Work Study of Winter Semester of 2024 Documentation

Sunu Pious Joseph - 100555297

The main purpose of this project is to get the frames from a video that will be turned into animations for our character within the game. Students will record themselves acting out these animations and put themselves into the game.
The animations we recorded for each student:

- Jump
- Idle
- Run
- Death
- Spawn

2D Platformer Template taken from Unity's tutorial templates available on their site:
https://learn.unity.com/project/2d-platformer-template

The Unity template allows us to get the basic functionality we need for our 2D platformer and easily apply any of our assets. The downside of this template is the template as far back as 2017 or even further than that since version numbers listed on the site are unclear after 2017. The template may have some strange coding practices and old functionality that may cause errors when porting to a more modern version of Unity.

Creating the Custom Animator
We need to disable the current animator on our current character and create a custom animator that takes in a list of frames that corresponds with an animation. Thus, to have multiple characters with this functionality. We need to create a folder structure where each folder is a Character Folder containing Animation Folders that correspond to the animation and each animation folder contains our frames. The next functionality of the Custom Animator is getting the lists of animations and playing them within the game. There are multiple ways this could be done, but for this scenario I've used Coroutines to manage loading the frames being played and while having the ability to interrupt each animation.

Character Folder Structure Example:

- Character1 (Character Folder)
    - Jump (Animation Folder)
        - Frames (PNG)
    - Idle (Animation Folder)
        - Frames (PNG)
    - Run (Animation Folder)
        - Frames (PNG)
    - Death (Animation Folder)
        - Frames (PNG)
    - Spawn (Animation Folder)
        - Frames (PNG)

Incorporating WebGL Functionality
In order for the game to be playable itch.io we need the game to work on WebGL. The issue with this is that WebGL doesn’t have any access to any file reading to the local directory structure we normally have. Since our current game has to read files within the directory to create our character animation, we need to slightly overhaul the custom character animator scripts. A solution I found is creating Scriptable Objects of our Characters, this allows us to have the characters already exist within the scene rather than have the game load the characters on startup. This solves the problem with WebGL unable to read the characters from the local directory since the game scene already has them. These Character Scriptable Objects or CharacterSOs will have the frame data before the game is launched and we can have them loaded into the Character Select Scene on game launch.

Final Builds Results:
https://gdimcommunityactivities.itch.io/put-yourself-in-the-game-2d-gdim-activity-feb-21
https://gdimcommunityactivities.itch.io/event-put-yourself-in-the-game-thurs-feb-22-2024

Custom Player Animator

Firstly I'll briefly talk about how each script works and how to add these scripts to the scene to work. There are 2 main Scripts on how we create our Custom Character Animator and 4 other Scripts for loading said animations.

<u>Character Animation</u>

- CustomPlayerAnimator.cs
- FileManagerUpdate.cs

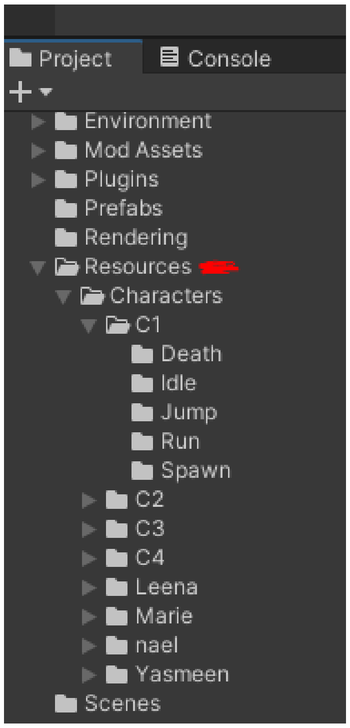
<u>Creating Characters from Frames</u>

- CharacterSO.cs
- CharacterTracker.cs
- CharacterLoader.cs
- CharacterSelectButton.cs

Creating our Characters

CharacterSO.cs
This is a Scriptable Object that's essential for the custom animator to work on any WebGL platform. We create scriptable character objects that contain all the frames and animations we need. The naming of the folders is important here, so name the characters unique from one another and make sure the Animation Names are consistent.

