Specifications

3A Group

ANTONIOS Elie | ARMAGAN Omer Ali | BAH Mamadou Mouctar | CHIKHI Lounas | DELGADO Darren | KRIKA Neila | GERARD Kylian

CONTENTS

1. Project Overview

1.1 Game Description

We are developing a multiplayer online board game, initially designed for local execution before being expanded to other connected environments. All clients, regardless of their platform, will interact with a central server.

Our game of choice is "*6 qui prend !*", as it perfectly meets the project requirements.

- It is a multiplayer game, fostering interaction among participants.
- It has a quick learning curve, allowing players to grasp the rules easily.
- Its dynamic gameplay, where each card played influences the game's progression, makes it strategic, engaging, and entertaining.

It is a card game for 2 to 10 players. The goal is to avoid collecting cards, as each card contains "bullheads" that represent penalty points.

The game consists of 104 numbered cards (1 to 104), each featuring between 1 and 7 bullheads:

- Cards ending in 5 or 0 tend to have more bullheads, making them riskier to collect.
- The most punishing card is 55, which has 7 bullheads.

1.1.1 Game Setup

- Each player is dealt 10 cards randomly.
- 4 cards are placed in the center to start four rows.
- At the beginning of each round, players simultaneously choose one card from their hand and place it face down.

1.1.2 Round Progression

All players reveal their chosen card at the same time.

The cards are then placed into rows, following these rules:

1. **A card must be placed in the row where the last card has the closest lower value.**
2. **If a card is the sixth in a row, the player must collect the five previous cards and place their card as the new starting card for that row.**
3. **If a card is too low to be placed in any row, the player must take an entire row of their choice, and their card becomes the first in the new row.**

1.1.3 End of a Round & Game End

- **A round ends when all players have played their 10 cards.**
- **Players count the bullheads they have collected (penalty points).**
- **The game is played over multiple rounds until a player reaches a predefined score (typically 66 points).**
- **The player with the fewest bullheads at the end of the game wins.**

1.2 Client-Server Roles & Data Exchange

The client acts as the interface for the player. It displays the board, the cards, and all necessary game-related information. It also captures player interactions, such as selecting a card or pressing a button, and sends them to the server.

The server is the core of the game, where all game logic is centralized. It enforces the rules, validates player actions, and updates the game state. Additionally, it manages player connections, synchronizes actions, and handles score storage.

The main data exchanges between the client and the server will include:

- **Client:** Displays the board, manages user input, and sends selected actions to the server.
- **Server:** Manages game logic, ensures rule compliance, updates game state, and synchronizes players.
- **Data exchanged:** Player actions, board state updates, scores, and game events.

By centralizing game rule enforcement on the server, we avoid inconsistencies between different clients. This also ensures seamless communication between players and guarantees game integrity.

1.3 Game Algorithm (Pseudocode)

```
Start game
Initialize the deck with 104 numbered cards from 1 to 104
Shuffle the deck
Distribute 10 cards to each player
Place 4 cards from the deck to form the 4 starting columns
While the game is not over
    Start of a round
    While players still have cards in hand
        Each player selects a card to play and places it face down
        Reveal all played cards simultaneously
        Sort the played cards in ascending order
        For each played card
            Find the column where the card should be placed
            If the card is the 6th in the column
                The player takes the 5 cards and places their card to start a new column
                Add the bullheads from the collected cards to the players score
            Else
                Place the card in the corresponding column
            End If
        End For
    End While
    Display the round scores
    Check if a player has reached 66 points or more
    If yes, end the game and declare the player with the fewest points as the winner
End While
End game
```

Illustration of the game flow through a structured pseudocode representation

2. Key Features

2.1.1 Database & Security

- **Database Integration:** A MySQL database will store player accounts, player data, best scores, saved game sessions, all lobby, game states...
- **Security Measures:**
    - Client-server communication will be secured with encryption and authentication mechanisms.
    - The database will be isolated from the main server to prevent unauthorized access.

2.1.2 Multiplayer & Game State Management

- **Real-Time Multiplayer:** The server will efficiently handle multiple concurrent game sessions, ensuring smooth interactions between players.
- **Game State Management:**
    - The server synchronizes the game state across all players.
    - Every action is validated before being broadcast to ensure consistency.

