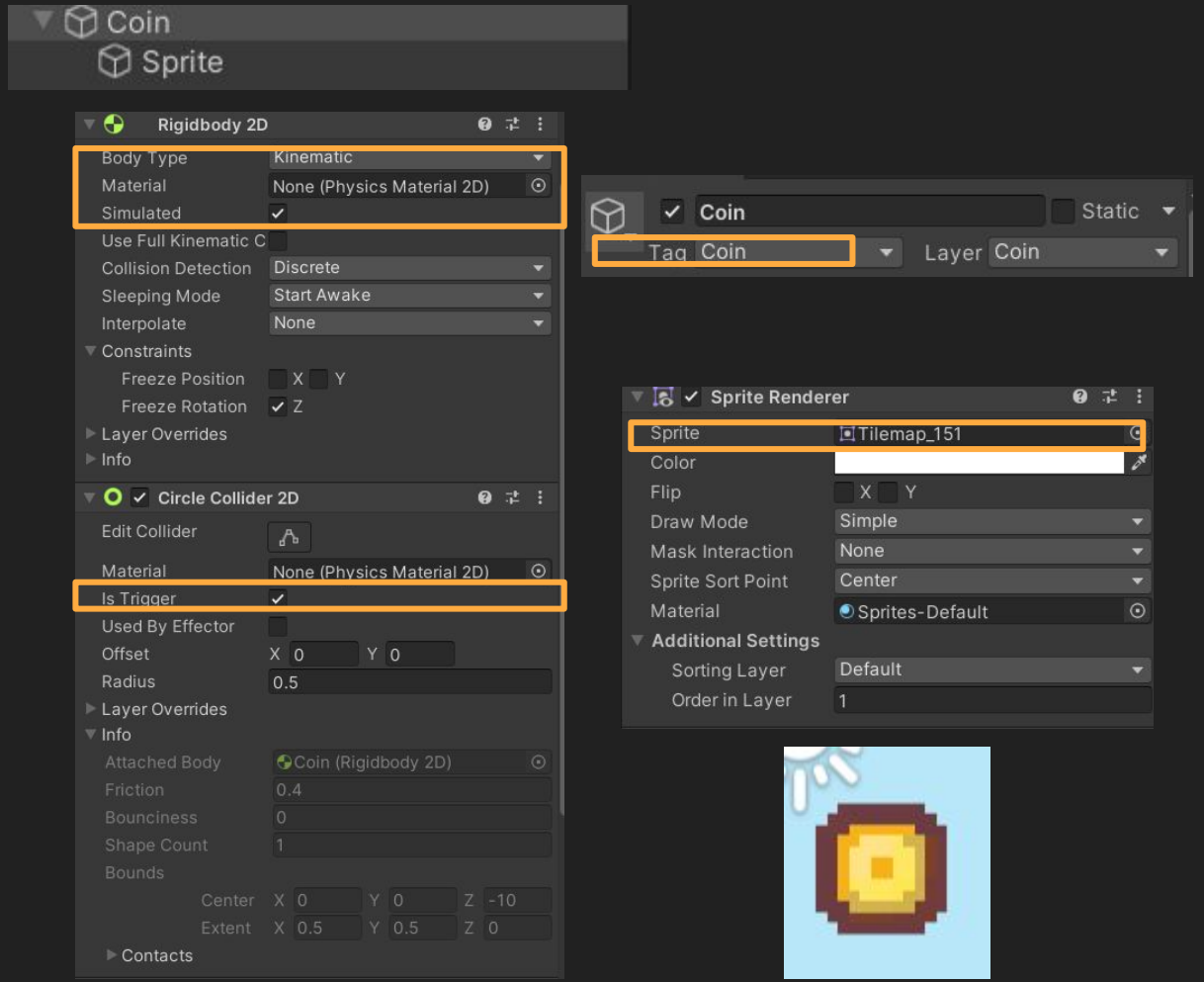


Coin

Create an empty game object, and then create a sprite object that's a child of that.

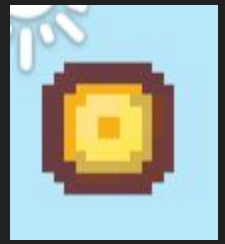
The parent should have a Circle Collider that's a trigger, a Rigidbody whose mode is set to Kinematic, and Z rotation is frozen. Set the tag to be "Coin" so that the code will react to it.

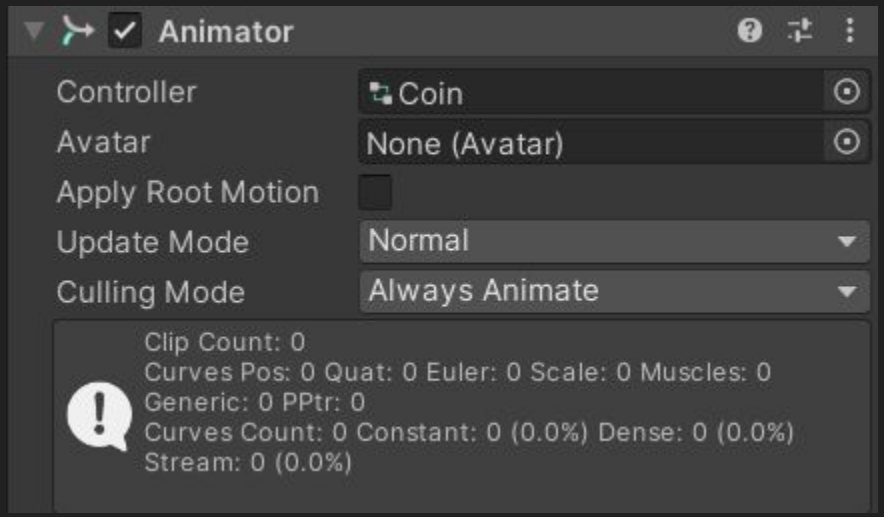
And set the Sprite in the Sprite Renderer to be that of the coin.

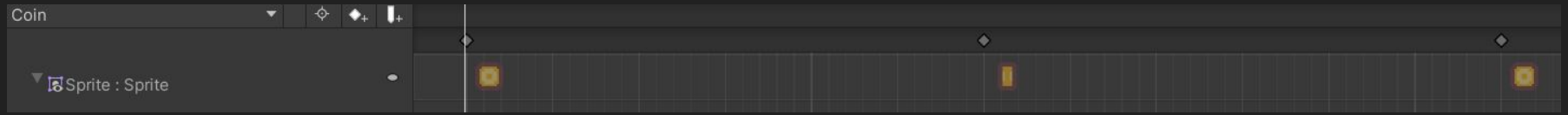


Coin Animation

We will create a simple animation for the coin where it switches between half rotation and back to front facing.





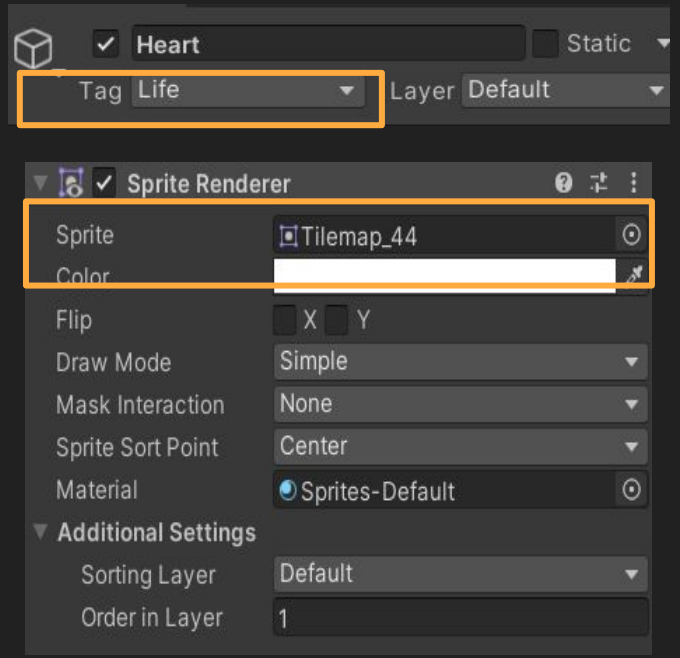


Heart

To create the heart, copy the Coin game object, and we'll change a few things.

Change the name to "Heart," the tag to "Life," and the Sprite in the Sprite Renderer to the heart tile.

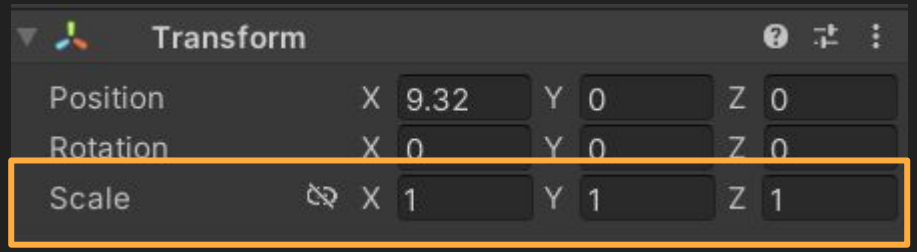
All the other components should carry over from the coin, and you don't have to set them up again.

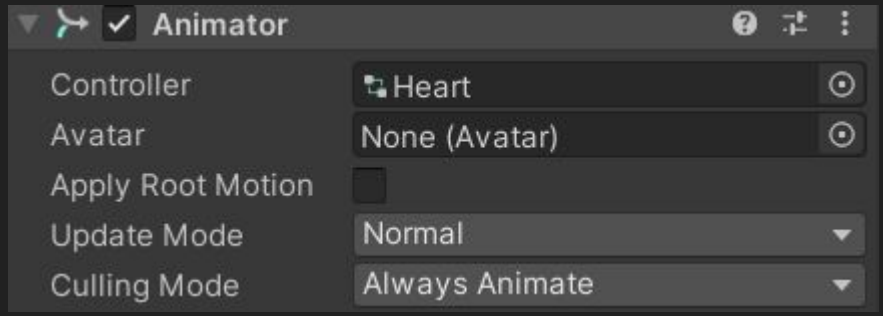


Heart Animation

We're going to go to the animator and remove the Coin controller and create a new Heart controller.

