Plan of Attack:

1. Decide on the basic classes we will use in the program, and their high-level relationships.
Together - 5 hrs

2. Implement players (only having names for now), the game loop, and the -init command line argument.
Ye Qin - 2 hrs

3. Implement skeleton functionality to load decks from a file called default.deck.
Ville - 3 hrs

4. Implement abstract cards and the ability for a player to have a hand of cards, including giving each player a deck and the functionality to draw from that deck. Implement the ability for players to start and end their turn, including drawing a card at the start of their turn if their hand isn't full.
Ye Qin - 2.5 hrs

5. Implement minions with no activated or triggered abilities, and allow them to attack players (with no limit on the number of actions per turn). Keep in mind that they will need to be enchantable later.
Fondson - 2 hrs

6. Implement spells which interact with minions.
Fondson - 2.5 hrs

7. Allow minions to attack other minions.
Fondson - 1.5 hrs

8. Implement rituals and triggered abilities.
Fondson - 3 hrs

9. Implement simple enchantments, such as enchantments that modify the attack of a minion.
Ville - 3 hrs

10. Implement activated abilities.
Fondson - 3.5 hsr

11. Graphical UI (text and xwindow)
Ville and Ye Qin - 6 hrs

12. Implement details that have been left out thus far (magic, actions, etc).
Together - leftover time

13. Implement the more complicated remaining cards.
N/A

Question: How could you design activated abilities in your code to maximize code reuse?

Since spells and activated abilities can perform the same tasks, we can implement a HAS A relationship where spells HAS A activated abilities. We can also make the trigger abilities a subclass of activated abilities with a field called condition that marks the trigger condition. Then we set the cost of trigger effects to 0.

Question: What design pattern would be ideal for implementing enchantments? Why?

Decorator. We treat enchantments as decorators that wraps around a minion, while the minion is an abstract class with a concrete component (the minion without any enhancement).

Question: 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. What design patterns might help us achieve this while maximizing code reuse?

We could use the Decorator pattern to add any number of different combinations of abilities to a minion by wrapping it.

Question: How could you make supporting two (or more) interfaces at once easy while requiring minimal changes to the rest of the code?

We can support multiple interfaces by giving each component that can be drawn their own “draw” function/method so that if we wanted to display something different (say according to different players) we can just swap the components that’s being drawn.