1st defense report

Ascendant Trials

A1

EPITA 2030

Marine MAILLOT
Noah CERIBAC Antonin BELIARD
Antoine TONNOT Adam GLEMET

Contents

Introduction

This technical report aims to present the progress of the Or!g!ns team project since the validation of the functional specification document.

In the last months, our group has developed the concept of our project, a game called Ascendant Trials. So far, we have worked on the codebase and the general design of the game. This report will show the difficulties we faced, the successes we made in our game and what we plan the next months.

General information

In this section, we will explain the technical specifications of our game that we defined at the beginning of the project and how we planned our work two months ago. Then, we will present the team members.

I. Functional specification document

A. Specificities of the game

1. Type of 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~~

Figure 1 – Board of the type of games

2. Features of the game

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~~	

Figure 2 – Board of the features of the game

B. Tasks planification

1. Progress planification

We determine our progress planification based on the defense schedule.

Task progress table			
Tasks	**Defense 1**	**Defense 2**	**Final Defense**
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%

Figure 3 – Board of the progress schedule

2. Tasks distribution in the team

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

Figure 4 – Board of the task repartitions between the Or!g!ns' members

Legend: S – Substitute I – In charge

II. Presentation of the team

Or!g!ns team is composed of five first-year undergraduate students of EPITA. Each member of the team has responsibilities and specific tasks on the project.

A. Marine MAILLOT – Project manager

Or!g!ns team is managed by Marine MAILLOT. She is responsible for scheduling and rescheduling if required the tasks in the group. To follow the planification of the game development, she is in charge of checking the progresses of every member in the team and giving them specific point to focus on if needed.

In the project, Marine MAILLOT is also in charge of the NPC AI; it is her responsibility to define their specificities such as their attacks and movement and to implement these. She works on

the character designs and the animation using her interest for the Greek mythology to feed into the visual aspect and the characteristic of the NPCs of the game. Her knowledge of ancient mythology helped to define the levels and the characters' specificities of the game.

As a substitute, Marine MAILLOT support the website and will help on the SFX and VFX (Sounds Effects and Visual Effects).

Short presentation:

"I have discovered computer science in high school; before then, it was an absolute unknown field for me. I barely thought to study computer science until last year. I joined EPITA class of 2030 after a year of biology that really opened my eyes on computer science and made me realize all the options I have studying computer science. Even if I am not familiar with video games, the project Ascendant Trials is a way to answer an old interest in how these games are made. I wanted to combine one of my personal interest, ancient cultures (Greek, Egyptians, etc) with Or!g!ns' project. I am really impatient to see the final version of our game."

B. Antonin BELIARD

The project Ascendant Trials is based on coordination-heavy puzzles requiring a responsive multiplayer base. Antonin BELIARD is in charge of the networking. He handles infrastructure to keep the project accessible and testable. He also handles the website deployment, its domain, the WebSocket lobby server to manage room and the admin dashboard. He is in charge of the loot system.

As a substitute, Antonin BELIARD helped on the basic logic and will help on the texturing in the next month.

Short presentation:

"My name is Antonin Beliard, and I'm a first-year EPITA student. I've been building tech projects for years, including robotics, small web tools, and mobile apps; so, I'm comfortable turning requirements into something that actually runs."

C. Noah CERIBAC

Noah CERIBAC is responsible of the core game logic (player controller, level loader, block creation, save system, …). He is also in charge for maintaining the codebase: whenever new code is added, he ensures that it does not impact the existing foundation or other systems, in

order to keep the codebase clean, well-organized, and free of bugs or technical limitations. In addition, he is in charge of creating the game's textures and implementing the tower system.

Noah CERIBAC helped as a substitute on the loot system. He will help on the animation.

Short presentation:

"I have always been passionate about computer science and new technologies, which is why I chose to study at EPITA.

The year-long academic project offered by the school allows me to explore new fields and to work under real professional conditions, within a domain I particularly enjoy.

Despite my previous experience in video game development, notably using Lua, I continue to learn every day through this Python-based project developed with the Ursina engine."

D. Antoine TONNOT

Antoine TONNOT takes care of the level design and the sound and visual effects used throughout the game. Since Ascendant Trials is a platformer and puzzle game, the level design must be done carefully to maintain balance while also letting the player trying different solutions and Antoine TONNOT is in charge of maintaining this balance.

As substitute, Antoine TONNOT helps on the NPC AI, the loot system and the tower/dungeon system.

Short presentation:

"I'm Antoine TONNOT and I am a first-year student in EPITA. I love coding and coop game. I also regularly play piano. I started coding in Python and then moved to C# a few years ago. I started coding during the 3rd year of high school."

E. Adam GLEMET

Adam GLEMET is in charge of the website and the level mechanics. He will work as a substitute on the level design, the character design and the GUI (Graphical User Interface).

Short presentation:

“I am Adam Glemet, a first-year EPITA student. Since I discovered web development for my ninth-grade internship, I’ve always been interested in computer science because I already had a passion for video games, so I took NSI specialty in high school where I could develop alone small one-month project in Python, a website group project in HTML/CSS.”

Ascendant Trials

I. Origins and inspirations

Ascendant Trials is a cooperative puzzle-action game built around short, room-based challenges in a vertically stacked dungeon. Our team has decided to create a game partly inspired by another title, Pico Park. However, this project will not be a simple copy but a new interpretation of that style. The player will be trapped inside a tower made up of five themed sections, each inspired by the Greek mythology (The Underworld, The Abysses, The Earth, The Sky, and The Space). Although most game genres and gameplay styles have already been explored, the objective of this project is to position itself within the puzzle gaming category while staying original by combining it with rogue-lite elements.

II. Objectives

A. Project objectives

We wanted to create a game that is easy to access for the players of all ages, fun and engaging. The aim of the game is not to be too complex while staying challenging for the players.

By the end of the year, we would like to produce a playable and functional game, not a prototype or a concept, but a fully functional product.

We aim to provide an experience that is:

- Fun (with an engaging gameplay mechanics that keep players invested);
- Accessible (a playable solo or cooperative game with difficulty scaling to suit different skill levels and style of gaming);
- Polished (a game that feels complete and professional).

B. Team objectives

Equally important to the final product are the skills and knowledge we will acquire throughout the development process. This project is fundamentally an educational exercise in:

- Team Collaboration (working efficiently within a structured team environment, coordinating between programmers, designers…);
- Project Management (planning and adapting to everyone's schedule, tracking progress and adapting to challenge over a defined timeline);
- Software Architecture (turning abstract ideas and design documents into a finished, maintainable codebase);
- Game Development Methodology (understanding the entire process from concept to release).

Project advancement

Each member will present their work on the project.

I. Marine MAILLOT's progress

A. NPC AI

Ascendant Trial is a puzzle-action game with room-based challenges in a dungeon. Every five rooms, the player has to face a boss. These bosses are considered as NPC (Non-Player Character) AI.

For now, the NPC AI actions are not coded. However, they are defined.

The player will have to interact with two types of NPC: bosses and character helping them, this second type of NPCs has simplest limited possible interactions.

1. Level bosses' specific features

The first level is based on the Underworld (from the Greek mythology). The bosses are based on Cerberus, the three heads dog. Cerberus will be able to run in the player's direction and to bite or to attack the player with its three head to be in accordance with the myths where this dog is known for attacking the humans coming in the Underworld. What most people tends to forget is that in reality, Cerberus is simply a dog. Following this idea, this monster will try to "play" with the player using his paw. It will inflict small damage to the player. In the mythology, Orpheus, a human, use his Lyre on Cerberus leading him to fall asleep; similarly, the player can find a hidden Lyre in the Underworld and use it to put Cerberus to sleep.

