RhombiChess: A Strategic Online Chess Variant – Development Plan

Table of Contents

1: Controls/ Versions

Date	Version	Developer(s)	Change
October 27th, 2023	0	Entire Team	Initial Plan

2: Contributions Table

Developer	Section(s)
Farzan Yazdanjou	#10, Final Review / Edits / Compile
Monica Bazina-Grolinger	#3, #4, #8
Nida Nasir	#5, #6
Anant Prakash	#9, #11
Philip Lee	#7

3: Team Meeting Plan

Our primary meeting is set for every Friday during our designated capstone class time. However, if our capstone class is occurring, we will reschedule our meeting to a later time on the same day (Friday). In addition to our Friday meetings, we will also meet every Tuesday at 9pm. Meetings with our supervisor, Paul Rapoport, will be scheduled as needed or requested.

Internal meetings will be typically conducted on Discord (further details about our use of Discord can be found in the "Team Communication Plan" section). However, if all team members would like to meet in-person, that will also be arranged. Team meetings are expected to last about an hour unless otherwise specified. Team members are expected to join prepared and actively contribute to discussions.

4: Team Communication Plan

During our collaboration period, team members are expected to respond within a 3-hour timeframe within our designated communication window. This window is defined as the hours between 9 a.m. and 9 p.m., occurring daily. Outside of these specified hours, group members are not obligated to respond.

Discord will be the primary platform for team communications. All members are part of a dedicated Discord server. Within this server general questions and online discussions will take place in the "General" text channel, while scheduled meetings will be held in the "General" voice channel.

Our **Microsoft Teams** chat, which includes the team and Paul Rapoport (our supervisor), will be the primary method of communication with our supervisor. The team will provide Paul Rapoport with milestone updates via this chat. The team will use Microsoft Teams to arrange a suitable time slot to communicate with the professor during "Instructor Support Sessions". To communicate with our assigned TA, group members will utilize the **Microsoft Teams** "Group 12" private channel.

5: Team Member Roles

- Project Coordinator: Farzan Yazdanjou

5.1: Functional Requirements & Roles

Front-End Requirements & Roles	
User Management	
Users can register for an account	Farzan, Monica
Users can use their existing credentials to log-in	Farzan, Monica
Game Interaction	
Users can start a new game	Entire Team

Allow users to click on a piece and make a legal move	Anant, Nida
Display message if a move is not valid	
Display message if a check/checkmate occurs	Farzan
Prevent further gameplay if a checkmate occurs	
Allow users to choose their opponent (themselves or online opponent)	Anant, Nida, Phillip
Display board (190 rhombuses) with the pieces on each side (16 unique, 47 total on each side with their designated design)	Farzan, Monica
Provide an option for choosing their side – black or white. This is random by default.	Implicit
Highlight selected piece and rhombuses that are legal moves for that piece	Farzan, Monica
Create a responsive UI that adapts to screen size	Farzan, Monica
Timer	
Allow users to set timer duration for real-time play	Anant
Countdown individual timer based on active player	Anant
Declare loss for player if timer ends	Nida
Allow for untimed games (default option)	Nida
Captured Pieces	
Highlight most recently captured piece off the board	Phillip
Display all captured pieces for each player off the board	Phillip
Messaging	
Allow users to send messages to each other during the game	Farzan
Sound and Localization	
Play audio based on game events (i.e. check, checkmate, chat ping etc.)	Monica

Back-End Requirements & Roles	
Game Logic	
Implement logic for validating moves detecting check/checkmate	Anant, Nida, Phillip
User and Game Management	
Manage user authentication / registration	Farzan, Monica
Store game state	Nida
Handle online match making	Phillip

Implement API for frontend to communicate with the backend	Farzan, Monica
Ensure data integrity	Nida
Save user preferences	Phillip

5.2: Non-functional Requirements & Roles

Appearance Requirements & Roles		**Style Requirements & Roles**	
Board Design	Monica	Color Scheme	Farzan, Monica
Piece Design	Farzan	Typography	Phillip
User Interface	Monica	Animations	Nida
Responsive Design	Farzan	Consistency & Accessibility	Farzan

Performance Requirements & Roles			
Speed & Latency	Anant	Robustness	Phillip
Safety-Critical Requirements	Anant	Capacity	Nida
Precision	Anant	Scalability	Nida

6: Workflow Plan

6.1: GitLab Management

We will be using GitLab as our Git-based version control system. We will create development branches for specific features, bug fixes, or improvements and they will all be created from the most up-to-date branch. The feature branches will be named so it is clear to all developers exactly what feature was created in that specific branch, for example, ‘feature/new-chess-piece’. Bug fix branches will also be named as such, for example, ‘bug/soldier-chess-piece-fix’. The changes will be committed to the respective branches and each PR (pull request) will need at least two approving reviewers before it can be merged. If there are any conflicts or additional changes needed during the review, then the developer assigned to the task can make the necessary changes and push to the PR.

