Bulldog: Using AI to Build a Java Swing UI

Author:
Michael Yattaw

February 22, 2025

Building a user interface (UI) with Java Swing requires extensive knowledge of the framework. For this part of the program, I attempted to convert a hand-drawn sketch of a potential Swing UI into a functional UI that integrates with the previously developed console application.

To begin, we collaborated in small groups during class, and my group generated ideas. Phil Lane then translated these ideas into a basic UI sketch, which I traced in Microsoft Paint. However, challenges arose when implementing the UI sketch in Swing. While Java Swing provides some straightforward layouts, significant manual coding was required at nearly every step of the process.

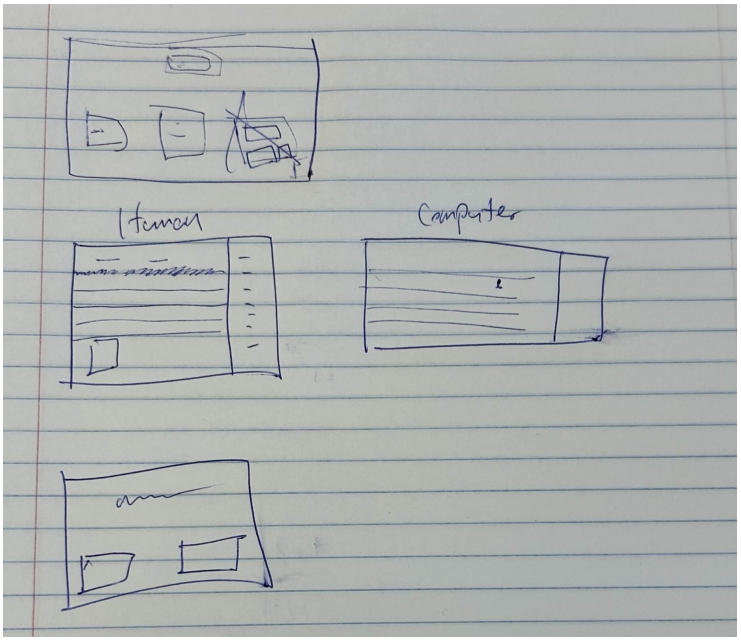

Figure 1: UI Sketch

When using the DeepSeek model, it was able to generate functional UI code, but it required several specific prompts, such as "Can you align the buttons at the top and bottom?" to fix minor issues along the way. As a result, the design process took approximately two hours of prompting and modifying the code.

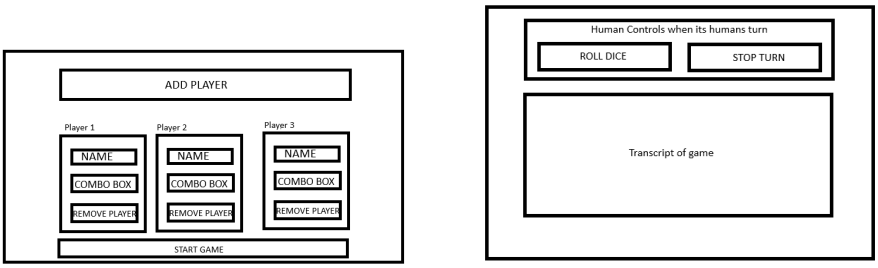

Figure 2: Sketches of Main Menu and Game Menu

The first prompt I fed into the DeepSeek AI was: "I've created a simple console application game in Java called Bulldog. The game involves multiple players, and the way you play is that each player gets a turn to roll a die until they choose to stop or they roll a 6, which counts as a zero for the round. There are multiple player classes that use different strategies, and now I would like to implement a Java Swing user interface based on the screenshot uploaded. I would also like the interface to be its own class because I will be changing the interface through game actions."

I then provided a screenshot of the UI traced in Microsoft Paint along with the current console application code. Instantly, the AI responded with functioning code; however, it did not follow the layout I had drawn in Paint. I then prompted the DeepSeek model to replicate the layout again, but I had no luck. I had to be more specific in my request, asking the AI to use a border layout with player panels inside the center of the border layout. This approach worked fairly well; however, I still spent almost two additional hours modifying the UI to match the style I believed best fit the screenshot.

After I had a working UI to select players, I needed to implement our other drawing, which was a UI for ongoing matches. When a user presses "Start Match," it switches the main menu panel to a game panel. So, I prompted the AI to keep a single JFrame but allow for different views corresponding to different match states, such as the starting game, ongoing game, and potentially the end of the game in the future.

Next, the AI successfully implemented this code on the first attempt, the UI displayed a transcript of the old console messages inside the transcript text area, which also contained timestamps that I was able to add easily. I also prompted the DeepSeek model to create buttons that appeared only when it was a human player's turn. These buttons were called "ROLL DICE" and "STOP TURN." However, I now had a new issue, which was the player still required console inputs. To fix this, I needed to implement basic roll dice and

stop turn functions specifically for human players. This part was simple enough without the use of AI. Since I was able to integrate this code into the UI class, I was also able to enable and disable these buttons as needed, such as disabling them when the game was over. At this point, I had a fully functional Bulldog application with a Java Swing UI.

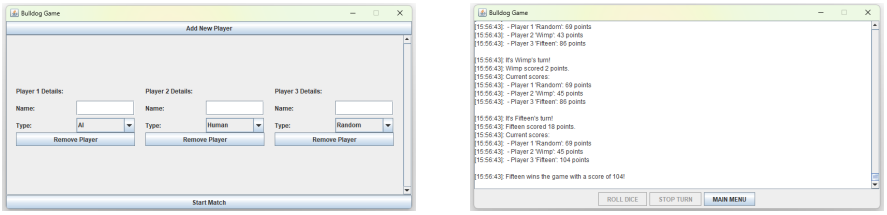

Figure 3: Swing UI Designs for Main and Game Menus

For the most part, I was satisfied with the style of my UI; however, I had the original problem similar to the first time using the AI model, which was that it produces functional code, but it does not provide flexible code. For a better example, the code was written in only a single class when it easily could be separated across multiple classes. Therefore, I prompted the AI to organize my code and to expand the game panels code into different classes. The prompt worked well, as it was able to create custom panel classes that extended JPanel to simplify and make the code much more flexible. So now, if any developer wanted to add more features to the application, it would be much more straightforward. Overall, this phase of the program took around 3 hours of prompting and another hour of debugging and modifying the code, which is far faster than I could accomplish on my own.