

# How to make a 2D platformer game

By: The Cool Guys

# Table of Content

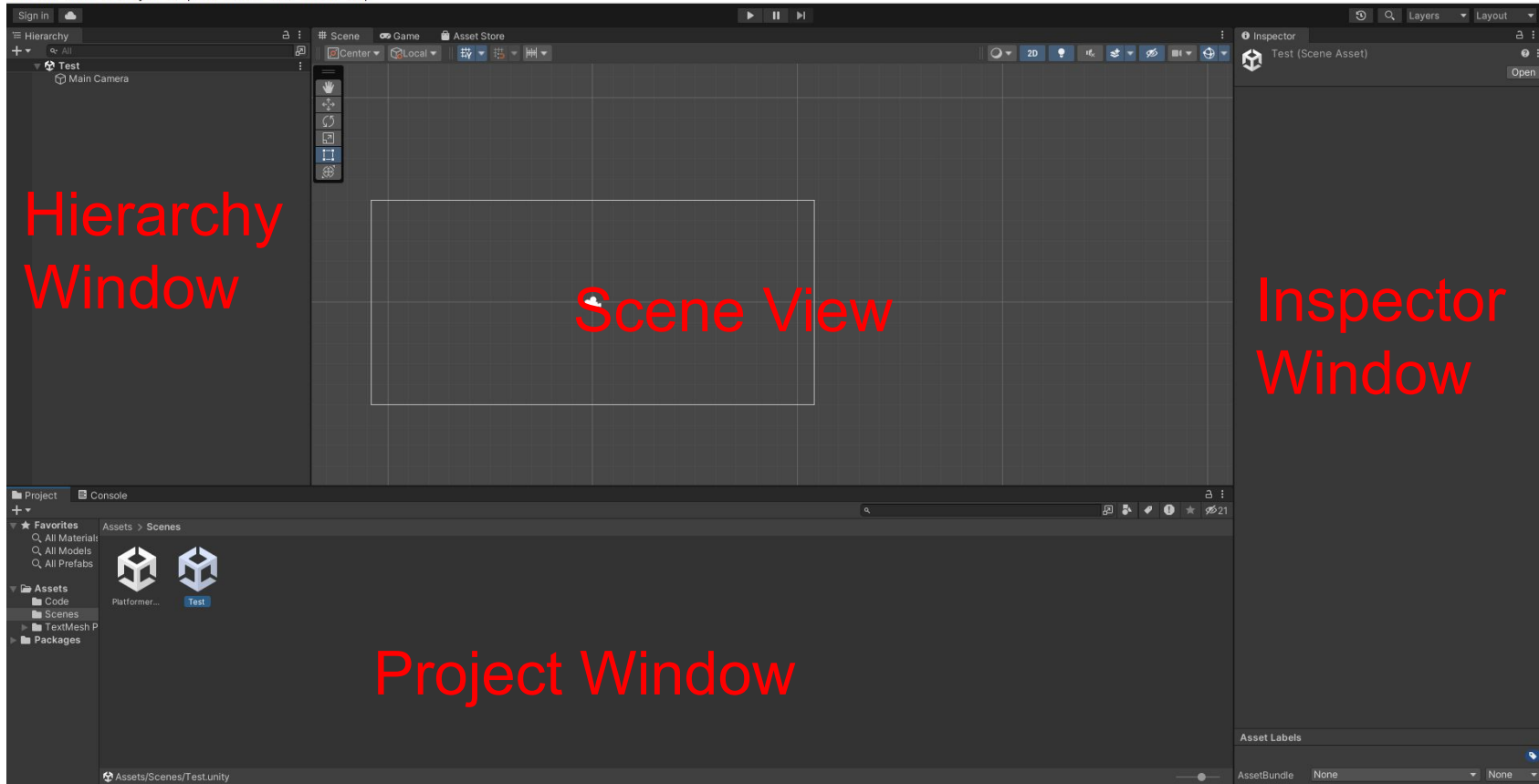
1. Unity Basics
2. Base Game/First Iteration
3. Sprites/Animation
4. Level Design
5. UI/Collectibles
6. Sound and Backgrounds

# Unity Basics

# Unity Basics

What we will be covering:

- Terms and Locations
- Game Object: Sprites/Objects under the hierarchy section
- Tags: A label given to objects in inspector
-



# The First Iteration

## Quick Note

No code will be put in this section all code you can access and see from the github and or the videos that should be posted after the lectures. They will be referenced but will not be posted here as they would be images and would be easier to see from the Github.

Makes it easier so code is not just plastered everywhere. As stated before there should be a video recording which you can follow along to.

# Making The Foundations

What we need:

- Game Manager
- Player
- Platform
- Goal
- Enemy/Obstacles
- Collectable



# Make the Game Manager

1. Create a empty object and call it GameManager
2. Make a script called Game Manager and attach it to you object
3. Add code over time to handle game data and triggers

# Making The Player - Part 1

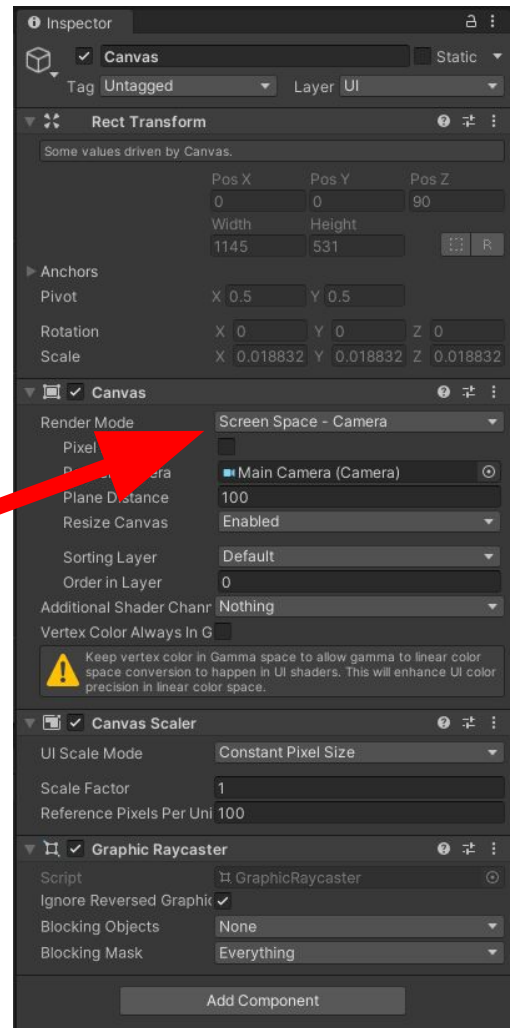
1. Make the Player Object and a Platform
  - a. Right click on hierarchy and create two 2D square objects. 2D Object->Sprites->Square
2. Make the Main camera a child of the player object.
  - a. In hierarchy window drag the camera over the player
3. Give both objects a Rigidbody2D and a BoxCollider2D
  - a. Go to inspector window and click the add button on the bottom
4. Change settings so both cannot rotate in the Z axis (do x and y as well for platform)
  - a. Under RigidBody2D and constraints for each object
5. Make a C# script called Player Movement and add it to the Player Object
  - a. Go into the project window right click and select Create->C# script

# Making the Goal - Part 1

1. Make the goal object (like the player and platform)
2. Make a text box (Under UI in hierarchy look back at image above)
3. Set canvas to camera render
4. Add code to GameManager to trigger the event and show text
5. Make the trigger script in order for the goal to tell game manager when to run function

# Image Reference

This is the box to  
change canvas  
render in Inspector



# Making the Trigger script

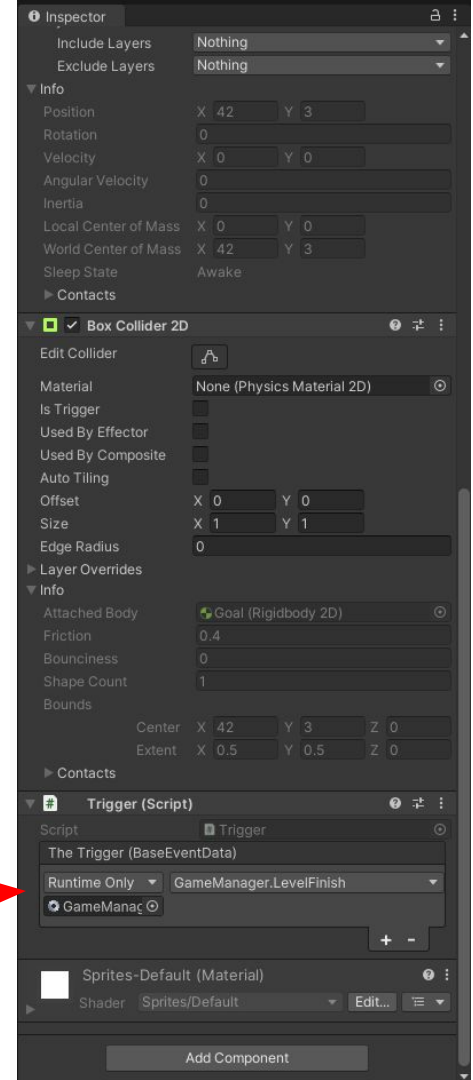
1. Import Event System form unity
  - a. `using UnityEngine.EventSystems;`
2. Create it so when these objects collide with a object with the tag player (like the ground check) that it triggers an even we will set in the inspector.

