Abdirahman Mohamed

COS 420

February 12, 2025

When it comes to creating programs for games like Bulldog, the differences between manually coding a solution and leveraging AI assistance become very apparent as you analyze the final results. My experience constructing two versions of the Bulldog game, Program 1 (manually coded) and Program 2 (created with AI support), offered an interesting opportunity to examine quality, independence, style, and efficiency. Below, I share my thoughts on these two approaches and how they compared.

Quality of the Output

Program 1, the version I coded manually, showed how much effort and thought I put into designing the classes, their interactions, and the overall gameplay mechanics. I spent time ensuring the program handled user input, game rules, and edge cases. The final version worked, but it wasn't perfect—the user interface was very basic, and there were minor bugs related to edge cases that I had to fix during testing. However due to some misunderstanding of instructions and the rules of the game, there was much left to be desired.

Program 2, on the other hand, benefited from AI assistance. The code was cleaner and more efficient right from the start, with fewer bugs during the initial run. The AI suggested better ways to optimize repetitive tasks and organize the program more modularly. However, some of the design choices felt generic and less tailored to the unique aspects of the Bulldog game. This revealed the AI's tendency to default to conventional patterns rather than make game-specific decisions.

Time to Completion

Manually coding Program 1 was a slow process. I had to write everything from scratch, debug each feature, and constantly test to refine the gameplay. Tasks like designing the turn-based mechanics or handling player interactions took hours, but it was an invaluable learning experience.

With Program 2, everything moved much faster. The AI generated boilerplate code, provided reusable templates, and even suggested fixes for errors, drastically cutting down the time needed to complete the program. Tasks that would have taken hours on my own were finished in minutes. For example, setting up the scoring system or implementing player interactions required little manual effort with AI support, giving me more time to test and fine-tune the game.

Independence in Approach

Even though Program 2 relied on AI, it wasn't a fully automated process. I had to carefully craft prompts, edit the AI's output, and adjust certain parts of the code to fit the game's rules. For instance, while the AI generated a solid base for a generic player class, I had to customize it to align with the specific requirements of the Bulldog game.

In contrast, Program 1 required complete independence. Every line of code was written manually, giving me full control over the design and implementation. While this approach made debugging more challenging, it was also more rewarding—each bug I fixed deepened my understanding of the program. That said, human error was more likely in this method. I personally struggled with misunderstanding certain game rules and parameters, which led to mistakes that required additional time to resolve.

Style and Readability

The code for Program 1 reflected my process as a student learning through trial and error. While functional, it wasn't always polished. Variable names were descriptive but often too long, and some methods became unnecessarily large, making the code harder to navigate. Despite these shortcomings, the structure—with clearly defined classes and logical flow—was solid and easy to follow.

Program 2, thanks to the AI, had a more polished and professional appearance. The AI suggested concise variable names, modularized code blocks, and efficient data structures. For example, it proposed using ArrayLists to handle dynamic player data, which made certain processes much smoother. However, the AI's focus on efficiency sometimes came at the cost of readability, which might make the code harder to maintain for someone unfamiliar with it.

Bugs and Debugging

Program 1 had its share of bugs during development. Issues with edge cases, such as handling invalid user input or unexpected gameplay scenarios, required extensive debugging. While this process was frustrating at times, it also gave me a deeper understanding of the program and how to handle similar issues in the future.

Program 2, in comparison, was remarkably clean from the beginning. The AI's suggestions preempted many common errors, such as null pointer exceptions and improper index handling. However, when bugs did arise, they were often harder to troubleshoot because parts of the code generated by the AI felt like a "black box" that I didn't fully understand.

Lessons for the Future

If I were to tackle these projects again, I'd approach them differently based on what I've learned. For a manual build, I'd focus more on planning and structuring the code before starting, which could help me avoid inefficiencies and mistakes during implementation. I'd also incorporate some stylistic practices suggested by the AI, like modularizing methods and using data structures like ArrayLists.

For an AI-assisted build, I'd refine my prompts to better guide the AI toward solutions tailored to my specific goals. I'd also dedicate more time to understanding the generated code, ensuring it aligns with my vision for the project. By combining the strengths of both approaches—the creativity and control of manual coding with the efficiency and polish of AI support—I could create a program that balances quality and speed.

Final Thoughts

Building two versions of the same program highlighted the trade-offs between manual coding and AI-assisted development. While Program 1 offered a deeper learning experience and gave me complete creative control, Program 2 excelled in efficiency and polish. Neither approach is inherently better; instead, they complement each other in unique ways. As AI tools continue to improve, the real skill will be learning how to balance these methods to achieve the best possible results—a lesson I'll carry forward in future projects.