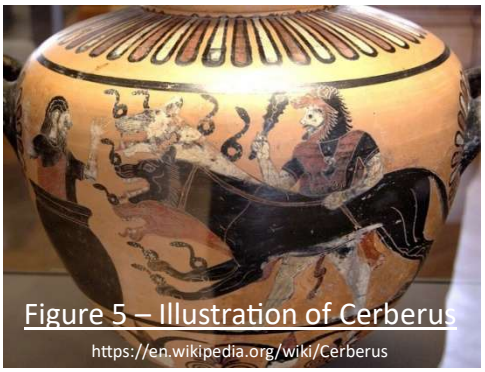


Figure 5 – Illustration of Cerberus
https://en.wikipedia.org/wiki/Cerberus

To sum up, this NPC will be able to:

- Attack or bite with his three heads separately and simultaneously;
- "Play" with the player, hitting them with his paw, it will inflict less damage;
- If the player found the Lyre in the Underworld, the dog can sleep player the Lyre.
- Run to the player's direction.

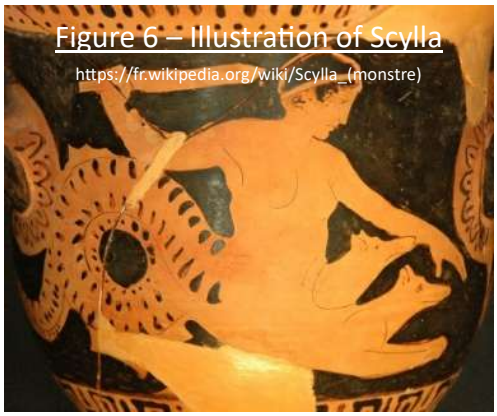


Figure 6 – Illustration of Scylla
https://fr.wikipedia.org/wiki/Scylla_(monstre)

The second level's boss, the Abysses, will be based on the two monsters Charybdis and Scylla. These two threats will mostly have complementary actions.

Scylla is a monster with multiple dog heads and a half human body. These dog heads were known for attacking and killing sailors in Greek mythology, as well as our character. With its hands, Scylla can catch players, they can avoid being killed by destroying its hands. Also known for breaking boats and having a strong voice, Scylla can destroy the player's boat with its voice or simply damage it, depending on the distance between the player and Scylla.

To sum up, this sea monster will be able to:

- "Eat" players;
- Catch them with her;
- Weaken or even break the player boat (as the player will be on a boat facing Scylla and Charybdis) with its voice depending on the space between the monster and the player.

Figure 7 – Illustration of Charybdis
https://histoiresfantastiques.com/charybde/

Charybdis will be positioned in the opposite corner of Scylla. Accorded to its myth, Charybdis swallow water and then spit out a black dark water, this leads us to decide that this sea monster will generate currents by "swallowing" and "spitting out" the water. Similarly, Charybdis was known to destroy boats, for that specificity, in our game, it will also have this ability. Finally, it can eat players by trapping them in the current like in the mythology.

To sum up, it will be able to:

- "Eat" players, as Scylla, by trapping them in the current of the water;
- Generate current by "swallowing" and "spitting out" the water;
- Destroy boats.

Figure 8 – Illustration of Scylla and Charybdis
https://rallye-lecture.fr/charybde-et-scylla-niveau-3/

Figure 9 – Illustration of the Minotaur
https://en.wikipedia.org/wiki/Minotaur

The third level is called Earth. It is inspired by the famous myth of the Minotaur and the Labyrinth. The boss of this level is therefore the Minotaur. Based on the taurus aspect, this monster can charge the player which can harm the player. Even if it has no real myth origins and is from the collective imagination, the Minotaur will have an axe that can be used to hit the player and destroys the player's weapon.

To sum up, this monster is able to:

- Charge the player;
- Hit the player or destroy its weapon with an axe;

The fourth boss is based on another mythologic monster: the Stymphalian bird. This level includes multiple waves of Stymphalian birds. These birds have limited actions, until they are killed, they would attack the player and its wings with their beaks in bronze repeatedly. They are described having metallic feathers, in our games, they will be used as a weapon: the Stymphalian birds can throw it to the player.

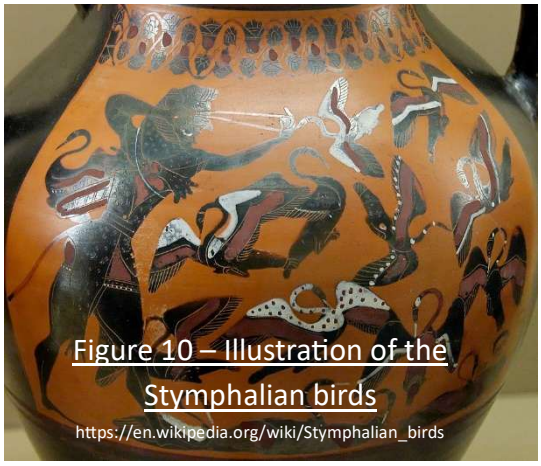


Figure 10 – Illustration of the Stymphalian birds
https://en.wikipedia.org/wiki/Stymphalian_birds

To sum up, these birds will be able to:

- Attack repeatedly the player;
- To throw feathers.

The last boss of the tower is a titan: Astraeus. His name is derived from the Greek word “star”. This boss mythology background is not well-known and for this reason, we decided to define his abilities based on his family. As the father of the three divinities of the wind, Astraeus is able to create wind that can destabilize the player and be used as an attack. He produced the stars according to the Greek mythology, for this reason Astraeus can throw stars to the player; these stars are like fireball and explode sometimes. Unlike the shooting star he can create, the stars are not superfast which make it easy to avoid. The shooting stars are superfast and do not explode. To sum up, Astraeus will be able to:

- Create wind more or less strong;
- Throw stars that can explode;
- Throw shooting stars.

2. Other NPCs

The player can interact with other NPCs, not only bosses.[1]

In the Underworld, the player will have to interact with Charon (or Kharon). This NPC will have to ask for a payment to help the player crossing the Styx; if the player found the Drachmas in the 3 other levels, Charon will only take it. In the other cases, he will give advice to the player to help them finding the Drachmas.

The Abysses level included interactions with the Sirens, mythical creatures that will be a threat for players. They will attack players and lead to death most of the time. These creatures will attack the player with:

- Songs that will slow the player;
- Hits that will inflict damage;
- Tridents that will be throw to the player.

It will be almost impossible to win over the Sirens; they serve as guards to the coast access to force the player to search another exit underwater.

In the Earth level, the player will have to interact with multiple NPCs: Daedalus, Icarus and Ariadne. Ariadne will give the player an enigma to solve. Solving it correctly will allow the player to get Ariadne’s thread. The player will have the choice to interact with this NPC, but trying to find the Labyrinth ends will be hardest without this thread to mark the player path. Icarus and Daedalus will hide in the Labyrinth. Enigmas will guide the player to them. Daedalus will give wings to the player (for the next level) and Icarus will give advice and warnings, linking its myth to the game.

[1] Some of these NPCs have only a limited possibilities of actions and are not truly NPC AI but we decided that they will be consider as NPC and NPCs will all be treated by the NPC AI team.

For now, we do not plan to put NPCs in the Sky level and the Space level is still in progress, the NPCs of the Space level will be decided later.

B. Design

1. Color palette

We decided to create a color palette to have an idea of the color identity we wanted to have. After discussing it, we choose to use pastel colors.

The color palette was drawn twice: once on a white background, one on a black background. This duplication of the palette is to be able to see the effect of the background depending on the color as our game use dark and light background depending on the section.

