First Report

Falling asleep studios

Luminosité Éternelle

Johan Emmanuelli
Ilan Mayeux
Luca Sarubbi
Matthieu Porte (*Project leader*)

Contents

1 Introduction

1.1 Team presentation

Our team is composed of four first-year students from the A2 class. We decided to join force together in this project to experience an adventure we will remember, forever. Therefore, this team is composed of:

Matthieu Porte (*Project leader*)
I discovered programming as a kid through front-end web taugth on khanacademy at the time, then I started learning C# as well to program games on Unity. I've done some game jams since, started some projects alone but never ended up with something I was completely satisfied. I hope for this project to be the one, but taking a look at my sidekicks I see no doubt about that!

Luca Sarubbi
Hi! I'm Luca Sarubbi. An aspirant low-level software engineer; although it all started from enjoying minecraft's game engine through custom Java plugins, I then got introduced to lower level languages such as C or Rust. I discovered epita thanks to a friend's recommandations and I am ready to give this projet my all. My contribution to the project will mostly be on gameplay and multiplayer in order to gain some experience to someday create my own game-engine.

Ilan Mayeux
I am wandering as Ilan Mayeux. A first-year Epita student from the A2 class. Previously from Le Bon Sauveur highschool, I learned precious knowledge during my scholarship and built a passion in computer science as a result. My goal is to renew this tradition and make this second-semester project a warm memory, something that we can be pround. Sincerely, your typical Linux enjoyer.

Johan Emmanuelli
Hello, my name is Johan. To this day, I really enjoy my education at Epita. I started programming in highschool and I immediately really liked it, to the point that I want to make it my profession. For this project, I am planning to learn and develop many skills. I am also really excited to work with this team, and I am sure that we will all be proud of this project !

1.2 Project presentation

Luminosité Éternelle is a co-op game in a 3D mountainous environment where you travel in hope to bring back your deceased companion. An epic adventure awaits the players. Monsters, puzzles, challenges, explorations. Discover the world, resolves enigms and find a path towards the light, the key of all mysteries.

This is a game played by two people. Each player fullfills a specific roles:

- The Human: Travel through the world in your weak body. Falls and enemies will try to bring you to the other side, your friend side, Death. You are able to interact with the living world and talk with living beings.

- The Ghost: You are freed of your weak body, but not safe either. While you are able to fly for a short duration, void enemies are a nightmare. Be careful, you can only interact with the dead world. You're also able to go through transparent objects.

Our strongest ability can easily become our greatest weakness.

1.3 Graphic chart

1.3.1 Logo

Our game project is called *Luminosité Éternelle*.

Its logo represent mountains, the players final destination and its shining top, the light that will guide them. A journey awaits them, and many perils with it.

In a world mixed with darkness, one of the reason it has not fallen down is this temple at the top of everything. The temple of light.

1.3.2 Art direction

The game wants to convey an important message. Thus the art and the global ambiance of the game will play an important part of it. While the gameplay is essential, it cant́ survive without art.

The art direction for the game will be based on a mountain landscape, with a shining temple of light at the top. The mountain itself will be rendered in low-poly style, with simple geometric shapes and bold, vibrant colors that give the environment a fantastical feel.

The temple of light will be a towering, radiant structure that stands out against the dark and rugged mountainside. The game's creatures will also be fantastical and low-poly, with exaggerated features and striking visual designs that make them stand out from the game's environment.

Overall, the art direction for the game will aim to create a surreal and imaginative world that is both visually striking and true to the game's low-poly aesthetic.

2 Project Advancement

2.1 Gameplay

2.1.1 What was done

While Luca was working on the multiplayer side, Ilan took the decision to start the gameplay as we were a bit late on this part. The codebase was later revised and updated by Luca.

The player

There is no gameplay without a player. Therefore, implementing it was necessary.

Creating the player movement and allowing to look around is the first step. Some basic PlayerMovement and MouseLooking class were created to handle these tasks. Thus, our player can now move with W A S D

Then, the main Player class where Human and Ghost will inherit from it. This class will handle animations, inputs and most of the references. The ghost player has a special jump: he can fly for a short time. This can be triggered with [ ].

Then, when the basis of the player were done, some equipments were needed to adventure in this dangerous valley. And therefore, an Item class was created. But to save these items, the player also needs an Inventory and to use them, an item bar that will later be called an item wheel. Weapons, Potions and others will inherit from the Item class.

Since the items needs to be picked up on the ground, the user needs an indicator, a message, "Press E to pick up the Demon King Sword" for example, that the item will display when dropped. Then, after the basis of an Item come the item themselves. In order to defend himself, the player needs some weapons. A sword to handle enemies at close range, and a staff to send fireballs on enemies. Later more weapons will be added. Potions to heal injuries. Yet, the player has no health nor mana. So a HealthManager and a ManaManager were added to fullfil these needs.

All the player's interactions are handled with the interact key, which is E by default.

Later on, a dialogue system was added. Interacting with the NPCs is an essential things to create a story and help the player. To add some further interactions, the player can talk back to trigger some special dialogues depending on the choice made.

Since the player can manipulate different weapons and items, the item wheel allows the user to change items and unequip them.

2.1.2 What needs to be done

A lot was done. The very foundation of our game. Some diversities needs to be added: more items, more dialogue. A very interesting feature would be the ice axe in order to climb towards the pic of the mountain. How it will be made is still a pending question. A kind of minigame where you needs to manually build your path in the mountain? This could be a very interesting way to implement it. Of course, in terms of weapons, bows, spears and magic spell are considered!

The player may also needs an inventory and be able to assign which items are used and how it is ordered.

We also needs to decide what happens when one of the two players dies. How much they are getting punished by the game, where they respawn, etc.

2.2 AI

2.2.1 What was done

As planned in the book of Specifications, Ilan handled the A.I. part and will explain what was done and what needs to be done.

Searching for the best Algorithm!

Our project is an adventure full of puzzles. Yet, there is no adventure without citizens, soldiers or monsters! Therefore, we had to develop an AI to play against or with. An AI capable of piloting the most dangerous foes while being easily maintainable and upgradable.

There exists many different algorithms to develop an AI pattern, without necessarily going towards machine learning. The most basic one would be an "If else" AI. The issue is that it would turn towards spaghetti code and upgrading it would get harder and harder. The second option we came accross was the Finite State Machine (FSM). It is well-known and very efficient. However, as the previous one, it can also quickly become a mess when trying to modify it. The third one was the Goal Oriented Action Planning (GOAP), an upgrade of the FSM into a dynamic one. It is the one we are currently using.

The Goal Oriented Action Planning Algorithm (GOAP Algorithm)

As said previously, we ended up choosing the GOAP system above the others. But what is it? Why is it better?

The GOAP is working with 5 main classes, which are the GAgent, GWorld, WorldStates, GPlanner and the GAction. These classes are working together to make an A.I.. This system was modified to fullfill our needs.

- WorldStates: It is a simple class that saves every states it is given, with helper methods.
- GWorld: A singleton with its own WorldStates and other data. This is the Global World States, where AI can communicate with each others and share information (e.g. the alive slimes)
- GAction: An action realizable by an A.I.. It can be a really simple action, such as walking or more complex, like abilities. Each action has four necessaries methods that need to be implemented: PrePerform, PostPerform, IsComplete and Action and attributes such as the cost, the duration, etc.
  - PrePerform: This method is executed before everything else. It is useful to check and setup additional things. If it returns false, the whole Action is canceled.
  - Action: The main action. It is executed after the PrePerform and contains the goal of this Action. This method is not in the original system.
  - IsComplete: This method returns true if the action is complete. It is useful to make an action last until a certain condition. This method is also no in the original system.
  - PostPerform: This method is executed at the end of an action. It can be delayed with the duration attribute.

