Ellis Fitzgerald
COS 420: Object Oriented Design
February 22, 2025
AI Lab Book #2

LLM: DeepSeek DeepThink R1
Time Spent Overall: 4 hours and 23 minutes
Time Refactoring/Fixing LLM Code: 3 hours and 46 minutes

In our fourth program we were tasked to convert our Bulldog game into a GUI with Java Swing and our chosen LLM. Using the sketch I made independently in our lecture the past week, I got started by continuing the original chat from our first task of writing Bulldog's logic so the LLM could grasp onto that same context.

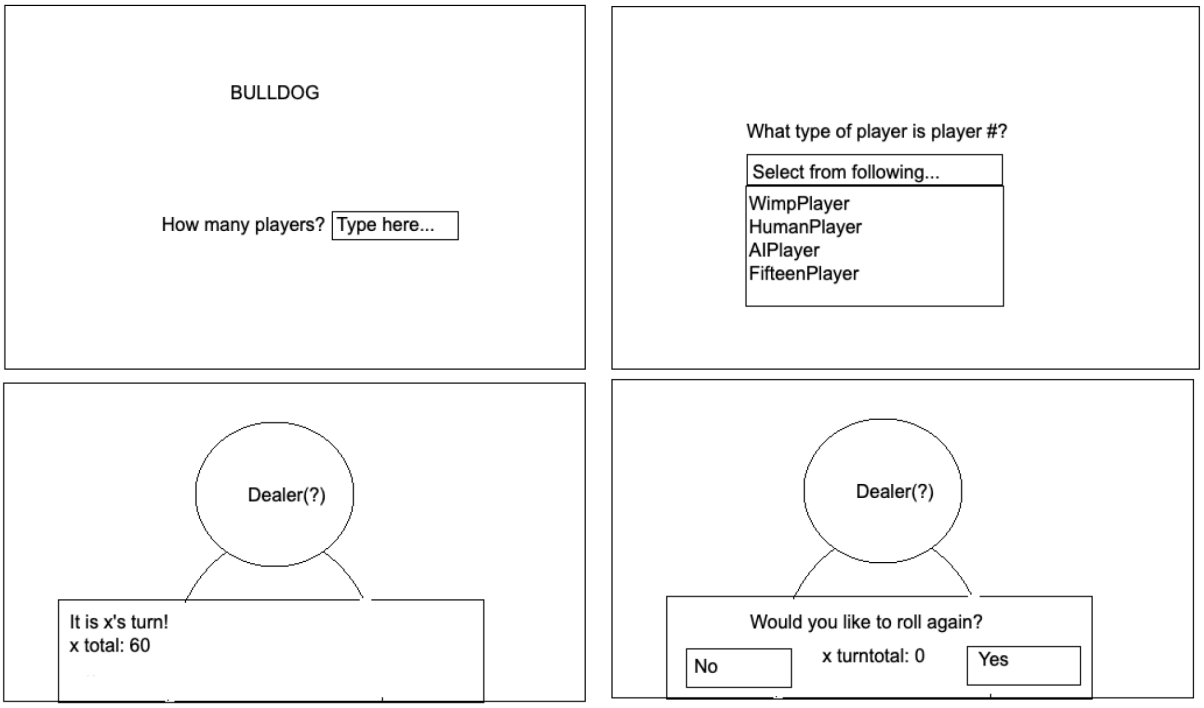

Figure 1. The microsoft paint sketch used as the blueprint.

The initial prompt was quite simple. I asked the LLM to create a title screen which contained the text "Bulldog" at the top, the text "how many players are playing?" in the center, and then an input field for typing the number of players also in the center. Recalling the issues with the previous version of the program not requiring numerical characters for certain actions I also asked DeepSeek to require only integers in the text field. It ended up creating a child of a Java class known as DocumentFilter which uses regular expressions to prevent certain characters from being in a document, which in our case was the JTextField. This was a solution and method

I was not familiar with, so it demonstrated how the LLM can provide knowledge and implement it simultaneously.

My prompt did ask for transitioning to a screen that would be used for player "creation" by listening to the event in the input text field. However, it seemed DeepSeek improvised (or perhaps ignored) by adding a start button that closed the window and continued the game back in the terminal/console. I was tempted to manually swap out the start button for the JInputField enter event, but then I realized that there was no issue with having both to make the experience more user friendly and flexible. I applaud DeepSeek for its suggestion and implementation of the button for this reason. I did however change the font and eyeball the size of the button to be more aesthetically pleasing.