## Making the Goal - Part 2

1. Now add the new trigger script to the goal object
2. Add a instance by pressing the plus on the trigger tab in inspector
3. Go to the hierarchy tab and drag the game manager into the slot on the trigger code base event data.
4. Then click on no function and go GameManager->LevelFinish() (or what you named the function)

# Image Reference

Drag the actual  
Game Manager  
Object into the  
box and then  
select not the  
script it will  
change after



# Health and Damage

1. Add a public health variable to game manager
2. Create a text to show health
3. Make two public functions in game manager to remove and add health
4. Add triggers to objects that want to remove or add health
  - a. (Same as image above different function)
5. Make health reset on death/reset



# Collectables and Score

1. Add a public score variable to the game manager
2. Create text to show score
3. Make a public function in game manager to add score
4. Add triggers to objects that will add score. (Same as goal)

# Making the Enemy - Part 1

1. Create a gameobject square and name it enemy (Same way as player)
2. Create a empty gameobject and call it ground detection. And make it a child of the enemy object. (Like game manager and camera for child)
3. Add a rigidbody2d and boxcollider2d to the game object. Like the player make it not rotate in the z direction.
4. Create a C# script named Enemy Movement and add it to the gameobject.

# Making the Enemy - Part 2

Coding the enemy:

1. Make patrol behavior
  - a. Apply a uniform speed to the enemy
  - b. Create a raycast off of ground detection pointing down and once it doesn't detect ground flip the object 180 degrees and the same once it hits the other side.
2. Create chase behavior once player is close.
  - a. Find the distance between the enemy and the player
  - b. If it reaches that distance then it should apply a vector towards the player
  - c. Otherwise go back to patrol pattern

# Sprite Animations

# Players/Enemy Animations Setup

- Pick a sprite sheet for your game that has a transparent background
- Click on the object you are trying to animate (In this example the player)
- Click Window -> Animation -> Animation
- Create a sprites folder (or use a current one)
  - Add a folder for player specific sprites
  - Drag downloaded sprite sheet into sprites -> player

# Displaying Animations

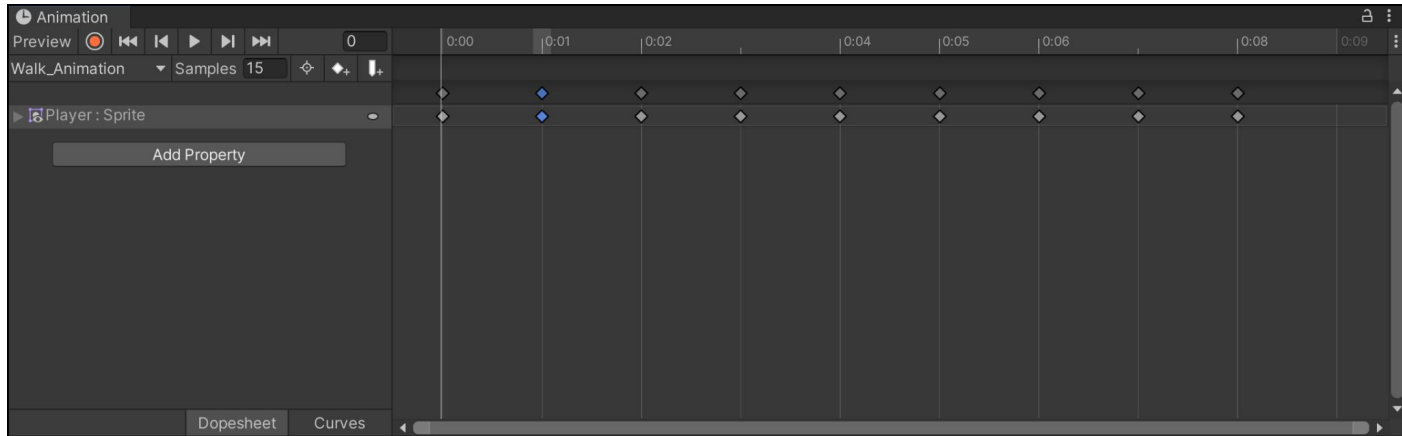
- In the inspector for the sprites change
  - “Sprite Mode” to “Multiple”
  - “Compression” to “None”
  - “Filter Mode” to “Point (no filter)”
- Click on sprite editor -> slice -> slice -> apply. This splits the sheet into individual sprites
- Take the sprite you want as the starting sprite and drag it to player -> sprite renderer -> sprite
- In player -> sprite renderer change “Draw Mode” to sliced and ensure width and height are both set to 1
  - This will change the size of the animation to the correct scale

# Displaying Animations Continued

- Go to sprites -> player and you should be able to click an arrow next to your sprite sheet and see all individual sprites
- Select the sprites you want for a specific animation such as walking and use ctrl+d. You should see all the sprites you picked at the bottom of the player folder
- Drag all the sprites into another folder for consistency and name it something such as “Walk animation sprites”
- In your animation editor click on “create” and name the animation. The name should be related to the specific animation. If it is for the player walking name it “walking animation” for example

# Displaying Animations Continued

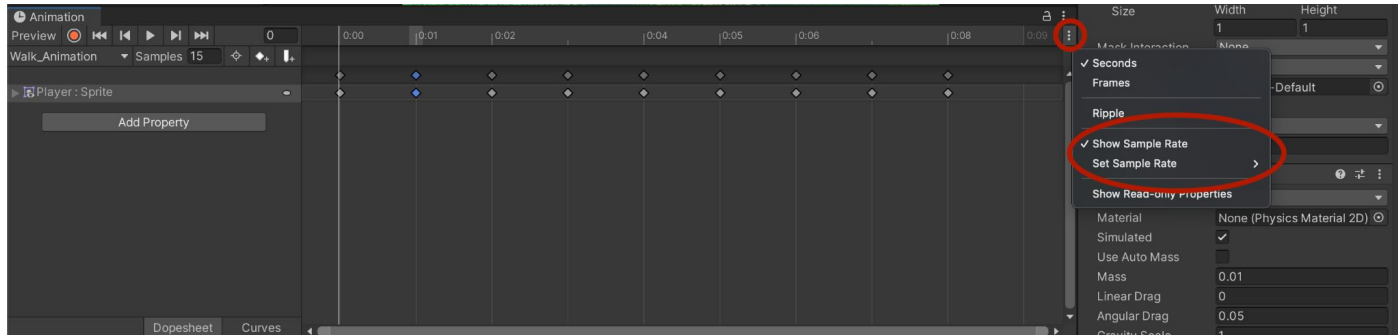
- Click and drag the related sprites into the animation editor and now you should see a bunch of rhombus shapes called keyframes, each associated with a sprite





# Displaying Animations Continued

- You can now adjust how quickly you want the sprites to change creating the animation.
  - An easy way to do this is to adjust the setting for the number of samples on the left side of the animation editor under the play button. Generally 12 works well for animations.
  - If you do not see a samples setting click on the ellipse located on the right side of the animation editor. Select “show sample rate”