The main importance of the GOAP system is residing in the Preconditions and Effects of an Action. Each action has Preconditions that need to be met in order to be executed. And each action passively brings some Postconditions, also called Effects that can be used to attain other actions.

- GPlanner: The goal of the GPlanner is to build a plan for the GAgent from a certain goal. A goal will be the name of an Effect. Then, the GPlanner will try to find a succession of Action to achieve this Effect. It will try to find the best succession of Actions to achieve the Effect. And to obtain this succession of Actions, it will use a Pathfinding algorithm which, in our case, is an A* Algorithm. Then, if it finds any plan, it will select the best: the cheapest.
- GAgent: The GAgent is the parent of the A.I.. Each A.I. will inherit from this class. The GAgent has its own WorldStates, which will be called LocalStates or ActionStates. These are states that are active and can be updated by the Actions. At the beginning of its lifetime or during processing, the GAgent will receive some Goals that repeats themselves or not, with a certain importance. The A.I. will take the most important goal first and complete it. With the current goal, the GAgent will create its Plan, helped by the GPlanner, and start to achieve one by one the Actions necessary to achieve the goal. If an action fails (The PrePerform returned false), it will start making a new Plan to achieve the goal.

This is the GOAP system that we used. Completely unique since our adaption. To compare a GOAP and a FSM (Finite State Machine), using graph is the best way. When making the A.I. in a FSM, actions are represented by a Graph and their relations with branches. And as any graph, when relations are getting complexe, the graph will look messy. If we take back the same graph in a GOAP system, the nodes are exactly the same. The only difference is that there is no branches, no link, no connection. Each node is alone, yet connected. More precisily, only the Preconditions and Effects matter.

This is where the choice of the GOAP took place. It can be easily maintained and adding an action is just adding its Preconditions and Effects and then make sure that it can be accessed. An action can also safely be removed if there is other way to achieve the Effects.

The First A.I.: Slime

After selecting and implementing the Algorithm, creating the first foes would become the next step. And like every fantasy Rpg games, the slime was chosen. The cute little slime will fast enough become a threat for the player.

The slime is a blob-like creature that is found throughout the game's world.

The slime is not particularly intelligent, but it is relentless in its pursuit of prey. It will follow the player character around, trying to attack with its slimy body. If the player character is hit by the slime, they will become slowed down and temporarily hindered, making them vulnerable to other enemies. The slime can be defeated by using weapons or spells to attack it, but it is resilient and may require multiple hits to defeat.

In addition to its normal form, the slime can also split into smaller slimes when it takes damage. These smaller slimes are weaker and easier to defeat, but they can still be a nuisance

if not dealt with quickly. The slime can also regenerate its health over time, so the player will need to keep attacking it to keep it at bay.

The slime is a pacific creature. It means that it won't attack its own environment. However, if an unwelcomed person enters its domain, the slime will become hostile to him. Thus, the slime will try to eliminate this threat, making him leave the domain or die defending it.

Slimes are often staying in groups and chasing their prey together. When they spot an ennemy, they'll alert all their kind in the Spot Transmission Radius to attack the origin of the threat.

When a slime is deeply wounded, they'll either try to fight to death or to flee. A slime prefers staying alive than elimating their target so it'll flee most of the time. If the slime choose to flee, they'll first try to take distances from the player. Then, they'll try to join another group of slimes.

When a slime dies, it will split in three smaller slime. Their stats will be equivalent to a third of its parent start. It can only occur once. If after 30 secondes, the slime survives, it will grow into a bigger one and increasing its max health, allowing it to regenerate additional health.

The slime special ability is a powerful jump towards the threat to close down the distance. The jump does a certain amount of damage to the player and also slow them. Luckily, a zone inform the player to its destination, which is, often, them. That's why a group of slime quickly become a threat.

This slime was thus implemented using the previous system. In order to differentiate different foes, a Monster class was created. Slime inherits from it. Allowing to handle its animations, stats and helper methods.

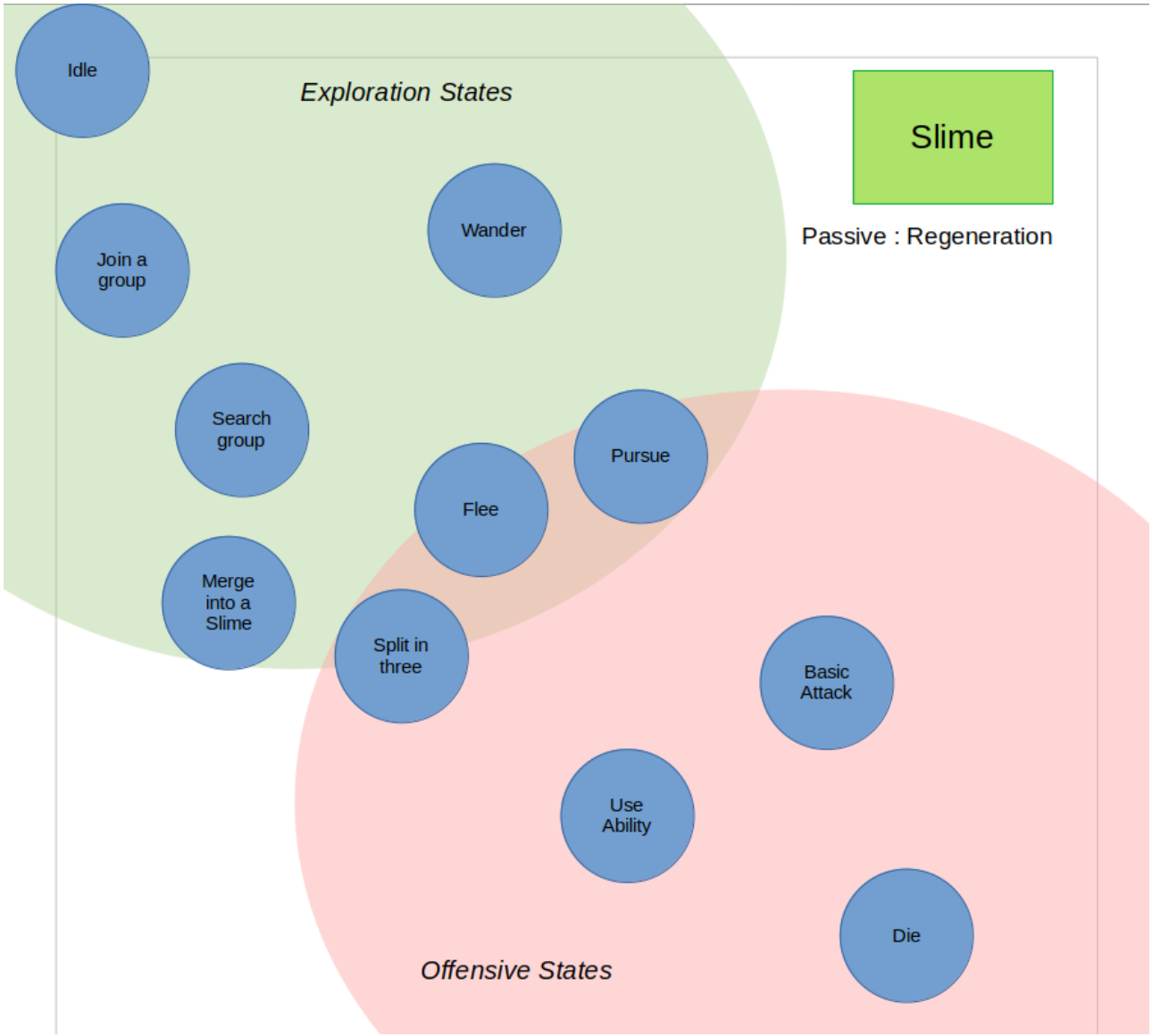

Figure 1: The slime actions as a graph

The second Monster: Dragon

The second monster is the dangerous Dragon. Indeed, there is no mountain nor adventure without dragons. Breathing fire is their specialty after all.

