Thesis Report

BLITZKRIEG Studio

By Yloan, Quentin, Aegon, David & Adam

TABLE OF CONTENT

I. THINGS DONE

Quick remainder of the CdST:

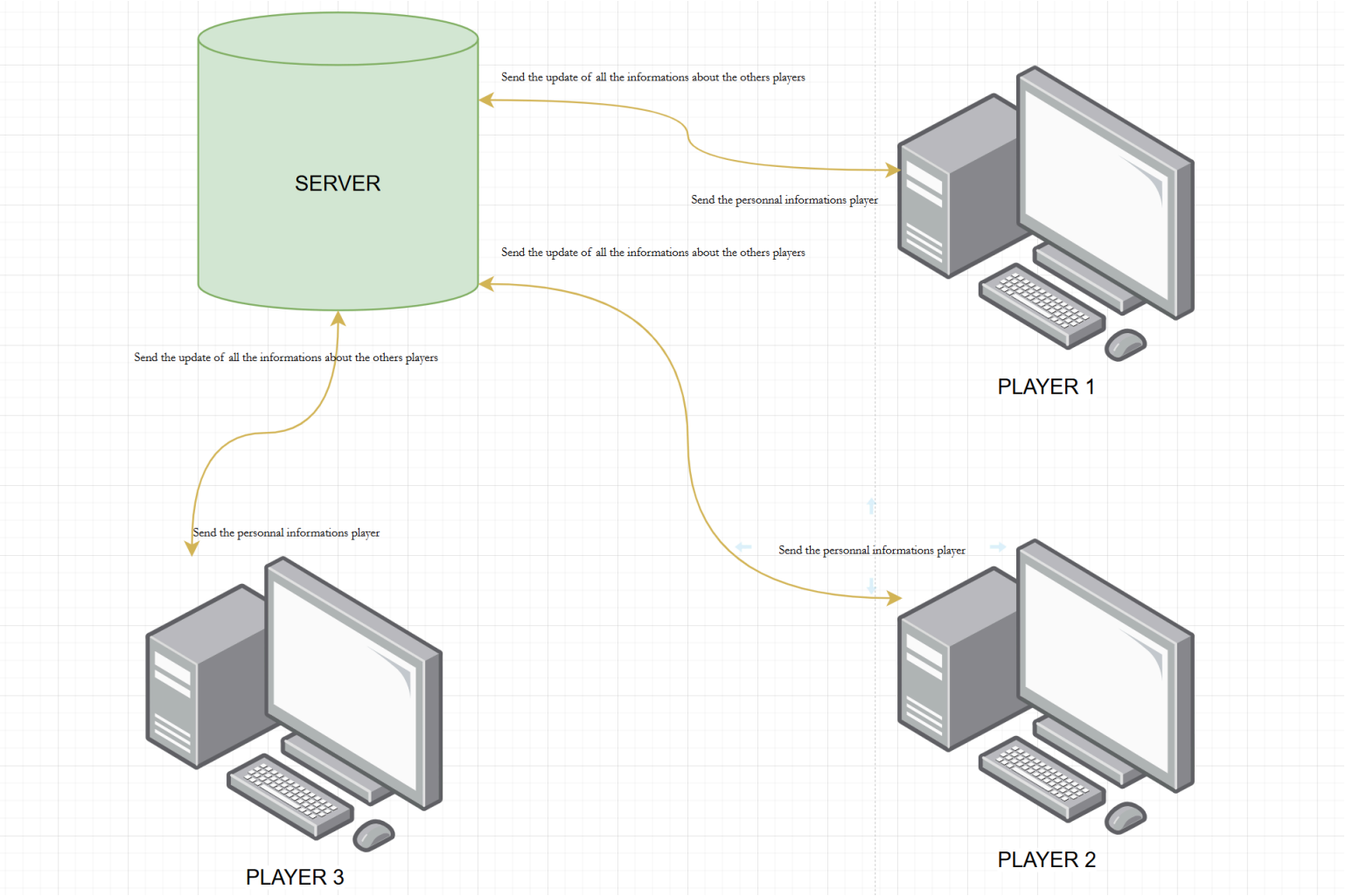
Task progress chart			
Tasks	**Soutenance 1**	**Soutenance 2**	**Final Sout.**
multiplayer	60 %	100%	100,00 %
game art	40 %	60%	100,00 %
music / sound effects	40 %	80%	100,00 %
menu	60 %	60%	100,00 %
character management	75 %	75%	100,00 %
HUD	20 %	60%	100,00 %
environment interaction management	50 %	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, is responsible for handling all the user interactions before the game starts. It allows players to navigate smoothly between different sections such as the main menu, settings, player selection, and character selection screens. Each menu option is represented by interactive buttons that respond to mouse clicks and visually change when hovered over, providing clear feedback to the player.

