Second Defense

Incroyqble

April 19, 2023

Abstract

This document covers what has been achieved during the second part of the project. We'll discuss how we handled work, separately and in a group, cover what challenges we've faced, how we tried to improve from the first defense, as well as what we hope to achieve during the following period.

Contents

1 Introduction

As we present the second defense of this project, we cannot help but feel proud of the progress we have made throughout these two months. Each member of the group has worked tirelessly to ensure that the game is as enjoyable and immersive as possible. From creating the initial concept to designing the maps, implementing multiplayer and player essentials, enhancing graphics, using Blender to design advanced objects, implementing the mobs, and developing the quest system, we have carefully planned each step of the way.

We have faced many challenges along the way, but we have also learned so much. Balancing the difficulty of the quests and mobs was one of the biggest hurdles we faced, as we wanted to create a game that was challenging enough to be engaging, but not so difficult that players would become frustrated. We also had to navigate the sometimes confusing and ever-changing landscape of Unity, adapting to new features and systems as they became available.

Despite these challenges, we feel confident in the progress we have made. Each member of the team has brought their own unique set of skills and ideas to the project, and we have all worked collaboratively to ensure that every aspect of the game is as polished and well-designed as possible. From the intuitive user interface to the detailed maps and challenging mobs, we believe that we have created a game that is both fun and engaging for players of all levels.

As we move forward with this defense, we are excited to show off the progress we have made and to share our vision for the future of the game. We have already begun to plan the next steps in the development process, including implementing new mobs and expanding the quest system to cover a wider range of situations. We believe that this project has the potential to be something truly special, and we are committed to working hard to ensure that it reaches its full potential.

2 Alexandre

2.1 Planning

In this part of the project, I was tasked with developing new levels, with the help of the tools I created and the individual pieces given by my colleagues. I also decided to try and optimize our team work.

- Fixing PokeZelda
- Developing MarioMap
- Website
- Colliders Update
- Project organization

2.2 Achievements

2.2.1 First Defense Fixes

After the first defense, I dedicated several weeks to creating a viable way for the team to track our progress and making it as easy to use as possible for the other working team members. I also spent some time trying to fix several issues I was having beforehand.

2.2.1.1 Github Project

I quickly discovered the "Projects" of our Github repository, and realized it's potential. I spent a few days learning how it could help the team organize itself, and how I could organize the now called "Develop" Project. At first, I worked with the List view, seen in Figure 1, that allowed us to easily see the status of the opened issues.

Status

When creating a task on Develop, it can either be a "Draft", or an "Issue". I realized that creating issues allowed us to assign tasks to working members, as well as discuss any troubles we were having. I could also create more "Status" labels to help organizing, as well as creating date field. Those reasons convinced me to use "Issues" instead of "Draft" for the project.
I first updated the labels to help us have a better understanding of the state of our work. We're currently working on five statuses:

- ToDo: list of every feature that needs to be worked on in a near future.
- In Progress: when a member starts a task, this status let others know that it is being taken care of.
- Late: If a task is overdue (a month and a half for some), or a member feels like he needs help on this task, the issue is moved there.
- Done: Tasks that have been completed, but still needs to be merged.
- Merged: Tasks that have been merged to the `develop` branch.

Deadlines

Once statuses were done, I implemented a way for us to set deadlines and try and follow those. This was done by adding two `date fields` to the project. This allowed for the creation of a roadmap view, as seen in Figure 2:

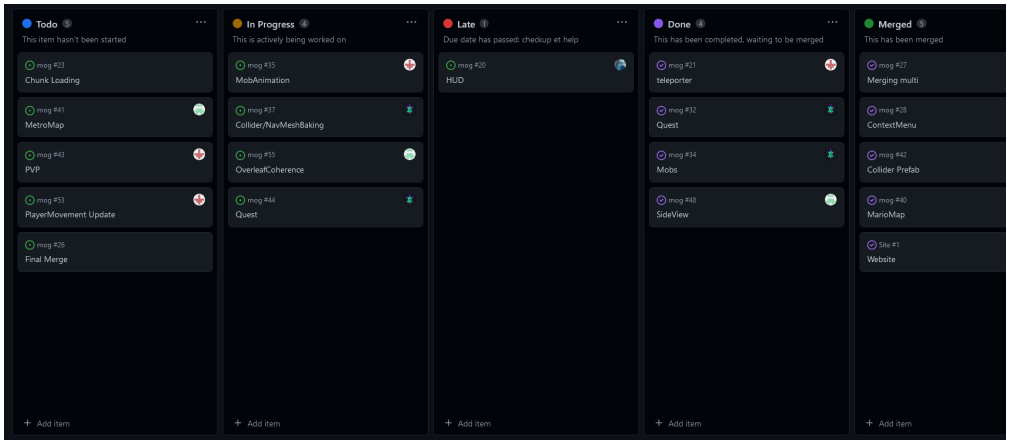

Figure 1: List view of the project.

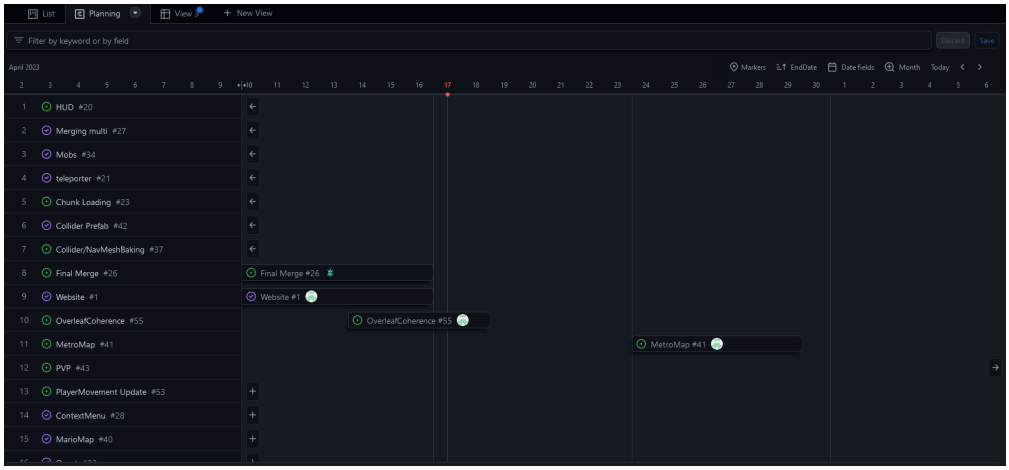

Figure 2: Roadmap view of the project.

