Ben Gaudreau

Professor Levine

COS 420

March 2, 2025

Writing code is similar to stacking playing cards on top of one another. As more cards are added to the stack, it is likely that some of the cards will slide slightly out of place, leading to an uneven, poorly balanced stack. Code refactorization is analogous to taking that messy stack and straightening it out such that it becomes the orderly tower of cards that it ought to be. It is a laborious process to be done by humans, made somewhat easier by IDEs with built-in refactoring tools. Can AI tools, like LLMs such as ChatGPT, make this process quicker without losing accuracy?

THE 104 CONSTANT

My previous of versions of Bulldog had only one instance of the literal value ‘104’ being used as the winning score. No AI tools were necessary to replace this with a constant variable. In fact, I had already made the change by the time I submitted the fourth program. In this refactorization, I renamed the constant variable from MAX_SCORE to WINNING_SCORE for clarity. A negligible amount of time was spent on this process.

CONSOLIDATING RANDOMNESS

The inclusion of a Dice class was rendered trivial by AI tools. Using only the paragraph provided in the assignment, I was able to have ChatGPT-4 produce a near-perfect implementation. Emphasis on “near”. It had originally decided that all Dice must have at least 1 side, and throwing an exception if one was instantiated with fewer than 1 side. However, a one-sided die is entirely redundant. There is no point in creating an instance of an object when a boolean “true” will suffice, so I

increased this minimum side requirement to 2. Additional refactoring of the Player classes to use Dice objects for their all their rolls made this process take about thirty minutes to complete.

JAVADOC

Although comments can be found through most iterations of my program, solid documentation in a consistent style is key to improving readability and the ease of future expansions. Similarly to the score constant, this was a refactorization I had started to implement before this assignment. Although I had not mentioned this addition in my previous report, I expanded upon it in this project. ChatGPT-4 was able to write "good enough" documentation in my previous work, but I believe that the risk of losing information is not worth the time saved by AI tools generating it in place of humans. It is prudent for a program's author to understand how that program works, down to the lines at least. If one cannot explain it, one does not understand it; unknown code is dangerous. All in all, this process took me the better half of an hour to finish.

FILE STRUCTURE

This was the most time-consuming process of them all. Not because it was difficult, but because I had been modifying the file structure in tandem with each of the other refactorizations. After the third program, I had already begun thinking of ways to better organize the classes in the program. These changes were not realized until after the fourth program, in which I was advised by my professor to hold off on refactorizing in the midst of adding new functionality. I ended up doing most of this work by hand since I had already completed most of the work at this point, but in the spirit of researching the AI tools available to us, I decided to give the same task to ChatGPT.

I had no idea if a LLM had the capacity to create multiple output files within a single session, so I was pleasantly surprised when I received a well-split group of files, each one containing just a single class. It likely would not have been any faster doing it this way, considering the primary operation I had

used to do the work by hand (copy and paste) was still necessary for me to transfer the AI output to my IDE. Overall, I spent at least an hour and a half refactoring the file system and the various classes within.

ACADEMIC HONESTY

And now for a complete digression. Is it right to consider myself the author of all the code that falls under this project? In the Bulldog project, there are two sides of this question that need answering: AI-generated code, and the two classes provided with the original programming task. We will begin with AI-generated code.

To what extent may we claim ownership of code that we did not write, but could if given the time? Many IDEs support automatic generation of getter and setter methods, yet it would be insane to be required to cite something so simple and automatic. The lines begin to blur once LLMs get involved though. ChatGPT wrote the majority of my Dice class. I added documentation and changed two lines. Is that enough to call the code my own? Can one truly plagiarize something written by a LLM? ChatGPT hardly cites its own sources; are LLMs plagiarizing to create their own answers?

The types of output we get when we are looking for an answer to a problem are most similar to a secondary source, but I feel hesitant to call ChatGPT a secondary source… given that it and other LLMs' responses do not involve thought as we know the term. One can argue that a LLM's corpus represents its knowledge (and its biases), but such a model only mimics that which has already been made. As such, I find it unlikely that AI tools will ever produce a unique train of rational thought that spurs the collective knowledge of the human race any further than where it is at currently. I believe that AI can still be a useful tool to summarize and provide a starting point for learning, but we, as humans, should be capable of seeking out knowledge on our own without asking a computer how we should think and act.

Ans so we return to the question: How much AI-generated code is too much for us to be able to say, “I wrote that,”? There is only one viable way to solve the problem that the Dice class solves. It can be written with different variable and method names, sure, but the underlying logic remains the same. Use a static Random object, initialize with a certain number of sides, and have a method to generate a number between 1 and that number of sides. No room for nuance, and as easy as it is for us to say, “I could have done that,” after seeing the AI-generated answer… a first-year Computer Science student should be able to do that. So I have no qualms taking credit for that.

But what about code written (presumably) by another human? The Player and WimpPlayer classes were provided as a starting point for the Bulldog project, but are necessary inclusions in the final program. Up to this point, these classes have not been modified. The original documentation style in these classes does not match with the rest of the code base. Changing that formatting may alter the file, but not in a significant enough way to not call it plagiarism. What if we change the contents of those comments? Rephrasing without citation is still plagiarism, and every high school-level student should know that.

That leaves us to modify the existing code and change its functionality. The inclusion of the Dice class allows to make a meaningful change to the way both Player and WimpPlayer work. All Player objects need a Dice object to roll with – that is why we consolidated randomness into a single class! We can then call that Dice object to generate dice rolls in a new public method, which I have named rollDice. Modify WimpPlayer (and the other subclasses) to make use of that method, and suddenly both Player and WimpPlayer do the same thing in a different way. Combined with the previous two changes and the argument from before – that this is the only feasible way to use the Dice class in the Player classes this way – and I feel better about calling this my code. I would still like to know the original author of the Bulldog code stubs to give them the proper credit, but I believe I have reached the “Fair Use” zone, to use an analogy from Copyright law.

CONCLUSION

Two-page tangent aside, I believe that AI tools can assist in refactorizations, with a few caveats. Refactorizations are done for human readability, and while LLMs can certainly write, not all of their output is as human-friendly as can be. Then again, what may be easy for one person to understand can be difficult for another. There will always be trade-offs, but it seems AI tools are best suited to refactorizations of file structures. Although the time savings were negligible on a project of this small in scope, larger projects could benefit greatly from AI tools doing the reorganization of code without changing too much of the code itself, and allowing humans to keep on writing the documentation that is most important to them and their projects.