The menu system uses a status-based logic to switch between screens. When a player clicks on a button, the current menu state is updated, which triggers the display of the corresponding interface. This makes the navigation simple and

intuitive. Back buttons are also implemented, allowing players to return to previous menus and modify their choices at any time.

During the character selection phase, players can choose up to three characters. The most recently selected character is displayed in a larger preview, while the previously chosen ones appear in smaller size. This visual system helps players easily track their selections. All images, buttons, and text are automatically centered on the screen to ensure a clean and well-organized layout, regardless of the resolution.

Once the character selection is complete, the menu status changes to “start game,” which launches the gameplay phase. Overall, the menu provides a dynamic, user-friendly, and visually clear interface that prepares players efficiently for a multiplayer gaming experience.

Image 2 in Annexes.

D. DESIGN (ART, MUSIC, ETC.)

Art

For many people, an art style can be the sole reason they are first drawn to a game, similarly to the thumbnail of a video, as such, the art style of Blitzkrieg had to be in line with the content of the game and the type of people we want it to appeal to. As such, we decided that we would entirely use pixel art for the game, which is fitting since the characters are robots. Art is omnipresent in a game as it is what is displayed on someone’s screen at all times. As such, a game’s art is what the game conveys to the player (the player only knows something happened if it is shown to them) and is of utmost importance. Blitzkrieg’s art encapsulate the menus, the buttons, the characters, the backgrounds, the HUD and

Among these, the buttons, characters and HUD are animated.

Every button in the game possesses a “pressed” animation, indicating that said button was pressed by the player.

The HUD is one of the most fundamental elements of the gameplay. Indeed, the HUD contains a plethora of information that the player needs while fighting that is condensed into a compact and readable interface. In our game, the HUD contains elements such as the current health of the character, the resistances and weaknesses of the character, the cooldowns of every move the character can do and unique elements tied to a character specific mechanic (ex: the number of ammunition a character has).

Characters are the most time consuming art-wise as, unlike the other elements of the game, they not only possess the most animations out of any elements of the game (generally 7 long animations, sometime more and sometime with variations depending on if an attack hit a target or not), but they are also the hardest to animate due to their high number (18 characters) and the complexity of their animations (it's easier to animate a button being pressed than a character doing an action as the button is only a rectangle with low details whereas a character has limbs, moves more dynamically and needs more frames (pictures) to be animated). Additionally, the visual effects of every attack of the characters also have to be drawn (ex: lightning, fire, slash, explosion,...), animated, and implemented only in specific cases (ex: when you touch an enemy) meaning that some attacks have multiple versions and that each version of said attack has to be animated independently. The pacing of the animations is also very important as not using something called the 12 principles of animation can have the effect of an attack not seeming strong or smooth which can make the player think that said attack is weak despite that not being true. A lot of thought has to be put while drawing each frame (picture) composing an animation so that when put together the result conveys the correct intention. On average, a character with every necessary animations possess 75 frames across every animations, meaning that optimistically every characters amount to a grand total o

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.

How will the AI work ?

This 2D battle game uses a combination of Behavior Trees and Finite State Machines to create intelligent enemy AI that provides clear decision-making, predictable behavior, and precise control over combat actions.

At the top level, a Behavior Tree determines what the character should do by continuously evaluating conditions arranged in a hierarchy—first checking key factors like whether the player is visible, within attack range, or if the enemy's health is low, then selecting an appropriate action such as approaching the player, attacking, defending, or retreating based on these conditions.

Once an action is chosen, a Finite State Machine (FSM) handles the execution by managing concrete states such as movement, defensive stance, attack animations, and recovery, with each state having clear entry and exit conditions that ensure smooth transitions and reliable behavior.

This separation keeps decision logic clean while making execution precise and dependable. The AI considers multiple variables when making combat decisions, including remaining health, stamina levels, cooldown timers, and distance to the player, with these parameters determining which branch of the Behavior Tree activates, allowing enemies to respond appropriately to different combat scenarios.

Reaction delays are built in to prevent instant responses and create more realistic behavior. This deterministic design means AI behavior follows predetermined rules and conditions, offering key advantages including being easy to debug and modify, simple to balance for competitive gameplay, requiring no machine learning, and creating responsive, believable opponent behavior.