# Displaying Animations Continued

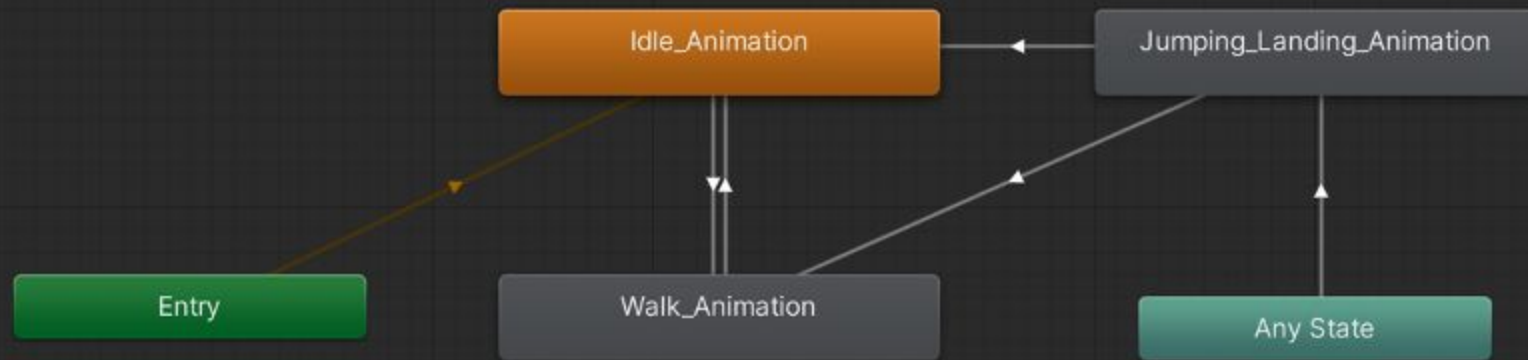
- Now you can click the play button in the animation editor to see how changing the sample rate affects the animation. Remember to adjust the sample rate when it is paused and click enter after entering a number
  - The higher the sample rate the faster the animation will be
- Repeat slides 17 through 21 to add as many different animations as needed

# Connecting Animations to Controls

- At this point we have an animation but need it to respond to player movement. Click on “Animator” next to “Game” and “Scene” at the top of the screen. Here you should see your animations and an entry rectangle in green with an orange rectangle next to it.
- Link the animations that you want to respond to one another by right clicking one of the blocks and selecting make transition
  - For example, if you want to have an idle animation to transition to a walking animation, make a transition between them
  - If you want to switch back to idle when your player stops walking, make another transition back to the idle animation

# Connecting Animations to Controls

- A basic platformer game would likely have any state make a transition to jump because you would want to be able to jump at any point
- We would make another connection from jump to idle and walking because we would want to be able to go from jumping to walking or idle
- We would then connect walking and idle to each other because we would want to be able to go from walking to idle and idle to walking
- Lastly, we can connect entry to idle because we want to start the game in idle since we won't be doing anything at the start
- Reference the next slide for a visual



# User Input Connection

- We can now transition between animations but want to be able to respond to user input
- In the animator select parameters and click the plus button to add a float
  - We can name this float something like “speed” as it tells us if the player is moving or not
- Select the transitions between idle and walk and set the conditions to the new float called “speed”
- If we go from idle to walk it would make sense to check if speed is greater than 0.1 and if we go from walk to idle it would make sense to check if the speed is less than 0.1 indicating, we aren’t walking
- Lastly, we want to uncheck “has exit time” in the inspector for any animation and change “transition duration” to 0 because we want the transitions to be instant

# User Input Connection Continued

- Go to Player -> PlayerControls (script) -> Animator and link the player animation
- Now, we can go to our player script and create a “public Animator animator;” variable which will allow us to interact with the animation
- We can then add to the update function “animator.SetFloat("Speed", Mathf.Abs(Move));” which will change the “Speed” variable we created in the animator to the absolute value of our movement which will tell us if we are moving or not
- These steps can be modified and changed to do the same thing for jumping
  - In this case it should be easier because we are either not jumping or we are jumping so we can use a bool since it only has two states

# Moving In Both Directions

At this point the animation should work completely, but if we move the other direction the animation won't change so we can add this code to the player update function:

```
if (Move > 0)
{
    gameObject.transform.localScale = new Vector3(1, 1, 1);
}
if(Move < 0)
{
    gameObject.transform.localScale = new Vector3(-1, 1, 1);
}
```

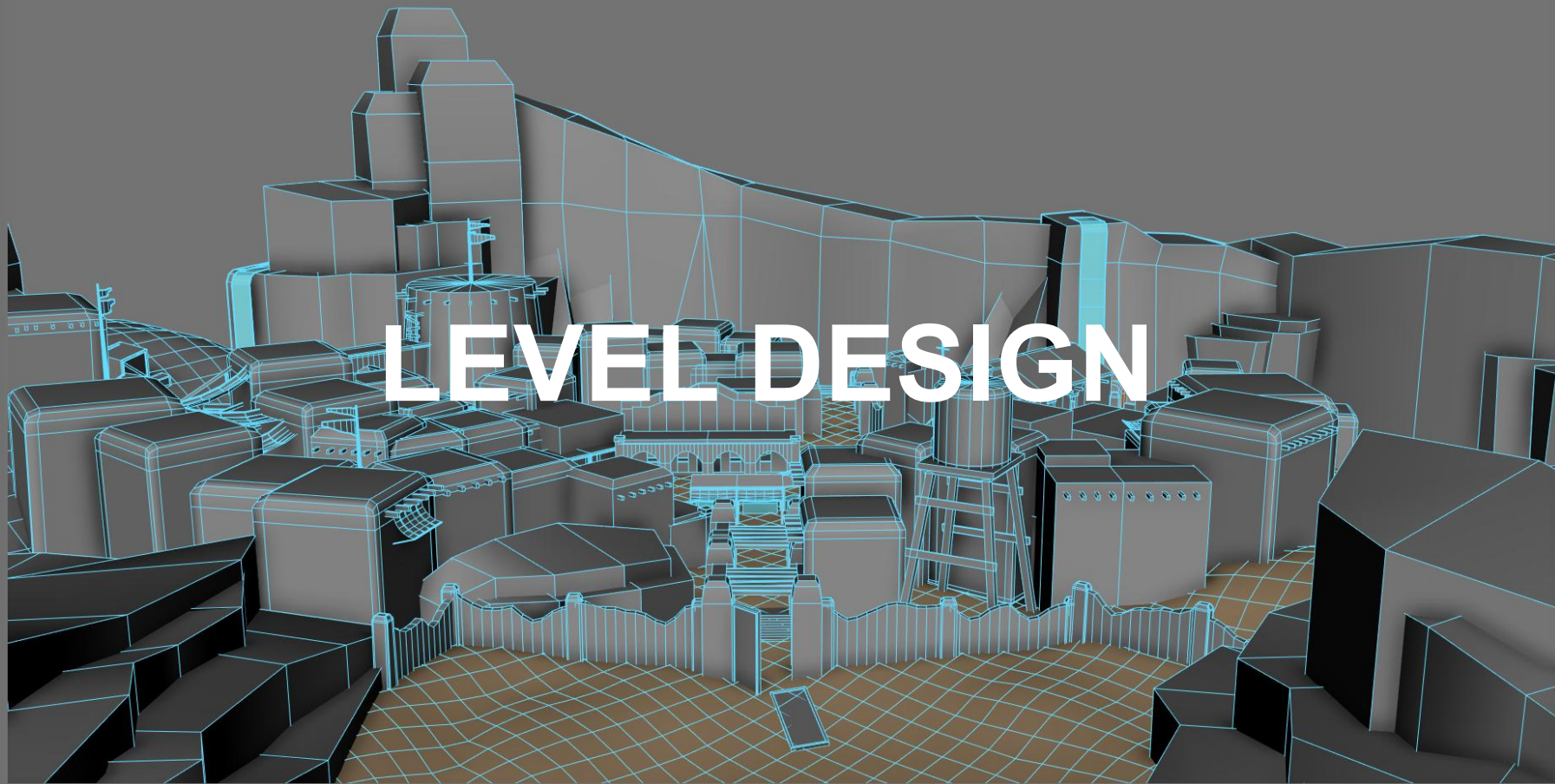
This will check if the movement is going to the right first and if so, change the player animation to move right. The next if statement will check if the player is moving to the left and if so, it flips the animation



