0r!g!ns

Technical Report n°2

A1

EPITA 2030 Promo

CERIBAC Noah
BELIARD Antonin
TONNOT Antoine
GLEMET Adam

Document Version: V1.1.0
Game Version: V 1.0.2

Table of content

Introduction

This technical report aims to present the progress of the Or!g!ns team project since the first technical presentation.
Over the past few months, our group has begun implementing the first levels, the networking system, player-related logic, as well as textures to make the game more visually appealing and original. This report will present the difficulties we have encountered, the successes achieved in the development of the game, as well as our plans for the coming months.

General Information

I. Specificities of the game

Type of game :				
Action/Adventure	Battle Royale	Beat them all	fighting	Simulation
FPS	MMORPG	MOBA	Party Games	Survival Horror
Platformer	Puzzles	Reflection	Rogue Like	TPS
RPG	RTS	Sandbox	Shoot them up	Racing
Others :				

General features of the game :				
IA :	Wander	Attack	escape	"Path Finder"
Multiplayer :	Cooperative	Battle (2-4)	Massive	
Network :	P2P	Lan	Online	
Graphic features :				
Dimensions :	2D	3D	Others :	
Special features :	Stereoscopy	AR	VR	
graphics :	Personal	Custom	Existing	
Details :				
Sound characteristics :				
Music :	Personal	Custom	Existing	
FX :	Personal	Custom	Existing	
Details :				
Other features :				
Website :	Personal	Custom	Prefabricated	

II. Tasks planification

Tasks	Report 1	Report 2	Final Report
Basic Logic	100%	-	-
Level Mechanics	25%	50%	100%
NPC AI	25%	50%	100%
Loot System	25%	50%	100%
Tower/Dungeon System	50%	100%	-
Sandbox Mode	-	-	-
Networking	95%	100%	-
Website	50%	100%	-
Level Design	10%	50%	100%
Character Design	75%	100%	-
Texturing	50%	100%	-
GUI	5%	50%	100%
SFX/VFX	0%	50%	100%
Bug Test	100%	100%	100%

III. Tasks distribution

Tasks	noah.ceribac	antonin.beliard	antoine.tonnot	adam.glemet
Basic Logic	R	S	-	-
Level Mechanics	S	S	S	R
NPC AI	S	S	R	-
Loot System	S	R	-	-
Tower/Dungeon System	R	-	S	-
Sandbox Mode	-	-	-	-
Networking	S	R	-	-
Website	-	-	-	R
Level Design	S	S	R	S
Character Design	-	-	-	R
Texturing	R	S	-	-
Animation	R	S	-	-
GUI	S	R	-	-
SFX/VFX	-	-	R	-
Bug Test	R	R	R	R

Summary of Technical Report No. 1

The first technical report introduced the foundations of the Or!g!ns project and the game Ascendant Trials. It presented the main concept: a cooperative puzzle-platformer inspired by mythology, with progressive levels and boss fights.
The team defined the project objectives, aiming to create a game that is accessible, engaging, and fully functional by the end of the year. A global planning and task distribution were also established to organize the work between members.
From a technical point of view, the initial codebase, core game systems, and main mechanics were designed. Early work also included level design ideas, NPC behaviors, and the first version of the website.
Overall, this report set up the structure, goals, and initial development needed to start building the game.

Project Advancement

I. BELIARD Antonin

At the first defense, my work was mainly focused on the multiplayer infrastructure around the project: the website deployment, the domain, the WebSocket lobby server, the health and status endpoints, and the admin dashboard used during tests. This first step gave the project a real online base, but the multiplayer system was still mostly separated from the game itself.

For the second defense, my main objective was to connect this networking stack to the actual game and make it usable in a real cooperative session. Between January 11 and March 17, 2026, I worked on both repositories used by the project: the networking repository, which contains the FastAPI room server and the website files, and the main game repository, which contains the Ursina game. The result is that the multiplayer system is now integrated into the game flow.

1. Progress on the server side

The biggest change since the first defense is that the lobby server now manages the full life cycle of a private room instead of acting as a simple message relay. At the previous milestone, the server already accepted connections and forwarded messages. It is now much more structured and follows explicit room rules.

I separated the networking rules from the room state by introducing dedicated modules. The protocol module centralizes the constants and helpers used by the multiplayer system: room-code generation, player identifiers, session tokens, allowed client message types, timing limits, JSON formatting, and level progression. This keeps the networking logic in one place and avoids duplicating rules across the server.

