Second Project Report

Version: 1

Date: 18/03/2026

Members : Damien Hasenfratz, Ruben Desbonnet-Ramdenee, Lilou Tardif

Game : Control+Sea

Summary

Introduction

- Group organisation
- Reminder of our game

Evolution

- Core
- Network
- AI
- Graphics
- Website

In the future

- Our priorities
- what we are going to bypass

Conclusion

Introduction

Unfortunately the crew lost one of his members changing our team from four to three members. It changed our organisation and priorities for this second soutenance and the game in general especially as we were notified quite late.

Nevertheless, this event helped us to adapt ourselves quickly depending on the situation which is a valuable skill in team management especially.

Our previous group organisation

Name	Role
Damien Hasenfratz	Enemy AI, Networking, Project
Sophie Mendez	Level design, Sound effects, Storyline
Ruben Desbonnet-Ramdenee	Core, Gameplay mechanics
Lilou Tardif	Assets, UI

New group organisation

Name	Role
Damien Hasenfratz	Enemy AI, Networking, Project
Ruben Desbonnet-Ramdenee	Core, Gameplay mechanics, Sound effects
Lilou Tardif	Assets, Level design, Storyline, UI

Game reminder

Our game is a pirate theme game. It is a collaborative multiplayer game where we fight together against a common enemy powered by AI. When we win our battle we are rewarded with a treasure chest containing either money or resources needed for the crew and the boat(such as wood, rhum or cotton). In

the other hand, if we lose then our boat will be damaged and the crew will take longer to recover. Our goal is to improve our boats and weapons as we sail to find the most wanted treasure.

Core (Ruben)

My responsibilities

I am in charge of developing most of the core of our game, including: collision systems, weapons, player syncing, graphics, and other misc logic.

Collision system

As this was completed by the first soutenance, there have been little changes to the collision system. It has proven robust over the rest of the development so far, leading to it being kept as is.

Weapons

Weapons being a large part of our game, they were the logical thing to add next. So far, there are 3 types of weapons: Bows, Daggers, and Guns.

- Bow → *shoots physics-based projectiles with an initial velocity, they are destroyed on hit with the world or a player*
- Dagger → *stabs a given distance*
- Guns → *hitscans with a max distance, or cut short by an obstacle*

All these weapons have cooldowns and other attributes dictated by an easy-to-edit JSON file containing these weapons and their variants (a Bow could be a standard-bow, x-bow, super-bow, ...). A custom JSON object loader was implemented in order to make loading of weapons (or in the future other JSON-based items) easy.

Weapons are controlled through the mouse; left click to shoot, the weapon always points to the cursor.

Pointing towards the cursor was not simple. In Panda3D, the cursor is positioned in the viewport space, so has a position from (-1,-1) to (1,1) with (0,0) as the middle of the viewport. However we want to get the position of the mouse in the 3D world space, but the mouse can have an infinite number of positions in said world space. But using our camera, we can narrow down the possible positions to an infinite line in the world space based on our camera's

perspective. Finally, with a plane at z=0 (the weapons z-axis position), we can find the point where the cursor's infinite line and plane intersect, giving us a workable coordinate. Afterwards, simply converting the computed point into its cartesian form can give us an angle to put the weapon at with a fix radius r.

Placeholder weapon texture pointing towards the player's cursor

Weapons stay fixed to players, rotating around them, giving a fluid, satisfying, yet challenging combat experience. This mechanic was chosen to encourage players to move around to dodge projectiles, whilst still being able to attack enemies.

Player syncing

Our client-server tick-based topology (as explained further down by Damien) led to the need for a modular way to handle arbitrary 'events' such as a player damaging another. Hence the existence of an event queue for each client which, on each client tick, sends all the events to the server's own queue, and on a server tick, the server handles all queued events depending on the event type, params, sender, etc... For example a respawn event will distribute that the sender is back at max health to all clients except the sender itself.

Previously, there was no custom class for other players, instead they were simply normal entities whose coordinates changed. With the introduction of weapon synchronisation, we introduced ghost players, which are cut-down versions of a player's own local player. Similarly ghost projectiles follow the

physics of a client's real projectile, but don't do damage, they simply provide the visual.