This monster is specialized in attack, making it a dangerous foe. Even more when its a flying creature, meaning that melee attack are not very efficient against them.

The difficulty of making the Dragon is the 3D movement. A dragon can move also move on the y-axis. Meaning that it has to choose when gaining altitude and when losing it. If it is not done correctly, the A.I. will look stupid or eays to beat. Ruining the dragon reputation is not part of the game so they need to get the best A.I. possible.

The current solution to this problem is to implement a Volumetric Pathfinding with Astar.

The NavMesh system is optimised for AI to follow the terrain, but unity provides not alternative to implement a Pathfinding in volume.

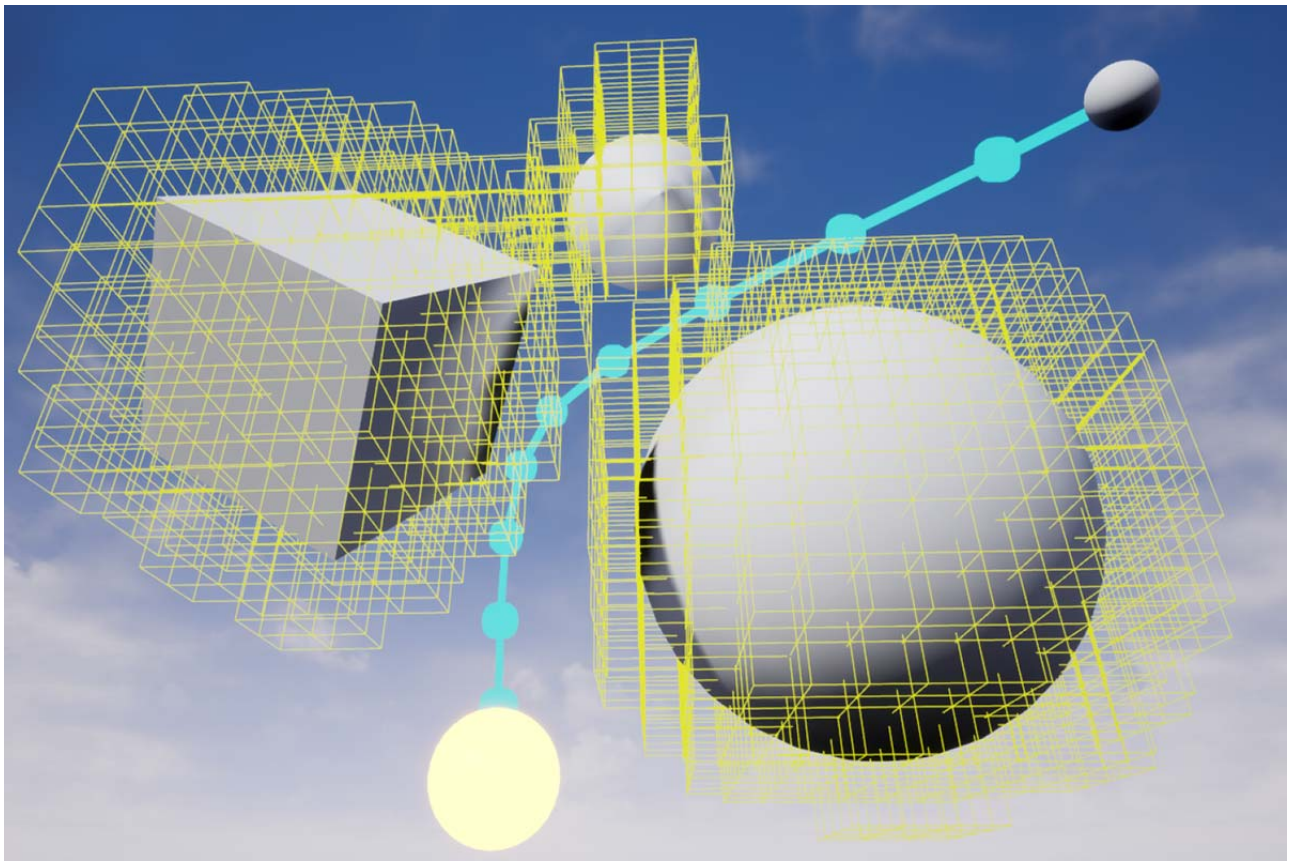

Figure 2: An example of results from a Pathfinding 3D. The feature expectations. Image from https://github.com/darbycostello/Nav3D, a solution for Unreal Engine

For now, there is not much things that was implemented for the dragon. It can breath fire on enemies, move like the slime and some other actions. It has yet to reach the state we want.

2.2.2 What needs to be done

The foundation of A.I. were built and its efficiency proven with the slime. It only remains to implement the remaining monsters. The goal is not to have plenty of simple-minded monsters, but a few smart ones. Of course, at the ends some monsters will be added and maybe animals that can be preys for these monsters as well, to add some wildness.

The volumetric Pathfinding needs to be finished and optimised enough not to influence the frame rate cost.

The slime also needs a noise detection and a behaviour when it occurs. The game not having much sounds yet, this was not implemented.

Therefore, for the next presentation, we hope that the Dragon will be completed and maybe some other monsters with it. It depends how challenging the Dragon ends up becoming.

Adding additionals entities such as animals could add some variety to the game. Since it is happening in the wild, some dears, goats or even bunnies could emphasis the atmosphere.

Only imagination remains!

2.3 Story

2.3.1 What was done

While the game foundation were made, the game story also progressed. The main ideas are in place. The map is already a visual proof to what we wants: a map that brings the player to the top of the mountain, discovering a colder environment and with it, many obstacles.

Moreover, the dialogue system is essential to tell a story. And this is now ready!

In order to write the dialogue, we created a dialogue language. These files will be parsed and later played.

```
1 [!
2 #Name
3 >NextNPC
4 Line 1
5 Line 2
6 Line 3
7 ]
```

Listing 1: An example of Dialogue

The brackets delimitate the dialogue. The exclamation mark means that the next dialogue will be played at the end of this one. The sharps indicate the names of the interlocutor.

The greater than indicates the name of the next interlocutor's gameobject. This means that the next dialogue played is from NextNPC. Then, from line 5 to line 7, it represents what the interlocutor will say. There are no limits.

The exclamation mark and the greater than are not mandatory.

This is used to create a dialogue, but choices can also be made.

```
1 {!2
2 #Name
3 What choice do you want to make?
4 -Choice 1 Sentence
5 DIALOGUE1
6 -Choice 2 Sentence
7 DIALOGUE2
8 }
```

Listing 2: An example of Choice

In this example, {} are used to delimitate the choice while [ ] are for a dialogue.

The hyphen indicates the beginning of a choice. It must be followed by a DIALOGUE. DIALOGUE means the structure used in the example above.

This means that a choice triggers a Dialogue.

The 2 in "{!2" indicates that 2 inputs are necessary for this choice. A choice is between 1 and 3 inputs. A choice is thus composed of the input sentence + the triggered dialogue.

Therefore, dialogue files are a succession of DIALOGUEs and CHOICEs for a single NPC.

Pieces are slowly getting brought together.

2.3.2 What needs to be done

Our story is only at its beginning. Puzzles, challenges and foes are awaiting for their awakening.

2.4 3d modeling

As we are not 3d designers, we had to look on the internet for some assets we could use. Yet, we had a quite precise idea of what we wanted in terms of environment. Matthieu did all the 3d modeling that is original to the project.

2.4.1 What was done

Firstly I learnt how to use the blender, then I focused on doing element of the environment. I created wood and stone bridges, fences, but most importantly I modeled modular houses so that we could get exactly what we need.