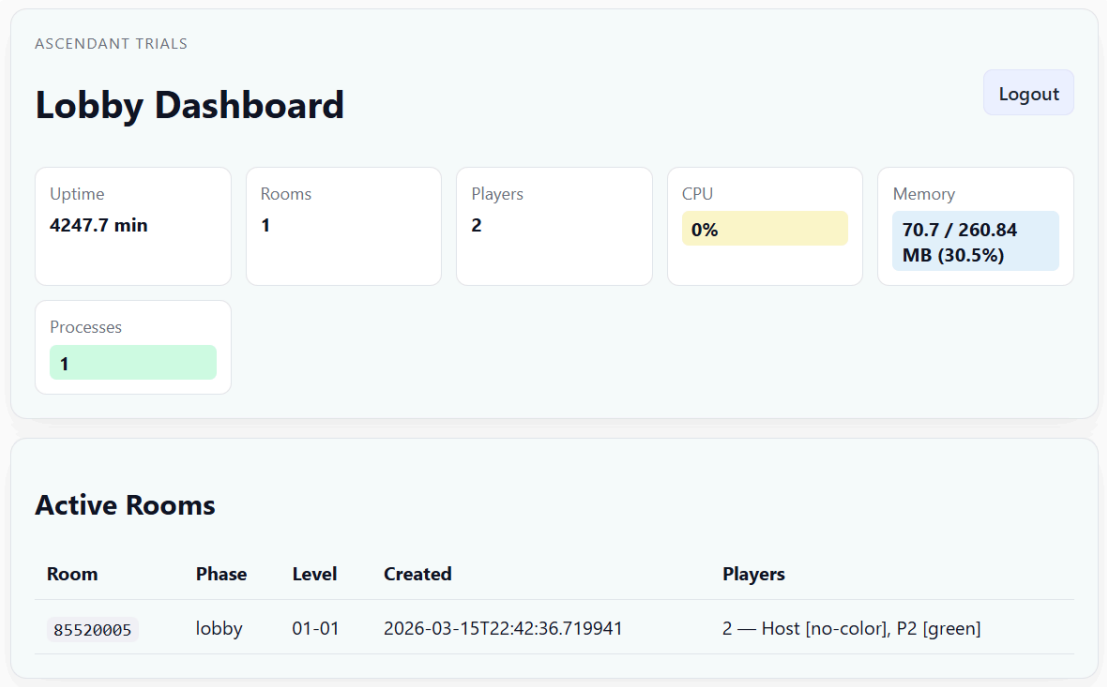
On top of this, I implemented a RoomManager responsible for the real multiplayer state. Each room now stores its code, host identifier, current phase, current level, start time, generation, and player sessions. Each player session keeps the information needed by the room flow: seat number, color, ready status, loading state, connection state, and timing data used to limit some messages.

This allowed me to turn the server into the source of truth for room state. It now manages room creation, room join, token-based session access, the host role, the ready system, color selection, synchronized level loading, room reset, room closing, and progression to the next level.

The room rules are also much clearer than before. A room must contain between two and four players. The host owns the room. Guests cannot send host-only actions. The game cannot start if guests are not ready or if all players have not chosen a unique color. If the host disconnects, the room closes. If a guest leaves during gameplay, the room is reset to avoid inconsistent state.

Another important change is the move away from player names. Instead of relying on custom names, the server now assigns each player a seat and expects a chosen color. This makes the lobby interface simpler and avoids conflicts or unnecessary complexity.

The HTTP API has also evolved. The client now creates or joins rooms through dedicated HTTP endpoints, then connects to the room through a WebSocket using a session token. This is cleaner than exposing too much logic directly in URL parameters and matches the current multiplayer flow more naturally.



I also kept and improved the monitoring tools introduced earlier. The /health and /api/status endpoints are still available, but they now expose richer information like room phase, current level, room count, and player count. The dashboard benefits from this as well and is now a real debugging tool.

2. Integration into the game

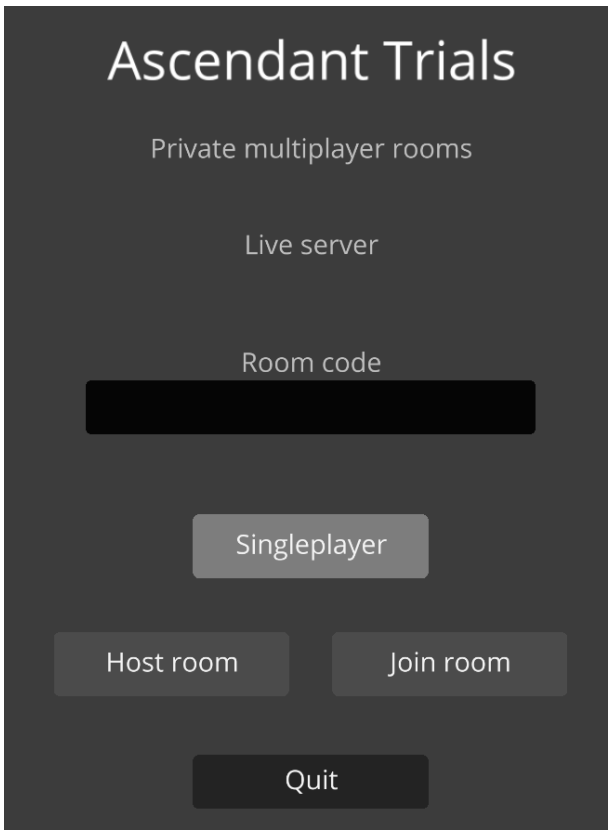
The second major part of my work was to integrate the multiplayer server into the main game repository. The original project structure was still mainly organized around single-player execution, so I had to reorganize part of the application to make multiplayer fit into the project.

