ÉCOLE POUR L'INFORMATIQUE ET LES TECHNIQUES AVANCÉES

UNDERGRADUATE 1^{st} Year SEM. 2

Project Q
FAUST

POLLO Y PAPA

DAVID CALDERÓN, MAXIME BARDOUIL, RAJ MAHAJAN,
WADHAH OULED AMEUR

IAN TERNIER

JUNE-2022

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

Faust, is a rogue-like game influenced by Dead Cells and Hades, which presents the story of a man driven with the obsession of gaining omnipotence. Faust is a 2-dimensional game created by Pollo-Y-Papa, with a pixel art style and progressively increasing enemy difficulty. Faust has a great variety of enemies that the players could face as well as numerous maps and biomes for fresh playability. Finally, while rogue-like games are single players, Faust will have a multiplayer option for the players to choose.

In this report we will explore the process of the creation of the game, this includes the difficulties we experienced, what we learnt, as well as the final product.

***Figure I**: The Faust logo*

Chapter 2

MODIFICATION TO THE BOOK OF SPECIFICATIONS

During the process of making the game, some assigned roles were changed. The following table shows what each person worked on.

This section states the distribution of the major elements that will be worked on for the Game. The word "Head" implies that the person is in charge of the task while, "Vice-Head" is someone that will be helping the person complete it.

	David Calderon	**Maxime Bardouil**	**Raj Mahajan**	**Wadhah Ouled-Ameur**
Project Manager			X	
Storyline	Head		Vice-Head	
Physics of the Game	Head		Vice-Head	
Map Generation AI				Head
Animation	Vice-Head		Head	
Interface		Vice-Head		Head
Character, Stage Design			Head	Vice-Head

Table I: *Distribution of work*

Chapter 3

PROGRESS

Presentation	Tasks

6 - 17 June, 2022	• Installation Manual and Operating Manual
	• Player upgrade system
	• Final enemy set-up
	  ○ Implementation for bosses and other enemies
	• Map Generation
	  ○ Fully operational and optimized map generation AI
	• Network
	  ○ Be able to host 2 player online game
	• Audio
	  ○ Ambient Sounds
	  ○ In-Game Music
	  ○ Sound Effects
	• Finish website
	  ○ Downloadable versions of the game
	  ○ Timetable of the game progress

Table II: *The tasks completed for this presentation*

In the following sections each of the group members will explain their progress over the course of the entire project. Each member will describe it in chronological order.

3.1 David Calderon

The things that characterize a rogue-like game are more often than not the aesthetic of the game, the story behind it, the enemies and the epic boss fights. As for me I will describe how I tackled the implementation of some of these elements in our game, as well as the good and bad moments along the way.

3.1.1 Enemy AI

The first thing that needed to be figured out was how the enemy AI was going to work, and after some discussion we decided to go for a patrolling behavior, which means that the enemy goes back and forth within the bounds of a specific area, and whenever the player stands in front of it, the enemy would attack.

Figure II: *Left and right bounds for an enemy*

At first this felt like a monumental task and it took quite some time to implement and debug, as working with scripts to describe behaviors and having them communicate with so many moving parts was something I had not done before. But once we got this feature off the ground it really was surprising and satisfying to see how it could apply to every single enemy across the entire game.

All of the enemies outside the boss rooms can be divided between the ones that shoot projectiles and the ones that do not. I handled the ones that dont while Raj handled the ones that do, and he will further elaborate on that in a specific section.

3.1.2 Bosses

If throughout all the gameplay we were to only have these two type of enemies, the game would feel incredibly repetitive, we needed to give a different feel of what the game was so far in order to change the pace and defy the player's expectation, making it really feel like a challenge; To do so we made bosses with their own special abilities, which at least for me, were by far the most interesting to code.

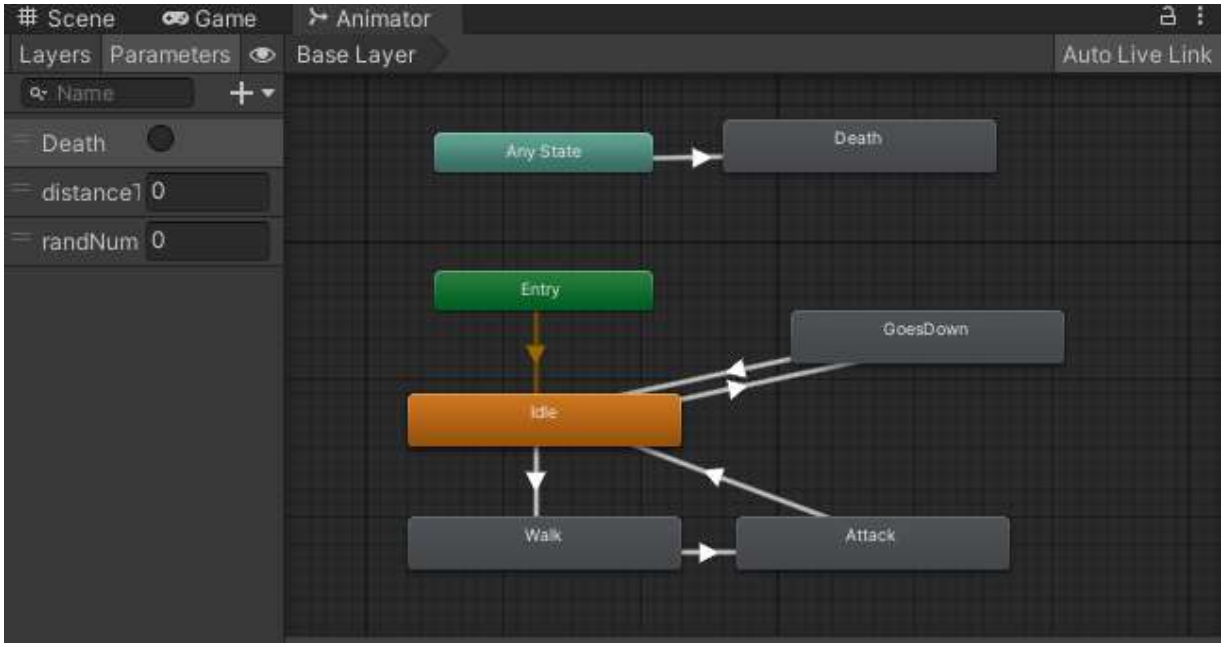


Figure III: *Animator for the first boss*

To make these abilities, it always came down to seeing the animator as a machine state diagram, describing at all times how the boss was acting, either walking, attacking, shooting, and so on. By attaching a script in the relevant actions one could queue anything at all when the boss was entering, staying and leaving a state.

Although confusing at first, the creation of new abilities started becoming really natural once a couple of them were done.

3.1.2.1 Vritra the Tyrant

For this first boss we went for a mix of ranged and close combat, the first biome did not really contain any ranged enemies so this was an important element to introduce in order to make the player adapt to new mechanics.

Figure IV: *First boss "Vritra"*

The ranged attack is characterized by its low damage but high damage per second, meaning if the player stands still, his life would be depleted rather quickly, while the melee attack is exactly that, and with an appropriate damage for being on the first stage of the game.

Figure V: *First boss ranged attack showcase*

Figure VI: *First boss melee attack showcase*

3.1.2.2 The Obese Baron Molikroth

For this boss we decided to go for really raw, and powerful melee attacks, as at this point the player should have somewhat better statistics.

Figure VII: The obese Baron Molikroth

The ability that really sets this boss apart from other enemies is his jump attack, for which he takes a small jump and deals some damage on landing. If the player wants to avoid taking the damage, he most likely than not needs to use his dash, and be quick about it, making this one a really interesting fight.

