Final Report – CS 246

# Design:

See the attached UML for a class overview of the functions and details each class includes.

Our main class is the driver behind initialization of the chess board, and initializes player classes. The main class uses pointers and allocates almost all of its memory on the heap. We could have used references and allocated the memory on the stack, but chose not to do so out of personal preference.

# Questions:

1. Undo Feature: We implemented an undo feature on the board by using a stack. The stack is located in the board class, and contains the moves that have been used already, and the piece type of a piece that has been taken. One of the hard parts of implementing an undo feature is the need for remembrance of taken pieces – they must be reintroduced to the board after they’ve been deleted. The undo feature would not be able to use the move function to undo a move as if the piece was in check before move will not work. Thus we need to make the undo function directly affect the board and create a new piece as well. By creating a stack we allow for an unlimited number of undos, otherwise we’d just keep track of the listed things above for the last turn.
2. Four-handed chess: Fourhanded only requires small changes in implementation. First in our layout, we’d need to add 3 rows at the top, bottom, and sides. A way to do this would be to use a 14 x 14 array, and fill the 3x3 corners with a pointer that cannot be taken. Our pieces are identified by the char ‘b’ ‘w’, and we’d just add ‘B’ (blue), and ‘r’ for red. In the move implementation, we check to see if any move is in the valid moves. In the validMoves() function, we’d need to make almost all of our changes. Firstly, pawns will need to be altered to add side-side movement if the piece is ‘B’ or ‘r’ (this will also need to be done for castling). In addition, validMoves iterates through all possible moves. It currently checks to see if a cell is occupied and stops, and adds to the list of valid moves if the cell is enemy owned, and stops prematurely if the cell is self-owned. We’d need to add a little check there to see if (posX < 4 || posX > 11) && (posY < 4 || posY > 11)