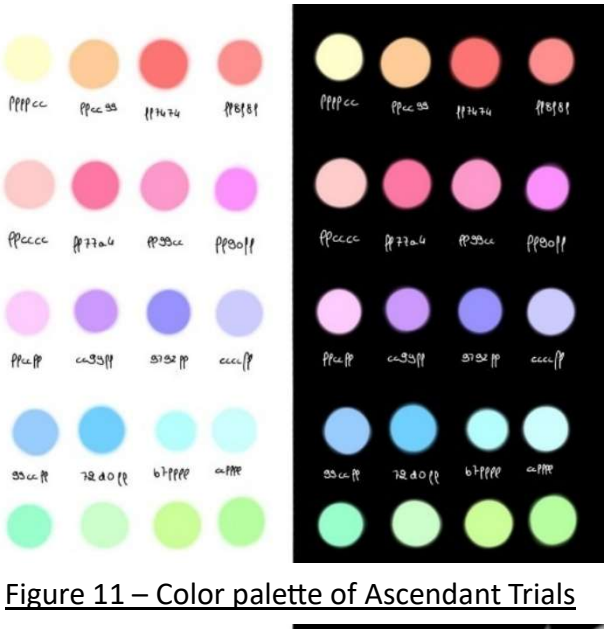


Figure 11 – Color palette of Ascendant Trials

2. Character design

Each level will have its unique design based on the environment.

The general design was defined. We have drawn it as it is not going to be change in the future.

The inspiration was a coffee cup and it evolved to a minimalistic little character.

The character will have specific design depending on the level:

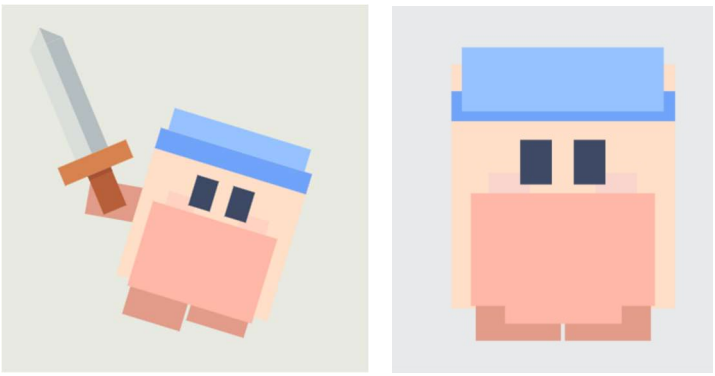


Figure 12 – General character design

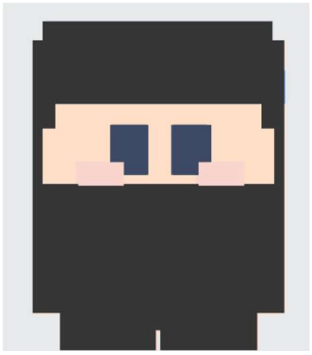


Figure 13 – Beginning of the Underworld character design

- For the Underworld, it was decided to use Charon as an inspiration (using the image of the collective imagination); the player will have the possibility to change the base outfit of the character to a dark outfit.
- For the Abysses, the character will wear a wetsuit.
- For the Earth, the player will have the possibility to change the base outfit to a traditional Greek outfit.
- For the Sky, the character will wear wings and can still wear a traditional Greek outfit.
- For the Space, we are still debating the design of the character.

3. Animation

We did not start to work on the animation part; we planned to work on it for the second defense.

4. Level design

Even if level design is not one of Marine's responsibilities, she participated in the definition of eighteen levels; these are defined in different levels of details[2].

First of all, before creating each level, the global structure of the section had to be defined. Then the structure of each level could be defined.

For the first section, the Underworld, the level 2 and 3 are to define.

The first level is an introduction for the player to the game's universe. It aims to be simplistic and to illustrate the puzzle-solving aspect of the game: the player is in an empty strange place with a door and an enigma to solve.

[2] Annex I – Level design details

Based on the Underworld characters, an NPC was added in the level 4. This NPC, based on Charon from the Greek mythology, is a way to introduce the concept of die and retry to the player. Dying gives the player the possibility to find secret items: Orpheus' Lyre and Drachmas (ancient Greek currency).

Finally, the level 5, the boss level, includes an empty space with Cerberus.

The Underworld section is based on a simplistic design to illustrate the description of some part of the Underworld and to let the player discover the concept of the game with a simple-to-understand environment.

The second section, the Abysses, is more complex.

An NPC is introduced in the level 6, Helios, to give some light to the player. It is a bonus item to help the player in the next level. The levels 7 to 9 are based on a system of cave with or without oxygens that can be opened solving an enigma. Gradually, through these three levels, the player discovers how to manage the oxygen and avoid some threats such as hydrothermal vents.

The level 9 is also the level with the surface access. The sirens are used as a threat to force the player to choose the "difficult" path, the one with multiple enigmas, to join the boat that will save the player.

The level 10 is meant to be more difficult than the precedent boss: two bosses are placed in the opposite corner of the map, between the player and the exit.

The Earth section was the more interesting to design. The player will meet several NPCs in this section: Ariadne, Daedalus and Icarus. These three characters are helping the player to survive the Labyrinth and then the Sky section. Unlike the other section, the boss is placed in the level 13.

The player will enter the Labyrinth at the end of the level 11 and will solve enigmas to find the Minotaur in the level 12, in the Labyrinth. After killing the Minotaur in the level 13, the player will go back into the Labyrinth to find the way out, this is level 14. Finally, level 15 is the moment the player goes out of the Labyrinth.

This section was hard to create combining the "classis" logic of level progression (meeting the boss in the last level of the section after several levels in the section and switching to the next section after killing the boss) and the concept of the Labyrinth (a complex place composed of multiple roads, crossroads and dead ends). After a few discussions with people external to the project about the difficulties of the design of this section, it became evident that changing the order of the level was more adapted and will give us more liberty on this section.

Every level of the Sky section is based on the same system: an immaterial grid with clouds, storms, a wind current system and a sunray influence system. For this reason, the structure of each of the levels were not draw with precisions.

The storms will be a threat for the player. The cloud can be used as shield to avoid being pushed on the storm by the wind current system.

The level 20 is the boss level where the player will face the Stymphalian birds. These mythological creatures will attack the player unceasingly.

The Space section is still to define.

In every section, the levels will use a die and retry system. The player will understand through the levels that dying can be an occasion to obtain secret items. These secret items can help the player to pass easily the boss levels.

Every level aims to be intuitive or to be understandable with fails and to encourage the player to thing carefully in every environment to find the threat and the secret items.

C. Website

As the substitute on the website, a second version of the website was created.

We first generated a draft of the website with ChatGPT which code is not used in our project. This first version was generated for a preparation oral. We wanted the website to be more personal. For this reason, we decided to coded it ourself, from zero, instead of using the AI-generated website.

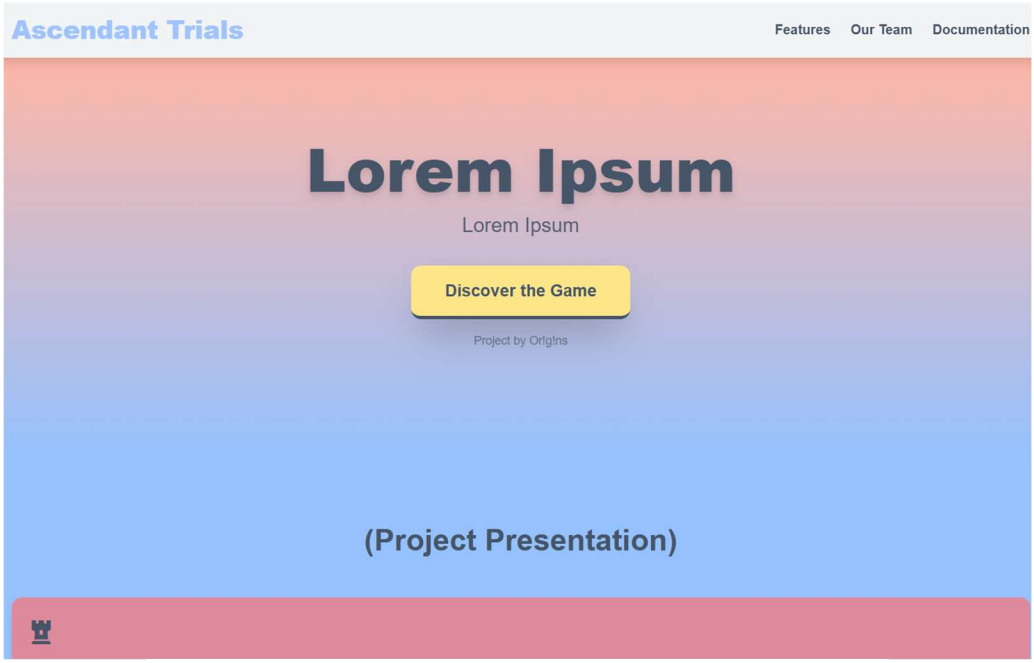


Figure 14 – AI generated website (alpha version)

After this alpha version created by Adam, a second version was created. For this version, it was decided to use an HTML file and a CSS file for the style of the website.