In addition to this, the boss melee attack occurs in bursts of three, offering quite a challenging and fresh encounter.

Figure VIII: Molikroths jump attack

Figure IX: Molikroths burst attack

For the third biome, we wanted to do something special and storyline coherent, so what we ended up doing is putting these two pass bosses in a room simultaneously, the problem is that we did not build the scripts for this, so after changing some of them, we got it to work.

Figure X: *Snow biome boss room*

3.1.2.3 Mephistopheles

As this was the final boss, we wanted to do something special and different from all other encounters, so with the help of an A* Pathfinding module and with the methods detailed before for creating the abilities, we settled on the final boss having the ability to fly while we jump around platforms trying to avoid his second attack which is a poisonous gas.

At this point in the game, and if the player has grabbed most of the upgrades(the upgrade system will be described just after), they should be able to defeat the boss.

Figure XI: *Mephistopheles lair*

Figure XII: *Mephistopheles poisonous gas*

Figure XIII: *Mephistopheles melee attack*

3.1.3 Upgrade system

After testing most of our game, we came to the realization that it would be too hard and honestly too one dimensional if the player had no way to improve its abilities as he goes through the biomes, so that is when we came up with the idea of “upgrade books”.

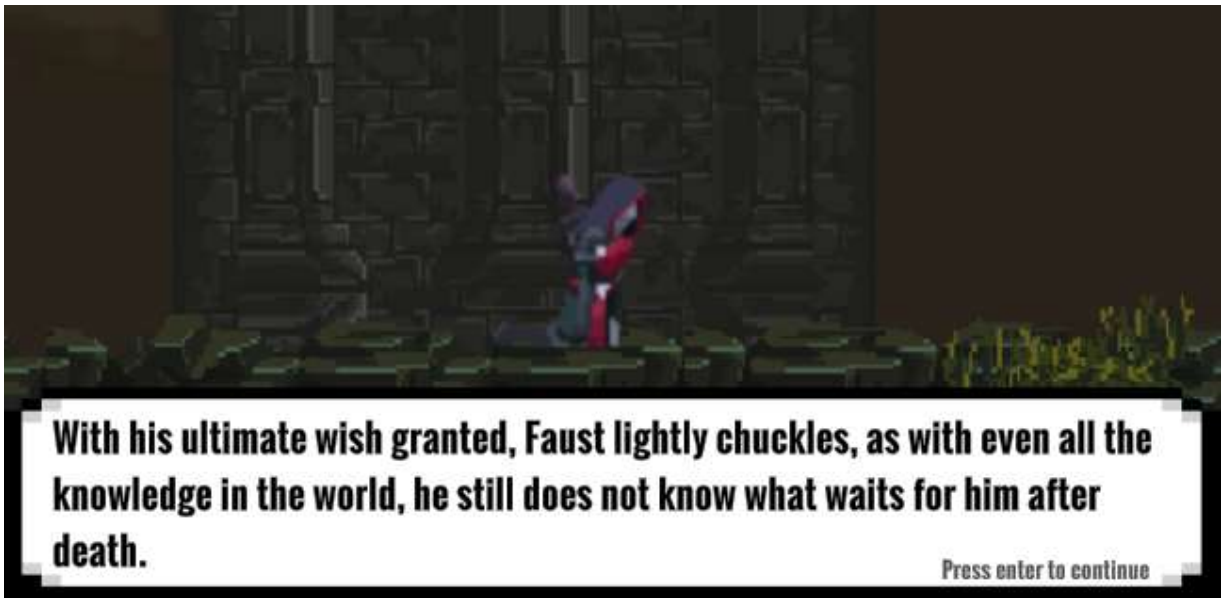
Around the world there are books that increase the statistics of the player, those statistics can be its speed, strength, attack speed and maximum health, as well as giving him some health back.

Figure XIV: *Book images*

Personally, this is one of the ideas I am most proud of for implementing, because I really was able to put a lot of the knowledge I had acquired before to create something unique and without the help of any online resources other than when getting the images.

3.1.3 Storyline

After finishing most of our "programming-heavy" ideas we noticed that the game lacked some depth, some story to follow, so we decided to implement something that made sense given the context we were working in.



Figures XVI: *Story extracts*

3.2 Raj Mahajan Aka Mali

3.2.1 Player and Enemies

To start the development of the game the group decided to create a prototype to understand how Unity works, therefore I started by making a prototype of the player. The following picture shows the character we choose as a prototype for the game.

Figure XVII: *Faust Prototype Design (Martial Hero)*

There were 2 major parts for the Player. First was the Animation and second was the Physics behind it. Firstly, the animations were created by putting frames together and creating a loop. Then using the Unity Animator, each animation was linked to another depending on some conditions. For example, if you are currently in the "Idle" animation (in the case of the picture below Idle animation is called "Martial Hero") and want to switch to the running animation, you need to check if the player is moving hence creating a condition. Each arrow from an animation to another requires the fulfillment of some conditions. These conditions were fulfilled by writing some code that kept track of things such as the health or the speed of the character. The following figure is the animator created for the Martial Hero.

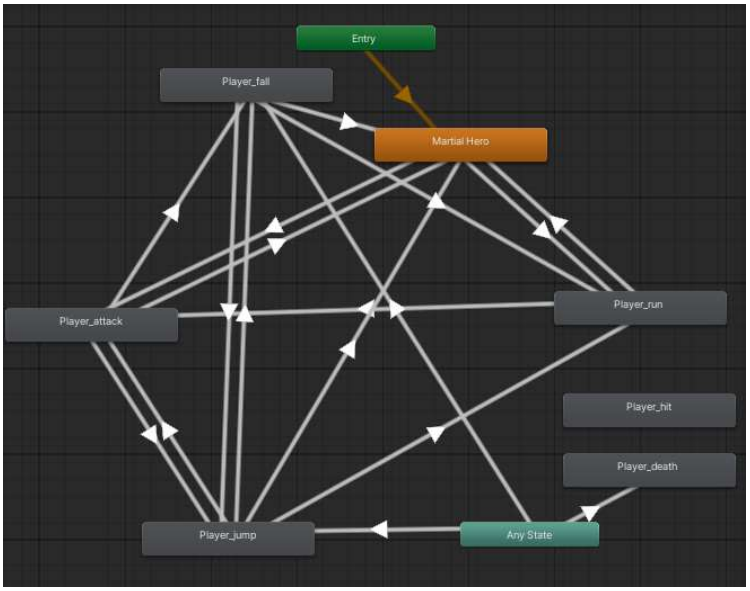


Figure XVIII: *Animator for the Martial Hero*

Once that was done, I needed to allow the character to move, hence I created a Movement script that took in the inputs of the player and moved the player accordingly. Alongside that I needed to sync the animations with the actual movement.

Finally for the player I had to create smaller scripts that would be essential to the game such as a script that allows the camera to follow the player.

A similar process had to be conducted for the enemies where I created animations for the enemies and attached created triggers to trigger the animations. 2 types of enemies were created: a melee type (Figure XIX) and a ranged type (Figure XX).

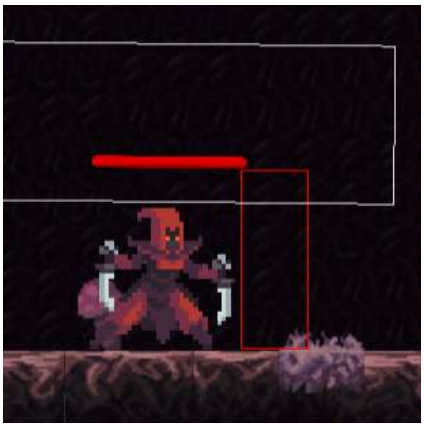


Figure XIX: *Melee Enemy*

Figure XX: *Ranged Enemy*

While the melee enemy did not prove to be a major difficulty, the ranged was a challenge since it required dealing with the ranged enemy and a fireball animation. To accomplish this task I had to create the animations for both the enemy and the fireball and then use a principle called object pooling. To do this we would have a pool of fireballs, stored in a holder, who could be activated when needed. I used this principle for every ranged enemy.

