ReConnect

Second Technical Defense Report

Louis Harel
Yoann Balasse
Victor Tendron
Adam Franco
Maxime Houwenaghel

A1 EPITA Paris

Tuesday 11th March, 2025

Table of contents

1 Introduction

1.1 Report presentation

This document is the second defense report of the ReConnect project, led by the LYVAM Studio. It describes the progress of the project from the first defense in November 2024 to the second defense in March 2025. This report explains the difficulties and issues encountered in each task and mentions the planned improvements.

1.2 Quick reminder of the nature of the project

ReConnect is a 3D singleplayer and multiplayer educational video game project. It takes place on an extraterrestrial planet named Edenia. The player is sent to this planet for communication maintenance. His goal is to fix these communications and repair his ship so he can return to Earth.

To do so, the player will have to solve electrical and electronic puzzles to evolve in the game and progress. The further the player advances in the game, the more difficult the puzzles will be. At the end of the game, the player will have acquired electrical and electronic skills equivalent to those taught in high school.

1.3 Reminder of the previous defense

Our first defense took place the Tuesday 14th January, 2025. The features available were:

- A character with a third person camera, movements and animations.
- A multiplayer system with LAN and WAN capabilities and a distant online server.
- A main menu, not yet implemented in the game, with buttons to launch the game in solo, multiplayer, open the settings, and close the game.
- The map design on Blender was still in progress, using Voronoi diagrams for a random generation of the terrain.

The advancement was as follows:

Scenario	70%	55%
Menu	75%	75%
HUD	0%	0%
Physics	30%	30%
First tutorial	0%	100%
Music & FX	0%	100%
First level design	0%	100%
3D modeling	40%	50%
Interactions	0%	0%
AI	0%	0%
Multiplayer & networking	100%	100%
Website	100%	100%
Communication	0%	100%

Table 1: Previous advancement compared to the planed advancement

1.4 Main topics

As mentioned previously, the document describes the progress of the project between the first and second defenses. The main topics that will be presented are the implementation of the electricity part of the game, the creation of the map with the choice assets, and the implementation in the game and the general improvement of menus.

Before going into such technical work, the report will also witness the more global issues encountered by the LYVAM Studio, concerning the organization of the work.

2 Organisation issues

2.1 Planning change

At the beginning of our project, the team designed a Gantt chart that defined the distribution of the tasks necessary to complete our project with respect to time. Due to their lack of experience and knowledge, in particular regarding Unity 3D game development, the LYVAM Studio overestimated their capabilities. The negligence of the general workload at Epita aggravated the consequences of this overestimation, and as a result, the progress was late on the planned schedule.

This issue was tackled during our previous technical defense. We decided, with the agreement of the jury, to change our Gantt chart to a more realistic one that we would be able to follow. The new Gantt chart of the Reconnect project can be found below.

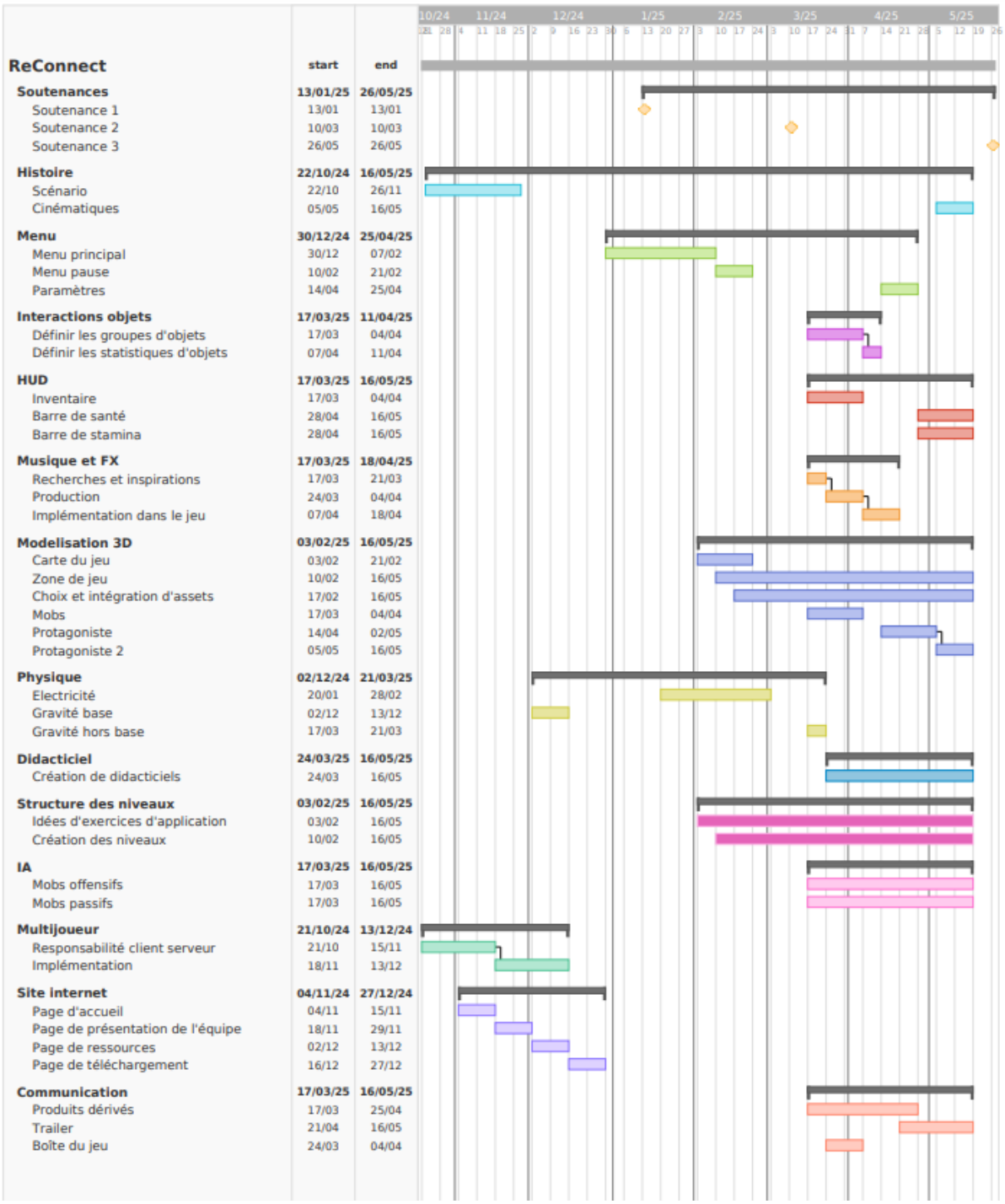

Figure 1: New Gantt Chart of the Reconnect project

2.2 GitHub restrictions

2.2.1 LFS quota

Since the very beginning of the project, the LYVAM Studio has been using Github to host their git repository and manage their coding tasks. Unity projects often include large binary files such as 3D models, audio files or video files. However, these files are not natively supported by Git, the underlying version control system of GitHub, which is mostly used for text files. This is the reason why an extension of Git was needed: *Git LFS (Large File Storage)* that allows to track these files.