2.1.3 Game Mechanics

- **Card System:**
    - Each card is an object containing a number and bullheads (penalty points).
    - Players have a dynamic hand of 10 cards.
- **Game Board Structure:**
    - The board consists of four columns, implemented as arrays or linked lists.
    - Cards are placed according to specific game rules.
- **Turn Logic:**
    - Players select and reveal a card simultaneously.
    - The program sorts cards in ascending order and determines placement using an optimized search algorithm.
- **Rule Enforcement:**
    - If a card is the 6th in a row, the player collects the five previous cards and updates their score.
    - Special cases, such as forced column pickups, are handled automatically.

2.1.4 User Interface & Player Experience

- **Intuitive UI:** Players interact using mouse and keyboard controls.
- **Visual & Audio:**
    - Game animations and sounds enhance immersion.
    - UI components ensure a smooth and responsive experience.

2.1.5 Networking & Matchmaking

- **Secure Multiplayer System:**
    - Ensures only authorized players can join sessions.
    - Uses WebSockets for real-time data exchange.

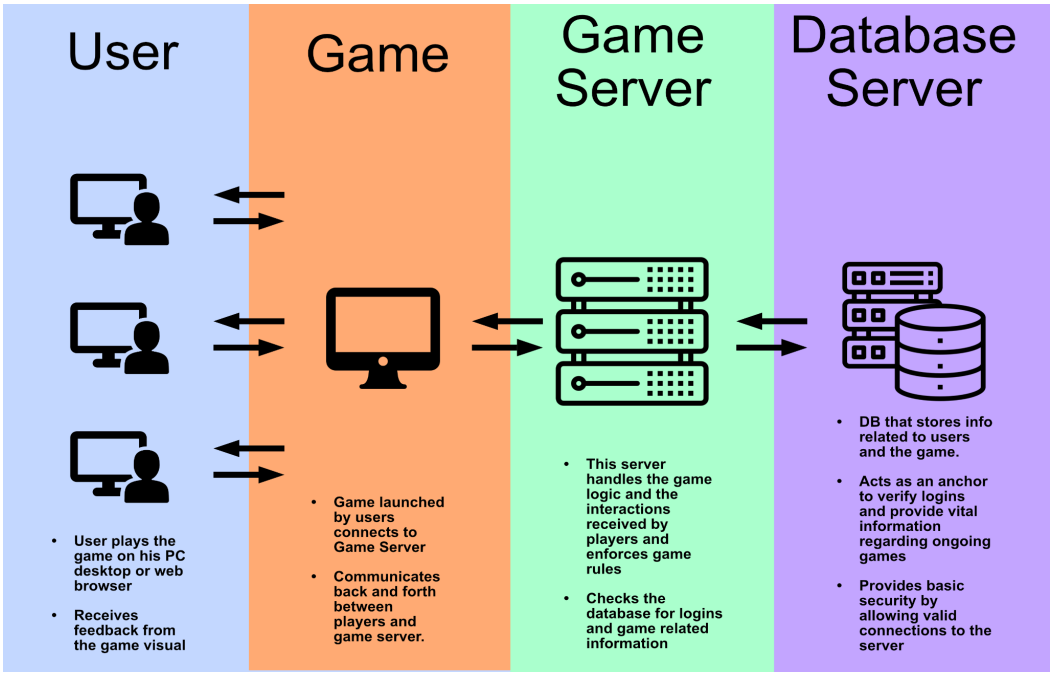
2.1.6 Community & Communication

Packaging & Distribution: The game will be properly packaged for desktop (standalone executables) and potentially for web deployment in future updates.

Project Promotion & Awareness:

- **Beta Testing**: A test version will be released to **students** for feedback.
- **Market Study & Industrialization Considerations:** Analyzing the potential for scalability, possible monetization models, and industry standards for future expansion.

2.2 Game Architecture



Here's a simple diagram of our game architecture

3. Technologies Used

The choice of technologies plays a crucial role in ensuring the efficiency, scalability, and a good final game product. Below, we provide a detailed justification for the tools and frameworks we have selected.

3.1 Game Client: Godot

Godot is a **powerful and lightweight open-source game engine** specifically designed for game development. It enables us to create an **optimized desktop game** as a foundation while maintaining flexibility for future expansions **to web and mobile platforms**.

Unlike other engines where **2D is treated as an extension of 3D**, Godot offers a **fully dedicated** 2D engine. This makes it an **ideal choice** for developing a card-based board game like "*6 qui prend !*", ensuring better performance and more precise rendering.

