Bulldog: Models and Mimickry

Author:
Michael Yattaw

March 26, 2025

When starting the bulldog program, we started out with a basic layout of classes that contained player classes and simple data structures to handle participating players. However, as we built the program out, it became important to follow the principles of Model-View-Controller. For this part of the program, I needed to create a player model for the user interfaces to interact with, which is seen as the view, and the inputs are seen as the controller. To begin, I asked the DeepSeek model to create a player-model class that allows players to be added, scores set, and players to be retrieved by index. The prompt looked like this:

> "Create me a PlayerModel class that allows me to add players, set player score, and get player by index. This is a class to handle player data instead of using a ArrayList for accessing players. Here is my player class to reference:"

The next step was to create a view for the bulldog program to display live scores of each participating player. The model already had the previous prompt, so it knew how to integrate the PlayerModel class into the UI. Therefore, I was able to create a simple prompt that looked like this:

> "Create me a ScoreboardViewer using JFrame and my PlayerModel class. The class contains an update method that takes PlayerModel as a parameter and displays one name and one score per Player. Also demonstrate the program works by writing a main method that populates a game roster."

The prompt was very basic, but it provided me with a lot of code that saved me time. I spent only about 25 minutes total prompting, then I had to integrate the code into my main UI.

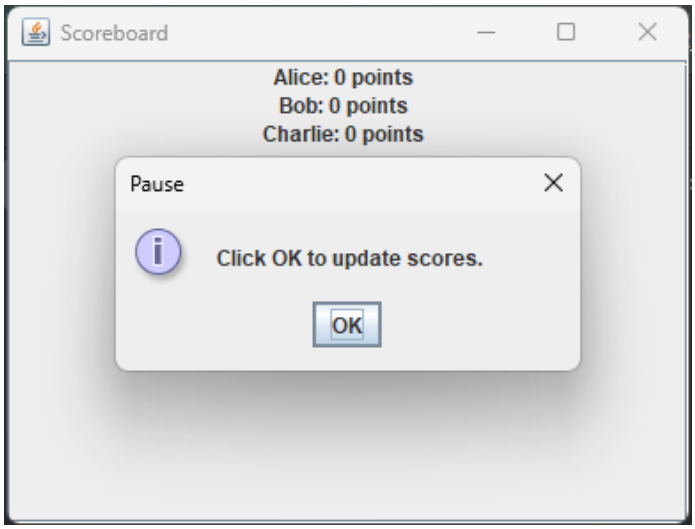

Figure 1: Scoreboard Viewer with JOptionPane dialog

To start the integration process of the new player model and scoreboard viewer, I decided to add a JCheckBox to the main menu user interface that allows users to enable live score updates. Next, I commented out every line that contains a list of players and placed the new PlayerModel object below it. I scanned the code for errors and easily updated the code manually to use my new model. The entire integration process took about 40 minutes because I have already have some experience refactoring Java code, and I believe it would have taken longer waiting for the DeepSeek model to process each class individually instead of manually integrating the player model. However, I did run into a few issues with my player model, which were easily solved by manually creating methods such as getting player count and getting players by names, which did not require the use of an AI tool to create.

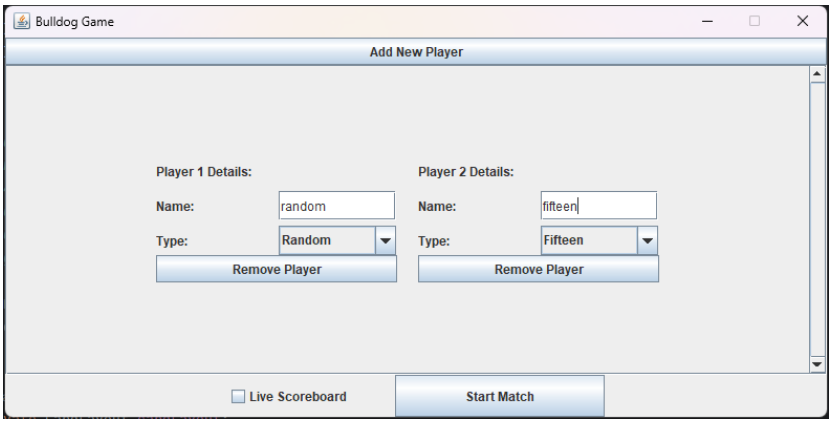

Figure 2: Updated Main Menu with Scoreboard Checkbox

My final thoughts on this program are that it is much easier to work with a data model when a program becomes larger because it is easier to retrieve

and modify the data of whatever object you are working with. Additionally, I believe that I could have structured the UI to be more consistent instead of using a new JFrame to view scores if we started out as an MVC application. Another useful feature I plan to add to this program is a custom component that displays player scores. So instead of a basic JLabel that contains player name and player score, I could possibly design a more visually appealing component that contains score, player name, current round number, and possibly a progress bar for game progress.