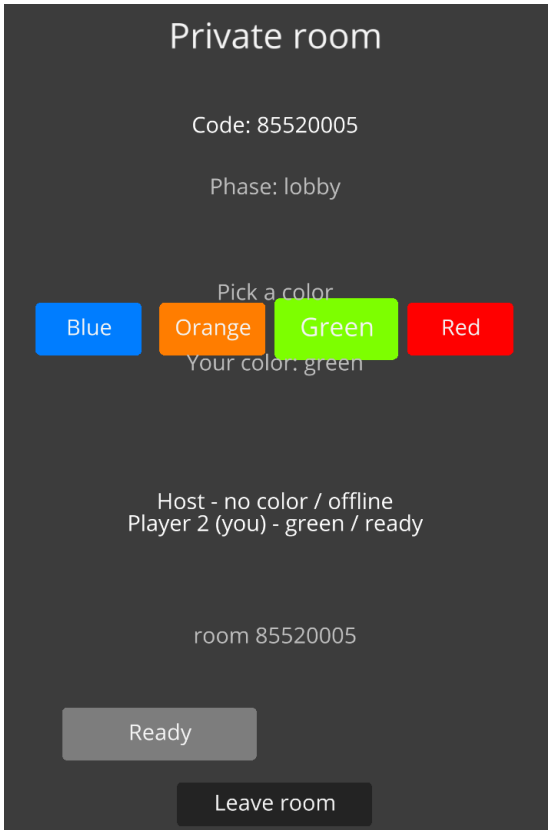
I created a dedicated client layer that communicates with the room server. This client handles room creation and join requests through HTTP, opens the WebSocket connection, runs it in a separate thread, stores outgoing and incoming messages in queues, handles transport errors, and keeps a small server-time offset through ping/pong exchanges. This time synchronization is useful because level starts and animated hazards should remain coherent between players.

On top of this client, I implemented a MultiplayerRuntime responsible for the in-game multiplayer session. This runtime knows whether the local player is the host or a guest, keeps the current room state, reacts to room events sent by the server, loads levels when instructed, creates the local and remote players, and sends either player inputs or world snapshots depending on the role of the client.

The model used here is host-authoritative gameplay. The dedicated server remains the source of truth for room state, but the host simulates the actual gameplay. Guests send their inputs to the host through the server, and the host sends world snapshots back to the guests. This is a practical compromise for our game because it gives us real online cooperation without making the dedicated server simulate the full game.

I also refactored the main application flow so that the game can switch cleanly between single-player and multiplayer. The menu now includes room creation, room join, room leave, color selection, ready state, and game start controls. There is also a hidden developer panel that lets us switch between the public server and a local test server, which is very useful during development.


Ascendant Trials
Private multiplayer rooms
Live server
Room code
Singleplayer
Host room
Join room
Quit


Private room
Code: 85520005
Phase: lobby
Pick a color
Blue
Orange
Green
Red
Your color: green
Host - no color / offline
Player 2 (you) - green / ready
room 85520005
Ready
Leave room

3. Synchronization and gameplay adaptations

Once the server was connected to the game, I also had to modify the gameplay code so that multiplayer rules could actually work. This part was needed because forwarding messages is just not enough in a platform game: timing, collisions, failures, and success conditions also need to follow a shared logic.

