Assignment-1

The Chess Problem

Date January 26, 2024

Name: Shrey Patel

Email id: shrey.patel@dal.ca

Dalhousie Id: B00960433

Overview:

In the chess problem, we try to implement few functions in order to make sure that the chess pieces move the way the original chess rules are. The user input is in the form of a txt file in which there are series of moves for both white and black in alternate turns. The logic of those few functions is defined such a way that it checks whether move is valid by some mathematics and then returns true or false. The another input is the configuration of the board which can be a N*M rectangle of any arbitrary size where columns are less than 26.

The problem then further is mainly divided into three different phases. The first phase reads the board configuration file, the next phase prints the file if the format is valid according to the constraints and the last one make sure that the move sequence provided are correct according to the rules of chess and then makes the move. All the three phases either return true or false. Apart from this there are few functionalities to check whether the king is in check or not and to store the captured pieces for both the white and black pieces.

The input of the board configuration can contain any letters of chess board game like K(king) ,R(rook), N(knight), Q(queen), B(Bishop), P(Pawn), '.' or blank rows at start or end. The capital pieces represent black pieces while small pieces represent white pieces for the same letter. The dots represent the blank spaces where the piece can move. The other input that is of move sequence is in the form of a string that contains start row, start column, end row, end column; all separated by a space. The output is expected to be a Boolean for move.

The constraints of the problem are to be handled while running the code. The handling of proper inputs for move sequence and board configuration is a key feature that is to be taken care of. Few approximations are taken for making sure that the problem does not go out of scope and the outcomes overlap those with the expectation.

Files and External data:

The program mainly requires two file as input for implementation. One is the board configuration file that has the board size and piece details and the other is move sequence file which can have more than one move defined for the game alternately for white and black piece. Both these files are read in the java files mentioned later.

The program has two java files that are used to implement the whole logic of the problem. The first file is the followChess one which has the whole logic for the reading the input files of board configuration and move sequence and accordingly performing the operations on the

board as per the sequence input. The other file is the main class file named A1 where the followChess constructor is called and all the functions in that constructor are called. Here both the txt files are opened and passed in the followChess constructor to perform the load and print board. Also, moveSequence function is called here which takes the sequence file as input and performs the required operations.

Below is the data flow of the implementation of the chess problem:

