

Impori Log — CIE Senior Project

Roan Silver

May 6th 2024 → May 22nd 2024

AT Computer Science & Senior CIE “20% Project”

Git Repository: <https://github.com/rsilver24/ImporiSP>

Important Links and Resources

Log #1 (Semester 1, August-December 2023):

https://docs.google.com/document/d/1Do030o2F4BzBiT_deoaRCjeU_klZSTDkwxZCc2_flj0/edit?usp=sharing

Log #2 (Semester 2, January-May 2024):

https://docs.google.com/document/d/18-BKVnmJ50E0cmidN1Co-FlbyfXwVeD_jNls8HBrjXg/e_dit?usp=sharing

End of Semester 1 Presentation (December 2023):

https://docs.google.com/presentation/d/1CzJB79dGrJpUx4Nxxi8_jwYVwpcZ7Is1PHeZ0poLyvI/edit?usp=sharing

End of Semester 2 Presentation (May 2024):

https://docs.google.com/presentation/d/1LdIPUCz6j9cpcLec0tKdrH3_PYABpk1Q7QGmx6YpmUg/edit?usp=sharing

Initial 20% Proposal (August 2023):

https://docs.google.com/document/d/1yuCyuOO4D0ZAb-_pgp_jNs5j_iDYKeSPOiLT6JbDDMc/edit?usp=sharing

20% Continuation Proposal (January 2024):

https://docs.google.com/presentation/d/1GZuFc5CRPGaPSsjkpK7R_kwvTCLZv0DkNFh0IxJX0-Q/edit?usp=sharing

End of Semester 1 Project Write-up (December 2023):

https://docs.google.com/document/d/19OuSGCpS9Xcgz5GKhTVo_p1CQVh66Dr8bZ2Hs8GUtHc/edit?usp=sharing

End of Semester 2 Project Write-up (May 2024):

https://docs.google.com/document/d/1taaxjUWW8-hyt_X5hjDGenpy5jbJftNEsDGwNyMiB8M/edit?usp=sharing

CIE Proposal & Goals Document (May 2024):

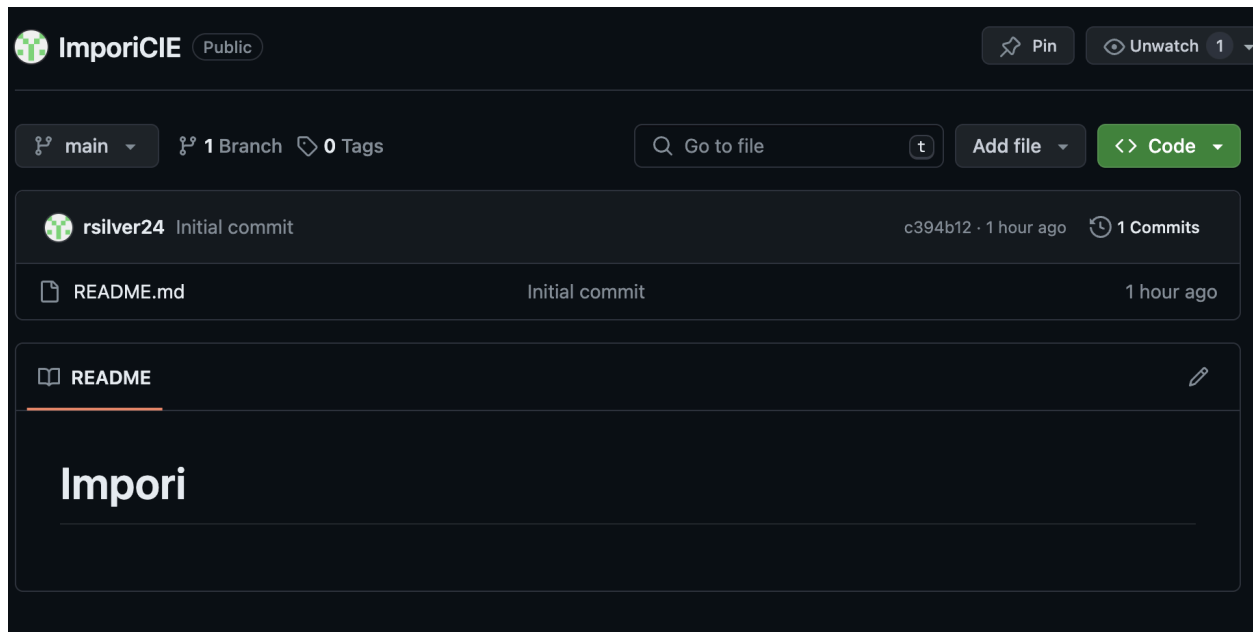
<https://docs.google.com/document/d/1mgrrqCnKD-zTknOf22zUHR1pjT0MVyYn76Uxjbp4oJo/edit?usp=sharing>

May 6th, 2024 — Relocating Files, Forming a Git Repository, and Making the Log Document

