CS 152 PROJECT REPORT
Chess

NAME OF STUDENT

1. **ABHINAV KUMAR 180050003**
2. **BHASKAR GUPTA 180050022**
3. **NIMAY GUPTA 180050068**

Problem Description-

The idea behind the project is to simulate the game of chess and to make a chess engine with one player and two player mode options.

Program Design-

Program folder contains three files
We have assigned piece values to the pieces different from their materialistic value for the purpose of writing lesser code.

1. Rules- This file contains all the rules required for 2 player and 1 player as well as other functions required in other files i.e. checks whether a move is valid or not and also checks that the player doesn't captures its own piece.
 a) Valid-move? -This function is like the heart of rules file. It determines whether a move from initial position to final position on the board is valid or not. Code common to pieces like king, queen, rook has been written as a separate helper function inside this function to make code non-repetitive.
b) Make-move - It makes changes to the board given initial and final position and it has been taken care that this function is called only after valid-move? So that illegal moves aren't made.
c) Direc - It returns the direction vector for the movement of the piece given piece value as argument.
d) List-of-moves-global -This function returns all the list of possible moves that can be made by the player which is later used in functions like checkmate and minimax.
e) get-piece-pos - Returns the position of a piece given it's assigned value and board.
f) Under-check? - Determines whether the given king is under check or not.
g) Checkmate? - Determines whether the king is under checkmate or not by using under-check and list-of-moves-global function.
h) Stalemate? - Determines whether a player is stalemated or not.

2. Chess-main- This is the main executable file which generates the graphics interface on running. This part of program is based on 2htdp package file.
We have used 2-d vector to represent the board because vector-ref provides access to element in constant time and this function is called millions of timesBest in each move of computer.

a) State - This a struct which defines the state of the game. It contains chessboard stored as a 2d-vector and other information like piece selection.
b) mouse-handler - This is the function which makes user interact with the code, i.e. it controls the mouse events by which player can make moves. It has been made separately for 1 player and 2 player modes. It uses function defined in Rules.rkt to execute within rules.
c) board-maker - This function creates the UI of the game. It takes the State as argument and creates an 8x8 chessboard with pieces at appropriate positions.

3. Minimax- This file contains the functions for determining the best move from computer's side upto certain depth level.
a) evaluation - It evaluates the score of given chessboard on the basis of materialistic value+ piece position value from piece-value table.
b) minimax - This function is the key function for making best move from computer's side.
It takes six arguments namely- board, depth (current depth of search), maximising player (to find whether it's computer's turn or not), alpha (maximum score that can be achieved till now),
beta (minimum score), curr_score (the score of the given chessboard).
c) final-depth - A global variables to put limit on depth of searches.
d) bestmove - Determines the best move using minimax function.

POINTS OF INTEREST-

Usage of Macros-

1. List-comprehension- lc has been used in making the list of possible directions of movement for a given piece.
2. While- It has been extensively used in functions like list-of-moves-global where traversing through the whole board is required, also in functions where possible moves for queen, rook, bishop are required.

Use of state variables- Numerous global and local state variables have been defined like whose-move in main file for keeping track of the player whose move is now, structs for king and rook, computer-move in minimax required for checkmate conditions while searching through a given depth.

Higher-order functions- Various higher order functions have been defined to prevent repetitive coding.

1. In the definitions for rules of valid-moves for rooks, queens, king, bishop checker has been abstracted out to check whether intermediate space between movement are empty or not. Move-through helper for valid moves of queen, rook, bishop, function f for valid moves list of king, pawn, knight because they have fixed length move.
2. In the minimax file
3. In the definitions in main file

Use of abstraction in coding- To make code shorter, many functions have been defined
Get-piece- it returns the piece at given coordinates (i, j) on the board. We have used this function instead of repeated use of (vector-ref (vector-ref board i) j) to make things short and understandable from name only.

My-vector-copy- It returns a copy of 2-d vector.we haven't used the vector-copy because in case of 2-d vector, it works on the principle of pointing to objects so when the copy is changed, original also gets changed.

Heuristics- We have used the piece-value table for each piece except king in the score evaluation function in minimax which captures all the ideas from mobillity to space advantage and it is quite fast too because we have used 2-d vector to represent the matrix for each piece. For rooks and pawns we have given special preference for the seventh rank, and the table helps the computer in taking control of the centre.

Hash-t- We have used hash table to store the materialistic value of a piece required for score evaluation because it has the features of using key instead of index and constant time access.

Update- Instead of evaluating the score of board each time we update the curr_score in minimax function for the next board to lessen the time.

Efficiency- The score function is called millions of times in an normal move of computer so we have made it robust by assigning very large values to king so that it is tempted to save it's own king and capture other one. Also since there is a piece value table too, it makes the best move even in depth 3 and not any reckless move so overall it's ability in depth 3 is higher than that of any other depth 3 chess engine.

Alpha-beta pruning- We implemented alpha-beta pruning along with minimax algorithm to cut off those branches in search which cannot affect computer's move or which cannot be one of the best moves in any case. In mid-game, this increases the efficiency of algorithm in depth 3-4 by 15-20 times.

Packages- Used 2htdp/universe and 2htdp/image for making the interface for the game.

Features-

Undo- Any move in both the modes can be undone any number of times by pressing 'U'

Highlight- Certain tiles of the chessboard get highlighted

- yellow for selected pieces
- green for possible moves of the selected piece
- red if the king is under check

Limitations and bugs-

- We have tried to implement castling and also made the structs for king and rooks to keep track of conditions for castling but later on dumped the idea due to complexities.
- In one player the user is always white,and computer makes move only after pressing enter.
- Since the game of chess is based on thinking many moves further in advance and endgame requires higher number of depths to be searched and due to time limitation, this program doesn't works well for depth > 3,it doesn't plays well in the endgame.
- Sometimes,in depth 2 ,left rook is moved by the computer for some unknown reasons whilst there are obvious better moves than doing that and if caught in that it ruins the game.