# Extra Information

- If you can't figure out how to remove the background of a sprite sheet try the steps below
  - Download the image
  - Use an instant alpha tool and click on the background (this should remove most of the background)
  - Use a lasso tool to remove the rest of the unwanted background and now you should have a usable sprite sheet.
- If on mac you can use the Preview app which works pretty well for me
- If on windows try paint.net
  - <https://www.getpaint.net/>

# LEVEL DESIGN



# COMMON RULE IN DESIGN:

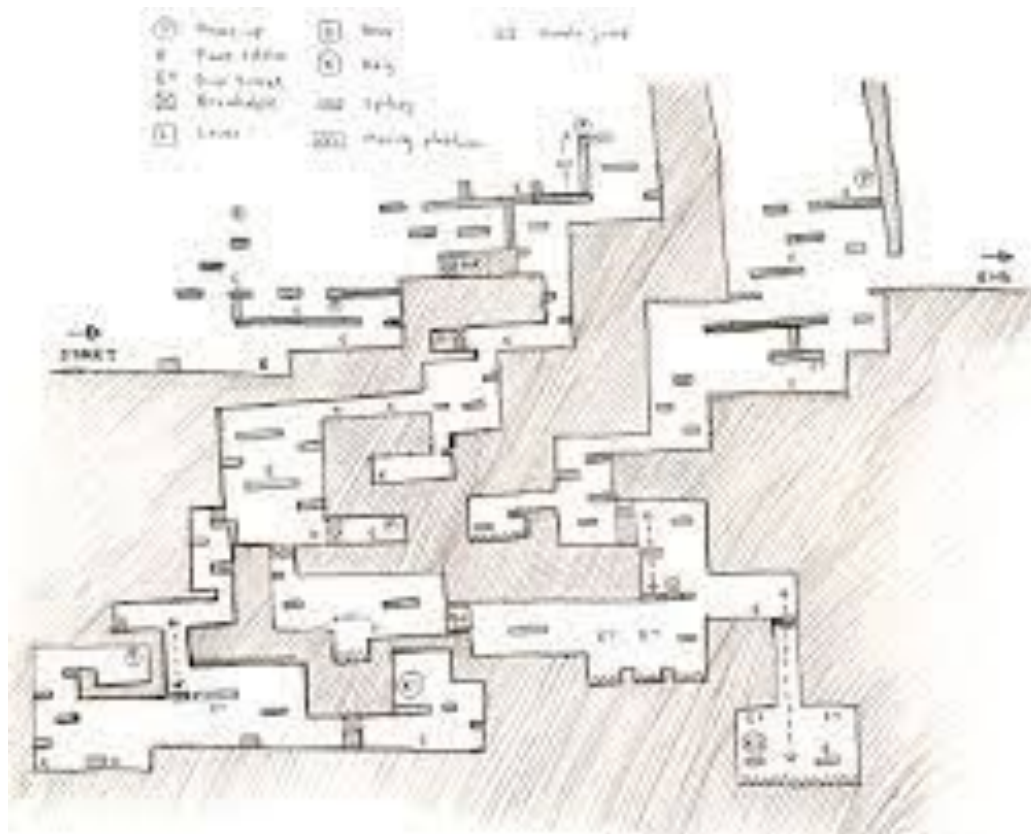
## Suit the customer needs



# Ask:

- What do you want the player to **experience**?
- What are your **goals** for the player?
  - e.g. learn to jump in fifteen seconds
- What **feelings** do you want the player to walk away with having played your game?

# LAYOUT



Consider both the player and what you want your game to be like.

→ Incorporate your vision into an enjoyable experience for the user

## Some things to think about:

- The feeling you've had playing other games
  - What kind of satisfaction you want the player to have





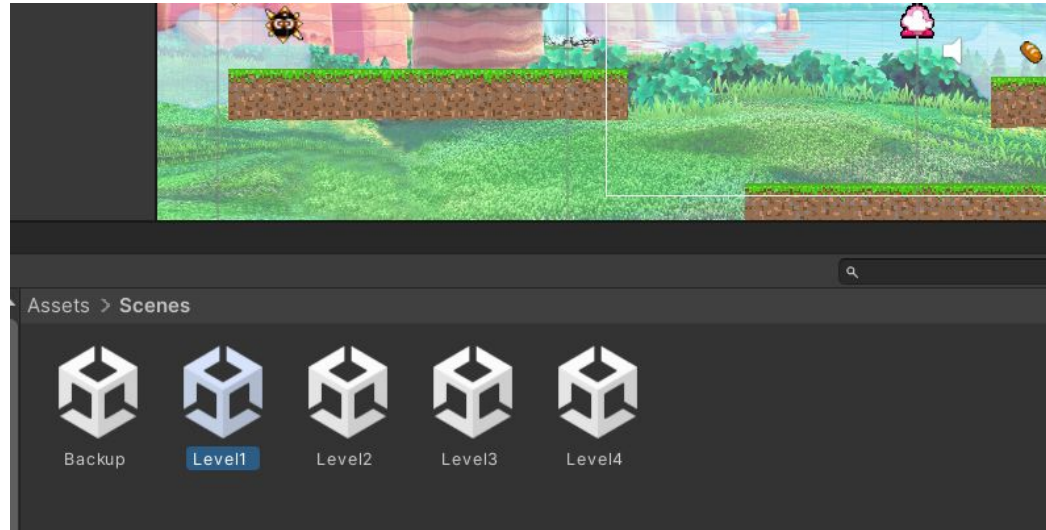
*Level design **creates the experience** for the player.*