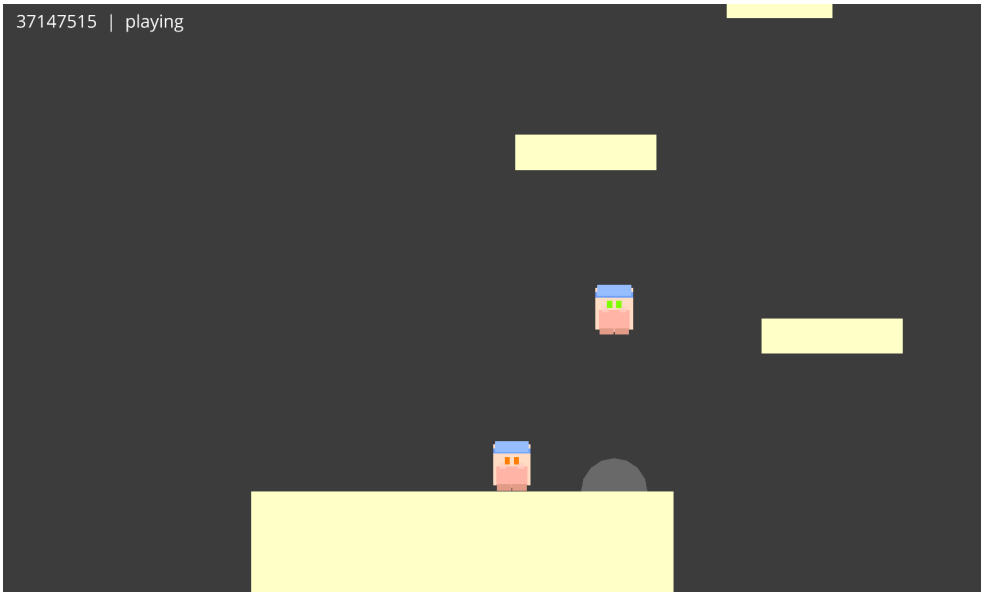
One of the main changes concerns the block and level systems. I extended the block manager so that moving platforms, blinking blocks, damage zones, and level endpoints are registered explicitly instead of being handled only through isolated local behavior. This makes it easier to keep one central rule system for the whole room.

The block manager now uses a shared level clock. This means that animated hazards and moving blocks can follow the same timing reference instead of each client relying only on its own local timing. I also centralized damage and endpoint handling. If one player dies, the whole room must restart the level. If all players reach the endpoint, the room must progress together.

The player controller also required major work: I implemented a custom entity that can handle both local input and remotely injected input. This allows the host to simulate guest players while guests still control their own local character before receiving authoritative corrections from the host.

I also improved the handling of remote players. Guests now keep a local player and receive remote player views for the other members of the room; these views interpolate toward the latest snapshot so that movement does not look like a sequence of teleports.

Finally, I fixed several issues that appeared during this transition. I stopped the player model from being rebuilt too often, corrected jump and ceiling collision behavior, and removed unnecessary interpolation for the local player so that movement no longer slides. These fixes were important because a multiplayer feature is not convincing if player movement becomes unstable.



4. Validation and difficulties

The main difficulty of this stage was architectural. The project was not yet fully organized around online multiplayer constraints, so integration required several refactors in the core game flow. I had to separate menu logic, application flow, single-player session management, level loading, and networking responsibilities more clearly than before.

Synchronization was another important difficulty. In a platform game, inconsistent timings are quick to create visible problems, especially with our moving platforms, jumps, and full-room restart rules. Because of this, the work was not just on the networking code: gameplay systems also had to be adjusted so that multiplayer sessions would remain coherent.

I also had to keep the solution lightweight. For now, the dedicated server stores room state in memory and does not simulate the full game. The host-authoritative approach is therefore a practical compromise: it gives us real online cooperation while keeping the server simple enough for our current scope.

From a validation point of view, the current system already supports meaningful tests. We can create a room, join it from another client, select colors, toggle readiness, start a shared level, relay guest inputs, propagate host world snapshots, restart a level after failure, advance to the next level, and reset or close the room when a player disconnects.

5. Current state and next work

At this stage, I consider the multiplayer foundation operational and integrated. The main objective announced in the first report, specifically connecting the networking stack to the Ursina game, has been reached.

However, this is still an intermediate version rather than a final product. Some limits remain clear:

- room state is stored only in memory, so it is lost if the service restarts;
- the host remains authoritative for live gameplay, so a host disconnection still closes the room;
- world snapshots currently focus mainly on player state;
- the multiplayer path still needs broader playtesting over longer sessions and more level cases.

So, my next objective is not to redesign the system again, but to consolidate it: I need to continue testing the room flow, extend synchronized gameplay state where needed, improve robustness in edge cases, and make sure the rest of the game's upcoming mechanics coexist correctly with the networked session model.

6. Conclusion

Since the first defense, my contribution has moved from multiplayer infrastructure and deployment to real integration inside the game. I kept the deployed room server, the status endpoints, and the dashboard, but I also turned them into the backend of an actual in-game room system. The server now manages room rules, player sessions, lobby state, readiness, colors, restarts, level changes, and disconnect cases. On the game side, I integrated the client into the application flow, created the multiplayer runtime, added a lobby interface, adapted gameplay systems to shared timing and shared room rules, and stabilized the player controller for networked use.