3.2.2 NPC

An NPC is a Non-playable Character, meaning these are characters that cannot be played but also have the connotation of being neutral towards the player. This means that enemies cannot be classified as NPCs.

Firstly, an NPC was created to do this, an asset of a flying eye was found, once that was done with the addition of an indicator and frame an NPC was created. A NPC works with 2 rules. Once the player gets in a certain range of the NPC, a small purple indicator will be visible to the player as in Figure VIII.

This gives the player the option to click enter, in which case the player will initiate a conversation, which will allow them to see a text in a white frame (Figure IX). NPCs will be used to provide buffs to the character before bosses and are used to explore the story.

Figure XXI: *NPC with an Indicator*

Figure XXII: *NPC with an Indicator & the Frame*

3.2.3 The Actual Player Character

Finally came the final stage where I had to move from the prototype of the Faust to the finalized version. We used the asset called Dark Knight as seen in Figure XXI.

Figure XXIII: *Dark Knight*

To turn this asset into the player I performed the same steps. I created the animations then attached the numerous scripts created to it and hence was born the new Faust. However, not every aspect of the knight was finished, I had to create an extra skill called the Dash. While the dash itself was not complicated, I decided to create an afterimage effect for the dash. This required me to use a technique I learned previously, object pooling. When the player dashes, he would instantiate the picture of himself every few milliseconds creating a dash afterimage.

Figure XXIV: *Example of Dash Effect Image (Not in-game)*

Finally, the new Faust was created and so were all of his abilities.

3.2.4 Storyline

While Faust is an action game, it still has a storyline and as David previously mentioned we used images to stimulate comic book type images to present the story. I worked alongside him to write the storyline allowing the players to explore it.

Figure XXV: *Comic Panel for the Story*

3.2.5 Audio

Now came the end of the game, we decided to add music to the game. Therefore, I created an audio manager, this manager contains all the sounds you will hear in the game in which I am able to change its volume and pitch. These sounds are then related to the actions of the player such as attack, dash etc... This allows for a more interactive game.

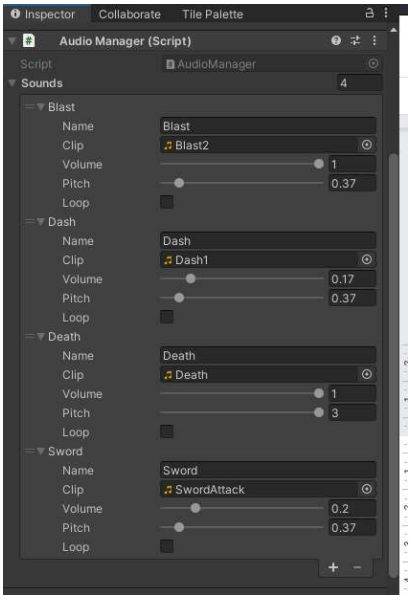


Figure XXVI: *Audio Manager*

3.2.6 Website

Over the course of the project, I have been working on the website. At first I added the progress page where all the progress is recorded. Once that was done, all the reports written were added and finally there exists a button which when clicked can download the game.

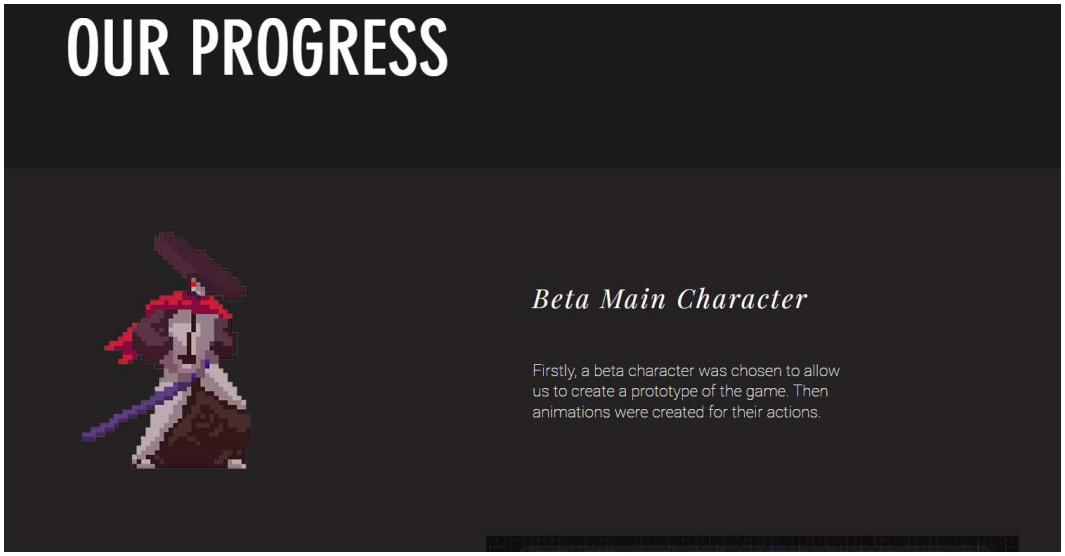


Figure XXVII: *Website Progress Page*

Website link:

https://polloyfaust.wixsite.com/website

3.3 Maxime Badouil

3.3.1 Health Scripts and UI

The first aspect of the game I worked on was the UI in the Main Menu and Restart Menu with Wadhah Ouled Ameur. We created one scene for each menu and set up buttons to navigate the menus.

Figure XXVIII: *Faust's Main Menu*

In the Main Menu shown above, we set up a Start Game button and a QuitGame button. To do so, we had to create an EventSystem that would detect when each button is pressed, and activate the code connected to it. The start game button sends the player to the first level of the game, and the Quit game button quits the program.

Figure XXIX: *Death Screen / Restart Menu*

The Restart Menu was made very similarly to the Main Menu, with an EventSystem, a button taking the player to the Main Menu, and a button sending the player back to the first level.

Once the menus worked correctly, I started setting up a UI containing each character's HP and a button to take the player back to the Main Menu.

Figure XXX: *Faust In-Game UI*

To make the UI appear the same to the player at all times, I created a canvas and set up a health bar and a Return to Menu button. The return to menu button simply takes the player back to the main menu using the function from the Restart Menu.

To make the health bar be connected to the health of the player, I made a slider with a script that would change its max value and current value to make it look like the bar was filling or emptying.

I repeated this process on enemies by giving them their own canvas and setting up an interactive health bar above their head.

Figure XXXI: *Example of an Enemy with a Health Bar*

3.3.2 Network Implementation

My next goal in this project was to fully implement the multiplayer functionalities into the game. To make sure I wouldn't get sidetracked or lost, I divided this task into multiple sections.

3.3.2.1 Documentation

The first thing I did to implement multiplayer was research what assets could be used for free on unity, and were compatible with 2D platformer games. The two assets I found the most fitting for our game were Mirror and Photon.

Figure XXXII: *Logo of Mirror*

Figure XXXIII: *Logo of Photon*

After implementing each asset into the test map individually, I decided to use photon for the rest of the project. This was because although mirror offers the possibility to make larger scale games, photon is a lot closer to what we wanted for a multiplayer asset. A drawback to this decision was that photon tends to be more complicated than mirror in a lot of categories, which required me to do more research when implementing it.