Cross-Platform Compatibility

- Seamless export to PC, mobile (Android/iOS), and web (HTML5/WebAssembly).
- Guarantees accessibility without requiring complex adaptations.

Optimized Performance

- Native game engine providing **better efficiency on desktop platforms** without relying on third-party compilation tools.
- Runs smoothly even on **low-end hardware**, making it ideal for an academic project.

Flexible and Modular Architecture

- Uses a **node-based system**, similar to object-oriented programming (OOP), allowing modular and **easier management** of game elements such as players, cards, and game logic.
- Encourages a **structured and scalable** codebase, simplifying future enhancements.

Versatile Scripting Languages
Supports multiple programming languages, including:

- **GDScript** (similar to Python) – perfect for quick development and scripting game logic. (**Here, we'll use GDscript for an interesting learning curve.**)
- **C# and C++** – offering additional flexibility and performance optimizations.

Integrated Multiplayer & Networking

- Built-in high-level networking API for real-time multiplayer interactions.
- Supports custom protocol implementation to ensure smooth communication between clients and the central server.
- Seamless WebSocket integration, making it fully compatible with our **Node.js** game server.

UI & Accessibility

- Intuitive drag-and-drop UI system for easy layout design.
- Fully supports responsive design, ensuring a smooth experience on both tablets (Samsung Tab S9) and PCs.

Database & Authentication Support

- Easily integrates with **SQLite, PostgreSQL, and MySQL** for storing user data, game progress, and scores.
- Built-in encryption and authentication mechanisms to secure player information and prevent cheating.

AI & Game Logic

- GDScript simplifies AI scripting, allowing easy implementation of computer-controlled opponents.
- Node system makes handling game rules and card interactions more structured and maintainable.

Development Efficiency & Team Collaboration

- Full Git integration, ensuring seamless collaboration within our team using **GitLab**.
- Fast iteration cycles and real-time debugging, thanks to built-in development tools.

To conclude, Godot Engine is the **optimal choice** for our multiplayer board game due to its **advanced 2D capabilities, strong networking features, ease of use, and cross-platform support**. It allows us to develop efficiently, iterate quickly, and deploy to multiple platforms, ensuring a **smooth gaming experience for all players**. Additionally, its **modular architecture and open-source nature** align perfectly with our project goals.

By leveraging **Godot's strengths**, we can **focus on gameplay, player interactions, and network stability** to deliver a high-quality final product.

3.2 Game Server: Node.js

For a **real-time multiplayer game**, server **responsiveness and scalability** are essential. **Node.js**, with its event-driven architecture, is perfectly suited for handling **concurrent player interactions**.

- **High performance & scalability:**
  - Node.js utilizes a **non-blocking, asynchronous** model, allowing thousands of simultaneous connections.
  - Ideal for real-time multiplayer games that require low latency and fast updates.
- **Optimized WebSocket communication:**
  - **Socket.io** enables seamless and efficient data exchange between the game client (Godot) and the server.
  - Ensures **synchronized game states**, preventing lags and inconsistencies in player interactions.
- **Robust and flexible ecosystem:**
  - **Express.js** simplifies API and routing management.
  - **Redis** allows **session storage** and rapid access to connected player data.
  - Seamless integration with **Docker** and cloud environments facilitates easy deployment and scaling.
- **Perfect compatibility with Godot:**
  - Godot (GDScript) communicates effortlessly with Node.js using WebSockets.
  - If using Godot's JavaScript bindings, **Socket.io** and **fetch()** provide direct interactions.

Conclusion: Node.js offers the best balance between **speed, scalability, and ease of integration**, making it the optimal choice for our real-time multiplayer game. The combination of **Godot + Node.js** provides a coherent, high-performance architecture for these 2D multiplayer gaming.

3.3 Database: MySQL

Managing player accounts, scores, and game sessions requires a **reliable and efficient** database system. MySQL is a well-established **relational database,** offering excellent performance for high-volume queries.

- **Fast, reliable, and easy to set up:**
  - MySQL is **lightweight**, requires minimal configuration, and scales efficiently.
  - Well-optimized for handling large amounts of **simple queries** in online applications.

