Chess Game over cli

1 Project Description

For my project, I chose to design a Java application that allows two users to play chess. It is entirely programmed in the Java programming language.
This application does not only play the classic version of chess that has been enjoyed for centuries, but users can also play a chess variant of my own design called “Chess Game”. Chess Game is played just like chess for the most part. The first major change is how the game starts. Each player gets to rearrange their back-line pieces before the game, all except the king. There is no rule where the pieces can be placed except for they must be on the player’s back line and the king must stay in his same spot. The second (and last) change is that pawns can also move horizontally one space as well as their usual vertical movement. Pawns still capture diagonally.

2 Class Description

The chess game is designed by using multiple classes that work together to form the game. Following is a description of each class as well as some of the more important class methods. As the classes are described, the logic used to design the chess game as a whole will become apparent.

2.1 Class Player

Class Player defines and holds each human player’s information. That information includes: the current game board, the player’s piece color, the player’s user name, and finally the number of pieces that the player has remaining.
The most important methods in Player are its accessors. In fact, these make up all of the methods in Player. These “get” and “set” methods allow the other classes to get and use information about a player without having to store it themselves.

2.2 Class Piece

Class Piece is used to define each of the 32 pieces used in a chess game. Each Piece is con- structed in one of two ways: empty, or with a color, type, and row and column coordinate specified as given parameters. The different piece types are programmed into Class Piece as constants and are as follows:

- King = 1
- Queen = 2
- Rook = 3
- Knight = 4
- Bishop = 5
- Pawn = 6

The methods in Piece are used to access its data and other classes utilize these methods to view and change the data in a certain Piece. One method unique to Piece is getPieceName(), which returns the piece's shortened board name for display on the text based game in the terminal. This function is used by the Board class and will be talked about more in the Class Board section.

2.3 Class Board

Class Board holds the information for the chess board in each game. The board is designed to be completely dynamic, being updated each time a player makes a move. The board itself is an 8x8 Piece array, where blank spaces are empty Piece objects. Board also contains an ArrayList object filled with the 32 pieces with which each game of chess starts. I chose to use ArrayList objects throughout my program for a few reasons. First, they allow me to create type-safe containers so that the wrong object will never be placed into the wrong list. Next, because I cannot predict the order that pieces will be captured from game to game, ArrayList's ability to access its elements randomly is really handy. Additionally, as I will describe later, ArrayList can hold another ArrayList as its object which was extremely useful when keeping track of the location of each piece on the board.
In Board, the Piece ArrayList holding all of the game's pieces is filled using the createPieces() class. Pieces are simply added each game by using ArrayList's method "add(element)". This method either creates a classic chess board if the user chooses to play a game of classic chess. If the NEW NAME game was chosen, the user is prompted through the command line to choose the location of their queen, rooks, knights, and bishops.
Another method of note in Board is the populateBoard() method. This is used to update the board array each time the Piece object ArrayList is changed.
The method currentGameState() is the graphical user interface for my text based version of chess. Each time it is called, it outputs to the terminal a display of the current game state. This method makes use of Piece's method getPieceName() while building the game board each turn.
To update the game board, the method updateGameBoard() is called. This method first

calls “clearBoard()” which as the name suggests, clears the board and then uses populate-Board() to fill the board array.
The remaining methods in Board are used to access its members as well as “pawnPromotion(color)” that promotes any pawn of the given color that made it to the opposite end line.

