Sorcery Plan

Initial Plan Step-by-Step:

1.	Choose one game from the given three choices	*July 13th*
2.	Determine the basic classes needed for the game	*July 13th*
3.	Design an initial UML diagram for the game	*July 16th*
4.	Design the header files for each class	*July 16th*
5.	Implement all the .cc files for the header files	*July 18th*
6.	Create a Makefile to link all the files together	*July 18th*
7.	Implement our main.cc file and program the interactive features	*July 19th*
8.	Implement a demo for our game and test our game	*July 20th*
9.	Add bonus additional features to our game	*July 21st*
10.	Test the final game and debug if necessary	*July 24th*

Note: the dates listed are the estimated completion dates for each task.

Work Breakdown:

Yinghan Chen:

- Write part of the documentations *(question 1 and 2 in the initial plan)*
- Write part of the UML diagrams
- Write parts of the .h and .cc files
- Debugging the game

Xi Li:

- Write part of the documentations *(question 3, 4, and 5 in the initial plan)*
- Write part of the UML diagrams
- Write parts of the .h and .cc files
- Debugging the game

Dheeraj Pille:

- Write part of the documentations
- Write part of the UML diagrams
- Write parts of the .h and .cc files
- Debugging the game

Questions:

1. **How could you design activated abilities in your code to maximize code reuse?**

Since activated abilities have varying effects, we must be able to write each one as a unique function that has its own special effects and then assign that function to the appropriate card. We can do this by creating an ability function that take the name of an ability and various inputs and that function choses the corresponding function written outside. The inputs are given to the selected function to produce an effect. New functions and corresponding name can be added if there are cards with new abilities.

2. What design pattern would be ideal for implementing enchantments? Why?

A decorator pattern can be used for implementing enchantments since the decorators can be used to modify the existing minion class and stack each decorator over the other to compound the modifications in the order specified by the guidelines. The disenchant abilities will simply remove a “layer” of the decorators and the object will revert to a previous state without the outer most decorator.

3. Suppose we found a solution to the space limitations of the current user interface and wanted to allow minions to have any number and combination of activated and triggered abilities. How might you design your code and the user interface to accommodate this?

We put all the abilities (triggered/activated) into one file and we write a function which acts like a map. It takes a string as parameter and return an integer as index of the function. If we want limitless abilities in the minion classes, then we would use a vector to store all the indexes which are integers of the functions. Later on, when we use the ability of a minion, we would pass the value in the minion’s ability vector into the function then those ability functions will be called.

4. Suppose you chose to implement graphics as a bonus feature. How could you make supporting two (or more) interfaces at once easy while requiring minimal changes to the rest of the code?

We would implement an interface called graphics and make it be a subclass of board class. This class would have a Xwindow as its field. We would create an Xwindow in the main class and pass it to the graphics class. We may modify the board class and but leave other classes unchanged. Maybe we would use an observer pattern just like the previous assignment.

5. Describe your additions to the game. How do you expect them to affect your organization of Sorcery’s code compared to if they were not there? Which object-oriented programming concepts were used in your solution?

We would add a timer and display on the screen, or else the game might stay on one player’s turn forever. This will simply add an extra class called time and the board class may have a time class.

We also want to add a menu to the scoreboard so that the player would know what options he/she has. We still need to modify the scoreboard class, but we know we would have an extra class called menu.

The player would also have abilities, there are two abilities at all. Each player would choose one by chance. We would add an extra field in the player class and one extra public method called useAbility to use player’s ability. We would also modify the scoreboard class to let display the ability description.

More additions might be in the final version of our game.

The OOP concepts used in our solution are polymorphism, inheritance and encapsulation.