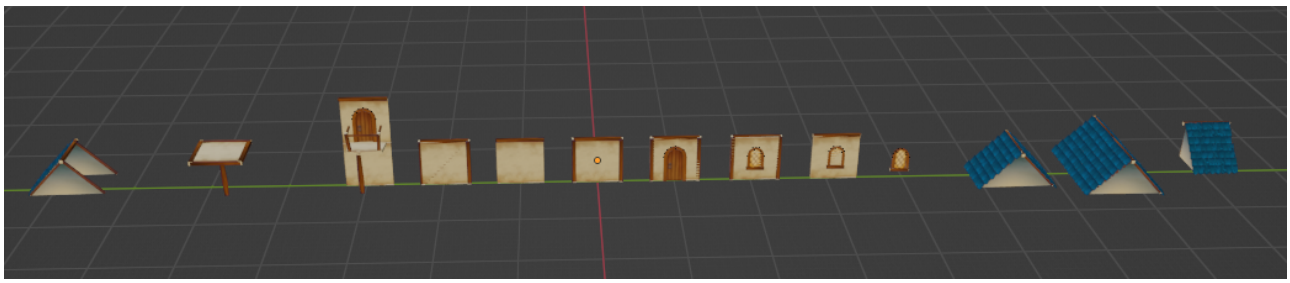

Figure 3: The pieces of the modular environment

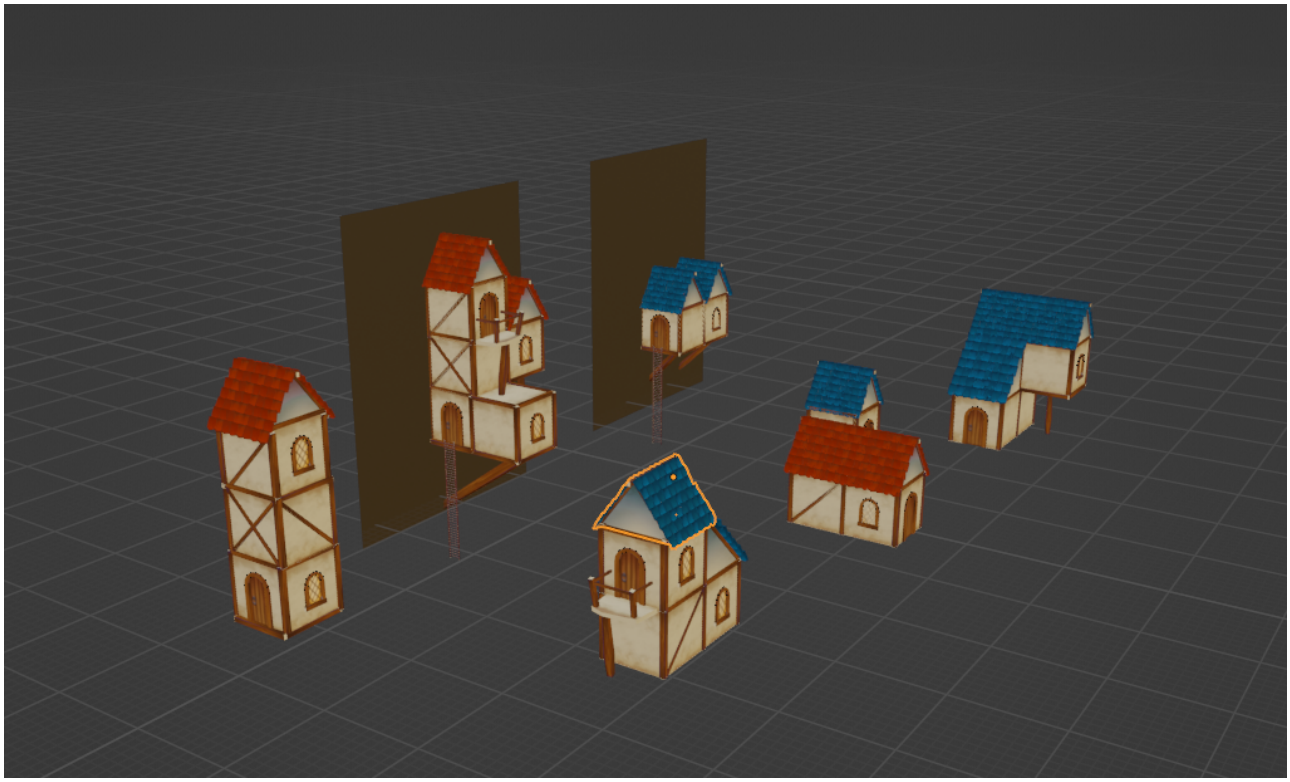

Figure 4: The pieces of the modular environment

Not everything can be shown here, neither everything will be used in the game. But through experimenting I have learnt more in depth how 3d models work.

2.5 Level Design

As the leader of this task, Ilan created the basis of the level design. While doing the puzzles, Matthieu created playable areas to integrate in the map.

2.5.1 What was done

A map, the place where everything happens. There is no game without a map. A map decides whether a game can be interesting to play or not, since it is the first visual things that the players will see. Our game is in a valley environment. Mountains, hills, snow, forests, lakes, rivers. This is the kind of environment we want to show to the player. A beautiful adventure needs a beautiful map.

The first thing to make is the terrain. In our case, a 1.5 kilometers squared map. The second part was painting the very ground. Creating mountains, hills, etc.

Then, the next step was adding textures to this terrain: snow, rock, grass.

Some temporary water textures were used for the river and the lakes. It may change. Then, after all that, trees were added on the map. Depending on the location, it was snow trees or normal trees, with a size that differs.

All of these concludes the basis of the map.

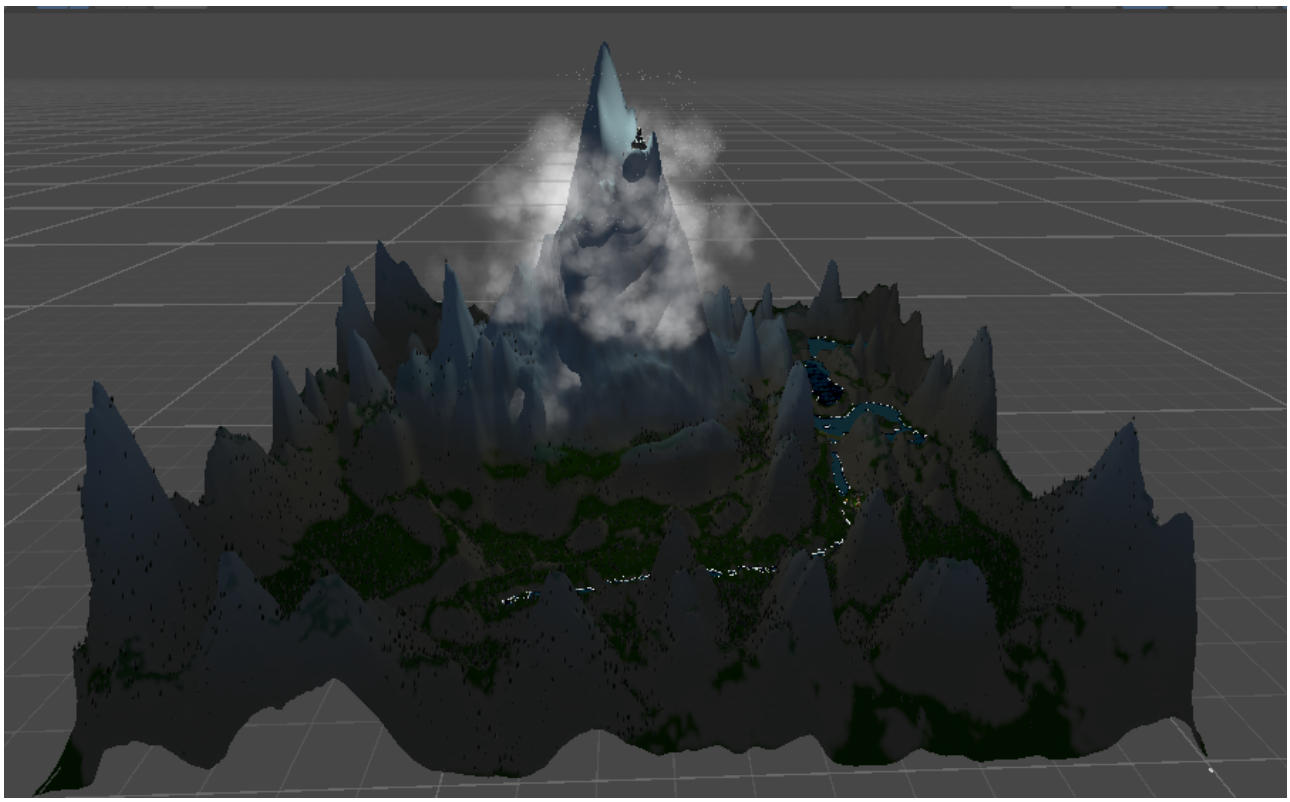

Figure 5: The textured map

In order to bring to life the snowy atmosphere, some particles effects were created: a fog particle and a snow fall particle. Particles can be very costful if not handled correctly. Mostly when its about rains and snow fall effects. So it needs to be carefuly set up.

