CS 246 Final Project Chess - Plan of Attack

Yuktesh Modgil (ymodgil), Reyansh Patange (apatange), Harsh Dave (h7dave)

Breakdown of Our Project:

For the CS 246 Final Project, our group has chosen to recreate the game of Chess using C++ with OOP principles to demonstrate the knowledge we have gained from this course. Furthermore, we have opted to implement the Model-View-Controller (MVC) architecture along with the Observer design pattern to allow us to separate the logic of the game from the presentation through the User Interfaces. This approach also allows there to be different types of interfaces such as Text and Graphical, which will be our ConcreteObservers in this project. We will break down the project into the three major sections that can be completed semi-independently: the Model, the View, and the Controller.

The Model will arguably be the toughest part of this project, as we need to structure our data in a way that allows us to code the logic for it relatively easily. We plan on representing the actual chessboard as a 2D vector of pointers to tiles. We have chosen this data structure after noticing that the Decorator representation of AsciiArt was somewhat slow when it came to displaying a grid in the graphical observer. Since Decorators are essentially a linked list, most operations such as moving pieces would be of O(n) time complexity. We can optimize this by using vectors that can be updated in O(1) time, given the indices of the part of the board that needs to be updated. Since this is arguably the most difficult part of the project, we plan on getting this part done as soon as possible. We will also need to implement the GameBoard class and overload the operator<< so that we can start testing this as soon as possible.

The View is another section that we need to tackle. As we are aiming to create low coupling between the classes in our program, it should be possible to have someone working on simultaneously creating the TextObserver and GraphicalObserver. Then, when both classes are complete, it should not be difficult to connect the two pieces together as seen in the UML diagram.

The controller is the third major section of our final project. When commands are read in, as described in the command interpreter section of the project description, the controller's methods will be called depending on the command. For example, if the move command is read in, it will call the corresponding move method in the controller. Then, this will update the chessboard in the model and the model will notify the observers to display the newly updated chessboard. Setup mode will also be something that is controlled through the GameController and thus, to facilitate testing, we plan to get this done early as well.

As far as the computer skill levels are concerned, ideally, we want to be able to reach level 4 for the computer capabilities, but at minimum, we will definitely be able to complete level 1. In level 1, the goal is to be able to compute random valid moves, so for this, we could simply implement a function that selects a random active piece on the board and moves it to a valid place - checked by isValidMove(). For levels 2+, further moves will be added accordingly once level 1 is working as the project guidelines mention to "plan for the worst".

Division of Work and Estimation of Completion Dates:

As there are a lot of components to this project, one of the biggest challenges we would face by splitting everything into independent parts is that when we combine them, it may appear inconsistent (since each of us may have different coding styles). So, instead of spending further time trying to fix inconsistencies, we have decided that it would be best to work on core areas of the project together. These core areas will include the main function and the Game class. The rest of the work will be split up as follows:

Section	Name	Date
Core Areas (Main Function and Game Class)	All Group Members	Dec 2nd, 2021
Implementing the Observers (TextObserver and GraphicalObserver) and the View portion of MVC	Harsh	Dec 3rd, 2021
Command Interpreter and GameController Class	Reyansh	Dec 3rd, 2021
Model (GameBoard and Player) and Logic for the Rules of Chess	Yuktesh	Dec 7th, 2021

With this timeline, we have left ourselves enough time after the 7th, for fixing bugs, and doing thorough testing of our program.

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.

Most openings are two moves - white's move and black's response - but some are as long as 5 (such as the Ruy López Opening). Considering this, a book of standard opening moves could be created with a vector of vectors of GameBoard pointers, where each element in the outer vector would be the unique opening moves and each inner vector would be the sequence of black and white moves that form or represent the opening sequence. This would be shown as pieces on a GameBoard. These inner vectors would, of course, be of varying length depending on the number of moves in each opening as mentioned above. A possible way to support this during a chess game is that for the first N moves, where N represents the max length of opening moves that exist in the book of standard openings, the chessboard could compare the current board's position to the positions of each element in the opening book's (AKA vector's) position and instead of iterating over all the moves in each opening in the vector, we would only have to check the gameboard at the index representing the current number of moves so far - if it exists of course.