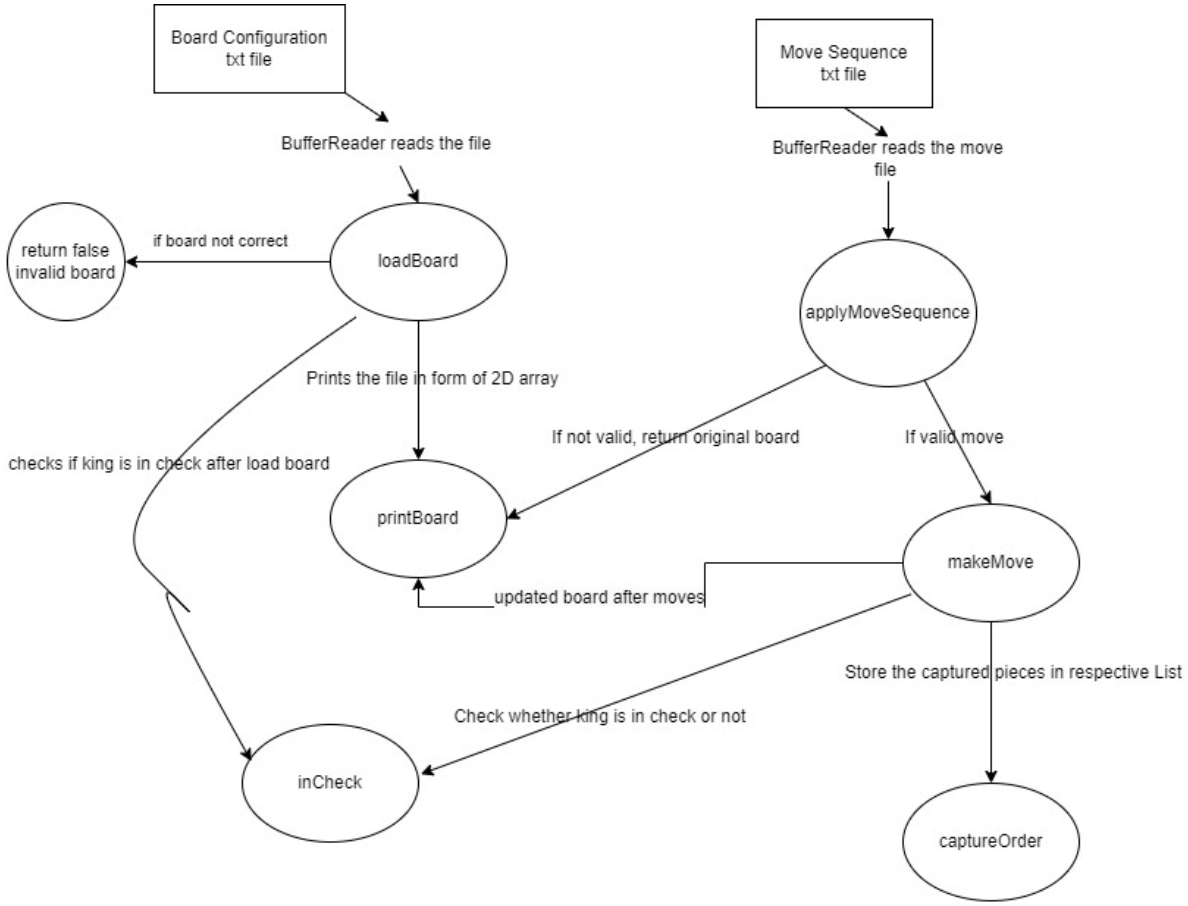

Figure 1. Data Flow chart

The whole implementation of the problem is handled by six functions that handle different tasks and make sure that every constraints mentioned are followed:

- loadBoard: reads the board configuration file using bufferreader and after checking the board size constraints, whether it is rectangle or not, checking for exact one king for both pieces and ignoring the blank lines at start and end of the board. The function returns true if the all the requirements are met. The board configuration is then stored in a 2D array that is used further in the code.
- printBoard: once the loadBoard returns true then this function makes sure that there is no null value and then prints the board considering the case where there can be blank spaces before or after the board. It flushes all the strings from the board configuration file.
- applyMoveSequence: reads the input file of move sequence and after checking all the constraints for the particular piece whose move is called, it returns true and makes the move. All the main conditions for different types of pieces like queen, rook, pawn etc

whether they can move for the given input string is checked here and after checking the mathematics true or false is returned.

- captureOrder: when the applyMoveSequence is called and the move is made to capture the opponent, it stores the data for the same in order. On passing 0 in the function as input, one can get the pieces captured by the white player and on entering 1 the black player captured pieces are returned.
- inCheck: checks whether a particular king is in check or not in the next move. It takes 0 or 1 as input. If the white king is in check then on passing 0 it will return true else false. On entering 1 one can know if the black king is in check or not.
- pieceCanMove: All the information regarding the movement of the pieces. Returns true if a particular piece has atleast one valid move when the function is called. The initial position of the piece is passed as the input

Apart from these functions, I have defined few more functions that are called in the above main 6 functions to perform sub tasks. For example, I have defined separate functions that returns Boolean for each piece to check the mathematics of whether it can move or not. These functions are called from applyMoveSequence after checking the validity of the board. After this to make the move in practical scenario a different function called makeMove is called in those functions.

Data Structure and their relation to each other:

For FollowChess:

As per the guidelines the efficiency of the code did not matter, the output only mattered. So, I preferred to use only simple data structures for my code. In the whole logic for all the functions mainly 2 types of data structure, one is array and the other is List. I have used this data structures as I had good knowledge of both of them and have used them in the previous projects as well.

Array is used in the whole code of followChess where the loops like for, do while are used. Apart from these arrays are used to store the whole board in the form of 2D array. The string from the file is read and is then converted in the form of a 2D array. The next place where array is used to assign the piece value to a variable. Each piece in the chess board array has one particular location which is represented in form of unique combination of row and column. The character value of that is assigned to the piece.

List is also used in this code. It is used to store the captured pieces in the data structure. The array list is dynamic in nature so it is flexible and easy to manage. Also the data is needed to be stored in order, so list provides the same functionality. So we can get sequential access to the pieces that have been captured.

For A1 main class:

There is no need of using data structure in this code, because the whole logic is defined in the followChess and called in the A1 class.

Assumptions:

- It is assumed that inCheck function will get only 0 and 1 as input; also pieceCanMove function will get start row and start column as input separated by a space.
- Only one input file of move sequence is given; code can't handle multiple files.
- The move sequence must be in a specific format in order to make sure that the system works else it will return invalid move sequence. The format is startrow, startcolumn, endrow, endcolumn all separated by spaces like "a 1 a 2".
- Knight moves without checking if the path is clear or not from the source to destination.
- Pawn can't move two steps at once.
- The input board must be a rectangle for this to work.
- The first move is always of white player.

