CS246 F23 - Plan of Attack - Chess - Kush (k23patel), Rehan (r2anjum), Ali (maaasyed)

Plan:

(1) Get the Text and Graphic Displays Set up and Resign (Kush)

- Goal: Thursday, November 23
    - Text Display and Graphic Display
        - Letters and Numbers
        - Blank Spaces (Black and White Spaces)
        - Space Allocated for ScoreBoard

(2) Add Pawn and Player Functionalities (Kush and Rehan)

- Goal: Friday, November 24
    - Add Black and White Pawns
    - Add the player changing turn features
    - Board should now be able to update every move
    - Vector containing possible moves should be accurate
    - Pawns Should be able to attack the opposition's pawns
    - Enum conversion from board id's to int function

(3) Add Functionality for Remaining Pieces (Kush)

- Goal: Saturday, November 25 (maybe go into sunday)
    - Bishop - only moves diagonally
    - Knight - moves in an L shape
    - Queen - can move left, right, up, down, or diagonally
    - Rook - Can move left, right, up, down
    - Appropriately calculate the algorithm for created the vector of possible moves
    - Add the attack feature for all pieces

(4) Attack, Check, Stalemate, Checkmate, King Functionalities Finalized (Kush, Rehan)

- Goal: Sunday, November 26 (maybe go into monday)
    - Algorithm checking and checkmate
    - Add all King Functionalities
    - Finalize all properties of each piece, calculate possible moves accurately, make sure attacking is done accurately.

(5) Add ScoreBoard and Finishing Touches with Human vs Human using Terminal (Kush)

- Goal: Monday, November 27 (Finish all steps above)
    - ScoreBoard for Both text and Graphic Displays
    - Test all possible terminal commands.
    - Make sure ALL Human vs Human Functionality is correct.

(6) Level 1 of Computer Functionalities (Ali)

- Sunday, November 26, (maybe earlier)
- Implement the concrete component (level 1) for the computer class. Implement the cpuCalculateMoves() method so that the vector cpuPossibleMoves is filled with all possible moves across all pieces remaining on the board for the computer.
- Each element in cpuPossibleMoves a random legal move. Do not include any attacking. Choose a random pair from the vector and pass it onto the board.

(7) Level 2 of Computer Functionalities (Ali, Rehan)

- Monday, November 27
- Implement features for the decorator component (level 2) for the computer class
    - Implement a ranking algorithm in rankMoves() to prioritize captures (queen, bishop, rook, knight pawn) and check (king) using the priority queue rankedMoves

(8) Level 3 of Computer Functionalities (Ali, Rehan)

- Wednesday, November 29
- Implement features for the decorator component (level 3) for the computer class
    - Implement a ranking algorithm in rankMoves() to avoid giving the opposing team any captures
        - If the move leads to an opposing pawn capturing you, ranking is lowest
        - If the move leads to an opposing queen capturing you, ranking is higher
    - Same ranking results in a random move being chosen

(9) Level 4 of Computer Functionalities (Ali)

- Friday, December 1
- Implement features for the decorator component (level 4) for the computer class
    - Implement a full scale ranking algorithm in rankMoves() which is capable of making informed decisions to player moves

(10) Begin Testing and Add Finishing Touches (All Members)

- Saturday, December 2
    - Graphics testing will begin

        - Custom test harnesses will be made to individually test various class methods

(11) Bonus Features and Level 4, Extra Testing (All Members)

- Monday, December 4 (Afternoon)
    - Begin consideration and implementation for possible extra features:
        - Minimax algorithm for level 4 CPU
        - Stockfish integration(above level 4)
        - Chess variants(i.e. Blitz)
        - Board rotation for human players
        - Move analysis chart
        - Change the graphic to be much more appealing.

(12) Complete Testing (All Members)

- Monday, December 4 - Tuesday December 5
    - Be ready for Submission and Test Everything

Questions:

1. Chess programs usually come with a book of standard opening move sequences, which list accepted opening moves and responses to opponents' moves, for the first dozen or so moves of the game. See for example https://www.chess.com/explorer which lists starting moves, possible responses, and historic win/draw/loss percentages. Although you are not required to support this, discuss how you would implement a book of standard openings if required.

Although there exist many different ways of implementing standard opening move sequences, two viable options which come to mind are the graph and hashtable data structures.

Firstly, a graph data structure could be used to store opening sequences and the moves following said opening sequences. A node would be representative of a specific move, and the respective adjacency list of that node would contain possible responses to the opponent's moves. Historic win, draw and loss percentages would be held in each node in the graph, as part of the move, depending on whether it would be stored as a struct or as an object.

Contrarily, a hashtable implementation could suffice. Many popular chess engines implement standard opening move sequences via hashing or "Zobrist Hashing".[1] By amortized analysis, a hashtable would allow search, insert and delete operations in O(1) time. Each unique store

[1] Lv Huizhan, Xiao Chenjun, Li Hongye and Wang Jiao, "Hash table in Chinese Chess," 2012 24th Chinese Control and Decision Conference (CCDC), Taiyuan, China, 2012, pp. 3286-3291, doi: 10.1109/CCDC.2012.6244521.

associated with the hashtable would contain evaluations of historic win, draw and loss percentages.

2. How would you implement a feature that would allow a player to undo their last move? What about an unlimited number of undos?

To implement undo functionality, a stack could be associated with each player. After each move either player would make, the state of the board and the move itself would be pushed onto the stack. Likewise, boolean flags would be utilized to keep track of the game's current state, letting either player undo more moves. This would result in both players having the ability to undo one or more moves depending on which player's turn it is. An empty stack for either player would indicate that the board is now at a starting state (i.e. no moves have been made).

- A stack that will store the state of the board after each move.
- A function that performs the push() function to add the current state of the board to the top of the stack
- Once the undo() function is called for either player, the state on the top of the stack is popped off. The current state of the board is switched to the new state which is on the top of the stack.

3. Variations on chess abound. For example, four-handed chess is a variant that is played by four players (search for it!). Outline the changes that would be necessary to make your program into a four-handed chess game.

Regarding the structure of the Chess Board by itself, the text and graphics displays must be modified to accurately, visually represent the four-handed chess board. With new dimensions, there would have to be a new board which has to be set up with enough pieces and appropriate positioning for each piece. In terms of the players and their respective rules, we must increase the maximum number of players to 4, with functionality for any combination of human and computer players. Additionally, we must keep track of which player's turn it is and constantly check for a possible checkmate or stalemate for each player following each move. It is also important to consider the possible interactions between pieces on the same team and those on the opposing players. A piece can interact with any of the 3 opposing player's pieces.