In February, the members of the team had issues trying to clone or pull the repository. They found out that they had depleted their GitHub organization's LFS quota. Indeed, Github is a freemium SasS (Software as a Service) that reserves some features to premium users in exchange of, most of the time, a monthly payment. In this case, Github defined quotas for the usage of Git LFS. The plateform allows no more than 1 Gb of storage and 1 Gb of bandwidth i.e. we could download up to 1 Gb of Git LFS tracked files. The issue was that the team exceeded our bandwidth quota, and consequently, 3 of the team members were unable to fetch the project and open it in Unity to keep working on their tasks.

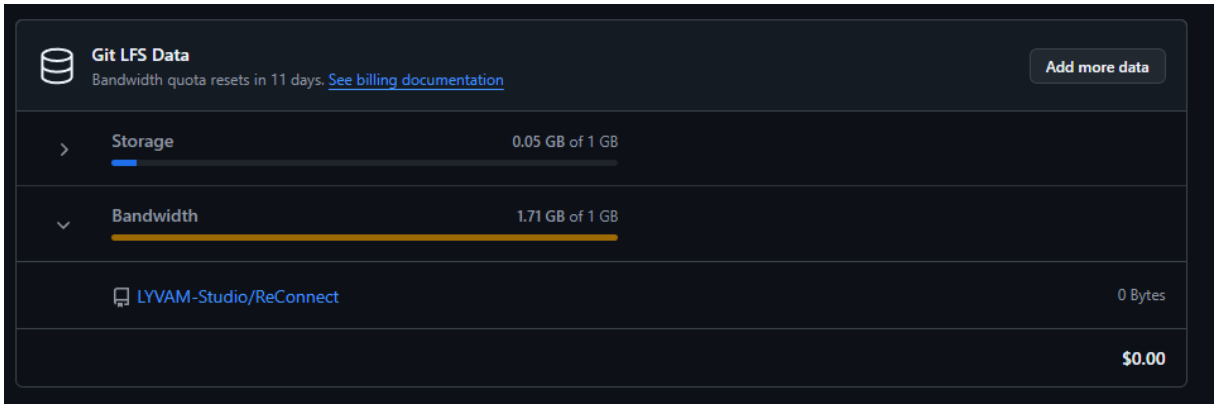

Figure 2: Our quotas of Git LFS on Github

The possible sources of the excessive consumption of the Git LFS bandwidth provided by GitHub is the continuous integration that we run several times. Indeed, the actions that the studio created, need to clone the repository before compiling the project. Moreover, the current GitHub actions clones the repository 4 times a run because it compiles the game for Windows, Linux, MacOs, and the server for Linux.

As the reset date approached, our quota did not reset. We therefore opened a Github support ticket to ask for our quotas to be reset to 0, so that we could get back to work as soon as possible.

2.2.2 GitHub alternatives

To solve this problem that blocked us from efficiently working on the project, we considered two main solutions. The first was to migrate the studio remote repositories from GitHub to Gitlab. Gitlab is a competitor of GitHub that provides a similar service, but unlike GitHub, Gitlab does not have any Git LFS limitations. However, since all members of the team were used to GitHub even before the beginning of the project, this solution was not chosen. It would have implied a new interface, different functionalities and, above all, incompatibility with our continuous integration, which would have forced us to remake it from scratch.

The second solution we considered was to migrate from GitHub to a self-hosted instance of Gitea. Gitea is an open-sourced reimplementation of GitHub which allows you to self-host a Gitea instance unlike GitHub which is software as a service. This solution ensured that there are no limitations at all since it is self-hosted. Moreover, unlike the Gitlab alternative, Gitea allowed to keep Github featured task management tools, interface, and continuous integration compatibility. Indeed, since Gitea is a reimplementation of the GitHub software, its interface is highly similar to the GitHub's one, and its continuous integration is compatible with GitHub actions that we used to make ours.

