

MODELS AND MIMICKRY

Sarah Lawrence, University of Southern Maine

03/26/2025

Description

The goal of this is to add a scoreboard to the GUI for the Bulldog program using AI. The four steps used to achieve this were as follows. First, build a model, second build a viewer, integrate into Bulldog, and finally format into Javadoc. The model used to complete these steps is ChatGPT 3.5.

Building a Model

Building the model took over 6 minutes and 23 seconds to build the model with the AI. This includes bug testing. This model was a class that encapsulated an ArrayList of Players. The class adds a Player, finds the name and score of a Player using an index, and uses the index to set the score of the Player.

Success:

Overall, it was successful on the first run. There were some minor changes I would make but overall, the goal functions work. The goal functions are to add a Player, find the name and score of a Player using an index, and use the index to set the score of the Player. ChatGPT 3.5 used two classes to complete this: the Player class and the Playerlist. The Player class has the name of the player and gets/sets the player's score. Playerlist has an arraylist of players, adding a player to the list, getting the player's name using an index, and getting/setting the player score with the index. This and the prompts to the AI can be seen below.

Prompt:

Human: Design a class that encapsulates an ArrayList of Players. This class should enable a client to add a Player, to find out the name or score of a Player (by giving the index of that Player in the list) and to set the score of a Player (by providing the index of that Player and the new score value.) The class should also have other data and methods to allow it to function in an "MVC world."

ChatGPT: Here's a Java class that encapsulates an ArrayList of Player objects. This PlayerList class serves as the Model.

Some of the Player class:

```
import java.util.ArrayList;
import javax.swing.JOptionPane;
import javax.swing.*;
import java.awt.*;

// Model class for Player
class Player {
    private String name;
    private int score;

    public Player(String name, int score) {
        this.name = name;
        this.score = score;
    }

    public String getName() {
        return name;
    }

    public int getScore() {
```

(This code keeps going)

Some of the Playerlist class:

```
import java.util.ArrayList;

// Model class for PlayerList
class PlayerList {
    private ArrayList<Player> players;
    private ScoreboardViewer viewer;

    public PlayerList() {
        players = new ArrayList<>();
    }

    public void addPlayer(String name, int score) {
        players.add(new Player(name, score));
        if (viewer != null) {
            viewer.updateView();
        }
    }

    public String getPlayerName(int index) {
        if (isValidIndex(index)) {
            return players.get(index).getScore();
```

(This code keeps going)

Improvements:

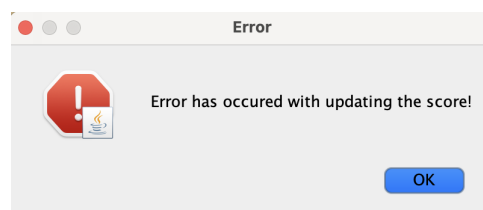
I would only add one improvement so far. That improvement would be to change the error message in `setPlayerScore`. I would like to change it from a print statement displayed in the terminal to something that would be displayed for the player. For example, right now the code looks like this:

```
public void setPlayerScore(int index, int score) {
    if (isValidIndex(index)) {
        players.get(index).setScore(score);
        if (viewer != null) {
            viewer.updateView();
        }
    } else {
        System.out.println("Invalid index. Cannot update score.");
    }
}
```

I would instead change it to this:

```
public void setPlayerScore(int index, int score) {
    if (isValidIndex(index)) {
        players.get(index).setScore(score);
        if (viewer != null) {
            viewer.updateView();
        }
    } else {
        JOptionPane.showMessageDialog(null,
            "Error has occurred with updating the score!",
            "Error",
            JOptionPane.ERROR_MESSAGE);
    }
}
```

This is what it would look like visually:



This helps the player know there's a problem and beware proceeding. This could also benefit a programmer making it easier to debug.

Build a Viewer

Building the viewer took over 14 minutes and 22 seconds with the AI. This includes bug testing. This viewer has a main method that initializes the game roster, displays the initial score values, shows a OK-only JOptionPane dialog to pause the score change, and finally updates the score and closes the dialog box.

Success:

Like the building of the model, the viewer was successful on the first run. After looking over the work I wouldn't change anything. The goal functions work and there were no improvements I could see. The goal functions include having a game roster (scoreboard) with player names and scores, OK-only JOptionPane dialog to pause the game, and finally if the player hits OK it updates the player's score.

ChatGPT 3.5 used two classes to complete this: the ScoreboardViewer and the Main class. The ScoreboardViewer class had the viewer (visual scoreboard) and the update viewer.

Prompt:

Human: Write a main method that populates a "game roster" (including both names and scores) and then display the values; your main program should then put up an OK-only JOptionPane dialog (to cause a pause) before changing the score of a Player. If all works as it should, clicking OK in the dialog will cause the Model to change, triggering a (visible) change in the Viewer/Scoreboard.