- **Superior memory management for real-time applications:**
    - **In-memory tables** (MySQL Memory Engine) allow **ultra-fast data storage** in RAM instead of writing to disk.
    - Ideal for temporary data storage, such as **matchmaking queues** or **game session states**, where rapid insertion, selection, and deletion are required without permanent storage.
- **Additional Tools for Database Management:**
    - PHPMyAdmin (Web-based MySQL Management):
        - Provides an intuitive web interface for managing MySQL databases.
        - Simplifies database administration tasks such as creating tables, running queries, and managing user permissions.
        - Useful for debugging and monitoring game-related data, allowing quick edits and data visualizations.

Conclusion: MySQL provides an efficient, scalable, and well-supported database solution, while PHPMyAdmin enhances database management with a user-friendly interface for easier administration and monitoring.

3.4 Our first database view

The **database structure** for this card game project is designed to ensure **efficient data management, prevent redundancy, and maintain consistency**. It follows a **relational model** that organizes key entities into separate tables, ensuring a **modular and scalable architecture**.

3.4.1 Normalization & Data Integrity

To optimize performance and data integrity, the database is **normalized up to Third Normal Form (3NF)**, which:

- Eliminates redundant data.
- Minimizes update anomalies.
- Ensures that each table contains only relevant attributes, **improving consistency and scalability**.

This structured approach enhances the efficiency of **data retrieval**, ensuring that game-related operations run smoothly without inconsistencies.

3.4.2 Key Database Entities

The database is structured around **six core components**, each serving a specific role:

- **Player's Table**: Stores player authentication details (username, password, email) and profile information.
- **Player Stats Table**: Tracks game performance, including wins, losses, and highest scores.
- **Lobbies Table**: Records active game rooms, distinguishing between **public** and **private** lobbies (with an access code for private rooms).

- **Lobby Settings Table**: Stores customizable game settings such as:

    - Number of players (human & AI).
    - Round timer duration.
    - Card themes and display preferences.
- **Sessions Table**: Manages user logins and session tokens to ensure a seamless multiplayer experience.
- **Games Table**: Logs past matchmaking history, including game results and participating players.
- **Cards & Sounds Tables**:
    - **Cards Table**: Stores metadata for the 104 unique game cards.
    - **Sounds Table**: Manages in-game audio assets for enhanced player immersion.

3.4.3 Scalability & Future Expansions

This structured approach allows for **future expansions** while maintaining efficiency. The modularity ensures that additional features, such as **ranked matchmaking, seasonal leaderboards, or new game modes**, can be seamlessly integrated without disrupting the existing structure.

4. Role of the Project Manager

4.1 Team Management and Communication

- ➔ **Listen to and consider team feedback**, valuing the ideas and contributions of each member.
- ➔ **Organize and lead meetings when necessary**, ensuring they are relevant and efficient.
- ➔ **Closely monitor the project's progress**, identifying obstacles and proposing solutions.
- ➔ **Inform the team in advance of any major changes**, except in urgent cases significantly affecting development or deadlines.

4.2 Decision-Making and Leadership

- ➔ **Make decisions when necessary**, consulting at least the lead developers and, if possible, the whole team.
- ➔ **Balance workload distribution among team members**, avoiding overload or favoritism.
- ➔ **Ensure that all members always have an assigned task**, anticipating upcoming project stages.
- ➔ **Request progress updates on tasks**, while providing support when needed.

4.3 Technical and Organizational Support

- ➔ **Step in when needed for development**, helping to resolve technical issues.
- ➔ **Be available and responsive to questions and problems faced by the team.**
- ➔ **Foster a positive and collaborative work environment**, identifying and addressing tensions or conflicts within the group.

4.4 Ethics and Recognition

- ➔ **Avoid taking all the credit**, instead highlighting the team's collective efforts.
- ➔ **Recognize and value each member's contributions**, ensuring no ideas are dismissed.
- ➔ **Encourage a spirit of collaboration**, avoiding favoritism within the team.

4.5 Reporting and Monitoring

- ➔ **Write and submit weekly reports**, ensuring proper project tracking.
- ➔ **Adapt and adjust planning based on project progress**, ensuring milestones and deadlines are met.

5. Tasks, Teams & Gantt Chart

5.1.1 Team Organization

The development team is divided into specialized subteams, each responsible for a key aspect of the project:

- **Project Manager:** Kylian
- **Back-End (Server):** Omer, Lounas
- **Database:** Neila
- **Back-End (Client):** Mouctar, Elie, Darren (*Neila will join after completing the database implementation*)
- **Front-End (UI):** Darren

To ensure efficient coordination, **two lead developers** have been assigned to oversee critical aspects of the project.