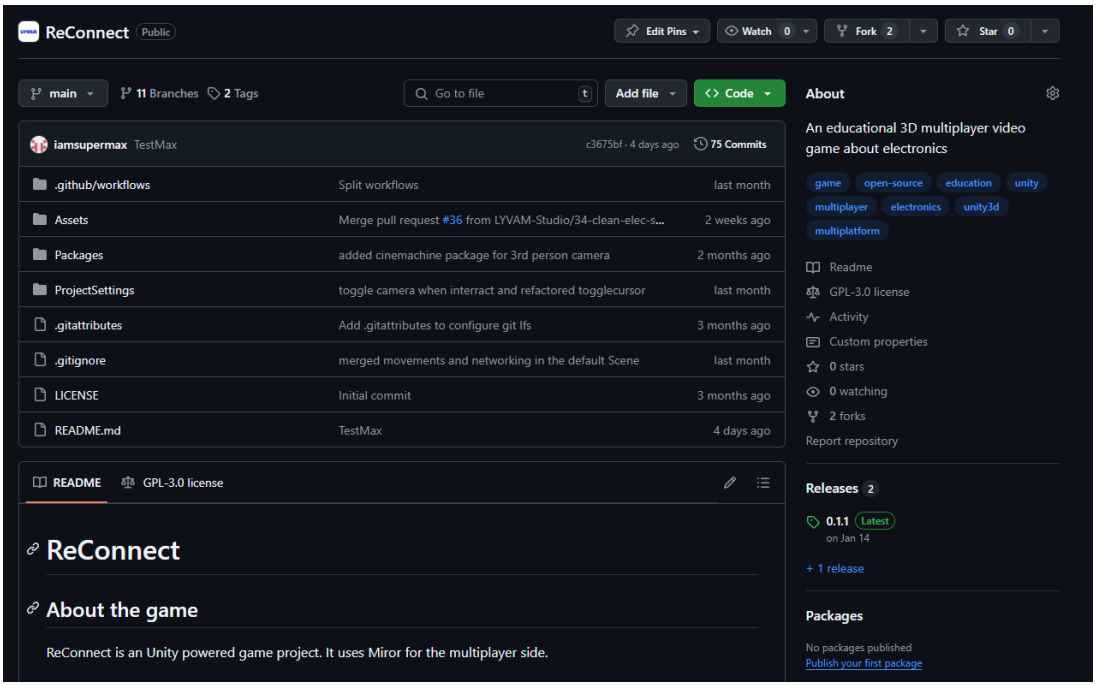

Figure 3: Our git repository previously on Github website

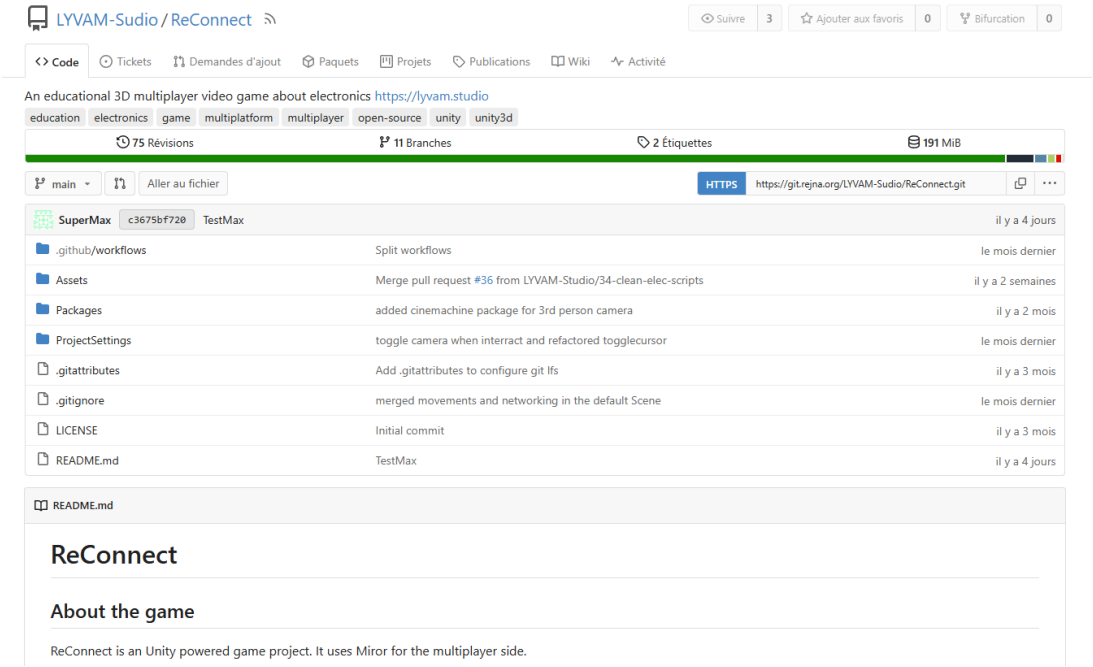

Figure 4: Our git repository now self-hosted with Gitea

Thus, the team created a container on a server owned by the studio and installed and ran Gitea using Docker. The team also set up another container on the same machine to host a continuous integration runner that is not included in the Gitea Docker instance. The runner also runs on a docker and allows continuous integration to be executed. Our Gitea instance is accessible to anyone at the following URL: https://git.rejna.org/.

3 Electricity Implementation

3.1 A breadboard in our game to draw circuits

3.1.1 The breadboard concept

A breadborad, also called a solderless breadboard or protoboard, is a construction base used to build electronic circuits. It is often used in schools to teach electronics because it does not require any welding. It consists of a board with holes wired according to a standard pattern, in which wires or electrical components can be plugged.

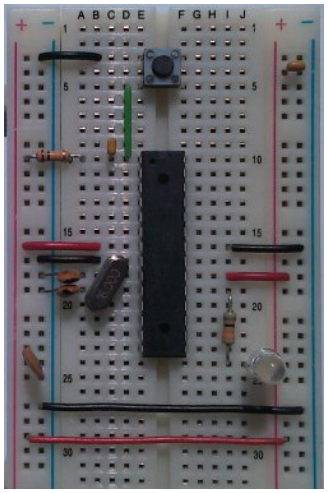

Figure 5: Example of a breadboard

In the case of ReConnect, the breadboard is a crucial component of the game. Indeed, the core of the game consists of electronic puzzles that take place on a virtual breadboard. Therefore, we had to implement the easiest to use and most intuitive breadboard so that the player can easily manipulate it and explore the multiple possibilities.

3.1.2 The breadboard implementation

Because we wanted the easiest to use and most intuitive breadboard interface in the game, we made our breadboard different from the real ones. The breadboard of ReConnect is made up of a solid plane on which there are nodes, physically represented by spheres. These nodes play the roles of multiple holes in the real breadboards: multiple wires (up to 8) and components (up to 4) can be wired to it.

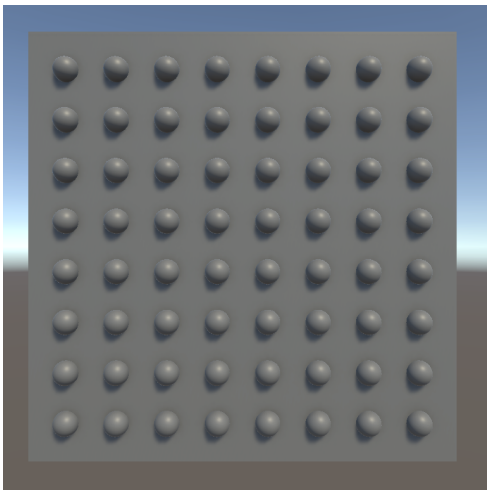

Figure 6: Screenshot of the ReConnect breadboard

3.1.3 Wires implementation

The first thing to implement in addition to the breadboard was the wires. In the Unity project, a wire is a prefab object. This allows any script to create wires. This will be needed especially by the breadboard script.

