Design Document for PP9K

DESIGN PATTERN:

The program uses MVC pattern:

It has a controller class which is a GameNotification,and the controller owns a view class with derived class textdisplay and graphicdisplay, and owns a board, and setboard which has two derived class setupboard and setloadboard.

Board owns two players(either human or computer). Each player has a same board, and owns 16 chessmen.

DIFFERENCES ON DESIGN:

1.the biggest difference is that, I split the Setboard into 3 class, Base Class Setboard, which setup default board; Subclass Setupboard, which enters setup mode; Subclass Setloadboard, which load a pre-saved game. It would be a large block of the code if I put everything in one class, and it won't be helpful at all, once I started to debug. So I split them into 3 class.

2.Additioanlly for the test, debug purpose. I reduce the relationship among Board and Player, Chessmen, so that Board will only pass the integer board to the player. Player will update the board, when the move is successful, and Board only need to check the integer board whether there is a updated information.

3.Controller will take responsibility to delete Board. Since I only want to delete the Board when the program is terminated

DIFF ON QUESTION:

Q1: No difference(a book of standard opening move sequences)

Q2: NO DIFF

Q3: No difference(four-handed chess)

Q4:First I will read each line from file into a string, and define a variable r=0 represents rows if the length of the string is 1 which means I read the player(W,B) then I will set turn to corresponding player, otherwise I will go into a loop and each time update r=r+1,so that I will have a coordinate for each char(r and loop variant=columns),inside the loop check each char if it is any pieces, if it is then I will call a function called set to set the integer board, which is a protected function used by all kinds of setup.(to remove or add piece to the board)

CLASS AND THEIR RELATIONSHIP:

GameNotification:
notify text and graphic dispaly to print relative information.

Controller:
It "is a" GameNotification, and it "owns a" View, Board, and Setboard.
1) setup
Once the program is executed, it will call Controller::play().First thing is it will setup an integer board(2D array) by creating different setboard objects(takes a board class) depend on the different setup modes. After that, setboard object will return an integer board that are already set up to controller, then controller will pass it to the board class. Secondly it will call setboard class to set up player once the `game player player` command has received. Finally it will create a textdisplay and graphicdisplay(optional depends on command line argument -g).
2) play;
After the TextDisplay and Board and GraphicDisplay(optional) have been setup. Thefunction play() will start to take all the commands input from the keyboard in order to play the game, such as move, game player player, resign. and after each successful move, Controller:notify will be called to notify textdisplay and graphicadisplay to print corresponding information, and Board::checkwin() will be called to check if any result occurs. For the purpose of playing game many times, the TextDisplay and GraphicDisplay and Board will be clear every time at the begin of the game.

Setboard:
It has two subclasses for different tasks. It will create an intger board, and return this board to the Board class which is passed by controller when the Setboard object created. It will also setup player for the Board class.
1.Setboard
Setboard will be created if player wants to play a normal game.
2.Setupboard
Setupboard will be created if player wants to enter setup mode,and check if the setup is valid.
3.Setloadboard
Setloadboard will be created if player wants to load a pre-saved game.

Board:
Board "owns a" players.
When the game has begun, the controller will only call this class to play game.
It stores all the information about board, such as player(computer or human)and the score for each player, whose turn, and the integer board setup by Setboard class. After each successful it will call GameNofitication,to print the relative information, and checkwin() will be called to see if any result occurs by checking if there is information updated on the integer board.

View:
It has two subclass which will print text and graphic information
1.TextDisplay
It has a char theDisplay(2D array),After the controller notify the textDisplay it will print theDisplay. It will clear theDisplay first once the game is started for the purpose of multiple games.
2.GraphicDisplay
It "owns a" Xwindow class to display the graphic information once the program executed with -g flag.
It will clear the graphic information first once the game is started for the purpose of multiple games.

Windows:
It opens a graphic display and allows to draw the information on the display.

Player:
Player is a pure virtual class, it has two concrete subclasses, one is Human, the other is Computer. There are two significant different between Human and Computer. One is input mechanisms. The input mechanisms for Human is based on cin, it means Human will respond information only after a real person give some input from the command line. However, computer will respond as soon as its tern comes. The other difference is computer will generate various valid move depending on its level, so during the Computer initialize part, a computer level should be set.

For the initialize part, Board will transfer some information to player, group(an integer, determines the player is a white player or black player), and a pointer to a 2D - 8 * 8 character array(this array represent a small character board. It shows all of the information about the chessboard. And it is the key to connect the Board class and Player classes). Of course, the Board class is also responsible for initial computer level if the Player is actually a Computer. Besides of this, in other to make the future check and checkmate step more easier, each player has a field called opponentPlayer, this is a pointer to another player. Basically, one Player "has a" Player. Also, one of the field for Player is a Chessmen array, it will store every chessman(maximum 16) which belongs to thie Player. After receiving the 2D-array from Board during the constructor, Player will go through this array and check each position, whether there is a character represents a chessmen that belongs to this Player. If it belongs, then the Player will new the corresponding Chessmen into the its Chessmen array.

After receiving(Computer generates them by itself) a starting and ending coordinates, Player will check if the coordinates are valid for the 8 * 8 board. If it is valid, Player will transfer the ending coordinates to the particular Chessman(finding the particular relies on the starting coordinates). The Chessmen will respond if the ending coordinates is valid(Valid means this Chessman can move to the ending coordinates, no block exists. More details will be discuss following when introducing Chessmen class). Also, Player will call Player::checkCaptured(int, int) to make sure that whether the chessman will move like that, will the king be in check by opponent player. If it's valid and king will not be in check, then player will call Chessmen::makemove(int, int), and transfer the ending coordinates to it, then the Chessmen will change its current information(will be discuss later). Also, Player will update the board(2D-character array) and leave some message on it, then the Board class will know what actually

happened and do the proper things.

Suppose Player-1 just completing moving step, it still has two task to do. The first one is checking if Player-2's king is under check. The second one is check if the game is draw(it means Player-2 have no valid move to a execute). If the check exist, Player-1 will call opponentPlayer->checkNotiy() to tell Player-2. If the draw exist, Player will leave some message on the 2D-array(board), then the Board class will know and do the proper things

Chessmen:

Chessmen is a pure virtual class, it has six concrete subclasses, which are Pawn, Rook, Knight, Bishop, King and Queen. Most of their functionality are same, the only different is that Pawn has a extra field, lastMoveDistance. Pawn use it to handle the enPassant case. When Player want to check if a Chessmen can move to a given place, Player will call Chessmen::checkMove(int, int) and Chessmen::checkBlock(int, int). This two function will determine if the ending coordinates is reachable and no block exists between the current position and the destination(it also will check if castling is valid and pawn promotion). Chessmen also have some supporting function like getType(), getCoords(), getGroup(), getLastMoveDistance(), resetLastMoveDistance(), Player may call these function to get some information for each Chessmen.

FINAL QUESTION:

1)

When working in team, the responsibility and the interaction between partners' Classes needs to be clearly demonstrated. The relationship between class should be as less as possible, especially between the partners' Classes, so that one developer makes a big change will not affect too much on the other, it also will be easy to debug. Finally a good Design pattern will save a lot of time.

For writing a large program, top-level design is very important. If the top-level design is ambiguous, the low-level(the more accurate and specific) part will be really hard to design. For example, if the top-level is not well-design, when working on low-level design, a small change may totally change the top-level design structure, and it may also influence back to other relevant low-level design structure, that changing is not easy to handle.

2)

If we have a change to start over it, we will spend more time on design the relationship and connection between each class instead of implement them rashly. We may find some better way to connect the Board and Player, and make the notify mechanism simple.