Workflows

Projects offers a really awesome feature of automating task management through so called `workflows`. I quickly thought about a few rules that could be useful:

- Branch Creation: once an issue has been moved for "ToDo" to "In Progress", a branch,bearing the name of the issue, should be created.
- Late automation: moving each issue in the "In Progress" category to "Late" if the end date has passed.
- Pull Request Creation: creating a pull request once an issue was moved to "Done".
- Pull Request Validation: moving an issue to "Merged" once a pull request has been completed.

Unfortunately, none of these workflows existed by default, and creating them required a lot of specific knowledge that was not deemed useful enough to justify not progressing on the actual game.

Team conversion

It also seems that, despite my efforts to make it as easy to use as possible, some of the members didn't find it as useful as me. It then was primarily maintained by me, but allowed for regular checkups on members to know if I needed to update anything.

The project option remains, and will remain, only a tool to visualize our work, and not a dynamic organization helper.

2.2.1.2 Tilemap Fixes

As explained in the previous report, the "PokeZelda" tilemap suffered from inconsistent tiling and tearing. After some digging, I found that this issue was caused by two factors.

Tile slicing

When reworking on the tilesheet sprite, I realised that the tearing I observed came from the "Wrap" and "Filter" modes I selected. Once changed to "Clamp" and "Point", respectively, the tearing issue was much less apparent.

Tilesheet

Since there was still some visual issues that were not reproduced on the other tilemaps, using the same parameters for the spritesheet processing, I realized that it came from a default in the tile's placement on the sheet. Some tiles were slightly larger than there supposed 16*16 pixel size, and some were offsetted, which was causing the discontinuity in the paths for example (Figure 3). Since this

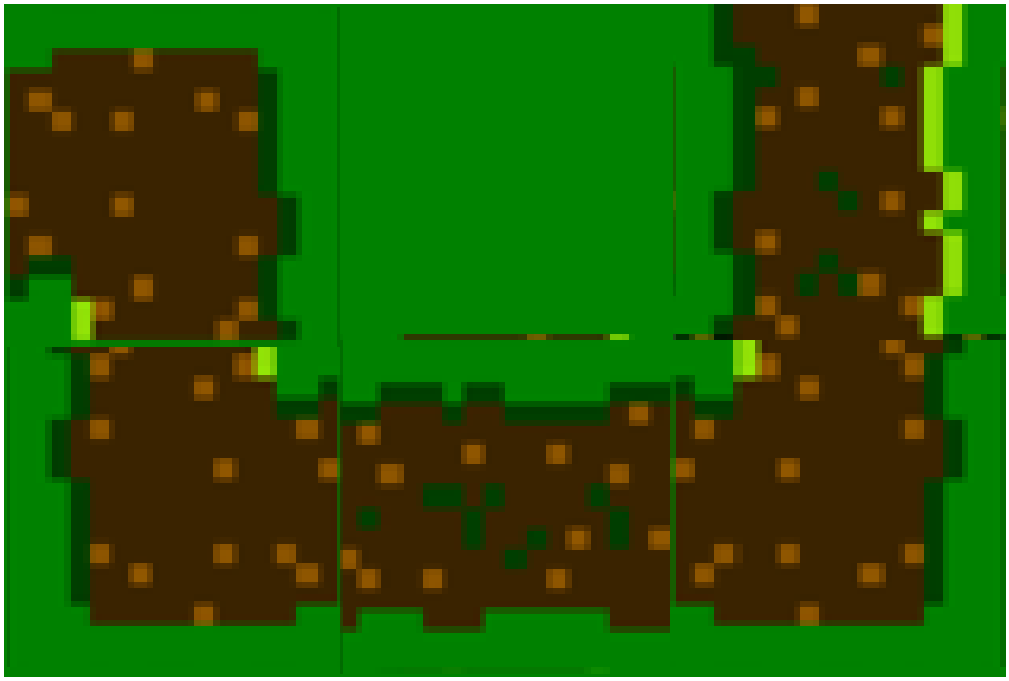

Figure 3: Examples of path discontinuity

particular tilemap has take a significant amount of work to achieve, they'll be no further work to fix the visual issues caused by the sprites. However, since this issue is now known, we will be more careful to which spritesheets we select.

2.2.1.3 Collider Fixes

Collider Instantiation

In the previous version of `TilemapColliderGenerator.cs`, each time a scene was started, a script would handle the creation of every collider, setting it's position, properties, ... In this iteration, a `public GameObject colliderPrefab` parameter was added, in order to instantiate every colliders, rather than create them each time. This allowed for a much easier modification of said colliders, since they now came from a Prefab, but also allowed me to create other types of colliders that could be used by the same script.

Game optimization

To further optimize this process, we decided that the script should only be ran once during the developing stages of the map, and the colliders, created as children of the `GameObject Colliders`, should be exported to the hierarchy when the game is not being ran. The first ideas, mainly using a context menu, or programmatically saving these assets, were quickly replaced by the much simpler method of copy-pasting the Objects from the GameView of the ran scene to the Hierarchy.
I then realized that keeping every single collider on the game would make it quite space inefficient. To further optimize, I implemented a condition that would allow the creation of colliders only if there was at least a null tile in the immediate cross proximity, as seen in Figure 4.

Figure 4: Example non-generated colliders on tiles surrounded by other tiles

This proved to be particularly useful in the "Mario" Map, where the number of colliders was cut in half saving quite some space for the user.

2.2.1.4 System View

The bugs of the "side" option of `CameraFollow.cs` was not fixed, as Damien decided to handle the view system differently for the maps. It will not undergo any more development.

2.2.2 New Maps

In order to showcase the Teleporter Object created by Damien, we needed another map to be teleported to. I decided to revisit one of my favourite game growing up: Mario. To add a feel of our own, I strayed a bit from the original scenery and also decided to create my own 4*4 spritesheet, seen in Figure 5.

