Project 1

Chess

By James Rungsawang

Introduction

Recently I've been trying to get a bit better at chess so I had the idea of creating this project as an application on Qt. Unfortunately I didn't think I could learn Qt fast enough so I decided to start with the baseline of having the entire code for fully functional chess. With the game having 1801 lines of code, it recreates gameplay exactly how you are supposed to be able to play. Making each type of piece move properly only accounted for about 400 lines of code. The rest of it is making sure that the player cannot make moves that leave their king vulnerable, dealing with checks and checkmates, implementing castling, pawn promotions, capturing via en passant and more.

Rules

The game contains the same rules as regular chess. All the pieces move the way they're supposed to and you are only allowed to make moves that obviously do not leave your king open to attack. The board has columns named by letters 'a' through 'h' and the rows labeled by numbers 1 - 8. There is no ai, so both players are controlled by typing in input.

Player 2 refers to white, pieces starting at the top of the board, and player 1 refers to black, pieces starting at the bottom of the board.

You first begin by entering the coordinates (I.E. a7) of the piece that you would like to move. Then you will be given all of the tiles that you will be able to move to from that piece. If you select the king, then you will also be given the option to castle from either side.

At any part of the game on any player's turn you may enter "captured" for a list of all the pieces that each player has captured (provided by two maps).

One of my favorite features that I've implemented was an undo system. If you type undo, it will revert the last move done by the last player and allow them to pick another move. This is done by having a stack containing a pair of two pairs. The pair first contains a pair of the coordinates of the piece before being moved and the data of the actual piece, the second pointer of the pair contains a pair of the coordinates of the piece after being moved and the data of the piece that occupied that space before the move was made.

The Game:

The board consists of an 8 by 8 2d array of data type Piece. Piece is a struct that contains:

String name: the type or name of the piece (I.E. queen, rook)

Int player: the player that it belongs to

Int moves: the number of moves this piece has made (needed for castling)

Bool en_passant: given to a pawn that has moved two spaces forward, is taken away if the opponent does not immediately capture

Check is calculated by determining if any piece is in range of the king and there are moves that the player can do to not lose the game. If the player has no available moves to remove the check (valid moves must not leave the king vulnerable) then checkmate is deemed and the game ends.

Data Structures

Queue:

A queue was utilized to store all of the possible moves that a user could make with a certain piece. I would first generate possible moves and if these moves did not leave the player's king vulnerable then I would push it into the queue. For example in the case of the queen, while you are able to move in a certain

direction, that tile would be pushed into the queue and we would continue to move in that direction until we capture an enemy, are blocked by a friendly piece, or have reached the end of the board (pieces that leave the king open to be captured are not pushed).

Set

After all the possible moves for a piece were placed in the queue, I would add the front to a set, pop the front and continue until the queue was empty. This makes the choices a lot easier for the player (sets are automatically sorted lexicographically) as well as make sure that the user picks a move that is valid. Using the count function I can tell if the user has picked a valid move by checking if it exists or not in the set.

Map

The map made storing captured pieces a lot easier. There are two maps, one for each player. Whenever a piece was captured by a player, I would use the name of the piece (I.E. pawn) as a key and then I would increment its value. This adds the functionality to enter “captures” and then quickly see the frequency of each captured piece for both players.

Stack

With stacks following a last in - first out principle, it was perfect for the implementation of an undo feature. The stack

contains a pair which holds two other pairs. It contains the original position of the piece that the player moved, the piece's values, and for the other pair it held the position it moved to and the piece that occupied that space before the move. At the end of every move, a pair is inserted at the top of the stack. At any point the player can enter 'undo' and it will take the last pair inserted and will use those values to undo the move made. Of course this approach does not properly work with en passant captures (captured piece was not on the location that the piece moved to) or castles so I had to account for those cases separately.

For en passant I checked if a pawn moved diagonally and the diagonal tile did not contain a piece, then this pawn must have captured the pawn that moved forward two spaces (the pawn on the tile below it).

For undoing a castle, because both the king and rook cannot be moved before the castle I reverted them to their original spaces and set the space between them to be empty tiles.

Other things used from the STL

Other than set, stack, queue, map, and pair, some other things were used to help with the game.

