Thesis Report

BLITZKRIEG Studio

By Yloan, Quentin, Aegon, David & Adam

TABLE OF CONTENT

I. THINGS DONE

Quick remainder of the CdST:

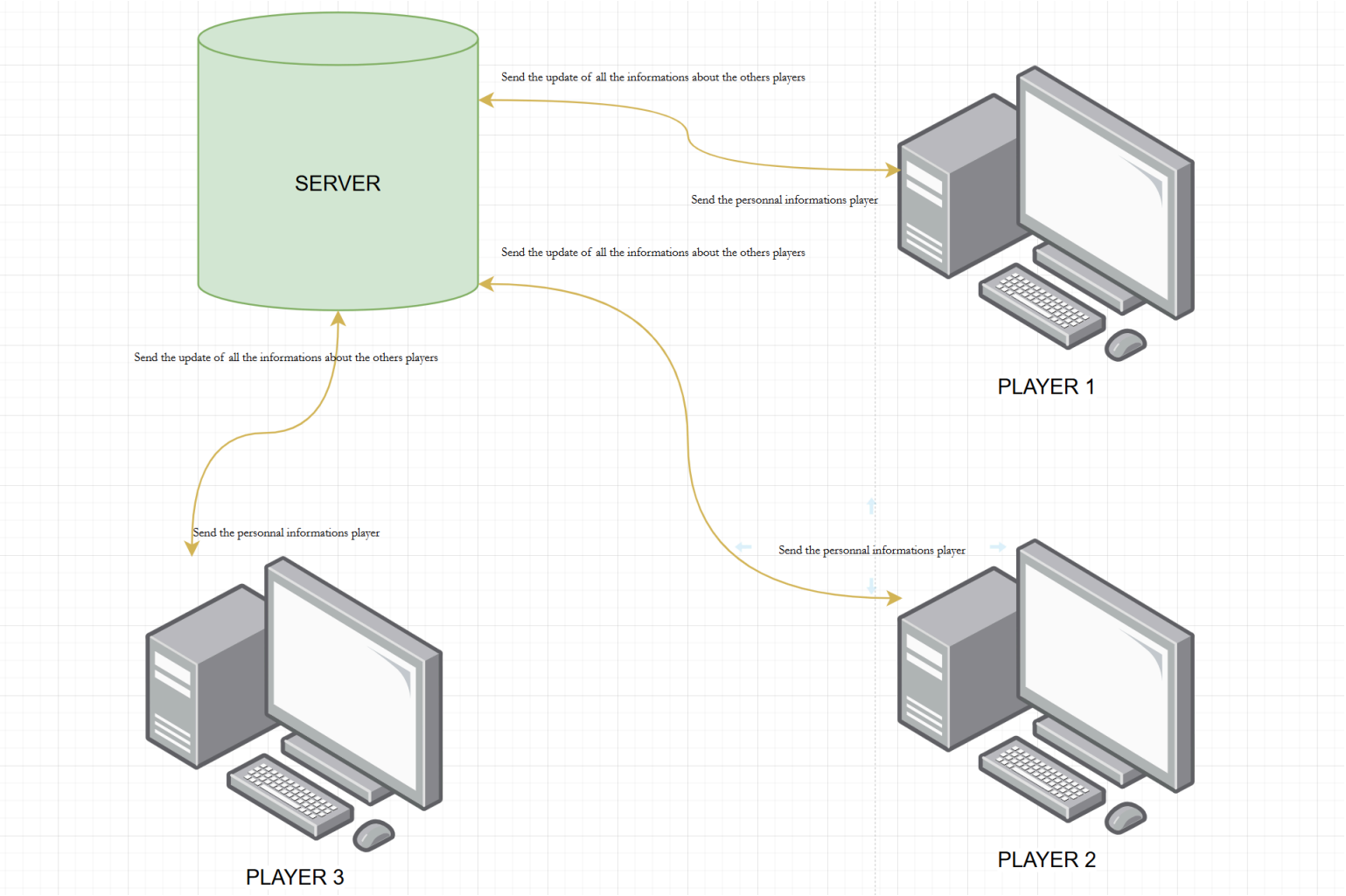
Task progress chart			
Tasks	**Soutenance 1**	**Soutenance 2**	**Final Sout.**
multiplayer	60 %	100%	100,00 %
game art	40 %	60%	100,00 %
music / sound effects	40 %	80%	100,00 %
menu	60 %	60%	100,00 %
character management	75 %	75%	100,00 %
HUD	20 %	60%	100,00 %
environment interaction management	50 %	80%	100,00 %
attack system	25 %	75%	100,00 %
website	0 %	10%	100,00 %
IA	10 %	60%	100,00 %

