Plan of Attack

We plan to start from working piece.cc and game.cc files together at first. The game.cc is a multi-functional file in this game which requires substantial methods to run the program. **Basically, the most important and shiny spot in our program is the moveBoard field which is a 3D vector that including a vector of all pieces that can move to a particular cell in the next move, and this field makes most of the difficult functionalities possible to solve. Moreover, it makes our testing easier.** Additionally, we think working on these two files at the same time helps us to have higher cohesion because these two file depends on each other. Thus, it's easier to add required fields when someone needs one. **We use Model-View-Controller(MVC) and observer in our program.** Then, we will test each small functionality based on the chess rules and show it in TextDisplay. After that, we will start the AI part. Lastly, the final debugging process will be done.

Timeline:

Date	Job	Assigned to
July,16th	Implement piece, including all other specific pieces	Houze, Xiao
July, 17th	Implement game.cc & game.h	Helen, Liu
July, 18th	Continue game part and TextDisplay	Helen, Liu
July, 19th	Testing TestDisplay to see if the game setup and some simple moves work	Everyone
July, 20th	1. Computer AI1 and AI2 2. Computer AI3	Siwen, Yang Houze, Xiao
July, 21st	Castling, enpassant	Siwen, Yang & Helen, Liu

Date	Job	Assigned to
July, 22nd	Graph Display	Helen, Liu
July, 23rd	Writing test cases and documentation	Houze,Xiao & Siwen, Yang
July, 24th	Debugging	Everyone

Q&A:

Q1:

Chess programs usually come with a book of standard opening move sequences, which list accepted opening does 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.

A1:

Implement a class that runs the game for the first dozen or so moves and stores all the moves. Reset the board after.

Q2:

How would you implement a feature that would allow a player to undo his/her last move? What about unlimited number of undos?

A2:

We prefer adding a Stack vector in game.cc which stores string to indicate every move both players have been made. For example, we have a function void notify(std::string cmd) that will make the required move called by player, like “move e2 e3”. Then, we need to change the command to “move e3 e2” to the Stack. When the player called “undo”, the last string will be call pop_back() and send it to notify(std::string cmd) to cancel the previous move. Because vector will automatically allocate and manage memory, infinite moves can be stored in the Stack, and player can undo as many times as he/she wants.

Q3:

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.

A3:

1. In the setup mode, the board will add four 3*8 areas on each side of the 8*8 square board.
2. Pawn promotion is applied when a pawn reaches the King’s row to the left, right or directly across.
3. The turn passes clockwise.
4. If team-mode is accepted, then team field should be added to piece class