II. CERIBAC Noah

During the first technical report, I was responsible for setting up the entire initial codebase, the file and folder structure, as well as all the core algorithms. Since most of my tasks were completed for the first presentation, there was little left for me to do for this one, as the remaining tasks are carried out progressively during the development of the levels.
For this presentation, from a coding perspective, I mainly fixed bugs; however, I also implemented new core logic useful for the progression of the game, as well as two textures. Here is the progress of my tasks in detail:

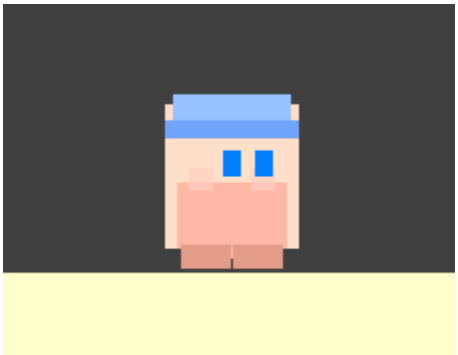
1. Code wise

As mentioned earlier, I mainly fixed bugs such as:
- Collision bugs between the player and blocks
- Security checks that were causing game slowdowns
- And many others

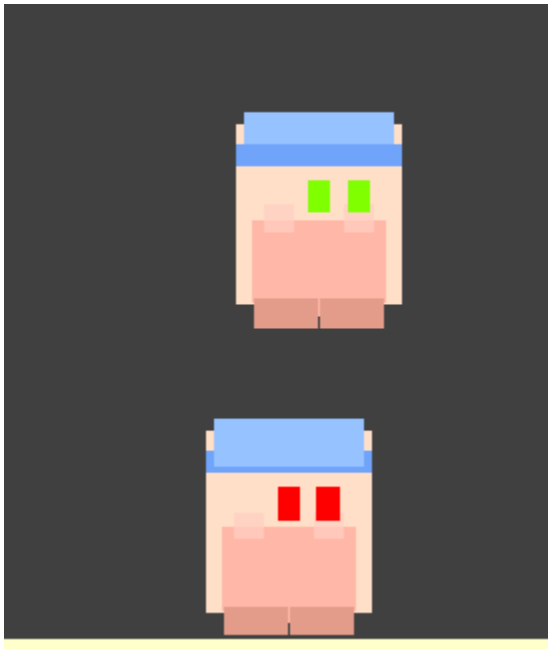
I also added functions necessary for the proper functioning of the game, such as:
- `wpn_endpoint`: a behavior applied to a block, allowing the unlocking of the next weapon fragment in the current level
- `dmg_part`: a behavior applied to a block, allowing it to deal damage to the player

I also introduced support for multiple behaviors per block. Previously, each block was limited to a single behavior; now they can have multiple ones.

Regarding new code not based on existing elements, I added a function `_build_player_avatar()` that allows importing the texture of our player and mascot into the game from a `.json` file.
In solo mode:



In multiplayer mode:



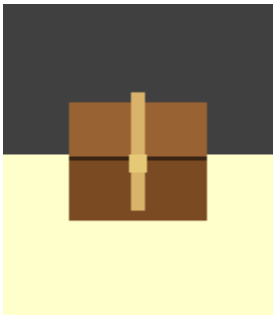
(The eye color allows differentiating the players)

2. Codebase wise

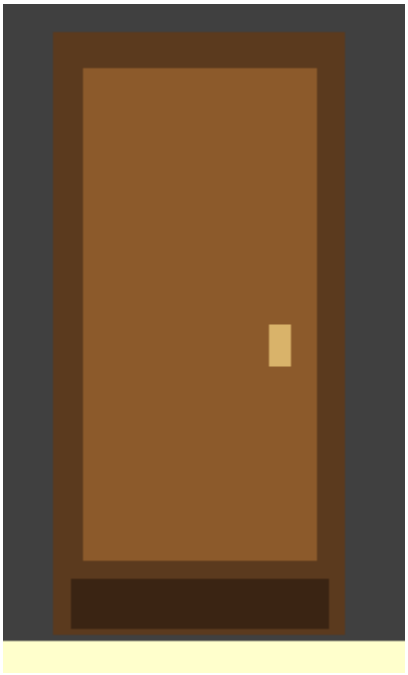
Since the codebase was largely established during the first presentation, I mainly focused on maintaining and adapting it according to everyone's additions. In particular, the networking added by BELIARD Antonin modified a significant part of the file architecture without affecting the folder structure.