A. MULTIPLAYER ENGINE

The multiplayer system is functional because it can handle multiple clients using a thread-per-client model, where each client is managed independently. However, the engine is not hosted yet.

We plan to host it and eventually make it publicly accessible online. Otherwise, the code would need to be modified, since the server IP address is currently not fixed to a single machine. For now, the system works in a LAN environment.

The server will operate as follows for now. If we later find a more optimized approach to improve request-handling efficiency, we will update the implementation.



The Server

The server serves as the central arbiter in this configuration. Players only communicate with the server; they do not speak with one another directly.

- Upstream: Every player provides the server with their "personal information" (coordinates, actions, or state).
- Downstream: After processing these inputs, the server broadcasts the "world state" to all users. This guarantees that Player 1 is aware of the precise locations of Players 2 and 3. The game worlds would rapidly become out of sync without this central hub.

The Thread-per-Client Logic

Because a thread-per-client model is currently in use, the server creates a unique execution path for each connection.

- Why it works now: It is simple to put into practice and prevents one player's data processing from instantly obstructing another's.
- The Scalability Challenge: Threads are "expensive" for a CPU, but they work well for small groups or LANs. The overhead of managing all those threads (context switching) can cause the server to lag if we go to a public release with dozens or hundreds of players. Here, switching to an event-driven or asynchronous I/O model would be the "optimized approach" we previously discussed.

Moving from LAN to Online

The transition from a Local Area Network (LAN) to a public server (if we do it) involves two main shifts:

- Static Addressing: As we noted, the server needs a fixed entry point. In a LAN, we use local IPs (like *192.168.x.x*). For the web, we will need a Static IP or a DNS (Domain Name System) so the client code doesn't have to be hardcoded with a changing address.
- Latency Management: On a LAN, communication is nearly instant. Online, there is "lag". We may eventually need to add Client-Side Prediction (where the client localizes movement immediately) and Interpolation (to smooth out the movement of other players) so they don't appear to "teleport" across the screen.

B. CHARACTER'S MANAGEMENT

Object-Oriented Character Logic

In the code, each "Player" on the diagram isn't just a set of random numbers; it is an Instance of a Player or Character class. This means that when Player 1 sends their "personal information," the server receives a data package and maps it directly to a specific object in its memory. This makes the code much cleaner than using individual variables for every player's *X* and *Y* coordinates.

Encapsulation and Data Integrity

Encapsulation is a class system's greatest benefit. One unit contains all of the functions (Move, TakeDamage, Jump) and data (Health, Position, Velocity, Inventory).

- Protection: By employing logic within the class, we can guarantee that a player's health never falls below zero or exceeds its limit.
- Organization: To "Send the update of all information," the server merely iterates through a list of Player objects, extracting the necessary data from each one.

Abstraction for the Server

Abstraction is made possible by using classes. The server simply needs to know that Player 1 is an object of the Player class; it does not need to know Player 1's exact identity.

Enemies and NPCs (Non-Player Characters) may inherit from the same base class in the future.

When determining collisions or broadcasting locations, the server can handle both human players and AI bots in precisely the same way, greatly simplifying our networking code.