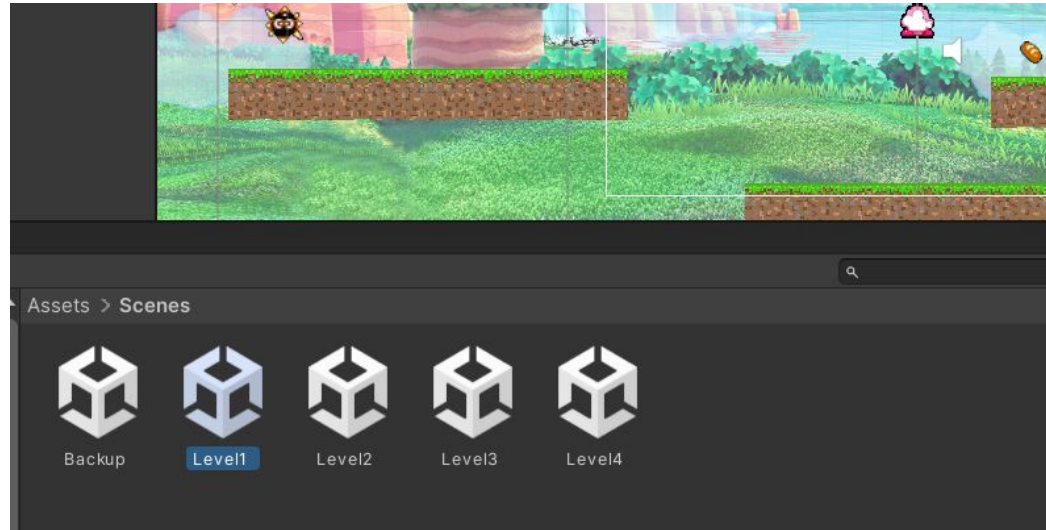
# *Building levels*

- Make the scenes you wish to create



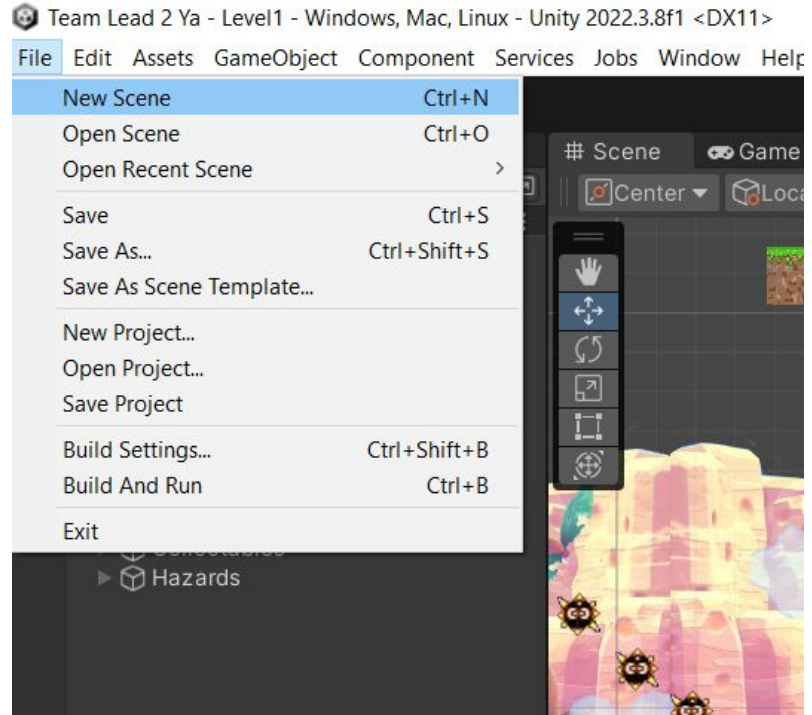
# *Building levels*

- Open the scenes in Unity (crucial)



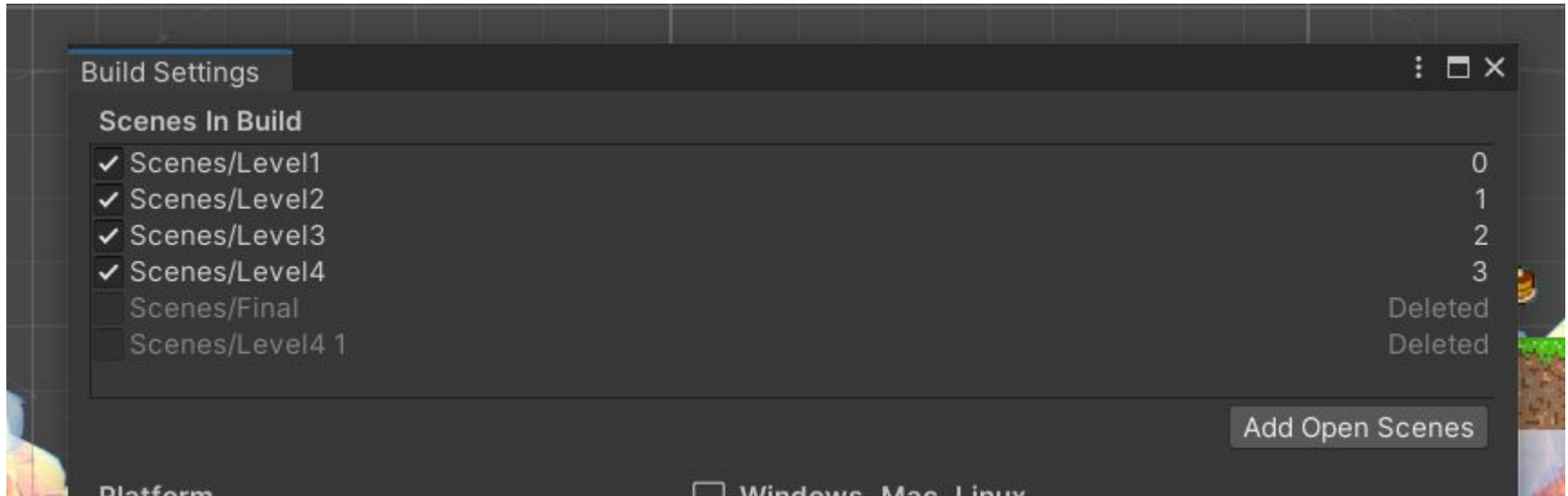
# *Building levels*

- Open Build Settings from File > Build Settings



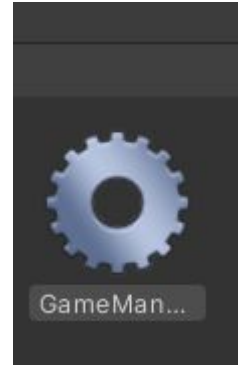
# *Building levels*

- If the scenes are open, should see their names in the Build Settings.
- Checkmark them and click “Add Open Scenes”



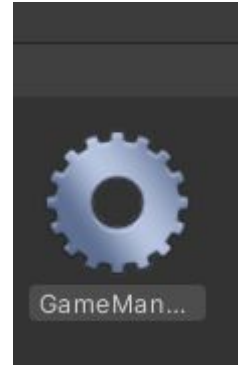
# CODE

- Begin with Game Manager
- Enter its script



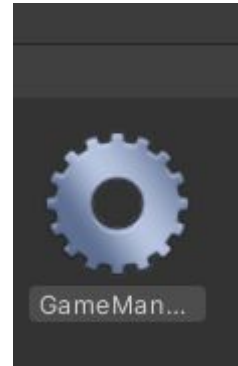
# CODE

- Add method `using UnityEngine.SceneManagement`
  - Scene Management is a library used by Unity
  - This will allow you to create your scenes



# CODE

- Add function to move onto next level
- Trigger
- Debug.Log



# Platforms

- Grab your sprites
- Insert them into game
  - Box collider 2D
  - Rigidbody 2D





# Platforms

- Rotate
- Scale
- Multiply
  - Drag & drop under main one
  - Symmetry



Design: End goal for user



What do you want them to have received as a reward/experience for finishing the game?

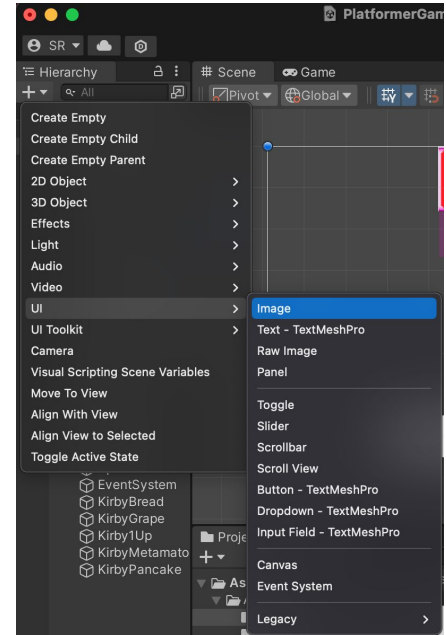
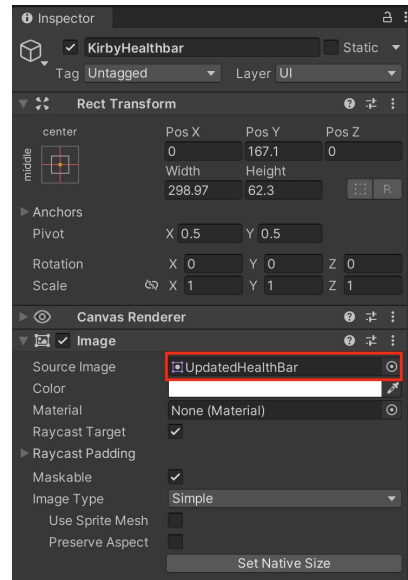
# Psychology of Building Levels

- Flow
- Dopamine
- Alternating challenge with ease

# UI and Collectables

# The Health Bar - Base Asset

1. Find or create the asset you would like to use and place it in your Assets folder.
2. Create an Image UI element in your scene: 
3. Drag and drop your bar asset into the "Source Image" section of the Image component: 
4. Create another Image in your scene and color it red (Name: HBFilled).
  - a. You do not need to put a source image in this Image.
5. Line this image up with the blank space of the base health bar asset.



# The Health Bar - Points/Lives Text

1. First, you will need to find and download the fonts you want.
2. Once you have the .ttf files you would like to use, drag them into your Assets folder.
3. Change the Points Text font to any fonts in your Assets and change the text to simply “0”.
4. Copy this text, paste it and rename the copy to “Lives”.
5. Line these text boxes up in your health bar and change the font size to fit the empty space.
6. Make sure each asset is part of the Canvas in your hierarchy.