Graphics

Our ship model was custom made, exported in the GLTF format; a format packaging UV mappings and the model geometry whilst also allowing for remote textures to be specified via a path. We used this feature to allow for texture sharing and facilitate texture-swapping.

The ship's collision model is seperated from the GLTF, instead being provided as a .obj, to prevent unnecessary, and compute-heavy extra collision surfaces.

Network (Damien)

Networking Recap

The multiplayer system is based on a client-server architecture using the Ursina Engine networking module.

- The server is authoritative and manages the global game state
- Clients send their state and receive updates

Communication uses TCP and RPCs, which simplify networking by handling data transmission and remote function calls automatically.

Improvements

Two main improvements were added:

▼ **Tick System**

The tick system was introduced to make the networking stable, consistent, and independent from frame rate.

Without ticks, network updates would be forced by the client's FPS, which causes :

- Different update rates between players
- Unstable behavior depending on performance
- Irregular packet sending (too fast or too slow)

By using a fixed tick rate :

- Updates are sent at a controlled and predictable rate

- The server maintains a consistent and fair game state

A tick represent a fixed time step, we set `tps = 20` on both the client and server.

Server-Side Tick Loop

The server runs a strict fixed loop. At each tick, the server :

- Processes incoming RPCs (`server.update()`)
- Update the authoritative state
- Broadcasts player states to all clients
- Processes queued events (damage, projectiles, ...)

Client-Side Tick System

The client does not block like the server. Instead, it uses an accumulator.

This allows :

- Running networking at a fixed rate
- While rendering stays variable (smooth FPS)

At each client tick (`newtork_tick()`) :

- Sends player position + velocity
- Sends synced children (weapons, ...)
- Sends queued gameplay events

▼ Interpolation

To improve visual smoothness, interpolation was added on the client side.

Because the server sends updates only at each tick, directly applying positions would result in laggy movements. Instead, the client smoothly transitions between received states.

How it works

For each remote player, the client stores:

- The previous state (`first`)
- The latest received state (`last`)

At each frame, the position is computed using linear interpolation :

- The interpolation factor depends on how far we are between two ticks
- It uses the networks accumulator (`net_accumulator / dt`)

Conclusion

Overall, our network implementation is now stable, consistent, and visually smooth.
In the future, we could add even advanced features such as client-side prediction, reconnection handling and security mechanisms.

AI Enemies

Overview

Enemy behavior is implemented using a simple state-based AI system.

Each enemy reacts to the player based on distance and environment, allowing basic but effective gameplay interactions.

State System

The AI is driven by a finite set of states :

- IDLE → Not target or inactive
- CHASE → Moves toward the player
- ATTACK → Damages the player when in range
- DEAD → Entity is removed when health reaches zero

The state is updated every frame depending on the distance to the target.

Behavior Logic

- The enemy continuously checks the distance to its target
- If the player is far → switches to CHASE
- If the player is close → switches to ATTACK

During CHASE:

- The enemy moves toward the player on the X axis
- Uses collision detection (boxcast) to detect obstacles
- If blocked, the enemy performs a jump

During ATTACK:

- The enemy deals damage at a fixed rate using a cooldown system

Physics and Movement

The AI includes a simple physics system:

- Gravity is applied every frame
- Vertical velocity is updated manually
- A raycast detects ground collision
- Prevents falling through the floor and handles landing

Conclusion

This AI system provides simple but functional enemy behavior.

However, several improvements are still possible:

- Add pathfinding to navigate more complex environments
- Implement more advanced behaviors
- Improve combat logic
- Enhance physics and movement

Client Features

Death and Respawn System

The client handles player death and respawn locally to provide immediate feedback.