Accomplished

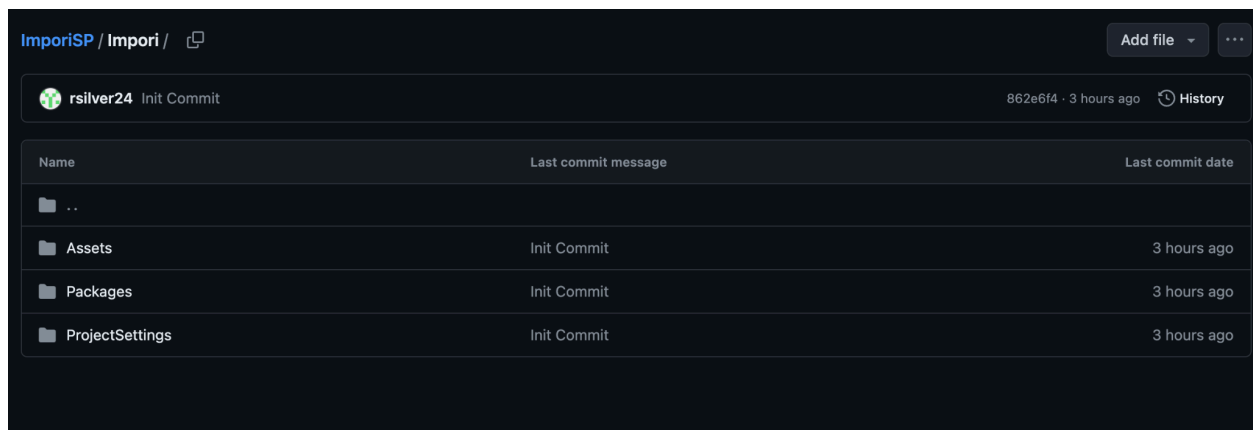
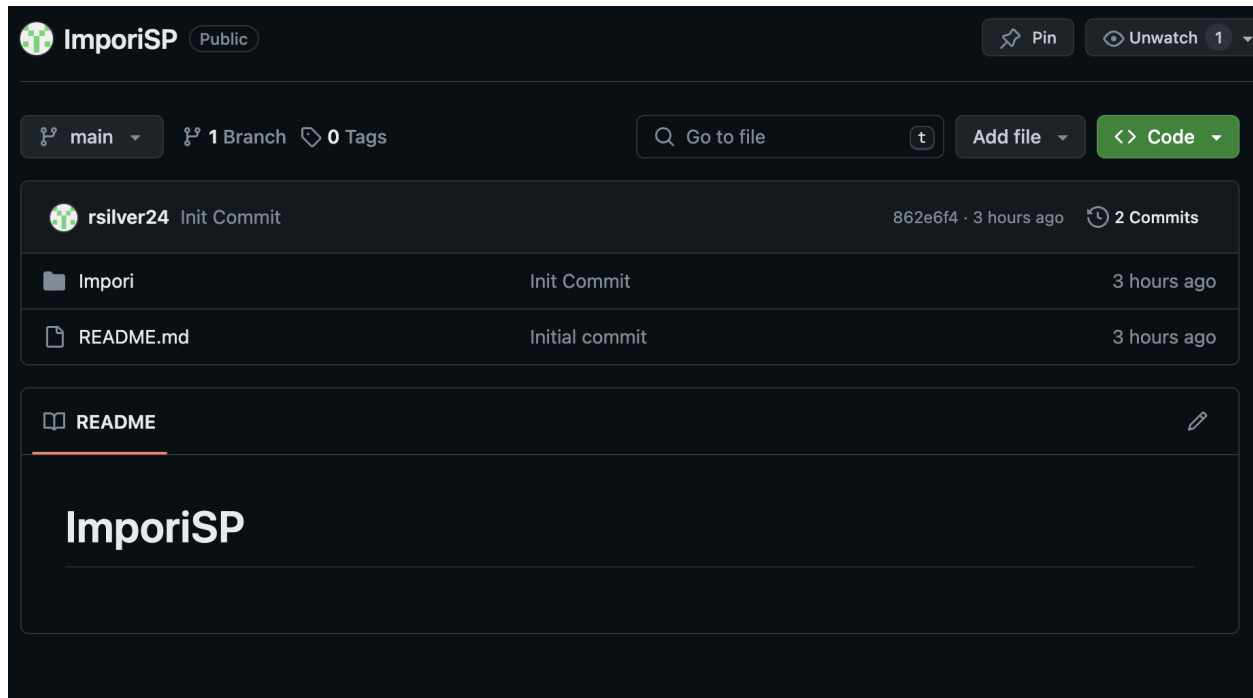
Today was a necessary formation day. I needed to make sure everything was up and running for the periods of work that were to follow. As such, instead of working purely on game-work and in-app-work, I dealt with everything *surrounding* the project. This included 3 main things: Making the Git Repository (*ImporiSP*), relocating files from the previous origin TO this new local repository, and making this document here that all these logs will be written in.

This first step of making the Git Repository took the most stress and effort... and a lot of trial and error. First of all, the current working repo can be found here: <https://github.com/rsilver24/ImporiSP>. This, funny enough, is the SECOND repository I had made. The first one, labeled *ImporiCIE*, was a complete failure.



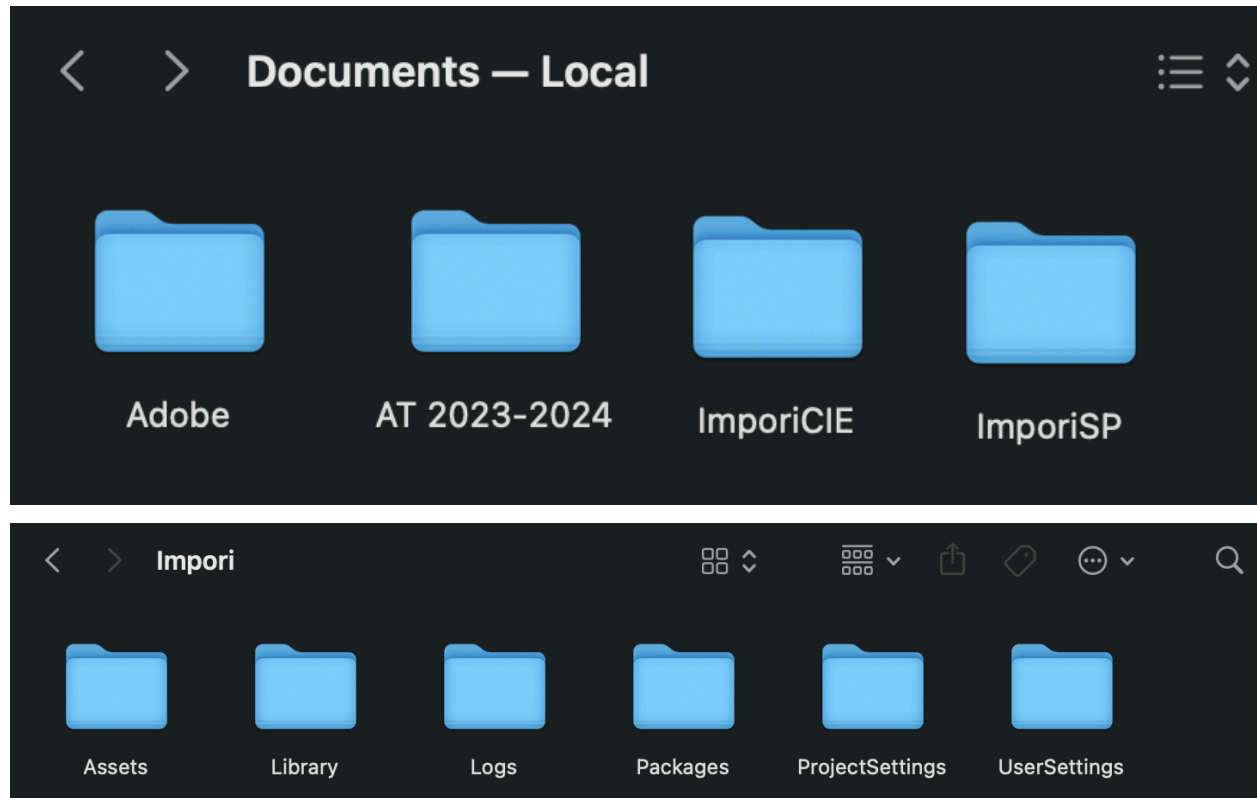
The issue here is that I had attempted to upload **ALL** files that were in relation to Impori. This included 6 main folders: Assets, Library, Logs, Packages, ProjectSettings, and UserSettings. I wanted to upload all of these at first to show my work, but there were a LOT of issues here. First of all, the Library folder, as you could imagine, is HUGE. Uploading all of these files was both unnecessary and far too costly. Second, there were a lot of *types* of files that Git just didn't know what to do with. Although Git

is handy for uploading *most* things, certain file types don't process well when jumbled up and shipped in whole. Third, the files within Logs and UserSettings just plainly weren't necessary to be stored. Yes they help make Unity itself function within the application, but they aren't necessarily unique and required for the functionality of this specific game. So, instead of doing all of that, I chose to focus on JUST three of these folders, being Assets, Packages, and ProjectSettings. I also renamed this repository *ImporiSP*, just in case I managed to delete my entire project somehow.