3.3.2.2 Basic Implementation

This second implementation (the first being the one I had done on the test map) was directly into the game. My goal with this implementation was to make all melee enemies work with 2+ players, have all player health bars, UIs and Cameras act independently, and create a Loading Screen / Lobby system to join and create online rooms.

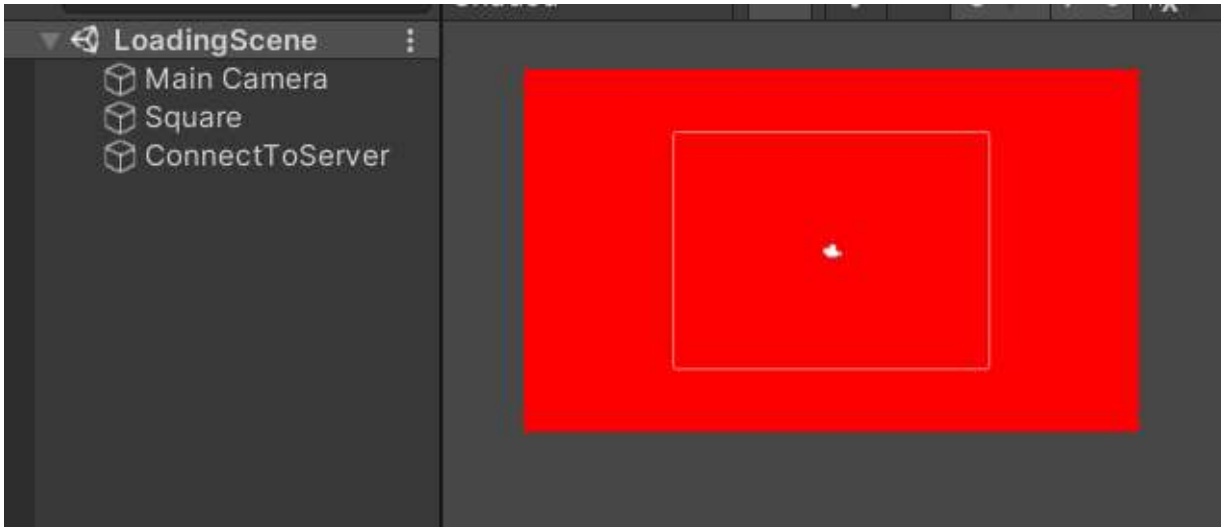


Figure XXXIV: *Loading Scene*

The loading screen is the first scene in our game. This screen allows Photon to detect the player and make sure that he is able to connect to servers when needed.

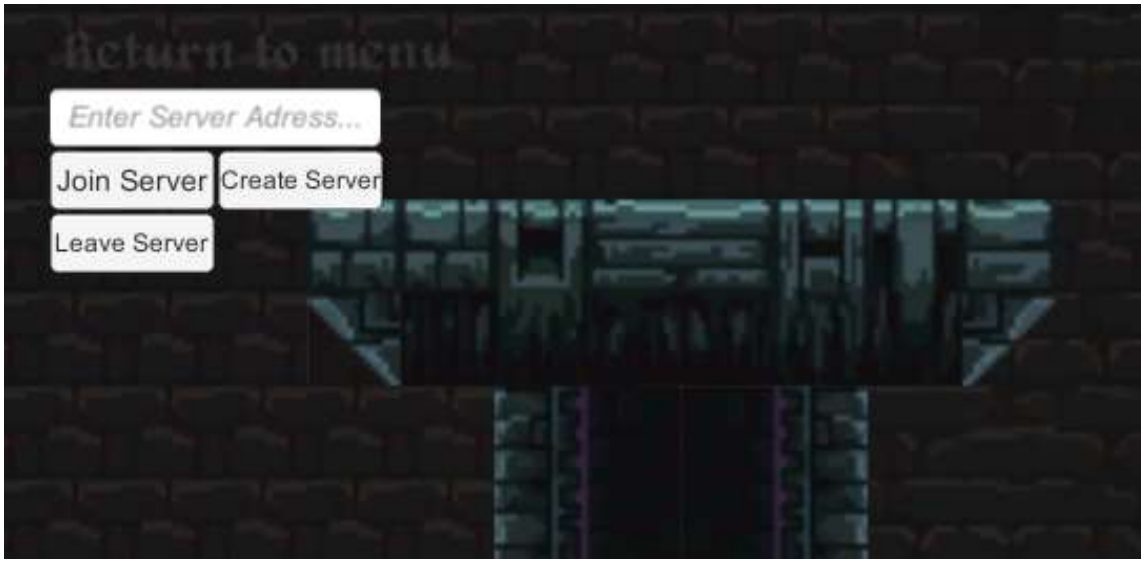


Figure XXXV: *Temporary In-Game Network Options*

At first, I added the buttons to join, create and leave servers inside of the game scenes themselves. However, after reflection, I decided it would be better to give these options their own scene.

Figure XXXVI: *Final Networking Scene*

This scene appears right after the Main Menu, if the player presses create room without text, a solo room will be created where no other player can join. However, if the player adds a key when creating a room, any other player who has the key will be able to join his game by using it.

In this implementation, I also learnt how to deal with synchronization issues. These issues were caused by an enemy/ player/ UI component looking different on the screens of 2 different players. To synchronize UIs and player / enemy HP, I had to modify the previously made scripts and create new ones to make sure that anything that affected one of the players only affected that specific player.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;

Unity Script | 2 references
public class PlayerAttack : MonoBehaviourPunCallbacks
{
    PhotonView view;

    private float AttackCooldown;
    private float RangedAttackCooldown;
    public float startTime;
    public float fireballStartTime;
    public Transform attackPosition;
    public float attackRange;
    public LayerMask enemies;
    public int damageOfSword;

    [SerializeField] private Transform firePoint;
    [SerializeField] private GameObject[] fireballs;


    public Animator animator;