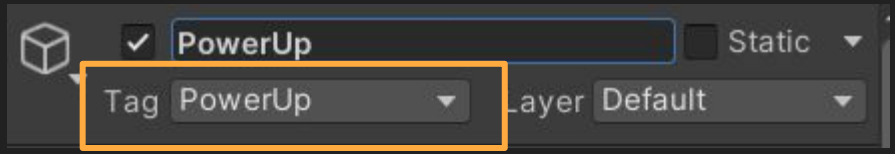
There you will change the scale from 0.9 at the x and y scale to 1, and then back to 0.9. This will make the heart pulsate and make the player want to grab it.

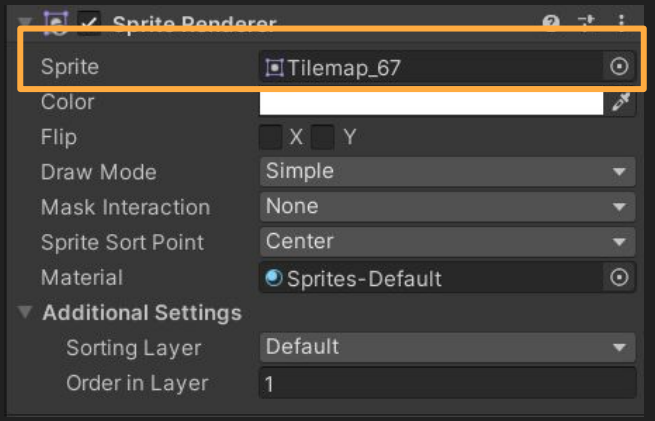




Power Up

For the Power Up, we'll do the same thing. Copy the Heart, and change the Name to "Power Up," the tag to "Power up," and the sprite to the diamond sprite.



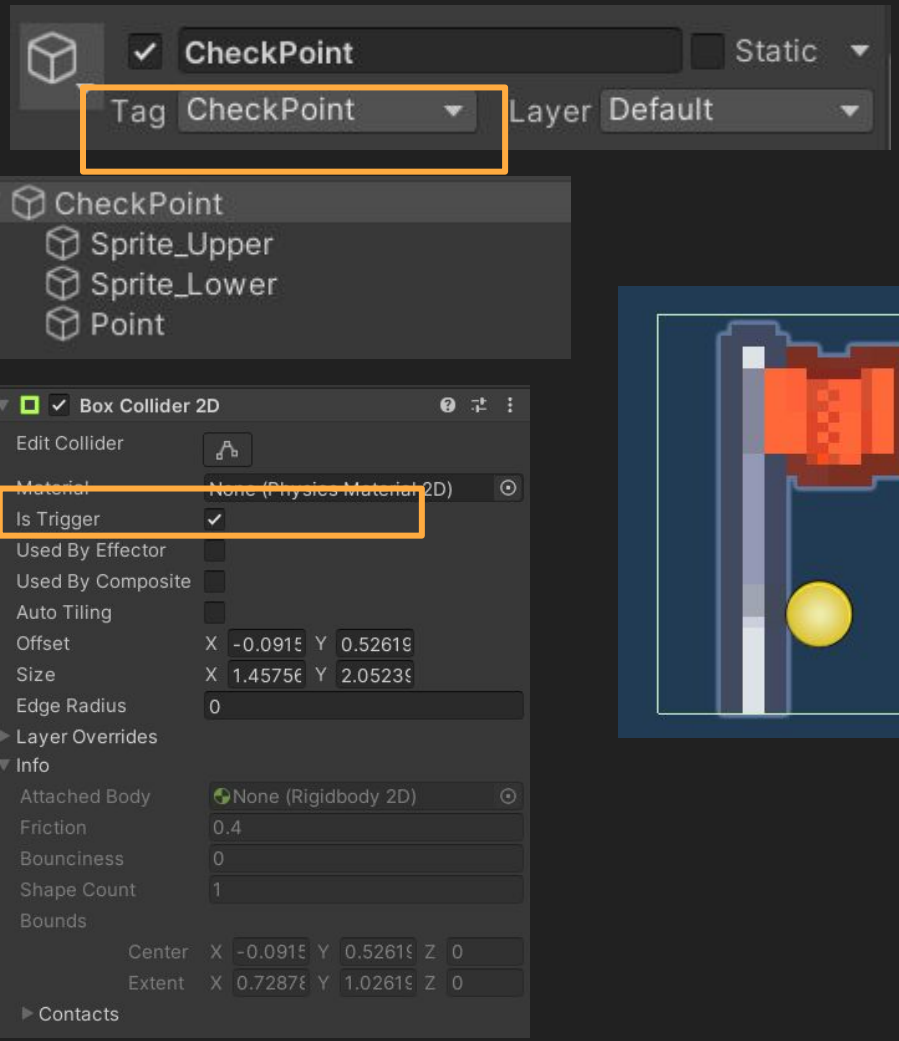


Checkpoint

The checkpoint will be made up of a few more objects than normal.

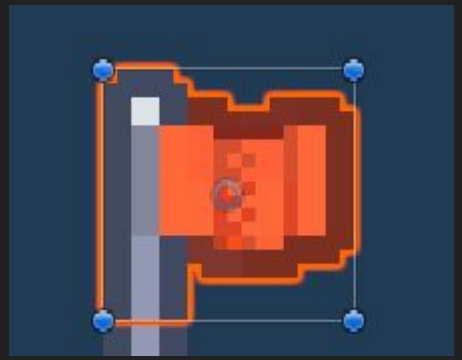
The parent will have a collider that will act as a trigger. Then it will have two sprites, an upper one with the flagpole and a lower one with the lower part of the flag.

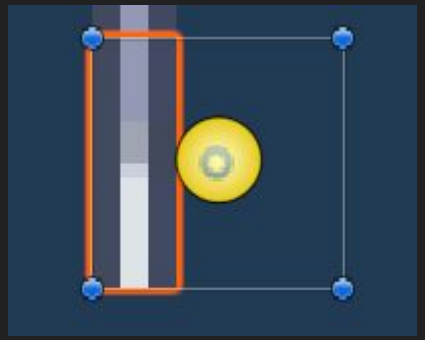
Lastly, it will have a point that will be used by the respawn.

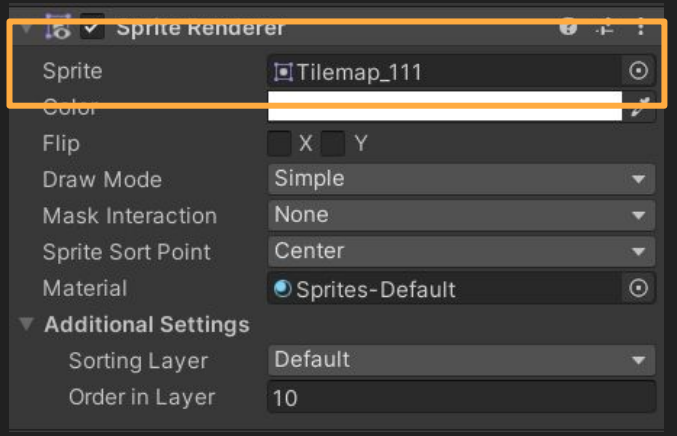


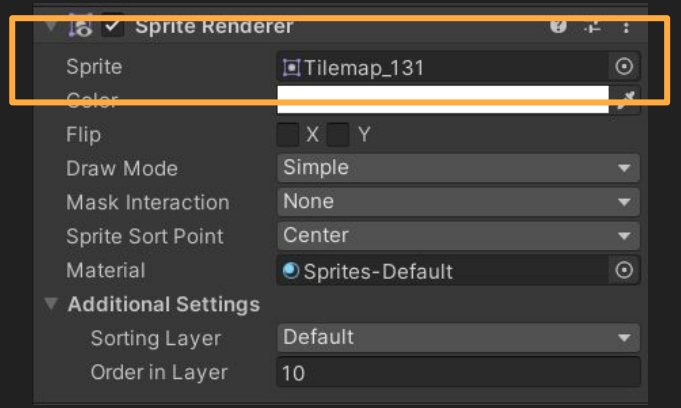
Sprites

You will have two children objects, both with Sprite Renderers, and each one showing half of the flagpole.







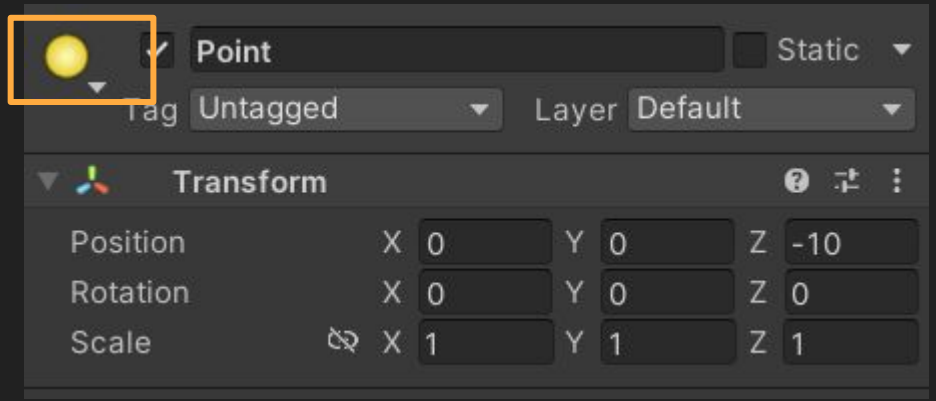


Point

Lastly, create a point, give it a yellow circle icon.

This will serve as a respawn point update; when the player triggers the checkpoint, this will be where the player respawns upon death.