Then, some structures were added: A village and the temple of light, also called the church. This church rings the bells at a certain time, triggering enlightnment on the world.

Most of the buildings are from external assets, but some were modelled by the 3D team.

Inside the mountain is the dungeon. Which is accesible towards the end of the game. This allows us to change the mood of the game, to constrain the player in a smaller area to make the fighting harder, to put more pressure on the player to make it less able to think about the puzzles.

2.5.2 What needs to be done

The map was done. 1.5km is very large so it needs to be filled. Adding dungeons where the player can resolve the enigms and fight monsters, adding village to exchange ressources and communicate information. Adding some treasure chests.

The design of the church may also change. It is a bit too big and impressive.

When the church rings its bells, some events could occurs. There could be also a negative counter part where the world is plundge in the darkness with a dangerous threat ahead.

Some additional effects could be added to add more ambiance. For instance, we need to create our own waterfalls effects.

2.6 Puzzles

Matthieu took the responsability to plan and setup the basis we will need to create all the puzzles on the map.

Our game in its duality of worlds has two facets: the first is the combat system which forces players to fight certain monsters and to leave the others to their partner. The second facet is the reflection around puzzles with a particularity in each dimension. The two players do not usually have access to the same information, nor do they have the same freedom of action. When we design a puzzle, we follow a method: each problem must be new in some way for the player, to some extent. The player must have almost all the knowledge to solve it and each puzzle has a new aspect, which we will be able to use in future puzzles. Each level is therefore in its own way a small tutorial in disguise. The puzzles themselves are composed as follows: The player should not feel lost, he should even have an intuition of the solution. By following this intuition the player will naturally realise the real difficulty of the puzzle: the "catch". It can take the form of an inconsistency, an action still unknown to the player or a detail that the player may have missed (a missing object for example). A problem that is too small The moment the player starts to find the game simple, he realizes that that he had underestimated it. This is how we hope to keep the player interested in our project.
Here is the puzzles concepts that are working now :

2.6.1 Connect 4

In order to offer puzzles that are not as linear as they sometimes are, we decided to implement a variant of Power 4 in one of our levels. This is a cooperative version of the board game (both players can play together, which encourages communication and listening). The two players play against a computer that predicts all moves to a certain depth using the minimax algorithm. This is an algorithm that applies to game theory for two-player zero-sum (and full-information) games consisting of minimising the maximum loss. It causes the computer to review all the possibilities for a limited number of moves and to assign a value to them that takes into account the benefits to the player and to his opponent. The best choice is then the one that minimises the player's losses while assuming that the opponent seeks to maximise them. This algorithm is very memory and computationally intensive when one wants a consequent search depth and a very high level of play. This is why we have reduced this depth to a medium level, so as not to block the player or consume too many resources.

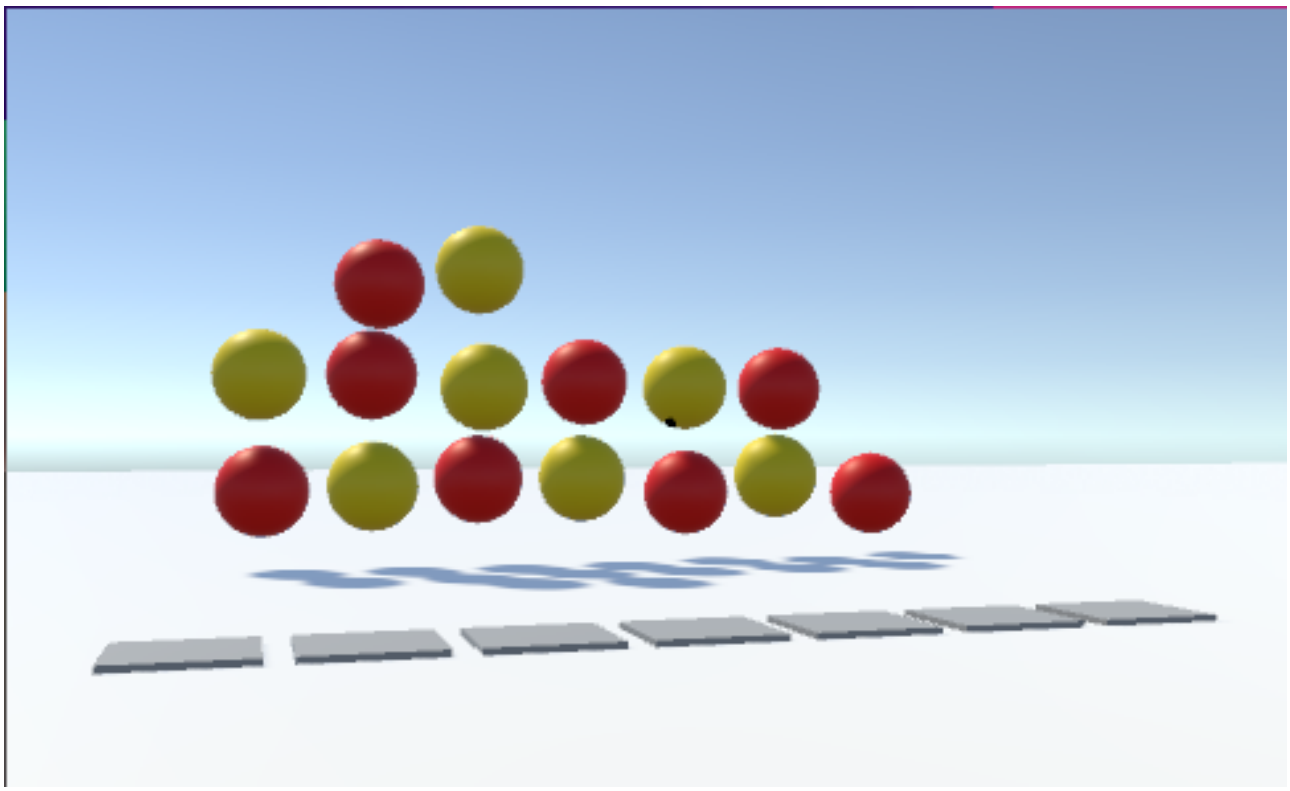

Figure 6: A game of connect 4

To come back to the idea of "catch" mentioned above, the players are naturally led to think that they need to win a game to keep going in the adventure, but in fact, it doesn't matter if you win a game or not. To keep going you need to jump on the stone of the game to reach a platform. We know that it can potentially be tough to understand for the player, so there will eventually be an npc here to give advice to the players. The players will have the possibility to come back and try to play again, this time with the possibility to earn some kind of reward.