This second version is still in progress. The color association needs to be changed and we are working on it.

Moreover, through this website, we want to convey our visual identity. This visual identity is still blurry; we do not have defined a clear logo that can mark players' mind. For this reason, the global visual aspect of the website is still in reflection.

We separate the website in different parts: the presentation of the game and its lore, the presentation of the team and the links to the resources we used and the different documents required.

Creating the website was difficult, mostly because CSS and HTML are new to the website team. For example, it took me time to find the correct parameters to separate "The concept" and "The lore" while keeping them in the same line inside the part of the presentation.

To use HTML and CSS and find the parameters explanation, MDN Web Doc[3] is the main reference.

[3] https://developer.mozilla.org/en-US/docs/Web/HTML

As you can see, the website is not fully done and its constructions is still in progress. However, the website can be used.

Figure 15 – Second version of the website (coded by our team)

II. Antonin BELIARD's progresses

A. Context and scope

Ascendant Trials is based on puzzle-solving. Puzzles rely on timing and coordination (switches, moving platforms, shared objects), the project requires a responsive multiplayer base.

One of my responsibilities is Networking. I handle the infrastructure that makes the project accessible and testable: website hosting (deployment and domain), the multiplayer lobby server (WebSockets), and tooling to observe rooms during tests and defenses.

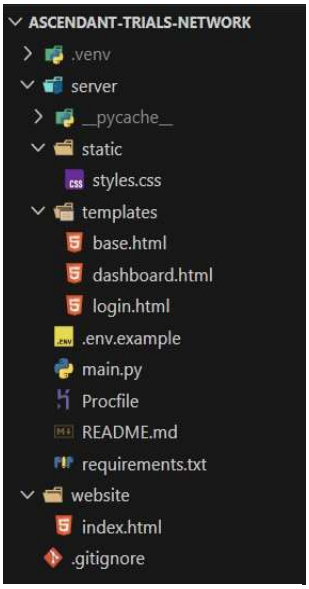


Figure 16 – Architecture diagram: Website (edge), Lobby Server (WebSocket) and Admin Dashboard

B. Architecture overview

To keep responsibilities clear, the game code and the networking stack are split into separate repositories. The main game repository contains the Ursina project, while the network repository contains the lobby server and deployment configuration. This avoids mixing

gameplay iterations with hosting concerns and allows the multiplayer stack to evolve independently.

At the current milestone, multiplayer is implemented as a lightweight room relay: a single server maintains active rooms in memory. Clients connect to a room over a WebSocket, exchange small JSON messages, and leave when the session ends. In parallel, an admin dashboard provides live visibility into server health and room activity.

1. Runtime data flow

A player either creates a room (host) or joins an existing room (guest) by entering a short room code in the game UI. The client then opens a WebSocket connection and sends two identifiers as query parameters: room and player.

On connect, the server registers the player, sends a confirmation message containing the current player list, and broadcasts join/leave events to the rest of the room. During the game, the server relays messages to all players in the same room.

Figure 17 – Sequence diagram: connect -> joined/player_joined -> message relay -> disconnect -> player_left

C. Hosting, domains, and deployment

The deployment is split into two tiers: edge hosting for the static website, and a regional server for real-time WebSockets. This keeps the website fast and simple to publish, while ensuring the multiplayer endpoint is always reachable for demos.

1. Website delivery (Cloudflare Pages / Workers)

The website is hosted as a static application on an edge platform (Cloudflare Pages / Workers). Although I do not develop the website content, I own the operational side: connecting the repository, configuring builds, linking the custom domain, and ensuring continuous deployment from the main branch.

This setup provides HTTPS by default and produces a preview deployment for each update, reducing the risk of last-minute manual uploads.

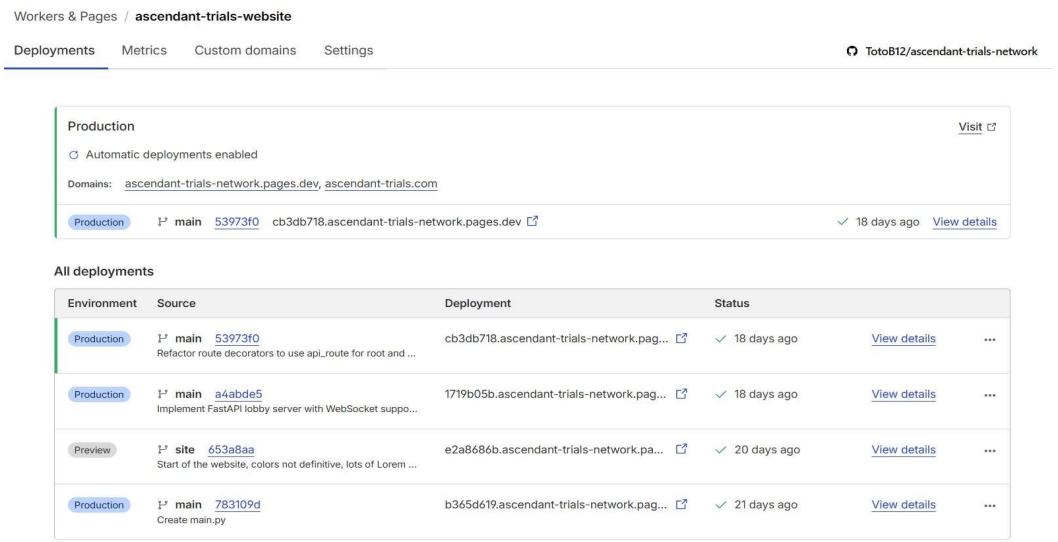


Figure 18 – Cloudflare pages project (custom domain and latest deployments)

2. Lobby server hosting (Koyeb, Paris region)

The lobby server runs as a small web service hosted in the Paris region. A nearby region reduces latency during local tests. The service exposes a public URL that forwards traffic to the internal port used by the FastAPI application (default 8000).

The instance is intentionally small (Nano tier) because early traffic is limited: the server stores inmemory room state and relays small messages. As development progresses, we can scale resources or add persistence if needed.