The erase function came in handy with parsing the information stored in the pair to undo a move.

Iterators were used to output contents of the set (list of possible moves) and map (types and frequency of pieces captured).

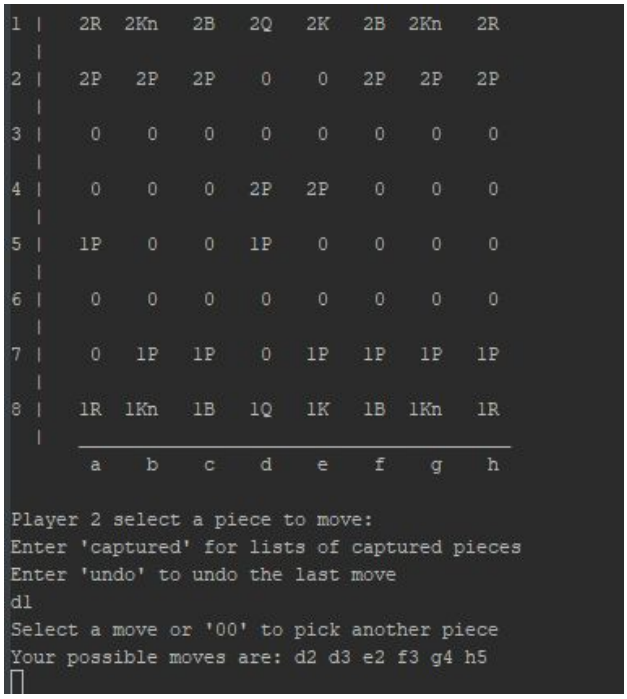
To_string was used pretty frequently as I had to parse user input several times throughout the game.

I threw in the max function to increment the number of moves that a single piece has made the whole game.

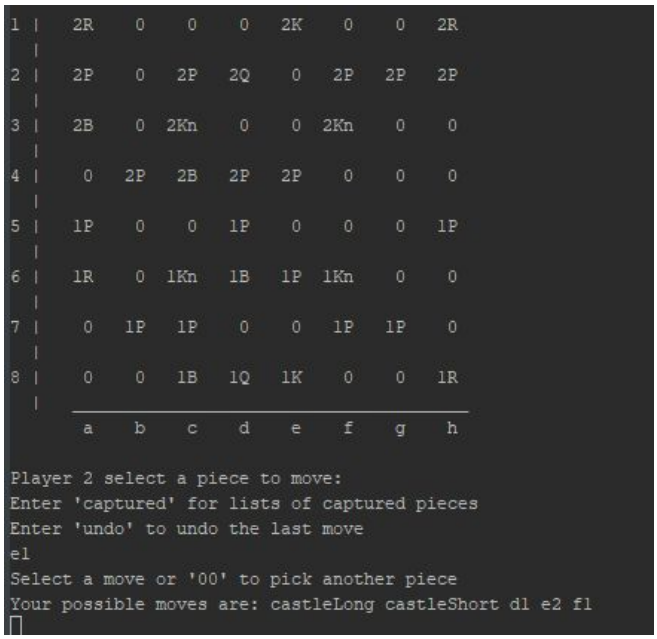
For the knight movement I originally noticed that its path is always the distance sqrt(5) but rather than using the sqrt function I squared the distance and checked if it was equal to 5.