To create a wire, the player simply has to hold the mouse from one node to another. When the mouse is held and collides with a node, the breadboard script creates a new wire if a wire between the previous node and the current one can be created. A wire can be created between two nodes only if they are sibling nodes or diagonal nodes.

A created wire can be destroyed in two ways. The first is simply by clicking on it. The second is to try to create another wire on top. In the latter case, the player enters a deletion mode in which no wire can be created, and every wire encountered is deleted.

3.1.4 Dipoles implementation

Currently, there are two types of dipole with similar implementations: resistors and lamps. As for the wires, they are implemented using two prefabs, one for each type. However, their behavior is slightly different from that of the wire.

First, the dipoles can neither be created nor deleted. They can only be moved and rotated. In the long term, dipoles will be objects contained in the player's inventory. The player will be able to move them from its inventory to the breadboard and vice versa. These functionalities will be implemented in the future as they are part of the object interaction task.

A dipole can be moved with a basic drag-and-drop mechanic. When the dipole is

released on the breadboard, it is automatically clipped to the nearest valid position. In the case where the nearest position is either already occupied by a dipole or outside of the breadboard, the dipole's position is set to the previous one

3.2 Circuit graph representation

3.2.1 Researches

The implementation of the circuit as a data structure and the traversal of this structure to compute the intensity and the tension were a great challenge for the team. Therefore, Yoann and Adam, respectively, the person in charge and the substitute of the implementation of electricity, decided to try different approaches, each on his own.

Adam saw the circuit as a recursive data structure. In its implementation, a circuit is either a combination of sub-circuits in parallel or a combination of sub-circuits in series. This implementation considers that a dipole is a sub-circuit of either a parallel circuit or a series circuit. Therefore, the end of the recursion of the circuit data structure is reached when a dipole is reached. This vision of the circuit allows for a very easy traversal that eases the calculation of the intensity and the tension going through every component in the circuit. However, we did not keep this implementation because we did not manage to develop an algorithm to build a circuit from the position of the prefab instances of the wires and dipoles.

Thus, the implementation of the circuit data structure and its traversal of the game is the one designed by Yoann. It is detailed in the following sections.

3.2.2 The graph data structure

An electrical circuit can be assimilated to a graph structure. A proper definition of a graph from the Wikipedia page is:

> *" In discrete mathematics, particularly in graph theory, a graph is a structure consisting of a set of objects where some pairs of the objects are in some sense "related". The objects are represented by abstractions called vertices (also called nodes or points) and each of the related pairs of vertices is called an edge (also called link or line). Typically, a graph is depicted in diagrammatic form as a set of dots or circles for the vertices, joined by lines or curves for the edges. "*

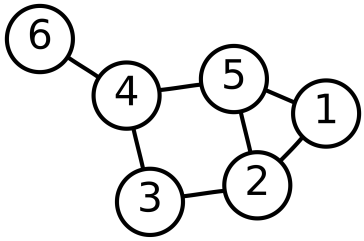

Figure 7: Illustration of a graph

For practical reasons, we chose to force the circuit to have only one generator which is outside of the breadboard. The breadboard contains an entry point node and an exit point node. Therefore, the graph representing the circuit goes from the entry point to the exit point and does not contain any generator.

We have decided to organize our data representation of a circuit using the following data structures:

1. **Graph**: contains a list of *Vertices*, an entry point of type *CircuitInput*, an exit point of type *CircuitOutput* and a target component of type *Vertex*. The entry point is a node where the current enters the circuit. It is the positive pole of the tension generator. The exit point represents the negative pole of the generator of the circuit. The target component is the tested dipole in the exercise, i.e. it is the component that has to receive a given tension and intensity for the player to succeed the level.
2. **Vertex**: contains a list of adjacent *Vertices*, as pointers.
3. **Node**: a *Vertex* that is not an electrical component but an electrical node. Therefore it does not have any resistance. It has a list of adjacent vertices of length greater or equal to 3. It is used as an indicator during the traversal.
4. **Branch**: contains a list of *Vertices* and 2 *Nodes*. It represents a branch in the circuit, i.e. a group of components in series, between 2 nodes.
5. **ElectricalComponent**: a *Vertex* that is a dipole and has a resistance. It can be either:
   (a) **Resistor**: a component that has for main feature to provide a specific resistance,
   (b) **Lamp**: a resistor that has a nominal tension at which the dipole produces light.
6. **CircuitInput**: a *Node* that has an input tension in volts, which corresponds to the tension produced by the circuit tension generator.
7. **CircuitOutput**: a *Node* that represents the output of the circuit.

Additionally, our circuits will contain switches, but their representation in the graph will be the one of their state: an open switch make the branch open, therefore the linked components will have a missing adjacent vertex; a closed switch is simply considered as a wire.

3.2.3 Graph traversals

We managed to build a list of branches from the circuit graph using a *Depth-First Search (DFS)* algorithm. A branch represents a list of components located between 2 nodes. The components of the same branch are then considered **in series**. This definition is useful because it permits to attribute a resistance to a branch. The total resistance of two or more resistors connected in series is equal to the sum of their individual resistances.

Then our goal was to simplify any circuit using parallel and series association of resistors to have the total resistance of the circuit. To achieve this, we used the two following formulas:

$$R_{eq} = \sum_{i=1}^{n} R_i = R_1 + R_2 + ... + R_n$$

Equation 1: Association of resistors in series

$$\frac{1}{R_{eq}} = \sum_{i=1}^{n} \frac{1}{R_i} = \frac{1}{R_1} + \frac{1}{R_2} + ... + \frac{1}{R_n}$$

Equation 2: Association of resistors in parallel

Then, using Ohm's law, we are able to deduce the intensity traversing the circuit, knowing the input tension and the total resistance of the circuit.

In our case U is the tensions at the circuit terminals (e.g., 230 Volts), R is the equivalent resistance that we computed and I is the current that goes through the circuit.