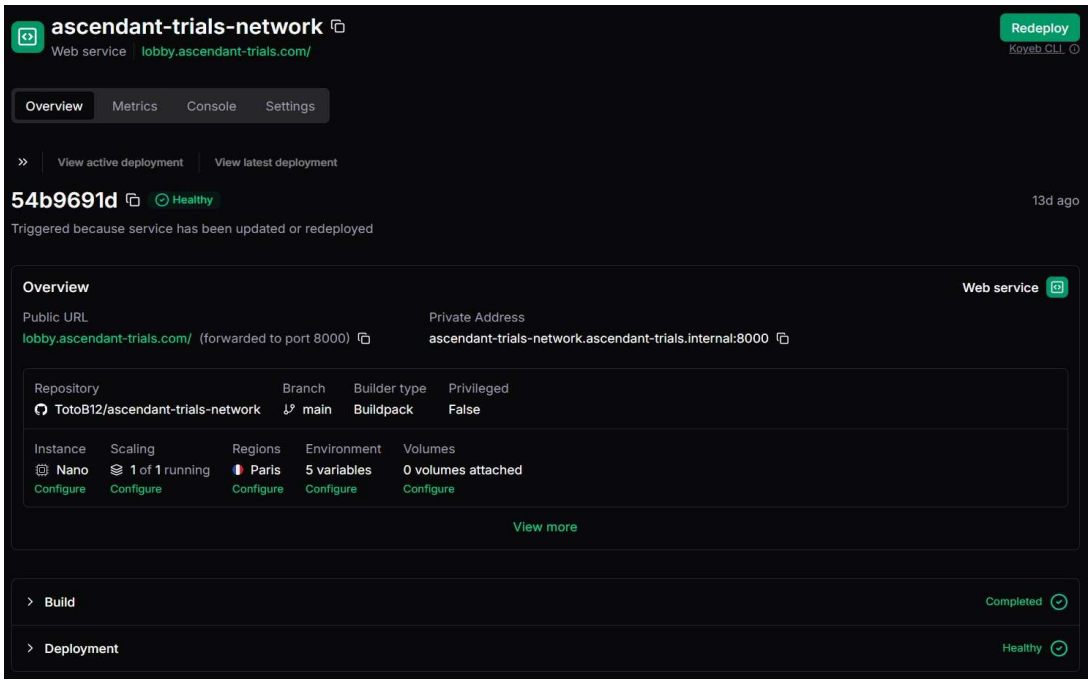


Figure 19 – Koyeb service overview (publir IRL, instance size, region, deployment status)

3. Configuration and secrets

Sensitive values are injected through environment variables: admin dashboard credentials and a session signing secret. This keeps secrets out of the repository while making deployments reproducible across local and production environments.

D. Lobby server: implementation details

The server is written in Python using FastAPI and Uvicorn. It exposes simple HTTP endpoints for monitoring and a WebSocket endpoint for real-time rooms.

1. Health and status endpoints

Two endpoints are used during development and demos: /health (returns status=ok) and /api/status (returns a JSON snapshot of active rooms, player count, and system metrics such as CPU and memory usage).


```
{
 "status": "ok",
 "stats": {
  "uptime_seconds": 1143435.28,
  "cpu_percent": 0,
  "memory": {
   "percent": 30,
   "used_mb": 69.29,
   "total_mb": 260.84
  },
  "process_count": 1,
  "started_at": "2025-12-20T14:18:46.258522"
 },
 "rooms": {
  "demo-room-01": {
   "id": "demo-room-01",
   "created_at": "2026-01-02T19:55:18.404183",
   "players": [
    "playerA",
    "playerB"
   ]
  }
 },
 "room_count": 1,
 "player_count": 2
}
```


Figure 20 – /api/status JSON response during a multiplayer test

2. WebSocket lobby endpoint

Clients connect to /ws/lobby with room and player parameters. Rooms are created on demand when the first player joins. The lobby is modeled with three structures: Player (id, websocket, joined_at), Room (id, created_at, players), and Lobby (rooms dictionary). An asyncio lock protects joins/leaves to avoid race conditions.

Messages are received as text frames. If the payload is valid JSON, it is wrapped in a common envelope (type, room, player, payload, timestamp) and broadcast to all players in the room. NonJSON messages are treated as plain text.

Disconnect handling is explicit: when a player leaves or when sending fails, the player is removed from the room. When the last player leaves, the room is deleted to free memory and prevent stale state.

```
INFO:     ('88.188.30.129', 0) - "WebSocket /ws/lobby?room=demo-room-
01&player=playerA" [accepted]
2026-01-02 19:55:18,404 [INFO] ea-trials-lobby: Created room demo-room-01 2026-01-02 19:55:18,404 [INFO]
ascendant-trials-lobby: Player playerA joined room demo-room-01 (players=1)
INFO:     ('88.188.30.129', 0) - "WebSocket /ws/lobby?room=demo-room-
01&player=playerB" [accepted]
2026-01-02 19:55:50,485 [INFO] ascendant-trials-lobby: Player playerB joined room demo-room-01 (players=2)
2026-01-02 20:01:00,598 [INFO] ascendant-trials-lobby: Player playerA left room demo-room-01 (players=1)
2026-01-02 20:01:01,199 [INFO] ascendant-trials-lobby: Player playerB left room demo-room-01 (players=0)
2026-01-02 20:01:01,199 [INFO] ascendant-trials-lobby: Room demo-room-01 deleted (empty)
```

Figure 21 – Server logs showing room creation, join/leave and room deletion when empty

E. Admin dashboard and observability

To debug multiplayer efficiently and to demonstrate that the system is under control, the lobby server includes a small admin dashboard. It is protected by a login page backed by environment variable credentials and a signed session cookie.

After login, the dashboard displays uptime, room and player counts, CPU and memory usage, and a live table of active rooms and connected players. A simple JavaScript polling loop refreshes data every 2.5 seconds by calling /api/status.

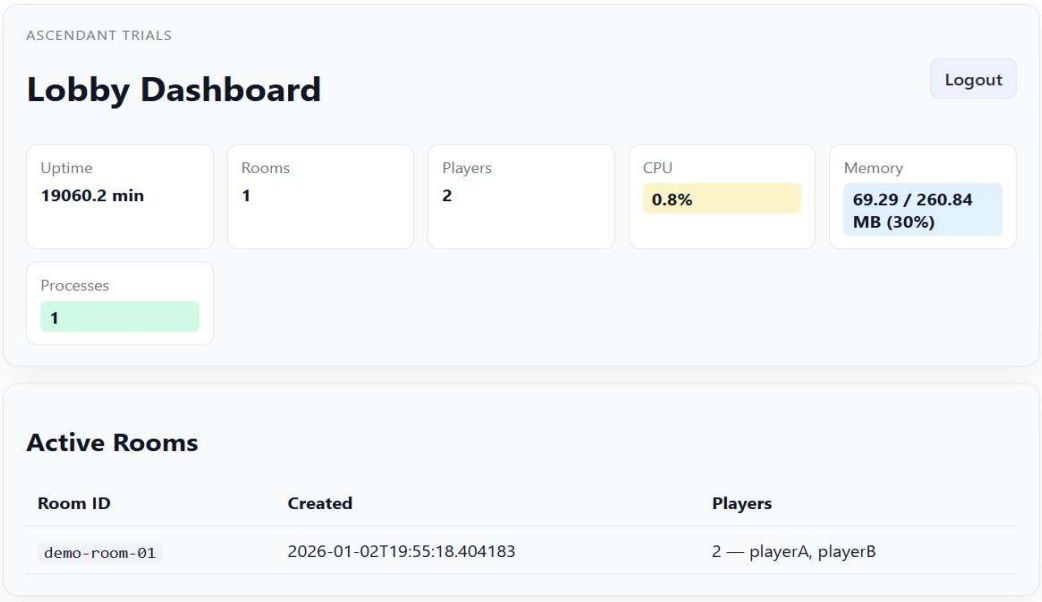


Figure 22 – Admin dashboard during a live test (metrics and rooms table)

F. Testing and validation

A multiplayer feature is only convincing if it can be validated quickly. My workflow focuses on short feedback loops:

- Local smoke tests: run the server locally and connect multiple clients to the same room to validate joins, broadcasts, and disconnect cleanup.
- Platform checks: redeploy on Koyeb after changes and verify /health and /api/status from an external network (not only localhost).
- Observability checks: open the dashboard during tests to confirm room counts and player lists match what clients display.

This is also where the system metrics become useful: during tests, I watch CPU and memory usage to ensure the Nano instance remains stable. Even if the lobby workload is small, this habit prevents surprises when we start sending more frequent player_state updates.

