plan.pdf

Plan Attack:

We will be using the MVC pattern to implement the text and graphics of the project. Text and Graphics class will inherit from View and will have notify functions that print the Board each time move is called. The commands from input will be processed by the test harness and the test harness will call the corresponding functions from the Controller class (game, resign, move, setup). Each specific piece has its own class inherited from the base call Piece. They all have the canMove() function which checks if that piece can do the Move. Move is its own class that contains two Squares, start and finish. There will be an Abstract Player class, the Controller class will own the Player class. Player class will own Move class that will be responsible for translating the input commands to Move class to feed it later on. There are 2 classes that inherit from the Player class, which are Human and Bot class. While Human is a concrete class, Bot is an Abstract class that will have 4 more classes inherited from, which are the 4 levels of the Both.

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. 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.

Answer:

Structure:

We would implement a tree of Positions. Positions is a class that inherits from Board and would be the nodes of the tree. Some private fields of Position: responses which is an array (or vector) of Positions, name which is a string of the name of the move (if there is one), turn which is a Color of who's turn it is next, and depth which is how far down the node is in the tree. The field, responses, contains all the possible Positions after the next move from the opponent.

How responses is calculated:

For each Piece of Colour turn, canMove() will be called for each possible position on the board (e2, etc.) and for all responses that are true, a new Board will be constructed with that move implemented (movePiece() is called on that Position) and that Board will be added to the list of responses. The tree knows to stop calculating responses when the tree's depth reaches 12.

How to implement historic wins/draw/loss percentages:

There are 2 ways to implement this, one involves storing the games that are played through the program, the other involves using a larger external chess database. For the first way, the games can be saved in a json file for example, where there is a field which stores the moves

that were played and the result of the game. Then there is a method on the Position class that reads from this file to calculate the win/loss/draw percentage. This method would work by searching through each past game to look for the passed in move and find the result and find the percentage of wins/draws/losses using the total number of games. The second way involves interacting with an external database either through using api calls or downloading a chess database locally. A method on the Position can then interact with this database to determine the win/loss/draw percentage.

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

Answer:

How making a move would be stored:

In the Controller class, we have a private field called prevBoards which is a STL container (vector) or doubly linked list of all the previous Boards in the game. When Controller is constructed, the prevBoards field is constructed to be a list of length one containing the starting Board. Every time a move is made, (using a method in the Controller class) the resulting Board is added to the end of prevBoards.

How undo() and redo() works:

Another private field in the Controller class is currBoardIndex which is an Integer that keeps track of the index of the Board we are currently using in prevBoards. We would also implement undo() which is a public method in the Controller class. When undo() is called, the currBoardIndex field is decreased by one. undo() can be called as many times until currBoardIndex = 0 (is the index of the starting Board). Then an error message is printed to stdout saying “There are no moves left to undo”. Another public function, redo() is implemented and can be called when the length of the list (either called by .size() if using a vector or we have a separate private field in Controller called length which is an Integer that contains the length prevBoards) is greater than currBoardIndex. currBoardIndex then increases by one. If a new move is made that was not a redo or undo, then all elements in indices after currBoardIndex in prevBoards are erased and the new Board is stored at the end of prevBoards.

How erasing works when a new move is made and there are elements in indices after currBoardIndex in prevBoards:

This depends on the container we are using. For doubly linked lists we can start at the last element (back pointer) and erase elements from the end until we reach the element we are currently at (each node can have a field where it stores it’s current index). For a vector we can use an iterator that starts at currBoardIndex + 1 erases each element (coding it where the iterator is available and updated each time by taking the return value of .erase()).

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. (If it’s important to your answer, state whether

you're assuming free-for-all or team rules and then answer the question. You don't need to get too specific into the rule set changes in answering the question though; your focus should be more on what would need to be altered at the high level of the design?)

Answer:

Free-for-All:

We have an enum Color that contains WHITE and BLACK, however this can be extended to contain more colours. The Board would be modified to be a bigger vector that would still be like a square board but the corners are invalid squares. We would add another public field to Square that is a bool valid which is true if it is a valid square and false if it is not valid (or out of bounds meaning it is not actually a square on the 4-player chess board). Each player could have an extra field called points which keeps track of how many points they have as well as another field with is a bool called inGame which is true if the player is still in the game and false otherwise. The program already has to check for a checkmate, so we can increment that player's points by 20 when they checkmate someone. We also have to check for stalemate so we increment by 10 multiplied by the amount of players in the game by checking each player's inGame field. We will already have a checkmate function and we can add if statements in the function to check which piece does the checkmating and if it is a queen, check if that queen is checking other kings and add 1 point for 2 kings and 5 points for 3 kings. We can do a similar if statement setup for pieces that are not queens. We would have a capture function that contains if statements to figure out which piece is captured and add 1 point for a pawn or promoted queen, 3 points for a knight, 5 points for a bishop, 5 for a rook, and 9 for a queen.

Standard Teams 4 Player Chess:

For the capture() function, we can add an if statement so that they cannot capture their teammate's pieces. The game would end when a checkmate occurs on one of the opponent's kings and again, we can add an if statement to check that the king is not their teammate's king.

Plan:
https://drive.google.com/file/d/1MRkCfMbcApw6Gmde8kcXpgeXW_Y-F7-S/view?usp=sharing

The work will be divided in 3 parts:

	Allison	DB	Quang
Fri 07/19	Set up project with respect to UML diagram		
Sat 07/20	Finish setup mode	Finish implementing Rook, Bishop and	Finish level 1

		Knight class	
Sun 07/21	Finish Text implementation		Finish level 2
Mon 07/22		Finish King, Pawn and Queen	Finish level 3
Tue 07/23	Finish Graphics implementation	Finish implementation of board and controller	Finish level 4
Wed 07/24	Testing Text and Graphics		
Thu 07/25	Testing		
Fri 07/26			