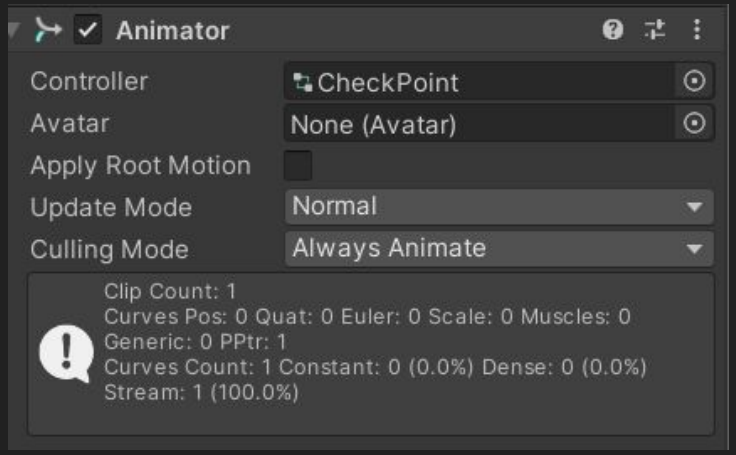
Make sure to place the point near the bottom of the flagpole.

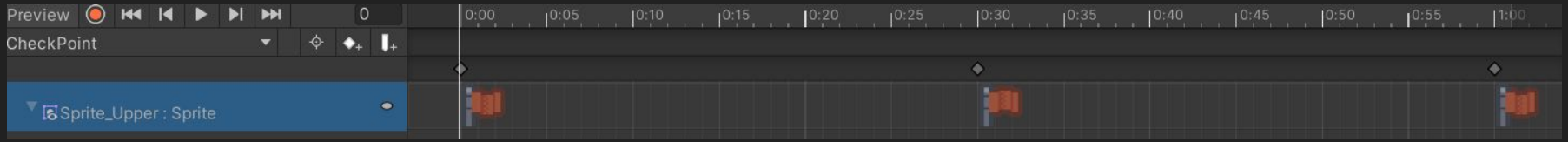


Checkpoint Animation

We will create an animation for the flagpole.

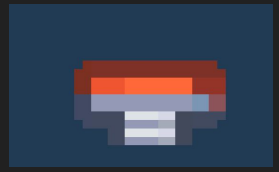
It will consist of changing the upper sprite between two states of the flag so it looks like it's waving in the wind.

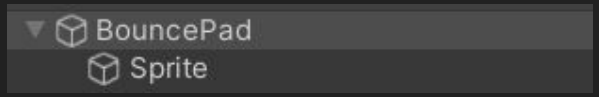


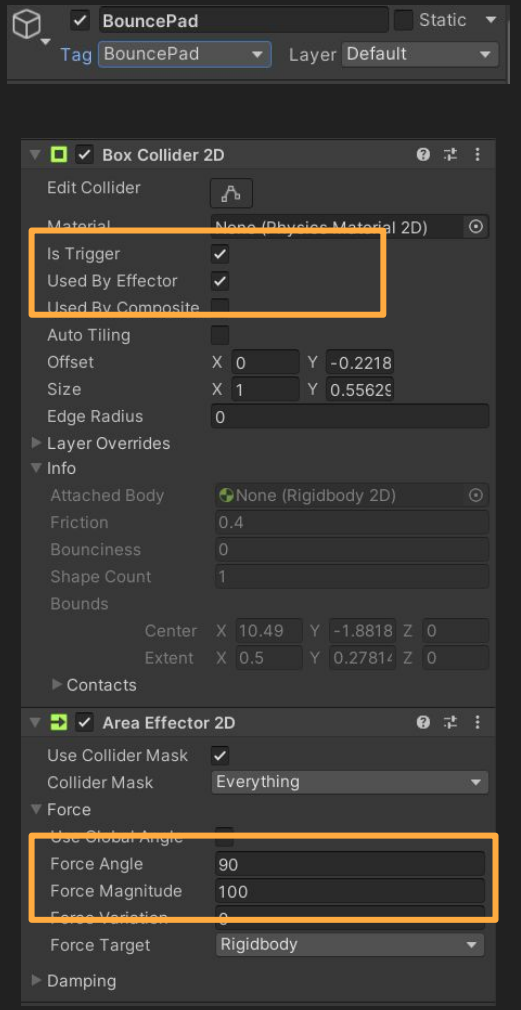


Bounce Pad

The bounce pad will be the simplest object. The parent will have a Box Collider that is a trigger and uses an effector. The effector we use is the Area Effector 2D to bounce the player 90 degrees in the sky. Its child will have a sprite of a spring.





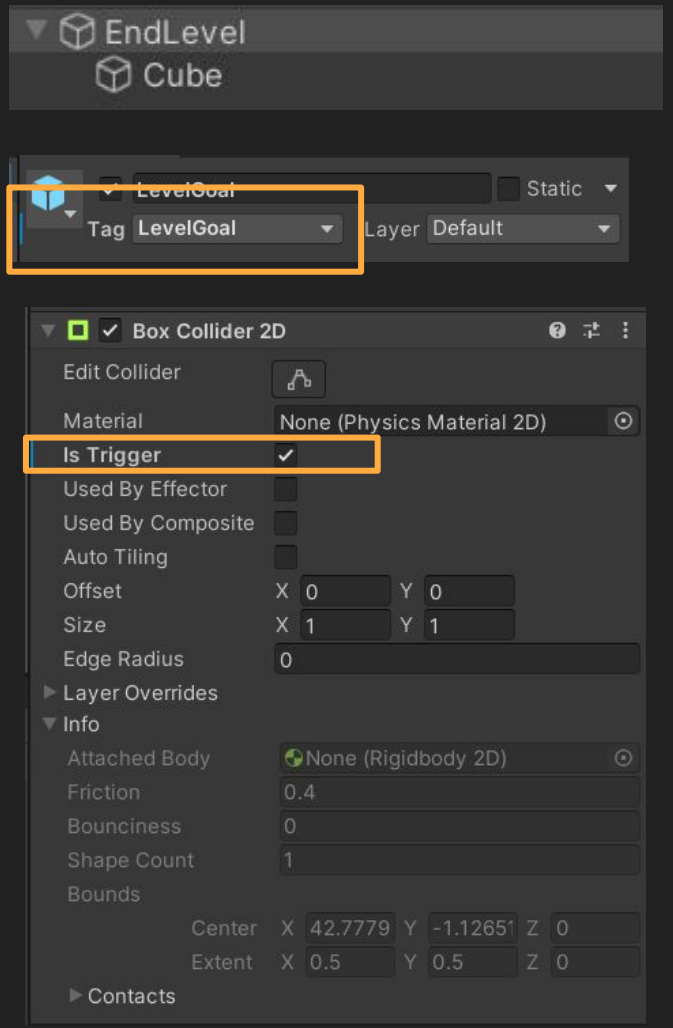


End Goal

For the last object, we're going to make it special.

We're going to place a 3D object in our 2D game.

The parent of the object should have a 2D box collider that's a trigger. And it has the LevelGoal tag attached to it.

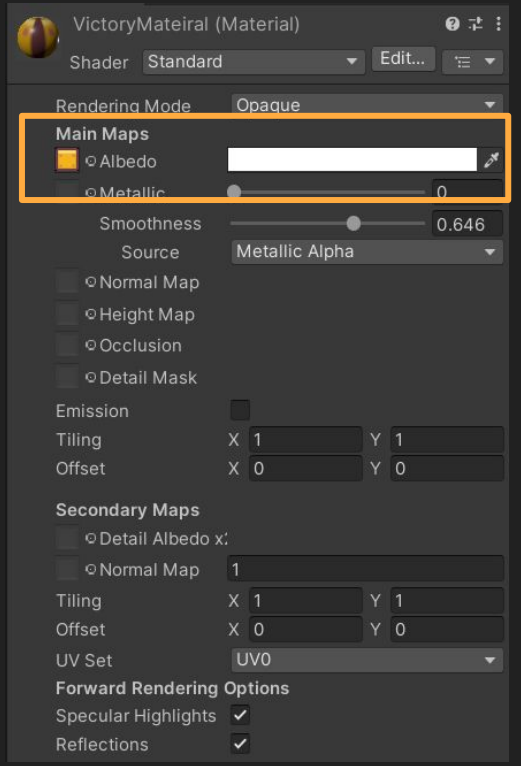


End Goal Texture & Material

In your Materials folder, you will find a material called "Victory Material."

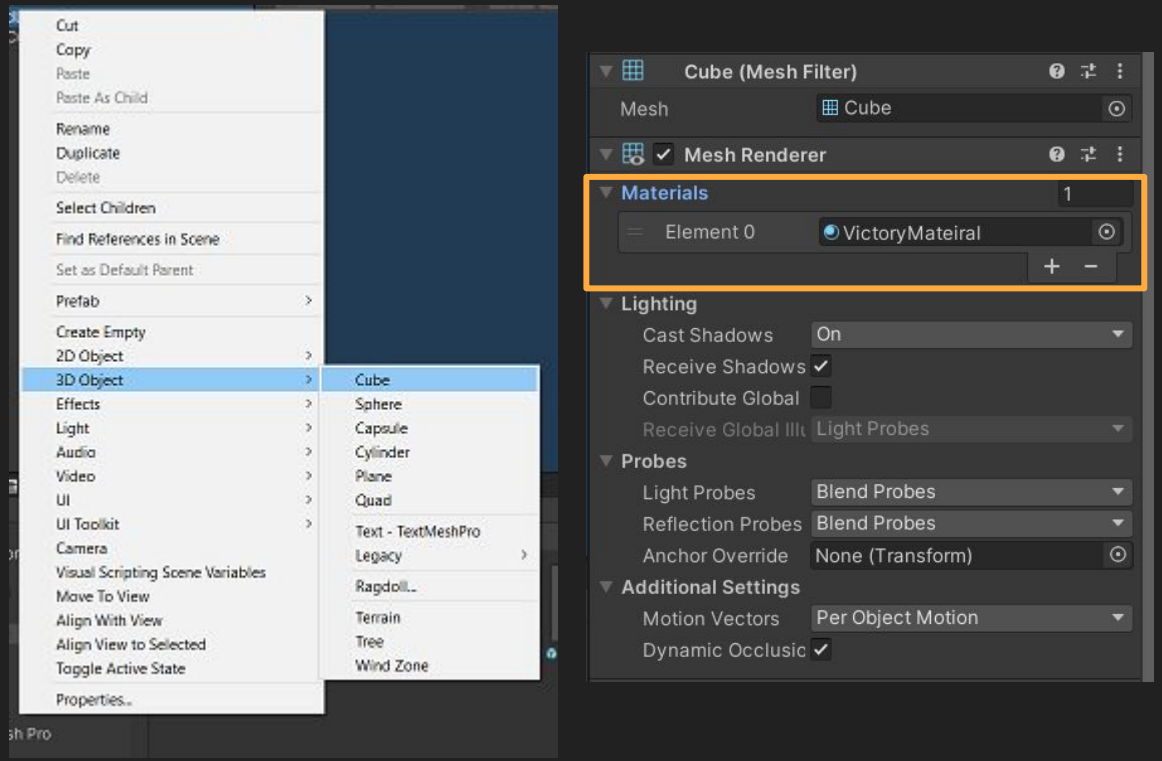
We're going to attach a Box Texture to it so that it looks like the box from our art.