3.2.4 Issues and resolution

We faced numerous issues with this traversal. In fact, we needed to simplify the circuit graph using the methods discussed just above **without** erasing our original circuit.

$$U = R \times I \iff I = \frac{U}{R}$$

With: $\begin{cases} \text{U} & \text{the tension in Volts} \\ \text{R} & \text{the resistance in Ohms} \\ \text{I} & \text{the intensity in Amperes} \end{cases}$

Equation 3: Ohm's law

However, we needed to delete branches in parallel once they were merged. Thus, we created a copy of the list of branches from our initial graph to be able to delete branches from this list without affecting our graph data.

We also needed to create new branches to replace the parallel ones, but we faced some issues trying to keep our original graph intact. This required editing the list of branches and the list of adjacent components of the nodes, which was highly complicated while preserving the integrity of our original graph.

Ultimately, to resolve our multiple issues, we decided that the traversal destructive process could be tolerated, since we would restrict our circuit exercises to the comparison of the intensity and tension of a specific dipole that the player cannot edit. Additionally, we will perform the graph traversal each time the player submits its answer, so we do not need to keep track of the graph once the traversal has been made.

4 First levels in our game

4.1 Circuit 1: components in series

We developed a first circuit to explain to the player how the components are affected by electricity when they are in series. The circuit is composed of a resistor and a lamp. The idea is to explain to the player that the lamp nominal tension is 100 Volts, meaning the lamp optimally needs 100V to light up.

We preloaded the circuit with the wires in the good places, leaving only one spot to put a resistor. The player's goal is to choose the right resistor that will provide the appropriate tension to the lamp.

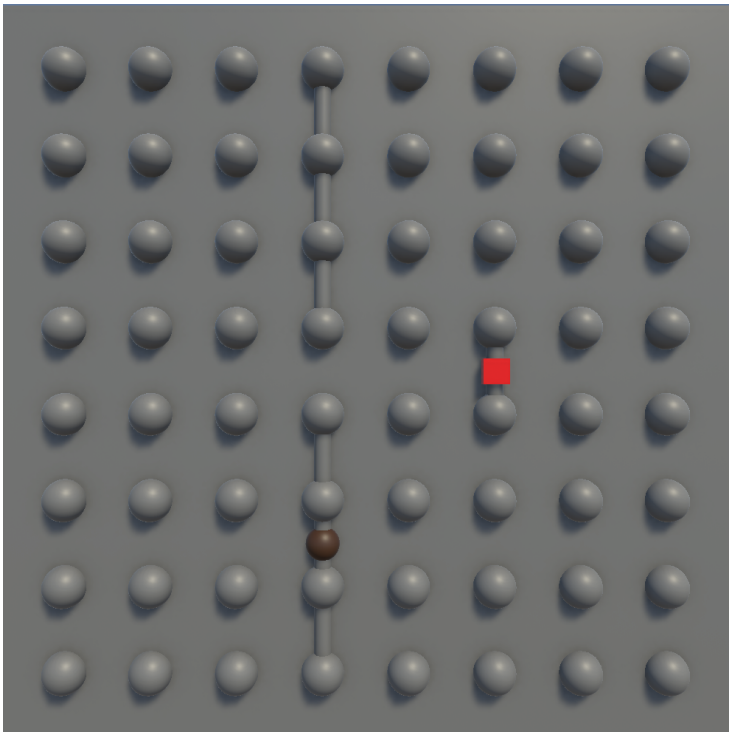

Figure 8: The first circuit about components in series

To assist the player and provide educational resources before asking him to solve the puzzle, we intend to give a lesson on the association of resistances in series, how voltage is distributed across dipoles and the current flowing through the same branch of a circuit.

4.2 Circuit 2: components in parallel

We designed a second circuit whose goal is to illustrate how current, resistance, and voltage behave in a circuit with components in parallel. The idea is the same as the Circuit 1: components in series, but this time we have a lamp with a nominal tension and 2 resistors to insert into the circuit.

We preloaded the circuit dedicated to this puzzle with wires illustrating a parallel circuit with two spots left empty where the player needs to put the appropriate resistors. The goal of the player is to choose the right resistors to have the right tension between the terminals of the lamp. The success of the player is illustrated by the lamp turning on.

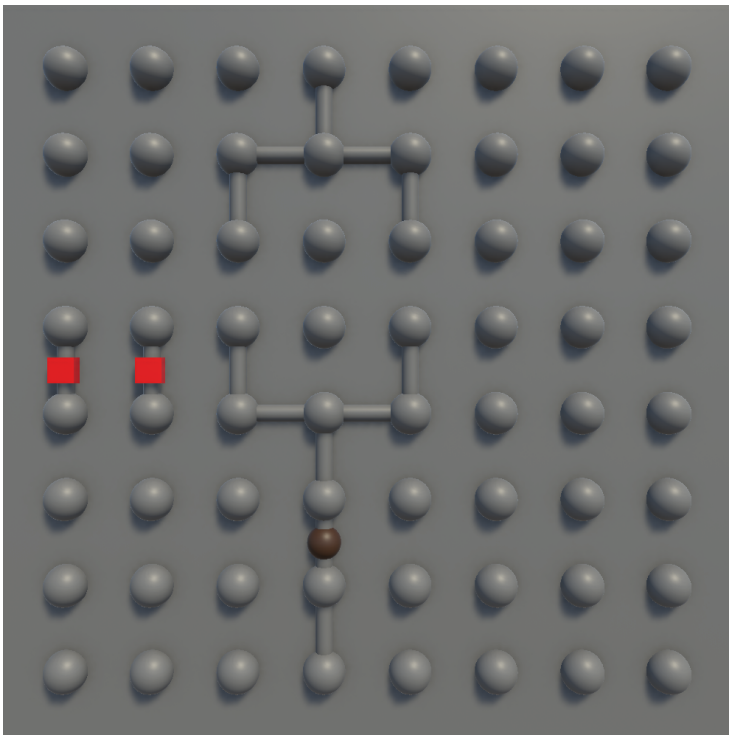

Figure 9: The second circuit about components in parallel

To assist the player and provide educational resources before asking him to solve the puzzle, we intend to deliver a lesson on the association of resistances in parallel, how voltage is distributed across dipoles, and the current flowing through the same branch of a circuit.

