Ellis Fitzgerald
COS 420: Object Oriented Design
March 3, 2025
AI Lab Book #3

LLM: DeepSeek DeepThink R1
Time Spent Overall: 1 hour 20 minutes

For prompting DeepSeek in this assignment, I once again continued the chat I had with the LLM for previous assignments. This allowed some of the context from previous parts to be used and to save some time and effort in the prompting.

Get rid of "104"

# of Prompts: 0

Changing the references to the winning score value of 104 was an easy task as it was an adjustment/optimization I made preemptively after hearing the professor express the recommended approach early on in the course. This was something I manually replaced in my code files, so no prompts were necessary. The method I used was by searching and replacing occurrences of the number "100" and manually selecting those that related to the maximum score value and replacing it with a defined static constant `WINNING_SCORE` in the `Prog6` class and file.

Consolide randomness

# of Prompts: 1 necessary, 2 exploratory

To get DeepSeek to create the `Dice` class I simply paraphrased the original instructions on how to create the dice class with the given parameters. The code for the Dice class itself had no issues whatsoever, it is a trivial task after all. I simply created a new file for the class and pasted what DeepSeek had output. Even though in its first response DeepSeek properly created the class and intended to create a new Dice object per player, I was curious to see other ways DeepSeek would manage the reference to the dice. Knowing Java treats all class data members as references, and Bulldog is a turn-based game, I assumed that the dice *could* be a shared pointer to the same object. I would not know if this is actually a good idea in practice, but it seemed no matter what it would take more effort to create a new 6 sided die for each player. Especially because DeepSeek seemed to interpret my question and prompt as the intention and desire of a singleton. Therefore I did not go through with it. Implementing the calls to the Dice's roll function within each Player took a little more hands-on as most of DeepSeeks references to the Player functions were flat out wrong or simply had been changed between the last prompt I had given it and now.

Javadoc

# of Prompts: ~5

For this step I used a mix of autocompletion from my IDE of choice (IntelliJ) as well as DeepSeek. I found it to be quite convenient to simply type "/*" and press the enter key above a function, where IntelliJ would auto fill in "@param" and "@return" making it easier to fill in the blank. DeepSeek provided these tags around 50% of the time while the other 50% chose to omit them. My method for prompting with the JavaDoc was by copying a chunk of the code that was related in functionality and asking DeepSeek to provide JavaDoc comments above the functions. Because DeepSeek seems to prefer staying focused to smaller tasks I chose to do it in this divided way as otherwise it could be prone to hallucinations like seen before. I also did it this way because some of the code was split between files, and it did not seem like a good idea to join text from separate files together for a single prompt. For example, a few of my Player classes that I have been working with since DeepSeek wrote them initially still did not have JavaDoc comments, so pasting their entire definition into the prompt bar and asking it politely to provide JavaDoc comments for all of the functions worked well. It is a little harder to "grade" the comments of an AI as it has less functional effect and comes down to personal preference on writing styles.

File structure

# Prompts Used: 0

My file structure was for the most part quite consistent with the style guide/standard. However, there was a rule breaker for the NumericDocumentFilter class. Although I believe there might be an exemption here as it was a static class, I still wanted to take this instruction here as literally as possible. Given this part of the task required me to create files, I could not use DeepSeek here and just used my IDE to make a new Java class which consisted of the code already present in the project.

Overall thoughts

I think that DeepSeek offers quite a nice solution to refactoring or tidying up code. Asking an LLM to create something from scratch for something you are not too savvy on results in a cat and mouse chase between you and the desired result. It becomes hard to know what is wrong and what is right. Whereas with refactoring you go into the process with a specific intent and goal limited in size and scope. There is little room for improvisation which makes the process close (but not quite) deterministic. The key aspect of course is the fact that functionality is not expected to change, just how it is organized and managed under the hood. It seems boring or perhaps *less sexy* that the LLM is not writing blazingly fast, performant, genius code, but instead is making small adjustments and writing comments to make the code neater. However, this is where I believe working with an LLM feels the most comfortable. I do not feel like the code is getting ahead of me, but instead I am getting ahead of the obstacle in front of me. Which in a lot of cases is the time a task requires.