Drag the texture into the albedo of the material.



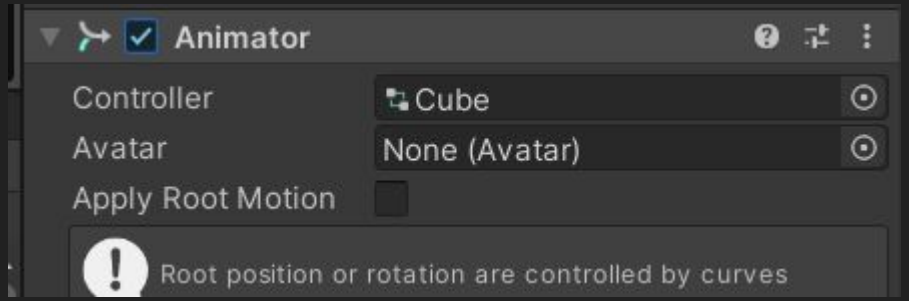
Cube

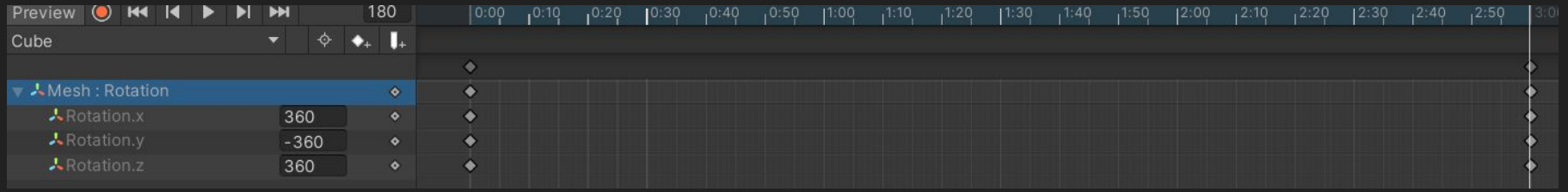
Create a cube as the object, and in the Mesh Renderer, attach the "Victory Material."



End Goal Animation

Select the parent and go to the animation window. We will create an animation that lasts 3 minutes and will go from 0,0,0 rotation to 360,-360, 360. This will keep the cube spinning and make it an interesting object in the game.

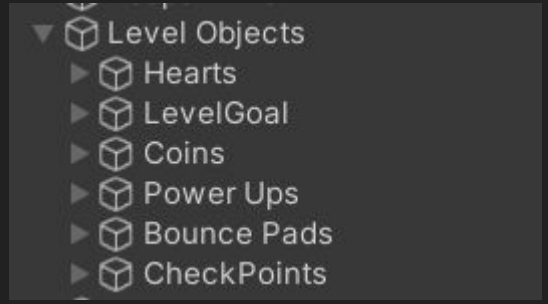


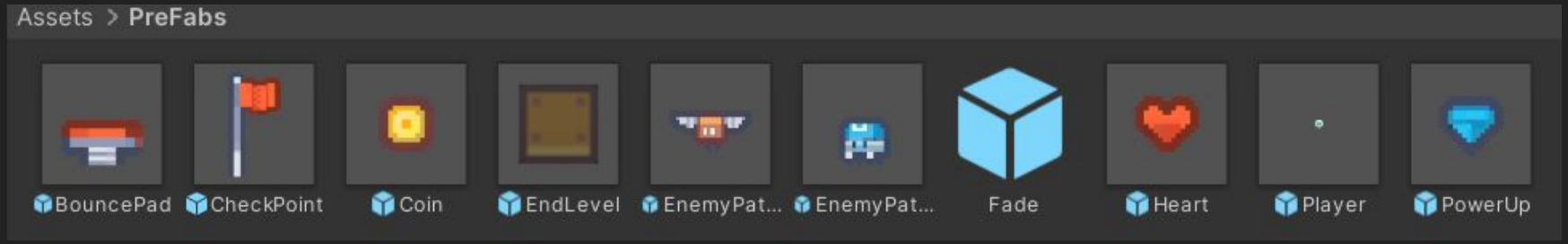


PreFabs

Once you've created all of your objects, make them into Prefabs by dragging them into the Prefab Folder from the hierarchy.

Also, make sure to create parent game objects for each of the items you've created. This will help you keep your level organized.

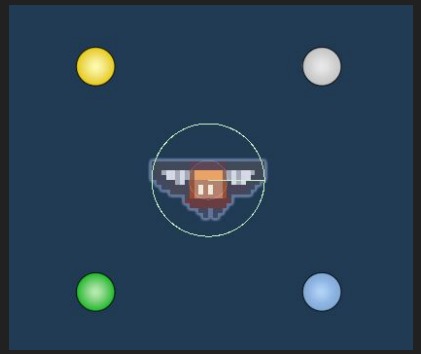


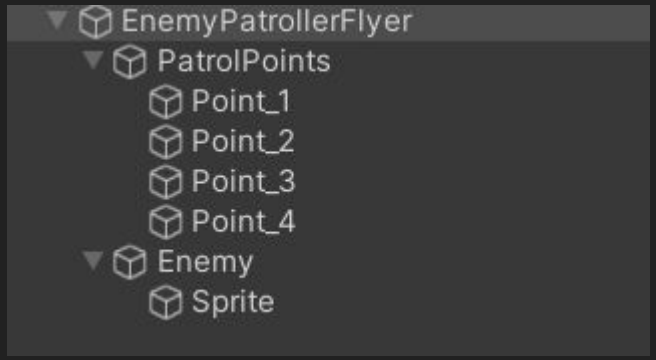


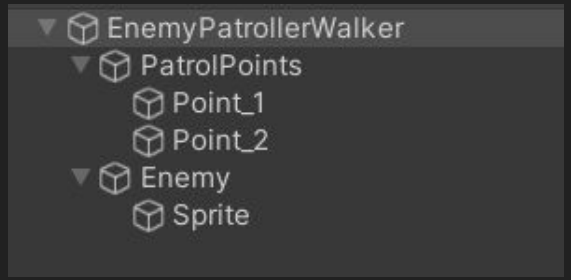
Enemies

In your Prefabs folder, you will find two enemies that are pre-made for you.

They are made up of two components: a Patrol Points component, which dictates their travel path, and the Enemy itself, which will kill the player if they are touched.





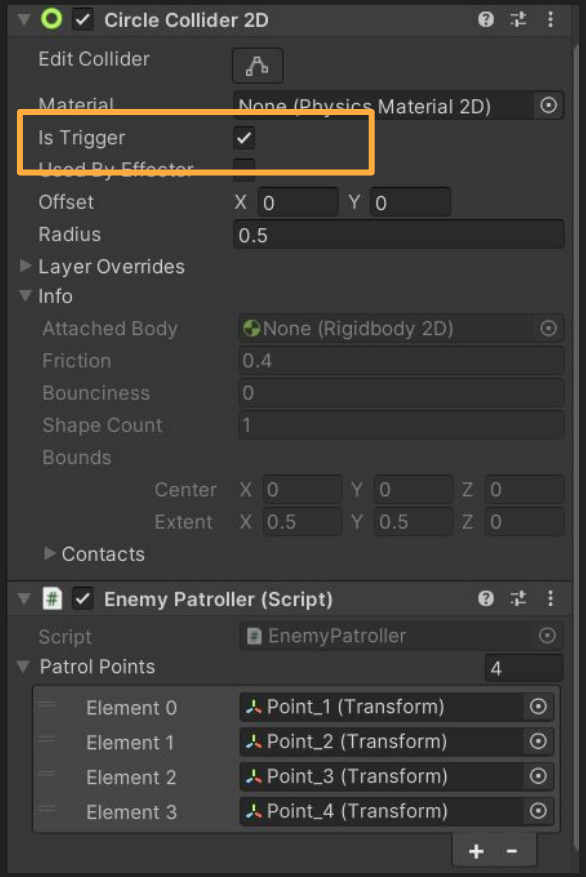


Enemies

The enemies have a circle collider that is triggered and with a script that will reference the patrol points.

You will notice that the points are separated from the enemy.

If they were under the enemy, the points would move when the enemy moves, meaning the enemy would never reach them, so we separate them to not affect their position during the enemy's movement.



Enemy Script

The script is made up of the points that the enemy will move between and the time it will wait at each location after getting there.