# The Health Bar - Coding the Health

1. Open the GameManager script and add the following public variables:
  - a. **public Text LivesText;**
  - b. **public GameObject HBFilled;**
  - c. **public float width = 1f;**
  - d. **public float fullwidth 1f;**
  - e. **public int lives = 0;**
  
2. In the DamagePlayer() function add:
  - a. **width = width - fullwidth \* 1/Health;**
    - i. This calculates how long the health indicator should be
  
  - b. **HBFilled.transform.localScale = new Vector2(width, 1f);**
    - i. The 1f is the y scale of the HBFilled Asset

# The Health Bar - Coding the Lives/Score Pt.1

1. In the GainScore() function change:
  - a. ScoreModifier to **ScoreModifier = 100;**
  - b. ScoreText to **ScoreText.text = “ “ + Score;**
  
2. Add a new public void function to GameManager called GainLife as followed:
  - a. 

```
public void GainLife() {  
    lives += 1;  
    LivesText.text = “ “ + Lives;  
}
```



# The Health Bar - Coding the Lives/Score Pt.2

1. In the Update() function add:

a. An if statement around the Death trigger as such:

```
if(lives <= 0) {
```

```
}
```

b. Within the new if statement:

```
width = fullwidth;
```

```
HBFilled.transform.localscale = new Vector2(width, 1f);
```

c. An else if statement after the Death trigger as such:

```
else if(lives >= 1) {
```

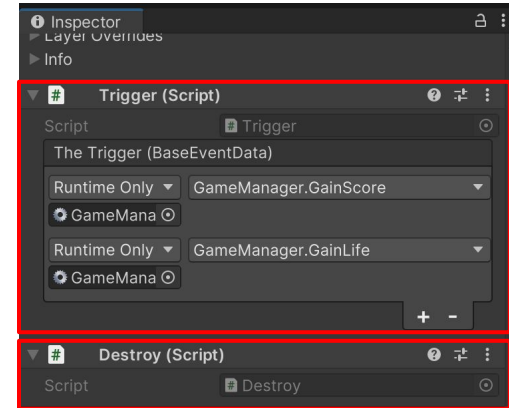
```
lives--;
```

```
LivesText.text = “ “ + lives;
```

```
}
```

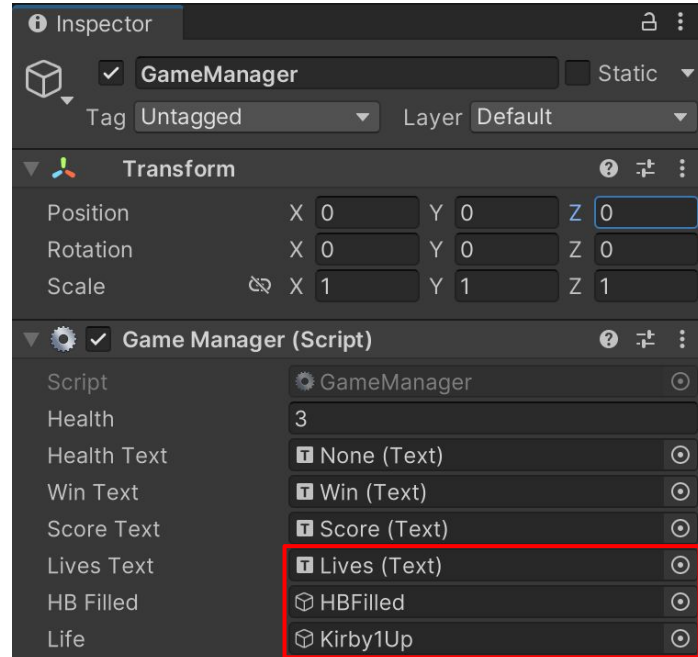
# Collectables - Lives

1. Find or create a life asset and drag it to your Assets folder.
2. Drag this life asset into the scene and rename it. (In this case Kirby1Up).
3. In the Inspector for the asset, add the Box Collider 2D and Rigidbody 2D components.
4. Add a Trigger component and add 2 triggers to it.
  - a. Drag the GameManager script to both of these.
    - i. Make the first function **GameManager.GainScore**.
    - ii. Make the second function **GameManager.GainLife**.
5. Add the Destroy Script as a Script component.



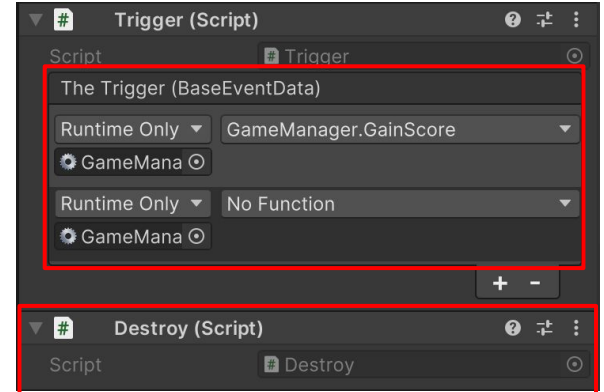
# UI - Tying the Health Bar Together

1. In the GameManager object, go to the Game Manager script component.
2. Tie Lives Text to Lives (Text).
3. Tie HB Filled to HBFilled.
4. Tie Life to Kirby1Up.



# Collectables - Health/Score Pickups

1. Find or create your pickup assets and drag them to your Assets folder.
2. In this instance, drag all the pickup assets into the scene and rename them.
3. In the Inspector for each asset, add the Box Collider 2D and Rigidbody 2D components.
4. Add a Trigger component to each and add 2 triggers to it.
  - a. Drag the GameManager script to both of these.
    - i. Make the first function **GameManager.GainScore**.
    - ii. Leave the second function empty for now.
5. Add the Destroy Script as a Script component to each.



# UI - Simple Inventory

1. Drag a new instance of each pickup to the scene and resize them to be slightly larger than the interactable pickups.
2. Rename these pickups and move them to the Canvas in hierarchy.
  - a. CollectedBread, CollectedGrape, CollectedMetamato and CollectedPancake are the names I used.
3. Move these sprites under the Health Bar in the scene and place an empty Image object behind them.
  - a. Color this image to whatever you'd like and decrease the A value to half.
4. Create a new Legacy Text Ui element, change the text to "Snacks:" for this game and change the font to your font of choice (In this case Lo.Sumires-2X 8l).
5. Move this Text to the Canvas in line with the collected items.

# Collectables - Coding the Pickups

1. Open the GameManager script and add the following public variables:
  - a. **public GameObject CollectedBread;**
  - b. **public GameObject CollectedPancake;**
  - c. **public GameObject CollectedMetamato;**
  - d. **public GameObject CollectedGrape;**
  
2. In the DamagePlayer() function and the if statement of Update() add:
  - a. **CollectedBread.SetActive(false);**
  - b. **CollectedPancake.SetActive(false);**
  - c. **CollectedMetamato.SetActive(false);**
  - d. **CollectedGrape.SetActive(false);**

This code sets the inventory elements to not hide on screen when the game starts.

# Collectables - Coding the Pickups

1. Create 4 new functions in the GameManager script following this outline:

- a. 