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

    Unity Message | 0 references
    private void Update()
    {
        if (view.IsMine)
        {
```

Figure XXXVII: *Example of changes in a simple script*

To synchronize movement and animations, I gave each object a Photon View component that gave the photon the ability to recognize it. I then gave these objects a Photon Transform View component, which kept track of each object's movements, rotations, and scalings. And finally, for any object involving animations, I had to add a Photon Animator View Component, which translated the animations to all players' screens.

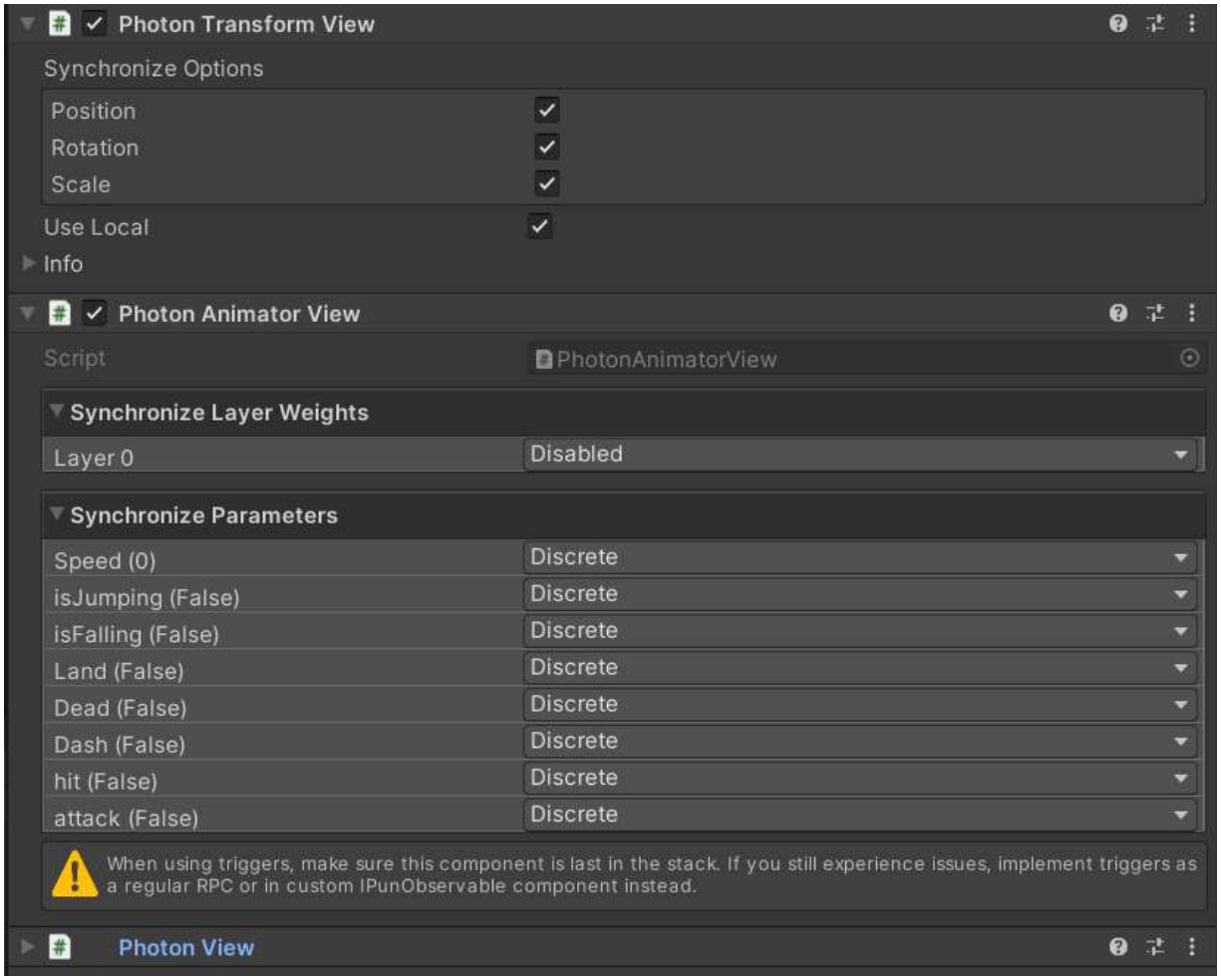


Figure XXXVIII: *Three Photon Components used to Sync the players*

3.3.2.1 Final implementation

This implementation had as a goal to contain the entire project as a playable multiplayer game. On top of the previous implementation, this involved the implementation of Bosses, Ranged Enemies, Books (used to buff the player), doors to change scenes, and cutscenes.

The Bosses and Ranged Enemies were the simplest part of this implementation since I already understood how they worked thanks to the melee enemies, so the implementation of these features was very similar to the previous one. However, extra components had to be added to the ranged enemies' projectiles, to make sure that all players saw them in the same place.

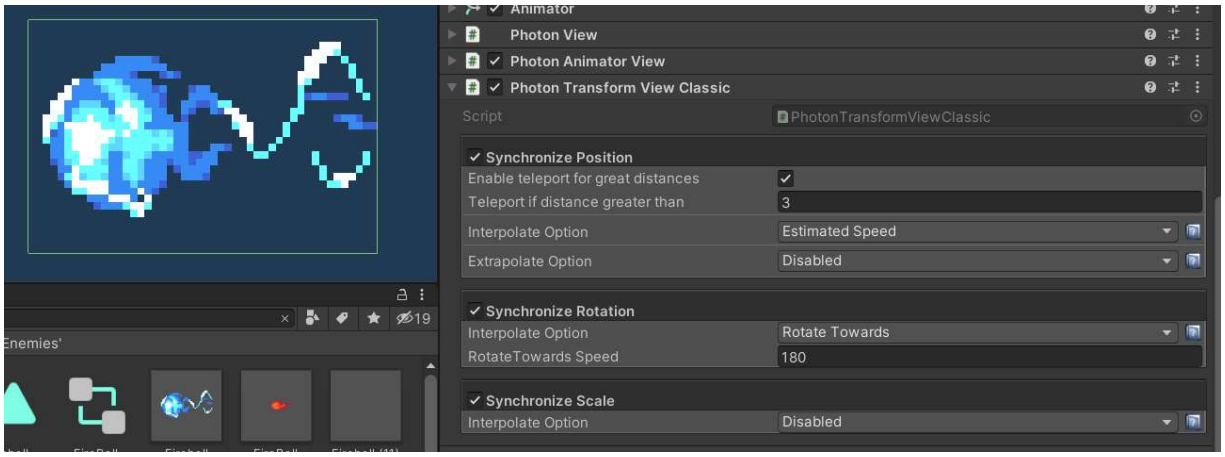


Figure XXXIX: *Components added to projectiles*

Books were more complicated since their hitboxes did not allow me to check what player had touched them, so I did not know how to prevent all players from getting the dialogue. To fix this, I changed what the book had to look for when colliding with the player, allowing it to access the player's PhotonView, and giving me the information of which player had triggered the dialogue.

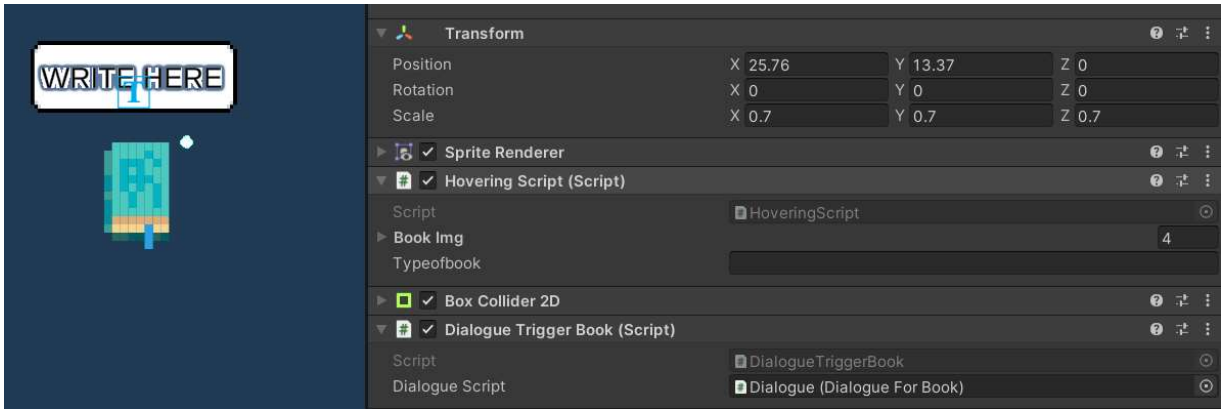


Figure XL: *Book dialogue used In-Game*

The most complicated implementation was the doors. Since I did not know how photons handled scene changes, I had a hard time understanding how to keep all players together when changing scenes and how to preserve their stats.

The main way to change scenes with all players together was to use the functions PhotonNetwork.AutomaticallySyncScene = true, when used in the Start function of the file, this function made sure that when the Host of the Lobby changed rooms, all the other players followed behind.

