Plan of Attack

Danyang Wang, Arsen Cui, Ethan Zhang

We will begin working on the project by setting up a github repository, so the group members (Danyang Wang, Arsen Cui, Ethan Zhang) can work on it together by committing and pulling changes.

For the first stage, we will start off with implementing the decorator pattern from the UML which includes the Concrete Decorators (i.e. Pawn class, Knight class, Bishop class), the Decorator (Piece class), the Concrete Component (Square class), and the Component (Chessboard class). In addition, we will create a main class that will serve as a test harness, so that we can write test cases and test our classes used in the decorator pattern. The implementation for the first stage is expected to be completed by Dec 4th, and we plan to finish testing and fixing any bugs by Dec 5th.

For the second stage, we will implement the observer pattern from the UML which includes the Subject (Subject class), the Concrete Subject (Chessboard class), the Observer (Player class), and the Concrete Observers (Text-based Observer class and Graphical Observer class). Furthermore, we will implement features such as undo to make the testing process easier. The implementation of the second stage is expected to be completed by Dec 7th, and we plan on finishing testing and fixing any bugs by Dec 8th.

For the third stage, we will implement the strategy pattern from the UML which includes the Context (Chess class), the Strategy (Strategy class), and the Concrete Strategies (Level 1 class, Level 2 class, Level 3 class, Level 4+ class). The implementation for the first stage is expected to be completed by Dec 10th, and we plan to finish testing and fixing any bugs by Dec 12th.

For the last stage, we will put all the classes together and test the program as a whole. Additionally, we will document design.pdf and uml-final.pdf as well as provide a demo. Ideally, all of the stages will be completed by Dec 14th.

The distribution of the workload will be as follows:
For the first stage, Danyang will do the Concrete Decorators and Decorator classes, Arsen will do the Concrete Component and the Component classes, and Ethan will write the test cases as well as create a main class as the test harness.

For the second stage, Danyang will do the Observer and Text-based Observer classes as well as writing the test cases for this stage, Arsen will do the Subject and Concrete

Subject classes, and Ethan will do the Graphical Observer classes as well as adding features such as undo.

For the third stage, Danyang will implement the Level 1, Level 2, and Level 3 classes, Arsen will do the Context and Strategy classes as well as performing the testing for this stage, and Ethan will be implementing the Level 4+ class.

For the last stage, we will all be writing our respective parts in design.pdf based on which parts we implemented. Danyang and Ethan will work on uml-final.pdf, and Arsen will work on the demo.

CS246 Project: Chess

Arsen Cui, Danyang Wang, Ethan Zhang

Question 1

Standard opening sequences can be implemented by forcing the computer player to do a certain pattern of moves before it is allowed to make moves on its own. However, in order for the computer to have several different opening sequences they could do, you can't just hard code the exact same moveset for the first few moves (as there are different opening sequences). Thus, we can have the computer do the opening sequence in accordance to what the other player does. It will look at what move the other player makes before selecting a move from the different opening sequences available to it, in order to make the most optimal move (could be to protect its own king, or put itself in a position to check the opponent's king, or in a good position to take an important piece from the opponent, etc). Once the first few moves have been made, the computer can then continue the opening sequence based on whatever their first few moves were, narrowing it down to a specific opening sequence it has in its book of standard opening moves.

Question 2

Player moves can be recorded in vectors. Each player will have a corresponding vector to store their moves made starting from their very first move. This vector will act like a stack, where subsequent moves will be added on top of the stack (added to the end of the vector). Then, if we wanted to undo a move, we could just pop the top item off the stack, which would be the most recent move, using the vector function **pop_back()**. If we were to implement an unlimited amount of undos, we can repeatedly use the pop_back() function to remove moves from the stack.

Question 3

To make a 4 handed chess game, we would have to change a few of the original classes and functions, as well as add in some new ones. The changes we make to the original code is to implement a different sized board to play on, as 4 players' pieces need to be fit onto the board. We would also have to make the program support up to 4 players instead of only 2. As for new things we need to introduce, we must implement a Points system. 4-player chess works by calculating the total amount of points of each player at the end of the game to determine the standings. This would require a new class, we can call Points, which would have a generalization relationship with the parent

class Chess (refer to the UML diagram). In 4-player chess, when a player is checkmated or stalemated, their pieces become grayed out and nulled. Capturing those pieces provides no points. Therefore, we have to implement a method that causes a player's pieces to be worth 0 points once they are out of the game. This is because their pieces are still obstacles on the board, so we cannot just remove them entirely.