For time efficiency's sake, if the number of moves the players have made in-game is currently greater than any move's max number of moves or if the number of moves is equal but the pieces differ at that point in time, the corresponding opening move (represented by a vector) could be removed from the outer vector representing the current possible opening moves. This way, the program doesn't need to unnecessarily iterate over and compute moves it already deemed irrelevant after each step. To finally output this to the user, so that the person playing could see the suggested opening moves, the chessboard could be colored in a way that shows where to move a specific piece. For example, considering the tileColor() method from the Tile class for each piece, the piece's current tile position could be changed to green and the recommended position's tile color could also be colored green to show the player where to move which piece. This would be applicable for the GraphicalObserver, and something similar could be done with the TextObserver but instead of coloring tile's (which obviously wouldn't be possible with TextObservers), the current coordinate and recommended coordinates could be outputted for the player to see which piece is recommended to move according to the specific opening being considered in the book of standard openings.

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

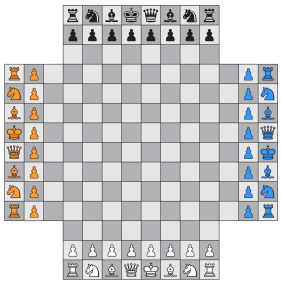
A straightforward and space-efficient solution to the first part of the question would simply be to store a single GameBoard object that creates a deep copy of the current setting of the board (before the player makes their move) in their corresponding Player object. This would allow either player to undo their past move by simply reverting the version of the GameBoard to what is stored in their corresponding Player object.

To give each player an unlimited number of undos is a bit different and will occupy more space. Since we need to remember the state of the GameBoard after every move made, a stack would be the ideal data structure for storing this. The last in first out (LIFO) design of a stack makes it only possible to undo the most recent move. Therefore, to maintain invariants of the game, we would pop moves from the stack until the desired game state that either player wants to revert the board to is encountered. Since every single move is stored in the stack, this would allow us to have an unlimited number of undos. This stack can be stored within the Game object and each item in the stack can be a GameState object. This GameState object would store a deep copy of the GameBoard at the corresponding previous move(s) similar to when simply undoing a player's last move, however, it would also store the corresponding player whose turn it was at that point (ex. in the form of a char). So while popping through the stack, we look for the player who wants to undo their last move. Of course, adding to the stack would also have to accommodate this change, where the GameState object created would also include who the current player moving is alongside the deep copy of the GameBoard.

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.

For the four-handed chess variant, a significant difference between traditional chess is the board. The board is now a 14x14 board with 3x3 cutouts from each of the corners. So this can be represented as a 14x14 2D vector. Of course, this means that pieces should not be allowed to access the indices which lie in the corners of the game board as these are not valid locations for pieces to land. This can be checked in the isValidMove() method from the Piece class. Another difference is, instead of the Game object storing 2 different players, it must now own 4 players which can all either be humans, or computers.

Furthermore, the TextObserver and the GraphicalObserver will have to be updated to now render a board with 4 different players, which looks similar to the image below. This will involve adding new colors aside from just the simple black and white colors offered in standard chess which can be controlled through the Player class, by storing the colour of the Player's pieces within the Player object itself.



https://en.wikipedia.org/wiki/Four-player_chess

For the TextObserver, we must also find a way of displaying the pieces of the 4 different players, we can no longer just use uppercase and lowercase letters as specified in the Chess game description for this project. One way, which may not be the most user-friendly, is to prefix the letter of the game piece, with a numerical ID assigned to each player, ranging from 1 to 4.

Next, there will now be 4 players whose turns are rotating in order, instead of switching back and forth between 2 players. This rotation of players can be stored as a vector containing each of the players and an integer that keeps track of whose turn it is in the vector.

Pawns can also reach left and right ends, instead of just across to change pieces. So now instead of just checking if it has reached a coordinate that is at the last row of the opposite side, we will also have to check if the pawn has reached a coordinate on the first or last column. This can be checked within the move method of the pawn, which after moving, checks if the aforementioned conditions have been met.