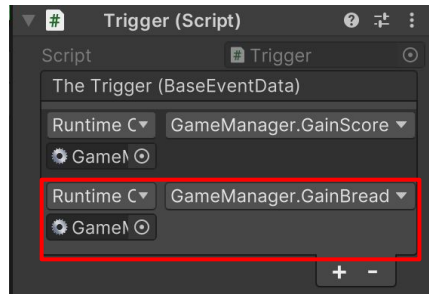
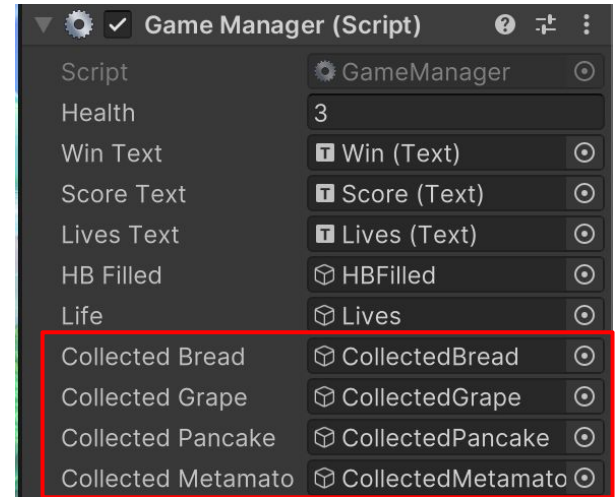
```
public void Gain(Item) {  
    Collected(Item).SetActive(true);  
    if(CurrentHealth != Health) {  
        CurrentHealth++;  
        width = width + fullWidth * 1/Health;  
        HBFilled.transform.localScale = new Vector2(width, 1f);  
    }  
}
```

```
//When the player picks up the bread, sets the bread active in the hud.  
public void GainBread(){  
    CollectedBread.SetActive(true);  
    if(CurrentHealth != Health)  
    {  
        CurrentHealth++;  
        width = width + fullWidth * 1/Health;  
        HBFilled.transform.localScale = new Vector2(width, 0.93f);  
    }  
}
```

```
//When the player picks up the grape, sets the grape active in the hud.  
public void GainGrape(){  
    CollectedGrape.SetActive(true);  
    if(CurrentHealth != Health)  
    {  
        CurrentHealth++;  
        width = width + fullWidth * 1/Health;  
        HBFilled.transform.localScale = new Vector2(width, 0.93f);  
    }  
}
```

# UI - Tying the Collectables and UI Together

1. In the GameManager object, go to the Game Manager script component.
  - a. Tie Collected Bread to CollectedBread.
  - b. Tie Collected Grape to CollectedGrape.
  - c. Tie Collected Pancake to CollectedPancake.
  - d. Tie Collected Metamato to CollectedMetamato.
2. In the Trigger component for each pickup, add the Gain(Item) function to the second trigger where (Item) corresponds to the item picked up.





# Background & Sounds

# Sorting Layers

- Tells Unity what order to render objects
- Useful for placing objects behind other objects (like with backgrounds)
- If you change your sorting layers, be sure to include the “ProjectSettings” folder when you push to git (located alongside the Assets folder in your Unity Project folder)



Top right of the screen you'll find the layers tab

Use the “+” button to add new layers. Layers are rendered in the order they're listed. (Lower number = more stuff on top)



# Creating a Background

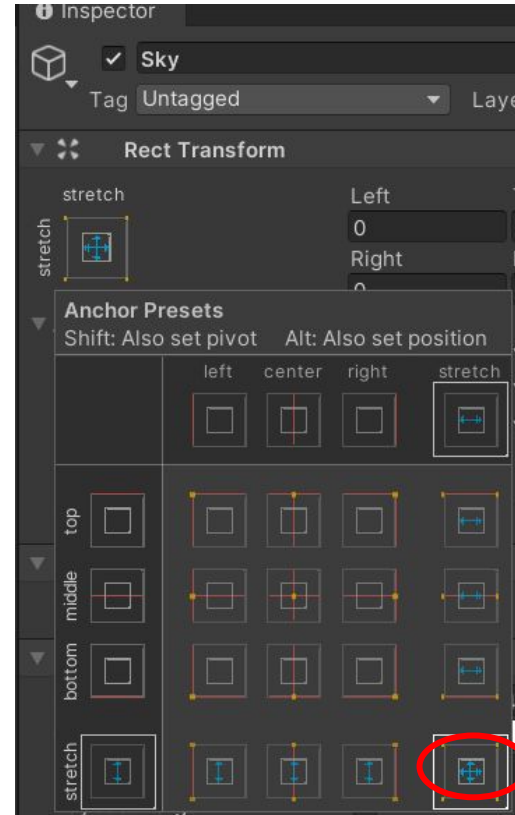
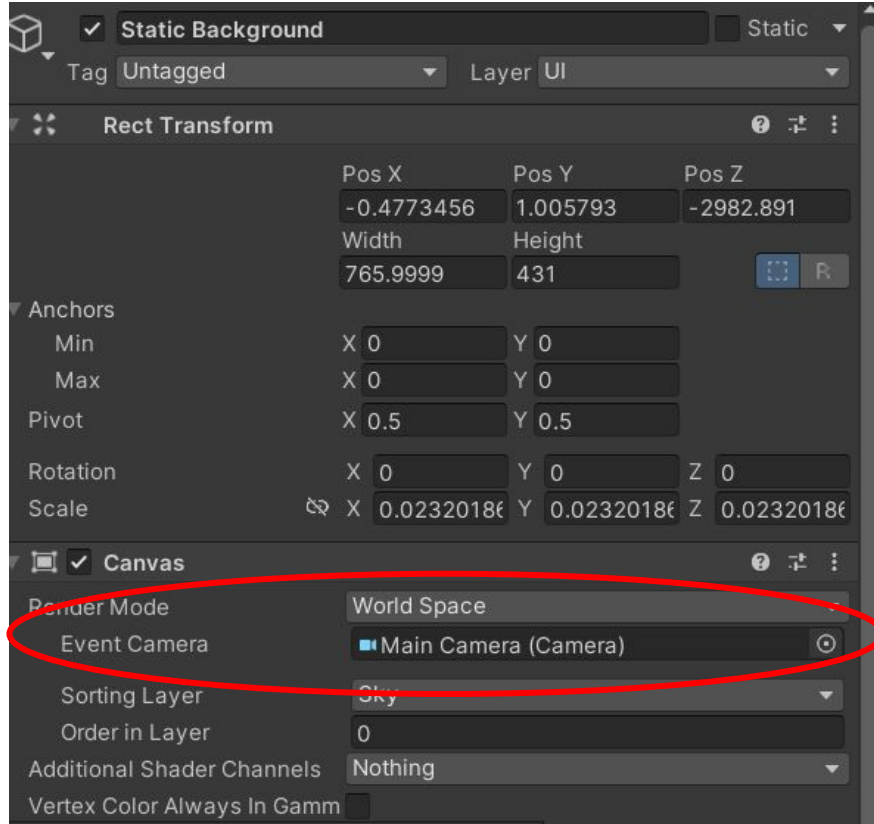
There are 3 types of Backgrounds in a 2D game

- Static (follows the camera, so it does not move relative to the player)
- Dynamic (Image doesn't move, so to the player it looks like they're walking past it)
- Parallax (moves, but independent of the camera)