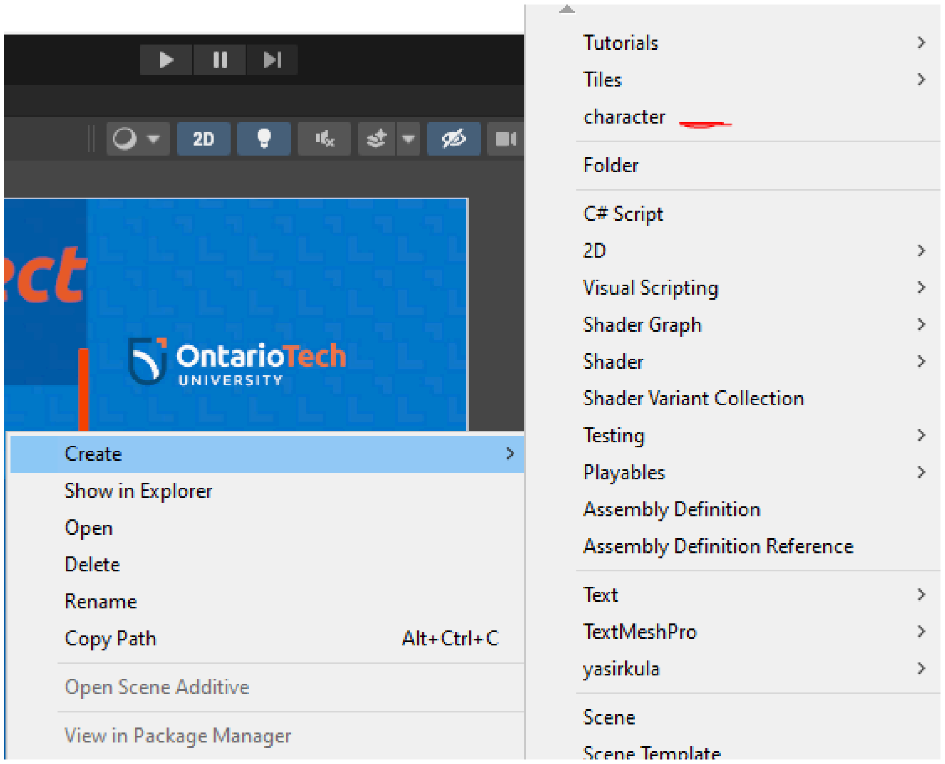
First we put all our Characters within the Resources Folder and in the Characters Folder. Each Character Named Folder, we have our Animation Folders containing the frames of said Animation. Following the example folder Structure below:



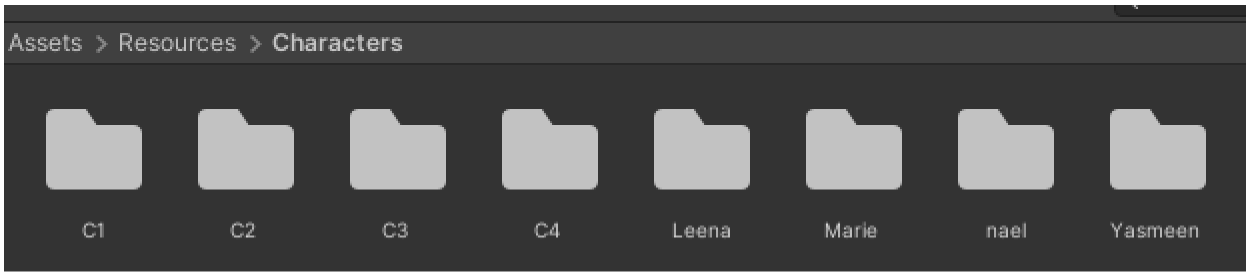
Character Folder Structure Example:

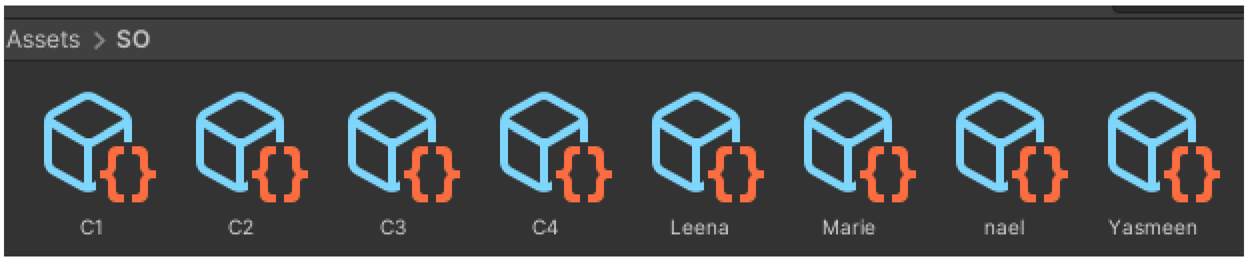
- Character1 (Character Folder)
    - Jump (Animation Folder)
        - Frames (PNG)
    - Idle (Animation Folder)
        - Frames (PNG)
    - Run (Animation Folder)
        - Frames (PNG)
    - Death (Animation Folder)
        - Frames (PNG)
    - Spawn (Animation Folder)
        - Frames (PNG)

Next create a New Folder within our project assets to contain all our Character Scriptable Objects. Within the empty folder, **right click -> "Create" -> "character"**. As shown below:

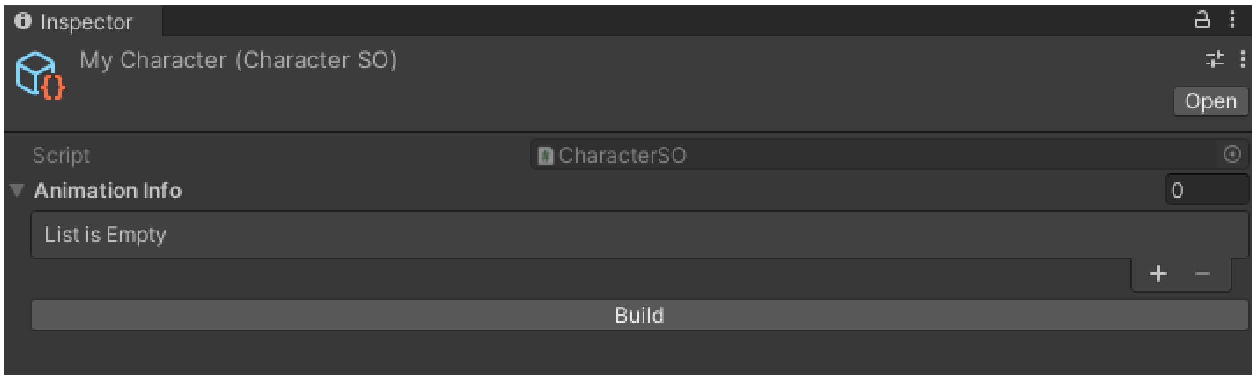


This would create a new Character Scriptable Object or CharacterSO. Name it the same as our Character Folder Names as shown here: This is how Unity will find our Characters in Resources and create our CharacterSOs.