We also pre-set it to appear at the first point when the game starts.

```
Unity Script (2 asset references) | 0 references
public class EnemyPatroller : MonoBehaviour
{
    //================ Movement
    public List<Transform> _patrolPoints = new List<Transform>(); //Holds all the positions that the enemy will travel
    private const float Speed = 2f;                                //Speed of the enemy
    private int _currentPointIndex;                                //Which point are they at right now

    //================= Pause
    private float _waitTime = 0.5f;                                //Timer
    private const float StartWaitTime = 0.5f;                       //What timer resets to


    // Start is called before the first frame update
    Unity Message | 0 references
    private void Start()
    {
        //Sets the position and rotation to point 1 or 2 based on isStartingOnPointOne state
        transform.position =  _patrolPoints[0].position;
        transform.rotation = _patrolPoints[0].rotation;

    }
```

Enemy Script

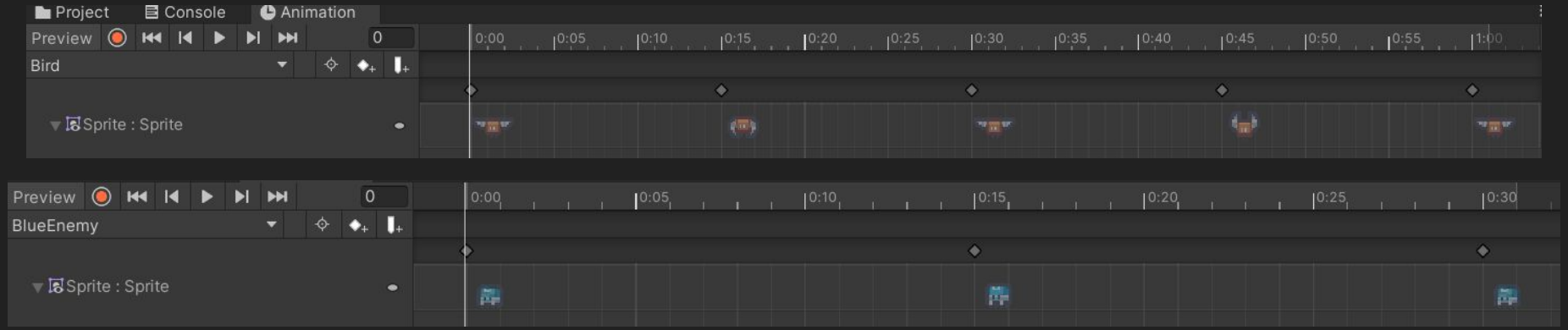
The enemy will move along the path to the current location.

When they reach it, they will set the next location as their destination. If the next current location they're heading to is the final one, they will look back to the first location and repeat the process.

```
//Updates the enemy to move between different points provided in the array
Unity Message | 0 references
private void Update()
{
    //Moves towards the current objective
    transform.position = Vector2.MoveTowards(transform.position, _patrolPoints[_currentPointIndex].position,
        Speed * Time.deltaTime);

    //Checks if the enemy has reached the point it's moving towards
    if (transform.position != _patrolPoints[_currentPointIndex].position) return;

    //Checks if it has stayed on the point for long enough
    if (_waitTime <= 0)
    {
        //If its last point in array go to first point
        if (_currentPointIndex == _patrolPoints.Count - 1)
        {
            _currentPointIndex = 0;
        }
        //Go to next point in array
        else
        {
            _currentPointIndex++;
        }

        //Update the rotation of the sprite
        transform.rotation = _patrolPoints[_currentPointIndex].transform.rotation;
        //Rest the timer
        _waitTime = StartWaitTime;
    }
    //Else count down till it leave the point
    else
    {
        _waitTime -= Time.deltaTime;
    }
}
```

Enemy Animations

The two enemies have different animations set to them where they swap between different sprites to make them appear as if they are moving.



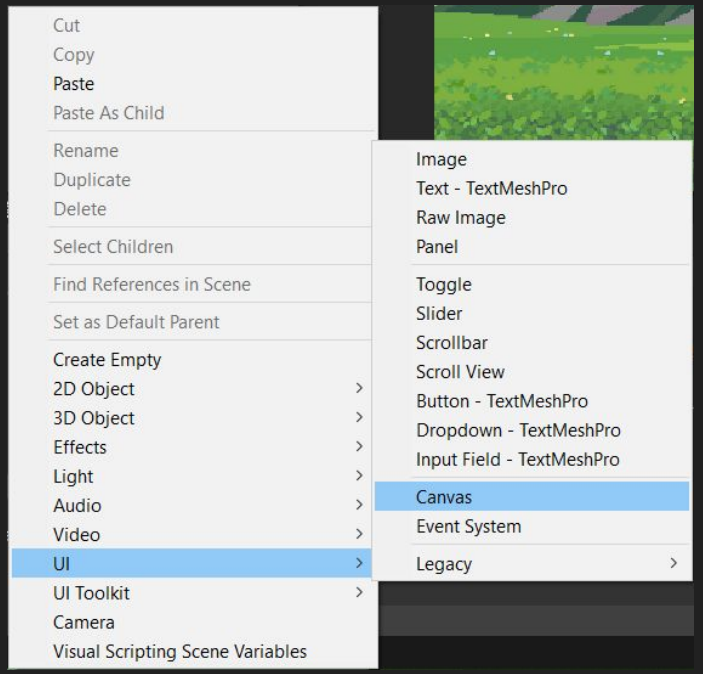
Level UI

Canvas

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

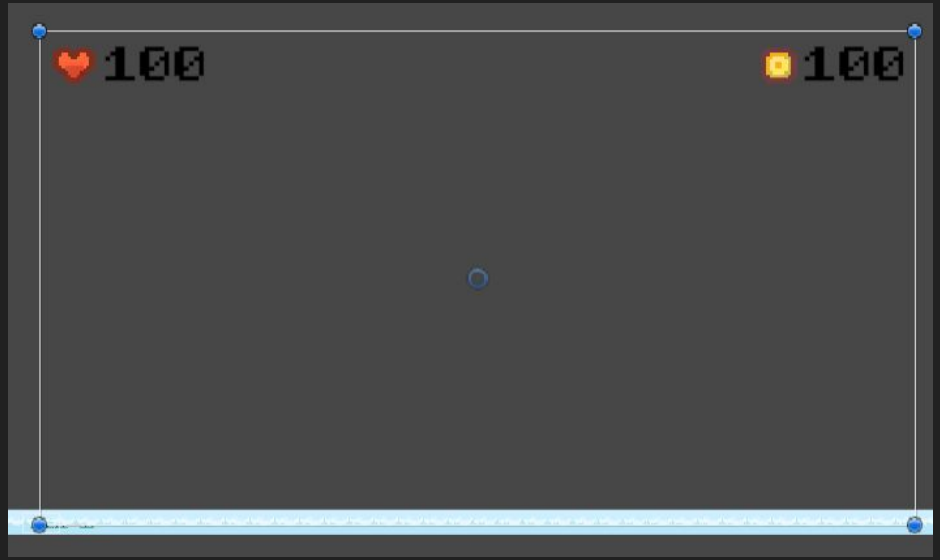
1. Game Over UI
2. Level UI
3. Victory UI
4. Fade

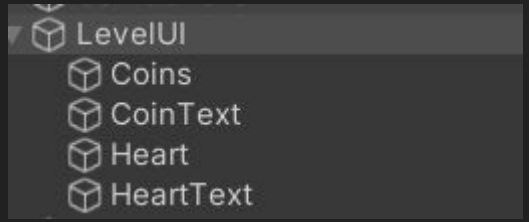
Like before, we will use Fade for transitions. The rest of them, we will construct from the ground up.



Level UI

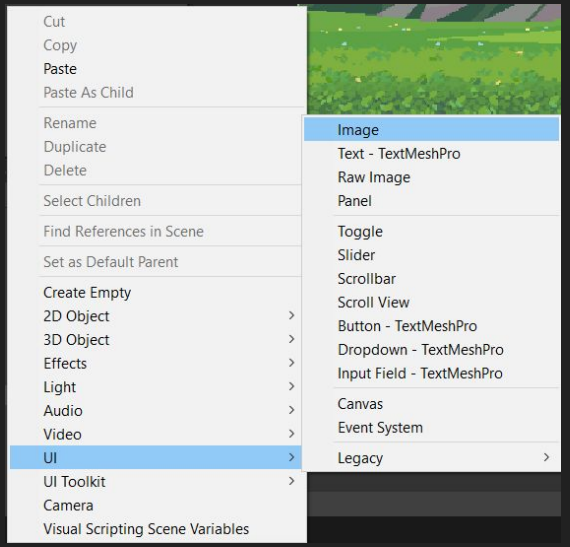
We will start with the Level UI; this is what the player will see when playing the game. We will have two Images and two TextMeshProGUI elements in it.

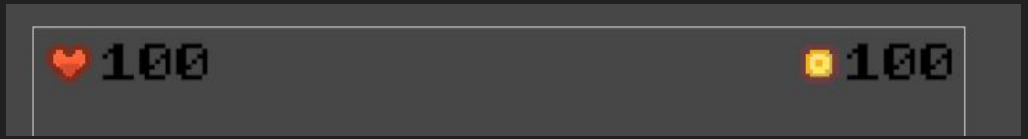


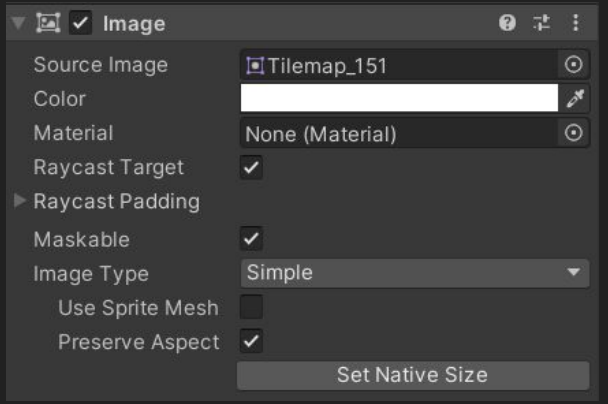


Images

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.