3. Texture wise

Regarding textures, I created two new ones:
- A chest texture linked to `wpn_endpoint`



- A door texture to indicate the end of a level, related to a mechanic implemented during the first presentation



Conclusion:

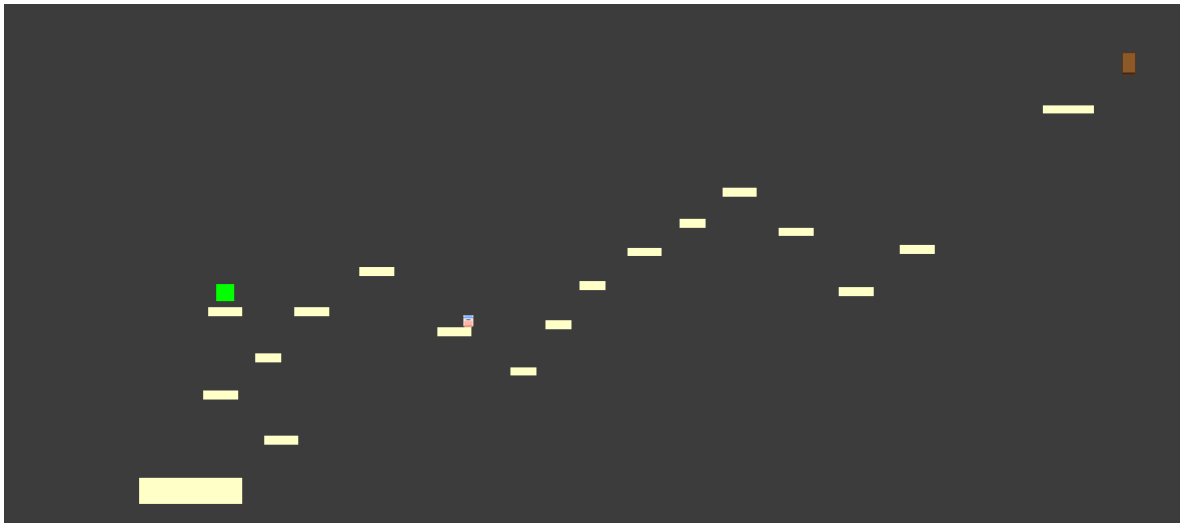
To conclude, many bugs still need to be fixed, both technical ones and those related to textures themselves. They will be resolved for the final submission, as the ongoing major changes to the main game code frequently cause these same bugs to reappear.

III. TONNOT Antoine

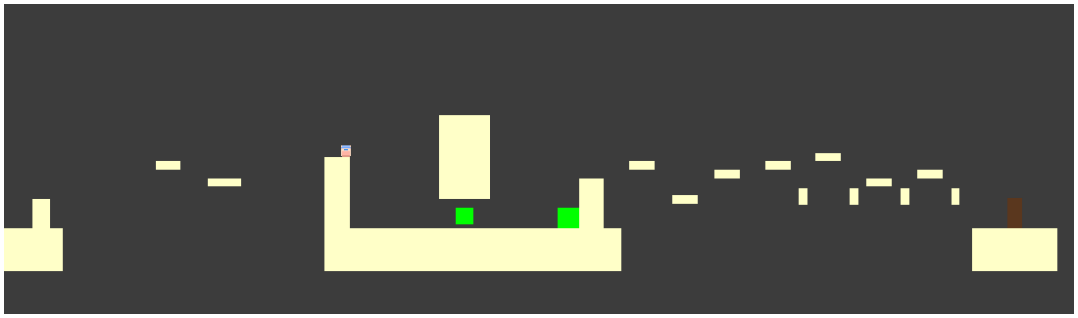
My main responsibility in this project was to design and develop the levels of the game. The game is structured into 5 sections, each composed of 4 platforming levels followed by a boss level. Because of this stricture, I had to carefully plan how each level would be built, assuring a smooth progression in difficulty as the player advances deeper into the game. At the same time, it was important to keep the experience accessible, making sure that even non-gamer players could understand and complete the levels without feeling overwhelmed.

During this half semester, I designed and implemented three different levels, which together form the tutorial section of the game. The goal was to introduce the player to the core mechanics in a gradual and intuitive way. To achieve this, I limited the number of platform behaviours at the beginning and introduced them step by step.