In order to actually get these files into the git repository however, I needed to of course migrate them into a local repository on my computer. Previously all of the files had been stored directly into my rsilver24 user on my computer, and then Unity drew the assets from there directly. However, not only is this plainly poorly placed, but I wanted to start keeping this in Github rather than Unity Cloud. As such, I moved the files from

their origin into this new local repository. Unity at first didn't like this, saying that my project did not exist, but after routing it everything *appears* to be working as intended.



Finally I spent a good deal of time making this very document. The reason this took a while was because of a few reasons. One, this is a long log (who would have figured). Two, I needed to format everything including the title page. And three, I wanted to link *every single 20% thing* that was done in AT CS this year. I figured not only would these be decent resources, but would also show the work I had done in the past.

Overall this was a very efficient day that set me up really well for progress later this week and beyond.

To Do List

Start in-engine work! Current state of the project will be explained later, but at the moment I need to work on scene swapping for boss encounters. I got a little bit of work done on this prior, but now I actually have to make it work.

May 7th, 2024 — Studying Unity Scene Rules, Formatting Scene 1, Creating Buttons, and Coding Transitions

Accomplished

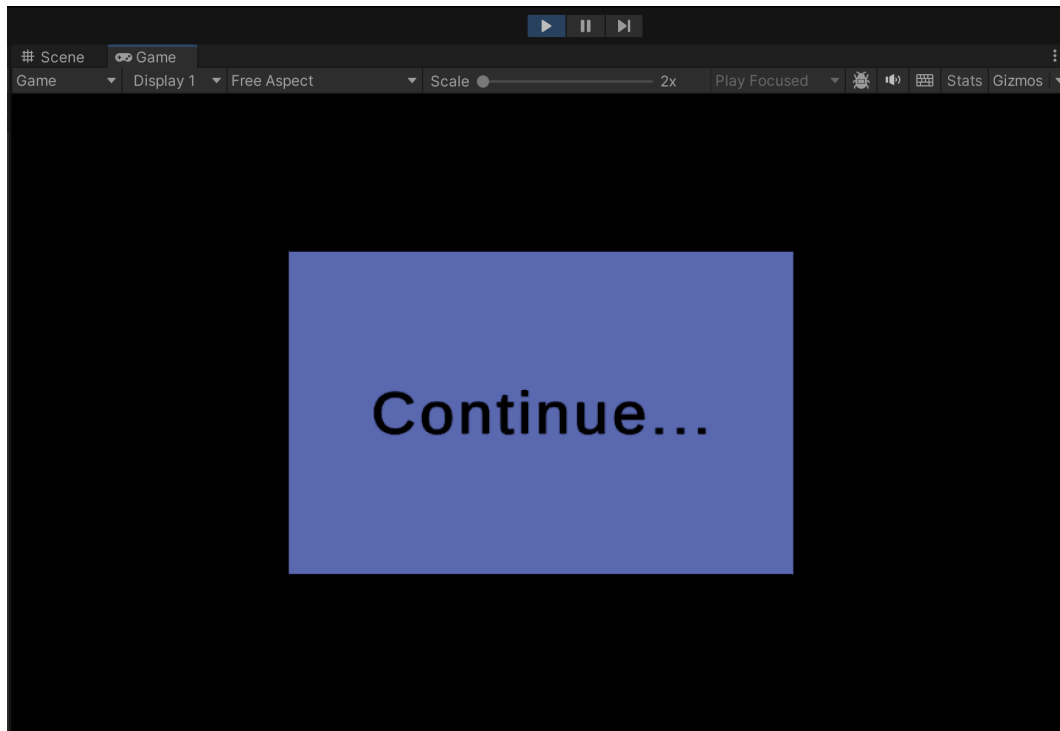
Today my goal was to get into a good place with scene transitions, both in fundamental game assets and code. In general, this was a success! It took me a while to get my footing, but once I started to understand what I had to do, progress came very quickly. I used three incredibly crucial resources for this work, which I've listed down below. In total I spent about 2 hours both in engine and on youtube playing around with features and trying to understand the fundamentals of it all. I wanted to make sure I had a solid understanding of what I was trying to complete before I actually dove in. With only two and a half weeks (ish) to work on this, I want to be able to make progress each day and not get caught up on things that aren't worth my time. But anyways, let me dive into my work today.

Unity 2D Scene Management: <https://www.youtube.com/watch?v=E25JWfeCFPA>

How to Create Buttons in Unity: <https://www.youtube.com/watch?v=gSfdCke3684>

Why Buttons do not Work in Unity: https://www.youtube.com/watch?v=gSil6P_5f1s

My process began with actually making this second scene, or the "boss 1" scene. There will eventually be a mini game here (hopefully), but for now I just wanted a general "click to progress" button that would return the player back to the main board. I ended up settling on a plain blue background at first while editing (in order to make sure I would be switching scenes correctly), but eventually made it a black one. I also made a square button that rests in the center of the camera that states "Continue...", and is a dark blue color at first that turns red when you hover over it. Making the button was a bit of a struggle because for whatever reason it would refuse to display. I ended up fixing this issue by establishing a Canvas asset to hold UI elements. This is typically always done first, but I neglected it because I thought the Canvas I made for Scene 0 would have carried over. Below is visually what I finished with in Scene 1.



The next step was figuring out how to go from Scene 0 (the board) to Scene 1 (the boss). I knew I wanted the scene to be triggered when either player landed on the boss hexagon, but I needed to know how to go about this. Through the first video listed on the previous page I made the script “SceneController.cs” to manage these transitions. This composed of three methods, which can be seen below.

```
C# FollowThePath.cs  C# Dice.cs  C# GameController.cs  C# ButtonUI.cs  C# SceneController.cs  C# BossFight.cs
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  rsilver24 *
7  public class SceneController : MonoBehaviour
8  {
9
10     public static SceneController instance;
11
12     rsilver24 *
13     public void Awake(){
14         if (instance == null){
15             instance = this;
16             DontDestroyOnLoad(gameObject);
17         } else{
18             //Destroy(gameObject);
19         }
20     }
21
22     rsilver24 *
23     public void nextBoss(){
24         SceneManager.LoadScene(1);
25     }
26
27     rsilver24 *
28     public void loadBoss(string sceneName){
29         SceneManager.LoadScene(sceneName);
30     }
31 }
```

