# CS246 Plan of Attack: Sorcery

## Breakdown of the project

|  |  |  |  |
| --- | --- | --- | --- |
| **Project Task** | **Task Description** | **Due Date** | **Assignees** |
| Initial UML | - Decide on a class hierarchy  - Identify object-oriented pattern | November 21, 2017 | Jafer, Berges, Catalin |
| Complete interface | - All header files (.h files)  - All functions should be properly commented  -Files affected: **all header files** | November 22, 2017 | Catalin |
| Complete load function in main logic | - Loading of decks, cards, abilities from text file  - Involves creating test cards, minions, etc.  - Files affected: **main.cc** | November 22, 2017 | Berges |
| Complete constructors for each card type | - Taking in the string containing information about a card and initializing using that  - Files affected: **Minion.cc, Enchantment.cc, Ritual.cc, Ability.cc, TriggeredAbility.cc, ActivatedAbility.cc, Spell.cc** | November 23, 2017 | Berges |
| Implement all card type logic | - Cast function (applies the card functionality)  - Files affected: **Minion.cc, Enchantment.cc, Ritual.cc, Ability.cc, TriggeredAbility.cc, ActivatedAbility.cc, Spell.cc** | November 25, 2017 | Catalin, Berges |
| Implement text-based graphical interfaces using observer pattern and MVC | - Implement textdisplay  - Implement observer pattern, setting the Board as the subject and textDisplay as the observer  - Handle all overloaded operators for deck  - Files affected: **TextDisplay.cc, Observer.cc, Subject.cc** | November 25, 2017 | Jafer |
| Implement board, player logic and main game loop | - Implement the interactions between the Board and Players  - Implement all methods in Board  - Implement command loop in main.cc  - handle parameters specified in requirements  - Files affected: **Player.cc, Board.cc** | November 27, 2017 | Catalin, Berges |
| Implement graphics based | - Implement graphics display of the game  -Files affected: **GraphicsDisplay.cc** | November 27, 2017 | Jafer |

# Questions

Question 1: How could you design activated abilities in your code to maximize code reuse?

Our design method used to maximize code reuse for activated abilities relies on the decorator pattern. We have a decorator of type ability which will be exclusively attached to minions. This allows us firstly to attach numerous abilities to a minion if we decide to enhance the game to include this feature. Moreover, the activated ability class inherits from this, thus differentiating itself from the triggered ability class which also inherits from ability. Beyond this, the design allows for more classes to inherit from activated ability in order to implement new activated abilities in the future by simply writing a new “cast” function. This allows for the maximum amount code reuse since we can simply add a new activated ability with a new cast function and apply this to any minions.

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

The ideal design pattern for implementing an enchantment is a decorator pattern. This mainly because we can stack enchantments onto a minion, by using a decorator pattern we can apply multiple overridden cast functions onto a single minion. Then we can apply the changes from each enchantment and remove them dynamically. This, like our activated abilities class is also maximized for code use since we can add a new enchantment with a new cast function without changing our current code.

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

# UML