I created quite an array of combinations for the rule tile, effectively covering 30 to 40 cases, out of 64. This exhaustiveness made it quite easy to paint the tilemap, and will make it even easier to create "random" maps if we intend to use procedural generation to extend this map.
I also designed pipes and spikes for this level, that are used in combination of the `InMapTeleporter` Prefab to allow for respawn when a player falls, or an implementation of working Mario-Like pipes.
Contrary to the "PokeZelda" map, this one has only two tilemaps: the "GroundTiles", parent of the

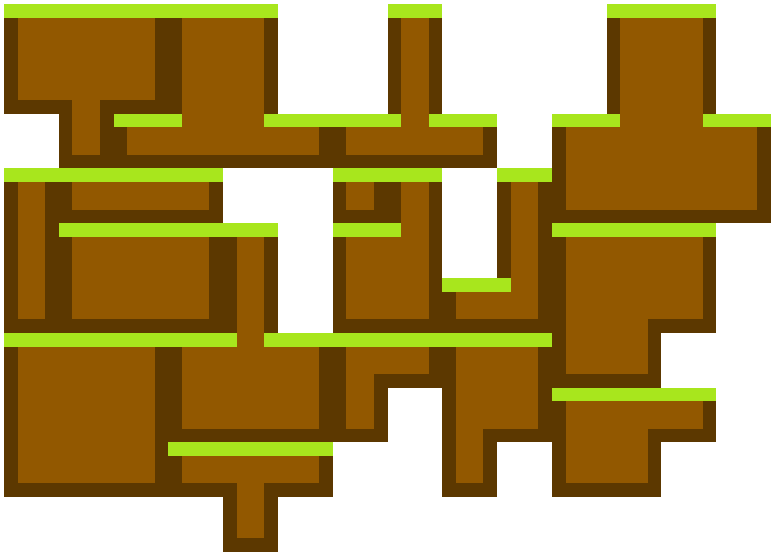

Figure 5: Custom spritesheet

colliders on which the player will move, and the "Teleporters" one that holds all the InMapTeleporter. Depending on how backgrounds are handled, there might be a third one.

2.2.3 Music

Our current choice of theme music goes to "Jet Force Gemini OST - Funky Disco (Big Bug Fun Club)"[1], as it communicates both retro and adventurous vibes. This music will probably be constantly active during a game.
For the other, more instantaneous actions, we will need to find a good retro music source.

2.2.4 Communication

2.2.4.1 Report

As part of the report's collaborative process, each member was responsible for redacting their own section, as evidenced by the use of the first-person singular pronoun "I." While each individual contributed their unique perspective and expertise, it was my role to oversee the compilation of the various sections into a cohesive whole. In addition, I had to create and organize the necessary commands for the group, such as `subsubsubsection`.

2.2.4.2 Website

During this second part of the project, I had more time to work on creating a simple, yet pleasant website, accessible through https://incroyqble.github.io/Site/index.html. Although my Web-Development days are starting to fade and fade, I was still able to create a structured website that would give all the necessary information to users. it has the following pages:

- Home: general information on the game, mainly extracted from the Book of Specification.
- Team: a short presentation of the working members of Incroyqble.
- Resources: a link to download our reports, and eventually the assets used in the game.
- Contact: a link to the Incroyqble's github page.
- Download: where any user will be able to download both normal and light version of the game.

[1] Youtube link: 'https://www.youtube.com/watch?v=9jJ2tXLJ_0g

The website is being hosted by Github pages, allowing fast, reliable and simple publishing of our work. It is currently online, and accessible through an URL, but not through text search, as we felt it wasn't necessary to have it referenced.

2.3 Future Progress

2.3.1 Bug fixes

2.3.1.1 InMapTeleporter

This `GameObject` seems to currently have some troubles positioning the player at it's input position. We still need a lot of testing to figure out how it could be bettered.

2.3.2 Maps

2.3.2.1 MetroidVania

One of the game mode we're really keen on having is a MetroidVania inspired game. Since it seems to be a quite vast genre, I need a lot of time to familiarize with each of the rules of this style. The creation process will be done in three steps.

SpriteSheet

The first step is finding a viable spritesheet. By viable, I understand one that will need the bare minimum of processing to be easily usable by Unity. This includes perfect edge alignment, and constant tiles sizes. It also needs to have a quite peculiar touch to it, as I refuse to create my own assets for a world this complex, but still would prefer not working with a sloppy artwork. The current top contestant is a s black and white sheet that would create an interesting contrast with the otherwise bright worlds.

ProMeLaGen

To help with the level design, I decided to turn to Logan's "Procedural MetroidVania Layout Generator" on itch.io. This tool will allow me to inspire myself from various level designs, in order to create the atmosphere that will make this level so enjoyable. However, this tool is not compatible with Unity, and will be more a source of inspiration rather than an actual building tool.

Populating

Once the level's architecture has been completed, and painted, the last step will be to populate that map with the portals, create the quest, add mobs, ...

2.3.2.2 Minecraft-like

As the end of the project approaches, we need to start working on the final 3D map. Fortunately, several minecraft replicas exist, and might be usable in our case.

2.3.2.3 Side backgrounds

At this moment, a side map only has the "interesting" tiles to it, meaning we can see the default environment of Unity behind. We'll then need to find a way to create a background image for each side map, by either using a canvas behind the map to hide the world, or simply painting different tiles to the grid.

2.3.3 Website

Currently, the website holds all the information that it is required to. Nonetheless, I would like to include a more in depth analysis of our work, but also showcase actual gameplay, extracted from the game, to showcase it better.

3 Damien

3.1 Tasks

In order to fix the various issues we encountered I had to deviate from some of my original tasks in order to endorse new, more diverse roles within the project. These changes led me to be the multitasking "expert" of the group, and to be assigned the following tasks :

- Multiplayer implementation
- AI
- Weapons
- 3D modeling & animations
- Player's abilities/ mechanics (player's inventory & death system)
- UI menus implementation

3.2 Multiplayer

3.2.1 Engine

For this defense we had to change the packages we were using to implement the multiplayer. Previously, we were using `Photon Fusion` which turned out to be unsuitable for the plans we had for our project. Indeed `Photon Fusion` had several limitations, mainly due to it's implementation of multiplayer using peer to peer, which prevents the implementation of a room system. Additionally `Fusion` also had the disadvantage of having very close to no documentation. Those two factors combined hence led us to switch to `PUN 2` (Photon Unity Networking) which seemed far more suitable for the room system we wished to implement.

3.2.2 Room System

As stated previously a game will take place in virtual rooms, where players can join, interact, and compete with each other. Each room serves as a self-contained environment, with a shared set of rules, objectives, and gameplay mechanics. Truly we wanted to implement our multiplayer in this way as having the ability to race anyone connected, without the need for peer to peer connection, provides an easier way to connect and therefore to play games with others.

