RhombiChess: A Strategic Online Chess Variant – SRS Document

Table of Contents

1: Controls / Versions 1
2: Contributions Table 2
3: Personnel and Roles 2
4: Naming Conventions and Terminology 2
5: Purpose of the Project 2
6: Stakeholders 3
6.1 Primary Stakeholders 3
6.2 Secondary Stakeholders 3
6.3 Tertiary Stakeholders 3
7: Mandated Constraints 4
7.1: Solution Constraints 4
7.2: Partner Applications & Off-the-Shelf Software 4
7.3: Schedule Constraints 4
8: The Scope of the Product 4
8.1 Product Boundary 4
8.2 Product Use Case Outline 5
9: Functional Requirements 6
10: Data and Metrics 8
11: Relevant Facts and Assumptions 8
12: Non-Functional Requirements 10
12.1 Look and Feel Requirements 10
12.2: Performance Requirements 11
12.3: Maintainability and Support Requirements 12
12.4: Security and Privacy Requirements 13
12.5: Legal Requirements 13
13: Risks and Issues Predicted 14
14: Tasks: Project Planning 14
15: Anticipated Costs 14

1: Controls / Versions

Date	Version	Developer(s)	Change
October 18, 2023	1.0	Entire Team	Initial SRS Content

2: Contributions Table

Developer	Section(s)
Farzan Yazdanjou	14, 12.1, 12.2, 15
Monica B-G	4, 5, 6
Nida Nasir	7, 8, 11
Anant Prakash	9, 12.3, 12.4
Philip Lee	10, 12.5

3: Personnel and Roles

Roles	Personnel
General Point of Contact	Nida Nasir
Compiling & Submissions	Farzan Yazdanjou + Monica B-G
Final Reviews	All team members
Front-end Development	Farzan Yazdanjou + Monica B-G
Back-end Development	Anant Prakash + Nida Nasir + Philip Lee
Web App Visuals	All team members will contribute to the website design
Meeting Notes	Rotation between all team members

4: Naming Conventions and Terminology

PIPEDA: (Personal Information Protection and Electronic Document Act) is a privacy law in Canada. This law governs how organization handle personal information. It sets rules for the collection, use and disclosure of this information.

FIDE: (Fédération Internationale des Échecs) is the governing body for the sport of chess. Responsibilities would include enforcing rules and standards for chess competitions, tournaments, and titles.

Epic: In the context of project management, an "Epic" is a term to denote a significant amount of work that can be decomposed into smaller tasks.

Minimal Viable Product (MVP): represents the smallest version of product that can be released to market while meeting core deliverables.

5: Purpose of the Project

The purpose of our chess variant project is to create an innovative, engaging, and challenging experience by introducing numerous gameplay elements that provide a new twist on the traditional game of chess.

6: Stakeholders

Stakeholder		Persona	Categorization
Client	Supervisor	Alex	Primary
Users/Players	New to chess		Primary
	Not new to chess	Avery	Primary
Designers/Developers/Testers			Secondary
Chess Community			Tertiary

6.1 Primary Stakeholders

Client: Our client provides us with a unique set of rules they have defined for this project. They also oversee the whole project, provide resources, and ensure the project adheres to the initial goals and objectives. Thus, they have a vested interest in the project's success and a great deal of influence on the project's requirements.

Alex (Supervisor): Alex is a 43-year-old elementary school teacher. They have been involved in the chess world for several years, participating in tournaments, and working as a chess instructor. Recently, Alex developed a new and exciting variant of the classic chess game. This variant incorporates innovative rules and a unique board setup. To bring this idea to fruition, Alex has proposed the concept to a team of software engineers and plans to oversee their creation of the game.

Users/Players: Users/players are the individuals the variant is designed for. Their preferences, needs and feedback will shape the game's design and development since they are the ones who interact with the project. Therefore, users/players play an influential role in the project's requirements.

Avery (Not new to chess): Avery, a 32-year-old physiotherapist, enjoys playing chess in their free time. They have been playing chess since childhood, having learned the game from their father. Over the years, they have learned various chess strategies to gain an advantage over their opponents. Avery loves learning new and interesting strategies to enhance their chess capability.