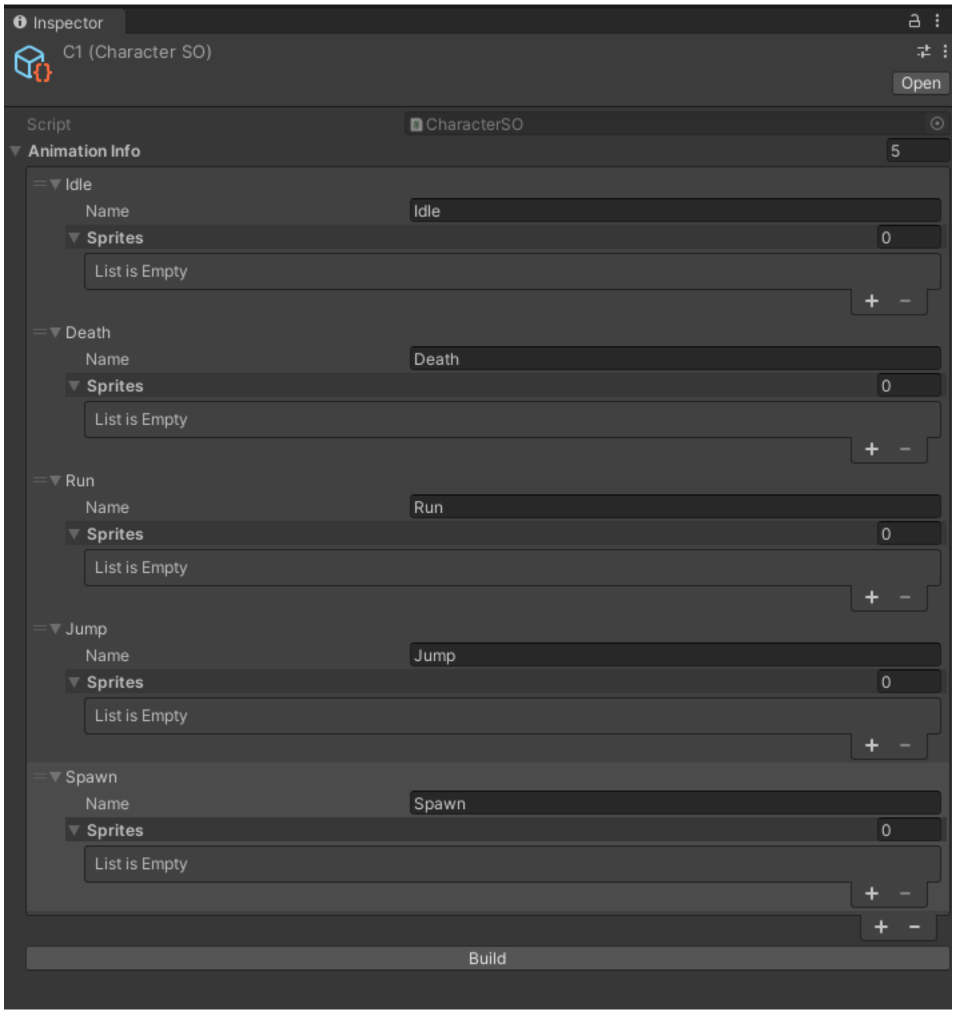


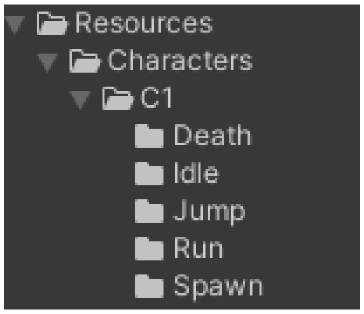


Select one of our CharacterSOs and look at what we have in the inspector. It would look something like this initially.


Inspector
My Character (Character SO)
Open
Script CharacterSO
Animation Info 0
List is Empty
Build

We need 5 animations for this project, so we will add 5 Animation Infos and name them all the same as our Character Animation Folders for every character. Remember they must have the same names as the folder names and it is case sensitive. It should look like this:


Inspector
C1 (Character SO)
Open
Script CharacterSO
Animation Info 5
Idle
Name Idle
Sprites 0
List is Empty
Death
Name Death
Sprites 0
List is Empty
Run
Name Run
Sprites 0
List is Empty
Jump
Name Jump
Sprites 0
List is Empty
Spawn
Name Spawn
Sprites 0
List is Empty
Build