2.6.2 Lasers reflexion

Some puzzles are meant to be more about reflexion. One idea we had it to introduce laser that can bounce on mirrors but not on anything else. The ray is comming from a turret and needs to get to an eye. If it does, the puzzle is solved. The turret throw a raycast, and the angle of reflexion is computed by unity every time it hits a gameObject tagged "mirror". A pickup script is also implemented to be able to move the mirror around and solve the puzzle. The ray also does damages so it can be used as a kind of weapon, but the player need to be careful because they will get damage from them eventually. This creates many opportunites of puzzles, with different difficulty throughout the game.

2.6.3 What needs to be done

We have still many ideas to implement, some are in progress and some are still projects. Among thoses we keep in mind we want to implement some platformer like areas, with some places a player can acces but not the other. We also need to properly design traps, enigmas and the environment that goes with it.

2.7 HUD and UI

2.7.1 HUD

This part was done by Ilan.

The main HUD

The Head User Diplay (HUD) is the link between game information and the user. This allow the player to be aware of hidden stats: his health, monsters health, his current weapon and way more.

Therefore, the basis of the HUD were made: the health bar, the username, the mana bar were made and synchronised with the gameplay. Thus allowing to be updated from the player script.

This was done without much issues. The next step was displaying items information. We didn't choose the easiest way to implement since its a wheel. The item that can be selected is on the East side.The player can rotate the wheel with F and G

To implement this system, a WeaponSlot class that can take an Item was created for the 8 slots. Then, the ItemWheel main class followed up with many useful methods. When selecting the next or the previous item, the wheel and slots have to rotate a certain degree in order to keep the images in the right direction and the selected item to the right.

The player can also directly select the item he wants with the keys from 1 to 8.

The Dialogue

While the gameplay handled the code part, some interactions were needed for the user. When a dialogue is entered, the dialogue box appears from the bottom while the HUD disappear.

Then, the player enters an unlocked stats. Which means that he switchs to a new input map. Press [ ] or the left mouse button to trigger the next sentence. Since the player is unable to fight back in the dialogue state, he can exit with ESC. Finishing a dialogue also trigger this effect.

When a dialogue needs the user to make a choice, at most 3 choices box will appear above the dialogue box. Selecting a choice triggers a special dialogue or an event.

Exiting a dialogue makes the main HUD re-appears while the dialogue is animated off.

2.7.2 UI

Johan's contribution to the project was the creation of the menu, which is the first screen that appears when the game is launched. One of the features Johan has implemented in the menu was the use of a moving camera tool to transition between different options selected by the user.

Menu Description

The menu was created using several tools in Unity such as buttons, a moving camera, sliders, sound effects, canvas, and input text. The layout of the menu was designed to be user-friendly and intuitive, with easy navigation between different options.

The buttons were used to allow the user to select different options, such as starting a new game, accessing the options menu, or quitting the game. The moving camera was used to create an animated effect when transitioning between different options.

In fact, the background here was the rendered map in different point of views. For example, the main menu background displays the mountain. By clicking on the "Settings" button, the camera will move in order to focus on the village, while providing game options and audio options adjustments for the user.

The sliders were used to allow the user to adjust settings such as volume and mouse sensivity.

By clicking on the "Play" button, the camera will focus on the top of the mountain, that is to say the church that has on its top the "Luminosité Éternelle".

Sound effects were used to provide feedback to the user when buttons were clicked, and to add to the overall user experience.

Input text boxes were used to allow the user to enter the room names for creating and joining a room when the room is private.

If the room is not private, it will appear in the scroll view with several attributes such as the room name, the number of player currently in the room and the number of players that the room can handle. Here, this last number is always set to 2.

Challenges encountered

One challenge that I encountered was ensuring that the user experience was consistent throughout the menu. This required careful attention to detail, and making sure that each tool was used in a way that was consistent with the overall design and user experience. Unfortunately, some tools need to be improved but we will see that in the next part.

What needs to be done

While I am generally satisfied with the menu I created, there are still several areas that can be improved to ensure that the user experience is as polished and enjoyable as possible. Specifically, there are four main areas that need to be addressed in future iterations of the game.

Firstly, some sound effects are currently missing from the menu. While the menu functions effectively without them, adding sound effects would help to enhance the overall user experience and create a more immersive environment for the player.

Secondly, while the moving camera tool I implemented is functional, it could benefit from further refinement to make the camera movements smoother and more consistent. Specifically, the transition between the main menu and the settings menu currently passes through a decorative element, so that could be improved.

Thirsly, while the font used in the menu is legible and clear, it may not be the best choice for creating a visually engaging and aesthetically pleasing menu.

Finally while there are audio and game options available in the menu, they are currently not fully functional. For example, modifying the slider value for audio does not actually change the audio levels in the game.

In conclusion, while I am proud of the menu I created, there is still work to be done to refine and improve the user experience. By addressing the areas of concern I have highlighted, future iterations of the game can provide an even more enjoyable and immersive experience for players.

2.8 Website

The Website of the project has not been strarted yet. We thought it would be more beneficial to focus on the task related to the game itself. Nevertheless we came up with some ideas of what we would like to see in it, and what tasks have to be done before the next presentation.

2.8.1 Our vision

Our website will be the main gateway to our project. It must give a concrete taste of the game. It must pique the user's curiosity, reflect the general atmosphere of the game, and make him want to download the project. For these reasons, we wanted to create an immersive website with a playful touch. The website must be accessible, we want to inform the user about our project. All the essential information will therefore be on the homepage, or it will be accessible with a click. For the immersive aspect we will take the artistic direction of the game, inspired by its large map, its duality between the world of the dead and the living, and its varied colours. We plan to use the three.js library to present 3D elements of our game on the website directly, especially the map which is at the heart of the game. To satisfy the most curious internet users, we decided to propose a page dedicated to the lore of our universe (its history, its characters), as the history and the characters are a very important element of our project.

2.9 Animations

The gameplay adaptation was handled by Ilan.

2.9.1 What was done

Animations are what brings the game to life. It brings comfort to the player.

Most of the players and monsters animations are from several assets. The basic animations were handled by us.

The dialogue opening and closing animation for instance. The zone attacks were also handmade as for the snow and the fog.

2.9.2 What needs to be done

Continue adding animations for the new monsters and player weapons.

2.10 Multiplayer

As a cooperation game its multiplayer aspect is essential. This part was done by Luca.

2.10.1 What was done

Since Networking is a complicated and tricky field of programming the first step in implement multiplayer gameplay is usually to choose a good framework. As a framework also designed for indie developers, Unity provides its own called Netcode For GameObjects but many other companies sell their own. Ultimately, we had to choose between Netcode and the PUN (Photon Unity Networking) framework provided by Photon. We ended up choosing PUN due to its emphasis on simplicity of its API while still having all needed features such as Component Syncing, RPCs, Event-Based Callbacks, etc.

PUN

Pun is based on a system of lobby and rooms so a PUN client is either disconnected, connected to a master Server (in a lobby) or in a gameServer (playing online). This system comes in handy as it is similar to the stages of our game: A closed game is disconnected, a

client being in a lobby is in the game's menu and finally a client in a gameServer is going on an adventure.

The implementation of Multiplayer is a bit different than other sectors as implementing multiplayer is mainly about integrating it. By that we mean that the main task is to edit assets and Unity Components to support multiplayer.

In The Menu

In order for friends or strangers to play together they first need a way of finding each other. Here PUN comes in handy with its lobby system as it allows to list all opened room. So a player can either join a pending room (a room that only has 1 player waiting) or create a new one possibly private so his friend can find its room by name.

In Game

When a player is in a game, he is according to PUN in a room (a multiplayer scene), but if many people edit the same map at the same time, conflicts may arise. In order to prevent those conflicts between clients we use a master-slaves design meaning that the Client Creating the room (the online game instance) has the authority over the Unity scene (the map), we will refer to him as the host (even though he technically does not truly host the game, he and only he can create and remove all Unity gameObjects outside of Players).