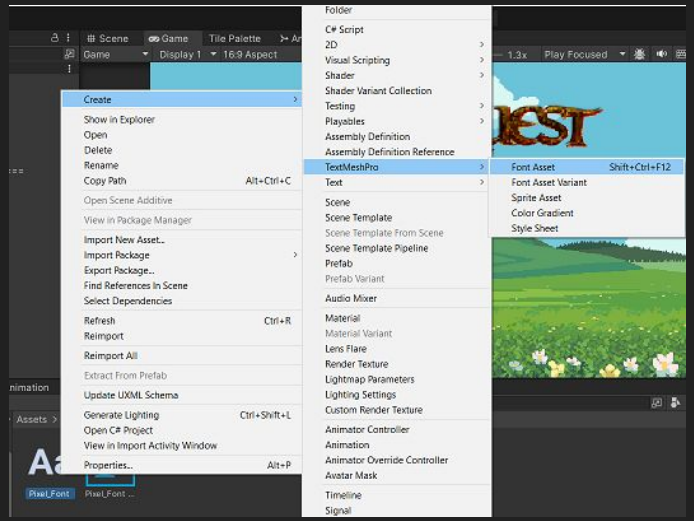


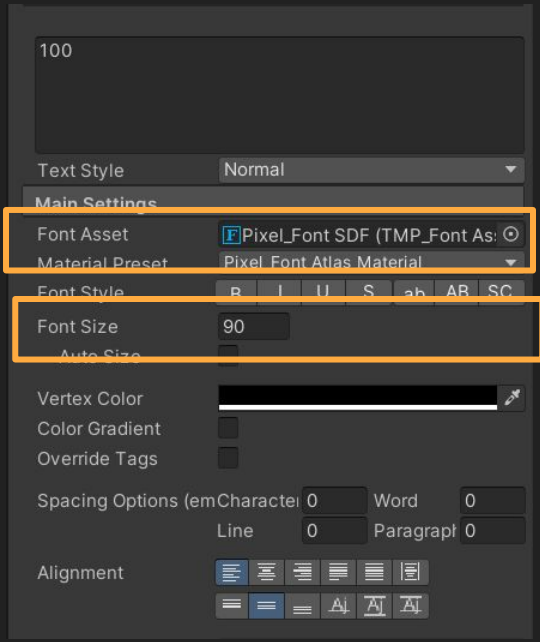
TextMeshProGUI

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.

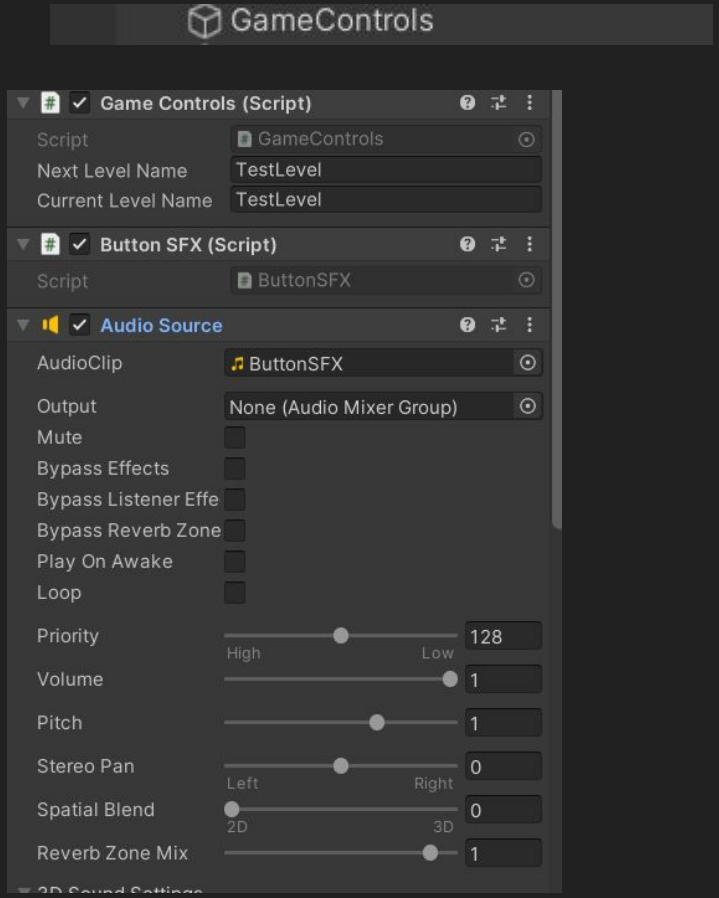




Game Controller

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.

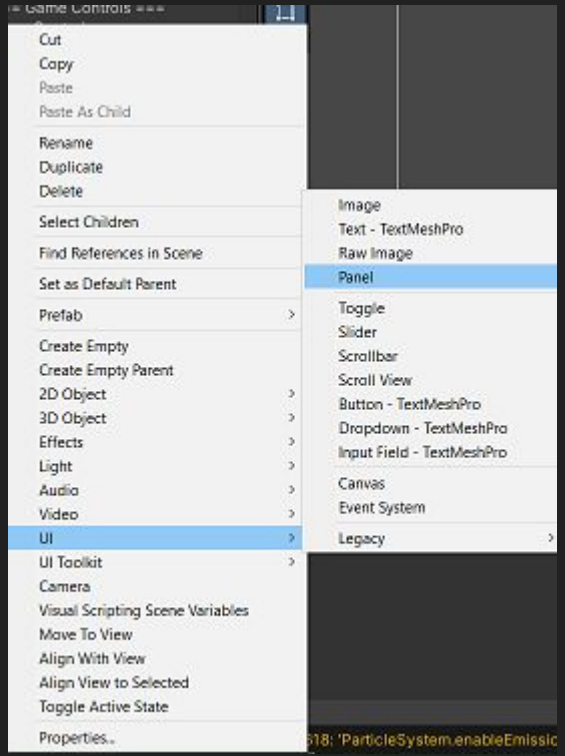
It will have two buttons: one to quit back to the main menu and another to reload this level. Visually, it will have a TextMeshProGUI and an image in the middle to give it some flare.

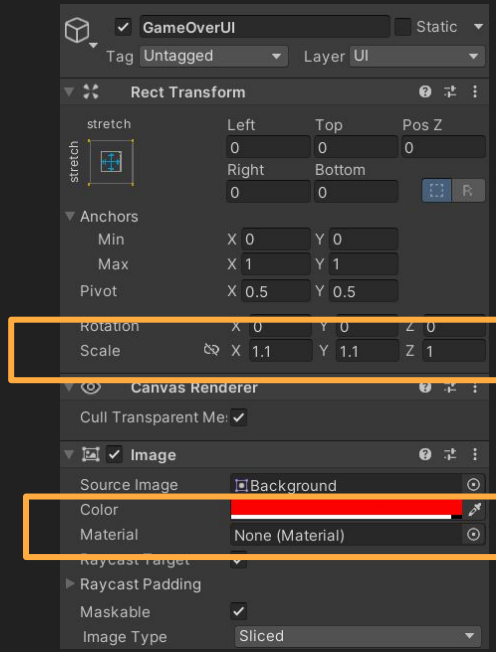
Game Over

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.

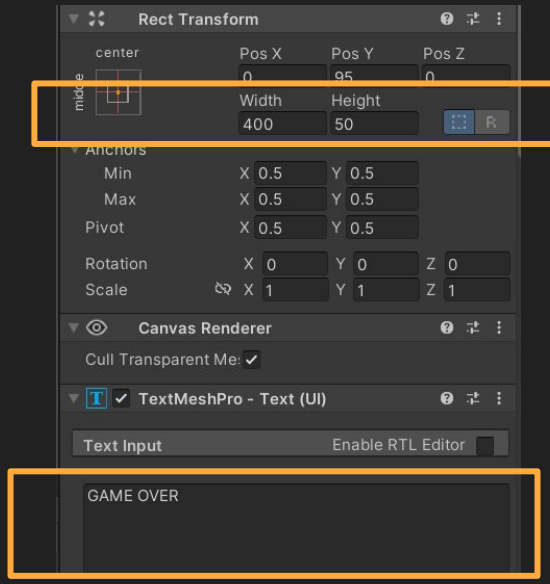
In the image settings, make it almost fully red, with some alpha to allow the player to slightly see the level.

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



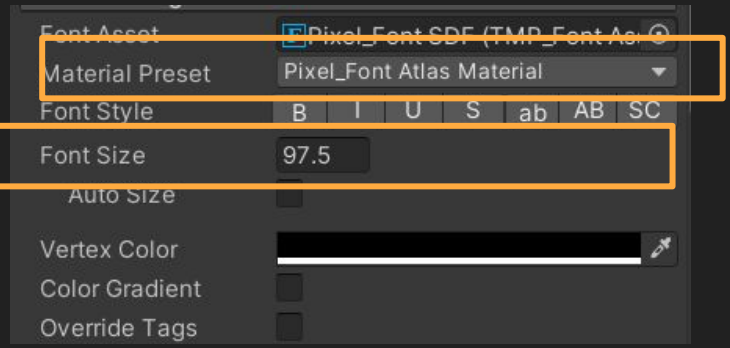


TextMeshProGUI



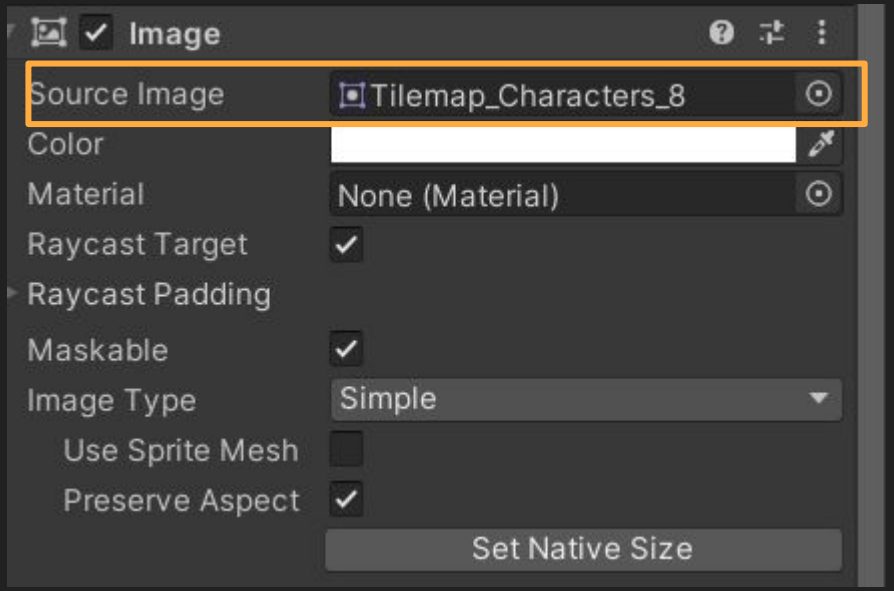
Create a TextMeshProGUI game object, make the width 400 so the larger text fits.