When the player’s health reaches zero:

- The player entity is disabled
- A death UI is displayed with a respawn button

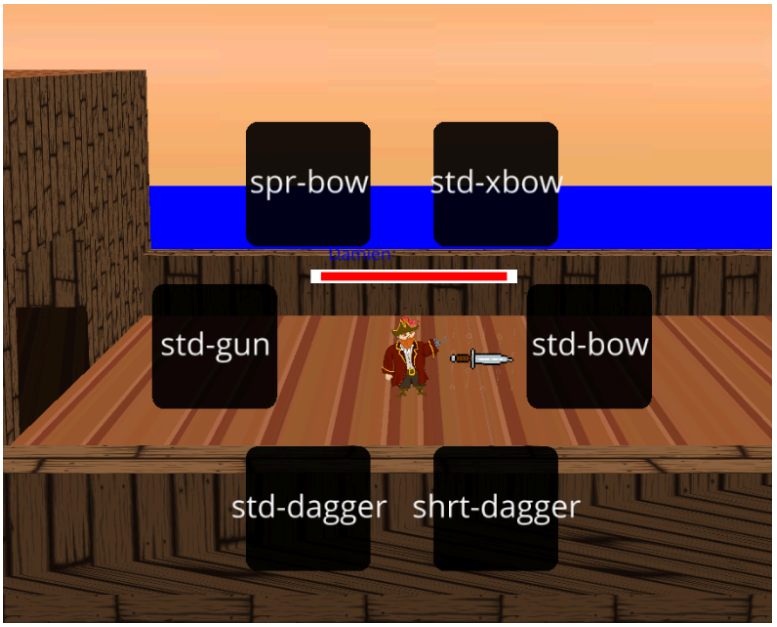
When the player respawns :

- Health is restored to maximum
- The player is repositioned at a spawn point
- The UI is removed and control is restored

Weapon Selection Wheel

A weapon selection wheel was implemented to allow quick and intuitive weapon switching.

- Triggered by holding a key
- Displays all available weapons arranged in a circular layout
- Each weapon is represented by a button positioned using trigonometry



Graphics

At this time of the project we wanted to have a clean game with nice elements as to have a view of the final product. Lots of elements were therefore added and implemented in our game.

Starting with the logo of our game that fade in and fade out when we call our game.

This logo will also be seen on our website alongside the other logo made to represent our team name.

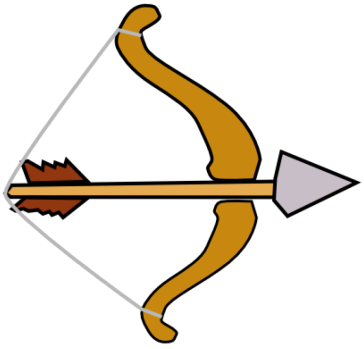
Moreover to have an immersive game we chose a special font which, of course, goes with our pirate theme.

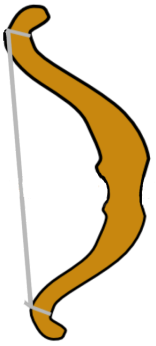
We need some weapons in order to attack and defend ourselves. There are three weapons which each have three levels which means that they can be upgraded(and downgraded).

Here are the bows and arrows:

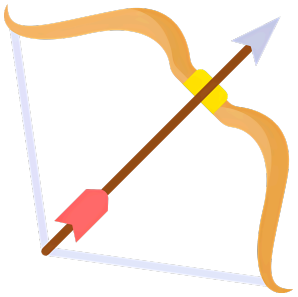
They all have two states to make the movement and the action while playing more realistic.

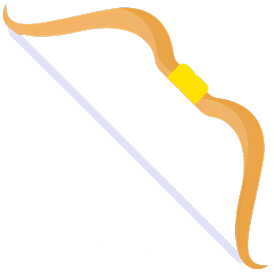
Level 1:



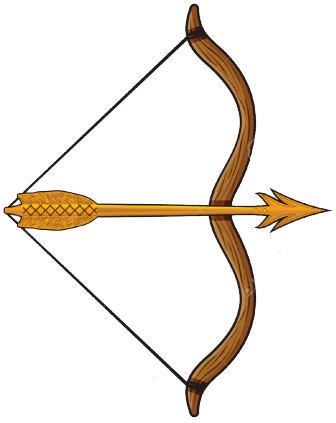


Level 2:



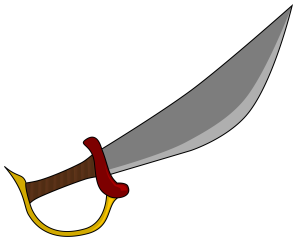


Level 3:



