AI vs. Human: Coding the Bulldog Game

The Bulldog game is an interesting programming challenge that makes you think critically. For this experiment, I coded one version of the game by myself and then used AI (ChatGPT-4) to generate another version. The goal was to compare the efficiency, quality, and approach between my own coding and AI-generated code.

At first, I thought AI would be much better than me at writing code without mistakes. In some ways, that was true, but what surprised me was how much effort I still had to put in to make the AI-generated code match what I wanted. It was not as simple as letting AI do all the work.

Code Structure: Step-by-Step vs. AI's Organized Style

One of the biggest differences between my code and AI's code was how structured they were. My version followed a step-by-step approach, making it easier for me to understand and debug. AI's version, on the other hand, was more structured and consistent but sometimes too complex for no reason.

For example, in the FifteenPlayer class, my version printed every roll, showing the decisions to continue or stop as shown in (*figure1*). AI's version skipped some of these details as shown in (*figure2*), making it harder to follow what was happening during a turn. There were differences across other parts of the code too but in general, AI's version looked more professional and somewhat complex, but my version was simpler and made it easier to understand what was going on.

```
Enter the number of players: 2
Choose player type for player 1:
1. HumanPlayer
2. RandomPlayer
3. FifteenPlayer
4. UniquePlayer
Enter choice: 1
Enter player name: human
Choose player type for player 2:
1. HumanPlayer
2. RandomPlayer
3. FifteenPlayer
4. UniquePlayer
Enter choice: 3
Enter player name: fifteen
It's human's turn.
Rolled: 1
Turn score: 1
Continue? (y/n): y
Rolled: 6
Turn over! Score for this turn: 0
   human's total score is now 0.
It's fifteen's turn.
Rolled: 1
Turn score: 1
Decided to continue.
Rolled: 6
Turn over! Score for this turn: 0
   fifteen's total score is now 0.
```

-figure1-

```
Enter number of players: 2
Enter player 1 name: humaai
Choose player type: (human/random/fifteen/unique-me/unique-ai/wimp)
human
Enter player 2 name: fiftai
Choose player type: (human/random/fifteen/unique-me/unique-ai/wimp)
fifteen

humaai's turn
humaai rolled: 2
Continue rolling? (yes/no): no
humaai total score: 2

fiftai's turn
fiftai rolled: 6
fiftai lost all points this turn!
fiftai total score: 0

humaai's turn
humaai rolled: 5
Continue rolling? (yes/no): no
humaai total score: 7

fiftai's turn
fiftai rolled: 5
fiftai rolled: 5
fiftai rolled: 3
fiftai rolled: 1
```

-figure2-

Speed vs. Accuracy: AI is Fast, But Not Always Correct

AI definitely had a big advantage in speed. While I spent hours planning, writing, and debugging my code, the AI-generated version was ready in just a few minutes. But speed doesn't always mean better, because I still had to fix mistakes and adjust the AI's code to match what I wanted.

For example, in user input design, my Prog6.java used a menu system where players selected their type using numbers. AI's version, however, required users to type the full name of the player type (e.g., "human" or "random"). This made AI's version less user-friendly, since people don't always type exactly as expected. But once I noticed this issue, I gave AI another query, and it fixed the problem in seconds.

So, overall, AI's speed is an amazing tool, and I spent much less time on the assignment when using AI compared to doing everything on my own.

Code Style: Practical vs. Over-Engineered

Another big difference was commenting and formatting. AI-generated code had a lot of comments , sometimes too many as shown in (*figure3*). While comments are useful, AI explained even basic things that didn't really need explaining.

```
@Override
public int play() {
    int totalScore = 0;

    // Continue rolling until the total score reaches at least 15
    while (totalScore < 15) {
        int roll = (int) (Math.random() * 6 + 1); // Roll a six-sided die
        System.out.println(getName() + " rolled: " + roll);

        // If the player rolls a 6, they lose all points for the turn
        if (roll == 6) {
            System.out.println(getName() + " lost all points this turn!");
            return 0;
        }

        totalScore += roll; // Add roll value to total score
    }

    return totalScore; // Return the final score for the turn
}
```

-figure3-

However, one thing AI did better was naming consistency and writing shorter, more structured code blocks. Even though its code was more efficient, it was sometimes harder to understand, at least for someone like me. And that's not always a good thing. Clear and simple code can be just as important as well-structured code.

Lessons Learned

This project taught me a lot about how AI and human coding can complement each other. AI is a powerful tool, but it still needs guidance to produce the right results. It doesn't always get things right on the first try, so I had to debug and refine the AI-generated code to match what I wanted. While AI was much faster, my own coding approach paid more attention to usability like, for example, making sure the game was easy to interact with. AI often assumed perfect conditions, while I considered how players might actually use the program.

If I were to do this again, I would start with a better initial query, clearly explaining the structure I want and the type of output I expect. I would also let AI handle repetitive code but still take charge of logic, usability, and debugging myself. Lastly, I would try to combine AI's strengths, like consistent naming and structured code, with my own practical approach, keeping things clear and easy to follow.