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



Image

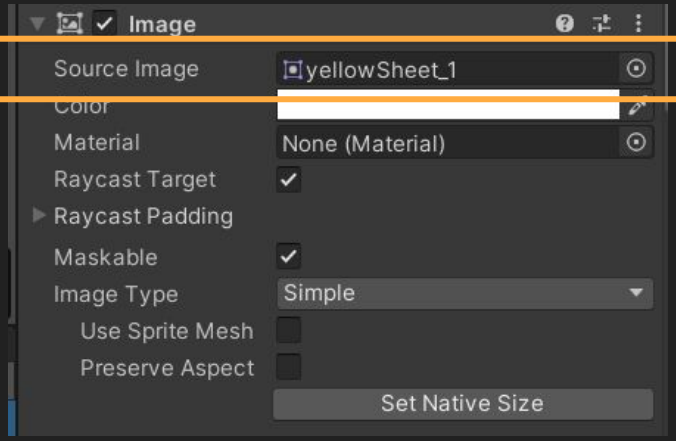
Create an image and place the Tilemap_Character_8. This will add some flare to the screen.

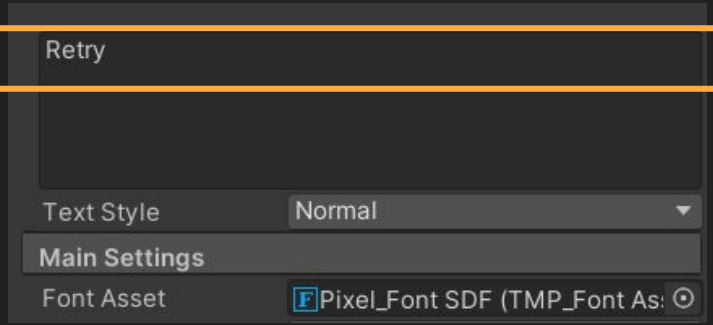


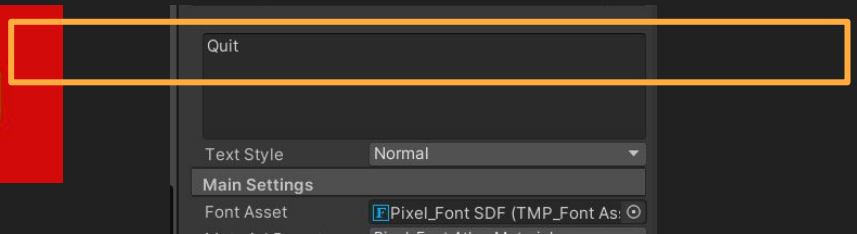
Buttons

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.



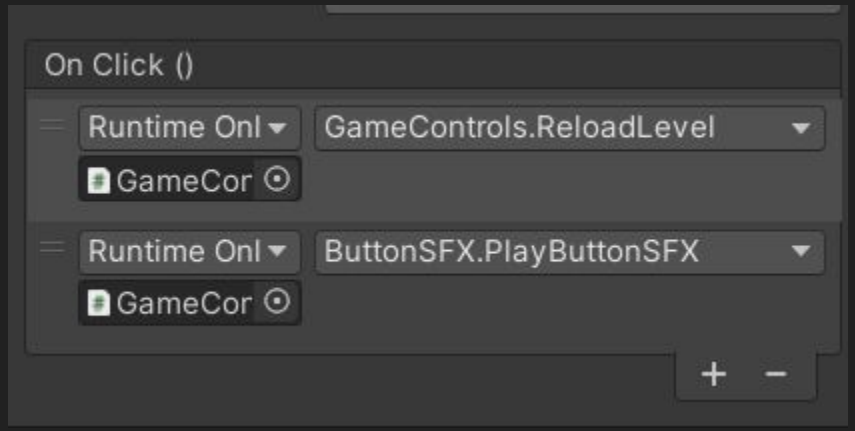


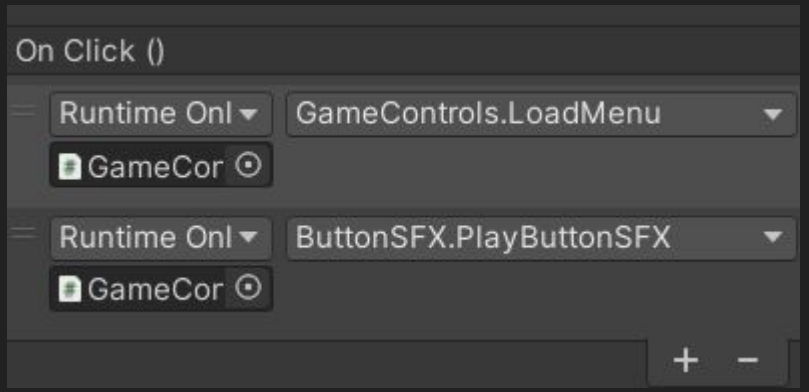


Button Functionality

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

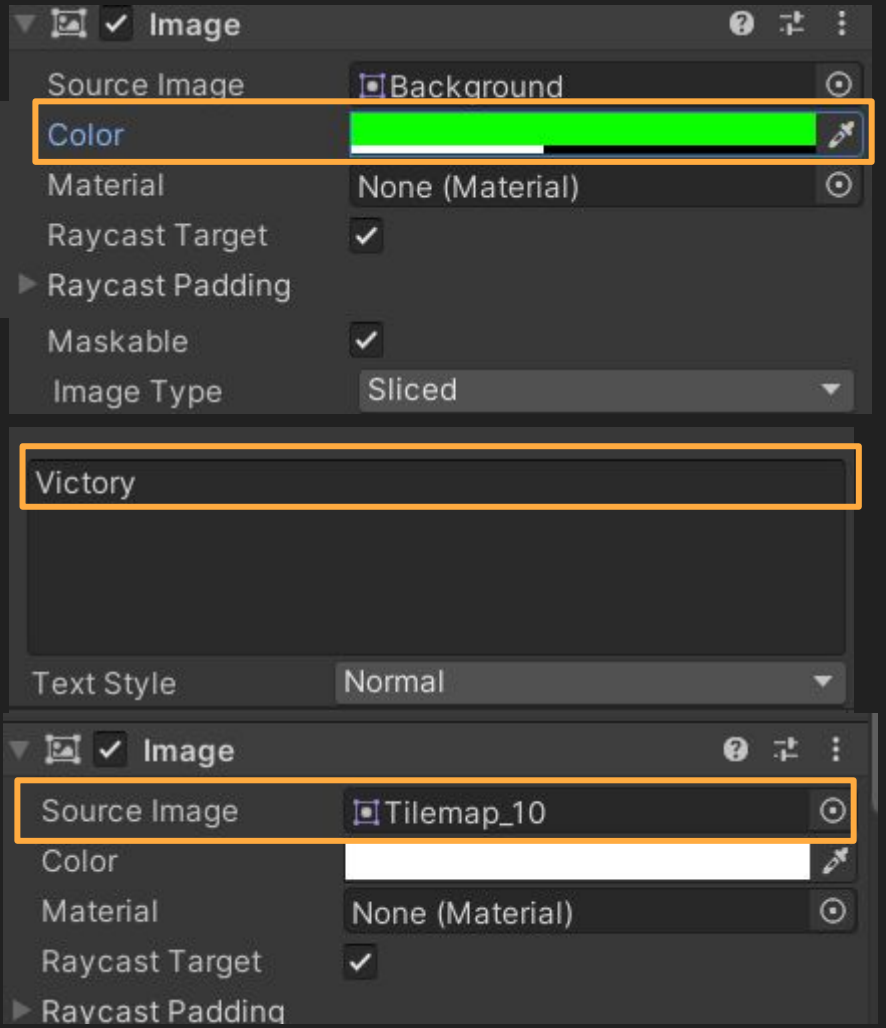
The Victory UI we are creating is basically the same as the Game Over UI, so we'll duplicate the Death UI and tweak it to work the way we want it to.

Victory UI

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

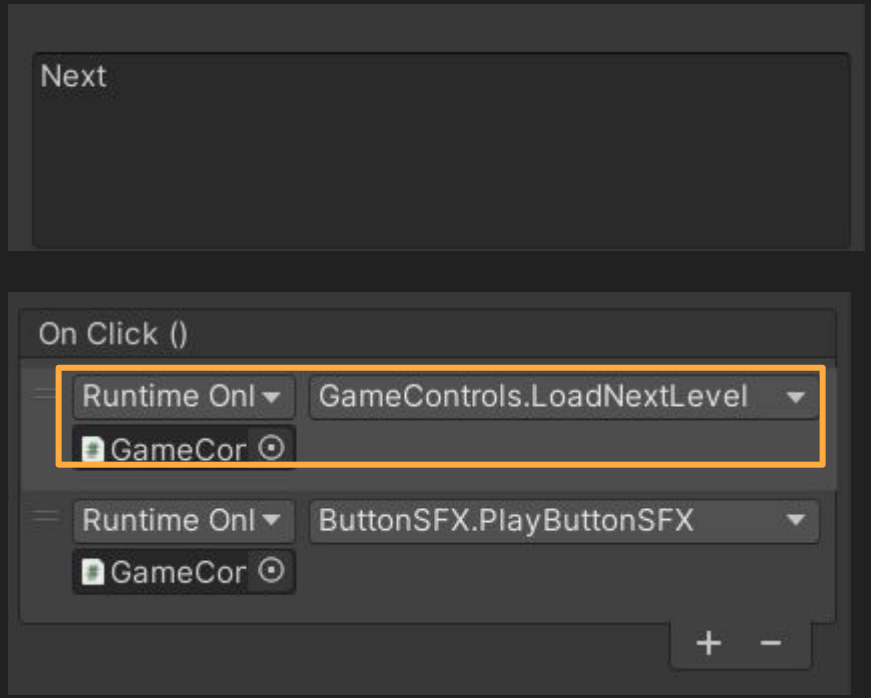
Set the color of the background image to be green.

