The Bulldog Improvements

Author:
Michael Yattaw

March 3, 2025

When I first began the Bulldog project, I started by reading the documentation and creating empty functions and potential constants to outline the project. To my surprise, this part of the project requires us to improve the code by eliminating the “winning score” magic number value “104” and replacing it with a constant. However, I had already done this because, as Professor David Levine pointed out, a constant can be modified during debugging, meaning you do not have to play a lengthy Bulldog match. On the other hand, when I used AI to create the Bulldog project, it did not implement a winning score constant. Instead, it produced a very simplified version of the code that still functioned. As a result, when I needed to debug the code, I had to manually replace all the “104” values with a smaller value manually.

For the next set of improvements in this assignment, I implemented the rollDie method within the Player class. Shortly after, I created a new object called Dice, which included a constructor with an integer parameter for the number of dice sides and a constant for the Random class in Java. This constant was used to generate a random number when rolling the die. The Dice class would generate a random number between 0 and the number of sides and then add 1 to the result, exactly the same way the code worked previously in all other classes. Therefore, I went through and refactored all my Player classes to use this new code. Next, I spent 10 minutes experimenting with the model to verify whether I had written my code properly. I prompted the DeepSeek model to create a Dice class and then compared it to my own code. The two were nearly identical, except that I had forgotten to include exception handling for cases where the die has fewer than one side. Thus, I incorporated this small change into my existing code.

After creating the Dice object, I decided to use AI tools to generate the Javadocs for this class, similar to what I had done for previous class files. I prompted the DeepSeek model with, “Create Javadocs and header comments following the style of this example class,” and then copied and pasted the Player class along with its header comments. As expected, the model was able to flawlessly add the Javadoc comments and header comments with proper descriptions in about 5 minutes of prompting. The model successfully described each function and even updated the specific header template text for the class files, making the process very straightforward.

For the final part of the assignment, we were asked to reorganize the file structure for better clarity and eliminate any potential inner classes. However, throughout the entire project, I had already separated my classes into their own files and packages. For this part of the assignment, I attempted to refine my package organization using the DeepSeek model to see if there was a better way to structure my classes. After about 30 minutes of prompting the AI, I had no success. I even tried to screenshot my class hierarchy and list each directory with its classes, but it did not turn out well, perhaps because my files were already organized. Furthermore, the model seemed to randomly delete class files during repackaging, such as the abstract Player class. For this reason, I decided to keep my original packages, as they were likely the best outcome I could achieve.