C. MENU

The game menu, managed by the Menu class, allows players to navigate between the main menu, settings, player selection, and character selection.

Each option is represented by interactive buttons that detect clicks and change the menu status.

Players can choose up to three characters, with an enlarged preview of the last one chosen and the previous ones in small size.

Back buttons allow players to change their choices. Images and buttons are automatically centered, and text is displayed clearly.

Once the selection is complete, the status changes to ‘start game’ to ‘launch the game’.

The menu thus provides an intuitive and dynamic interface for preparing multiplayer gameplay.

D. DESIGN (ART, MUSIC, ETC.)

Music

Music is an underappreciated part of the game since it adds to its engagement and gives it a unique mood and appeal. It is one of the primary reasons why a game is enjoyable to play aside from the gameplay. To complement our game's old-school pixelated design, we inspired ourselves with music from games such as Nuclear Throne or Pokemon. We chose to use the BandLab app, which is a platform that allows users to not only listen to music made by BandLab artists, but also make their own. The initial plan was to produce four songs: one for the game interface and character selection menu, and three for the various maps. We determined that because it is difficult to develop our own original music pieces, we would make two of them and the other two would be tunes that match the game and that we would modify slightly. As of now, we've generated one unique composition utilizing BandLab tools and settled on two songs that will be included in the game. These two tracks were taken from the game Nuclear Throne and do an excellent job of representing the theme of the game. Because the songs had a distinct beginning and conclusion, these sections needed to be cut so that the music could be played in a loop without pausing or stopping.

E. BOT DISCORD

To act as the focal point of our project management, we have included a specialized Discord bot into our workplace. We remove the hassle of hopping between several applications by relocating our process to the same space where we communicate, guaranteeing that our company stays cohesive and open.

Here's how our bot works and why it's essential to our organization:

Centralized To-Do List and Task Management

Our project roadmap is a dynamic "To-Do" list that our bot keeps up to date.

What it does: We may use Discord to create, assign, and monitor jobs.
Progress tracking allows us to mark jobs as completed, giving us instant visual feedback on our progress.
Data-Driven Insights: The bot monitors data, including the number of jobs left compared to those finished. This enables us to see how much work we have to do and modify our speed accordingly.

A built-in bug tracker

We employ a systematic ticket system for technical concerns in order to guarantee the caliber of our production.
Ticket Generation: We create a particular ticket whenever an issue is found. This keeps problems from getting lost in the broader conversation.
Resolution Workflow: Every problem goes through a lifecycle from "Open" to "Resolved" (with some other specified parameters like "Priority" …), providing us with a clear record of our technical debt and the speed at which we are fixing it.

Why This Enhances Our Organization

This bot contributes to the discipline of our team and is more than simply a tool. It benefits us in three main ways:

- ☑ Accountability: We always know who is doing what because our server's statistics and tasks are public. The team remains together because of this transparency.
- ☑ Efficiency: We avoid wasting time pondering "what's next?" The bot responds to our current priorities in real time.
- ☑ Clarity: We eliminate the "feeling" of being overwhelmed and replace it with concrete evidence by measuring our progress (remaining tasks vs. completed activities).

Hosting & Maintenance

At the moment, we are hosting using Katabump. This is a calculated move that keeps project expenses at zero while preserving a useful tool.

Note on Maintenance: The server needs a manual "refresh" and "rerun" every four days because we are utilizing the free tier. We handle this as a brief administrative "heartbeat" to maintain the functionality of our infrastructure and the accessibility of our data (see the server's image in the Annexe (1)).

F. Artificial Intelligence

AI Core: Implementation & Conceptualization

On the technical side, we have laid the groundwork for the game's Intelligence. We are focusing on a clean, modular architecture using Python to ensure long-term scalability.