SceneController.cs was useful for the method calls themselves, but now I needed to actually make sure these methods were being called. First I made a new empty object called “GameManager” that would hold the SceneController script. The Awake method would make sure that this game object would be transferable between different scenes, and the whole thing wouldn’t self-destruct. Second, I referred to SceneController.cs from within GameController.cs to trigger not only when the player reached the current waypoint index (found via FollowThePath.cs), but also when their turn had completely finished. The added code can be found below. This worked as intended, and when either player touched the spot, they transferred scenes.

```
46 // Update is called once per frame
47 rsilver24 *
48 void Update()
49 {
50     if (player1.GetComponent<FollowThePath>().waypointIndex > player1StartWaypoint + diceSideThrown)
51     {
52         player1.GetComponent<FollowThePath>().moveAllowed = false;
53         Player1MoveText.gameObject.SetActive(false);
54         Player2MoveText.gameObject.SetActive(true);
55
56         if (player1.GetComponent<FollowThePath>().waypointIndex == 12){ //Right here !!!
57             SceneManager.LoadScene(1);
58         }
59
60         player1StartWaypoint = player1.GetComponent<FollowThePath>().waypointIndex - 1;
61     }
62
63     if ((player2.GetComponent<FollowThePath>().waypointIndex) > (player2StartWaypoint + diceSideThrown))
64     {
65         player2.GetComponent<FollowThePath>().moveAllowed = false;
66         Player2MoveText.gameObject.SetActive(false);
67         Player1MoveText.gameObject.SetActive(true);
68
69         if (player2.GetComponent<FollowThePath>().waypointIndex == 12){ //And Right here !!!
70             SceneManager.LoadScene(1);
71         }
72
73         player2StartWaypoint = player2.GetComponent<FollowThePath>().waypointIndex - 1;
74     }
75 }
```

The next step was making the transfer between that new Scene 1 and the previous Scene 0. Since there were obviously no waypoints in this scene, I needed a different approach. Here I was going to use that previously described button. Upon contact, I wanted the scene to shift. Simple enough, right? It actually, luckily, was. Dealing with specific settings for each asset was an incredible hassle, but after enough trial and error it worked. All that was necessary to make the button capable of triggering the scene refresh was adding a collider, a companion object, and a companion script. The companion object named ButtonController connected to the button’s collider box, then the script “ButtonUI.cs” was applied, and NewGameButton() was called.


```
C# FollowThePath.cs  C# Dice.cs  C# GameController.cs  C# ButtonUI.cs x  C# SceneController.cs  C# BossFight.cs

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  IL code
7  public class ButtonUI : MonoBehaviour
8  {
9      IL code
10     [SerializeField] private string newGameLevel = "MainGame";
11
12     IL code
13     public void NewGameButton()
14     {
15         SceneManager.LoadScene(0);
16     }
17 }
```

At the end of everything it works! Scenes are transferred between and everything runs smoothly and for most use cases!

To Do List

So... where next?

Firstly I need to fix the reset of Scene 0. Although it does get transferred back to, everything completely refreshes as if it's a new game... almost. The waypoint index is saved, meaning that at the next roll the player catches up to where they were previously + the new roll, but it's inelegant and kind of stupid. As such, the first goal for tomorrow will be to make sure the players are returned to their respective spots.

After that I need to rinse and repeat for bosses 2, 3, 4, and 5. Since I now already know what to do and have most assets in place this should be pretty easy, but who knows? Maybe there will be a fatal error somewhere.

After that, if I have time, I want to start making the roll again spaces. These should be easy since it's literally just queuing the same player up to 'play again', but changing even a little bit of previous code often leads to disaster.

Then... it's just hoping for the best.

Accomplished

A screenshot of a game interface. At the top, there's a menu bar with options: '# Scene', 'Game', 'Display 1', 'Free Aspect', 'Scale' (set to 2x), 'Play Focused', and icons for settings, volume, and a keyboard. The main area is a dark gray board with a complex path of white hexagonal tiles. The path is composed of segments labeled 'P' (Path) and 'R' (Roundabout). There are several obstacles: a large gray rectangular block with two white squares, a red circular obstacle with a white 'n' inside, and several green alien-like creatures with large eyes. A blue butterfly-like character is at the bottom left. On the right side, there are two 'Your Turn' indicators, each with a red circular icon and a blue butterfly icon.

As you can also see in the screenshot, the player move text just stops working altogether. Obviously it CANNOT be both players' turns at the same time, but it says so anyways. This is an issue that only occurs after the button reset. But anyways, what did I do to get to this point today? Well quite a bit. Pictured below is the code I wrote to influence where the players would spawn after completing a boss fight. This was written within FollowThePath.cs, but right now after a single reset the button screen just appears over and over again. The issue might be somewhere in this code, or somewhere else entirely.

```
19     void Start()
20     {
21         transform.position = waypoints[0].transform.position;
22         if (SceneController.counter == 1)
23         {
24             transform.position = waypoints[12].transform.position;
25             waypointIndex = 12;
26         } else if (SceneController.counter == 2)
27         {
28             transform.position = waypoints[23].transform.position;
29             waypointIndex = 23;
30         } else if (SceneController.counter == 3)
31         {
32             transform.position = waypoints[34].transform.position;
33             waypointIndex = 34;
34         } else if (SceneController.counter == 4)
35         {
36             transform.position = waypoints[45].transform.position;
37             waypointIndex = 45;
38         }
39     }
```

Additionally I made sure that the boss sequence could be triggered at every one of the hexagon spaces. The code for this can be found below. Pictured here is only *half* of the code. The other half is the exact same thing but substituting player1 for player2. The commented out portions is a possible solution I drew to fix the current issues. Implementing it as is does not fix the issue, but I wanted to keep the framework for the possible solution.

```

49     if (player1.GetComponent<FollowThePath>().waypointIndex > player1StartWaypoint + diceSideThrown)
50     {
51         player1.GetComponent<FollowThePath>().moveAllowed = false;
52         Player1MoveText.gameObject.SetActive(false);
53         Player2MoveText.gameObject.SetActive(true);
54
55         if (player1.GetComponent<FollowThePath>().waypointIndex == 12){
56             //DontDestroyOnLoad(player1);
57             //DontDestroyOnLoad(player2);
58             SceneManager.LoadScene(1);
59         }
60
61         if (player1.GetComponent<FollowThePath>().waypointIndex == 23){
62             //DontDestroyOnLoad(player1);
63             //DontDestroyOnLoad(player2);
64             SceneManager.LoadScene(1);
65         }
66
67         if (player1.GetComponent<FollowThePath>().waypointIndex == 34){
68             //DontDestroyOnLoad(player1);
69             //DontDestroyOnLoad(player2);
70             SceneManager.LoadScene(1);
71         }
72
73         if (player1.GetComponent<FollowThePath>().waypointIndex == 45){
74             //DontDestroyOnLoad(player1);
75             //DontDestroyOnLoad(player2);
76             SceneManager.LoadScene(1);
77         }
78
79         if (player1.GetComponent<FollowThePath>().waypointIndex == 56){
80             //DontDestroyOnLoad(player1);
81             //DontDestroyOnLoad(player2);
82             SceneManager.LoadScene(1);
83         }
84     }

```

I also added a little counter to the button trigger that would be stored globally in order to keep variables active between scene transitions.

```

10     public void NewGameButton()
11     {
12         SceneController.counter += 1;
13         SceneManager.LoadScene(0);
14     }
15 }

```

So overall yes progress was definitely made and I have a greater understanding of what I need to do and what needs to be done, but I was really hoping that there would be no bugs and I could get past this step smoothly. I have AP's coming up so I doubt things will be finished at the next entry, but I want this fixed by the end of this week or so.