6.2 Secondary Stakeholders

Designers/Developers/Testers: Are involved in the creation of the game. They are essential to the project execution but do not have the same level of control as the direct (primary) stakeholders. These individuals are more concerned with the technical implementation.

6.3 Tertiary Stakeholders

Variant Chess Community: While not directly involved in the project, their interest and enthusiasm about it is an integral part of its potential success. The variant chess community is affected by the success or failure of this game.

7: Mandated Constraints

7.1: Solution Constraints

The game must follow the rhombus-based chessboard design which is essential for the development of the logic of the game and how the pieces interact with the board. It must include sixteen pieces on each side which will be a total of 47 pieces. Each piece has its own distinctive rules and must be implemented accurately to adhere to the requirements set by the supervisor. The application must enforce the standard chess objective of checkmate. The users must also have access to the Internet to be able to play with other players. This application must be accessible by a variety of devices and offer multi-player capabilities – as such, we constrain that it must be a web application to easily allow a variety of users to play our game in real time. This will streamline our development process as we are under time constraints and need an all-in-one solution.

7.2: Partner Applications & Off-the-Shelf Software

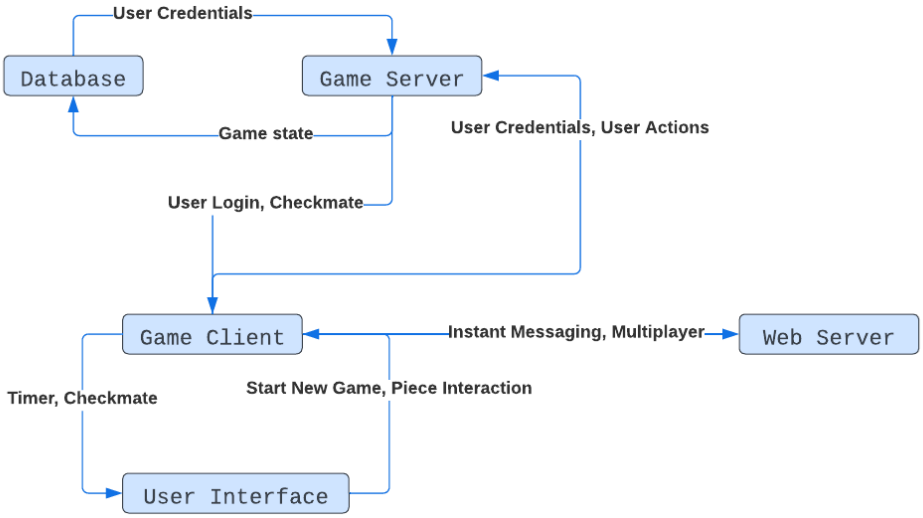
RhombiChess does not have specific integration requirements. We may potentially integrate it with the website https://www.chessvariants.com where other chess variants are hosted for players to enjoy collectively, but this is optional. This application will be built independent of any off-the-shelf software solutions, as it has its own distinctive set of rules, pieces, and constraints.

7.3: Schedule Constraints

This project is expected to adhere to the deadlines set by the professor and is set to be completed within the 8-month timeframe that the course runs for. Furthermore, we are expected to provide a proof-of-concept demonstration by mid-November.

8: The Scope of the Product

8.1 Product Boundary



8.2 Product Use Case Outline

The various use cases outlined below will impact the same three stakeholders: the User, the Supervisor, and the Developers. For each action, as we are providing real-time updates, the Actors in play will be the User(s), the Database for game state updates, the Server for webSocket updates, and our APIs for all actions.

User Registration: The user enters a username and password.

- Preconditions: The user does not already have an account.
- Outcome: The new user and credentials

User Login: The user enters their existing credentials.

- Preconditions: The user is not already logged into another account.
- Outcome: If the credentials have a direct match in the database, then the user is logged into their account. Otherwise, an error message will be displayed.

Start New Game: The user clicks on the button to start a new game and picks the game timer options, as well as if they want to play with themselves or another online user.

- Preconditions: The user must be logged in.
- Outcome: If the user is logged in, they will be able to start a new game. Otherwise, an error message will be displayed to register for an account.

Move Piece: The user selects a piece and moves it to one of the legal, highlighted moves.