So the host as the authority over all gameObjects outside of other Players, this is defined by the 'PhotonNetwork.IsMasterClient' boolean set to true, so he and only he will actually update any animation, the AI system as well as any NPC.

On the other hand, Players work a bit differently. Any Player has authority over its own Player Unity gameObject, meaning that he will handle, movement, fields and gameplay events such as attacking or doing damage.

But then, How can an AI handled by the host attack a player handled by a client ? Well, for cases like these we use RPCs (Remote Procedure Calls), this technology enable us to call a function over network. For Instance, a Slime that attacks a player will call the TakeDamage(dmg) function associated to the Unity gameObject on the host, this function will check if the Player is owned by the host, if so, it will simply call the true TakeDamage(dmg) function locally, if not it will call the client's TakeDamage(dmg) function over RPC. The client will then lower its HP (health Points), start animations, etc...

2.10.2 What needs to be done

The Player-Slime interactions are coming good but it needs and will be completed and polished. The drake also needs to be ported to the multiplayer as well as the Item system. Work will first continue on the slime.

2.11 Music and sound effects

2.11.1 What was done

Sound was not pushed really far yet. The player, the slime and many other things does not emit any sound for now.

The bells are ringing, the dragons screams can be heard but that's it for now.

2.11.2 What needs to be done

Since nothing was really done for this section, some sounds should be added, such as walking, fighting, ambiance sound, etc.

3 Personal experiences

3.1 Matthieu

I already had some small experience with unity/blender because I participated to game jams but this project has a size never seen before for me. It's very interesting and challenging since for the first time I have to cooperate on a project. It makes it sometimes slower to progress but I am learning a lot about my friends, myself, and github.

3.2 Ilan

We have been working on this project for 3 months now. And I learned a lot. Game development is way more accessible than I thought. So far, I already made a map, monsters' AIs, effects, player gameplay, dialogue, etc. Of course, I would not manage to this alone and this is the result of our work. This is already a lot of experiences and I am excited to discover what await us next.

3.3 Johan

For a person that had no experience at all in making a video game, I am proud of what I've done. I have learned a lot on GitHub and how a collaboration project works. I am really happy to do it with people that can help me to understand certain notions and I really feel pulled up. I am really excited to finish what I have to do the best way I can do it, and I am glad to see the amount of experience that it brings to us.

3.4 Luca

I like to discover new things. This project is an opportunity for me to broaden my knowledge of computers by working in a group and on an application (unity) that I am using for the first time. We have high ambitions for our project, so I'm looking forward to continue implementing new features

4 Task distribution throughout time

	1st	2nd	3rd	Current
Multiplayer	60%	90%	100%	60%
Gameplay	60%	80%	100%	65%
A.I.	40%	75%	100%	50%
3D (characters)	50%	80%	100%	100%
3D (environment)	50%	80%	100%	100%
Level design	50%	80%	100%	60%
Puzzle design	45%	65%	100%	45%
HUD and UI	50%	100%	100%	50%
Website	30%	70%	100%	5%
Animations	30%	60%	100%	30%
Music composition	50%	75%	100%	20%
Cinematics	0%	50%	100%	0%

We are mostly on time on most of our tasks. The one where we are late are the less important or the one not mandatory for the first presentation. Some were interrupted because we made what we wanted to do by ourselves and will fill with external assets. Lots of work remains but we are in a great position since the harded was done beforehand. Sounds, Cinematics and others are here to complete the project we want to handeout.

5 Conclusion

Luminosité Éternelle is a game about friendship, about relying on our teammate to achieve our goal. The context and art direction is fantastical, so that anyone would be willing to play the game even though we think itś going to be hard for children.

The game mostly count on the gameplay mecanic such that one player is alive while the other is dead. Thus each will have access to different conversations, different clues and different interactions with the environement in general. Their goal is to find their way up the mountain where they hope to find help for resurrecting the dead player.

We make it a point to ensure that the experience given while playing the game is moving, therefore we will pay a particular attention to the details like the color palette of the game, the music and the cinematics.

This project has already reached a development state we did not expect. Lot of works remains, small bugs as well and many upgrades await. But overall, we are really pround of what we did as a team. We are ready to bring this project to life.

Sincerely,
Falling Asleep Studio's team.