To Do List

Fix ALL of the pressing issues. This mainly comprises the below features:

- Make player icons reset at 'correct' positions
- Update waypointIndex accordingly to either both characters or a common variable between characters
- Stop the eternal repeat of button prompts
- (maybe) Create a new script to store character values
- (maybe) Find a way to keep player scripts + icons + values while not creating a new one in place of it upon scene reset
- Fix the player turn texts
- Make sure the final "player x wins" text still pops up upon game completion
- TBD

Hopefully progress over this week goes far smoother.

May 9th, 2024

AP Psychology Exam!

May 10th, 2024

AP European History Exam!

May 11th, 2024

Weekend!

May 12th, 2024

Weekend Part 2!

May 13th, 2024

AP Calculus AB Exam!

May 14th, 2024 — Working on Fixing the Player Location Issues and “Bootstrap” Efforts

Accomplished

After a five day break from this project thanks to the weekend and three AP exams in a row (painful), I was very ready to jump back into things. However, I really didn't realize how much of a pain it would be to get this working. Today was a day of not too much tangible progress, but a good day of both research and process development. Today my work was broken down into these steps:

- Do Research! What is the issue, how should I fix it?
- A strong *attempt* at “Bootstrap” scene management and code.
- Object permanence + data keeping.

After looking through the general issue I was facing (seen in the previous entry), I decided that there was one of three possible routes I could take. One was to keep the objects of the players stored and active during scene transitions, then when the initial scene was resumed, find a way to delete the “new spawns” of player icons, then resume play with the previous. The second idea was to make a new object that would be kept active between scenes that would work purely as a variable holder, of which variables would then be applied to newly spawned player objects and they would comply as such. The final solution was to make a “Bootstrap” async scene to hold the player icons, which I will describe in more detail. After deciding the first solution may be too costly (taking up a ton of time in the previous work day) and the second would be too messy and hard to organize (since I also played with the idea that session), I decided to go and try the “Bootstrap” method.

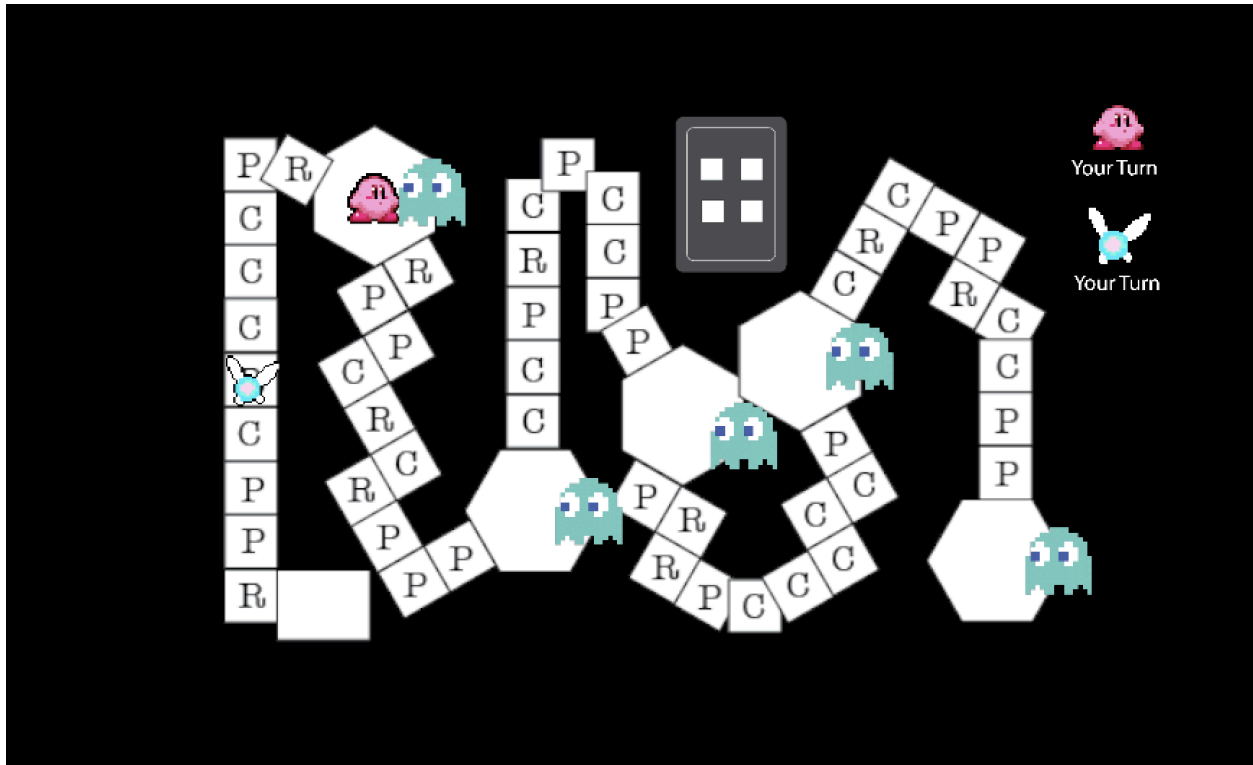


The video above (also found here: <https://www.youtube.com/watch?v=o03NpUdpdrc>) walked me through the general gist of both what a bootstrap method + scene is, and what it can be used for. It additionally gave me a sample bit of code that could be modified for greater use. Generally a “Bootstrap” is an ongoing scene that runs asynchronously from the current scene. Telling it to never be destroyed means that you can store objects, variables, and elements that you wish to NEVER change. This, as you can imagine, has a lot of useful cases. The code below is what I developed:

```
7 public class Bootstrap : MonoBehaviour
8 {
9     // Start is called before the first frame update
10    async void Start()
11    {
12        Application.runInBackground = true;
13        //await UnityServices.InitializeAsync();
14
15        if (SceneManager.loadedSceneCount == 1)
16            SceneManager.LoadScene(0, LoadSceneMode.Additive);
17    }
18
19    [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]
20    static void Init()
21    {
22        #if UNITY_EDITOR
23            var currentlyLoadedEditorScene = SceneManager.GetActiveScene();
24            #endif
25
26            if (SceneManager.GetSceneByName("Bootstrap").isLoaded != true)
27                SceneManager.LoadScene("Bootstrap");
28
29            #if UNITY_EDITOR
30                if (currentlyLoadedEditorScene.IsValid())
31                    SceneManager.LoadSceneAsync(currentlyLoadedEditorScene.name, LoadSceneMode.Additive);
32            #else
33                SceneManager.LoadSceneAsync(0, LoadSceneMode.Additive);
34            #endif
35    }
36 }
```

So, the question you are probably waiting for, did it work? Well... kinda. After making the scene itself, it *did* work. Both “MainGame” and “Bootstrap” ran simultaneously, but that’s where the issue began. I could only directly operate one of these two scenes at a time. Both were visible, but only one was operable. So why isn’t this normally an issue? Well think of it this way. In most games, you the person player operate and move your own icon/model throughout the gamespace, but besides maybe an “interact” button, you do not manipulate the environment outside of this character. But my game, being a board game, has things of the game space you “tap” without the influence of your character. As a matter of fact, you *technically* don’t move your character at all, as it is

completely self guided off of the randomized results of the dice pull. Because of this, although in theory with more tinkering it would work, without a specific guide as a part of this bootstrap, and it instead being completely dictated by the events of a single scene's events, this solution is not at all optimal. So, as such, I decided to retire this solution, keeping it commented out and unchecked, just in case I do wish to come back to it.