- Preconditions: The game must be in progress, and it must be the user's turn.
- Outcome: If it is the users' turn to make a move, the game state will be updated with the new piece position and the user will be able to see their chess piece move. If it is not, the user should not be able to move the piece.

Capture Piece: The user moves a piece to a position occupied by the opponent's piece.

- Preconditions: The game must be in progress; the opponent's piece should be in the position being targeted and the move should be legal.
- Outcome: If the move is legal, the game is in progress, and the opponent's piece is occupying the position, the piece should be captured and removed from the board. If not, the piece should not be removed.

Timer: The game starts, and the timer starts.

- Preconditions: The user must set their timer preferences when the game starts, and the game must be in progress.
- Outcome: If the user has set their timer preferences and the game is in progress the timer starts and the timer state is saved for each player's turn. The user should be able to see the timer displayed, reflecting the accurate time left. If not, the timer is not displayed as the game is not started.

Checkmate: The user makes a move that qualifies as checkmate and moves a piece.

- Preconditions: The game must be in progress and a move is made to ap position where checkmate occurs.
- Outcome: The game state should reflect the instance of checkmate by checking according to the rules and legal moves if a checkmate can occur. If this is the case, the user should be able to make the move to win the game, and the game should display a game-over message. If not, the game continues.

Instant Messaging: The user clicks on the messaging tab and types in a text message.

- Preconditions: The game must be in progress and in multi-player mode.
- Outcome: If the game is in progress and in multi-player mode, the opponent should receive the instant message.

9: Functional Requirements

Front-End Functional Requirements (React)	
User Management	
Users can register for an account	P0
Users can use their existing credentials to log-in	
Users can reset their password	P3
Game Interaction	
Users can start a new game	P0
Allow users to click on a piece and make a legal move	
Player must confirm or retract move before board state changes	
Display message if a move is not valid	
Display message if a check/checkmate occurs	
Prevent further gameplay if a checkmate occurs	
Allow users to choose their opponent (themselves or online opponent)	
Display board (190 rhombuses) with the pieces on each side (16 unique, 47 total on each side with their designated design)	
Provide an option for choosing their side – black or white. This is random by default.	P1
Highlight selected piece and rhombuses that are legal moves for that piece	
Allow users to save, recall and replay a game	P3
Create a responsive UI that adapts to screen size	P2
Flip board to active player's side	
Timer	

Allow users to set timer duration for real-time play	P0
Countdown individual timer based on active player	
Declare loss for player if timer ends	
Allow for untimed games (default option)	
Captured Pieces	
Highlight most recently captured piece off the board	P0
Display all captured pieces for each player off the board	P1
Messaging	
Allow users to send messages to each other during the game	P1
Sound and Localization	
Play sound based on events that occur during the game (i.e. check, chat ping etc.)	P3
Have game text in various languages for supporting different player groups – unlikely.	

Back-End Functional Requirements (Python/Flask)	
Game Logic	
Implement game rules and logic for validating moves and to detect check/checkmate	P0
User and Game Management	
Manage user authentication / registration	P0
Store game state	
Handle online match making	
Implement API for frontend to communicate with the backend	
Ensure data integrity	
Save user preferences	P2
Timer	
Save timer state for each turn and player	P0
Advanced Game Features	
Implement AI opponent to allow single-player mode	P3
Add features such as move suggestions and statistics	

API Functional Requirements: Endpoints	
Create an endpoint for user registration and authentication	P0
Create an endpoint for game management (i.e., game state, new game, game history etc.)	

Create an endpoint for player actions	
Create an endpoint for retrieving timer state	
Create an endpoint for chat messages	
Create an endpoint for sound and notification	P3
Create an endpoint to gather user data for statistics and insights	
Add support for social media integration (external chess websites, etc.)	
If Saving/Loading of games is not implemented, ensure that game data is periodically wiped from the database.	P1

10: Data and Metrics

Although our project does not require data to operate and train on, data and metrics are nonetheless important for it. Collecting reliable data and metrics can help us make informed decisions and better plan our development. By understanding our players' user behaviour, we can enhance the game's user experience. Additionally, we're exploring the potential of using datasets to create an AI opponent for our game. While this is an exciting feature to consider, it's quite ambitious. Given the complexity involved, it might not make it into the final product. For the general game, we're focusing on three main areas:

- **Software Reliability**: We plan to continuously develop front-end and back-end tests to measure our program's reliability and robustness. By tracking our software's performance, we can understand if our development is improving or degrading the game, which helps guide our programming decisions.
- **Game Response Time**: We know how frustrating delays can be in online gaming, especially for a strategic, time-based game such as chess. As a real-time game, speedy response time is essential. We'll monitor this metric, striving for the lowest feasible latency to ensure smooth online gameplay.
- **Stakeholder Feedback**: As our product is a game, user feedback is essential in making informed decisions. We need to understand how players and stakeholders feel about the game during development so we can make necessary changes before it's too late. We'll gather feedback, focusing on overall satisfaction and specific pain points.

11: Relevant Facts and Assumptions

Our chess variant has plenty of new pieces and accompanying rules:

White moves first. Players must checkmate the opposing King to win. There is no castling or en-passant capturing because of the way that the regular capturing works. When a White Pawn or Soldier reaches row 30 or beyond, or a Black Pawn or Soldier reaches row 8 or beyond, it must be promoted to any piece except those or a King. There are 16 unique, 47 total pieces on each side. The quantity (#) refers to the number of that piece per side. The pieces are distributed as so:

Piece	#	Move
Rook	2	Moves in 1 of 4 directions, 1 or more steps in a straight line across rhombuses with a common side. It does not leap.
Machine	2	Moves 1 or 2 rhombuses like a Rook but may leap to the second rhombus.
Bishop	3	Moves in 1 of 4 directions, 1 or more steps in a straight line along rhombuses with a common vertex and colour. It does not leap.
Elephant	3	Moves 1 or 2 rhombuses like a Bishop but may leap to the second rhombus. All 3 elephants may escape the initial setup on their first move.
Jester	3	Moves in 1 of 4 directions, 1 or more steps in a straight line across a vertex of its rhombus. It does not leap.
Dog	3	Moves 1 or 2 rhombuses like a Jester but may leap to the second rhombus. The move across a wide-angle vertex is to 1 rhombus only. All 3 dogs may escape the initial setup on their first move.
Queen	1	Moves in 1 of 12 directions, 1 or more rhombuses in a straight line like a Rook, Bishop, or Jester. It does not leap.
Mammoth	1	Moves 1 or 2 rhombuses like a Queen but may leap to the second rhombus. The move across a wide-angle vertex is to 1 rhombus only.
Cat	3	Moves 2 rhombuses (not 1) like a Queen but may leap. The move across a wide-angle vertex is to 1 rhombus only. All 3 Cats may escape the initial setup on their first move.
Hawk	1	Moves 2 or 3 rhombuses (not 1) like a Queen but may leap to them. The move across a wide-angle vertex is to 2 rhombuses only. The Hawk may escape the initial setup on its first move.
Shield	3	Moves to any rhombus in its 2 regular hexagons. A move in a Rook's direction may be a leap to the 3rd rhombus away. The left and right Shields may escape the initial setup on their first move.
Knight	2	Moves (and may leap) 2 rhombuses like a Rook followed by 1 rhombus outwards (at a 1/3 turn, i.e., 120°), or 1 rhombus followed by 2 likewise. A Knight's move always changes the colour of its rhombus. Both Knights may escape the initial setup on their first move.
King	1	Moves in 1 of 10 directions but only 1 step in any direction and omitting the Queen's move across a wide-angle vertex. It thus moves 1 step in either of its 2 snowflakes.
Prince	2	Moves exactly like a King but is not subject to check and may be captured.
Pawn	12	There are two types of Pawn. A Pawn always moves to the same type of rhombus and captures going to the other type by step. In their initial positions the Pawns are protected by other Pawns, Soldiers, and pieces farther back.

