Phase II. Software Engineering

Arbi Maloku (Leader)

Adrian Ahmati

Albjori Dollani

Gledian Alimema

Hysen Allushi

Orgest Shira

1. Chose Development Method:

The best development method for a chess game application could be the **Agile methodology**. Here's why:

- **Iterative Development**: Agile allows for continuous feedback and improvement with each iteration, which is ideal for a complex game like chess where user experience and gameplay can be refined over time.
- **Flexibility**: Agile is highly adaptable to changes. As the development of a chess game might require adjustments to rules, AI difficulty levels, or interface design, Agile's flexibility ensures these changes can be incorporated smoothly.
- **User-Centric**: Agile focuses on user involvement and satisfaction. Since a chess application needs to cater to players of varying skill levels, regular user testing and feedback are crucial for success.
- **Efficient Project Management**: Agile's emphasis on collaboration and communication helps in managing the project efficiently, ensuring that the development team is always aligned with the project goals.

2. User requirements

End-users: These are the players who will interact with the game. Their primary interest is in the usability, functionality, and entertainment value of the app. They are crucial for providing feedback during the testing phases.

Clients: If the project is being developed for a company or an individual who has commissioned the work, they are the clients. Their interests lie in the marketability, profitability, and overall success of the application.

Developers: The software engineers and designers who build the application. Their role is to translate requirements into a functional product, and their interests include technical challenges, code quality, and innovation.

Project Managers: They oversee the project's progress and ensure that it stays on track, within budget, and meets the deadlines. Their interests are in successful project completion and stakeholder satisfaction.

Quality Assurance Testers: They are responsible for ensuring that the application is free of bugs and performs as expected. Their interests lie in the reliability and quality of the final product.

Marketing Team: Their role is to create awareness of the chess application, attract users, and position the product in the market. They are interested in the app's unique selling points and user engagement strategies.

Each stakeholder has a distinct role and set of interests, and their collaboration is vital for the project's success.

3. Functional requirements

A brief description:

What a chess game system should do:

- **Start the Game**: Set up the chessboard and pieces for two players to begin playing.
- **Move Pieces**: Allow players to move their pieces according to chess rules.
- **Check Rules**: Make sure all moves are legal and follow the official rules of chess.
- **Detect Checkmate**: Recognize when a king is in checkmate and end the game.
- **Offer Draws**: Give players the option to end the game in a draw if they agree.
- **Record Moves**: Keep a log of all the moves made during the game for review.
- **Undo Moves**: Let players take back their last move if they make a mistake.
- **Save Game**: Save the current game so players can continue later.
- **Play Against the Computer**: Offer an option to play against an AI opponent.
- **Adjust Difficulty**: Change the AI's skill level to match the player's ability.

These are the basic functions that make the chess game enjoyable and user-friendly.

4. Non-functional requirements

Non-functional requirements for a chess game system are about how the system operates, rather than what it does. Here are some key aspects:

- **Performance**: The game should run smoothly without any lag, so players don't have to wait long for their moves to be registered.

- **Usability**: It should be easy for players to understand how to make moves and navigate the game, regardless of their experience with technology.
- **Reliability**: The game should not crash or have errors, ensuring players can always finish their games without issues.
- **Compatibility**: The game should work well on different devices and operating systems, so everyone can play.
- **Security**: Players' personal information and game data should be protected from unauthorized access.
- **Scalability**: The system should be able to handle a growing number of users without performance drops.

5. Application specifications

a. Architecture: The architecture for the chess game application would be a **Client-Server model** where the client is the user interface and the server handles game logic, AI, and database interactions. The server would also manage user accounts and store game states.

b. Database Model: The database has tables for **Users**, **Games**, **Moves**, and **Chat**. Relationships include:

- Users to Games (one-to-many): A user can have multiple games.
- Games to Moves (one-to-many): A game can have a sequence of moves.
- Users to Chat (many-to-many): Users can interact with each other via chat. Constraints might include data validation rules to ensure that moves are legal according to chess rules.

c. Technologies Used:

- **Programming Language**: JavaScript for its ubiquity and support across browsers.
- **Framework**: React for the UI to create a responsive and dynamic user experience.
- **Backend**: Node.js for server-side operations due to its performance and scalability.
- **Database**: MongoDB for a flexible, schema-less structure that can handle diverse data types.
- **AI Engine**: Stockfish or a custom-built AI for game logic and providing a challenging opponent.

d. User Interface Design: The UI should be clean and minimalistic, with a focus on the chessboard. It features easy-to-access controls for game options like undo, new game, and save. The design should be responsive to fit various screen sizes and devices.

e. Security Measures:

- **Encryption**: Use HTTPS to secure data in transit.
- **Authentication**: Implement OAuth for secure logins via social media accounts.
- **Data Protection**: Hash user passwords in the database and use access tokens for session management.