6.2: Sprint Planning

We will use the issue tracking system provided by GitLab to create tickets and keep track of progress. The issues will have various types of tasks such as features, bug or nice-to-have improvements that will be assigned priorities from P0 to P3. Each issue will be moved to the correct status when needed i.e., open or closed. The project will start with a bunch of user stories which reflect the requirements of the primary stakeholders, specifically the supervisor. The user stories will be assigned priority levels from P0 to P3 based on the functional requirements we created and their prioritization levels. The stories that are essential to the essence of the game will be given higher priority. The sprint lengths are determined by the epics assigned to each sprint which is outlined in the project planning section.

6.3: Storing Data

The game data such as the current board state, player moves, and captured pieces will all be stored in a database that will be hosted on Neon. All user account information will also be stored in the database. As our project does not involve machine learning, we do not require extensive data storage or training models.

6.4: Tools

We will be using Flask for the backend which primarily uses Python, so we will use a virtual environment and the package manager pip. In terms of the frontend, we will use React which would require the use of the package manager 'npm'. Since we will be using Neon to host our database, we will use PostgreSQL. In terms of testing frameworks, we will use Jest and PyTest.

7: Proof of concept plan

The primary challenge for our project is achieving a functional game. None of us have experience building chess or any 2D browser games, especially real-time web applications. We're unsure about the movement of pieces and defining unique move patterns for each piece type. While a standard chess grid makes movement logic straightforward, our rhombus-based board complicates this, demanding deeper game logic understanding. Given our tight deadline, this is a significant hurdle. For our proof-of-concept demo, we'll develop a web application showcasing the board and piece designs and their basic movements without specific constraints. Additionally, we aim to demonstrate real-time updates with a live demo of two players making updates to the same board (likely done locally). Along with user authentication, such a Proof of Concept will reflect our ability to succeed in this project. Our development schedule to implement the Proof of Concept is outlined in Section 10.

8: Technology

We have chosen Python programming language with Flask framework for the backend of our system. On the front end, we will be using JavaScript with NextJS and the React framework. As it is standard practice for software development, we will be implementing unit testing. Unit testing detects early bugs and improves the quality of the code, and documentation, among other benefits. To achieve this, we will use PyTest for the backend and Jest for the frontend.

We have the option to either deploy our backend on the server offered by our supervisor or choose a hosting service – this will be decided as we approach deployment. Lastly, we will use Neon for database hosting, which offers a free serverless Postgres plan.

9: Coding Standards

9.1: Code Review

Upon completing an issue on the project's GitLab, assignees must create a merge request before changes are merged into the main branch. Through this merge request, the team will review the code changes to ensure they make sense and meet the coding standards outlined below.

9.2: Backend Coding Standards

Category	Standard
Code style	Code must conform to **PEP8** style guide
Function and variable names	Descriptive and clear naming for variables and classes
Documentation	Code must include docstrings for major functions following the **Google Python Docstrings** style guide
Error Handling	Errors must be logged to a file for future access in the event of a failure
Testing	**pytest** will be used to create unit tests for the code
Dependency Management	**Pip** must be used in conjunction with a `requirements.txt` file to maintain dependencies
Version Control	Branches must have follow `name/feature` convention, along with descriptive comments for commits

9.3: Frontend Coding Standards

Category	Standard
Code Style	Code must conform to **Airbnb React/JSX** style guide in addition to the linting rules of the NextJS project.
Documentation	Code will be documented using the **JSDoc** format where necessary.
Error Handling	Errors must be handled gracefully, use error boundaries as needed.
Testing	Unit tests will be created and maintained using **Jest**.
Dependency Management	**NPM** must be used in conjunction with a `package.json` file.