ChatGPT: This implementation includes a ScoreboardViewer class that updates dynamically when scores change. The Main class demonstrates the functionality, showing a scoreboard and updating it after a JOptionPane dialog.

Some of the ScoreboardViewer class:

```
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;

// Viewer class for Scoreboard
class ScoreboardViewer extends JFrame {
    private PlayerList playerList;
    private JTextArea displayArea;

    public ScoreboardViewer(PlayerList playerList) {
        this.playerList = playerList;
        this.playerList.setViewer(this);
    }
}
```

```

setTitle("Scoreboard");
setSize(300, 200);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLayout(new BorderLayout());

displayArea = new JTextArea();
displayArea.setEditable(false);
add(new JScrollPane(displayArea), BorderLayout.CENTER);

updateView();
setVisible(true);
}

public void updateView() {
    StringBuilder sb = new StringBuilder("Player Scores:\n");
    for (Player player : playerList.getPlayers()) {
        sb.append(player.getName()).append(" - ")
          .append(player.getScore()).append("\n");
    }
}

```

(This code keeps going)

Main class:

```

import javax.swing.JOptionPane;

// Main class to demonstrate functionality
public class Main {
    public static void main(String[] args) {
        PlayerList gameRoster = new PlayerList();
        ScoreboardViewer viewer = new ScoreboardViewer(gameRoster);

        gameRoster.addPlayer("Alice", 10);
        gameRoster.addPlayer("Bob", 15);
        gameRoster.addPlayer("Charlie", 20);

        JOptionPane.showMessageDialog(null, "Click OK to update
            Bob's score.");
        gameRoster.setPlayerScore(1, 25);
    }
}

```

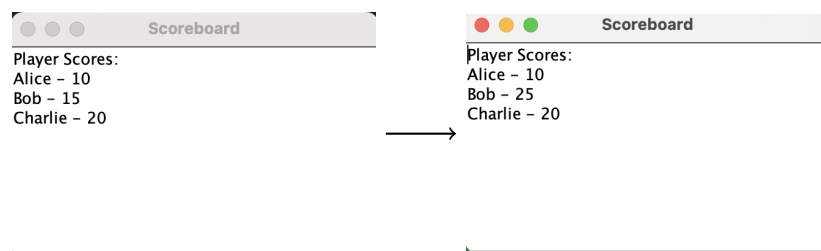
While looking at the AI's code I did compare it to code provided by David Levine. This helped me see if the AI was on the right track or not. For example, the Main classes in mine and David Levine's code both use a controller. The MVCMain uses "Controller" while Main uses "PlayerList".

After comparing the codes, I checked for any improvements. The only one I would consider is formatting but that will be corrected later. For now, I theorize this code is good or that my lack of knowledge of Java Swing is hindering my ability to improve this code.

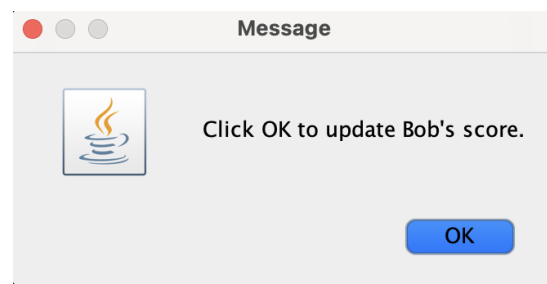
Results:

I can't provide improvements, but I can provide results.

This is an image of the scoreboard and what it looked like when it updated:



This is a image of the OK-only JOptionPane:



The OK-only JOptionPane works as intended and updates Bob's score from 15 to 25 successfully.

Integration into Bulldog

Integration of both previous steps into Bulldog took over 40 minutes and 3 seconds with the AI. This includes bug testing. This implementation uses the Model rather than the ArrayList of Players, and it integrates the Scoreboard into the Bulldog GUI.

Problems:

Originally, I was trying to keep all the prompts in one conversation with the model. This is so it could build off of its prior knowledge. I ended up going down the wrong direction multiple times. This caused the model to keep using the wrong information.

One of the unsuccessful Prompts:

Human: what is this error `java.desktop/java.awt.EventQueue.dispatchEvent(EventQueue.`

ChatGPT: The error trace you provided seems to be related to the event dispatching in the `BulldogGameGUI` class...

This kept going for a little bit till I figured it out on my own. Overall, what made this a struggle was fixing code I'm still not entirely proficient at. I kept running into errors I was unfamiliar with with code that I didn't make myself making it hard to figure out what was happening. In total, I took 33 minutes and 12 seconds trying to fix everything. I got tired of hitting a wall, so I decided to scrap serval conversation with the model and start over.