```
Unity Message | 0 references
private void OnTriggerEnter2D(Collider2D collision)
{
    GameObject collisionGameObject = collision.gameObject;

    if (collisionGameObject.tag == "Player" && Bossdead && PhotonNetwork.IsMasterClient)
    {
        LoadScene();
    }
}

2 references
void LoadScene()
{
    if (!Lastlvl)
    {
        PhotonNetwork.LoadLevel(RandomRoomSelect.lvl1randroom[RandomRoomSelect.Currentroom]);
    }

}
```

Figure XLI: *Part of the code used to implement doors in multiplayer*

This function did not always work at first since it needed the Host to be the one changing scenes, meaning that if another player changed a scene, he would go to that scene alone. The way I fixed this problem was by making it so that when any player reached a door, the Host would be teleported to the next room, which allowed me to keep all players together and keep track of what rooms the party had already gone through. This was done by using the function PhotonNetwork.IsMasterClient (demonstrated in figure XLI);

Figure XLII: *Example of a door to change scenes*

3.4 Wadhah Ouled-Ameur

3.4.1 Map Prototype

The First step in creating this project was to familiarize ourselves with unity. So to better understand Unities Tiles Feature, I created a prototype map.

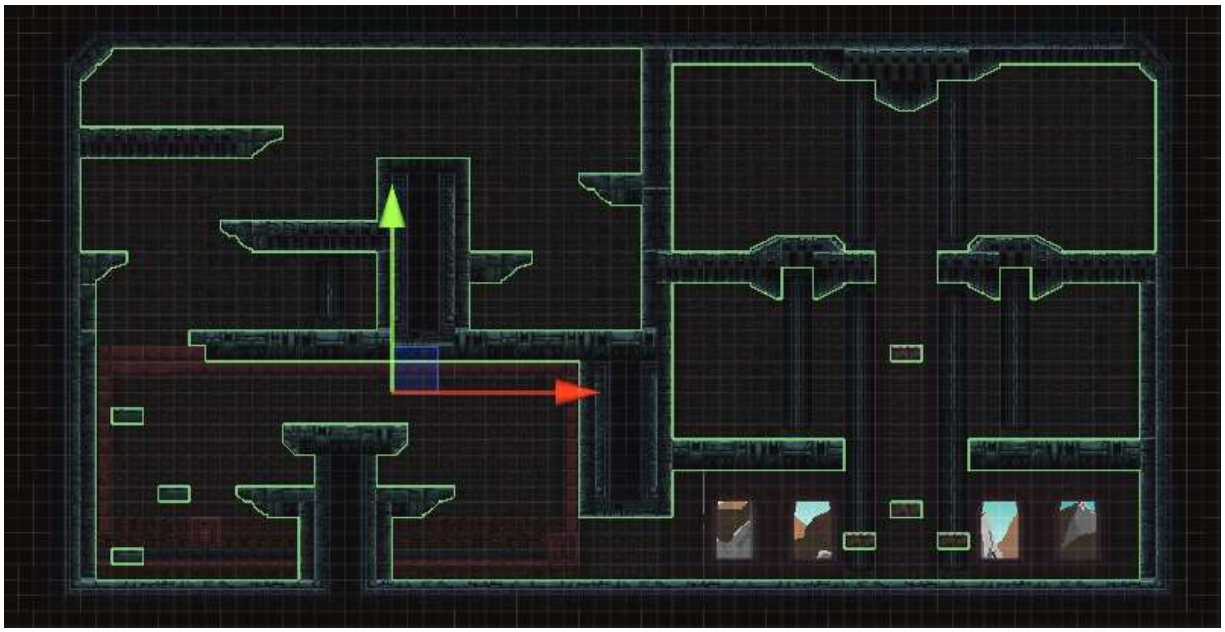


Figure XLIII: Prototype Map

To create this map I first add a background layer and set the background. Afterwards, I added a grid with multiple tilemaps on it, so that one is used for the terrain, one is used for the platforms, and the rest was used for details and decorations. After creating the map, I added the tilemap collider 2D and the composite collider to the terrain tile map so that the player can interact with the map.

3.4.2 Map Creation and generation

One important aspect of rogue-like games is that each playthrough has to differ a little from the one before it, so before creating the maps for the game we first had to agree on a way to implement them while still satisfying this aspect.

At first, we briefly looked at procedural generation, similar to how it's done in dead cells as can be seen in this document[1], but that idea was quickly scrapped due to many factors like complexity and possible complication multiplayer and enemy generation.

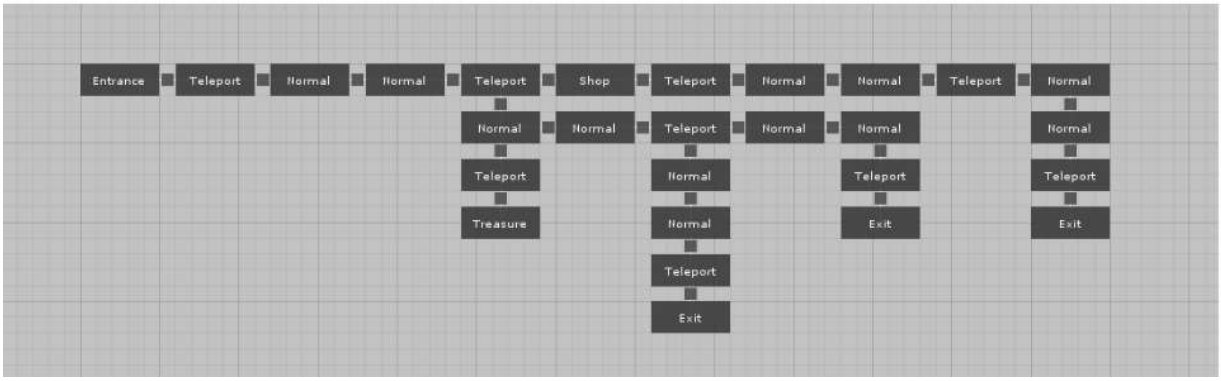


