When creating the initial version of Bulldog we were asked to implement a few basic Player types and the game loop logic. The idea was to keep the instructions and the respective program minimal too. This allows us to improve the program as time continues, with both our human brain and our LLM counterpart. As mentioned in the ReadMe, the LLM used for this repository is DeepSeek DeepThink R1.

One of the main differences between the code I had written myself and the code that the DeepSeek LLM provided for me was the fact that the process was not iterative to the same extent. For example, when writing the basics of Bulldog, I had to do one player at a time. Although DeepSeek demonstrated step-by-step "thinking," its output was the full program. Because of this, there was no need to independently debug each player and verify their validity. The entire program could (and was debugged) at once. When writing code yourself, it is easy to stop in your tracks and debug. Meanwhile, when a LLM gives you the entire codebase you request, it is much easier to just debug it at once.

Despite the whole program being done at once with DeepSeek, there were significantly less bugs. This I believe relates to the paragraph above. Because the whole program was given in one output there was no context lost. The program was "designed" by DeepSeek from entry to exit. To be fair, the bugs I ran into when writing Bulldog was due to my lack of understanding of the side-effects that stem from

```
Scanner sc = new Scanner(System.in);
sc.close(); // Good luck making a Scanner with System.in after this…
```

As well as

```
sc.nextInt(); // Need a sc.nextLine() to clear the buffer!
```

These bugs in particular were also the reasons why my version took much longer than the LLM. When I prompted DeepSeek to create Bulldog with DeepThink it mentioned to itself how sharing a `Scanner` or not closing the `Scanner`'s with `System.in` was necessary for HumanPlayer to work properly. The feelings that this brought out of me were conflicting: pleased but also a little threatened.

Another interesting thing I noticed about the LLM's program was how it handled the selection of player types. There is no safety checking for the console input to prevent the user from typing non-integer characters. This results in the program erroring and exiting. Albeit the program is simple in its current state, I still feel as though my implementation is more friendly to the average user by instead verifying the selection can be made (though it still does not prevent alphabet characters). It would make sense that the LLM would need further instruction for understanding nuances of human interaction as it can not process logic in the same way as it processes tokens.

The two biggest issues I had working with the LLM were not directly its fault. The first being DeepSeek's inaccessibility due to the servers having trouble once I asked for better commenting. The second being my own modifications to the program in order to get `UniquePlayer` and the respective LLM version that I call `AIPlayer`. For the player selection, in order to get both `UniquePlayer` and `AIPlayer` to work I simply added an

additional `case` to represent a sixth option, and reformatted some display prints. However, with my new `case` I made a lapse in judgement and forgot the `break` statement that Java requires. This resulted in the `default` case executing on top of a `AIPlayer` selection which as mentioned by the Professor creates a `WimpPlayer`. I embarrassingly did not catch this, and admittedly it was not easy to catch as the names for both the selected `AIPlayer` and the default `WimpPlayer` become the same. I think there are really two (2) interesting things about this bug:

1. This shows the challenge of working with a LLM and it demonstrates why it's important to still have an understanding of the program being written. After DeepSeek generated the program in its entirety I had to absorb what it had done and modify the code. It could be compared to being a benchwarmer and randomly getting swapped for the best player on the team. If I was paying attention on the bench, maybe I would have caught this. If I am dilly-dallying (assuming the LLM will do exactly what I intend) you are bound to run into some problems.
2. This is the antithetical of what programming *has* been (for the most part). One thing I remember hearing early in my decision to be a Computer Scientist was: “a computer program does exactly what you tell it to do” which is kind of a double edged sword in that what you wrote and what you intend are not always the same. Understanding how the computer interprets your instruction is one of (if not the most) valuable skills for programming. A LLM does not always do what you tell it to. The language we prompt it with is our mothers tongue which requires infinitely more context than a programming language. What your instructions mean depends on different interpretations. Additionally, a LLM is not deterministic. They contain different weightings from different datasets and temperature parameters that enable variable answers to the same prompt. This all resulted in DeepSeek deciding to add something I never mentioned. It improvised using `WimpPlayer` as a default. This is unexpected behavior. The computer did what I told it to… and more? This shows that with the current design and concept of what a LLM represents, we cannot treat it as coding in natural language, but instead communicating with an entity that can code. Because just like ourselves, the LLM has room for interpretability and “expression.”

Some additional small differences (and similarities) were things such as no score constant being defined. Much like my original program but opposite of a recommended practice. For player classes, the LLM opted for using a `while(true)` for their `play` method and breaking when necessary. I instead opted for a `do while(true)` since players always play at minimum one turn. I prevented myself from refactoring the condition in the loop to continue a `continueRolling` boolean or function, which all `Player` classes could override because we were told to keep it as simple as possible. ]

While on the topic of `while` loops (pun unintended). I also noticed that the LLM included a “game over” boolean that had essentially no use because as soon as it was set to true the loop was broken anyways.

The final difference I want to cover between the LLM and my own version of Bulldog I want to mention is the difference in console output. The LLM formatted the print statements to be more pleasant. With delimiting characters and indenting. When I input my `UniquePlayer` it surely felt humbling to see my primitive formatting.