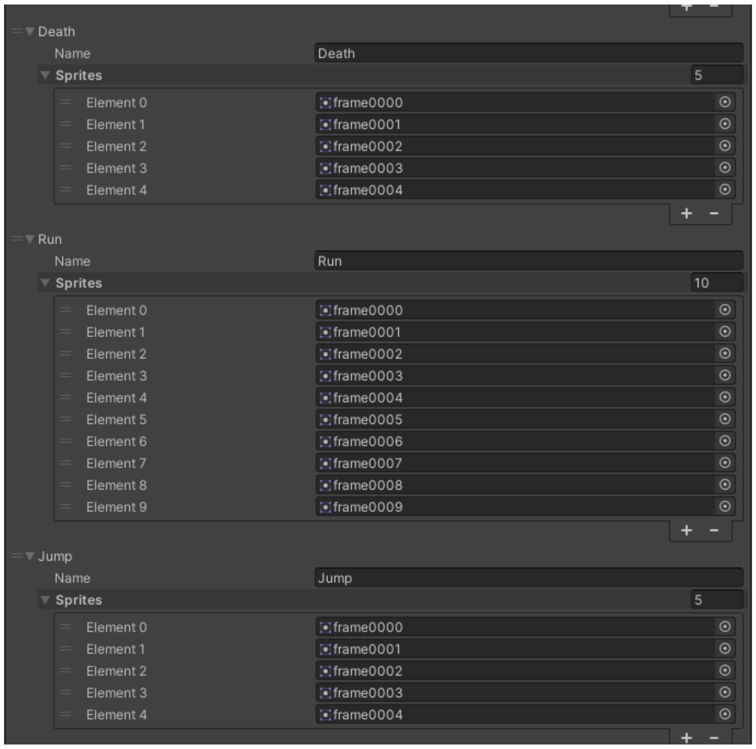

Resources
Characters
C1
Death
Idle
Jump
Run
Spawn

See how the Character Name and the Character Animation Names match.

This allows Unity to find our Character and its Animations.

Here I recommended repeating this till we have all our characters.

Once you're done creating each character and naming them appropriately. We click the Build button within the inspector. This will take all the frames within the character folder and store them within the appropriate animation. You see frames are loaded within each animation:



```
▼ Death
    Name          Death
  ▼ Sprites                    5
      Element 0   frame0000
      Element 1   frame0001
      Element 2   frame0002
      Element 3   frame0003
      Element 4   frame0004
▼ Run
    Name          Run
  ▼ Sprites                    10
      Element 0   frame0000
      Element 1   frame0001
      Element 2   frame0002
      Element 3   frame0003
      Element 4   frame0004
      Element 5   frame0005
      Element 6   frame0006
      Element 7   frame0007
      Element 8   frame0008
      Element 9   frame0009
▼ Jump
    Name          Jump
  ▼ Sprites                    5
      Element 0   frame0000
      Element 1   frame0001
      Element 2   frame0002
      Element 3   frame0003
      Element 4   frame0004
```

Repeat this step and Build every character and have their animation frames loaded within each CharacterSO.

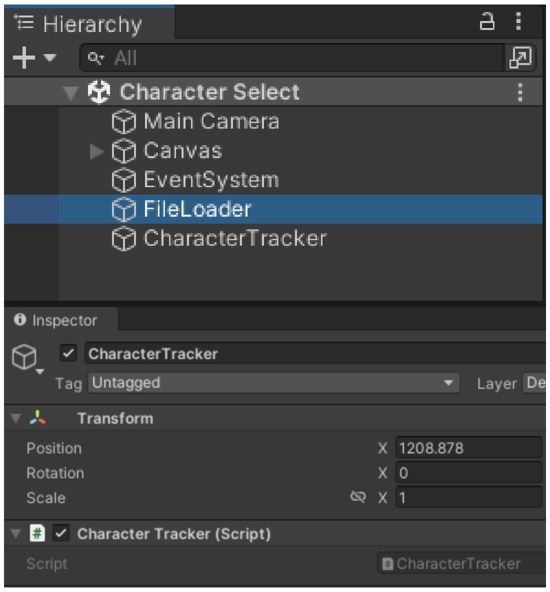
With this the CharacterSOs are ready to be used, no more loading animations from file directly since we have the CharacterSOs already existing within the Scene.