Success:

With the last new conversation, it was successful on the second run. This is due to a tiny undefined method getTurnTotal() error. The goal functions work ended up working fine. The goal functions are, using the Model rather than the ArrayList of Players, and it integrates the Scoreboard into the Bulldog GUI.

ChatGPT 3.5 implemented a viewScoreboardButton and made it so that if the total was calculated it showed on the board.

Successful Prompt:

Human: My model is "playerlist code" I want to change the BulldogGameGUI to use the Model rather than a specific ArrayList of Players. "BulldogGameGUI code".

ChatGPT: To modify your BulldogGameGUI to use the PlayerList model instead of directly managing an ArrayList<Player>, we can refactor the code accordingly....

Edited part of the BulldogGameGUI class:

```

JButton viewScoreboardButton = new JButton("View Scoreboard");
gameScreen.add(viewScoreboardButton);

viewScoreboardButton.addActionListener(e -> {
    // Check if the scoreboard is already added
    if (mainPanel.getComponentCount() == 0 ||
        !(mainPanel.getComponent(mainPanel.getComponentCount()
            - 1) instanceof ScoreboardViewer)) {
        // Create the scoreboard viewer panel dynamically
        JPanel scoreboardPanel = new JPanel();
        ScoreboardViewer scoreboardViewer = new
            ScoreboardViewer(playerList); // Pass PlayerList
            to ScoreboardViewer
        scoreboardPanel.add(scoreboardViewer);
        mainPanel.add(scoreboardPanel, "ScoreboardScreen");
    }
    cardLayout.show(mainPanel, "ScoreboardScreen");
});

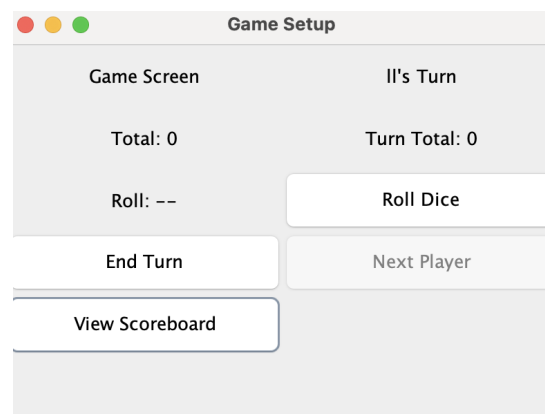
```

(This code keeps going)

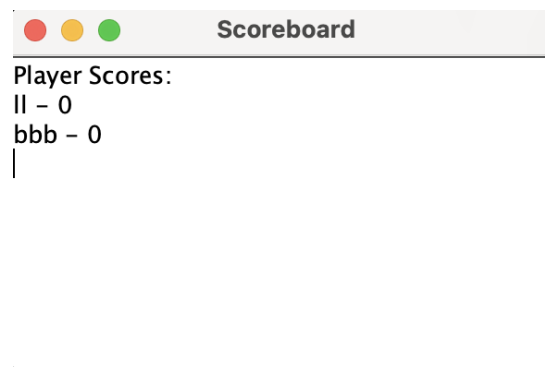
Results:

The results were as expected and are as follows.

This is the game board: (where the dice rolls take place).



This is the scoreboard: (where the user scores are shown).



Overall, after the total is calculate it would update the Scoreboard.

Javadoc

Putting the code into Javadoc style took over 4 minutes and 6 seconds with AI. The headers "Written by Sarah Lawrence, with the assistance of Robby the Robot." were done by hand.

Success:

The processes were to do one class at a time and ask the model to format the code in javadoc. The model had no problems completing the tasks. Here is some example code showing the successes of the model.

Main class Example:

```
/**
 * Main class for the Bulldog game.
 */
import java.util.Scanner;
```



```

import java.util.logging.Logger;
import javax.swing.*;

/**
 * The Main class initializes and runs the Bulldog game.
 */
public class Main {
    private static final Logger logger =
        Logger.getLogger(Main.class.getName());
    private PlayerList players;
    private Scoreboard scoreboard;

    /**
     * Constructs a Main game instance.
     * @param numPlayers The number of players.
     * @param scanner The scanner object for user input.
     */
    public Main(int numPlayers, Scanner scanner) {
        this.players = new PlayerList();
    }
}

```

(This keeps going)

I did this because in another experiment I tried asking the model to repeat the same step of changing the code to Javadoc. But I did it without specifying the step each time. When I did this the model had problems repeating the task. Further studies are needed, but this little experiment for now shows that the new method is more successful at the repetition task than the old one.

Conclusion

Altogether, ChatGBT did well in implementing all the steps. The one thing it had the most trouble with was when I sent the wrong code. I learned that the model has trouble redirecting when it's been given the wrong task. However, it did well with the improvements, putting all the code in javadoc, and building a small model and viewer. Overall, the steps were completed successfully with the combined effort of humans and AI.