Technical Foundation

We have successfully implemented the base AI class structure. This includes:

- Class Architecture: Making the specific Python file and class structure.
- Initialization: The `__init__` method is implemented, which enables us to create and save the crucial variables required for the AI to operate.
- Future-Proofing: The system's "skeleton" is currently prepared for integration, but the precise logic techniques will be created later this year.

Strategic Logic & Research

We have spent a lot of work on the AI's conceptual design in addition to the programming. We've established:

- Behavioral Patterns: The AI's response to player activities.
- Implementation Strategy: The particular logic flows and algorithms we'll employ to make the AI difficult yet manageable.
- Integration: The way the AI built on Python will interact with the other components of our game engine.

II. EACH MEMBER'S GROUP IMPLICATION

A. Yloan

A significant portion of the project's technical structure and organization are under my control. I contribute to the group's organization by creating and hosting a JavaScript Discord bot and by actively participating in team coordination on the Discord server.

In addition, I established and oversees the project's GitHub repository. To keep the codebase manageable, this includes defining the project architecture, organizing the scripts, and setting up a clear structure.

From a technical perspective, I implemented the character management system, including character movement already integrated into the game. I also developed the multiplayer engine, covering the entire server-side logic.

To organize the project more effectively, I proposed and applied an object-oriented programming (OOP) approach. Given my prior experience with OOP, this choice improved code organization, readability, and scalability.

B. Aegon

C. David

As the music composer, I am in charge of the game's music implying finding what music to put where and how to make that music. Since I had zero previous knowledge in song making, I started by learning using Youtube videos to learn how to make music for video games, I

D. Quentin

As the co-director and support developer, I am in charge of the game's menu,

E. Adam

As the interface designer and web developer of the project, I am responsible for the design of the game interfaces and the development of the project website. I have coded it in collaboration with David and have completed it. The website serves as a platform where the information about the project concept and development will be shared. This sharing of information will be done in an understanding manner. At the same time, I concentrate on the design of the project interfaces from a graphic aspect and do not code. I design the menus, character selection scenes, buttons, icons, typefaces, and color schemes for the project. I make sure that the project will have a visually cohesive look in a readable manner.

