By Henry Zhang and Yingjia Zhang

DESIGN

1. OVERVIEW OF THE PROGRAM

A. This program utilizes decorator design patterns, separate compilation, object-oriented programming. Inputs are read in by console and the outputs are printed to screen.

B. The program consists of several interfaces and each of them contains one or more class:

i. main.cc is the test harness of the program, it reads in the inputs and calls the related functions from other classes. It provides a platform for players to interact with the program.

ii. chesshelpers.h is where all the helper functions are stored by organizing them into a class. In main.cc, we created a class of ChessHelpers to call the related functions when necessary. Methods are in ChessHelpers also responsible for some error message outputting. // and coordinates class

iii. board.h contains the class Board, it is responsible for setting pieces, retrieving pieces, initializing the checkerboard and printing the checkerboard.

iv. decorator.h, pieces.h, noPieces.h are the components of decorator design patterns, they contain a method canMove to determine if it is a valid move for a kind of pieces.

v. bishop.h, king.h, knight.h, pawn.h, queen.h, rook.h are the concrete decorators, they are specified to contain the unique moving rules, capturing rules and special moves of the pieces.

vi. computer.h contains the class Computer. Computer is responsible for running computer1, which randomly moves pieces.

2. DESIGN

A. Separate compilation

i. Utilize the decorator design patterns to implement the unique moving rules for the pieces:

1. *the Component being the Pieces class,*
2. *the ConcreteComponent being the NoPieces class,*
3. *the Decorator being the Decorator class,*

4. *the operation of component is canMove, a function that determines whether a piece can be moved from a certain position to the desired position*

ii. Overwrite the operation of Decorator class, canMove, to be the moving methods of each kind of pieces:

canMove (int locx, int locy, int destx, int desty) is a function that returns 1 if the pieces at location (locx, locy) can move to location(destx, desty) on the board following the pieces' moving rules, otherwise it will return 0.

iii. Visualize the board to screen by creating the Board class, the Board class is used to:

1. *setPiece(char piece, int x, int y) places the piece to the location(x, y) on the board;*
2. *getPiece(int x, int y) returns the character at he location(x, y) on the board;*
3. *emptyBoard() sets up the empty board with the number and letter boundaries;*
4. *initialBoard() initializes the empty board and places the pieces on the board to begin a new game;*
5. *printBoard() prints the board to screen;*

iv. Create the ChessHelpers class to store any helper functions that would use in main.cc. Some examples of how ChessHelpers helper functions work in main.cc:

1. *switchturn(std::string &whichColour, std::string &whoseturn, std::string blackPlayer, std::string whitePlayer) switches turns between two players;*
2. *checkBounds(int x, int y) checks if the location(x,y) is in the boundary of board;*
3. *moved(Board * myboard, char loc, char dest, int movefromx, int movefromy, int movetox, int movetoy, bool move = true, bool print = true) checks whether the movement rules of pieces are satisfied. If so, it will move the piece and return the Boolean value true; What's more, whether or not to print the board or move the piece can be controlled by the parameter "move" and "print".*
4. *checkExitSetup(Board * myboard) checks if the conditions of exiting setup mode are met;*

B. main.cc implements the gaming commands and allows users to interact with the game by typing in commands in the console

i. If input is “setup”, then setup mode will be opened,
   1. *use “+ char loc” to place pieces char to location loc on the board,*
   2. *use “- loc” to remove misplaced pieces from location loc on the board,*
   3. *use “= colour” to decide which colour of camps will make the first move in the new game,*
   4. *use “done” and “game” to exit setup mode and start a new game with the board that was just setup, the program will check if the conditions to exit setup mode are met and alert the player with text message.*

ii. If the input is “game player1 player2” and setup mode has not been visited, the program refreshes the original board and updates with new players. If setup mode has been visited, then a setup board will be displayed as discussed above.

iii. If the input is “resign”, the program will automatically decide which player will be credited with a point based on the parameter whichColour, and a text message will be displayed to the screen to remind the player;

iv. If input is “move”, the program will decide if the piece can move to a desired position based on each piece's canMove operation, the program will alert users:
   1. *if this piece's move does not conform to the rules,*
   2. *if this piece’s move will go beyond the board,*
   3. *if the king is in check,*
   4. *if the player wants to move the opponent's piece or an empty square on the board.*

v. If the player wants to end the game manually, control+d could be used to end the whole game and exit, the final score will be printed on the screen.

C. Computer // to be completed

3. RESILIENCE TO CHANGE

A. High cohesion and low coupling are achieved in this program:

i. Association is utilized: Computer class and ChessHelpers class

ii. Inheritance is utilized: Pieces class, NoPieces class, Decorator class and classes of each kind of pieces

iii. Dependency is utilized: main.cc and chesshelpers.h

B. Chesshelpers

i. Instead of putting everything in main.cc, we chose to move some functionality to ChessHelpers class, store them as the methods under that class. In main.cc, we include the ChessHelpers myhelpers to store the helper functions and when certain functionality is needed, we call the related functions from myhelpers. This decision highly increases the cohesion between the modules of codes and make it much easier to locate the errors.

C. Specialization

i. By using the decorator design pattern, the complicated repetition process is omitted, we overwrite the pure virtual method defined in the parent class and custom that to be the unique moving rules of each kind of pieces

ii. We also add the additional moving rules to Pawn class to decrease coupling so that the main.cc is left with only necessary lines like reading in input and calling related functions from ChessHelpers class.

4. ANSWERS TO QUESTIONS

A. To implement a book of standard openings, create a file that stores the moves commands according to the number of moves. When the move sequence is needed, the file would be read.

B. To implement the undo feature, create an array that store the previous checkerboards and write a function to retrieve the checkerboard after undo. Pass the number of undoes to the function, clear the current checkerboard, and return the checkerboard wanted. Use dynamic memory when all the space in current array is occupied.

C. Necessary changes to implement the four-handed chess game:

i. The checkerboard sizes

ii. The rules of check and checkmate

5. EXTRA CREDITS FEATURES

To better serve the players and provide a relaxing environment for playing the game, we enable:

A. Round number count:

i. Be able to count and print the rounds of a game

B. Error message output:

i. Prints out error specific messages when wrong inputs or wrong moves are given

C. Special move prompt:

i. Prints out prompts when special moves like castling are executed

6. FINAL QUESTIONS

A. UML is the first thing to consider when writing large programs, whether working in a team or alone, it gives a clear scope of how classes are related to each other and how the program will be implemented. It is more like a guide to the order in which the various classes will be implemented.

B. When transferring the pieces to the board, we delayed some time in the 2-D array coordinates. It is a different kind of arrangement compared to normal coordinates. We forgot to check it at the very beginning when we wrote the program, so it took some time to fix the errors.