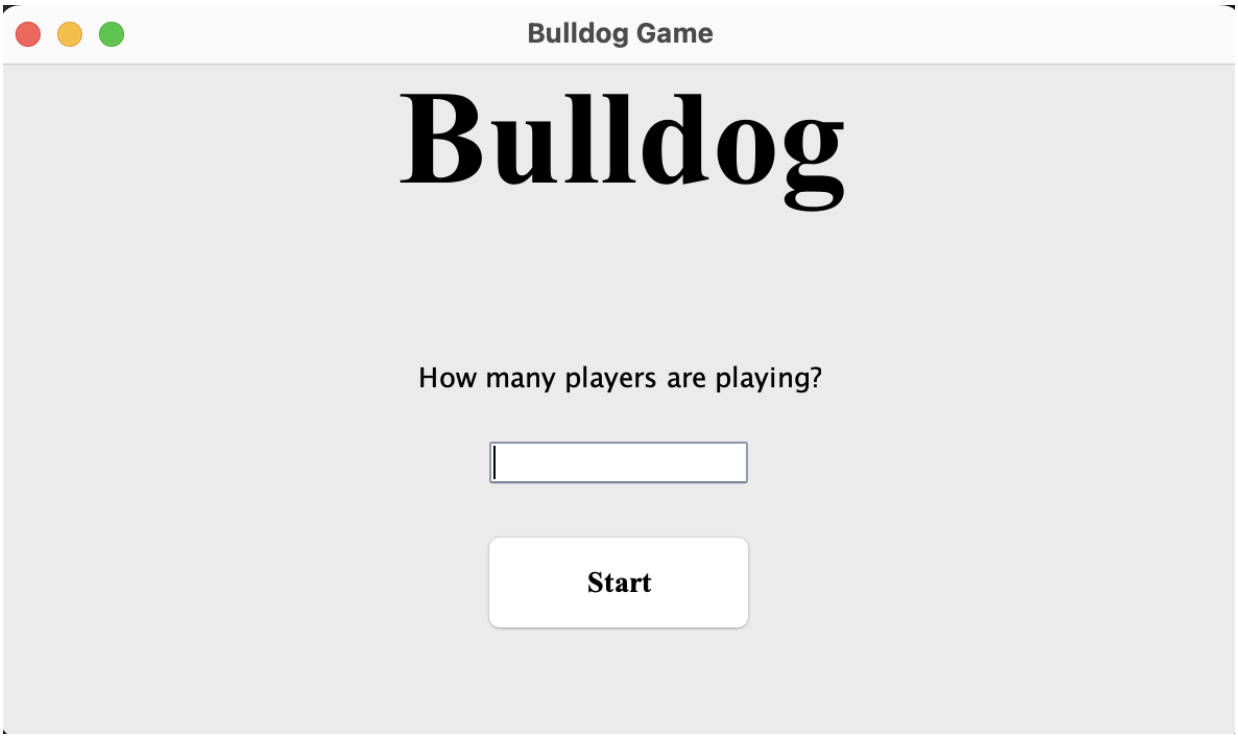

Figure 2. The result title screen

The next prompt I gave was for DeepSeek to add the player selection screen. Although I specified when the GUI should transition between game states, I did not specify how the player selection screen should become visible, let alone how it should look. Yet, the LLM properly opted for the cardLayout. Though the way DeepSeek went about adding cards to the JPanel was hard to understand. It seemed like the LLM could not decide on how many different functions to use and what their responsibilities should be resulting in an indistinguishable pattern. Similar to how there is no perfect formula for defining objects and their responsibilities, it feels as though the LLM was having trouble defining a function and its responsibilities. This should come as no surprise given there was no example to follow. (*Using chain-of-thought prompting or multi-hop-reasoning were things I should have invested more energy into for this assignment, but*

I digress). DeepSeek decided to write a function which instantiated and returned the title screen, but within that function it created the player selection menu. Adding insult to injury, it also wrote and added a "temporary" player selection `JPanel` in that same function which was never referenced in the whole program.

```
// Somewhere in a GUI "setup" function
cards.add(createTitleScreen(), "TitleScreen");
cards.add(new JPanel(), "PlayerSelection"); // Temporary
. . .
// Somewhere in createTitleScreen
cards.add(createPlayerTypePanel(), "PlayerScreen");
```