Here are the daggers:

Level 1, 2 and 3



And the guns:

Level 1, 2 and 3



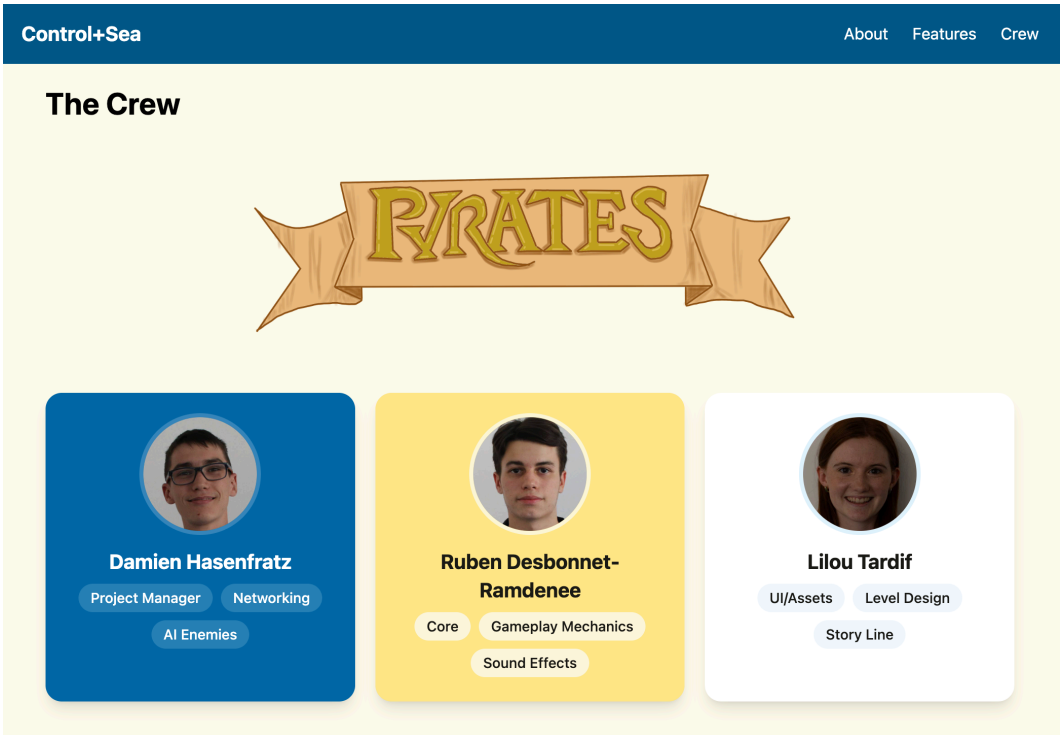
The weapons were unfortunately not drawn by hand as time and resources were lacking(a team member left and was suppose to help in the graphics). But they are all open source and free to use in our game(as it is a personal project). The bows were manually redrawn without the arrows.

Website

Overview

A simple website was developed to present the game.

The website is built using HTML and styled with the Tailwind CSS framework, allowing rapid and responsive design.



Structure and Content

The website is organized in several sections :

- Navigation Bar
- Hero Section
- About Section
- Features Section
- Crew Section

Design Choices

- Use of Tailwind CSS for fast and consistent styling
- Color palette inspired by the pirate theme (blue, amber, white)
- Clean and minimal interface for readability

Conclusion

The website effectively showcases the project in a clear and structured way.

Future improvements could include :

- Adding screenshots or gameplay videos
- Provide a download section
- Improving interactivity (animations, transitions)

Conclusion

In the future:

- Improve AI enemies
- Create and implement the last characters
- Being able to see the boat on the map
- Create a lobby when joining the game
- Implement music and sound effects

What do we bypass:

- Complicate gameplay mechanics such as seeing the boat sail or events

- Storyline such as dialogues

For this second soutenance all the main aspects have been dealt with which leaves us with the implementation of levels and the upgrading of the different elements such as the boat or the weapons. As well as the reward system which goes with the previous point. Moreover we will work more in depth the AI to make them attack and not just follow as they do now. And sound effects will be implemented to add to the atmosphere.