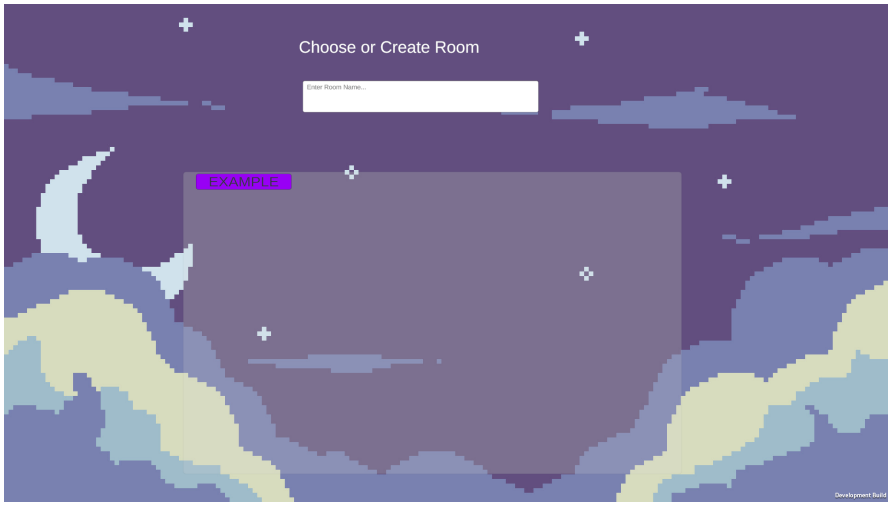

Figure 6: Room Selection Scene

3.2.3 Synchronization and Script execution

Since we are using Photon PUN's networking system, there is two types of game object to synchronize in the game: room objects and personal items. Room objects are objects that are shared between all players in the room, such as mobs. Personal items would be objects that belong to individual players, such as weapons, cameras, or inventory items, also called keys.

When a player interacts with a room object, such as by getting chased by a mob, the script for that object is executed on the master client's side. This is because room objects affect the game state for all players, so it's important to have a single source of truth for their behavior.

On the other hand, when a player interacts with a personal item, such as equipping a weapon, dying, attacking, the script for that item is executed on that player's client side. This is because personal items only affect the game state for that individual player, and it's more efficient to have each client handle their own items.

To ensure that all clients are in sync with the game state, the clients periodically send updates to each other with the latest information about the game objects and events such as each player's movements.

3.2.4 Main Difficulties

3.2.4.1 Documentation

Although PUN is more documented than Fusion, the main challenge that I faced when using Photon PUN is the lack of documentation available for the platform. Indeed, Photon PUN documentation can be incomplete, outdated, or confusing, making it difficult to understand how to use the framework effectively. This often led to frustration and wasted development time as I tried to navigate the framework without clear guidance.

The most common issue that arose due to the lack of documentation was the confusion generated around the various network components, and their functions, but mainly the various options they might have. This issue led to hours of debugging, disproportionate of the easiness of the fix, which was simply to tick a specific box in a specific component.

3.2.4.2 Conversion of the game to fit the Multiplayer

Another difficulty was implementing the various maps of our game with the multiplayer, indeed I originally wanted to implement each map as "sub-rooms" of a room in order to have each sub-room behaving independently from each other. However once the system was implemented it turned out to be limited by the locked Time-To-Live of empty rooms which lasts for a few minutes at most which wasn't specified in PUN's documentation. The system was hence changed in order to have all maps on one single scene, but different grids and tilemaps.

3.2.5 Future Progress

Our multiplayer system has been performing well and we do not anticipate any major changes to its functionality. Although there most likely will be some improvements made to the lobby room selection scene

3.3 Weapons

I have completed the development of four out of the five final weapons for our game. These weapons include the sword, bow, bomb, and fireflower. All weapons were either created or reworked from the ground up since the last defense, the exception being the bow.

The sword is a classic melee weapon, providing players with a close combat option. The bow, on the other hand, offers a powerful ranged option that allows players to attack their enemies from a safe

distance. The Zelda themed bomb is a powerful explosive that can be used to deal massive damage to groups of enemies as well as destroy some specific `GameObject` s, while the Mario themed fireflower provides a mid-range attacking option.

3.3.1 Sword

The sword now works by detecting the collision between it's 3d model and potential enemies which is a major improvement from the previously used killzone.

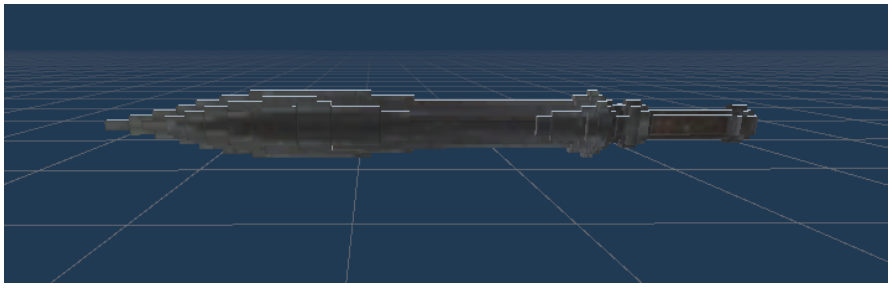

Figure 7: Generic Sword 3D model

3.3.2 Bomb

The Bomb's damage area now works by instantiating an explosion animation which listens for collision to either damage an enemy or destroy destructible scenery.

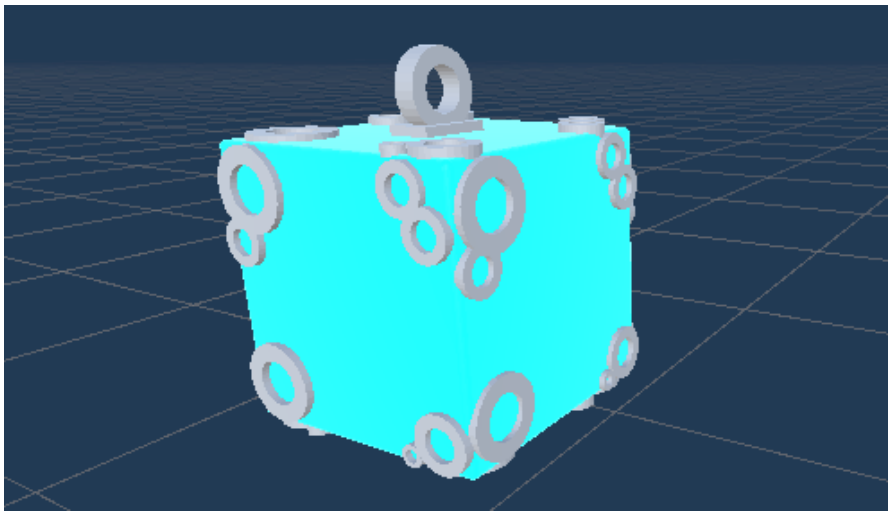

Figure 8: Zelda-themed bomb 3D model

3.3.3 FireFlower

This weapon is the Mario-themed fireflower, it summons a fireball that bounces when it hits the ground and destroy itself when it hits something. Upon collision if the `GameObject` it collides with is an entity then it deals damage to it, and if it is a burnable scenery, it destroys it.

