Chess Program Documentation

November 30, 2017

9:07 AM

# Description of Program

This program is not complete, not by a long shot, but I don't have the time to finish it right now, as I have other tests to attend to. That being said, this program is supposed to be a Chess playing program, where two people can enter moves to play Chess if they don't have a board handy. I suppose it targets Chess players, but then again, there are many better programs than mine. Currently, the program supports moving pieces, checking if the King is in check (although checkmate is not implemented). The program only allows people to move pieces to valid locations, disallowing them from moving them in strange ways. The program also prevents people from moving their King into check. At this point, it's basically a movement and board system for Chess.

# Design

The program has two major classes. There is the abstract ChessPiece class, which is the superclass for all the chess pieces (Rook, King, Queen, Knight, Bishop, Pawn) along with the empty space class, as empty space needed the ability to be stored in an array of Chess Pieces. The second major class is the Chessboard class, which contains a 2D-Array that is the board, which contains all the chess pieces used in the game. The Chessboard class also contains a variety of methods that draws the game board, and updates the position of all the pieces. Each piece has a specific icon, along with an internal position on the board stored as Point object. This internal position is used when try to move the pieces, and is updated every time the piece is moved. Each piece also has an ArrayList of Points containing possible moves, which is updated using a method specific to each type of piece every time a piece is moved. This ensures that pieces don't move in ways that are illegal in the game.

# Design Principals

I approached this project with a fairly mixed approach for design principles. I had a vision of where I wanted to go, and I had a fairly good idea on how to do these things, which I suppose fits the top-down design approach. However, I decided to pick a specific starting point class to work with, that being the King, and then I expanded towards other things as needed, which is part of the bottom-up approach. I find that this hybrid way might not be the best, as sometimes you realize that you need to add additional things that you did not account for before in order to implement something that you planned for in the beginning. In the end, checkmates turned out to be much too complex for me to implement, as it required a variety of factors from each of the pieces on the board, including every position attackable by pieces of the opposing colour.

# Test Plan

I've tested the movement of every single piece, and have tried some invalid inputs as well. I've also tested the feature where you cannot move the King into check, by trying to move the King into harms way. Unfortunately, I do not have the time to implement the ability to reject command input not in the valid format. Despite this, you can enter invalid moves to the pieces using the proper format, upon which the game will reject you. If you would like to see sample output, please see the accompanying .txt file.