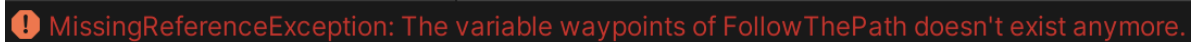
As you can see from this photo, I did end up getting it to “work”. Upon clicking the button and being transported back to the original scene, the players remain in their positions and the new player icons are thus deleted. I did this by writing a slight bit of code that, as long as the scene button has been hit at least once, the new object just destroys itself, while the original one is preserved (commented code from last entry).

```
if (SceneController.counter > 0)
{
    Destroy(gameObject);
}
transform.position = waypoints[waypointIndex].transform.position;
```

However, as much as I wished the solution was this easy... it is not. I shall describe this problem in greater detail in the To Do List section below.

To Do List

Currently when everything is reloaded back into scene one, although everything is in their necessary spots, nothing else is able to happen... kind of. The dice is still able to be rolled and CAN be rolled continuously without any error, but the results of the dice are never applied onto the player icons, so they stay completely still. The player text additionally says "Your Turn" for both players, but that is a later issue to be solved. The reason this is the case is very very likely because of this issue that only appears when I get to this stage of the game:



! MissingReferenceException: The variable waypoints of FollowThePath doesn't exist anymore.

The variable waypoints is a variable that holds all the data for the waypoints... as you could imagine. This variable comes in the form of an array, with each value in said array being a Transform variable. This is how the player icons locate where they should go.

So what is my assumption? I think while resending the scene, the waypoints disappearing and then reappearing mess with the variable, as fundamentally the new waypoints are completely wiping out the old. So, how to fix this? Well I think I'm about to go in and do the thing I had to with the players and DiceCards: make sure they don't disappear, and new ones aren't loaded in. So.. what does this entail. Well, the code is about to get really busy. I must:

- Assign a "waypoint" script to each waypoint that has the sole purpose of deleting its object if the counter is > 0.
- Make sure the waypoints never get deleted. This is going to take a lot of repetitive code even with use of loops. Maybe I should make a new method?
- See where I am by then.

Well, good luck to me.

May 15th, 2024 — Trial and Error, Error, Error...

Accomplished + To Do

Today was fundamentally a 'try and see what works' kind of day. I had tried so many different solutions to my issues, and every single one of them just brought more and more problems with them. So... what am I going to do? I've realized that the vast majority of my issues stem from the scene transitions. I have a lot of objects that do many varying things, and the scene transitions keep on messing things up. It's gotten to the point where I make a ton of "patch" fixes that work for the current rendition, but the moment I add or take away anything I have to find a way to fix every single thing up again.

This is becoming very straining on both my mental energy and my time, so I've decided I need to do a complete re-initiation of my efforts. I am going to ditch scenes all together. It seems like a strange idea at first, given how much effort I've put into everything, but I think getting this done will allow me to exponentially fast forward progress in this department. Realistically with the presentations next Wednesday, I do not have remotely enough time to be weighed down like I have been right now. So, here's the plan:

- Keep the current rendition of this project saved and completely untouched. Make sure it is completely updated in the Git Repository.
- Make a new project with all of the exact same files and settings. This will be the 'new' project that will be changed in order to attempt the scene migration.
- Instead of scenes in this new project, with how simple everything should be to run, I am going to purely do camera transitions and nothing more. If elements are outside of the camera range, then the player will not be able to interact with them, and thus the encounters will never truly begin.

Generally I am doing this because I believe dealing with scenes has been too costly and too taxing on my mental wellbeing since I've been super busy over this time.

Because of this level of stress I am feeling, I may take a mental health day tomorrow, just to get ahead on all the other things I am behind on. We shall see.

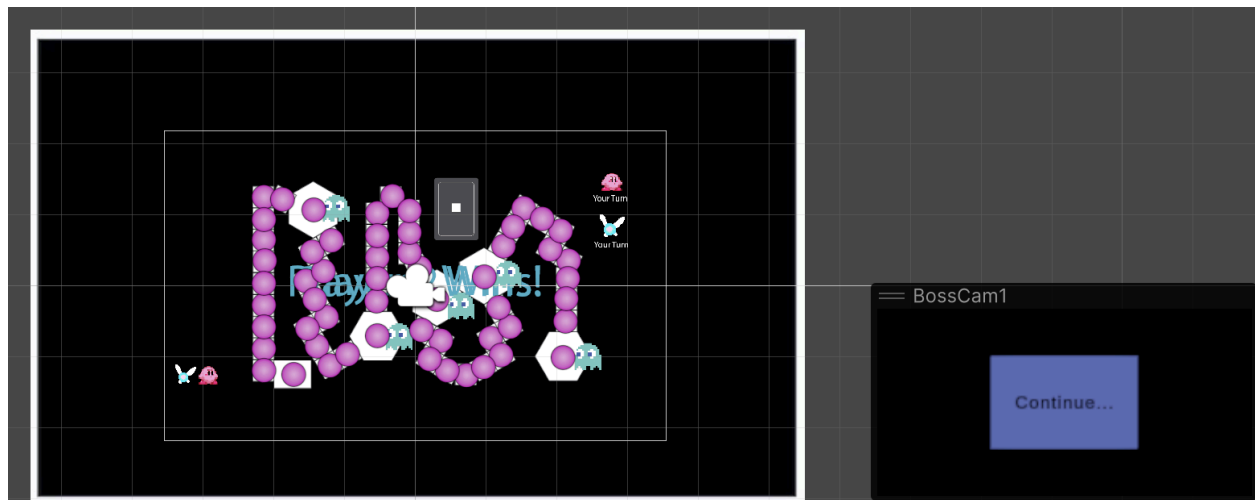
May 16th, 2024

Mental Health Day!

May 17th, 2024 — Reformatting Scene Transitions into Camera Shifts

Accomplished

Today, as described in the previous entry, I worked extensively in transferring all of the scene transitions into camera shifts instead. This was because scene transitions were giving me so many issues and were taking up so much time. I figured it would be better to completely transition my work today rather than struggling along with my previous system. I also took today to clear up some assets that were seemingly unnecessary, and cleaned up some of the scripts as well. I will not display all of these smaller efforts, as generally they are insignificant towards my greater progress.