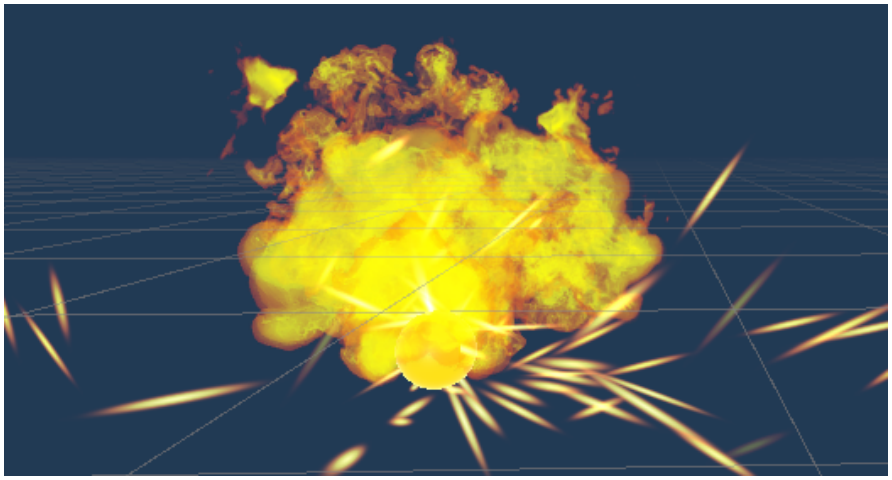

Figure 9: Fireball

3.3.4 Main Difficulties

It was mainly difficult to create, manage and synchronize multiple layers of animations on both the objects and the players at once, both locally and across the network. Indeed some animations needed to be played simultaneously for instance when the player moves and charges the bow at the same time. Such precision in animation was harder than expected as `PUN` does not inherently supports the synchronization of methods such as `Play()` and animation events

3.3.5 Future Progress

For the end of the project I intend to develop the only main weapon that is left out of the five. However I also intend to develop the multiple player abilities that will be linked to each individual weapons for instance the ability to throw bombs instead of simply dropping them, to shoot specific types of arrows that will have specific effects on the enemies (slowing them down for instance).

3.4 Player's Essentials

3.4.1 Inventory & Inventory HUD

Another new feature is the inventory. This was done by implementing an array to store weapons and keys that the player has picked up. These are visually displayed in the player's inventory UI with weapons being visible on the top right and keys becoming visible on command by pressing "R". A weapon can be added to the inventory by instantiating a visual representation of the weapon's sprite and setting its parent to the appropriate weapon holder in the inventory UI, as well as adding the weapon into the inventory array. A similar process takes place if a key is picked up instead.

The Inventory classes also listens for the player's change of weapons which also triggers a change in the HUD's selected weapon. While weapons are pretty self explanatory, allowing the player to deal damage, the keys are used to access the final map.

In the case the player dies, his weapon inventory is kept as is, but the keys remains lost, to force players to use PVP.

3.4.2 Death System

The death system implementation works as follows: when a player's health reaches zero, the game creates a "Grave" object and disables the player's `GameObject` . The **Grave object** is used to store the player's inventory keys, which are cleared from the player's inventory. Any player in the game can then touch the grave in order to receive said keys. The player's current weapon is unequipped, and a "currentDeathCam" camera is activated, which displays a death screen to the player.

When the player clicks the respawn button, the death screen disappears, the player's health is set to maximum, and the player is teleported to a spawn point while their inventory and current weapon are kept untouched, except for the keys. Finally, the player's `GameObject` is re-enabled.

3.4.3 Teleporter

I worked on two types of teleporters : The the first type of teleporters I worked on allow the players to switch to a random map among every map except the final one ("The End") and the one the player is currently on. There will be one of these teleporter on each map allowing the player to hunt his enemy as well as keys on all maps.

Th second type of teleporter is a unique teleporter that requires all the game's keys in order to teleport the player to the end map.

3.4.4 Main Difficulties

The main difficulty here was the synchronization of the death system for all clients as it required the synchronization of not only the player's `GameObject` state but also the synchronization of the grave's inventory.

3.4.5 Future Progress

The player will most likely gain the ability to switch the weapons of slots and to see what mode his weapon is in.

3.5 Graphics

3.5.1 3D Modelling & Animating

Throughout my time working on this project, I've continued to work on 3D modelling, particularly on creating models and animations for four Pokemon-themed mobs. I've therefore put in the time to create these models, making sure that they are as accurate as possible to the source material and fully animated to make them come to life in the game world.

Unfortunately, some of those models are only meant to be options, and will most likely never be used in the game.

- Charmender
- Poliwrath
- Gengar

In addition to modelling mobs, I also designed all of the 3D models for the weapons, displayed from Figure 7 to Figure 9.