5.1.2 Lead Developers and Their Responsibilities

Lead Server Developer – Omer

Responsibilities:

- Stay updated on the progress of the back-end team.
- Keep the project manager informed of any major updates or issues.
- Ensure the server operates correctly and adheres to the defined architecture.
- Organize and oversee communication between the server and the database.
- Work closely with the lead front-end developer to implement seamless client-server communication.
- Manage and maintain the Ubuntu servers.

Lead Client Developer – Neila

Responsibilities:

- Stay updated on the progress of the front-end team.
- Keep the project manager informed of developments.
- Supervise Elie, Mouctar, and Darren to ensure smooth rendering of cards and the game board.
- Work closely with the lead back-end developer to integrate client-server communication.
- Oversee the database server in collaboration with Omer.

5.2 Tasks and Gantt

To ensure smooth development and meet deadlines efficiently, the project has been divided into distinct phases, each with well-defined tasks and dedicated subteams. Our development plan prioritizes **specification, implementation, testing, and optimization**, ensuring a structured workflow.

5.2.1 Development Phases & Task Dependencies

- ➔ **Planning & Design (Weeks 1-3)**
  - Define game rules and project requirements.
  - Select technologies and create the Gantt chart.
  - Design database structure and client-server architecture.
- ➔ **Learning & Technology Onboarding (Weeks 1-4)**
  - Team members familiarize themselves with Godot, Node.js, MySQL, and Git workflow.
  - Specification phase ensures clear guidelines before development begins.
- ➔ **Backend Development (Weeks 3-9)**
  - Server setup (Ubuntu environment, API structure).
  - Core game logic, rule enforcement, and game state management.
  - Multiplayer lobby management, user authentication, and bot AI.
- ➔ **Database Development (Weeks 3-8)**
  - MySQL setup and database implementation.
  - Secure storage of game data, leaderboards, and user accounts.
- ➔ **Client Development (Weeks 4-10)**
  - Implement game logic, authentication, and game board mechanics.
  - Develop single-player and multiplayer functionalities.
- ➔ **Frontend Development (Weeks 5-11)**
  - UI design for menus, game board, animations, and user experience enhancements.
- ➔ **Web Adaptation (Weeks 9-12)**
  - Porting the desktop version to the web, ensuring compatibility.
- ➔ **Testing & Optimization (Weeks 10-13)**
  - Unit testing, debugging, performance improvements, security validation.
- ➔ **Finalization & Release (Weeks 12-13)**
  - Last adjustments, stability tests, and final presentation.

5.2.2 Milestones

Main Turning Points :

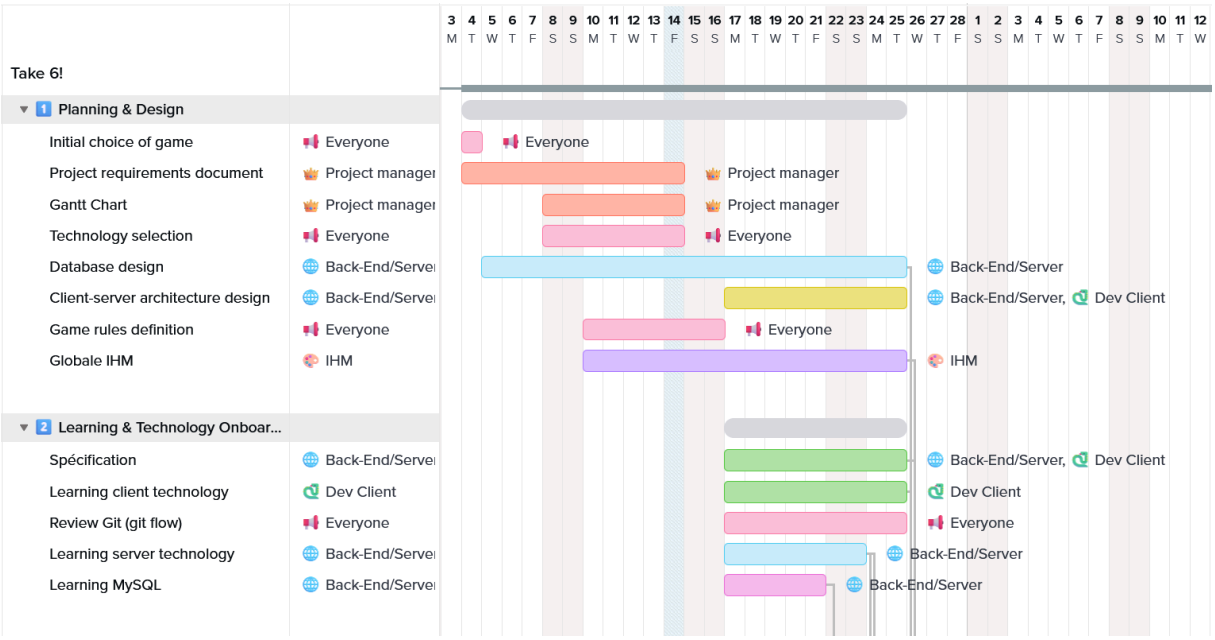
- **Alpha Version (Week 7):** Core game mechanics functional, basic networking.
- **Beta Version (Week 10):** Fully working multiplayer and UI integration.
- **Final Version (Week 13):** Optimized, tested, and ready for release.