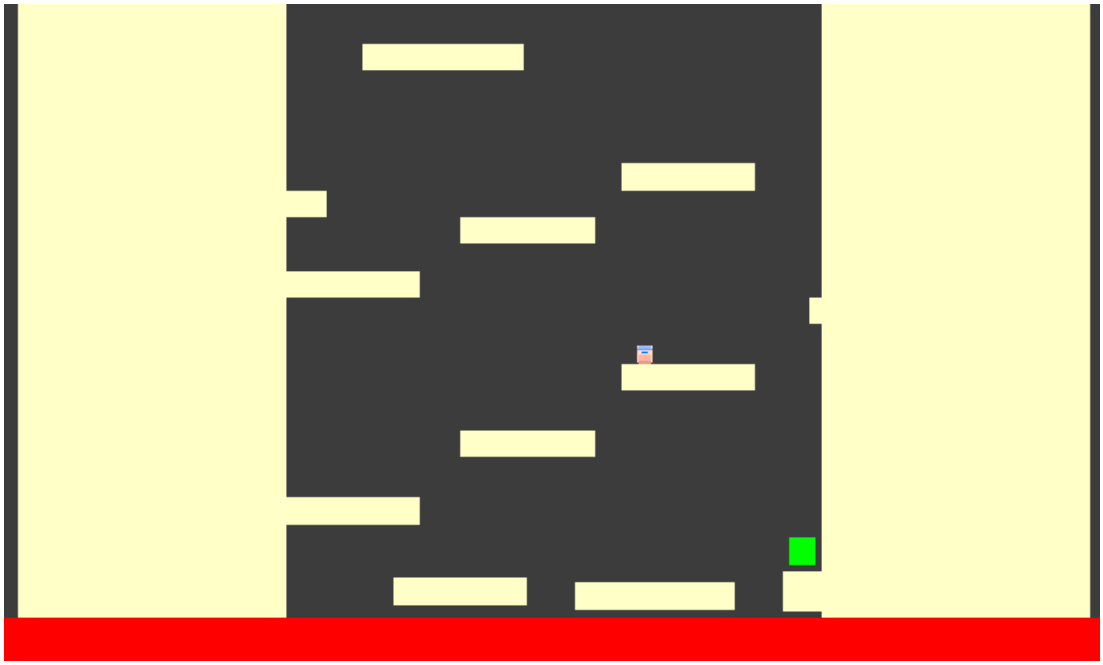
The first level focuses on “blinking” platforms. This mechanic requires players to pay attention to timings as platforms appear and disappear at regular intervals. Through this level, players learn the importance of observation of the level and precise movement, which are essential skills for the rest of the game.



In the second level, I introduced a new mechanic: the sliding platforms. These platforms move from a point A to a point B at a fixed speed, adding an extra layer of challenge. Players must now combine their sense of timing with movement anticipation, adapting to their environment.



The third level is designed as a vertical race. In this level, rising lava constantly pushes the player upward, creating a sense of urgency and pressure. The objective is to reach the top as quickly as possible while avoiding obstacles and making efficient jumps. This level tests the player's mastery of the mechanics introduced earlier and prepares them for the increased difficulty of the next stages.



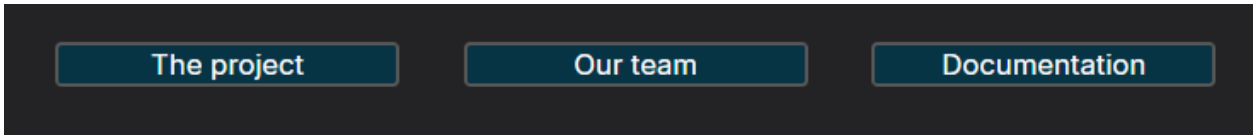
Together, these three levels showcase the main mechanics of the game and establish a solid foundation for the player's progression.

In addition to level design, another major task I worked on was to identify bugs within the game. These issues ranged from collision detection problems, such as the player getting stuck into platforms, to usability issues affecting the overall experience. This process required careful testing, attention to details, and collaboration with the rest of the team to ensure to fix easily and quickly the bugs encountered to ensure a more polished result in the end.