3.5.1.1 Charmender

This Pokemon is meant to be the Pokemon world's "Weapon". It was necessary to choose a relatively small pokemon, in order not to bother the player, and I was therefore charged to make and fully animate its model

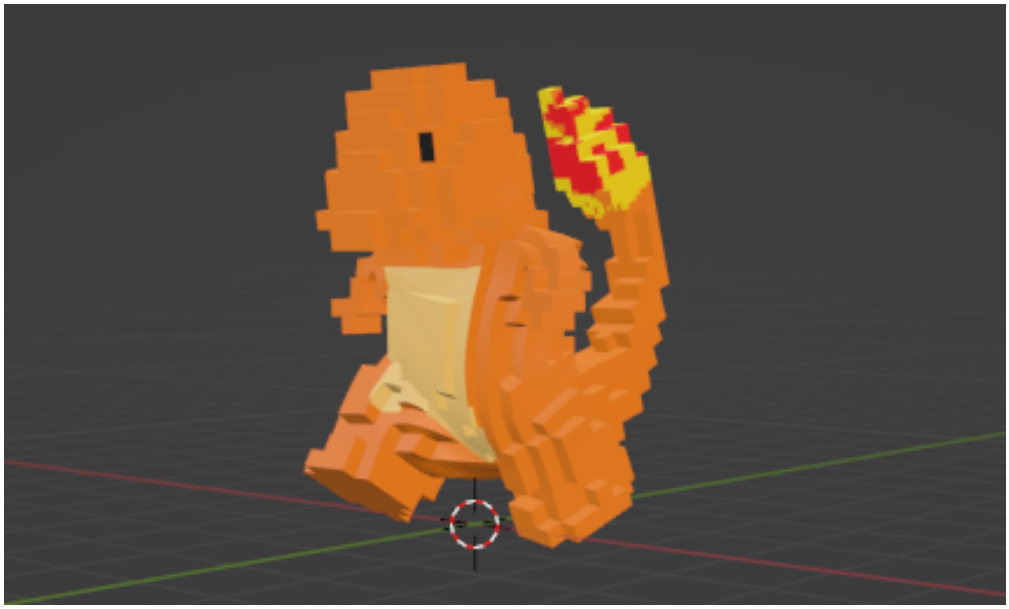

Figure 10: Charmender3D model

3.5.1.2 Poliwrath

This Pokemon is meant to be the Pokemon world's "default enemy" as it perfectly fits the the default melee mob behaviour our default AI implements.

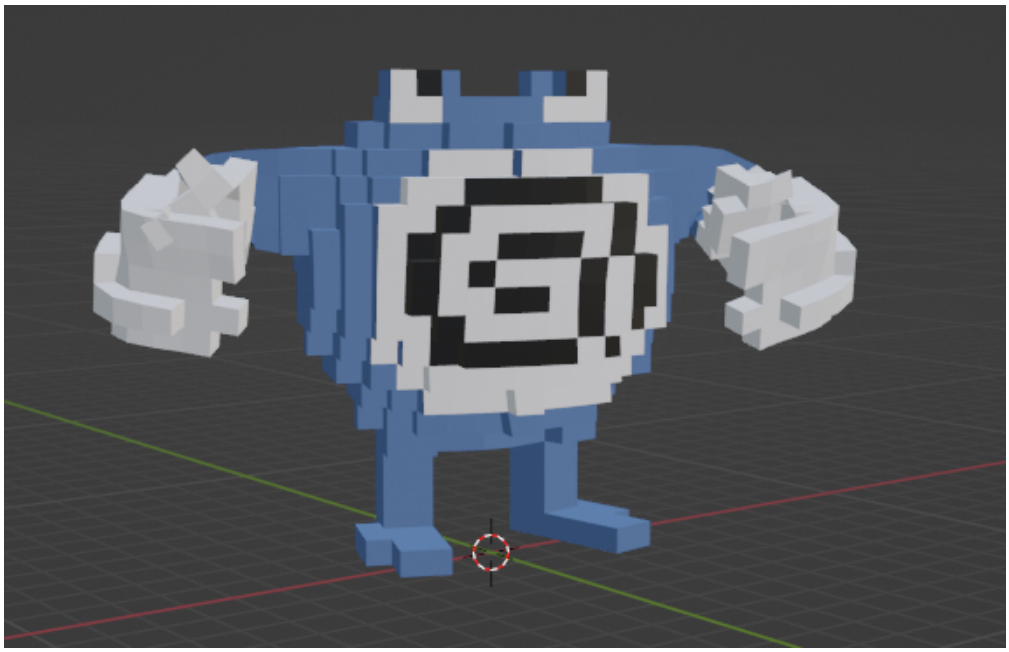

Figure 11: Poliwrath 3D model

3.5.1.3 Gengar

This Pokemon is meant to be the Pokemon world's ranged enemy as it will throw a projectile.

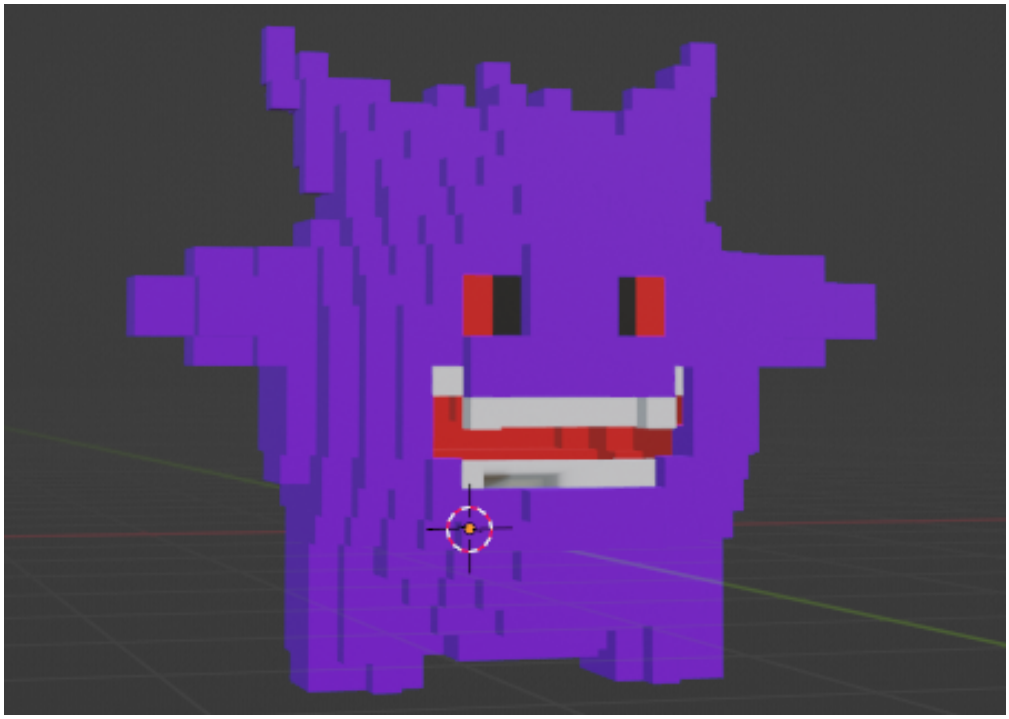

Figure 12: Gengar 3D model

3.5.2 Finding and Installing assets

Additionally to making assets I also scoured the web in order to find available ones. This was particularly useful for the weapon's animations and abilities. Indeed, although I am now familiar with making 3D models, I still haven't mastered the conception of particles and VFX graphs which are essential for the reproduction of the various assets our game's items take inspiration from.

