Plan of attack

We will first establish a basic bare-bones version of the project that addresses all core features of a chess game including the basic mode for player vs. player. Our efforts will be largely focused on creating the board (Victor) and establishing our piece classes (George and Andy). We will also ensure that all basic commands are working including set-up mode and that our program properly produces text-based display by November 30. We will wrap up final functionality including an observer for graphics based display along with the computer opponents for all 3 levels. We will also spend this time troubleshooting any issues we may face when combining our code. All standard criteria for the project will be completed by December 2. Our remaining time leaves us enough buffer room to debug if any of the prior internal deadlines need to be extended. Furthermore this will also leave us some time to add bonus features for our project depending on how efficiently we are able to implement those.

Question: 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. Although you are not required to support this, discuss how you would implement a book of standard openings if required.

Note that we can represent the current state of the game through a sequence of moves which can be represented as a string. Every move will be a string of length 4 as it contains the starting square of the piece and the ending square of the piece i.e. e2e4 means the piece at e2 moves to e4. Since the starting board is constant, we will always know the board state on any particular move. As such, we can match all openings that follow the current opening sequence and provide the continuation of that opening as suggested moves for the player

To start, we will have two main maps.

The first map will contain the move sequence (string) as the key and and the name of the opening (string) as the value.

The second map will contain the name of the opening (string) as the key and the full sequence of the opening (string) as the value

By keeping count of the number of moves, we know the length of the move sequence. Every move is of length 4. As such, we know at the beginning of the game, our game sequence string is of length 0. After move 1, our game sequence string is of length 4. In general, the game sequence string will be of length 4n where n is the number of moves that have been played.

We will put all opening sequences greater than length 0, 4, 8, … and so on into a vector of maps. Each map will contain a key-value pair where the key is the move sequence and the value is the name of the opening. Index 0 contains all key-value pairs where the move sequence (key) is of length >= 0. Similarly, index 1 contains all key-value pairs where the move sequence (key) is of length >= 4 and so on.

As a result, during the game, when the total step count is 0, we will do vector[0] to access all potential openings. It will return a map where all the opening sequences (keys) are of length 0 or greater. If there are multiple potential openings, it will pick one at random.

Once the step count is 1, we will do vector[1] which returns a map where all the opening sequences (keys) are of length 4 or greater. Once step count is 2, we will do vector[2] which returns a map where all the opening sequences (keys) are of length 4 or greater. We will keep track of the values (opening names) of all elements that match the current sequence of the game. Through this process, we will know which openings the current game is following.

At the same time, each step, we will utilize our second map with the key as the opening name and the value as the entire sequence of moves in that given opening. Additionally, we also have knowledge of the current step count of the game. Thus, we know the exact step we are in the opening sequence and know the next steps that can be recommended to the user.

If the current move sequence is not found in the map, then we know the game does not follow any map opening and thus, we don’t need to compute this any further.

If the current opening does not have any further moves, then there might be other potential opening names that extend off the current sequence. Thus, we will search for openings that extend further steps. Otherwise, there are no further suggestions for the user as the game no longer follows a recorded opening sequence.

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

As UML shows, we will set a previous board (starting pointer is nullptr) and the current board pointer in the chess class, if we want to undo, then we will delete the current board pointer, then let the current board pointer = previous board pointer, then let the previous board pointer = nullptr. There are only two cases when the previous board pointer = nullptr , case 1: the first case is when the game has just started, and case 2: the game has already been undo once. Note that: When the previous board pointer = nullptr, undo is no longer legal, in order to satisfy the rule that you can only undo once in real chess. Finally, graphics render and text render the withdrawn board.

If we need to undo unlimited times, we need to use the standard template library - vector<string> , which records each player's move as the index changes, e.g. vector <string> v {"a1 a2", "b1 b2"}, then v[0] ("a1 a2") means that white's first move was from a1 to a2, and v[1] ("b1 b2") means that the black side played b1 to b2, and so on to keep track of how each player played each move. This makes it easy to undo, for example, if we need to undo n times then we just need to remove the last n strings in the vector, and then replay each remaining move in the vector on a new board according to v[0], v[1] and so on, and finally we get the board after undoing n moves. Note: 1 <= n < len(vector) where n is an integer.

Question: 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.

- Extend the size of the board
    - Change boundaries for detecting valid move
- Increase player count/computer opponents
- Increase to 4 players and have turns alternate across all 4 players
- Have options for 4 teams instead of just black/white
- Give players choice to work as teams/free for all
- Include game modes (i.e. first to checkmate or play for points)