Design document

**Final Design Explanation**

In our final design, the program is built around three main objects: Board, Pieces, and Controller. To make life easier, we provided the same interface to computer players and human players through a abstract class : Controller. The Controller class communicates receives commands from players and notify the Board to change game state; simultaneously, the Board will notify the Pieces to update their state. In addition, the View class provides latest game state and visual enjoyment. Based on the View class, players can make choice more conveniently. Different from human players, computer players are far more difficult to make dicisions based on graphic display (Actually impossible based on what we have learnt). Therefore, we added an extra field (a pointer to the Board) for the Computer class (A subclass of Controller, so is Human class)which gives the computers players the necessary data to make moves.

**Deviation from original design**

Initially, we underestimated the difficulty of implementing the AI part; consequently, our Pieces class were designed quite simple. During our implementation, we found it very hard to meet some of the specific requiments of the game. Thus, we re-designed the frame of our program by adding a few more fields and methods to Board class and Pieces class to make our life easier. For example, we now have a range field in our Pieces class containing all the possible moves of a piece. We also have a enpassant field in our Board class helping us solve the move of Pawns, etc.

**Explanation of design pattern used**

In our program, ***Model-View-Controller design pattern*** is employed. Basically, we break the program into three parts:

Model : The Model is the data, which is the game state and consists of the Board and Pieces classes (abstract).

View : The View is the display, which consists of the View class (abstract).

Controller : The Controller receives user input and is responsible for mediating between the Model and View parts.

The ***singleton design pattern*** is applied to the Board class, which ensures the uniqueness of the board.

**Questions and Answers (Due date 1)**

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.

*Original Answer:*

First, we can find some standard opening moves sequence online and store them in standard notations. We store the moves of our board also use the same notation, so that the moves can be matched with the standard openings we store. When the opponent make a move, we search through all the openings in our library and match one, then perform the next move according to the standard openings we store.

*New Answer:*

Same thought and same design.

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?

*Original Answer:*

Creating a vector field in Board class is preferred. The vector stores every move of each player in the form of “string”. For instance, “e2 f3 P” means a move of a piece from e2 to f3 capturing another piece P(Pawn). In each player’s round, push\_back() method is needed to push the string containing the necessary information of the move into the vector. To undo the moves of a player, we need to call the pop\_back() method of the vector class which contains the information about the specific moves. If the string being popped ends with a uppercase letter, then a piece has been captured in the last move. Otherwise, no piece has been captured in the move. For example, the string being popped is “e2 f3 P”, we move the piece on f3 back to e2 and put an opponent piece Pawn at f3. Since vector will automatically allocate and manage memory so the vector could store infinite move; therefore, enabling unlimited number of undos. :)

*New Answer:*

We actually finished the full implementation of undo function briefly based on our original answer. For convenience, we used vector data structure in our program. Additionally, functionality of undo for castling and enpassant is added.

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.

*Original Answer:*

1. Add necessary cells to the board (four 3\*8 areas on each side).
2. Add corresponding fields of pieces indicating the player it belongs to.
3. If team-mode is enabled, check algorithm should be changed to prevent friendly-check.
4. The Pawn promotion should also be applied when a Pawn reaches the King's row to the left, right or directly across.
5. :)

*New Answer:*

Same thought , same design.

**Questions and Answers (Due date 2)**

Question 1:

Communication is the key. Since the work is distributed to two people, it is extremely important for team members to work together. For example, the developers in a team must maintain the consistency of the interface. We made a horrible mistake during our implementation. One of us wrote a function that takes in 2 integers representing the position of a Piece on the board; however, the other one thought the previous function takes in a string instead of two integers. Disaaaaaster !!!!

**One spirit, one team, one win !!!!!!!!!!!!!!**

Question 2:

If time goes back, we would spend more time on the design process. At first, we implemented based on our initial design. As time goes by, we felt more and more difficult when we add on more features to the program; consequently, we have to re-design our program and modify a lot of our original code. A good design is half of the success.