5 Menu and user interface

5.1 Current State of Advancement

The menu system has been fully integrated, providing a cohesive navigation framework. The **Main Menu** (Figure 10) serves as the central hub, featuring four core options:

Singleplayer: Launches a local server-client session.

Multiplayer: Directs users to a dedicated server interface (Figure 12) for IP input or hosting.

Settings: Currently a placeholder (redirects to the main menu).

Quit: Terminates the application cleanly.

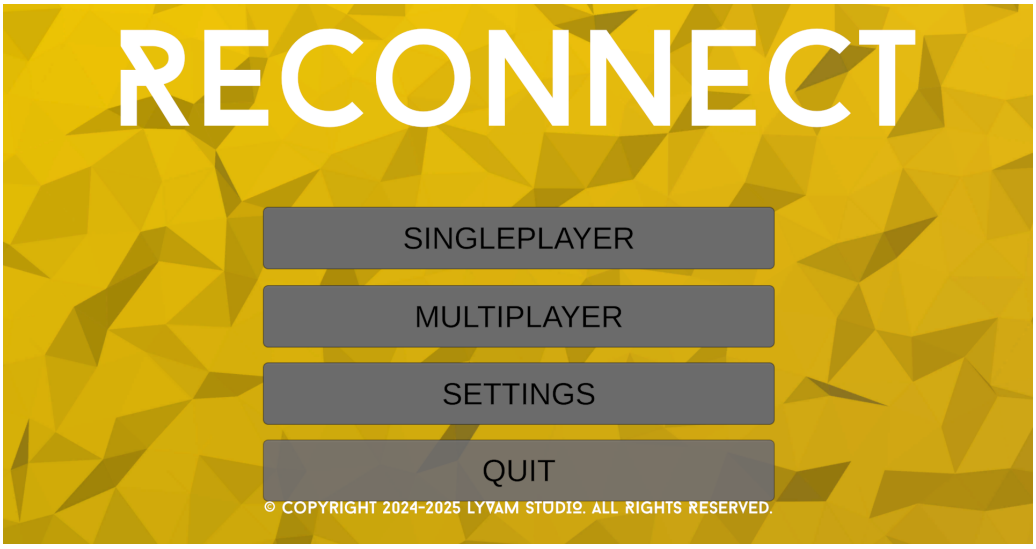

Figure 10: The Main Menu, offering access to core game modes and settings.

During gameplay, pressing the `Escape` key triggers the **Pause Menu** (Figure 11), which immediately suspends all player controls. This action freezes character movement and locks input scripts to prevent unintended interactions while the menu is active. Simultaneously, the mouse cursor is unlocked and made visible, allowing players to interact seamlessly with the menu options: Resume reactivates gameplay by restoring controls and re-locking the cursor to the screen center, Settings (pending implementation) will eventually provide the possibility to customize the Game, and Quit returns users to the Main Menu and turn off the server if the game was launch in multiplayer. The dual functionality—blocking gameplay inputs while enabling UI navigation—ensures a clean separation between active play and administrative actions.

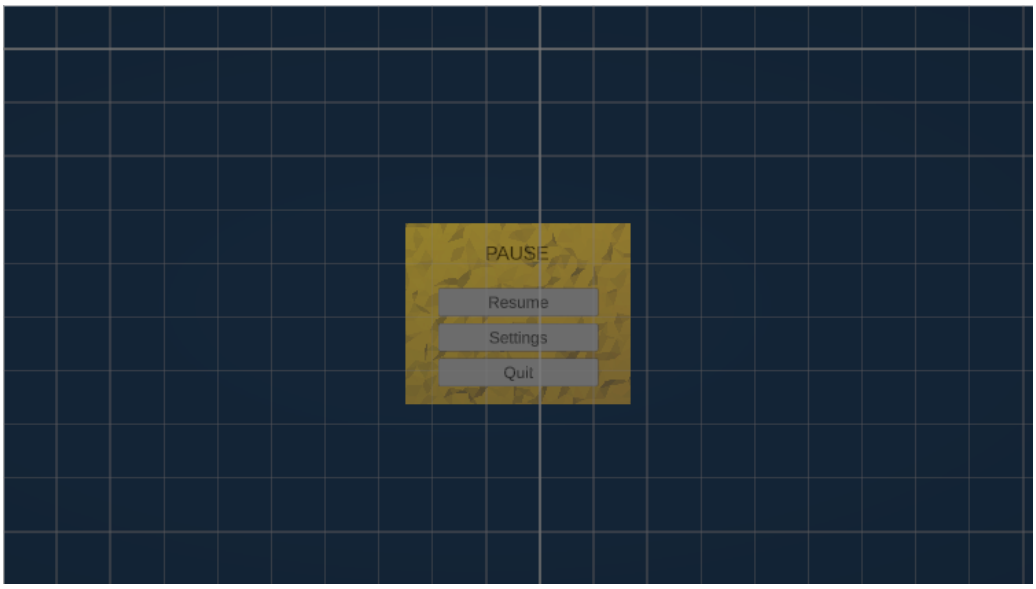

Figure 11: The Pause Menu, activated during gameplay via the `Escape` key.

The **Multiplayer Interface** (Figure 12) supports manual IP entry via an input field, with buttons for joining external servers, hosting sessions, or navigating backward. Transitions between menus are instantaneous, prioritizing functional clarity over visual flair.