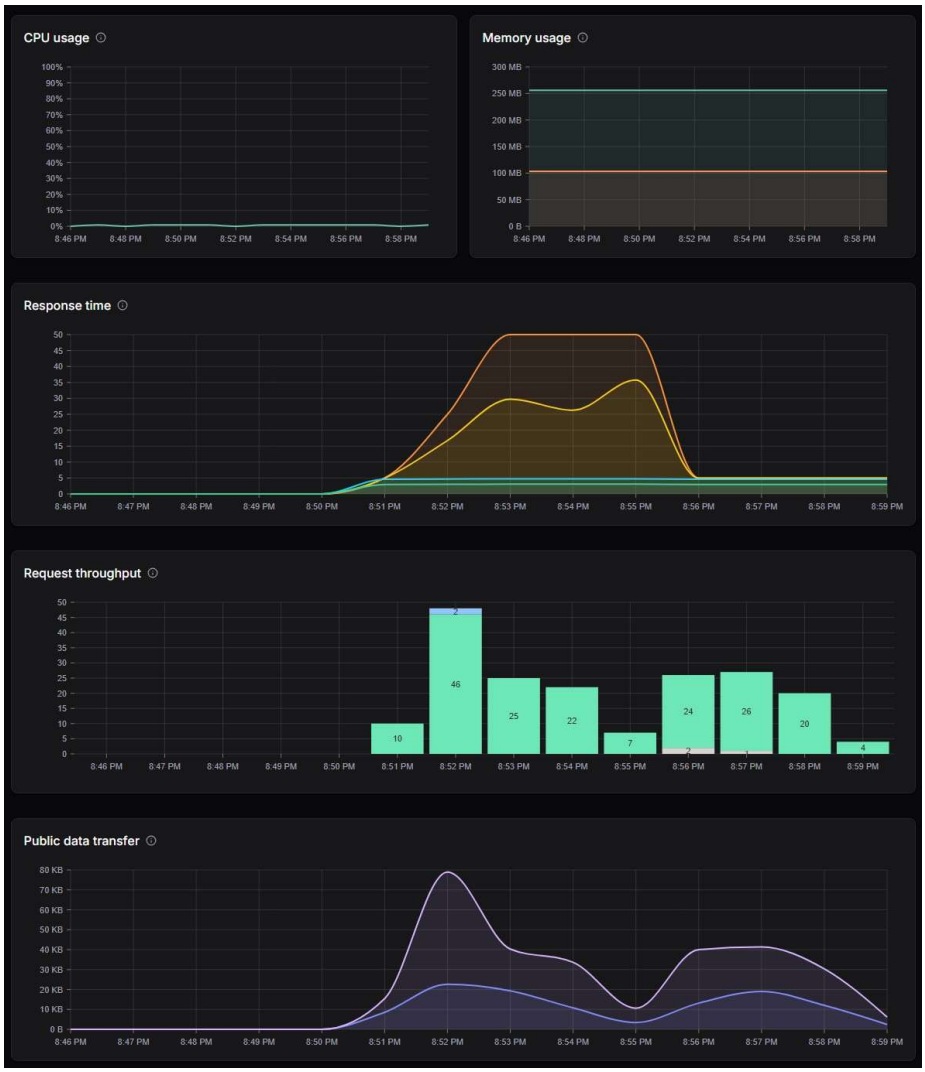


Figure 23 – Dashboard metrics while several clients are connected (show resources usage)

G. Roadmap (work in progress)

The current server is an MVP for room-based cooperative play. The next multiplayer milestones are:

- Integrate the WebSocket client into the Ursina game and define a minimal message set (player_state and interaction_event).
- Add room lifecycle features: max players, version checks, and faster disconnect detection (heartbeat).
- Implement simple matchmaking: create/join flows with short codes, and optional automatic room assignment.
- Improve robustness: input validation, payload size limits, and rate limiting to prevent spam or accidental loops.
- Document scaling limits (single-instance in-memory rooms) and, if needed later, evaluate shared state (e.g., Redis).

H. Conclusion

My contribution provides the project with an operational multiplayer foundation: a deployed WebSocket lobby server, health and status endpoints, a protected admin dashboard, and a stable hosting setup with a public domain. This infrastructure allows the team to iterate on cooperative mechanics while keeping deployment and debugging under control.

III. Noah CERIBAC's progresses

A. Global vision

1. Files and folders architecture

To ensure modularity and enable parallel development, the project is organized with clear separation of concerns:

- **scripts/core/**: a folder containing the main scripts files, required to the game functioning
- **scripts/player/**: a folder containing the scripts related to the player
- **scripts/items/**: a folder containing scripts related to items functioning
- **scripts/sections/** :we Zone-specific mechanics (sections_01_XY.py through sections_05_XY.py)
- **assets/** – Resources: levels (JSON), textures, audio

This structure allows multiple developers to work in parallel on different sections without centralizing logic in main.py, making the codebase easier to navigate and maintain.

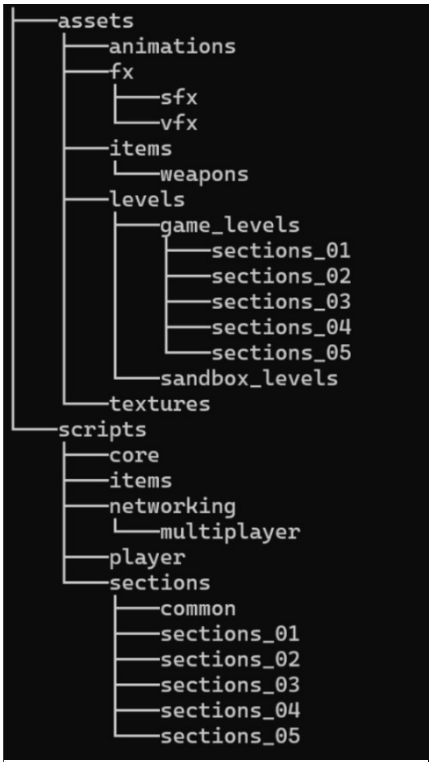


Figure 24 – Files and folders architecture

2. Core design

All gameplay data is externalized into JSON files: level layouts, block properties, behaviors, and weapon definitions. This is a game development choice to keep the game light, and to keep every asset easy to edit and use.

Common systems are centralized in shared modules (scripts/core and scripts/sections/common), reducing duplication and ensuring consistency across the game.

B. Basic logic

1. scripts/core/block_manager.py

Blocks are instantiated and managed centrally, enabling easy creation, deletion, and bulk reset operations.

create_block(position, size, tint, collision, name): creates a 2D platform block within the level.

- Input: position tuple, size tuple, tint color, collision boolean, name string.
- Output: the created entity.
- Purpose: mandatory for level construction; instantiates the visual and physical platforms.

delete_block(block): removes a specific block from the level and cleans up any associated animations.

- Input: the block entity to remove.
- Output: none.
- Purpose: mandatory for dynamic environmental changes or destroying platforms.

delete_all_blocks(): clears the entire level by removing all blocks and canceling active animations.

- Input: none.
- Output: none.
- Purpose: mandatory for cleaning up the scene before loading a new level to prevent memory leaks.

2. scripts/core/effects_manager.py

Manages the audiovisual atmosphere of the game, including background music and visual backdrops.

load_ambience(sound_path, volume, loop, autoplay): loads and plays a looping background sound.

- Input: sound file path, volume settings.
- Output: the audio entity.
- Purpose: ensures continuous audio atmosphere (e.g., dungeon wind) persists across level loads.

load_background(texture_path, position, fit_to_camera): spawns a background quad behind the game blocks.

- Input: texture path, position vector, scaling options.
- Output: the background entity.
- Purpose: provides visual context for the level. It scales the image to fit the orthographic camera view.