F. All members' group

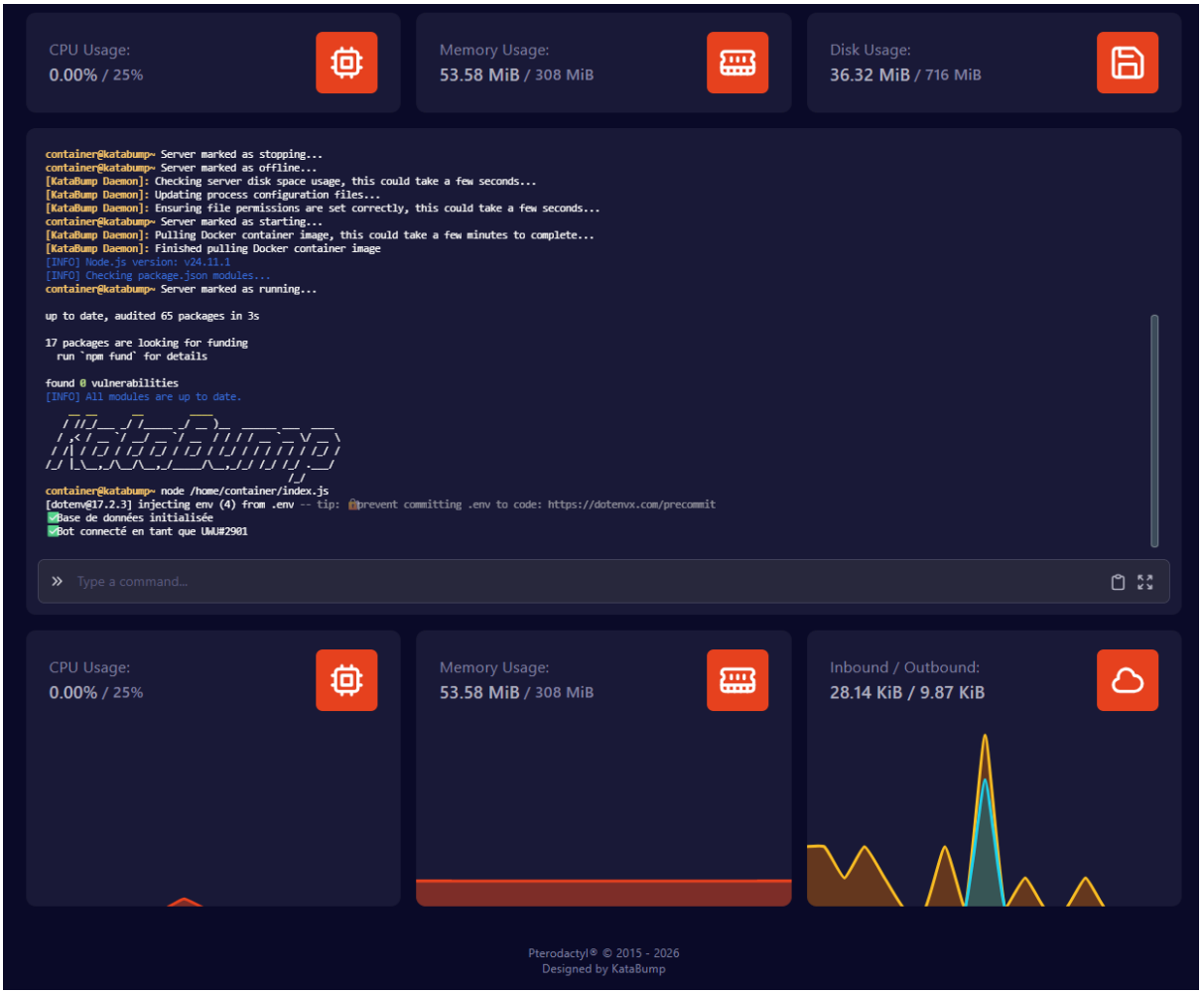
All team members must consistently and strongly participate in the project. In addition to their specific technical activities, each member contributes to the project by taking part in shared duties like organizing the workload, creating presentations, and composing the project report.

All team members are active on the Discord server, which facilitates consistent and organized communication. This platform is used to discuss developments, exchange technical details, and work together to resolve various development-related problems.

When organizational or technical challenges emerge, the team collaborates to identify issues and suggest fixes. Despite limitations and difficulties, this collaborative approach guarantees consistent progress and helps preserve project coherence.

III. ANNEXE

1.


CPU Usage:
0.00% / 25%
Memory Usage:
53.58 MiB / 308 MiB
Disk Usage:
36.32 MiB / 716 MiB
container@katabump~ Server marked as stopping...
container@katabump~ Server marked as offline...
[KataBump Daemon]: Checking server disk space usage, this could take a few seconds...
[KataBump Daemon]: Updating process configuration files...
[KataBump Daemon]: Ensuring file permissions are set correctly, this could take a few seconds...
container@katabump~ Server marked as starting...
[KataBump Daemon]: Pulling Docker container image, this could take a few minutes to complete...
[KataBump Daemon]: Finished pulling Docker container image
[INFO] Node.js version: v24.11.1
[INFO] Checking package.json modules...
container@katabump~ Server marked as running...
up to date, audited 65 packages in 3s
17 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities
[INFO] All modules are up to date.
container@katabump~ node /home/container/index.js
[dotenv@17.2.3] injecting env (4) from .env -- tip: prevent committing .env to code: https://dotenvx.com/precommit
Base de données initialisée
Bot connecté en tant que UWU#2901
Type a command...
CPU Usage:
0.00% / 25%
Memory Usage:
53.58 MiB / 308 MiB
Inbound / Outbound:
28.14 KiB / 9.87 KiB
Pterodactyl® © 2015 - 2026
Designed by KataBump