# Sorcery

## Design

**Wilson Chen**
**Ahmed Mohammed**

## I. Introduction

Sorcery is a terminal-based collectible card game inspired by titles such as Hearthstone and Magic: The Gathering. For our CS 247 spring 2025 group project, we implemented a full-featured version of Sorcery in modern C++, leveraging object-oriented design principles and patterns to maximize modularity, extensibility, and maintainability.

This final design document presents the actual structure of our implementation: explaining how our system meets all core requirements and any changes we've made to our original plan. We describe key design patterns used, discuss how our architecture anticipates change with minimal impact, and provide complete answers to the specification's design questions. Subsequent sections cover: an overview of the system structure, detailed design techniques, resilience to change, answers to questions, our reflections to the final questions, and a conclusion.

## II. Structure Overview - H

## III. Design - SH

## IV. Resilience to Change - H

## V. Questions

1. **How could you design activated abilities in your code to maximize code reuse?**
   Our implementation for this question is mostly accurate, but there are a few differences. To start, we do use the **AbilityCommand** abstract interface with **ActivatedAbility** as a concrete implementation and we function objects for flexible execution. However, we have previously stated that minions can hold on or more **AbilityCommand** objects, but that is not true. We realized that minions can only have one activated ability. Our **getAbilityCost(int abilityIndex)** method suggests support for multiple abilities, but the actual storage is a single unique_ptr.

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

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?**
   Multiple activated abilities for minions was not implemented and thus our answer does not differ.

## VI. Final Questions

What lessons did this project teach you about developing software in teams?

Building a bigger project like Sorcery taught us important lessons about developing software in teams. A vital practice we adopted was using branching. By creating separate branches for different features or fixes, we were able to keep our main codebase stable while still making progress. This also gave us a clear history of changes, which made it easier to trace and debug issues.

We also spent a lot of time resolving merge conflicts, which pushed us to be more careful about our changes. It taught us the importance of writing clear commit messages and regularly pulling from the main branch to avoid large, conflicting diffs.

Finally, we learned how to organise our work to avoid dependencies. By first dividing responsibilities, we avoided being blocked by each other's progress. This made our development process more efficient.

What would you have done differently if you had the chance to start over?

## VII. Conclusion - L