This type of code could be expected if each of my prompts demonstrated someone who could not make up their mind, but this was the response to a single prompt. This also showcases how LLMs simply do not have "memory" in the same way that we do, because it did not revisit the "temporary" line once completing its replacement. The game panel was no different in that the way DeepSeek instantiated it was completely unexpected and confusing. Regardless, by the end of the assignment I rewrote the code to be more simplistic by creating helper methods for each card and adding them to the JPanel all in the same function like so.

```
// Initialize panels
cards.add(createTitlePanel(), "TitleScreen");
cards.add(createPlayerTypePanel(), "PlayerSelection");
cards.add(createGamePanel(), "Game");
```

From this point on, I inferred I was not being as descriptive or decisive with my prompts as I should be. Though I desired to pretend that I had no knowledge of how Java Swing worked to test DeepSeeks limits, I needed to specify which Java Swing components and containers I wanted to use so the LLM would not have the freedom to improvise. Who would've thought an AI with free will would be a problem. It is not like almost every movie ever about AI shows them turn evil or produce something equivalent (evil code.)

For the player selection there was little to no issue with its structure which can be seen in **Figure 3**. This is mostly due to the fact that the logic was already contained in the main Java file from previous assignments. I would consider changing two things in the future. First, I may use radio buttons instead of the dropdown menu, but I could imagine that being an issue in the future if too many strategies are created. Second, I would prefer having the name of the player be an input text field right next to the drop down menu. DeepSeek instead opted to put a dialog with an input text field popping up after selecting the "Next Player" button. I asked DeepSeek to prevent players using the same name (since I have a history of that causing issues in my code), but it never got to it(?) Instead I wrote my own `isDuplicatedName` function which looped over

the existing players and compared the input string to their names. In the future I would consider storing Players in a HashMap so this process would take less operations. I also made it so the input text field would have "Player {number}" as the default text in case the user wanted to be lazy.