3. scripts/core/level_loader.py

Levels are defined entirely in JSON. The loader reads player spawn position, instantiates blocks via the block_manager, and applies behaviors.

load_level(json_path, clear_previous): loads a level from a JSON file, creates blocks, and parses behaviors.

- Input: json_path (str), clear_previous (bool).
- Output: dictionary containing level info (like player_spawn).
- Purpose: the core function for transitioning between game stages. It parses JSON to spawn blocks and assigns behaviors.

_parse_color(color_value): checks and ensures that a given color follows a valid representation format.

- Input: color_value (string hex, list RGB/RGBA).
- Output: Ursina color object.
- Purpose: mandatory safety function; prevents crashes by handling multiple color formats defined in JSON.

4. scripts/core/progress_manager.py

save_progress(current_section, current_level): saves or loads the global progression state to/from a JSON file.

- Input: file path (optional).
- Output: progress dictionary.
- Purpose: ensures player progression is persistent across sessions.

mark_level_completed(sections, lvl): updates the internal state to mark a level as done and calculates the next level index.

- Input: current section and level numbers.
- Output: tuple (next_section, next_level).
- Purpose: handles the logic of advancing through the dungeon (e.g., Level 5 → Section+1).

5. scripts/player/player_controller.py

The player is implemented as a PlayerController wrapper around Ursina's PlatformerController2d prefab. This provides stable collision detection, gravity handling, and jumping mechanics.

__init__(position, tint, jump_height, walk_speed, max_jumps, gravity, health): initializes the player controller with configurable physics and stats.

- Input: position tuple, tint color, jump/speed parameters, gravity, health.
- Output: none.
- Purpose: creates the player entity and sets up global access via PLAYER_INSTANCE.

set_position(position): sets the player's position to specific world coordinates.

- Input: position vector (x, y, z).
- Output: none.
- Purpose: mandatory for spawn mechanics and resetting the player after death.

health_change(change_value): modifies the player's health by the given value, clamped to max_health.

- Input: change_value (int/float).
- Output: none.
- Purpose: manages survival logic. If health reaches 0, the main loop triggers a level reset.

change_gravity(new_gravity): updates the gravity applied to the player and recalculates jump height.

- Input: new_gravity (float).
- Output: none.

- Purpose: allows zone-specific physics changes (e.g., low-gravity sections)

get_health(): returns the current health value.

- Input: none.
- Output: int (current health).
- Purpose: mandatory for the main loop to determine death state.

6. main.py

update(): updates the game state for block animations (like sliding platforms) at each frame.

- Input: none (uses time.dt).
- Output: none.
- Purpose: makes animations FPS-independent by using delta time to interpolate positions.

main(): launches the game, initializes the window, loads the first level, and spawns the player.

- Input: none.
- Output: none.
- Purpose: essential entry point; sets up the Ursina application and starts the engine.

get_sections_path() / get_weapon_path(): helper functions to construct file paths dynamically.

- Input: none (uses global current_sections and current_lvl variables).
- Output: string (file path).
- Purpose: centralizes path logic so the game can easily load "the current level" without hardcoded strings.

7. scripts/sections/*

This path contains all function (=mechanics) per section, with 6 folders, common, section_01 to section_05.

C. Tower/Dungeon system

In this section, there is no code or functions, since the functions related to it (e.g. scripts/core/progress_manager.py) were considered part of the Basic Logic.

More generally, it is interconnected with the other sections because it is part of the theme of the game.

1. Progression Structure

The game is organized into 5 sections (sections_01 to sections_05), each containing multiple levels (sections_0X_lvl_0Y.json). This structure prepares the vertical progression: as the player climbs, floors become more difficult.

2. Dynamic Pathing

Functions like get_sections_path() in main.py dynamically build file paths based on current_sections and current_lvl variables. This simplifies management: the code simply asks for "the current level" without needing hardcoded paths for every stage.

D. Loot system

1. scripts/items/weapon_manager.py

Weapons are defined in JSON files (e.g., assets/items/weapons/sections_0Y.json).

Each weapon consists of 4 fragments, which are unlocked one by one as the player completes levels in a section.

render_fused(): renders all currently unlocked fragments of the weapon.

- Input: none (uses internal unlocked_fragments count).
- Output: none (updates UI entities).
- Purpose: Displays the player's reward progress. It reads the weapon JSON and spawns quad entities for each shape in the unlocked fragments.

unlock_next(): increments the unlocked fragment count and saves it.

- Input: none.
- Output: none.
- Purpose: called upon level completion to reward the player with the next piece of the section's weapon.

E. Texture

Since Texturing is approximately 50% complete, I took charge of creating the textures for the weapons used in the game. As shown below, there are 4 fragments per weapon plus the final fused weapon. Fragments are collected by completing puzzle levels, and the fused weapon obtained by combining them is used to defeat the section boss.

Each weapon is themed according to its section:

- A fire sword for the Underworld
- A trident for the Abysses
- A hammer for the Earth
- A bow for the sky, to shoot flying entities
- A scepter for the space

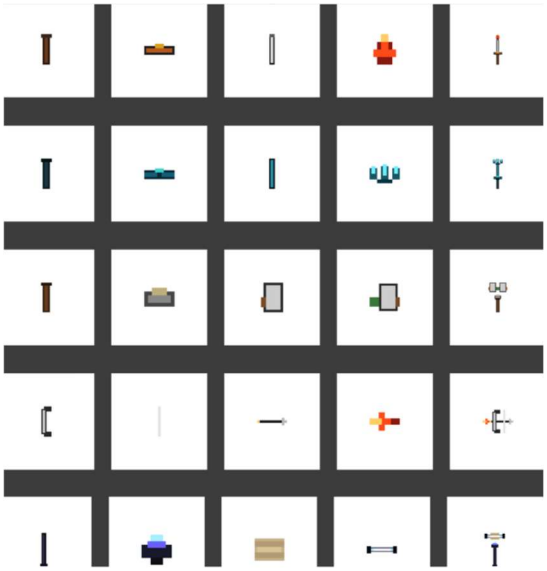


Figure 25 – Weapon designs with separate parts and complete weapons

IV. Antoine TONNOT's progresses

A. Game Mechanics

All the group helped to think about the mechanics according to what they wanted to see in the game. We started thinking about these mechanics in early November, but we only wrote them down during the holidays.

The game is divided into 5 distinct stages each representing a unique environment with its own graphical and musical identity as well as mechanics that differs significantly from one stage to

another. Our goal is to align the gameplay mechanics with historical or mythological events and facts connected to each stage.

Each stage is composed of five different levels: 4 puzzles-focused levels and one boss encounter. This structure has for goal to establish a clear sense of progression. Boss fights serve to test the player's understanding of the mechanics introduced throughout the game.

The first stage, titled Hell, will serve as a tutorial for the game, establishing the game's core mechanics. Rather than relying on explicit instruction, it will introduce mechanics trough environmental constraints and player experimentation.

The primary mechanic introduced will prevent the player to go back in the level. This means that the key used to go left will be disabled during the whole level. The mechanic was inspired by the myth of Orpheus and Eurydice in which Orpheus cannot turn around to look at Eurydice while leading her out of the underworld.

The second stage is named the Abyss. The Abyss is a dark and hostile environment filled with water and dangerous life forms. Everything in this stage poses a threat from aggressive wildlife to unstable geological formations.

One of the most important mechanics of this stage is the ability of the character to swim. The player will be able to move freely in the levels allowing the player to look for different approaches to solve the levels puzzles. The player will be able to move freely in all directions within a two-dimensional place. The floating-based control system allows for nuanced navigation and encourages the player to explore alternatives paths when solving puzzles.

A critical mechanic in this stage is the oxygen management. Each player will need to monitor their oxygen to avoid drowning. There will be several ways to find air to breath. For example, players will have to find air pockets to breath or bubbles coming from the ground to survive.