3.5.2.1 Universal Render Pipeline

Changing the render pipeline from our previous one to a Universal Render Pipeline (URP) has allowed us to achieve a greater degree of versatility and customization in our graphics rendering. The URP is designed to be highly customizable and modular, making it easier for us to tailor our rendering to the specific needs of our project.

One of the benefits of the URP is that it allows us to easily import assets that were previously not possible. For example, we were able to import a Zelda-themed explosion asset into our game.

3.5.2.2 Integrating Assets

When searching for particle assets to use in Unity, I looked into the numerous options. It was time consuming to find exactly what we needed as our vision was extremely specific. However, by looking into various videos as well as githubs I was able to import a lot of those much needed assets.

3.5.3 Main Difficulties

The main difficulty I faced while modelling was that some blender files got partly corrupted which led to partial animations that had to be remade. The fact that some of these models probably won't make it into the game also means that many efforts were wasted.

Another difficulty was integrating the new render pipeline with the rest of the group as some members had created their working branches from older versions of the project where the pipeline was not integrated.

3.5.4 Future Progress

I will keep making and animating models as well as finding online assets for upcoming mobs and features.

3.6 Overall

I am overall pleased with the work I was able to provide for this project. My main issues remained finding the necessary online resources that I needed, for both the assets and the multiplayer implementation. I also struggled with implementing / merging the pipeline with the other working team members' current branches, as it was quite an important change in the project.

4 Arthur

4.1 Tasks

As for this part of the project I was in charge of

- Managing the project
- Quest System
- Implementing the mobs
- Managing merges and Pull Requests

4.2 Achievements

During this phase of the project, I had a multifaceted role that included overseeing the development of the quest system, managing Github Pull Requests, and implementing new mobs in the game's new map. This was a challenging but exciting experience that required me to use a variety of skills and knowledge to achieve our team's goals.

One of the most significant challenges I faced during this phase was ensuring that the quest system was balanced and provided a good challenge for players without being overly difficult or frustrating. This required careful consideration of the objectives and pacing of the quests, as well as testing and iteration to refine their difficulty level. It was essential to strike the right balance between providing players with a satisfying challenge and not turning them away from the game due to frustration.

Managing Github Pull Requests was another critical aspect of my role during this phase. Collaboration is essential in any software development project, and Github provided a centralized platform for our team to work together efficiently. I had to ensure that code changes were merged quickly and accurately, and conflicts were resolved promptly to prevent delays in development. This required me to stay up-to-date with the latest changes in the codebase and coordinate with other team members to ensure everyone was on the same page.

Implementing new mobs in the game's new map was an exciting opportunity to expand the game's universe and challenge players with new creatures with unique abilities and behaviors. This involved designing and developing the AI and navigation systems for the new mobs, as well as balancing their strength and difficulty level. It was essential to ensure that the new mobs were coherent with the game and added value to the player experience.

In summary, this phase of the project was challenging but rewarding. The quest system, Github Pull Requests, and new mobs were crucial to the game's success, and it was my responsibility to ensure that they were developed efficiently and effectively. Through careful planning, testing, and collaboration with the rest of the team, we were able to create an immersive and engaging gameplay experience for our players.

4.2.0.1 Quest System

The quest system we choose for this project is fully modular, which means quests can be diverse as both Killing objectives and Delivery objectives are possible.

The quest system is also entirely no-code which means that anyone even if they didn't understand the scripts can add their own quests to the game. This was done to prevent any hurdle or misunderstanding among members of the team. It also allowed us to gain some times since we didn't have to understand every aspect of this system to make it work properly. Killing Objectives takes the number of mobs that need to be reached in order to complete the quest. It also takes in account a time limit which can be useful for quests rewarding a special weapon. If the time limit is reached the quest simply resets and both players are able to start this quest again.

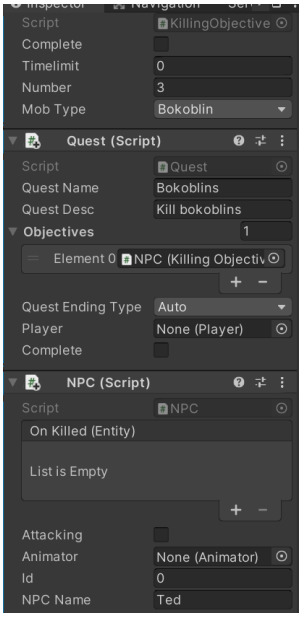

Figure 13: Bokoblins Quest

Each quest can be started by only one player, the quest remains blocked for the other player unless the time limit is reached.

The quest ending has two possible modes. One called "NPC" where you need to come back to the NPC that gave you the quest in order to be rewarded.

The other one is called "Auto" which means the quest will simply reward the player when every objective is completed. This mode is currently in operation to get the key in order to be able to exit the first level.

In conclusion, the implementation of the modular quest system in the project was a challenging task that required a great deal of effort and attention to detail. One of the biggest struggles encountered was ensuring that the killing objectives were balanced and challenging enough to provide an engaging gameplay experience, without being too difficult and causing frustration for players. Additionally, the implementation of the time limit and the number of mobs to be killed required careful planning and programming to ensure that it functioned smoothly and did not cause technical issues or glitches in the game.

Another challenge encountered during implementation was ensuring that the quest system was properly integrated with other aspects of the game, such as the NPC interactions and the reward system. This required extensive testing and debugging to ensure that all the components of the system worked seamlessly together.

Despite these challenges, the implementation of the modular quest system was ultimately successful, and it added significant depth and complexity to the gameplay. The result is a more engaging and challenging experience for players, which will hopefully lead to a more enjoyable and rewarding game.

4.2.0.2 Pull Requests

Managing pull requests was a tedious but interesting task, it involved resolving many conflicts that arose due to the complex structure of Unity. As a matter of fact, each time we branched to create a new feature conflicts were created since everyone was modifying the same map. Resolving conflicts also implied team management as we had to discuss how things needed to be organized in order to have a complex but clear layout for the project.