Images Of Gameplay

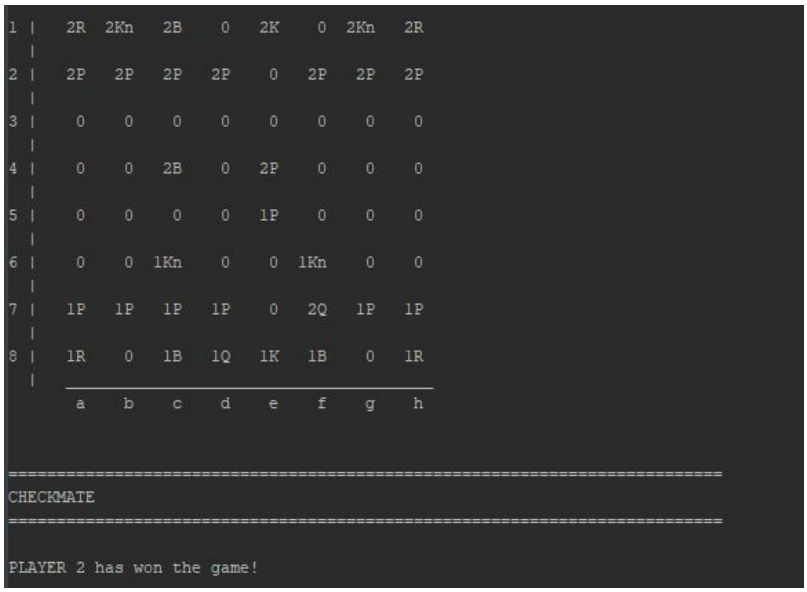
Regular moves



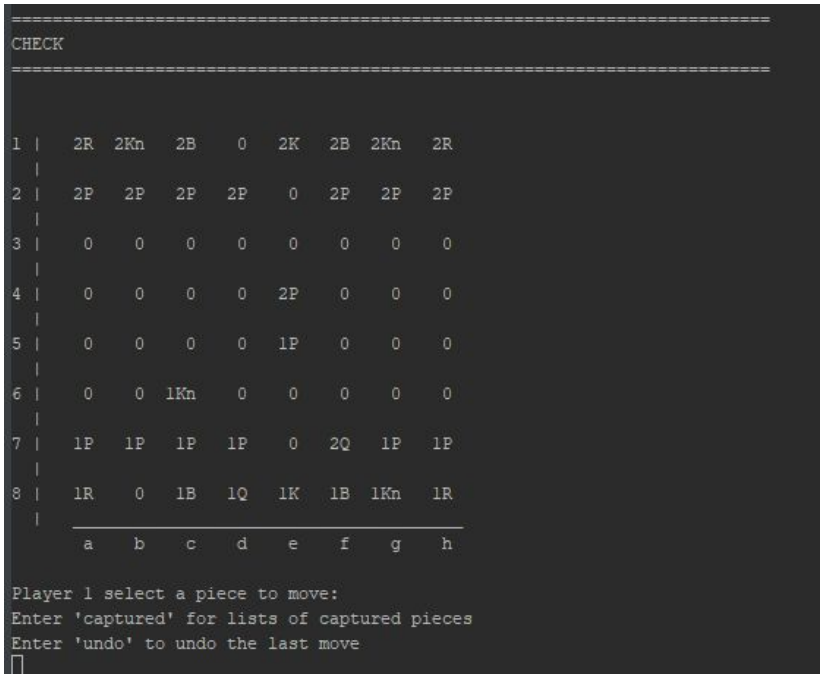
Player being able to castle



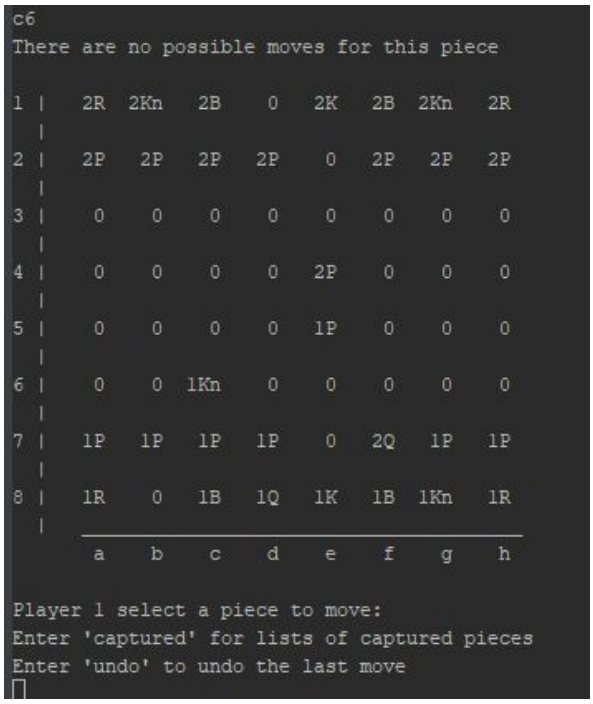
Checkmate:



Check:



Notice how even though the knight can clearly move to other tiles, the options are not presented because the king is in check



The option is castle is not given while in check