		Pawn 1	Pawn 2
		On a vertical rhombus. Moves 1 step forward like a Jester and captures 1 step forward onto the nearest non-vertical rhombuses, contiguous by side or vertex.	On a non-vertical rhombus. Moves 1 step forward like a Rook, or 2 such steps if there is no intervening piece. It captures 1 step forward onto the nearest vertical rhombuses, contiguous by side or vertex. (This pawn does not capture in the direction of a Jester or Dog.)
Soldier	5	Soldier Type 1	Soldier Type 2
		On a vertical rhombus. Moves and captures like a Pawn on a similar rhombus. Without capturing, it also moves to the rhombuses of capture.	On a non-vertical rhombus. Moves and captures like a Pawn on a similar rhombus. Without capturing, it also moves to the rhombuses of capture. (This soldier does not capture in the direction of a Jester or Dog.)

12: Non-Functional Requirements

12.1 Look and Feel Requirements

Creating a digital game platform is not just about functionality, but also about the experience we provide to our users. The look and feel of our chess variant platform should be engaging, intuitive, and visually appealing, ensuring that users enjoy every aspect of their interaction with our application.

12.1.1: Appearance Requirements

Board Design	The board should have clearly defined cells using a rhombus shape, as per our game variant.
	Distinct colors or shading should be used to differentiate between cells, ensuring clarity and visual appeal.
Piece Design	Each piece should be distinct and easily recognizable.
	Use a consistent color scheme for each side (e.g., black and white).
User Interface (UI)	UI elements such as buttons, panels, and menus should be consistently styled and positioned. Icons and text should be clear and readable.
	Feedback mechanisms (like pop-ups or notifications) should provide relevant information without being intrusive.
	The layout of the UI should be familiar – the board to be the center of attention, the taken pieces on the side of the board, the chat feature to the side,

	etc. Keeping consistent with what users expect from previous experiences will ensure a memorable UI.
Responsive Design	In the context of a web application, the game should be visually appealing and functional across various devices, including desktops, tablets, and potentially smartphones. Smartphone gameplay may be infeasible as the board is large.
	UI elements should adjust and rearrange based on the screen size.

12.1.2: Style Requirements

Color Scheme	We must choose a palette that is both aesthetically pleasing and aligned with the theme of chess. This palette will be inspired by the board design the supervisor has proposed while refining it for the web. This could be a mix of classic colours (like black and white) with some modern twists for our variant.
Typography	We must use fonts that are readable and consistent throughout the app.
	Titles, subtitles, and body text should have clear hierarchies and distinctions.
Animations	Smooth transitions should be implemented for piece movements, capturing, and special game events (e.g., end scenarios) – avoiding flashy animations.
Consistency	We must ensure that the style remains consistent across different sections and components of the game. This includes colours, button styles, panel designs, and modal windows.
Accessibility	Ensure that color contrasts are sufficient for users with visual impairments. We will use WCAG standard and ensure a 4:1 contrast.
	UI elements should be easily navigable using the keyboard or screen readers.

12.2: Performance Requirements

12.2.1: Speed and Latency Requirements

- The application must be responsive and provide real-time feedback to the users. Any action taken by the player (e.g., making a move, sending a chat message, etc.) must be immediately reflected on the interface.
- When playing online, the latency for piece movements between the two players should be under 1 second under normal conditions.
- The game timers should update in real time without any noticeable delay.

12.2.2: Safety-Critical Requirements

- User data, including game states and personal chat messages exchanged between players, must be stored securely, and encrypted to prevent unauthorized access.

- The application should have measures to prevent cheating or manipulating the game state unfairly.

12.2.3: Precision or Accuracy Requirements

- The game's logic should be accurate. All moves should be validated according to the rules of RhombiChess. Incorrect or invalid moves should be immediately flagged and not be allowed.
- The timers should be as precise as possible, with a latency of at most 0.1 seconds.

12.2.4: Robustness or Fault-Tolerance Requirements

- In the event of a sudden disconnection or crash, the current game state should be saved and retrievable once the user returns to their session.
- The application should gracefully handle and prevent invalid moves and actions.

12.2.5: Capacity Requirements

- The backend should be capable of storing game histories, player sessions, game states, and other relevant information for numerous players.

11.2.6: Scalability or Extensibility Requirements

- New rules, pieces, or board configurations should be easy to integrate without rewriting major portions of the code.
- The codebase should be well documented to ensure that future developers can understand, maintain, and extend the application.