10: Project Scheduling

To reliably develop the project and deliver for our November Demo, we have strategically split up the project into 5 Epics which are outlined below. This split allows us to focus on the key aspects of the project that produce a minimal viable product that showcases our vision. The project lifecycle will follow the deliverable requirements for the course – a demo presentation in November and the entire completed project by April.

Epic A: Setting Up the Project

Task 1.1: Create a project repository on GitLab and plan a Git development process.

Task 1.2: Organize the basic file and code structure and set up NextJS for the frontend and Flask for the backend.

Task 1.3: Connect the frontend and back end using an API, ensuring seamless communication between the two.

Task 1.4: Set up the various required routes on NextJS.

Task 1.5: Set up a Postgres database connection and implement a chessboard model that allows us to store game states.

Epic B: Creating the Board and Pieces

Task 2.1: Design the general theme/branding for the application.

Task 2.2: Implement the board and various pieces on the frontend.

Task 2.3: Allow the user to move pieces around on the board using their mouse.

Task 2.4: Ensure pieces are restricted to moving within the board's cells.

Epic C: Enabling Online Play

Task 3.1: Set up sessions using Flask to allow us to determine unique players.

Task 3.2: Implement a lobby system for players to create and join games using a lobby code.

Task 3.3: Establish WebSocket communication for real-time play.

Task 3.4: Ensure both users can move pieces on the board, taking turns, with real-time updates – updating our database with the changing game state as mentioned previously in Epic A.

Task 3.5: Add a clock to keep track of each player's remaining time (this is a P3, by default, there is no time constraint).

Epic D: Implementing Game Logic

As we move into the second semester and begin work on the final game, we will expand on this epic and split up the work further. Until then, we can keep it somewhat broad.

Task 4.1: Define and code how each piece should move on the frontend and backend.

Task 4.2: Display highlighted cells when a piece is selected, showing the user possible moves.

Task 4.3: Keep track of the game status, including checks, checkmates, and taken pieces.

Task 4.4: Display taken pieces on the side of the chessboard.

Task 4.3: Implement logic to determine game-end scenarios.

Epic E: Adding Extra Features

Task 5.1: Implement a chat feature to allow players to communicate during the game.

Gantt Chart

The following is a Gantt Chart representing our Project's Scheduling. It has been broken up to be easily viewable in a PDF file.

	Week 1 (Oct. 16)	Week 2 (Oct. 23)	Week 3 (Oct. 30)	Week 4 (Nov. 6)	Week 5 (Nov. 13)	Week 6 (Nov. 20)	Week 7 (Nov. 27)	Week 8 (Dec. 4)
Epic A: Setting Up	🟥							
Epic B: The Board / Pieces		🟥	🟥					
Epic C: Online Multiplayer			🟥	🟥	🟥			
Documentation and Polish for Proof of Concept Demo					🟥	🟥		

	Week 9 (Dec. 11)	Week 10 (Dec. 18)	Week 11 (Dec. 25)	Week 12 (Jan. 1)				
Casual Development / Improvement of the Application during Winter Break	🟥	🟥	🟥	🟥				

	Week 13 (Jan. 8)	Week 14 (Jan. 15)	Week 15 (Jan. 22)	Week 16 (Jan. 29)	Week 17 (Feb. 5)	Week 18 (Feb. 12)	Week 19 (Feb. 19)	Week 20 (Feb. 26)
Design Document Deliverable	🟥	🟥	🟥					
Epic D: Full Game Logic		🟥	🟥	🟥	🟥	🟥	🟥	🟥
Combining the Game Logic with Online Gameplay							🟥	🟥

	Week 21 (Mar. 4)	Week 22 (Mar. 11)	Week 23 (Mar. 18)	Week 24 (Mar. 25)	Week 25 (Feb. 5)	Week 26 (Feb. 12)	Week 27 (Feb. 19)	Week 28 (Feb. 26)
Epic D: Full Game Logic	🟥							
Combining the Game Logic with Online Gameplay	🟥							
Epic E: Extra Features	🟥	🟥						
Documentation and Polish for Final Demo & Submission	🟥	🟥	🟥					

11: Signatures

By signing below, you agree to follow the development plan outlined in this document and any required changes for the development and success of this project.

Name	**Signature**
Anant Prakash	[illegible]
Monica Bazina-Grolinger	[illegible]
Farzan Yazdanjou	Farzan Yaz.
Nida Nasir	[illegible]
Philip Lee	[illegible]