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.

Music & SFX

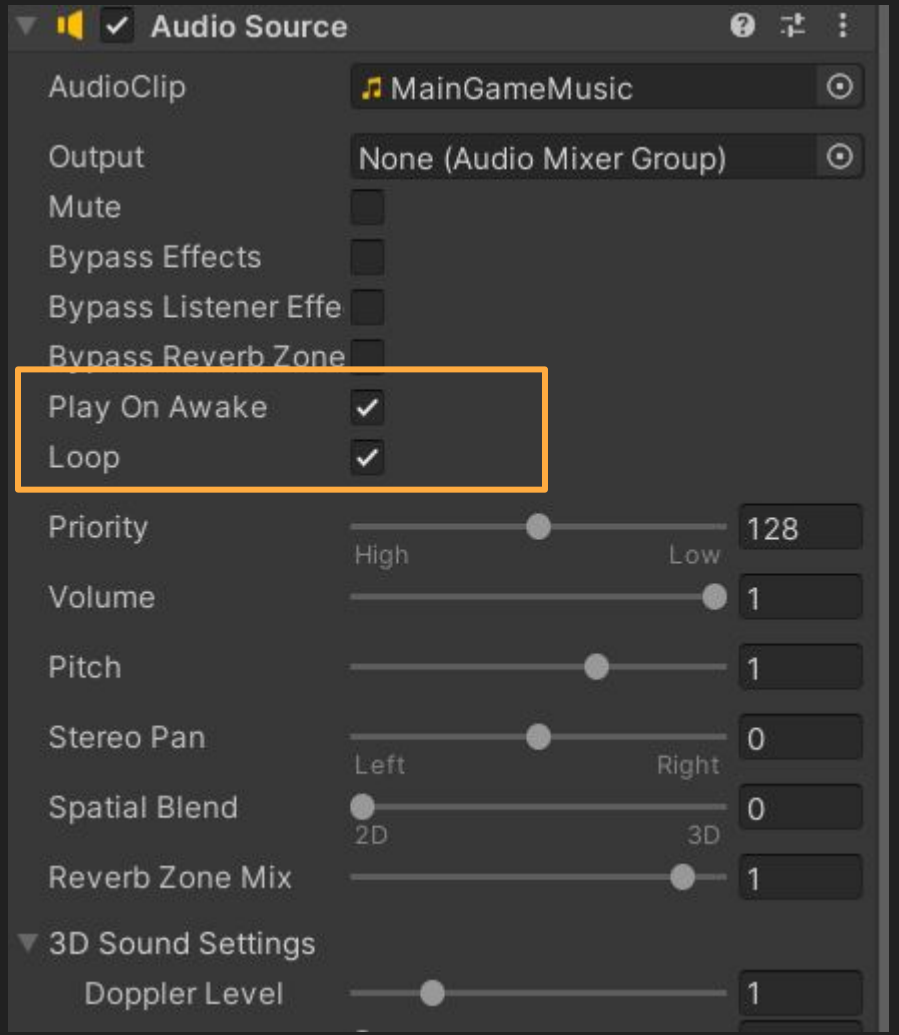
Music & SFX

We will create two parent empty objects that will hold our SFXs and our Music.

We will then create one game object with the AudioSource Component and duplicate it to create all of the different SFXs and Music sources. Make sure you spell the names just as shown, as the code searches for those objects specifically.

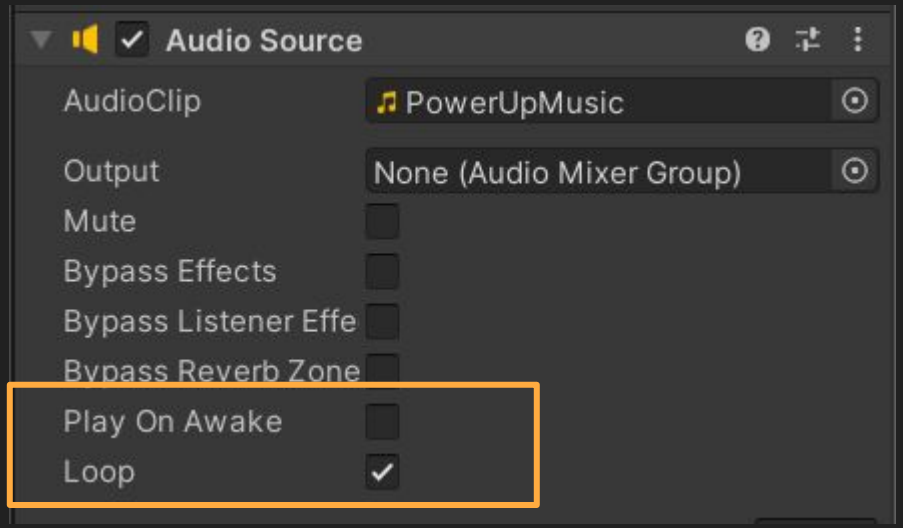
Level Music

Starting with the level music game object, make sure it has the "MainGameMusic" audio clip and the "Play On Awake" and "Loop" enabled.



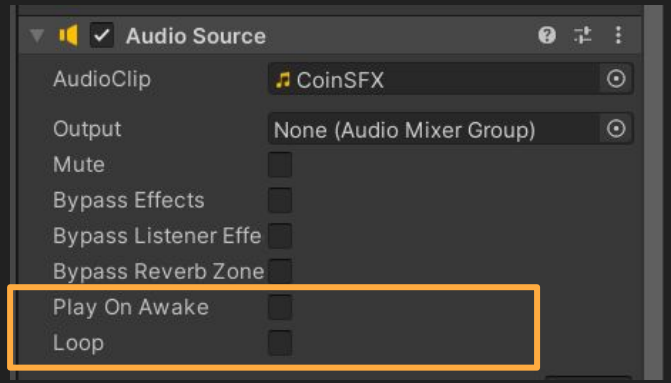
Power Up Music

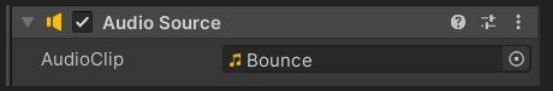
Duplicate that object and replace the audio clip with "PowerUpMusic," and let "Loop" be on, but turn off "Play On Awake."

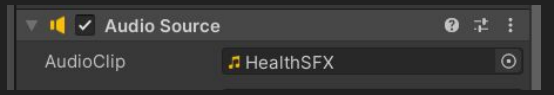


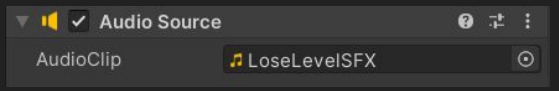
SFX

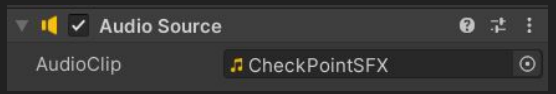
Finally, duplicate the audio sources and connect them to their specified SFX. Make sure to turn off both "Play on Awake" and "Loop" for these Audio Sources.

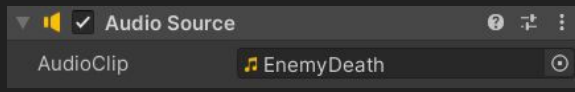












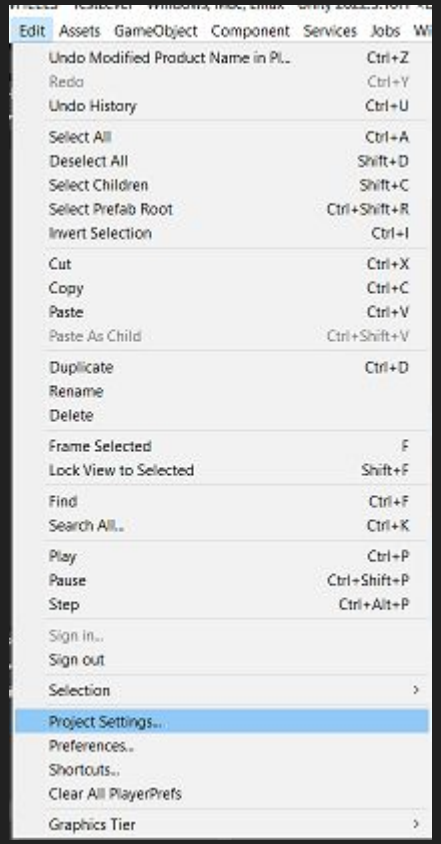
Projects Settings & Building the Game

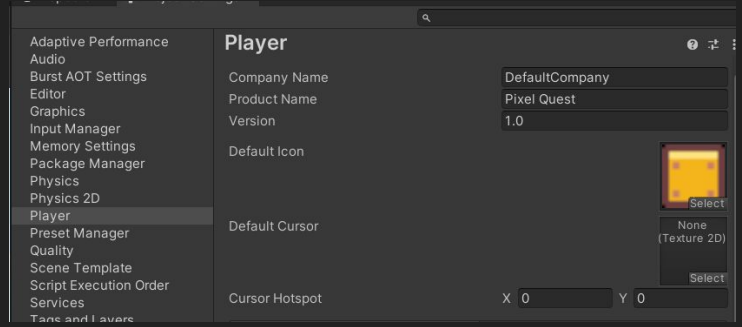
Project Settings

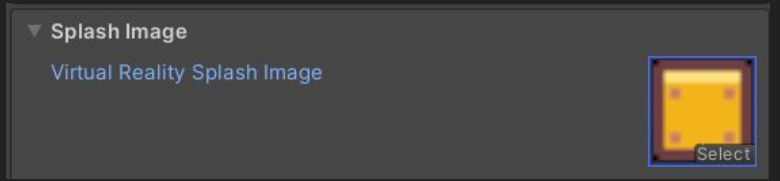
Open up the project Settings; here, you can go to the Player tab.

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.