As you can see from the little clip in the bottom right corner, I have successfully migrated all assets from the boss scene into the main game scene. This mainly just consisted of dragging and dropping, but because the button was a Canvas element, I elected to combine them into one, which only took marginally more effort. Although really simple, the only thing this realistically accomplished was getting all of the assets loaded correctly upon game start, which was crucial, but not too difficult. What took far more effort was the coding and system endeavors, which I will describe now.

```
C# FollowThePath.cs  C# WayPoint.cs  C# Dice.cs  C# GameController.cs  C# ButtonUI.cs  C# Cameras.cs x  C# SceneController.cs  C# Boot:

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Cameras : MonoBehaviour
6  {
7      public GameObject MainCamera;
8      public GameObject BossCam1;
9
10     public static int state = 0;
11
12     void Start()
13     {
14         MainCamera.SetActive(true);
15         BossCam1.SetActive(false);
16     }
17     // Update is called once per frame
18     void Update()
19     {
20         if (state == 0)
21         {
22             MainCamera.SetActive(true);
23             BossCam1.SetActive(false);
24         }
25         if (state == 1)
26         {
27             MainCamera.SetActive(false);
28             BossCam1.SetActive(true);
29         }
30     }
31 }
```

Above is the code I wrote for camera management. It is very simple in concept. This script was attached to an empty object so that it would always be present while the scene (and now the only scene) would be active. At the very start of the game, detailed by the Start() method, the main camera would be activated while the boss cam would be deactivated. The camera state would be continually updated so that based on the int value of state, the cameras' status would be altered. The value of state would be changed by the following code below (this was written for every space and both players, so an additional 9 times).

```
if (player1.GetComponent<FollowThePath>().waypointIndex == 12){
    Cameras.state = 1;
}
```

I also altered the function of the button in a very similar way, which I've also pinned below.

```
public void NewGameButton()
{
    SceneController.counter += 1;
    Cameras.state = 0;
}
```

After all of this was written, progress became super super steady as I had hoped. All the code that I had previously written worked perfectly in the context of no-stupid-scene-shenanigans, and player movement became flawless. Upon landing on each boss, the button camera is activated and thus can be pressed. After being pressed, since nothing is being reset, the players remain with their correct variable values and positioning. This ALSO included which player's turn it was, meaning that I could actually *delete* all of my previous measures. In other words, today was an amazing day where so much progress was made.

Am I a bit disappointed I couldn't get the scene transitions done correctly? Sort of. I'm generally really happy with the state of my game at the moment, but having distinct scene transitions would have been good knowledge for future use. At the very least I have so much of a better understanding of how the feature works, and should, if I were to make a new game, be able to implement it all far easier.

To Do List

Well... start modifying the space elements! With only a few more days of CIE work left, I don't want to get too preoccupied with making the boss elements unique. I would rather make the board spaces (marked by letters) complete actual actions, and increase the actual 'game' bit of the 'board game'. However, with scene transitions FINALLY out of the way, progress should move super smoothly, and I will probably be able to go even further with everything.

It's just a bit disappointing it took this long to get to this point.

May 18th, 2024

Weekend!

May 19th, 2024

Weekend!

May 20th, 2024 — The Final Work Day! Implementing the “Roll Again” Space and Trying at the Power Spaces

Accomplished

Being really short on time tomorrow due to obligations, I decided that today would be my final day to implement features, with tomorrow dedicated towards compilation, presentation, and conclusion. With everything in place for the boss fights (at least the placeholders, yes), I wanted to start working on the board spaces themselves. I will re-explain what these do.

- R Space → Reroll! Go again, and see how far you can go.
- P Space → Increases power on a random scale from 1-6. Meant ideally for boss fights in the final rendition.
- C Space → Pull and Event Card!... tbd on details... but probably will never tbd...

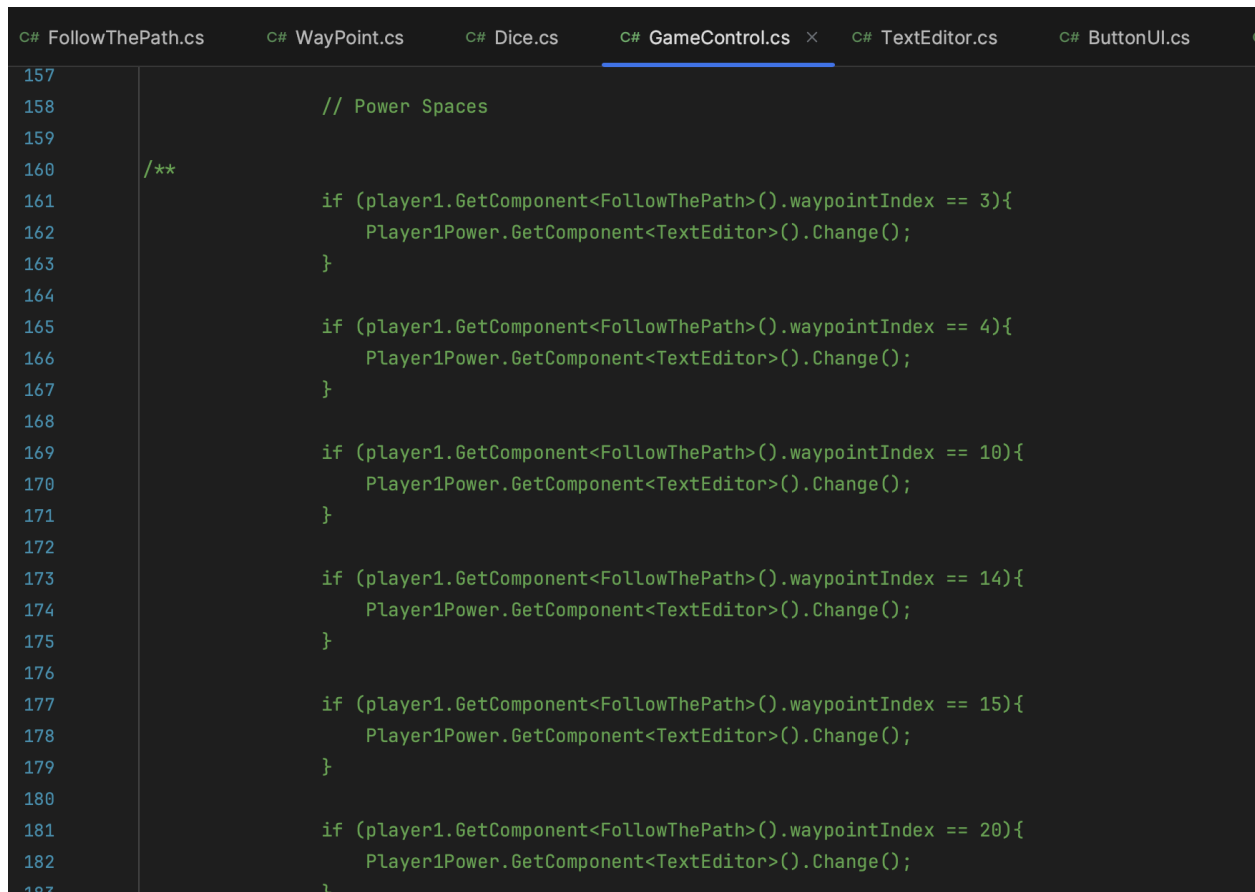
I decided that realistically the two of these I should focus on are the R and P spaces. I wanted to start with the R space since I thought it would be a pretty simple feature to implement. Afterwards, I wanted to give at least a shot at the P spaces. Since I wouldn't have to actually implement the features of power, I figured I could just make a simple counter on screen. The issue here was that I have consistently had major issues with UI elements, so I was just going to hope for the best.

```
C# FollowThePath.cs C# WayPoint.cs C# Dice.cs C# GameControl.cs X C# TextEditor.cs C# ButtonUI.cs C# Camera
90 // Roll Again Spaces
91
92 if (player1.GetComponent<FollowThePath>().waypointIndex == 2){
93     Dice.whosTurn = 1;
94     Player1MoveText.gameObject.SetActive(true);
95     Player2MoveText.gameObject.SetActive(false);
96 }
97
98 if (player1.GetComponent<FollowThePath>().waypointIndex == 6){
99     Dice.whosTurn = 1;
100     Player1MoveText.gameObject.SetActive(true);
101     Player2MoveText.gameObject.SetActive(false);
102 }
103
104 if (player1.GetComponent<FollowThePath>().waypointIndex == 11){
105     Dice.whosTurn = 1;
106     Player1MoveText.gameObject.SetActive(true);
107     Player2MoveText.gameObject.SetActive(false);
108 }
109
110 if (player1.GetComponent<FollowThePath>().waypointIndex == 13){
111     Dice.whosTurn = 1;
112     Player1MoveText.gameObject.SetActive(true);
113     Player2MoveText.gameObject.SetActive(false);
114 }
115
116 if (player1.GetComponent<FollowThePath>().waypointIndex == 17){
117     Dice.whosTurn = 1;
118     Player1MoveText.gameObject.SetActive(true);
119     Player2MoveText.gameObject.SetActive(false);
120 }
121
122 if (player1.GetComponent<FollowThePath>().waypointIndex == 19){
123     Dice.whosTurn = 1;
124     Player1MoveText.gameObject.SetActive(true);
125     Player2MoveText.gameObject.SetActive(false);
126 }
```