Figure XLIV: Procedurally generated map graph

[1] https://ondrejnepozitek.github.io/Edgar-Unity/docs/examples/dead-cells/

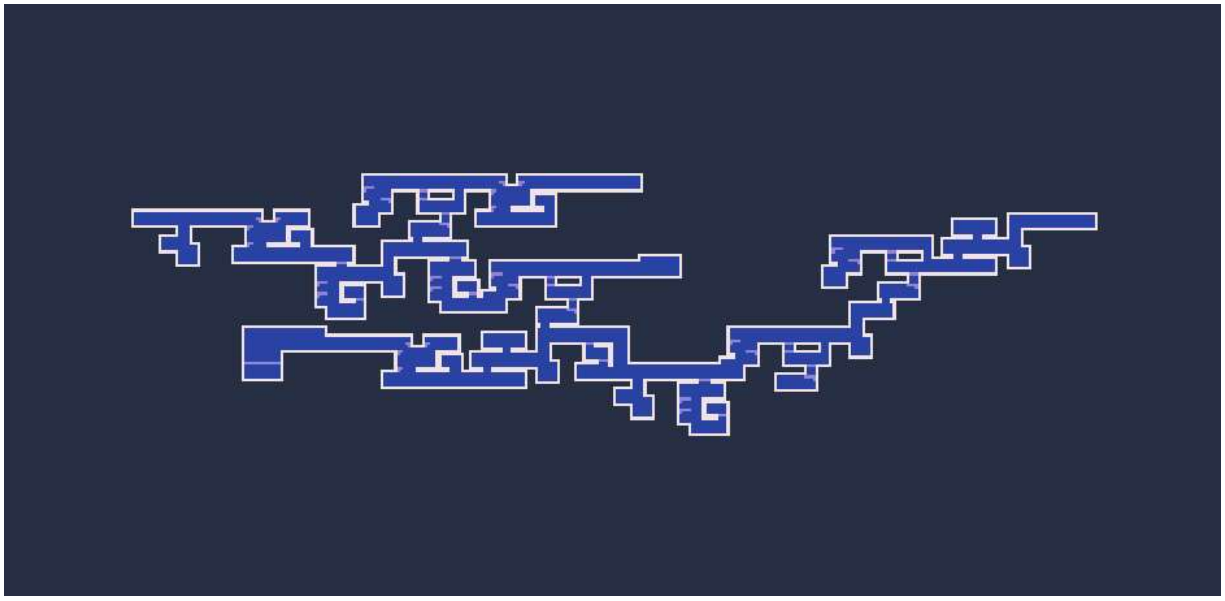


Figure XLV: Example of how the map would have looked like

Another idea we came across was to create a large number of really small "rooms" which we could then generate in a random pattern and connect with hallways. but this idea was also scrapped.

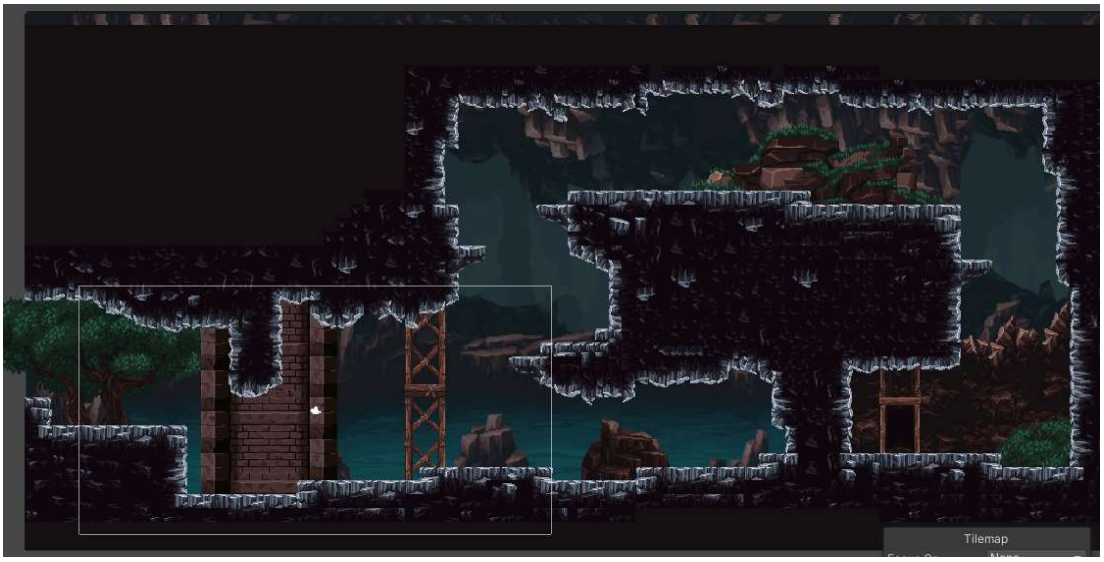


Figure XLVI: Scrapped small rooms design

The final idea and the one that we went with was to design 4 levels with the first 3 levels containing 8 rooms each and the 4th one containing only 2 with every room being built on a separate Scene. So that when the player starts the game 4 rooms from each level are chosen at random and the player would cycle through these scenes when they reach the end and pass through the door.

And with that we managed to create our four levels, those being:

1. The Cave
2. The Deep forest
3. The Snowy mountain
4. Mephistopheles Castle

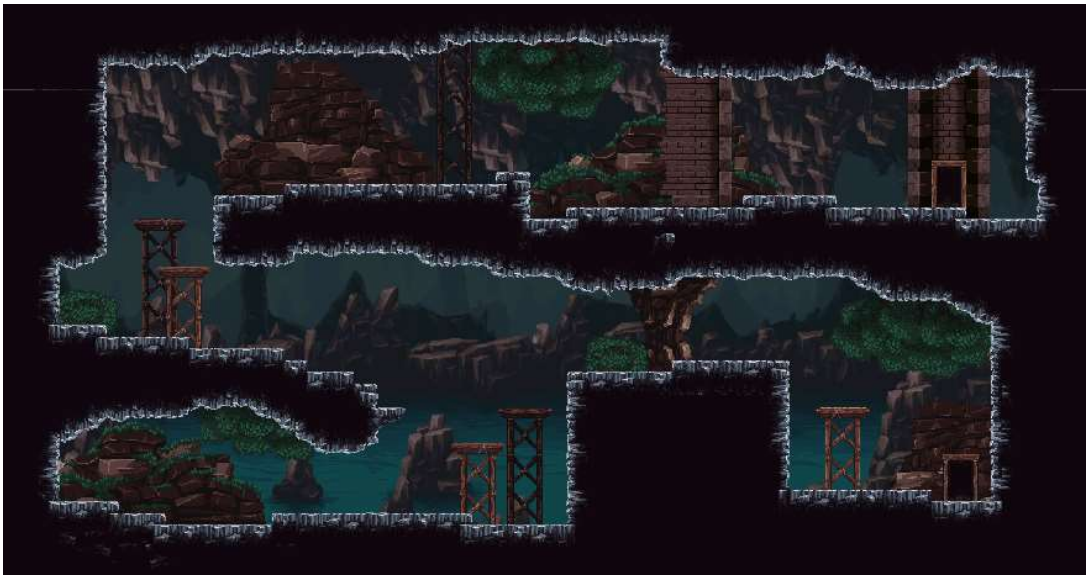


Figure XLVII: Example of one of The Cave rooms

Figure XLVIII: Example of one of The Deep Forest rooms

Figure XLIX: Example of one of the Snowy Mountain rooms

Figure L: Example of one of the Castle rooms

Now that we had our maps we needed to add a couple of features so that they would feel bland, those being platforms and traps.

One way platforms allow the player to pass through them from one direction but stand on them from the other. To create these we used the "Platform Effector 2D" So that when the player presses "Space" or the jump button They would be able to pass through the bottom of the platform but land on the top and when they press the "S" The player will fall through the platform.

Figure LI: Platform mechanism

To stop the player from skipping Boss fights we made it that the game constantly checks for the presence of a Boss in the scene, if the game locates one the doors would not activate, but if no "Boss" enemy was detected then the doors would work as normal.

3.4.3 Challenges with Map Creation

One thing that is important to note is that the map-making process could be an extremely time consuming one due to the combination of unities tiles feature requiring you to place every individual tile separately and the design of the tile maps that we chose to use. so to counteract this I learned to create larger portions and then use them like building blocks.