12.3: Maintainability and Support Requirements

12.3.1: Maintenance Requirements

- The database must be backed up every month. We need to protect against data loss if there is an issue with our server.

12.3.2: Supportability Requirements

- The code must have clear documentation for each method. Any new developer joining the project must be able to understand the codebase with ease.

12.3.3: Adaptability Requirements

- The code should have an interface for any common properties. By having an interface for common properties, we allow for easy expansion in the future. For example, having an interface for the chess pieces would allow for easy addition of new pieces in the future, as well as modification of current ones.

12.4: Security and Privacy Requirements

12.4.1: Access Requirements

- Each game must have a unique code for the session. Each player should only have access to the game against the opponent they pick, and not anyone else.
- Each game must prevent more than 2 players from joining the session. Only the two intended players of the game must be permitted into the session. Any other additional players should not be allowed to join the session.

12.4.2: Integrity Requirements

- Each player must only be able to move their own pieces. If players were able to move any piece, they would be able to move their opponents' pieces leading to an unfair advantage.

12.4.3: Privacy Requirements

- The game chat must be end-to-end encrypted. A third party may be able to see/modify chat between players if data is not adequately protected.

12.4.4: Audit Requirements

- The current board state must be compared with the state on the database after every move. Since session data is stored in the database, we need to ensure that the board state on the database corresponds to the current board state.

12.5: Legal Requirements

12.5.1: Compliance Requirements

- The chess variant must not infringe on existing intellectual property rights. Infringing on existing intellectual property rights, such as trademarks, patents, or copyrights, can lead to legal action. This includes lawsuits and payment of damages which could be costly.
- The chess variant must adhere to Canadian privacy laws. Compliance with Canadian privacy laws, such as PIPEDA, is a legal requirement for any business located or operating within Canada. Infractions of these laws could result in fines and legal action.

12.5.2: Standard Requirements

- The chess variant must adhere to the FIDE Laws of Chess. We are concerned specifically with the rules regarding gameplay and the roles and responsibilities of players. Regular chess players should already be familiar with this set of rules.
- The chess variant's timing feature must adhere to the FIDE-provided standards for chess clocks. Chess players would already be familiar with how this timing feature works resulting in an easier transition.

13: Risks and Issues Predicted

Each of the following risks have the possibility of turning into an issue:

- **Scope risks:** Unforeseen commitments, such as exams, additional school assignments, or family emergencies, may lead to a change in timeline and resources, impacting the ability to deliver planned deliverables.
- **Resources risk:** Key resources, such as personnel or software, becoming unavailable during the duration of the project.
- **Schedule risks:** Other commitments, such as deadlines for other classes, clashing with predetermined project deadlines, impacting the schedule that has been predefined in the **Tasks** section.
- **Technical risks:** One or more team members may encounter challenges in implementing certain system functions due to a considerable learning curve.
- **Legal risks:** Accidentally infringing on a legal right, such as intellectual property rights.

14: Tasks: Project Planning

To reliably develop the project and deliver for our November Demo, we have strategically split up the project into 5 Epics. 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. Please refer to our Project Development Plan document for a breakdown of the Epics and an outline of individual tasks.

Epic A: Setting Up the Project	Epic B: Creating the Board and Pieces	Epic E: Adding Extra Features
Epic C: Enabling Online Play	Epic D: Implementing Game Logic	

15: Anticipated Costs

We anticipate a few affordable costs as we develop and maintain our web application. Our primary expense may be a server to deploy our backend to. Our supervisor has offered to let us use a server he has access to for free. Still, we may opt for a hosting service to handle our backend as it will save us a considerable amount of time with integration and various tedious setups. For database hosting, we will likely opt for a service such as Neon that offers a generous free tier for serverless Postgres. Lastly, we may purchase a domain for our application which will be a one-time cost for the year. The other tools and software we intend to use are free, leaving us with the following potential cost summary. Besides monetary costs, this project will require about 6 months of development time.

Product / Service	Cost	Frequency	Potential Option
Backend Server	Free - $10	Monthly	PythonAnywhere
Database Hosting	Free - $10	Monthly	Neon
Frontend Hosting	Free - $10	Monthly	Vercel
Domain	$10-20	Yearly	NameCheap