The **Gantt chart** below provides a visual timeline of task distribution, ensuring efficient workflow management and a balanced workload for every team member.

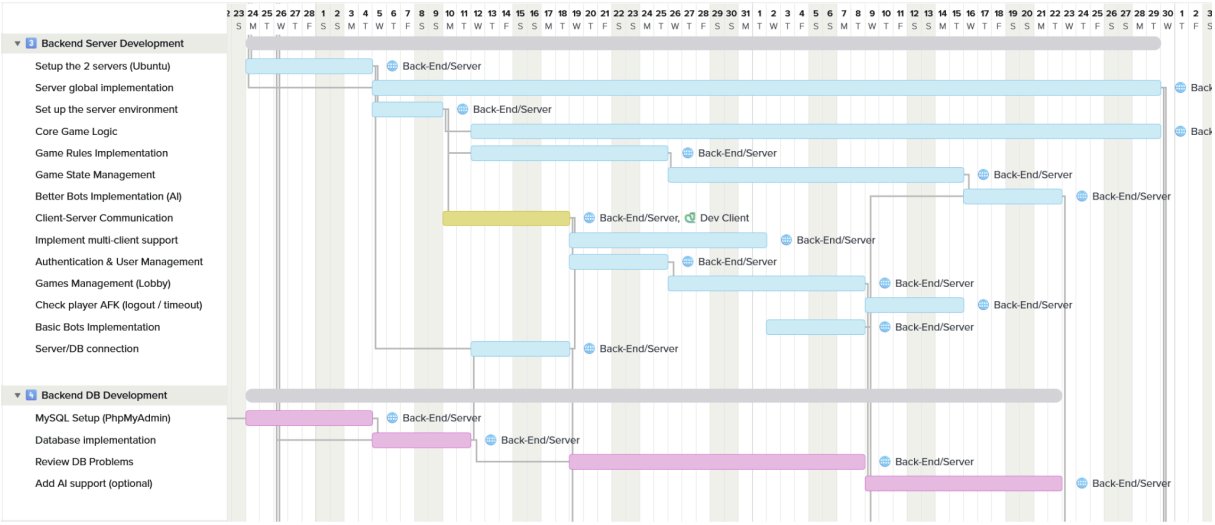
5.2.3 Gantt

Below are the **screenshots of our initial estimation and first version of the Gantt chart**. While this serves as a solid foundation for our development timeline, it is subject to **potential adjustments** as the project progresses.

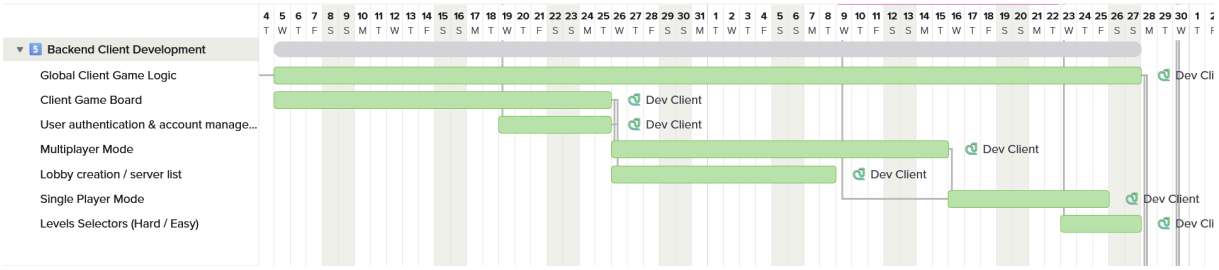
To ensure proper version tracking, **each updated version of the Gantt chart will be stored on the team's Seafile repository**. Additionally, a **PDF version of the Gantt chart is available**, providing a clearer and more detailed visualization compared to the screenshots.