Loading Characters into Scene

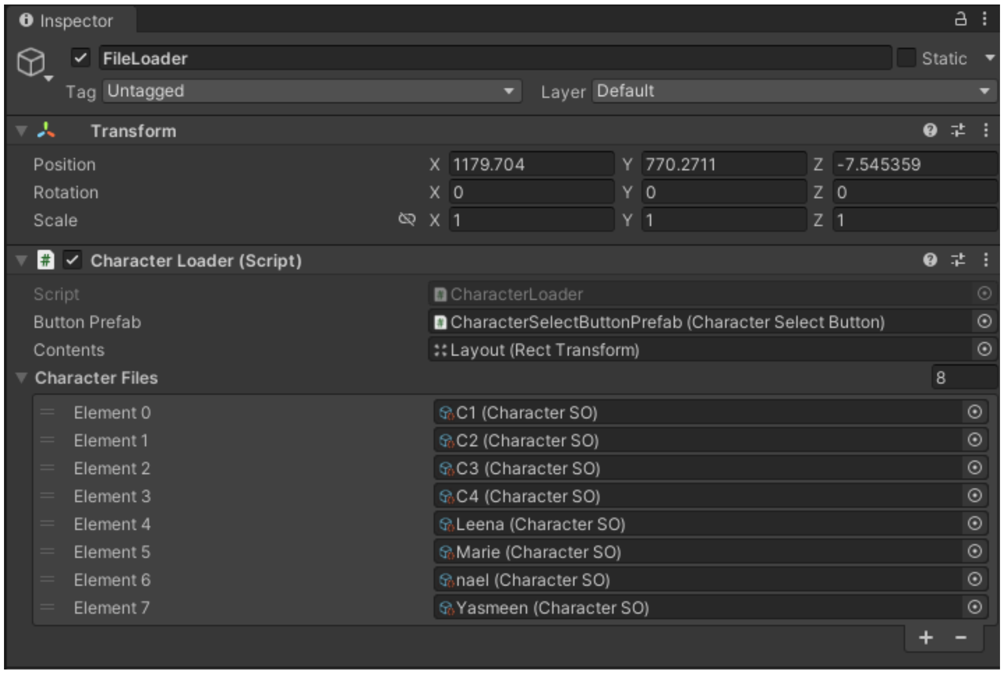
In our Hierarchy, we will create 2 Empty Game Objects;
FileLoader holding CharacterLoader.cs
CharacterTracker holding CharacterTracker.cs.

File Loader will take all our CharacterSOs within the Character Files List, allowing us to have them exist in the scene without any file reading.

Character Tracker is an instance object that exists through all Scenes. It holds Character Data of the selected character the player chooses within the Character Screen Selection.



CharacterTracker
Tag Untagged Layer De
Position X 1208.878
Rotation X 0
Scale X 1
Script CharacterTracker



FileLoader Static
Tag Untagged Layer Default

	X	Y	Z
Position	1179.704	770.2711	-7.545359
Rotation	0	0	0
Scale	1	1	1

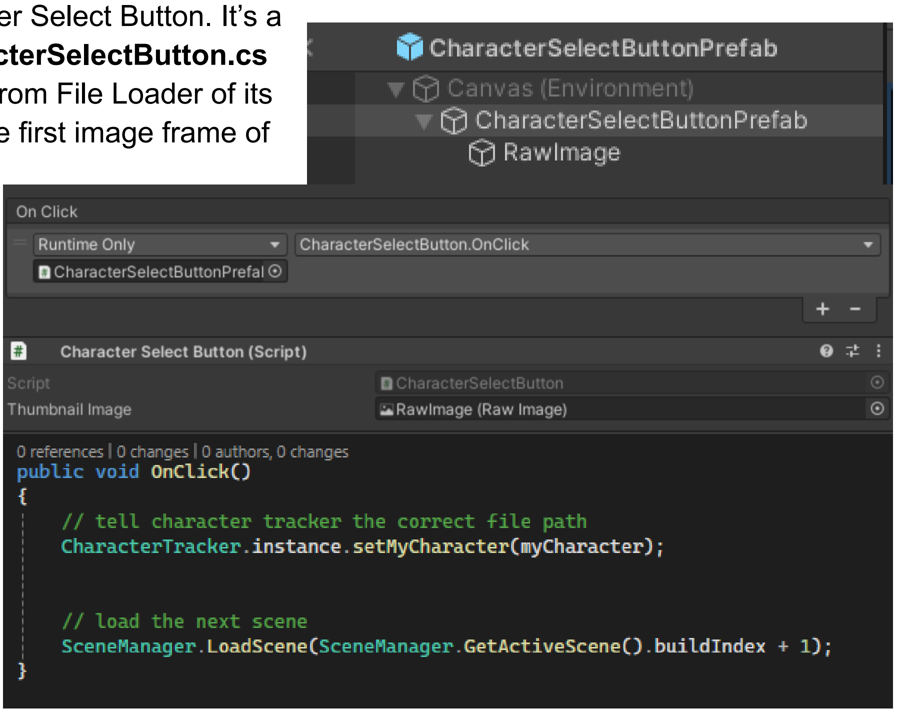
Script	CharacterLoader
Button Prefab	CharacterSelectButtonPrefab (Character Select Button)
Contents	Layout (Rect Transform)
Character Files	8
Element 0	C1 (Character SO)
Element 1	C2 (Character SO)
Element 2	C3 (Character SO)
Element 3	C4 (Character SO)
Element 4	Leena (Character SO)
Element 5	Marie (Character SO)
Element 6	nael (Character SO)
Element 7	Yasmeen (Character SO)

File Loader also loads and creates Character Select Buttons. Associating them with CharacterSOs and loading them within a Layout in the Canvas. It will create as many Character Select buttons as there are CharacterSOs.

A quick run down on our Character Select Button. It's a simple square button with **CharacterSelectButton.cs** Script attached to it. It gets data from File Loader of its relevant CharacterSOs, taking the first image frame of Idle Animation as its thumbnail for the button as a RawImage.

We then attach the OnClick() function within Script.This stores our CharacterSO data within our Character Tracker and then loads the next Scene.

This function is called when a player selects a Character within the Character Selection.