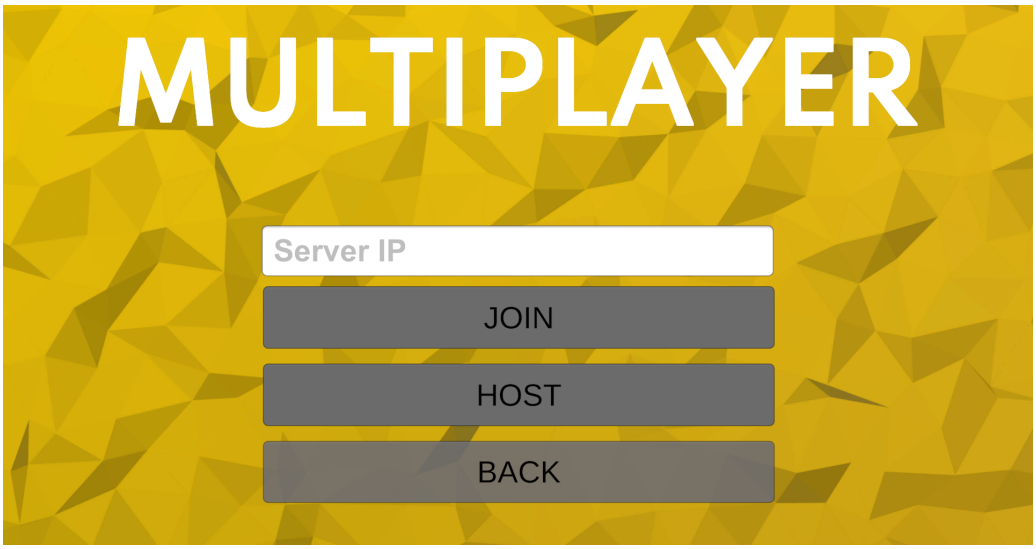

Figure 12: The Multiplayer Interface, enabling server connections or local hosting.

5.2 Difficulties encountered

Several technical challenges arose during the menu system implementation:

- **Multiplayer Menu Replacement**: Migrating from the legacy multiplayer interface caused unexpected scene-loading issues. The new menu occasionally failed to initialize correctly, leaving players stuck in empty scenes.

- **Pause Menu Behavior**: Initially, the pause menu did not properly block player movement or unlock the mouse:
  - **Movement Lock**: Player input scripts continued processing keyboard events despite the menu being active.
  - **Mouse Handling**: The cursor remained locked to screen center. Added `Cursor.lockState = CursorLockMode.None` and `Cursor.visible = true` when opening the menu, with reversal on resume.
  - Created a dedicated `Pause` function in the Input System to avoid conflicts with gameplay controls.

5.3 Future Improvements

The **Settings Menu** remains a priority, requiring implementation of *key rebinding* and *graphics/audio customization*. **UI/UX polish** will focus on smoother transitions (e.g., animated overlays) and *responsive scaling* to accommodate varied screen resolutions. These adjustments aim to reduce visual abruptness while retaining the system's current efficiency.

6 3D Implementation

6.1 3D Unity Assets

In Unity, an asset is a representation of an item that can be used in a game or project. This may include files created outside Unity, such as 3D models, audio files or images, or items created inside Unity, such as a host controller or audio mixer. The main advantage of using assets is the 3D implementation of component for the game, which widely accelerates its production. However, some problems arise in their implementation. First, finding an open-source asset corresponding to our criteria is uncertain. Then, the asset's version compatibility can alter its implementation.

Up to today, we decided to introduce the "Sci-Fi Styled Modular Pack" asset, the "Electrical shield" asset. The first one is going to be used to create the interior of our crashed spacecraft. We have some struggles with its implementation as unity does not recognize entirely the elements of the asset as walls or grounds. The second one is going to be used to simulate a electrical shield. It will be linked to the first level of our game.

Figure 13: Spacecraft crashed interior

In addition we have selected other assets that we have not considered necessary to implement at this time. It is assets of entities creations, environment decoration and assets that increase the immersion of the player.

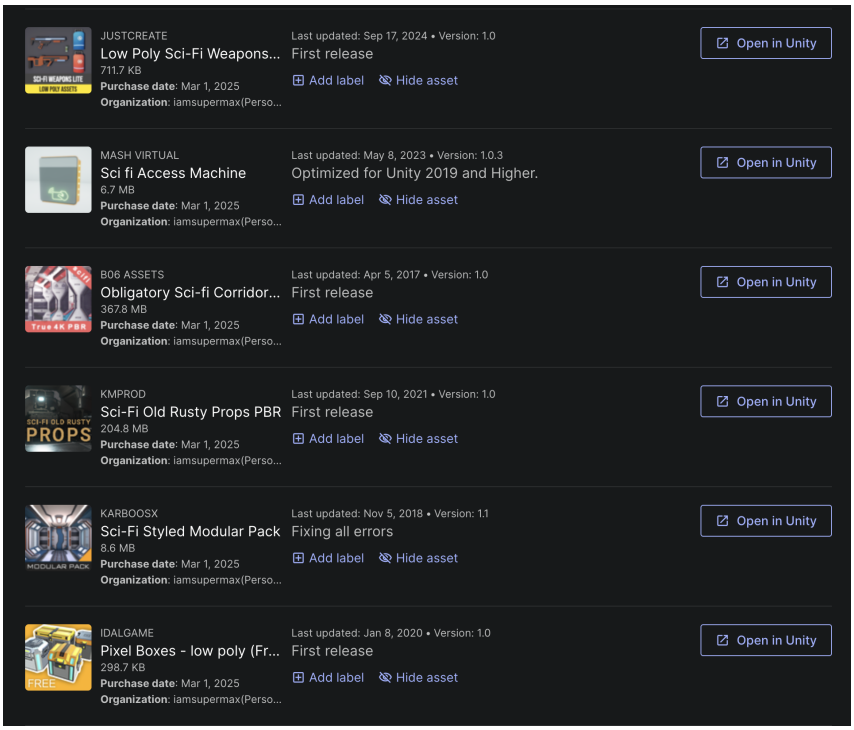

Figure 14: Extract of our 3D Assets selection

6.2 3D Maps

Alternatively at the 3D Assets we wanted to create a map from zero. We mostly used Blender to create and make a render of the map.

We first sought to create a random ground regarding volumes (hills, hollows) and then covered it with an open-source earth texture. To ensure that the patterns weren't repetitive due to the map's large surface area, we used a Voronoi diagram to create random orientations with subspaces of the partition that is the plane.

We then modeled trees to arrange them on the map. Finally, we laid out various assets on the ground, in large numbers and with a random pattern, to create ground vegetation. We used a grass asset, a pebble asset, a flower asset and a mushroom asset.

The main difficulty was to implement the creation. In fact, once the map was uploaded, we had to create a Terrain collider which is an invisible shape used to handle physical collisions for a Terrain GameObject.

