Paul Lantow

Final Report

It's fair to say that I dreaded the prospect of the Dominion project after its announcement on the first day of this course. Visions of copy-pasted functions inside main, variables declared in the middle of a function, variables that would change names, and other horrors danced through my mind. On a personal level, Dominion was a card game that I held in particularly low regard, since my fellow board-gamer friends were infatuated with it and would regularly drag me into games that I felt were dull and weren't particularly interactive. In short, I was worried that my experience with Dominion promised to be painful and uninspiring.

As is often the case in the human experience, my expectations were nowhere near correct. Yes, dominion.c was a hideous witch's brew of the worst coding practices witnessed this side of Freshman CS 161, but outside of the 700+ line switch statement, very little felt purposefully malicious. Instead of evil, the code appeared to have sourced its misery from that less thrilling, but more common vice of man: willful incompetence. In retrospect, this was probably a more accurate reproduction of the legacy code that will be encountered in my career as a software engineer. Coders are people, and like people they try to do their best and regularly fail. Programming malpractice should only occur when a project lead quits and decides to leave a mark.

The first project was in many ways the most challenging. Unused to the code, it took an hour before I realized that 'make playdominion' would allow me to, well, play Dominion. Peering through that hellacious switch statement to make sense of the cards I needed to refactor was like swimming through mud. It was a struggle to apprehend the dominion project, but I was

ultimately successful and now had a base of knowledge to carry me through the rest of the project.. The second assignment introduced me to the importance of writing out custom debugging functions in C, as the standard ones would exit the debugger after a failing case was found. In addition, they would also not deliver any message about *why* the test case was failing. From these areas, my confidence in my test-making improved. The third project was perhaps the easiest, since I had already been using random testing elements in the previous assignment, since it seemed like the natural way to improve coverage. The fourth assignment required the most intrinsic knowledge of the dominion.c file in order to produce randomly generated games, but since I had spent almost two months mucking around it's byzantine functions, it was a challenge that proved to be surmountable.

Coverage was an area that I was happy with my results in. In a single card test, I was able to achieve 24.62% coverage of the entire dominion.c code. I was even more happy with my 51.38% code coverage for a random sample game. While a ~50% coverage doesn't cover every edge case, it's still a healthy amount for testing. If I were to spend more time crafting it, I feel reasonably confident that I could bring it up even higher, perhaps even to 80% coverage.

Christian Morello's code was the hardest to implement since he had modified his card effects to be a separate file from the dominion.c file. This necessitated me modifying every single one of my test cases in my makefile in order to ensure that my tester programs could even compile. It also made me question the coverage I was achieving with his code, since part of it was offloaded from the dominion.c file. On his code, I was only achieving a maximum of 21.42% coverage for a card test, although I was able to achieve a none-too-shabby 60.34% coverage for a randomly played game. I also found that his Village, Council Room, and

Adventurer cards were all in need of being debugged. Outside of some basic functions, I have moderate concerns about the reliability of Christian's code based on the failure of his code to pass these tests. The fact that his code can play a game without crashing is a promising sign, but if I was working with him to release this to the public, I would not feel confident releasing this product yet.

Jacob Broderick's code suffered from the same failed tests that Christian did: Village, Council Room, and Adventurer. The most coverage that was able to be gleaned from a single test was 22.34%, which is a nice middle ground between Christians and my coverage. A healthy 55.21% was covered by the randomly generated game. Ultimately, his and Christain's code seem to perform at similar levels of reliability, so it would seem prudent to apply the same number of misgivings I had about the reliability of Christian's dominion.c file towards Jacob's. In all fairness, my code also performed similarly to Jacob and Christian, so it would only be reasonable to lump my code in with the same reliability concerns.

The final project felt like a great synthesis of the techniques that had been studied all quarter. The tarantula project was the first time as a student that I created a real independent debugging program instead of relying on other people's work to do the debugging for me. Tarantula showed also demonstrated to me an alternative way of debugging. Rather than slogging through the code line by line to find the problem, I could now highlight the problem areas immediately. No other tool has yet made such a strong visual showcase of the importance of not only coverage breadth, but also coverage variety. Sections of code that could be safe in one test have the potential to be problematic in another, so the ability to differentiate between these instantaneously is powerful indeed.

Ultimately, my experiences in testing Dominion have provided me with some much-needed skills in debugging. These skills are of particular importance for me since I will be interning for Amazon this summer. Looking back, the skills I learned through debugging that morass that was dominion.c seem obvious now, but they weren't apparent to me at the start of the quarter. Given that most of my time programming is me staring at code wondering what's going wrong, my experience with dominion.c has helped me take serious strides towards honing my craft as a programmer.