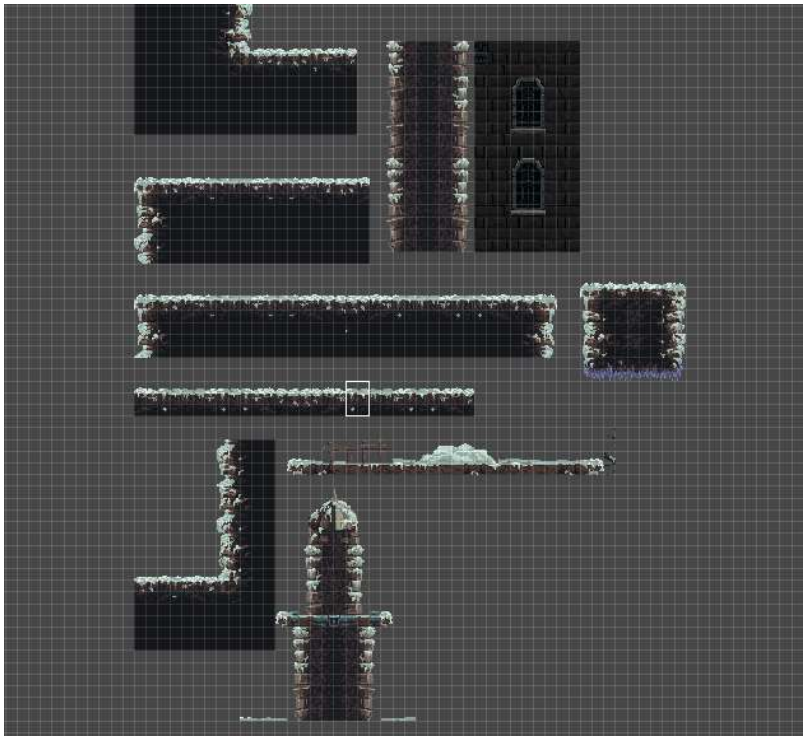


Figure LII: Example of the Building Slices

As we had mentioned in the previous presentation we had an issue with our Unity Collaboration which made it to where we could no longer work together on the same project, So after finishing the maps the first time the project file on my computer got corrupted and we had lost all the progress that we made with map design which meant we had to start from scratch when it came to the levels. From this point onward we made sure that we created backups regularly to prevent this from happening again.

Chapter 4

CHALLENGES

4.1 Unity Collaboration

At the start of the development of the project, we used Unity Collaborate, this in itself took a few weeks to get used to. For example, on many occasions we encountered problems while syncing which led to days worth of work to be overwritten by the Unity collab feature. However, once we finally understood how to use Unity Collaborate, the project was migrated to a new way of collaboration called Plastic SCM. We were forced to use Plastic SCM because the previous solution (Unity Collab) was disabled on the project.

Therefore, we had to familiarize ourselves with the new application, and while we tried, we were not able to make it work. Due to this we were forced to use the “EXPORT” and “IMPORT” options in Unity Project. This meant we would each work on out section and once finished would import the specific files that were edited and send them to the others in the team to import.

However the problem with this method is that sometimes the settings of characters would change, which would in turn mess up the entire project. We would then have to do tedious work all over again.

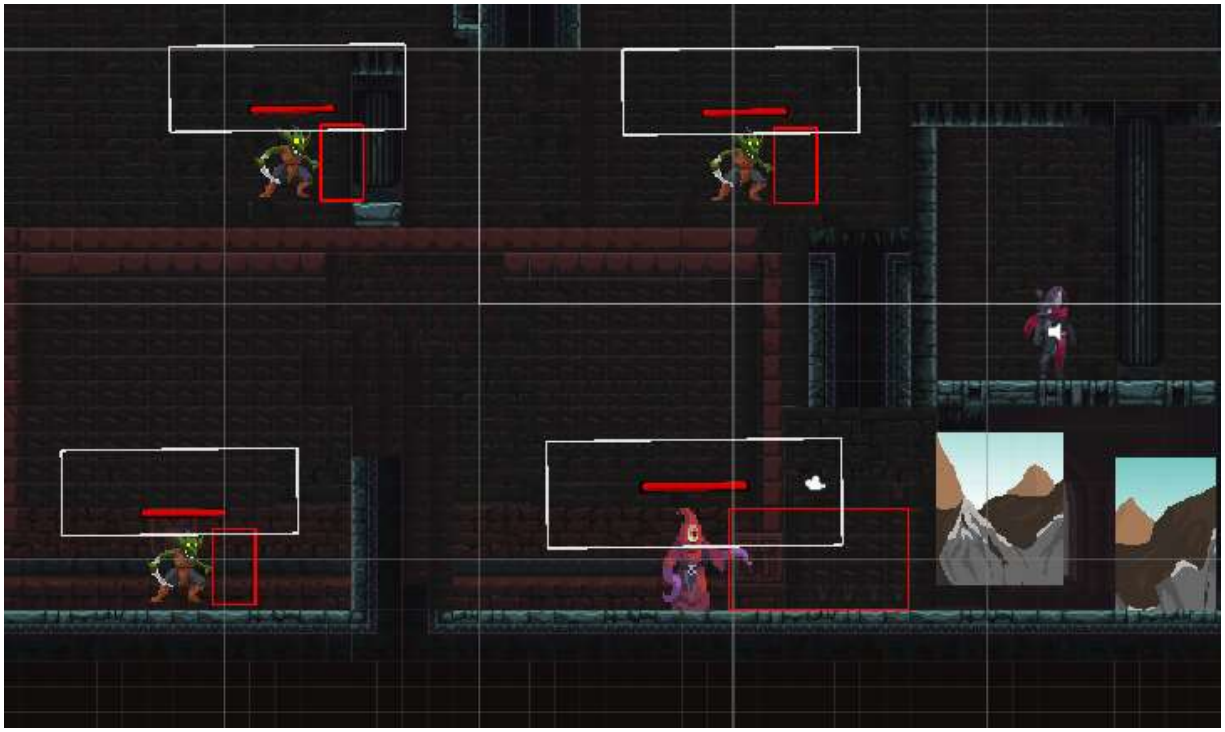


Figure LIII: Map with missing blocks

As you can see in this figure above, some of the enemies seem to be floating, however the reason is the actual floor layer is under the background layer. This was caused due to change in settings due to the EXPORT and IMPORT method.

Chapter 5

APPENDIX

Figure LIV: Prototype of the Player

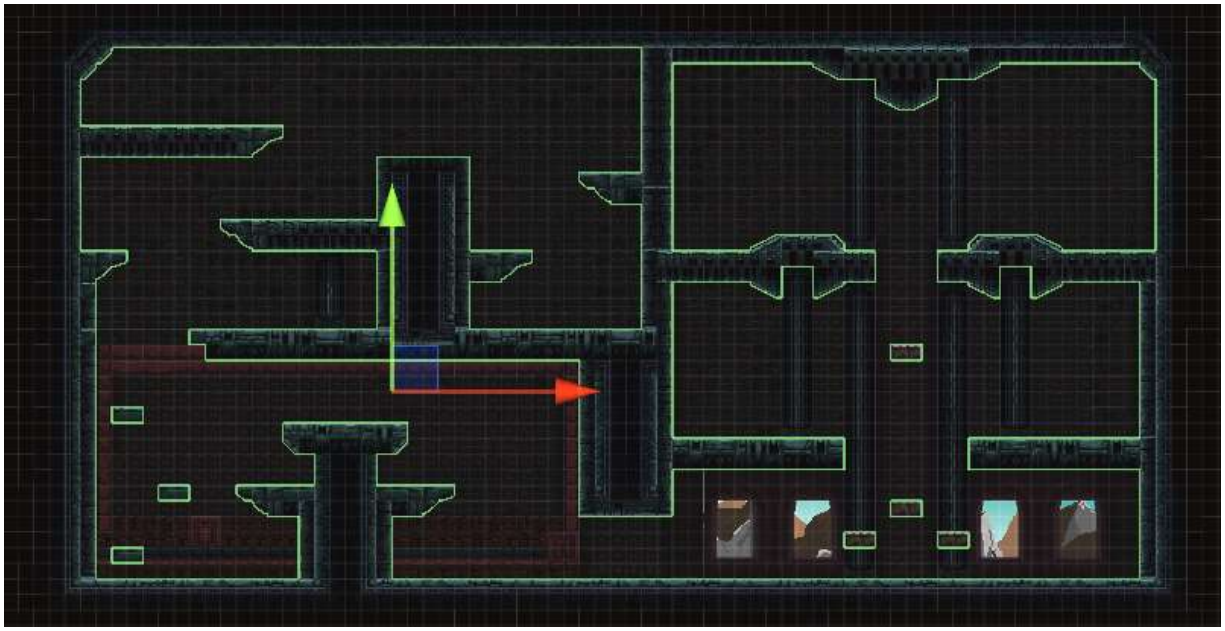


Figure LV: Prototype of the maps