CS246 A5 Chess Plan of Attack

The plan of attack is as follows:

After carefully analyzing the dependency between the objects, we have decided that an observer pattern will be the best fit for our program. The finalized design pattern helped Jiwook in creating the initial UML design. That gave us a fair idea of how each class should be built in the shell of our program. The UML model along with a group discussion helped Prabhapaar in answering the given project questions. The plan of attack was made by Nipun. After submitting the Due Date 1 documents, we plan to create the command interpreter for our program. The completion of the command interpreter in advance will allow us to work on different parts of the program in an organized way and help us build a clear picture of program's flow as well. We have divided the program's functionality into two components: human and computer gameplay. Human gameplay will be our first priority as computer gameplay should follow as a complement of that in our program.

The human gameplay will consist of program's general functionalities including taking input from the human player through the command interpreter and moving the pieces accordingly by calling the move function. Since chess has 6 kinds of pieces where each piece follows a different move pattern, we'll have to take care of each piece's movement differently while making sure to include special cases such as castling, en passant and pawn promotion. Jiwook will be responsible for writing the command interpreter and the move function. Since we are working with observer pattern, the move function will also be responsible for notifying the observer cells of the cell at which a piece has been moved or removed. Prabhapaar will be responsible for writing the notifying functions. The task of implementing an additional feature, setup, which allows users to set up their own initial board configurations, will be carried out by Nipun. Textual display, which prints out the board to the output stream will be implemented by Jiwook and the Graphical display will be implemented by Nipun and Prabhapaar once the general flow of the program has been implemented without any bug.

Computer gameplay consists of 4 difficulty levels. The implementation of our first level will be based upon the functions implemented in human gameplay. We plan on tackling the second and third level by categorizing their strategies into three components: capturing, checks over captures, and avoiding capture, which will be done by Prabhapaar, Jiwook and, Nipun respectively. The fourth level will be implemented after the main features of the project are completed.

After every main feature of the program has been implemented and the program is running without any bug, we will start adding the enhancements to our program such as features mentioned in the questions, printing a history of all moves made, suggesting possible moves and accepting input in standard class notations.

Project Timeline:

Saturday, 23rd March:
Plan of attack, UML, Answers to questions listed in the project

Sunday, 24th March:
Creating the modules

Wednesday, 27th March:
Implementing the move, setup and notifyObservers functions

Friday, 29th March:
Implementing the first three levels of computer function

Sunday, 31st March:
Debugging the program so that general flow of the program runs

Monday, 1st April:
Implementing the enhancements in the program

Wednesday, 3rd March:
Debugging the program with the newly added enhancements

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

To tackle this problem, we'll use a map structure, **OpeningMoves** in our class Board. The key part of the structure will store the name of the opening move, for example "Two Kinghts Defense" and then we'll have another Map structure for the value part of it. The "value" map structure will contain a specific Text_Display of the board in the key part and respective next move based upon the composition of the board. We can store all our opening moves this way and these will be immutable and fixed structures for the moves. Now to implement the correct counter-move in the opening sequence, we'll create a Cell_compare function in our class board so that we can compare two board representations using that function. Next, after every command move from the other player we'll iterate over the given opening sequences' Map structure and compare it to the current state of our board and use the value part to get the coordinates for our next move.

Question 2:

How would you implement a feature that would allow a player to undo his/her last move? What about an unlimited number of undos?

We'll make a vector of vectors, **Moves** in our class Board to store the moves made by our two players. Each element of the vector will be a vector that will store the coordinates of the piece from our earlier move command. The positions will be interchanged such that the new starting position to be stored will be the earlier ending coordinate and the new ending position will be our earlier starting coordinate. For example, if our command was **move e1 g1** then the new coordinate will be stored as [g1, e1]. Now, if we want to undo the last move we can use vector.end() to get the coordinates and use the move command to carry out our undo function. Also if we want to undo more than one move, then we simply iterate from the back of the array, the required number of times and carry out the move command in each iteration using the stored coordinates.

Question 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.

Beginning from our class Board, we'll first need to add an option to make a team so that if two players want then they can plan together. Secondly, we'll need to modify our functions such that the game represents a four-player chess board and moves instead of the original one.

For example, our function that creates the board (init()) and stores in a vector will now have 12 instead of 8 associated vectors and the middle 8 of them will have 12 instead of 8 cells. Next, we'll change our game_default_setting() method and add another two set of pieces on the specified positions. The setOberver will be changed accordingly to accommodate the additional cells and pieces. Next, we'll change the winner and gameEnd methods. The gameEnd will be changed such that three kings need to be checkmated before the fourth can be declared as the winner and two kings of the same team for the other team to win and the game to be ended. The specifications of the TextDisplay and the GraphicDisplay will be changed to represent the new board.