The code pictured above is what I wrote. It follows a very similar format to the previously mentioned Boss Fight system, as I tracked where a player ended up via their WayPointIndex. I also repeated this code for a few additional spaces, and for the player2 game object as well. The three elements of this code is as follows.

- Resets the player turn to be the same player's turn again
- Set active the "Your Turn" text for the same player
- Set inactive the "Your Turn" text for the other player

Since I had already of course written what was to happen in a player turn, this code was just meant to repeat things all over again. I had a slight hiccup updating the wrong variable contextually, but I quickly fixed that up and the feature works as intended. If I had more time, however, I would have liked to make it more flashy.



```
C# FollowThePath.cs  C# WayPoint.cs  C# Dice.cs  C# GameControl.cs x  C# TextEditor.cs  C# ButtonUI.cs
157
158         // Power Spaces
159
160     /**
161         if (player1.GetComponent<FollowThePath>().waypointIndex == 3){
162             Player1Power.GetComponent<TextEditor>().Change();
163         }
164
165         if (player1.GetComponent<FollowThePath>().waypointIndex == 4){
166             Player1Power.GetComponent<TextEditor>().Change();
167         }
168
169         if (player1.GetComponent<FollowThePath>().waypointIndex == 10){
170             Player1Power.GetComponent<TextEditor>().Change();
171         }
172
173         if (player1.GetComponent<FollowThePath>().waypointIndex == 14){
174             Player1Power.GetComponent<TextEditor>().Change();
175         }
176
177         if (player1.GetComponent<FollowThePath>().waypointIndex == 15){
178             Player1Power.GetComponent<TextEditor>().Change();
179         }
180
181         if (player1.GetComponent<FollowThePath>().waypointIndex == 20){
182             Player1Power.GetComponent<TextEditor>().Change();
183         }
```

This code above (which is intentionally commented out) was meant to work for the Power 'P' spaces. Using a very very similar system to the previous kinds of spaces once again, this called upon the method Change() within a new class that I had made to manage text changes. I have listed that code below.

```
C# FollowThePath.cs  C# WayPoint.cs  C# Dice.cs  C# GameControl.cs  C# TextEditor.cs x  C# ButtonUI.cs

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5
6  rsilver24  IL code
7  public class TextEditor : MonoBehaviour
8  {
9      //public static Text canvasText = gameObject;
10     IL code
11     public static int state = 1;
12     IL code
13     public static int power = 1;
14
15     // Start is called before the first frame update
16     rsilver24  IL code
17     void Start()
18     {
19         gameObject.GetComponent<Text>().text = "Power: 1";
20     }
21
22     rsilver24  IL code
23     public void Change()
24     {
25         power += Random.Range(1, 7);
26         gameObject.GetComponent<Text>().text = "Power: " + power;
27     }
28 }
```

This TextEditor script was individually attached to both power listing text boxes. These text boxes are not visible because I have hidden them for the sake of presentations on Wednesday, since the feature is very incomplete. This was meant to very simply update the power mathematically through the Random.Range() method, then alter the text to accommodate. However, because of my complete lack of understanding on how UI elements work (especially text ones), I was unable to get anything visualized. The text would forever stay in its “Power: 1” state, and even if I updated it in the system, for whatever reason it refused to change. Unfortunately that is where I must leave off this feature. I think if I had at least 2 more days of design work I could definitely get this done and working. It just requires research and trial and error—both of which I do not have the time for.

So, feature-wise, this is the final stopping point of this project.

To Do List

Welp. Time to finish it all up! This will mainly comprise of these actions:

- Make a google-slides presentation for Wednesday.
- Update the ReadMe file in the git repository.
- Delete some of the ugly objects and scripts that are unnecessary, while keeping most for the sake of preserving what I got done.
- Build the project and make sure it can run.
- Upload the build into the git repository.
- Make the final log entry for May 21st
- Add a 'final thoughts' page in this document? This will only be if I have time. And Energy. I do (hopefully) graduate in 9 days, afterall.

May 21st, 2024 — Presentation and Conclusion

Accomplished

And just like that, I am done with my senior project. Today my work should be pretty self-explanatory, but I will describe it anyway.

The first thing I did was make a build of my project and export it onto my desktop. This file is not only necessary for showing “yes, here is my product,” but also so that I can easily access it during my presentation tomorrow. This completed game file will also be uploaded into github, right after I finish this document. I decided I will export this current document as a pdf and throw it into github, just for the sake of it.

After this I made my presentation. This presentation can be found here: <https://docs.google.com/presentation/d/1yS2zvhHpj8noXFjss3AlFdJiv2LjKrfIRbd6yr6oZYQ/edit?usp=sharing>. This took me about an hour to complete, and mainly just detailed the process of me making Impori. This particularly included this documentation, my code, my unity files, my git repository, and a game demo.

I uploaded everything to git! This includes all final code edits, project settings changes, the game build, (eventually) this documentation, and any necessary assets involved.

But yes! I am now done. Is there room for improvement? Absolutely. There are so many features and elements of the Impori draft that I was just never able to get to, and I could definitely spend more time finishing it all up if I want to.

Overall, I would say that my time spent on this project was very worthwhile. It took a lot of time and energy, but I have learned so much about one of my favorite things in the world, being video games. I think, if I have time this summer, I would like to start again, but fresh this time. I think I could make something really good. I thank Kent Denver for the opportunities presented to me by this CIE period, and my greater highschool experience as a whole.

To Do List

Present tomorrow! Both during the official one at 2:00pm, and Senior Sunset in the evening.

And then graduate I suppose...