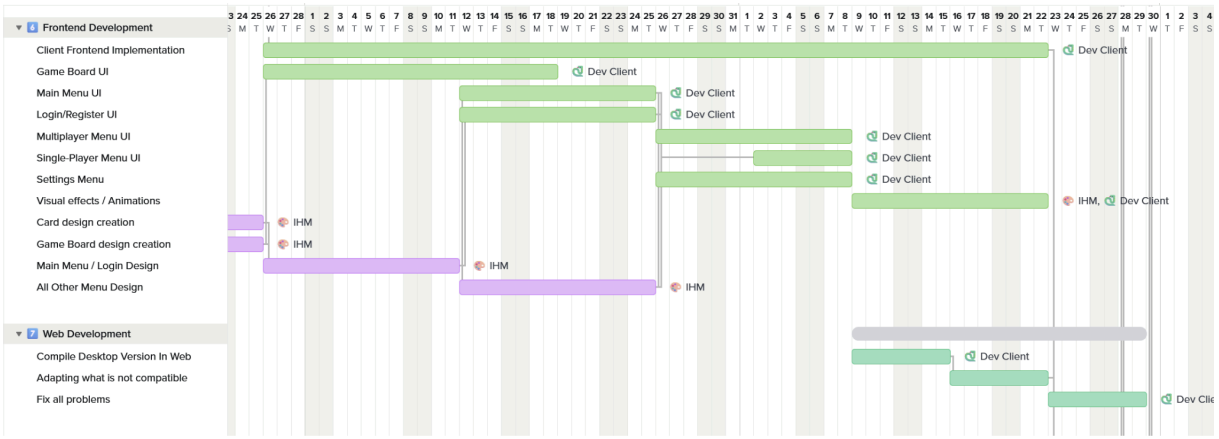
First and second group of tasks | TeamGantt



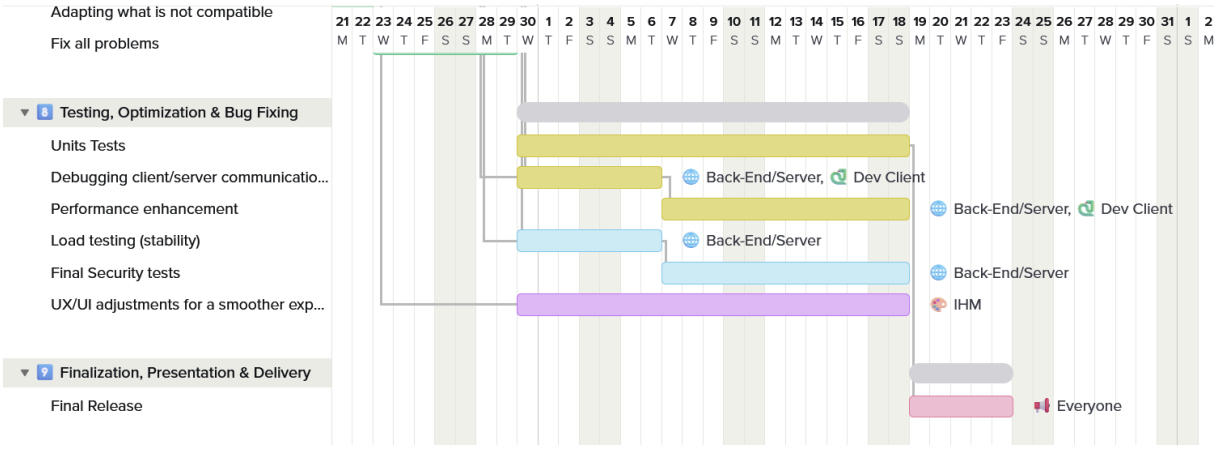
Third and fourth task groups | TeamGantt



