Application or use cases of various design patterns

Final Project

Lemichel Thibaut

GitHub Repository:
https://github.com/ThibautLemichel/FHWizardSpellManager.git

April 20, 2025

1 About the project

I chose to continue he project that I've started for the first assignment, So you may read some informations that you have already read.
I chose to create branch on for each pattern that I've added so you can have a better overview, notice that I've added tests to the project and refractored the code as you have mentionned in the first assignment review, but I did it quite late in the process and you will only see them in the prompt-orchestrator branch and improved in the proxy branch.
I've stupidly leaked the gemini API key, so you can easily find it by looking at the github commits however you'll be able to find it in the `Link_github.pdf` that is on moodle.

2 Description of the project

This project is coded in java and resolves one of the most relevant problem that every wizard have : How can they manage their spells ?. The Wizard Spell Manager allows any wizard to add, cast and undo a spell, while taking care that most of the spell don't last forever and automaticaly undo them. This project is based on various design patterns which garanty the maintainability and scalability of the project.

3 Main Features

3.1 Core of the project (from first assignment)

First it is possible to manage and add spells in a unique SpellBook. Then it is possible to cast and follow the state of a spell (recovery time after casting one). It is possible to undo a spell and some of them (FireBallSpell) have an active time, and will automatically be undo after the time is gone. Some spell (NecromancerSpell) must be undo by yourself. Easy to manage all the spell because of the message that is written when any change in spell's state.

3.2 AI features added

The user is now able to ask some questions to different AI models, the user can choose between a suggestion model, a description model and gemini API. The first two models are definetly not AI, they are just here to illustrate the project by simulating an AI response (they are just if else) .

4 Design Patterns used

4.1 Agent Pattern

This pattern is used to encapsulate specific behaviors and functionnalities into different agents that are independant, reusable.
So each agent has its own behavior that makes it easier to manage and extend, for example GeminiAgent has other method than the 2 other fake AI agent that is loading `API_KEY`, extracting the meaningfull data from the response and the resquest itself.
The system is easier to maintain because all agents are self-contained. And now when we want to add a new agent we are not affecting the rest of the system.

All the agents are coordinated by the orchestrator pattern.

4.2 Orchestration Pattern

The orchestration pattern is used to centralize and manage the different agents.
It handle the logic of how different component interact, therefore it is reducing the complexity of individual components. There is a seperation between the components that are doing their task and the orchestrator that is managing the workflow.
We are registreing the different models into the DefaultPromptOrchestrator. The orchestrator is choosing which model should be use using the input (suggestion, description or gemini).

4.3 Proxy Pattern

The Proxy Pattern is used to manage and optimize interactions with the models. It is an intermediary between the client and the model, it is caching the different request and response so that a request that have already been made is not done again, that could be really usefull when implementing a real external API, such as the gemini one here, that is allow us to save some money by not doing redundant request.
Proxy can preprocess the requests before sending it to the API by adding some other data to have a better response.
It could also handle the errors of the request by retying it.

4.4 Agnostic Model Pattern

The Agnostic Model Pattern is kind of the direct follow of the orchestration pattern and the agent pattern because we make all models interchangeable and decoupled from the rest o the system. We can now interact with each models in an uniform way.
It makes it really easy to add, modify and replace models without affecting the rest of the code.

5 Difficulties

The main difficulty was to find a proper way to implement AI in the project, because the core project didn't need AI in it. As it was difficult the only thing that was kind of relevant to add was suggestion for what spell that should be cast and to have a description of spells.
Additionnaly asking things to LLM here is definetly overkill, it is working if the user want to have a description of a spell, however to make some suggestion on what to do, it would be be better to create a custom AI using the data from the game as training, that could be stored from the users, training a tree model for example to have a better response. Here a proxy would be really usefull, it would add some data that could be useful for the model to have a more relevant response.