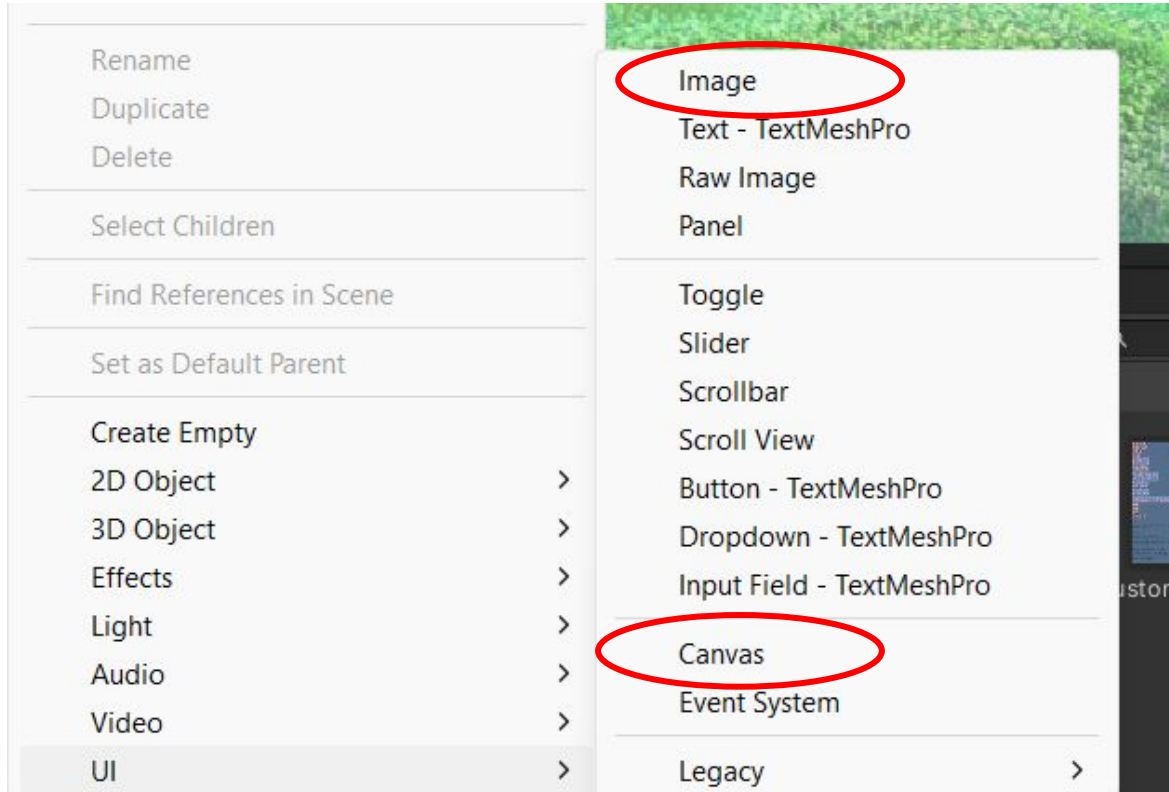
# Static Background Steps

1. Create an empty object named "background."
2. Create a child under that object that's a UI canvas.
3. Under that canvas UI, create a child image object.
4. Select "Screen Space" under canvas render mode, and set the **main camera** as the event camera.
5. Go to that image child, go to anchor settings, and select stretch (bottom right option).
6. Set the "Up," "Right," "Left," and "Down" values to 0. This will fit the image to the camera's shape.
7. Set the image of what you want as the background that follows the player.
8. Refer to the "Background Control Script" for the code to have the image follow along with the camera.

# Static Background Screenshots



# Static Background Screenshots Con.



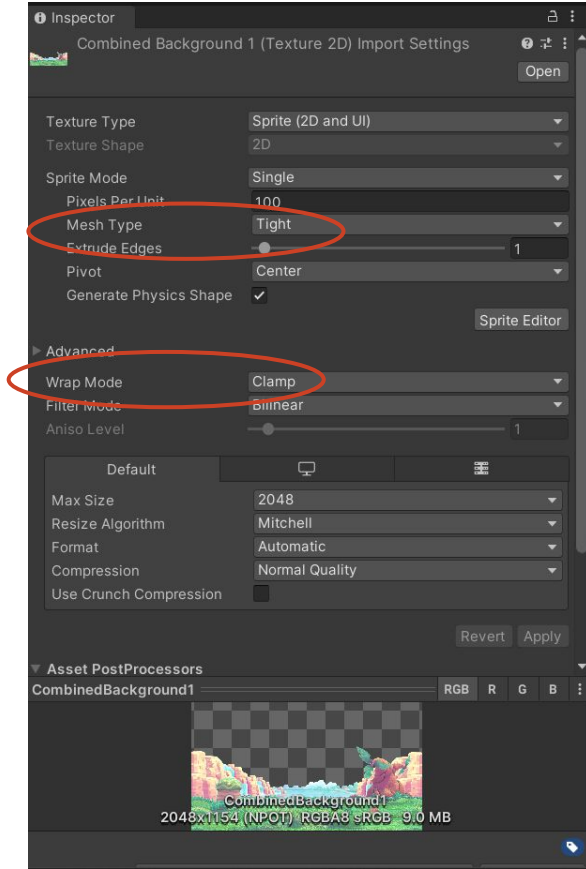
(Right click on Hierarchy tab to pull up this menu)

# Dynamic Background Steps

- 1.) Click on the image in unit and change mess type to “full react” and wrap mode to “repeat”
- 2.) Drag image onto workspace. Look over at inspector and change draw mode to “tiled” and multiply the width by like 3x (or more if you want a longer image)
- 3.) Mess with scale and transform to place it where you want

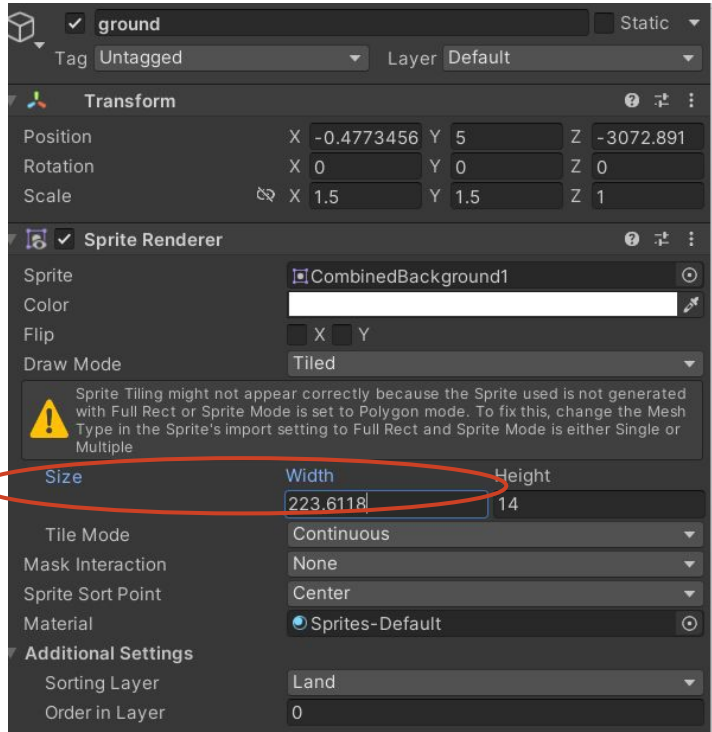
# Dynamic Background Screenshots

When you click on a image that's in the project tab (bottom left, has all the files you're using)  
You'll see these options





# Dynamic Background Screenshots Con.



Clicking on the image after you've dragged it into the project will bring up these options in the inspector view

# Parallax Scrolling

- Useful youtube video:  
[https://www.youtube.com/watch?v=W9aVuOsc\\_k0&ab\\_channel=BluefeverSoftware](https://www.youtube.com/watch?v=W9aVuOsc_k0&ab_channel=BluefeverSoftware)
- Same steps as with Dynamic Background, but this time you'll need to add a script for the image to move on it's own. (script shown in video)

# Music and SFX

- 1.) Create a “sounds folder” in your Assets folder
- 2.) Import your sounds (mp3 or wav). Separate by music and sfx folders in sounds
- 3.) In Unity, create empty object “Audio Manager”
- 4.) Create and assign the tag “Audio” to audio manager
- 5.) Under Audio Manager, create two children of Audio Manager, “SFX” “Music”

See Audio Manager script for specifics on the script

Youtube video:

[https://www.youtube.com/watch?v=N8whM1GjH4w&list=WL&index=8&t=351s&ab\\_channel=RehopeGames](https://www.youtube.com/watch?v=N8whM1GjH4w&list=WL&index=8&t=351s&ab_channel=RehopeGames)

# Useful tips for music

To call sound effects from a different script:

**//declare this as a variable**

```
AudioManager audioManager;
```

**// Add this function**

```
private void Awake()  
{  
    audioManager = GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();  
}
```

**//Anytime you want to play a sound effect**

```
audioManager.PlaySFX(audioManager.soundEffectName);
```

# Background and Audio Manager script links

[https://drive.google.com/drive/folders/13p9a1GiLJnrSI8Ar78mQIPVCbcppTzI?usp=drive\\_link](https://drive.google.com/drive/folders/13p9a1GiLJnrSI8Ar78mQIPVCbcppTzI?usp=drive_link)

# Resources

# Links

- <https://github.com/TheOneTrueGibby/T2-Platform/tree/main>

Sprites (Kirby and Waddle Dee)

- <https://www.deviantart.com/nintendosteven/art/Basic-Kirby-Sprite-Sheet-updated-284952708>
- [https://www.spritters-resource.com/ds\\_dsi/kirbymassattack/sheet/40847/](https://www.spritters-resource.com/ds_dsi/kirbymassattack/sheet/40847/)
- [https://docs.google.com/document/d/168GU8KY3fULm3l8JuG\\_ez-2gBtwb6Srua6jzEuE-o/edit?pli=1](https://docs.google.com/document/d/168GU8KY3fULm3l8JuG_ez-2gBtwb6Srua6jzEuE-o/edit?pli=1) (player and enemy sprites we used)