By cleanly separating decision-making (Behavior Tree) from action execution (FSM), the system produces intelligent enemies perfect for action-oriented 2D games, and in practice, this tree structure can be implemented using conditional (if-else) patterns in Python, as shown in the visual schema provided in Annex (3).

II. EACH MEMBER'S GROUP IMPLICATION

A. Yloan

As the lead programmer, the project's technical structure and organization are under my control. I contributed to the group's organization as Team Coordinator by creating and hosting a custom JavaScript Discord bot that facilitated project management and communication. I actively participated in team coordination through our Discord server, where I helped maintain clear communication channels, tracked progress, and ensured everyone stayed informed about deadlines and responsibilities.

Additionally, I led regular meetings attended by all team members, where I set agendas, guided discussions, facilitated decision-making, and ensured that everyone had the opportunity to voice their ideas and concerns while keeping us focused on our project goals.
In addition, I established and oversaw 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.

One of the main issues I encountered involved the multiplayer engine and understanding network architecture. While I successfully created a socket server in Python using the documentation, I struggled to grasp the fundamental differences between online servers and local servers.

I understand the basic concept: if you have the correct IP address, you can connect to a server. However, what confused me was the scope and accessibility of different types of IP addresses. For example, if I run my server in Fontenay-Sous-Bois, France (94120), and it gets a private IP address on my local network, can someone in Villejuif (another nearby city) connect to it if they're on a completely different network? This question led to a bigger one: How do you make a multiplayer server truly accessible online?

The core of my confusion is this: if a server running on my home computer only has a private IP address, it's restricted to my local network—meaning only devices connected to my router can access it. So how do servers located in the US become accessible to players in France? What's the technical difference between a server that only works on a local network (LAN) and one that's accessible over the internet (WAN)? I realized that professional online servers use public IP addresses and proper network configuration from the start, but what steps or configurations—such

as port forwarding, using my router's public IP, or cloud hosting—are required to transform my local Python socket server into one that can accept connections from anywhere in the world?

Basically, at the beginning I had trouble understanding the network system.

I was also debating how to construct the opponent AI system, which was a significant decision. I was torn between two quite distinct strategies. Using machine learning, which would enable the AI to learn and modify its behavior via experience and training, was the initial choice. Since the AI might theoretically get better over time and react to novel player tactics, this strategy sounded more contemporary and possibly more dynamic. In order to provide deterministic, predictable behavior based on preset rules and conditions, the second option was to design a hierarchical system utilizing well-known techniques like Behavior Trees and Finite State Machines.

Ultimately, my lack of machine learning skills was the main reason I went with the hierarchical architecture. My understanding of neural networks, training procedures, and the time and computational resources needed to effectively train an AI agent for combat scenarios was inadequate. Furthermore, in a 2D fighting game where predictable, balanced, and debuggable behavior was more crucial than adaptive learning, machine learning seemed like an unneeded complexity. For my skill level and project scope, the hierarchical approach was the most practical option because it provided more dependable performance, easier debugging, and clearer control.

Let's now discuss the hosting problems I ran into, which ended up being one of the most annoying parts of the project. Finding a suitable, trustworthy, and—most importantly—free option to host the Discord bot proved to be somewhat difficult. This became a significant obstacle since, although creating the bot locally was simple, maintaining it constantly so that people could engage with it at any time required a dedicated server, and the majority of hosting options have monthly fees that were out of my means as a student.

I eventually discovered a solution that satisfied my needs after spending several days doing in-depth research and browsing through innumerable forums, Reddit posts, YouTube videos, and hosting provider websites. But it wasn't flawless. One major drawback of the free hosting option I found was that it required a manual server restart every four days. This means that the bot goes down about every ninety-six hours, and in order to get it back online, I have to manually restart the server by logging into the hosting platform.This was the best option I could discover between having a working, publicly accessible Discord bot and keeping hosting costs free, even if it isn't ideal for user experience because the bot occasionally goes down. The other options would have been to either pay for premium hosting or run the bot continually from my personal computer, which would have meant dealing with possible internet connection problems and keeping my machine on all the time.

B. Aegon

As the director of the project, I am the one coming up with ideas for characters, mechanics, and many more. I am also the lead artist of the game, drawing and animating almost all of the content of the game, from the characters and their attacks to the menus and interfaces (such as the in-game HUD who shows things such as health, resistances, and cooldowns). I have many years of experience as a pixel artist using Aseprite. In addition, I am also the one responsible for creating the sound effects (from attacks, button presses, and more). For this, I use an app called FamiTracker, with which I can produce authentic sound effects from an SNES/Famicom system, giving the game a retro feel (in addition to the pixel art style). I also have multiple experiences in modding games, where I would draw sprites and code a mod in order to add a character/item to a game I like, which will be proven useful when implementing the sounds/sprites in game.