On Click

Runtime Only | CharacterSelectButton.OnClick

CharacterSelectButtonPrefal

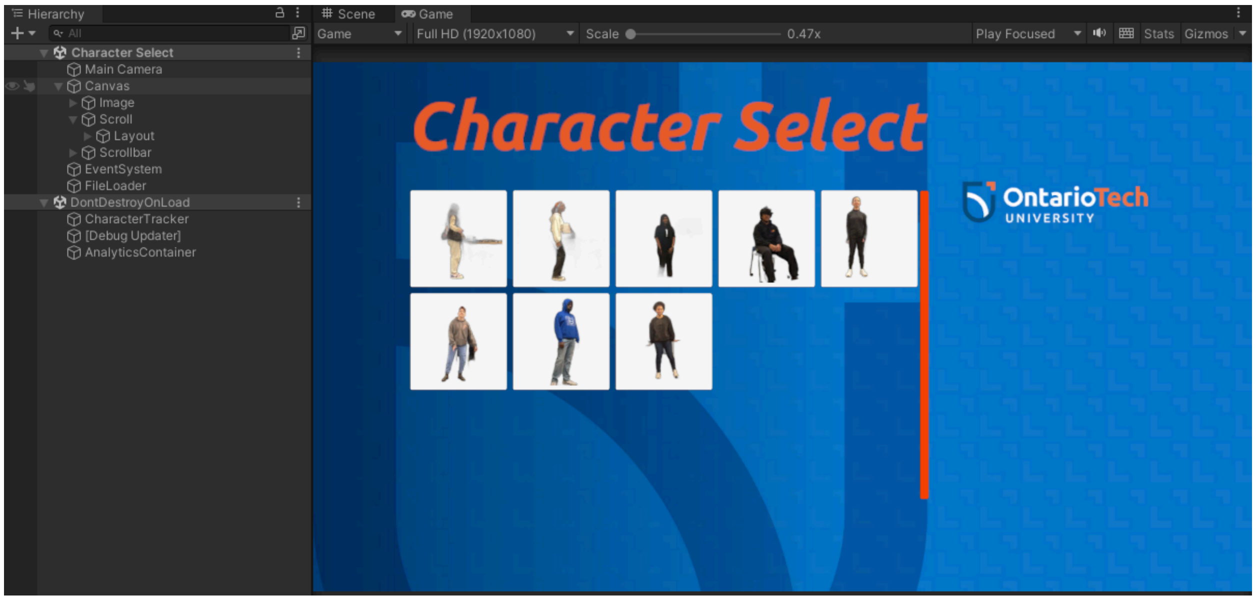
Character Select Button (Script)

Script: CharacterSelectButton

Thumbnail Image: RawImage (Raw Image)

0 references | 0 changes | 0 authors, 0 changes
```
public void OnClick()
{
    // tell character tracker the correct file path
    CharacterTracker.instance.setMyCharacter(myCharacter);


    // load the next scene
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
}
```



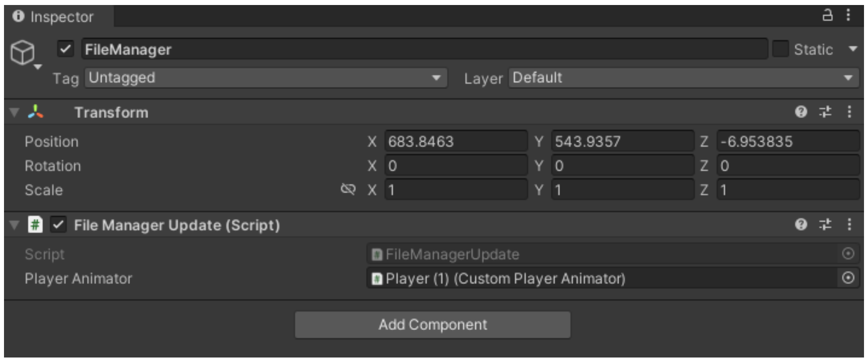
When we click play, our CharacterSOs are loaded and associated with a Character Select Button within our Canvas Layout, thanks to our File Loader. Character Tracker is currently waiting for the player to select a character for it to track and send to the next scene.

Player Character Animation

There are 2 Scripts are mainly used for actually animating the player; **CustomPlayerAnimator.cs** and **FileManagerUpdate.cs**

FileManagerUpdate.cs saves the animations from CharacterTracker's current CharacterSO and sends it to our CustomPlayerAnimator.

FileManagerUpdate.cs will be attached to an Empty Game Object within the Scene. It will take our Player's CustomPlayerAnimator as a parameter.