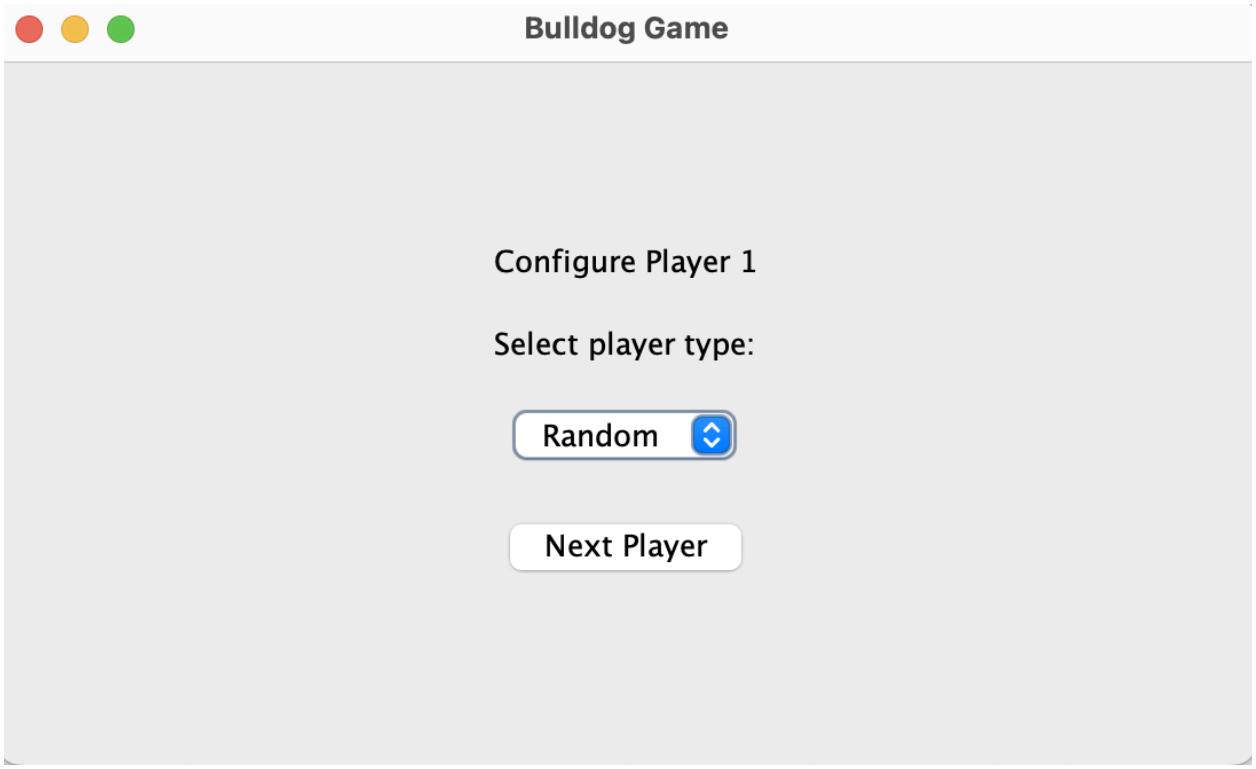

Figure 3. Player selection screen

In addition to specifying what Java Swing components and containers I wanted DeepSeek to use, I also decided to use naming abstractions so the LLM could associate a name with a functionality and intention. My best example of this was for the "Game" panel. If you look at **Figure 1** you will see the bottom left and right boxes visualize a character behind a box that contains text and sometimes a dialog intended for a `HumanPlayer` to interact with. Since I wanted this panel to serve multiple purposes, I decided to refer to it as the "visual console." Using this name allowed me to prompt DeepSeek to write the program without it getting confused as it did earlier on how to divide up the responsibilities. It seemed to be slightly beneficial, though there were two issues. Firstly, using the term "console" seemed to influence the LLM to associate its functionality with an actual console you would see in a development environment. I was intending just a simple text box that would constantly have `setText` called on it. You could attribute this to my word choice, but it is quite literally impossible to determine if the LLM, which is not deterministic and is communicated to in a non-context-free language, would behave in the way I intended if I had just used different words. Secondly, the dialog of "yes" and "no" buttons I intended to use for `HumanPlayer` were being shown at all times.

This resulted in unexpected concurrent behavior if pressed when another player was in the midst of a turn and overall visual confusion. This was something I descriptively asked DeepSeek to hide when it was the turn of a non-human player.

Eventually this "iterative process" began to feel just like how untangling christmas lights is an "iterative process." I realized DeepSeek forgot about our `Player.play()` methods and instead rewrote the entire player turn logic in a single `static` function with an if-ladder using `instanceof`. At that rate, we might as well use C and have player types be character arrays if we do not care about objects and their methods. Despite the fact that I would not mind the simplicity of a procedural program like that, this is an Object Oriented Design assignment tasked in perhaps the most Object Oriented language of all time. I would expect the LLM who wrote the beginning of the program to associate these concepts closely in their weights. Nevertheless, I acknowledged my own mistakes and decided to pour more descriptions into the cauldron.

Despite having been given little guardrails on how we should have designed our program, I knew our program was something we were expected to work with in the future. Therefore, I suggested to DeepSeek in order to use our objects methods whilst still having our GUI we take some inspiration from the Model View Controller structure. DeepSeek then responded with an interface called GameEventListener intended to be implemented by our main program class which contained some functions that should be called once a player had decided to end their turn, when they rolled, when they won, etcetera. The idea was that each of the players would have a data member of a GameEventListener. However, DeepSeek made some lapses in its judgement. The interface contained non-static functions, but most of the current program was carried out using static functions. This resulted in logical error as a static function attempted to call a non-static function given no instance. I asked DeepSeek to fix this issue, which required a lot of hands on to change almost everything to not be static. Looking back, I would have simply had the player classes just reference static functions instead: `Prog4.OnEvent()` as this would double in preventing cyclic dependencies.

I never got around to creating a texture that represented a "dealer" of some sort, but I thought the concept could maybe be interpreted rather inappropriately and I was plenty occupied with what DeepSeek had given me.

Figure 4. The game screen demonstrating the invisible dealer and the visual console

The biggest lesson I learned this time around is to continue to not assume. Larger prompts with repeated information and clarity is preferred for both the LLM and the developer. Otherwise you run the risk of deviating too far from the overall intended goal, circling in a specific issue which swallows the context window of the LLM. Next time, I would suggest reviewing the containers and components we learned about, labelling my sketch with each of their respective position and component type, and then formatting a prompt in an HTML-like structure to communicate the relationship between objects. Then of course, using additional text to describe the interactions between objects in respect to certain events.

Despite the complications, DeepSeek still provided knowledgeable solutions to real problems that I would have not known how to execute. Such as the DocumentFilter and using a scroll box for the visual console. Similarly how pair programming influences better productivity, once understanding the nuances of an LLM and its behavior, I believe you could similarly boost productivity when working with one.