SRS for terminal chess program:

Introduction:

• Chess is a two-player game which is played on a chessboard. Chess comprises 64 squares arranged by eight rows and eight columns. It is a very popular game played internationally by millions of people. This document explains all features, functions, and constraints of this program. This Chess game is built to allow remote users to play each other in chess. Its focus is just to let people play; as users log into the system, they can play with either bot or are paired with the first available player and can proceed. The program needs to be intuitive, reliable, and easy to use. The scope of this project is to provide chess game that is intuitive and entertaining for players of all skill levels (Easy, Medium, Hard). In general, it focuses on providing a simple, streamlined playing experience.

Use Cases Specification:

- User enter the interface and select for playing with bot or another player
- If user select to play with bot, difficulty level is selected and game starts but if player is selected user is connected to another user and game starts.
- In bot mode user will have choice of piece color but in multiplayer mode piece color is decided randomly.
- User with white piece start the game by getting to move first.
- Player select a piece to move then select a box where to move it, program check if it is a valid move if it is a valid move it is executed and chance to move goes to another player but if it is invalid move user will need to select some other place for the piece to move.
- If you are playing against the bot then the bot will perform best possible move after checking whether move is valid or not. The move will be according to the difficulty of the bot selected by the user.
- After every move win condition is check, until one player wins or game ends in tie.

Purpose of Terminal Chess:

- Accessible Chess Gameplay
- Chess Learning Tool
- Quick and Lightweight
- Flexible and Customizable

Scope of the Software Project:

1. Core Chess Gameplay

2. Command-Line Interface
3. Game Management
4. Help and Instructions
5. Configurability 6. No Graphics
7. Error Handling
8. Platform Compatibility
9. Testing and Debugging
10. Documentation
11. Scalability

Functional Requirements

Enable users to start new chess games. Accept and validate chess moves in standard algebraic notation. Enforce turn-based gameplay between two players. Display the current board state after each move. Detect and declare checkmate or stalemate conditions. Allow players to resign and offer a win to the opponent. Provide options to save and load game states. Maintain a record of all moves made during the game. Allow users to review the move history. Offer a help command for instructions on program usage. Allow users to configure game settings, such as starting positions.

Non-Functional Requirements

Ensure efficient program performance with responsive input. Prioritize a user-friendly and intuitive command-line interface. Ensure compatibility with various terminal environments and Postimplant robust error handling and clear error messages. Maintain data security and avoid introducing vulnerabilities. Consider scalability for potential future enhancements. Provide comprehensive documentation, including installation instructions. Conduct rigorous testing, including unit and integration testing. Develop maintainable code to support future updates and bug fixes.

Use Cases

1. Starting a New Game

2. Making a Move
3. Resigning from the Game
4. Saving and Loading a Game
5. Reviewing Move History