

Software Requirement Specification (SRS)

Manan Doshi, Roll No. 140100015
Tejas Srinivasan, Roll No. 140100025

Introduction

Purpose

The purpose of this document is to specify the requirements for the chess game software.

The goal of the program is to allow users to play chess on the computer against the AI.

Scope

This document is meant to be used by end users, developers and testers of the game.

Definitions

- **Bishop:** A piece on the board that can move diagonally as long as no piece is blocking its way
- **Pawn:** A piece that moves one step vertically in the opponents direction when moving and diagonally when capturing.
- **Rook:** A piece on the board that can be moved horizontally or vertically as long as no piece is blocking its way.
- **Knight:** This piece can move 1 space vertically and 2 spaces horizontally or 2spaces vertically and 1 space horizontally. This piece can also jump over other pieces.
- **Queen:** This piece can move horizontally, vertically and diagonally as long as no piece is blocking its way.
- **King:** This piece can move only one space in any direction.
- **Check:** A situation in which the king is under direct attack.
- **Checkmate:** A situation in which the king is under check and it cannot avoid being captured. This brings the game to an end.
- **Castling:** A move in which the king moves two steps horizontally and the rook comes to the square passed over. The king and rook must be unmoved and there should be no piece between them
- **Stalemate:** A situation in which a player is not under check and has no legitimate moves to play. This leads to a draw.
- **Player:** The person playing the game.

Overall Description

Product Perspective

This software allows the user to play recreational chess against the computer.

User interface

The interface will resemble a common chessboard. The player to move will first click on the piece he wants to move and the final position he wants to move it to. The interface will also include information about captured pieces.

Hardware Interface

The hardware requirements for this program are:

- A memory storage
- A mouse
- A Visual Display Unit

Software Interface

-An environment capable of running C++ programs.

Product Functions

The program will provide the following functions:

- Check if moves are valid:
- An AI which will play good, valid moves against the human.

User Characteristics

The user is expected to know the rules of chess. Details of the Rules of Chess are provided in the User Manual.

Dependencies

This program is not platform dependent and should run in any environment.

Requirements Apportioning

- Priority 1: Highest priority. All items must be implemented and verified.
- Priority 2: Expected to be implemented, but omission will not result in unacceptable program.
- Priority 3: Lowest priority. To be implemented after P1 and P2 are perfectly implemented.

Specific Requirements

Priority 1:

- Users should be able to play a two player game of chess on the system.
- The rules of chess should be properly implemented.

Priority 2:

- Chess AI to be implemented for 1 player chess.

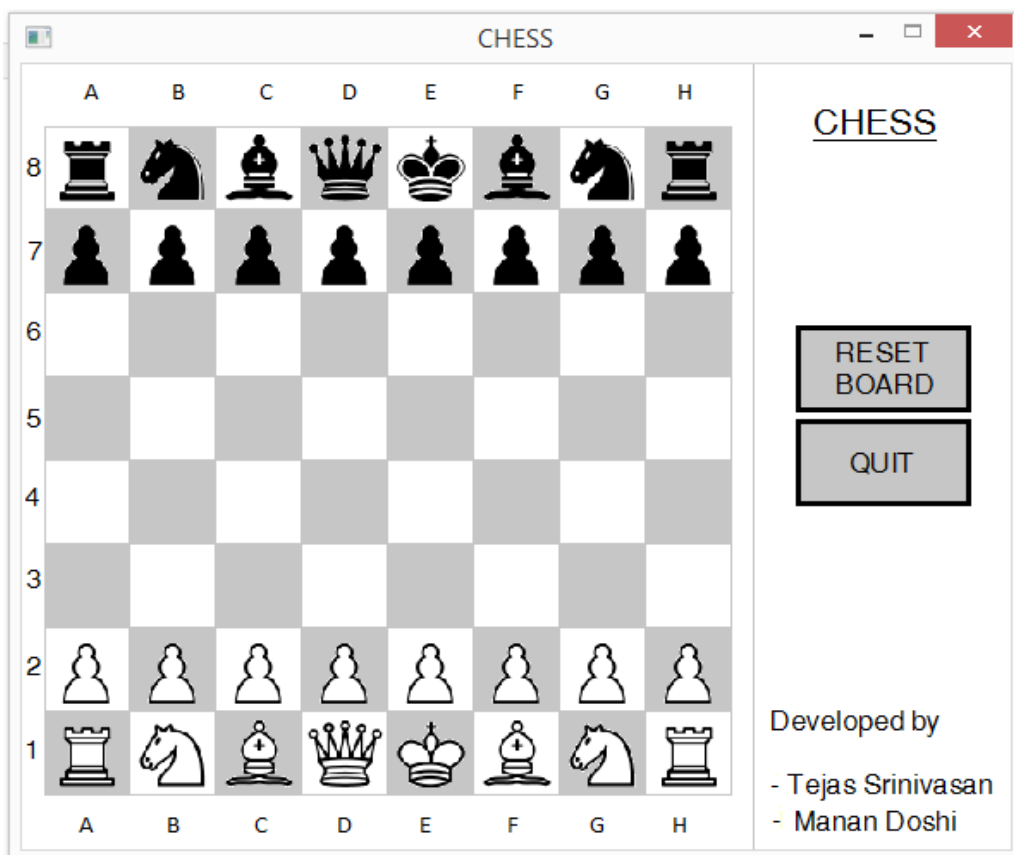
Priority 3:

- Graphical elements to be implemented

User Interface

The UI consists of a clickable chess board with appropriate options.

- User clicks on piece to be moved. The chosen square will be highlighted.
- User clicks on the destination of the piece.
- Program checks if said move is valid.
- If invalid, program displays an error message and asks the user to choose another move.
- If valid, the move is made, else, error is displayed.
- A message is returned in case of check/checkmate/stalemate.
- The AI calculates the position and makes an appropriate move.
- The move played by the AI is denoted by red boxes.



Basic Schema

- An 8x8 array of pointers point to objects of the class piece.
- The class piece has subclasses corresponding to each type of piece.
- The type of piece pointed to by the pointer signifies the piece on the corresponding square.(A null pointer signifies an empty square)
- Input consisting of source and destination is taken from the user.
- The virtual function validMove() checks if the said move is legal.
- If legal, the destination pointer is made to point the object pointed to by the source pointer and the source pointer is made a NULL pointer.
- The validMove() function is different for every piece and is implemented according to the rules of the games.
- The attack() function checks if a square is attacked by an opponent piece.
- If the square containing the king is attacked, the program states the king is under check.
- If there is no move through which the check can be removed, the program states that the game has ended.
- If there is no legitimate move and the king is not under check, the situation is considered stalemate and the game is declared drawn.

Graphical Library

- The graphical library to be used is SDL(Simple DirectMedia Layer).
- Simple DirectMedia Layer is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D.
- The following sprites will be used in the graphical interface:



AI

- A simple chess engine will be implemented. The engine will use several algorithms to make an efficient AI.
- MiniMax algorithm is used to minimize the loss in a game, while Alpha Beta Pruning is used to minimize the number of evaluations used by MiniMax.

Functions Used

1. main.cpp

void game()

Runs the game.

void clear()

Deletes memory allocated to the square pointers.

2. structures.h

bool attack(int playerTurn, piece *sq[8][8], int row, int col)

Returns true if (row,col) is being attacked by opponent.

void getMove()

Gets move input from user

bool check(int playerTurn)

Checks if playerTurn is under check.

bool checkmate(int playerTurn)

Check if playerTurn is checkmated.

bool stalemate(int playerTurn)

Checks if player is stalemated.

void game()

Runs the game.

int sumPieceValues(piece *sq[8][8])

Returns the total value of all pieces on board to judge if the game is in endgame or not.

In Class Piece:

int getPlayer()

Returns the ID of the player to which the piece belongs.

char getColor()

Returns the color of the piece.('W' or 'B')

char getName()

Returns the name of the piece.('K','R','N','B' etc.)

virtual bool validate(...)

A virtual function which return true if (sRow,sCol) to (dRow,dCol) is a valid move.

virtual bool getHasMoved()

Function returns true if piece has moved atleast once in-game.

virtual void moved()

Function is triggered when piece is moved. It sets the hasMoved variable to true.

int getPTable(int i, int j)

Returns the positional value of piece at the square (i,j).

virtual int possibleMoves(...)

Function adds list of possible moves to moveList.

3. ai.h

int listPossibleMoves(...)

Function makes a list of all possible moves for playerTurn and stores it in moveList

void makeMove(...), void undoMove(...)

These functions make and undo the i^{th} move respectively. It is used by the AI to make moves to evaluate different positions.

int evaluate(piece* sq)

This function evaluates the position. The value is high if black is in a better position.

The AI uses the minimax algorithm with alpha-beta pruning to choose the best move. We have used a dynamic linked list of nodes to implement this algorithm. Each node has:

listOfMoves[][] – The list of moves playable on that node.

numMoves – The total number of moves playable on that node.

alpha, beta – Alpha and beta values for the node.

childValues – Value of the children nodes.

value – Value of the current node.

type – Specifies if node is a max-node or min-node.

in – Pointer to incoming node.

out – Pointer to outgoing node.

int getValue()

Gets the value of the specified node(uses recursion to find value of children nodes).

int getCompMove()

Returns the best move for the AI after ply 4 analysis.

4. graphics.h

void getMove()

Gets the next move from either the user or the AI

void gPrint()

Prints the game window.

void gPromote()

Provides interface to handle pawn promotion.