Environmental hazards complicate the navigation. While active, Hydrothermal vents both push players away and damage them as well, potentially temporarily blocking the puzzle solutions. The hydrothermal vents will turn on and off, following a given pattern.

The third stage is Earth. This stage transitions the player into a maze-based environment. It places a strong emphasis on cooperation in multiple scenarios (for two players games).

One mechanic of this stage is a string linking the two players, preventing them from going too far from each other. This mechanic will reduce their movements possibilities and will force them to watch carefully their environment to avoid being pulled apart. If the string breaks, the level will restart.

A second major mechanic splits the players (if there are two players) into different parts of the maze. Each player will experience distinct versions of the level, with unique layouts and obstacles, which will introduce an asymmetrical gameplay. Action performed by one player will automatically affect the other. This will ensure cooperation and communication between players.

To maintain logical consistency, the player separation occurs during the pervious level, avoiding abrupt or narratively unjustified transition.

The fourth stage is named Sky. It mainly focuses on verticality, environmental forces and large freedom of movement.

In this stage strong winds will reduce the player's movements. These will be generated either artificially by large fans, pushing the players away and slowing them down when they try to come closer or naturally through environmental currents leading to stronger winds which will push the players in one direction no matter what during a certain amount of time. This mechanic will push the player to correctly time his actions.

In certain levels, players will be able to fly, allowing the player to solve puzzle in new ways. However, this ability introduces significant risks. While flying, crossing a ray of sunlight immediately resets the level. To avoid this, the player must use environmental cover, such as floating platform, to shield themselves from exposure. This mechanic is inspired by the myth of Icarus, who used wings to escape imprisonment but flew to close to the sun, leading to his downfall.

Additional hazards include frequent lightning strikes in some areas. When struck the player temporarily shrink in size allowing access to narrow passages that would be otherwise inaccessible. This mechanic introduces trade-offs: while reduced enables the exploration of restricted areas, remaining in these areas after the effect expire results in a level reset. Furthermore, if struck by lightning again while already shrunk also triggers failure.

Finaly, Sky reintroduces a resource-management system like oxygen mechanic in the Abyss. At high altitude, the player requires oxygen to survive, limiting how long they can remain airborne. This forces the player to seek safe zones to recover during his exploration.

The fifth stage is named Space. It is a place where gravity is very variable and an unpredictable force. In some locations gravity allows the player to jump higher or to almost float while in other places extreme gravity severely restrict movement. These shifts occur cyclically, creating dangerous situations such as sudden fall from great heights.

In certain places, gravity can be manipulated directly through switches. Activating these switches will inverse gravity, transforming ceilings to floors and vice-versa allowing to change it by pushing a simple switch. This mechanic unlocks new puzzle-solving opportunities while introducing new hazards.

B. Level Design

During the holidays, we started to create a prototype containing some mechanics to test them and see if they are viable.

C. SFX/VFX

For the SFX, we started to write down themes for the music in each zone. For example: stressful music for the Abyss, joyful for Sky, …

V. Adam GLEMET's progresses

A. Website:

First, I generated a structure with AI (Chat GPT) to have an idea of what to do for the website then modified it to match what we wanted to do more. From the code given by the AI, I worked on a few features to improve the UX/UI side of the website:

I implemented our own palette to the Tailwind CSS configuration with a wide variety of pastel colous according to the style we want for our game, moving beyond the initial limited set given by the AI. This allowed for a more vibrant and diverse look while maintaining a soft, professional aesthetic.

Then I tried different color sets, modified the icons because the game used as an example by the AI was not our kind of game so there were icons not related at all.

After that I tried different types of backgrounds of the several key sections to create better visual separation. This includes using specific shades like soft pink for the team area, a light periwinkle for the documentation section, and a gentle violet for the footer for example. Within the Team Presentation part, I assigned a random pastel color to each member of the group.

In the end I sent what I have done to Marine, substitute on the website, to have another point of view of the whole and to do the CSS.

Progression and global organization

In the last two months, we have work on different tasks. We redefined our schedule for the next months depending on our recent progresses.

A. Progression

As we planned, the basic logic is complete.

We planned to finish 25% of the level mechanics. We defined the different mechanics of the sections and began to code it. We reach our goal for this first deadline.

The NPC AI is less advanced than we planned: we defined every NPCs' actions but did not start to code them.

We wanted to achieve 25% of the loot system, we reached our goal and created the weapons manager.

The tower/dungeon system was planned to be half done. It is still in progress but we almost reached our goal.

The networking was supposed to be almost finished. This part was not too long and we met our expectations.

The website was planned to be almost finished. The HTML part is done and the CSS part is partly done. We met our goal in this part.

The level design was supposed to be done at 10%. We defined almost every level aspect and structure and exceeded our goal.

The character design is less advanced as we planned to due to difficulties with the use with the tool we chose for it.

The texturing was not started. We did not reach our goal.

The GUI and SFX/VFX are planned for the next deadline, we did not start them as planned.

B. Future schedule

For the next deadline, the second defense, we have defined our goals.

We want to achieve half of:

- the level mechanics which means coding more than 50% of the mechanics;
- the NPC AI which means coding all the NPC AI boss;
- the level design which means implementing the background and the structure of at least ten levels;
- the GUI;
- The SFX/VFX which means finishing at least the sound effect or the visual effect.

We progressed more on the loot system than expected and we hope to almost finish it for the next defense.

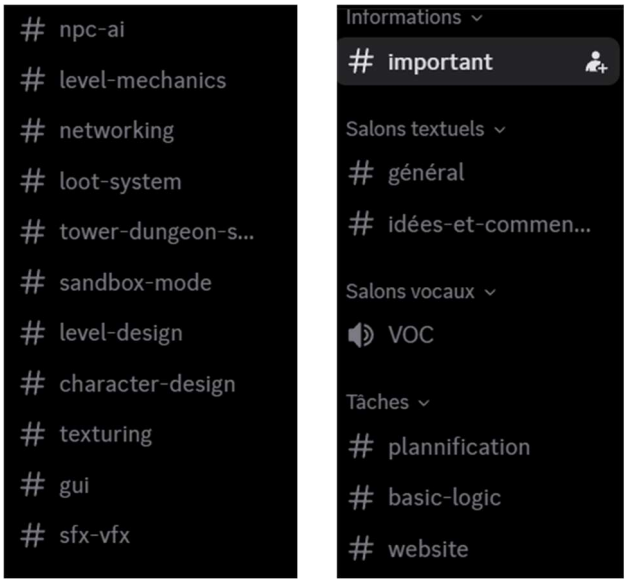
We expect to finish the tower/dungeon system for the second defense as well as the website, the character design, and the texturing. Considering the delay on the texturing, we may not be able to achieve that goal.

C. General impression and organization

When we started this project, we did not realize the organization required but we quickly understood that it would help us a lot.

The team is really confident in achieving the scheduled tasks for the next defense.

In terms of organization, we defined separate canals or communication of our work.



We use Discord as our main mean of communication. We created specific channels for every tasks. We also use Discord for voice call to work on the project together.

Figure 26 – Or!g!ns server's channels

We also use Penpot for the design to work together; we can share our design to every member of the team and work together on the same design.

We chose to share the different document (Functional Specification Document, Technical Specification Document, presentations, …) on Google Docs and online PowerPoint.

Finally, every line of code of the project is on Github and accessible to the whole team.

Conclusion

Regarding the individual parts, our project is not progressing following the exact schedule we defined a few months ago. However, even though we did not work on the exact tasks we planned to; on a global scale, we are developing our game counterbalancing less advanced task with other tasks more advanced.

Our team managed to get organized with different sites or applications (Discord, Google Doc, Penpot) to work more efficiently and share our work with everyone in the team.

Annexes

I. Level designs