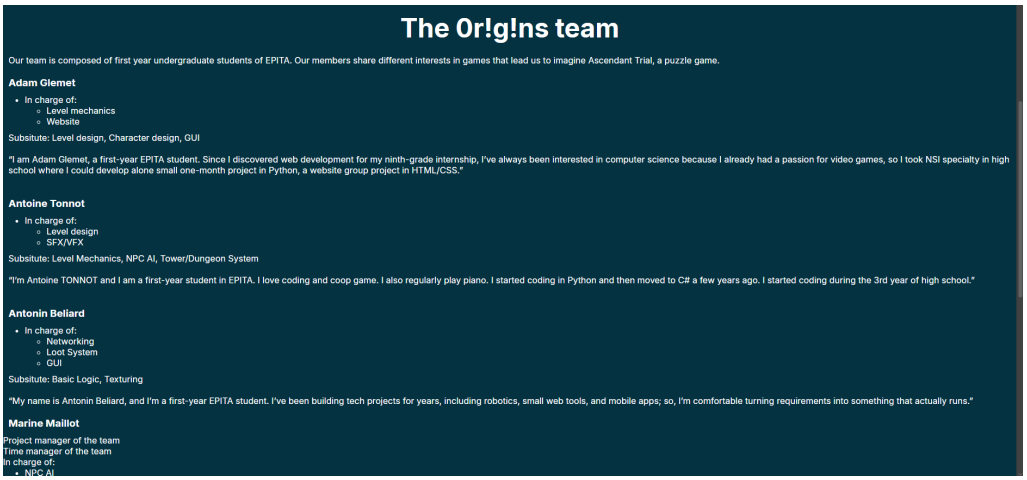
IV. GLEMET Adam

1. Website

We've modified the user interface : we switched from clickable texts to buttons flowing with the rest of the UI, created a background for the team presentation part and added our game logo in the page itself and as the page icon.
The new buttons:



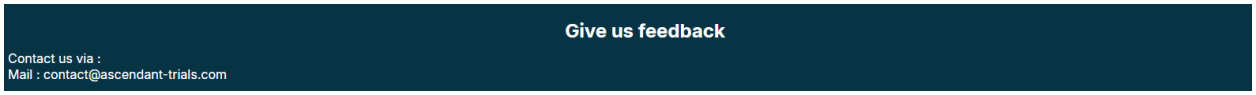
Before:



After:

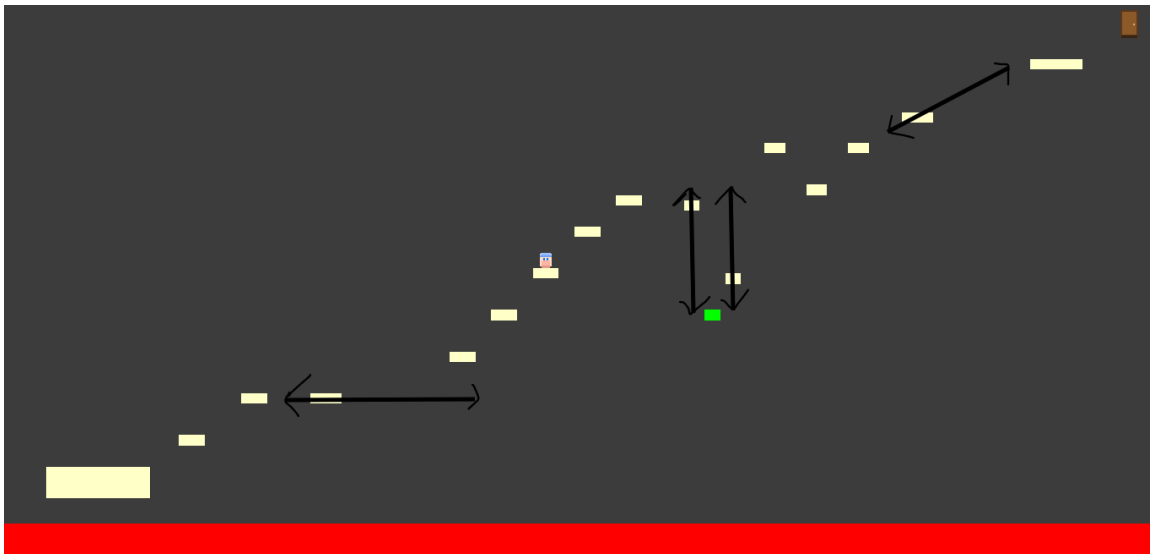
Our new game logo made by MAILLOT Marine:

We added a Feedback section with only a mail address created with our domain name for now, more to be added later.



2. Level Design

For the level design part, I did the 4th level of section 1. I tried to make it in a way to make it harder than the tutorial levels to put on challenge before the boss chamber. In the level I tried to have a way which seems easy but needs a lot of timing and control to achieve it with 5 blinking platform with different intervals one long horizontal-slide platform, two alternating vertical-slide small platforms not so easy but not too difficult to jump on and finally a diagonal-moving one for the final platform with a jump for the door. Here's a global view of the level:



Conclusion

To conclude, the project is progressing well despite delays in certain tasks compared to the initial planning, delays that have been offset by progress in other tasks. Furthermore, as the main tasks of CERIBAC Noah and BELIARD Antonin have largely been completed, they will now assist TONNOT Antoine and GLEMET Adam with the two most time-consuming aspects of the project: level design and level mechanics. Finally, contingency plans have been considered in case the team realizes that certain tasks or features initially planned for the game were too ambitious and cannot be implemented within the available timeframe; should this occur, suitable alternatives will be put in place.