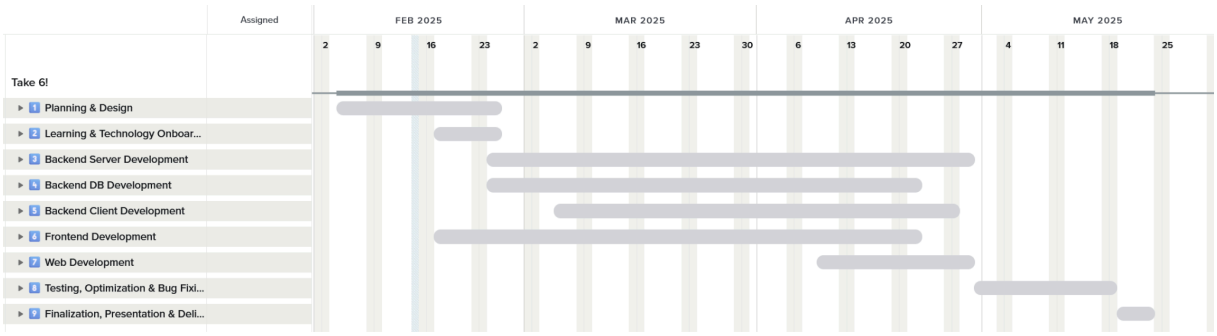
Fifth task group | TeamGantt



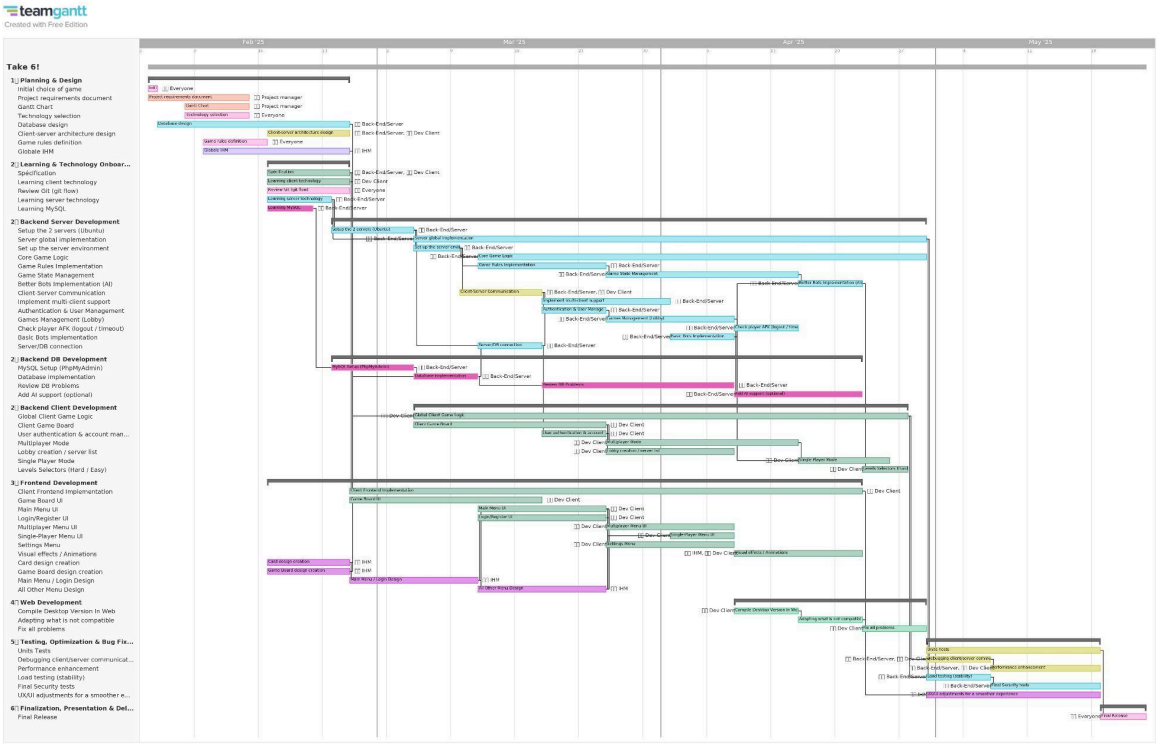
Sixth and seventh task groups | TeamGantt



Eighth and ninth task groups | TeamGantt



Global view with reduced tasks



Gantt chart overview PDF | TeamGantt

End