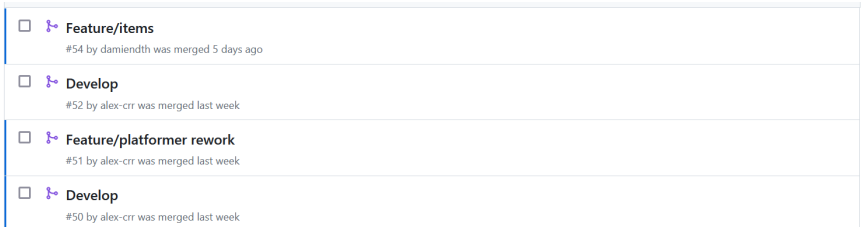

Figure 14: Examples of frequent Pull Requests

The most difficult merge was implementing 2D maps inside the correct scene. Tiles consistently created conflicts. We had to resize them and changed some of their attributes to make them work smoothly.

Most conflicts arose when all team members weren't in the same room. As a consequence of this inconvenience, we were forced to use github built-in issue features to organize some of our actions regarding these pull requests.

4.2.0.3 Implementing mobs

Initially mobs were designed for both 2D-top and 2D-side maps. However some adjustments were needed due to changes in the project's structure mentioned earlier. As a consequence, I personally changed some parts of the script involving distances to make them look more real when put in the 2D-Top map. The player can now fight bokoblins on the Zelda first map, using either the sword or the bomb. One of the most tedious part of implementing mobs was the unprecise documentation from

Figure 15: Implementation of Bokoblins on the first map

Unity. Since the navigation and AI system changed in 2018, lots of features were either deprecated or simply suppressed. We had to download an experimental package in order to be able to use NavMeshSurface component. The latter was essential since every maps stand on the same scene, implying that we need to build multiple navmeshes for every different map.

Another parameter that was taken into consideration is the multiplicity of mobs that were created. So far bokoblins and charmanders (which will be implemented in the minecraft map at the next defense) have proper navmesh agents. The standard AI package doesn't allow for multiple agents to be bake. This was resolved by the com.unity.ai.navigation

Unity AI uses NavMeshSurface component to determine on which surfaces the mob can move, using this and attaching a NavMeshObstacle to every colliders of the map, we were able to make very precise and detailed navigation paths for the mobs that are currently implemented on the map. Paths

Figure 16: Perfectly baked NavMesh

are obviously calculated to ignore houses, bushes and trees, bokoblins are roaming in a 5 radius circle from their spawning locations. For this defense, mobs were placed close to the player spawn for demonstration purposes.

They also have an attack range of radius 3 which means they have to be close to the player in order to attack. This value is not set to one in order to avoid bugs that arose during early developpement when mobs made damages even though they were far from the player.

Current tests show responsive mobs that were able to deal with many situations that were put to tests. Implementing mobs in a game can be a very challenging but crucial task. Mobs are essential to create a more immersive and interactive gaming experience for the player. They provide a level of complexity and excitement to the game that keeps the player engaged and interested. However, implementing mobs requires a lot of effort, attention to detail, and a deep understanding of game mechanics, programming, and AI.

The development team must carefully design and program the behavior of each mob, ensuring that they are challenging but not frustrating, and that they can respond to the player's actions in a realistic and believable way. Furthermore, different types of mobs require different programming approaches and need to be tailored to the specific game mechanics and environment.

In the case of the project mentioned earlier, implementing the mobs involved many challenges, such as dealing with imprecise documentation from Unity, adjusting the script to make the mobs look more realistic, and building multiple navmeshes for every different map in the project. Additionally, the team had to ensure that each mob was responsive, adaptable, and could deal with many situations that were put to tests. Despite these challenges, the development team managed to create a responsive and challenging mob system that added a new level of excitement and complexity to the game.

4.3 Future Progress

The next tasks for the project involve expanding on the existing quest system and implementing new mobs that fit within the context of the game. In terms of the quest system, the plan is to make it more comprehensive by covering all types of situations that would make the game more coherent. This means adding more objectives and creating a quest with a time limit. There are already multiple ideas in place for the quest, and a reward has been chosen, which will be a diamond sword that fits the next map theme and mobs present to challenge the player.

Another crucial aspect of the project is to ensure that the codebase and project files are well-organized

on Github. This will make it easier for contributors to understand and contribute to the project. As the project expands, it will become increasingly important to maintain a well-structured codebase that can be easily understood and modified.

Finally, the implementation of new mobs is a significant undertaking. The charmander mob is the next one planned for the Minecraft map, and it needs to be coherent with the game's mechanics and design. The process involves developing new AI, implementing new animations, and creating new models. Ensuring that the new mobs fit seamlessly within the existing game world requires a great deal of attention to detail and testing. Overall, these next steps will require careful planning and execution to ensure the continued success of the project.

5 Conclusion

As a general trend, it appears that most of the team members achieved their part of the project. We made sure to have frequent reunions about the current project state, allowing for a fair task distribution. We also learned from our last defense errors and took the time to regularly merge our different branches, to ensure compatibility throughout this third of the project. We also took it upon ourselves to be as thorough as possible when it came to fixing the different technical difficulties we experienced before, so that they wouldn't impact the future state of the project. We also made sure to not create new problems, for self-evident reasons.

Although this part of the project allowed most of the team to grow closer, and work in a tighter manner, it must be pointed out that none of the workarounds we tried to ensure the involvement of the last team member failed miserably. After spending countless hours trying to help said member in it's task, and as he had no impact whatsoever in the output project state, we decided the our time was most valuable when actually working on the project.

Despite this negative factor, and it's effect on meeting our deadlines, the project, although not at the desired state, has still seen major improvements during the last few weeks. We also managed to improve on the most important part of the project: teamwork.

6 Sources

6.1 Maps and Tiles

Pixilart, [online]
ProMeLaGen - Procedural Metroidvania Layout Generator, by Logan, [online], https://loganames.itch.io/promelagen
2Bit Micro Metroidvania Tileset, by user 0x72, [online], https://0x72.itch.io/2bitmicrometroidvaniatileset
Relic and Pursuit, by ChrisLHall, [online], https://chrislhall.itch.io/relic-and-pursuit
LDtk Editor, by Deepnight Games, [online], https://deepnight.itch.io/ldtk

7 Annex - outline

7.1 Introduction

7.1.0.1 Our game

7.1.0.2 Our Progress

7.2 Member presentations

7.2.0.1 Damien

7.2.0.2 Alexandre

7.2.0.3 Arthur

7.2.0.4 Pierre

7.3 Outroduction

7.3.0.1 Expectations

7.3.0.2 Improvements - next steps