Two problems occurred, they have been fixed, however we are not satisfied of the result. Firstly, by building a most realistic map, we could not import it in Unity. We had to revise our design and degrade the quality. Fortunately, once the revision has been done, we were able to implement it in Unity. Secondly, because of an incompatibility

of the assets, all the grass and flowers had disappeared during the importation in Unity. However, no solution has been found yet.

Figure 15: Screenshot of our 3D map design in Blender

7 AI

Today, when it is mentioned the use of an AI, people most often think about MistralAI, ChatGPT and Llama. However, these AI are generative and general. First, they are made to multitask. Second, They are way too big for a game like we are creating. Hence, our AI will focus on one task: pathfinding.

7.1 Pathfinding

Pathfinding is an important subset of AI planning, widely applied in GPS such as Google Maps and Waze. We will use this AI to create autonomous entities like game NPCs. The goal of pathfinding is to determine the most efficient route between two points in a given environment. In our case, we aim to implement this AI to create autonomous mobs that can navigate from a point A to a point B.

At this point, it seems easy to implement, we have to look for the best path between two nodes in a graph. Notwithstanding, pathfinding becomes a complex problem when you try to take into account various additional constraints as real-time execution, resource constraints and environment. Hence, we decided to use the "A Star" Algorithm (A*).

7	6	5	6	7	8	9	10	11		19	20	21	22
6	5	4	5	6	7	8	9	10		18	19	20	21
5	4	3	4	5	6	7	8	9		17	18	19	20
4	3	2	3	4	5	6	7	8		16	17	18	19
3	2	1	2	3	4	5	6	7		15	16	17	18
2	1	0	1	2	3	4	5	6		14	15	16	17
3	2	1	2	3	4	5	6	7		13	14	15	16
4	3	2	3	4	5	6	7	8		12	13	14	15
5	4	3	4	5	6	7	8	9	10	11	12	13	14
6	5	4	5	6	7	8	9	10	11	12	13	14	15

Figure 16: Astar graph

At first sight, implementing pathfinding might seem straightforward: it involves searching for the shortest path between two nodes in a graph. Notwithstanding, the problem quickly becomes complex when considering additional constraints such as real-time execution, limited resources, and environmental changes. Hence, we have chosen to implement the A-Star (A*) algorithm due to its efficiency and optimality.

The A* algorithm combines elements of Dijkstra's algorithm and Greedy Best-First Search by using a cost function:

$$f(n) = g(n) + h(n)$$

where $g(n)$ represents the current cost from the start to n and $h(n)$ is the heuristic function estimating the remaining cost until the goal. This cost accumulates as the algorithm moves through the graph. The function $f(n)$ ensures a balanced approach between exploring promising paths quickly $h(n)$ and maintaining an accurate cost calculation $g(n)$. This makes A* the most efficient algorithm.

7.2 Implementation: A-star

A classic implementation of A* is a graph or a matrix, which is a 2D implementation. Nonetheless, Reconnect is a 3D game, thus the implementation of A* has to be modified to our game.

A close implementation of a graph is a Navigation Mesh (NavMesh) which is a set of polygons. We decided to implement A* using this way. By comparison to a 3D matrix, a voxel, a NavMesh is light and less detailed but can integrated sloping areas which is sufficient to understand the world. Thus, NavMesh is more optimized than Voxels since it calcules way faster. For example, NavMesh implementation is used in Assassin's Creed, Skyrim and Call of Duty. These are complex games where optimization are crucial.

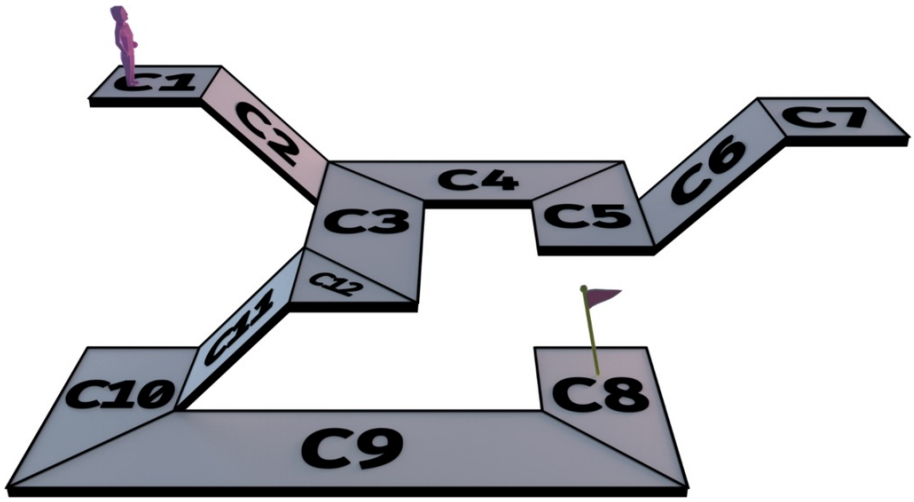

Figure 17: Representation of mesh navigation

8 Overall progress

The following table shows the current progress on the project in comparison with the advancement that we predicted in our new Gantt Chart.

Story & Cutscenes	80%	80%
Menu	75%	75%
HUD	0%	0%
Physics	80%	85%
Tutorials	0%	0%
Music & FX	0%	0%
Level design	30%	30%
3D modeling	40%	50%
Interactions	15%	0%
AI	15%	15%
Multiplayer & networking	100%	100%
Website	100%	100%
Communication	0%	0%

Table 2: Current advancement compared to the planed advancement

9 Conclusion

For this second technical defense, in addition to the functional base of our game in terms of networking and player and camera movements, we added a map, a menu and implemented our simulation of electricity with 2 levels.

The menu system now offers a clean and functional interface for players to navigate the game seamlessly. Key features like the main menu, pause menu, and multiplayer interface are fully operational, enabling smooth transitions between gameplay and administrative actions (e.g., hosting or joining sessions). While polish like settings customization remains for future work, the current implementation prioritizes clarity and responsiveness.

We still have some work to do on the implementation of electricity in our game, by adding information about resistance and tension, a list of resistors to use, tutorials, instructions and obviously launch an action when the player succeeds a level. However, most of the work has been done regarding this task.

Our website is still available at the URL lyvam.studio. It contains information about our studio, our team, and our project, including an overview, this report, and downloadable versions of the game for Windows, Linux, and MacOS.