Plan For Chess

By

Stan Zhang, Steven Shi, Steven Sun

The plan for developing chess is as follows. It is really hard for multiple people to write on the same pieces of code because it may cause confusion, so we decided to split the project up into 3 parts and three of us work on each part independently. Steven Shi is responsible for writing the core logic and data for the chess game, he is responsible for classes Gamestatus and Board which contains the data of the game, and classes BoardOperatorBase and MoveFactoryBase which determines the correct move and operation of the chess units. Stan Zhang is responsible for writing the Display classes and the Observer classes which display the chess board corresponding to the game status. Steven Sun is responsible for writing the classes Player, Pieces, and Moves which constitute the player in the game, the chess pieces in the game, and chess pieces' movement. After all the classes are finished and assembled, we discuss and write the level of difficulties together.

We decided to write the project in this way because we can divide the whole project into three parts: Display, Gameboard logic, Player and Chess movement. And each of us can work on our own part independently. When our independent work is done, we can link our individual parts with minimum code addition. For example, when we need to modify some parts of the project like Steven Shi needs an extra function A that is responsible for a promotion of a chess unit, Steven Sun can just modify the Moves class he wrote without other partner's help or revisions in other part of the project. In this way, we can avoid having to update multiple files due to some small functionality added. We just need to make some little modification to the specific classes that is responsible for that function. Moreover, each member of the group is responsible for writing the documentation for the classes to make partners know what each part is doing and its functionality.

All of us plans to finish our part by the end of November 26th. This leaves us at least one week's time to assemble our own parts and discuss and write the level of difficulties together. Also, we have plenty of time to debug and test the functionality for each other's code.

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. 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.

To implement a book of standard opening move sequences, we would make changes to our logic parts of the program. Instead of creating only one MoveFactoryBase class, we declare another StandardMoveFactoryBase that inherits from the MoveFactoryBase class. For the class we have

a reference to the Gamestatus object to get the board's status in order to determine the next standard possible moves depends on the game status. When the game starts, we create two lists that one lists contains the possible opening at the starts of the game and the second one contains the next moves that should take place corresponding to the first move. We can use index or hashmap to link these two lists.

Question 2: How would you implement a feature that would allow a player to undo their last move? What about an unlimited number of undos?

We would modify the Piece class which is responsible for the chess units in the game. In addition to the current coordinates of the piece, add the coordinate of its previous position, and the pointer to the piece that is previous at its current position. And add an undo function to the class so every time the object calls it, it returns to the previous position and fill the current position with the previous piece. If the pointer to the previous piece is nullptr, just leave there empty.

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.

To implement a four-handed chess, a four-handed chess has a wider board. So we first need to add a four-handed chess version board to the Board class. And modify the Player class to add a four-player option inside the class. Also, we need to modify the display class so that it can render the four-handed chess correctly. Before the game begin, we give players an option to choose play alone or play in four-handed chess mode.