C. David

As the music composer, I am responsible for the game's soundtrack, which includes deciding what music to place where and how to create that music. Because I had no prior experience with songwriting, I began by watching YouTube tutorials to learn how to produce music for video games. I immediately found that the work at hand

was more complicated than I had anticipated because you needed to know how to play an instrument to make music. After some trial and error, I decided to compose some music with Banlab, and it paid off handsomely since it allowed users to utilize preset "songs" of certain instruments, which I mashed up and altered to create what I might call my own work. Then, my teammate suggested some songs that might work as the in-game battle music. In order to avoid interfering with the player's concentration during a game, I slowed down the extremely fast tempo of those songs. I also contributed to the design and development of the website for the game. It was really difficult because I had never coded a website before, but with my partners' assistance, we were able to get a solution that pleased us all.

D. Quentin

As the co-director and support developer, I am in charge of the game's menu.
I designed and implemented both the main menu and the in-game menu system.
My role includes creating an intuitive user interface for smooth navigation.
I also ensured that the menus are responsive and adapted to different screen formats.
This work improves the overall player experience and accessibility of the game.

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

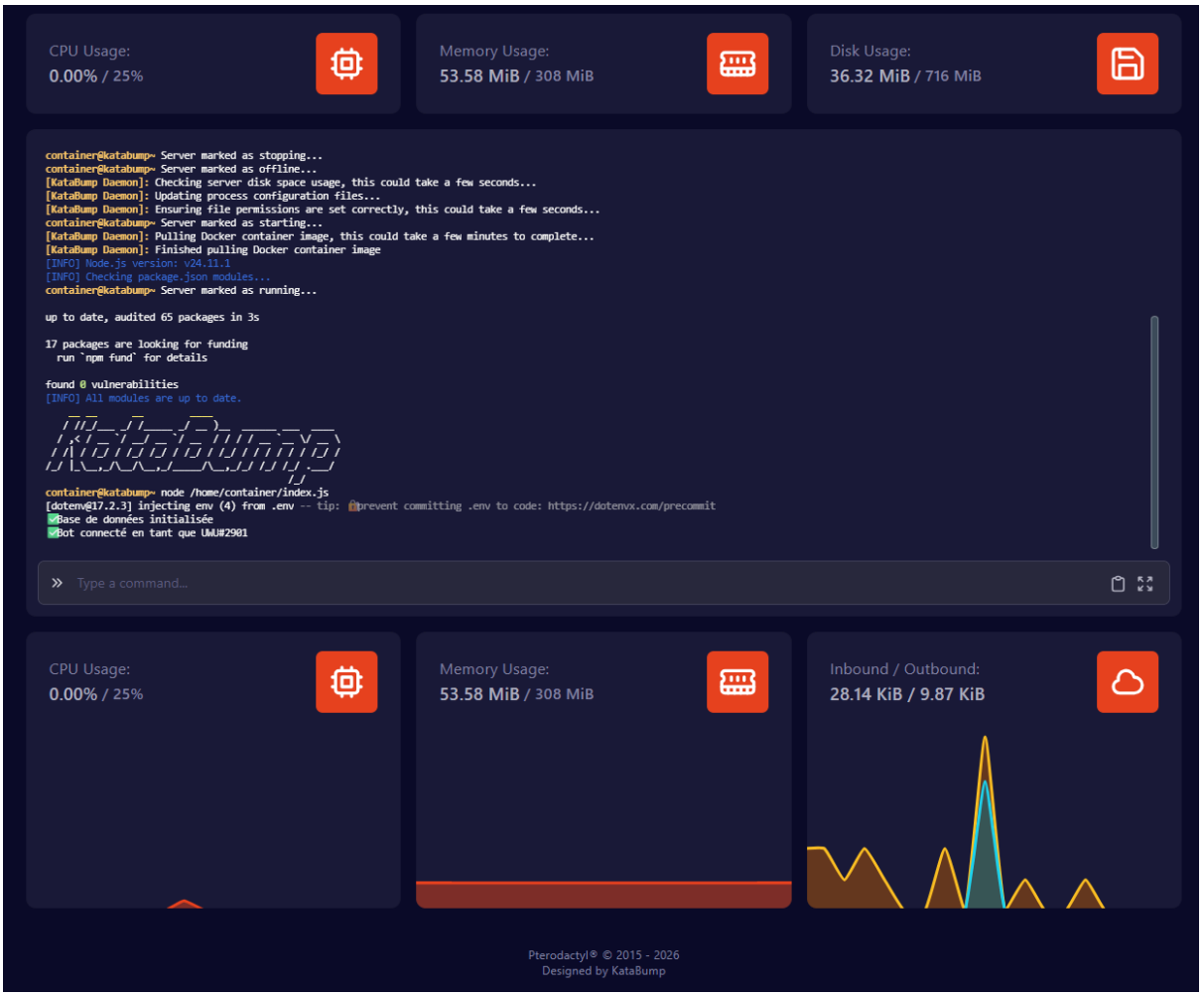
All team members must consistently and strongly participate in the project. In addition to their specific technical activities, each member contributes to the project by taking part in shared duties like organizing the workload, creating presentations, and composing the project report.