Choices:

- I don't store the updated board file. When a user enters the new move sequence file then, the old file is flushed and all changes are made in the new fresh board configuration.
- The code is designed in such a manner that there should be exactly one king of both pieces. But if we want to make a dynamic chess of any size then there can be more then one king with different rules, we never know.
- Given a number of move sequences, the final board is printed till the move is correct, after some moves there is a incorrect move, then the system will return the board till the last valid move.
- I have used character to store or pass the piece because the piece is exactly single lettered; if it would have been multiple lettered then I would use string.

Key algorithms and design elements:

loadBoard: Here the main goal was to read the board configuration file and make sure that it follows the constraints and then return true.

Key Algorithms:

- Use bufferreader to read the file.
- Read every line in the input until you get to the finish using a loop. Put every line into a variable so that can be processed later.
- Make sure that there are an appropriate number of rows and columns in the board setup.
- Verify that there is precisely one black king ('k') and one white king ('K').
- Confirm that the pieces on the board (for example, "P," "N," "B," "R," "Q," and "K" in standard chess) are inside the expected set either in lower or upper case.
- Construct a 2D array to represent the chessboard and store data.
- Assign piece characters to the appropriate locations on the board as we go through each line in the setup.

printBoard:

Key Algorithms:

- After verifying the load board pass the data and use print writer.
- To iterate across the board's rows and columns, use nested loops.
- Print each board element by having access to it.
- Look for any vacant squares, then print a space or a placeholder character there.
- Print the board to the output stream after accepting it as a parameter.
- Look for any vacant squares, then print a space or a placeholder character there.

applyMoveSequence:

Key Algorithms:

- To read every move from the input sequence, use a loop.
- Break down every move into its constituent parts (e.g., starting square, ending square).
- Verify each move's format (e.g., "e2 e4").
- Verify that the movements comply with the chess rules.
- Connect the row and column indices (e.g., [1, 4]) to the algebraic notation (e.g., "e2").
- Make sure the player in play is the one whose piece is at the starting square.
- To reflect the changed piece positions, update the internal representation of the chessboard.
- After every legal move, switch players and keep track of who is now in the lead.

isValidMove:

Key Algorithms:

- First step is to check for the boundary of the board whether the given input is within the range of the boundary or not.
- Then we have to check if it is white player’s move then it should attack black only and vice versa.
- Then check for the both the pieces whether the move is valid or not by defining rules for each piece like queen, king, knight, rook etc. using proper mathematics[1][2].

NOTE: I have referred AI for logic of knight and pawn movements.

captureOrder:

Key Algorithms:

- To store captured pieces, keep a data structure (such as a list or priority queue).
- Set the data structure's initial values.
- Add the captured piece to the data structure whenever a capture event happens during the game (e.g., one piece capturing another).
- Utilising the specified comparison function, sort the data structure
- Create a method that will allow you to get the captured parts in the stored order.

inCheck:

Key Algorithms:

- Get the king's current position for the designated player (e.g., black or white).
- Analyse every piece your opponent has on the board.
- Check whether each piece belonging to the opponent can legally move to the player's king's position.
- The king is in check if any piece of the opposing side can legitimately attack the king's position.
- Give back false if not in check and true otherwise.

pieceCanMove:

Key Algorithms:

- First get the start position and end position of the particular piece in terms of index of 2D array
- Then find whether there is atleast one move possible for the piece to move in any direction around it for atleast on step.
- Make sure that the piece cannot move when it is surrounded by all its same team players.
- Call the isValidMove function by adding and subtracting rows and columns in current piece location.
- If it returns true then there is atleast one move possible else piece cannot move.

Limitations:

1. In this chess system there is no choice for the user to undo the moves that has been entered. The whole system runs again when the user wants but cannot undo his moves.
2. Castling and pawn promotions are not handled or defined in this system.
3. The user is forced to enter the input in a particular format only to run the desired move.

References:

1. *Chess Pieces Names, Moves & Values*. (n.d.). Chess.com. https://www.chess.com/terms/chess-pieces
2. *Design a Chess Game*. (2019, July 7). GeeksforGeeks. https://www.geeksforgeeks.org/design-a-chess-game/
3. *Implementing "Check" in a Chess Game*. (n.d.). Stack Overflow. Retrieved January 27, 2024, from https://stackoverflow.com/questions/23380770/implementing-check-in-a-chess-game