6 Appendix

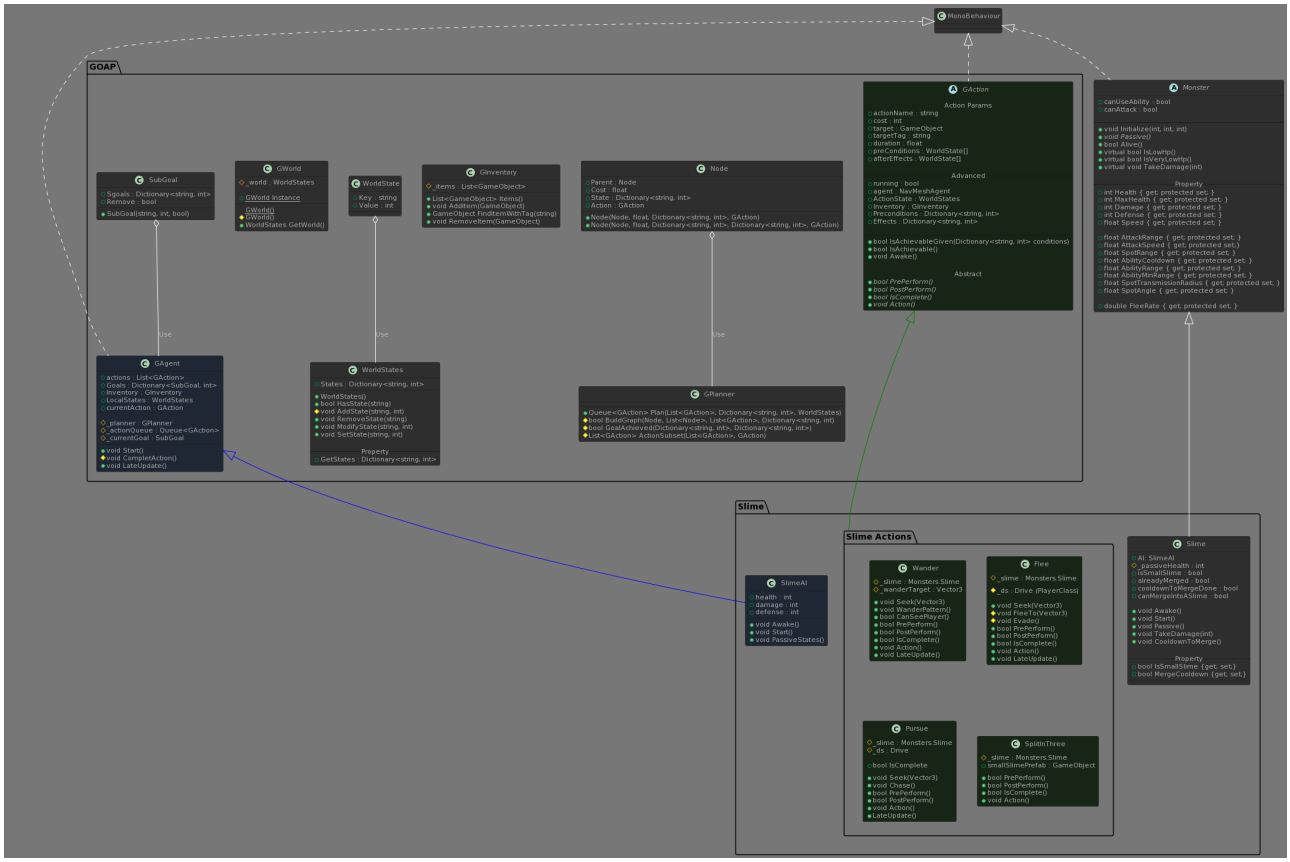

Figure 7: The GOAP system UML with the slime

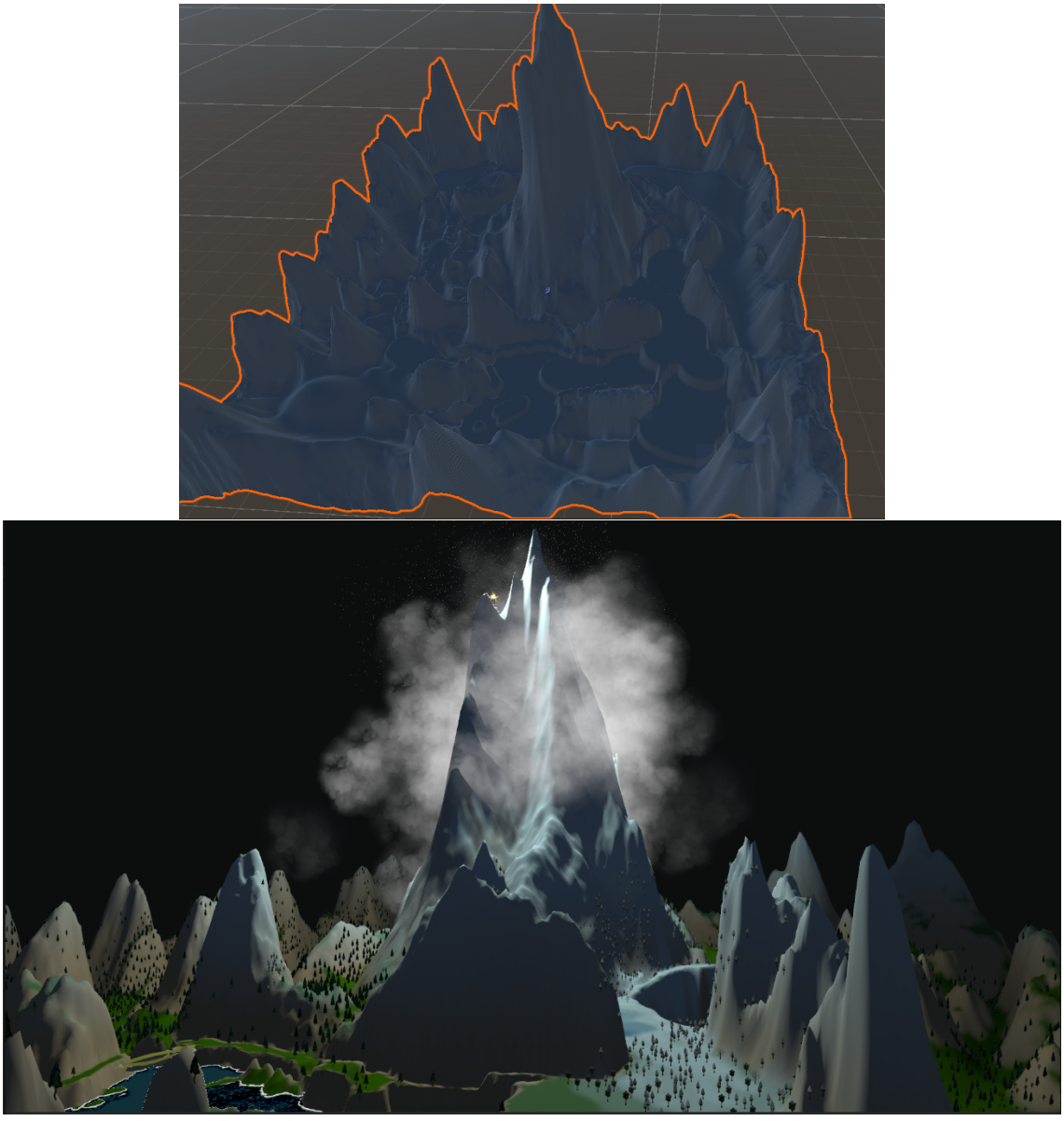

Figure 8: The textured map

Figure 9: The fortified village

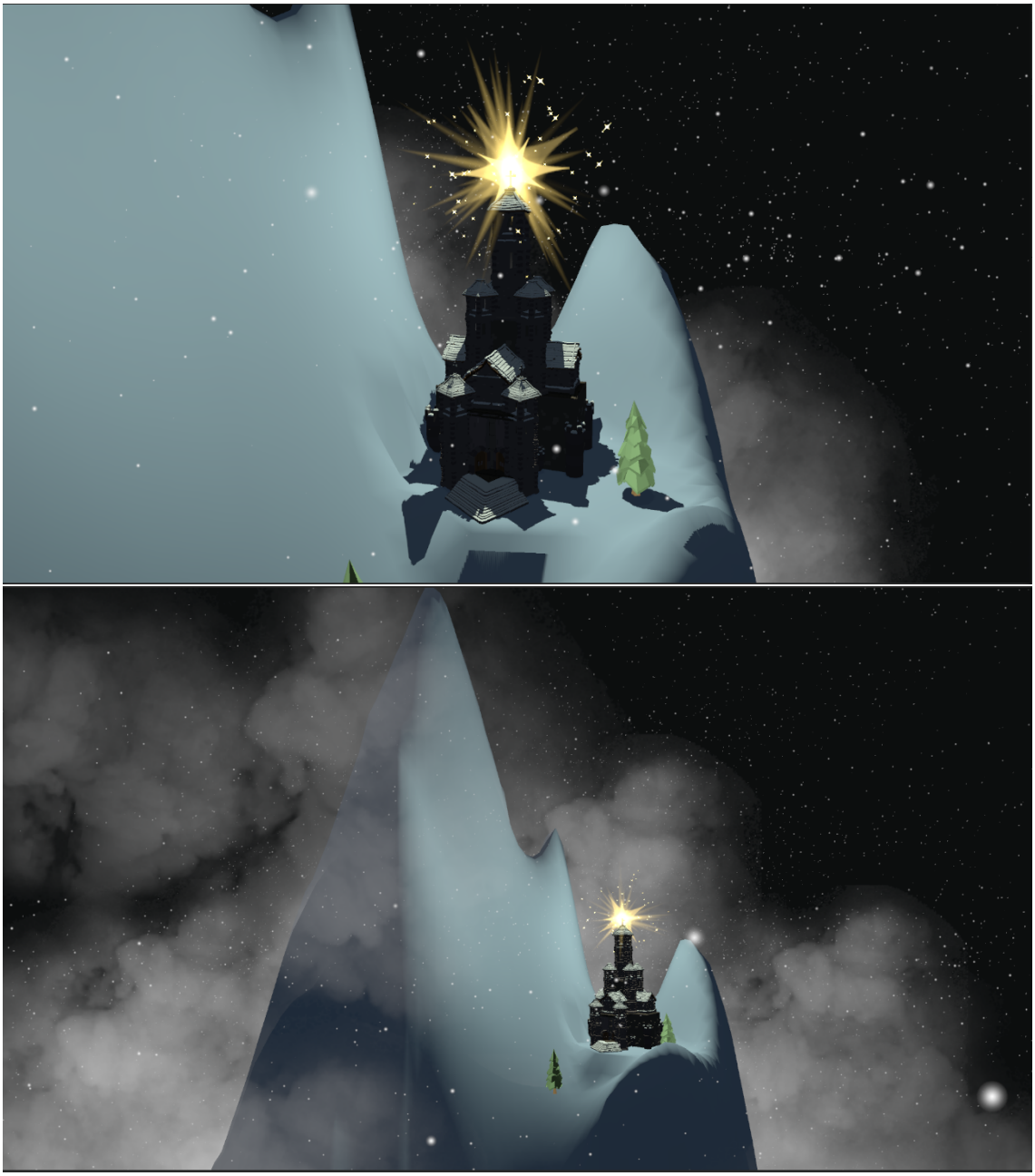

Figure 10: The church / Temple of light

Figure 11: Some gameplay images