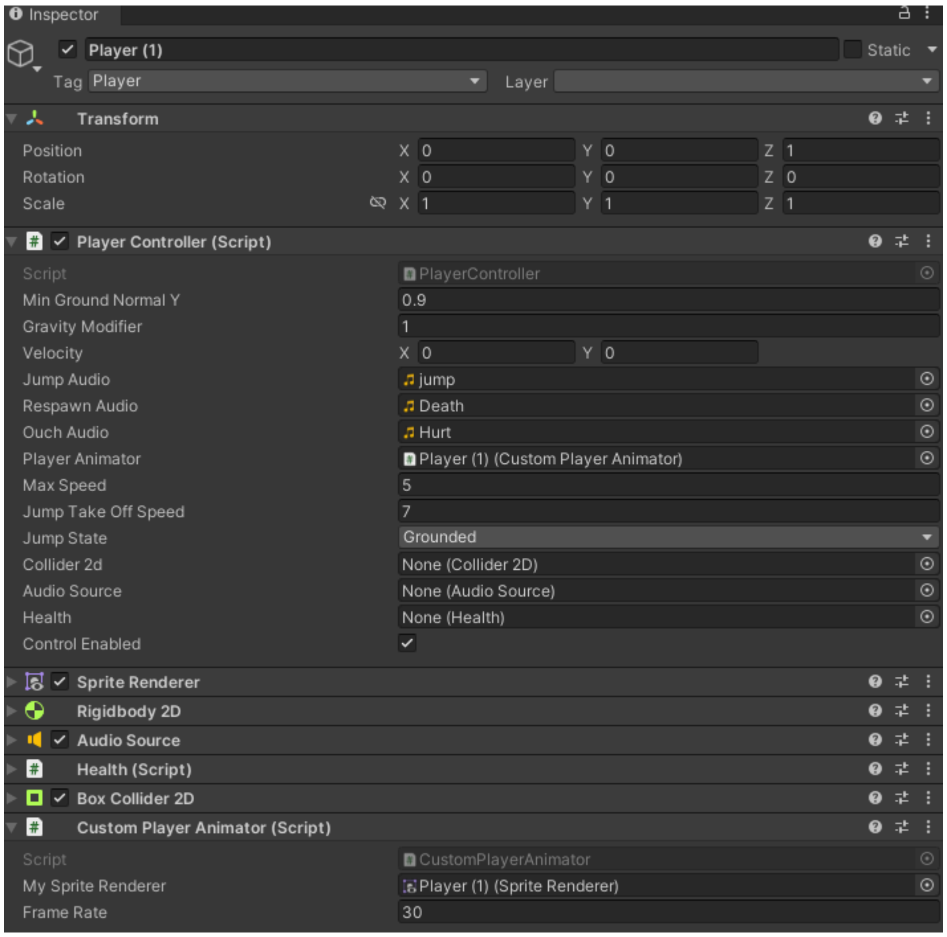


```
FileManager
Tag Untagged    Layer Default
Transform
Position   X 683.8463   Y 543.9357   Z -6.953835
Rotation   X 0          Y 0          Z 0
Scale      X 1          Y 1          Z 1
File Manager Update (Script)
Script            FileManagerUpdate
Player Animator   Player (1) (Custom Player Animator)
```

CustomPlayerAnimator.cs will be attached to our Player Gameobject.

The PlayerController script will have a reference to CustomPlayerAnimator, allowing us to play any animation when calling CustomPlayerAnimator.

CustomPlayerAnimator handles playing different animations and entire animations playing themselves. We only need a reference to the Player's Sprite Renderer and set the Frame Rate we desire for our animations.



```
Player (1)
Tag Player    Layer
Transform
Position   X 0   Y 0   Z 1
Rotation   X 0   Y 0   Z 0
Scale      X 1   Y 1   Z 1
Player Controller (Script)
Script                 PlayerController
Min Ground Normal Y    0.9
Gravity Modifier       1
Velocity               X 0   Y 0
Jump Audio             jump
Respawn Audio          Death
Ouch Audio             Hurt
Player Animator        Player (1) (Custom Player Animator)
Max Speed              5
Jump Take Off Speed    7
Jump State             Grounded
Collider 2d            None (Collider 2D)
Audio Source           None (Audio Source)
Health                 None (Health)
Control Enabled        ✓
Sprite Renderer
Rigidbody 2D
Audio Source
Health (Script)
Box Collider 2D
Custom Player Animator (Script)
Script                 CustomPlayerAnimator
My Sprite Renderer     Player (1) (Sprite Renderer)
Frame Rate             30
```

To play any desired animation, you only need to call a single function from CustomPlayerAnimator called **PlayAnimation**. Where it takes two parameters and the second parameter is an optional parameter if we require it to interrupt an animation currently playing.

PlayAnimation(**String AnimationName, Boolean ShouldInterrupt = false**)
First parameter takes a String of the Name of the Animation we want to play.
Second Parameter takes a Boolean value to tell the CustomPlayerAnimator to interrupt an Animation and play a new one. By default it's considered false.

For Example:

First a Reference to our CustomPlayerAnimator:

public CustomPlayerAnimator playerAnimator;

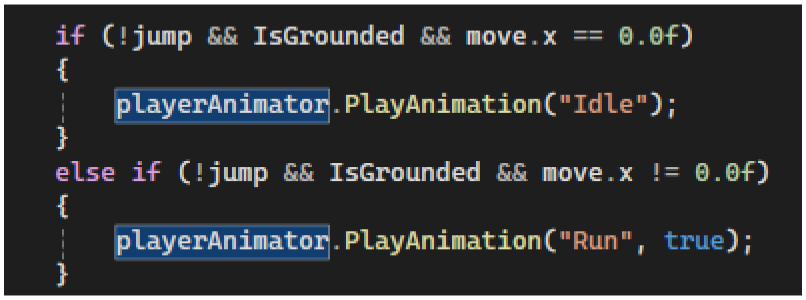
To play an animation

playerAnimator.PlayAnimation("Idle");

To play an animation and interrupt any animation currently playing

playerAnimator.PlayAnimation("Run", true);

Here we have standard Idle animation playing but we may want to interrupt the Idle animation when the Character runs, hence we call the Run Animation to be played and interrupt the currently playing Idle animation.



```
if (!jump && IsGrounded && move.x == 0.0f)
{
    playerAnimator.PlayAnimation("Idle");
}
else if (!jump && IsGrounded && move.x != 0.0f)
{
    playerAnimator.PlayAnimation("Run", true);
}
```

That should cover the entire setup on how the Custom Player Animator works. The github of the entire project should be located here: https://github.com/sunujoseph/2DPlatformerOntarioTechU

Known Bugs

- While playing the game through Editor, sometimes the frames saved on CharacterSOs' on some animations are removed and have to click Build again on the CharacterSO.
- On the 2DPlatformer template, the token animations are sped up randomly but will return normal after some time. Only happens on WebGL and not in Editor.