2.4 Class Move

Class Move was the most meticulous class out of them all to write. Chess has so many factors to consider when moving pieces across the board: is the path clear? If not, what type of piece is blocking the path, your own or your enemy? If it is an enemy, can you capture it next turn?
These are the type of questions that I had to ask myself while I was coding the Move class. Move is constructed with a single parameter of the current game board. This is because Move needs an updated board to determine if the proposed move is valid.
The primary method in Move is “boolean move(currentSpace, destinationSpace, player)” and I will use its description to describe many of the other notable methods in Move. Both currentSpace and destinationSpace are Strings that are input by the user from the command line. The strings are converted from input form: a letter followed by a number (example - “d3”) into an integer row value and column value. This conversion is done using the methods “inputToRow(String)” and “inputToCol(String)”. Once the current space’s coordinates and the destination space’s coordinates are known, these values are checked to make sure that they are legal values. If they are, the piece at the current space coordinates is found using the method “findPiece(row,col)”. Then an array list of an array list of integers (ArrayList¡ArrayList¡Integer¿¿) is formed with a list of all of the found piece’s available moves. This is done using the method “legalPieceMoves(piece, boolean)” which takes input parameters Piece and a boolean (true if king is in checkmate, false otherwise) and returns all of the input piece’s allowed moves. It does this by filling the first ArrayList in the returned ArrayList with allowed row values and the second ArrayList is filled with the column values that correspond to each row. The method knows the possible values by utilizing methods like: “possKingMoves”, “possQueenMoves”, etc. that when called return row, column ArrayLists with that specific piece type’s allowed moves. Once the to-be-moved piece’s legal moves have been determined, the list of moves is iterated though until the destination coordinates match up with a legal move space’s coordinates. If they match then the piece can be moved to the target space. Additionally, if they match and an opponent is on the space, the opponent is captured. However if the destination coordinates do not match any of the piece’s allowed move coordinates, the move is a failure and the user is prompted to input a valid move.

2.5 Class Help

This primary function of this class is to help the user, as the name states. When Help’s method “display()” is called, a few options are shown to the user: see application information, see basic chess rules, see chess scramble rules, castle piece, and quit game.
This class, along with FileSystem, are the two classes that did not get completely get finished.

Ideally, the player could use Help to access some of chess's special commands like: castle, surrender, quit game, save game, load game. However currently it is being used for just basic information output to the user.

2.6 Class FileSystem

Class FileSystem is an addition to the application that I attempted to make once the main game was functioning correctly. It was my hope that the user could store the game board in a text file to save the game and then read the data from the same file to load a game. However, it was a late addition and something that I hope to improve in my application in the future.

2.7 Class ChessTop

This is the chess game's top module, which allows the user to actually play the game. Primarily this class is programmed in the main method. This is because my game is based in the command line and so all of the user's inputs have to go through the terminal. This class has nothing too special in it except for compiling everything together to play the game!

3 Playing the Game

Playing the chess game is very straightforward. Users only need to have all of the classes in the same folder and only have to compile and run ChessTop in the terminal to play the game.
When the game starts, the user is presented with the main menu. Due to the fact that FileSystem was not implemented successfully, the main menu's list only includes two options: start new game and view help menu. If FileSystem is implemented in the future, then both load game and save menu will be added to the menu options.
Game play is just like chess games of old: the white player starts off the game and then players take turns making moves until a checkmate is achieved!

4 Testing the Game / Analysis

The most amusing and tedious aspect of coding the game was testing each method. It was tedious because there are so many methods to check. It was amusing because of how often I found myself having to add in methods because I had previously missed something. The game play itself is pretty efficient; ArrayLists work very well in keeping track of dynamic lists and almost all of the lists used in chess are dynamic. There are some aspects of the game that were not finished because I could not figure out how to efficiently program their methods. The main two were checking the "check" condition and castling players. In both I felt as though I was using too many conditional statements and as such did not add it into the application just yet.
While working on the FileSystem class, I felt as though my class code would have worked

for saving a game but I could not get my top module (ChessTop) to call the save method and save to the file.
All in all, two players can play a game of chess one of two ways: regular and scrambled.

5 Conclusion

In this project, I designed a chess application in the Java programming language. While the application is not complete quite yet, the main function of playing chess is present and the application is open to future improvements. Potential improvements include a GUI, finishing the saving/loading system, interfacing with the cloud to play against someone remotely using Hadoop or something similar, etc. This project certainly tested my ability to use all aspects of the Java programming that was taught in this class and truly helped solidify my ability to visualize many aspects of the language. I have always loved the game of chess; now I have an even greater love for the game because I now know how much actually goes into the game.