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.

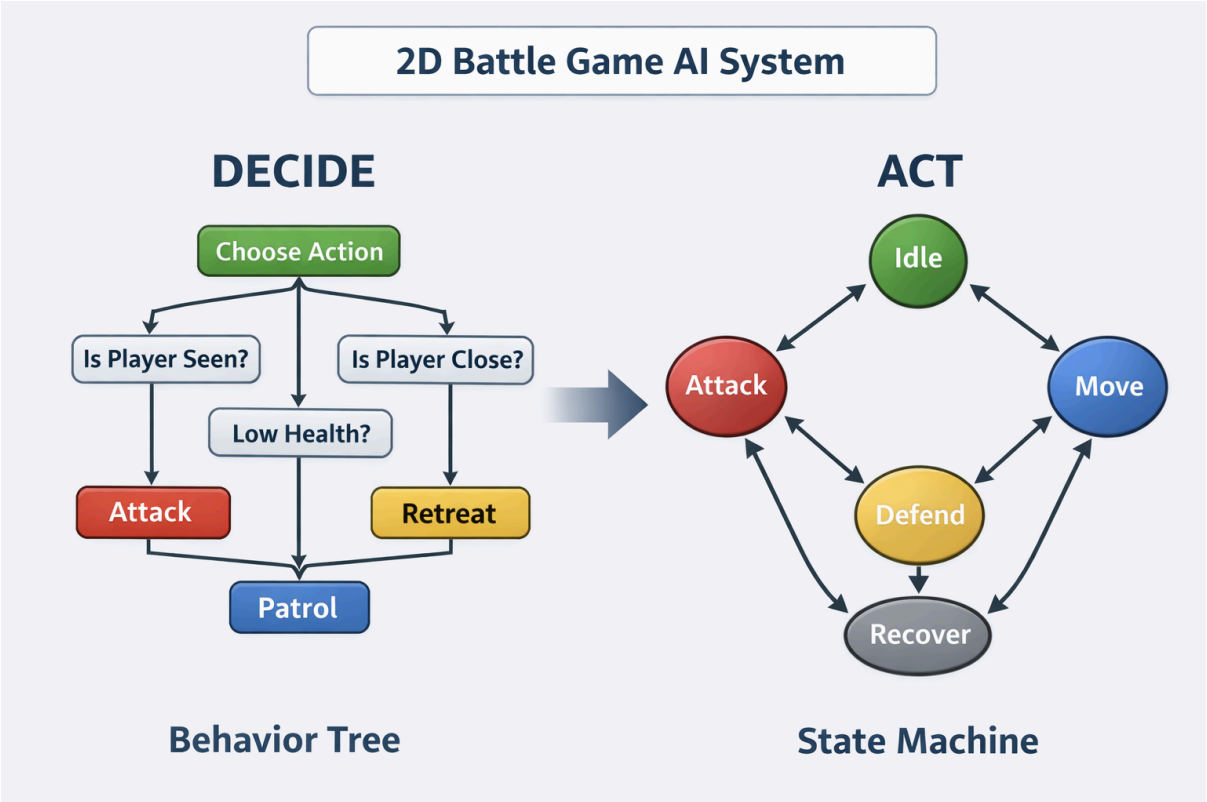
III. ANNEX

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

2)

3)


2D Battle Game AI System
DECIDE
ACT
Choose Action
Is Player Seen?
Is Player Close?
Low Health?
Attack
Retreat
Patrol
Idle
Attack
Move
Defend
Recover
Behavior Tree
State Machine