Known bugs seem to mainly relate to the Unity Engine itself and WebGL. Inconsistent when trying to replicate bugs. Possibility fixed on a newer version of Unity.

Game Ideas

Some free Unity Templates from the asset store:
https://assetstore.unity.com/?category=templates%5C2d&free=true&orderBy=1

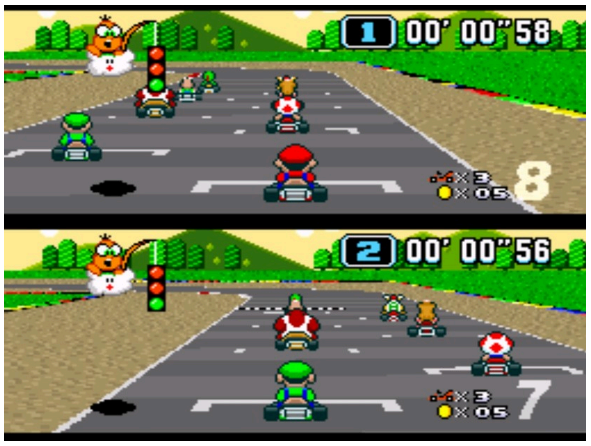
Mortal Kombat 2D fighter
Students can record basic fighting moves within a simple fighting game template.
I think it would be enjoyable for students to come up with their own moves for this fighting game.

Possible simple template to borrow from:
https://www.youtube.com/playlist?list=PL1bPKmY0c-wmOi4Ki-6ryydPeHEcQpul0

Infinite Runner:
We can possibly reuse the 2D platformer code and turn it into an endless runner.
Can use the same animations required that are the same for the 2D platformer.
Probably the easiest way to do another project since we are applying the same rules here.

Possible Templates:
https://www.youtube.com/playlist?list=PLfX6C2dxVyLylMufxTi7DM9Vjlw5bff1c
https://youtu.be/yshKlol5pHM?list=PL6ynPcXXvDY-dMil96IHEI8wvHO3MmQc1

2D/3D Mario Kart



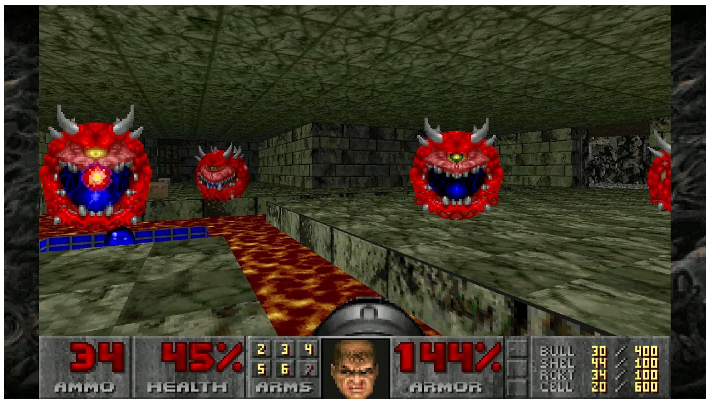
Do a mix of 3D with 2D or full 2D.

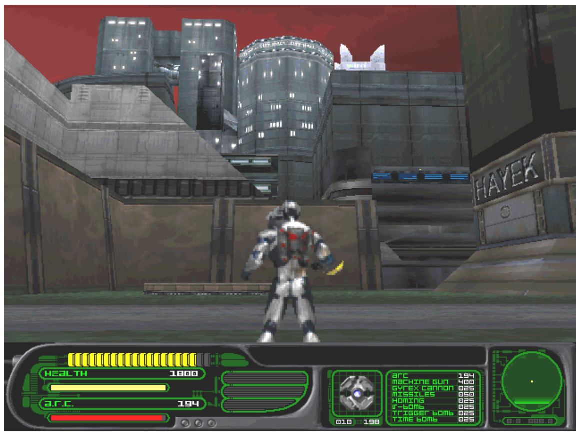
Mario Kart template made from Unity
https://learn.unity.com/project/karting-template
https://learn.unity.com/tutorial/session-3-karting-microgame-part-1-mar-29-2021?uv=2020.2&projectId=604012b3edbc2a002023338c#

Old School Doom style game

For Student engagement:

- Have students have fun pretending to be monsters
- 3rd Person Camera where they pretend to be the player
- A mix of both possibly lets students enjoy their creativity.
- Not sure how to make mobile friendly, compared to rest





Possible Templates:
https://learn.unity.com/project/fps-template
https://www.youtube.com/playlist?list=PLe3aEP7K3NT56BIXMMMLw1Waz3AwN8IPx