My experience testing the dominion code in CS 362 was nothing short of a strange and confusing adventure. I knew nothing about testing really coming into this class, and I have now tested literally thousands of lines of code and generated many more thousands of files doing all sorts of crazy testing. The code itself was pretty disastrously hard to understand and read, but I get that this is by design. As programmers going out into the real world, we won't often be making our own code from scratch. Much of our work will be maintaining legacy code, and some of that code will make your head scratch in such ways that you will run out of head before you're done. Instead, you need to do your best to make bug fixes and reports here and there, make sure the code is being properly tested with as much coverage as possible, and refactor it in a way that makes it more readable for the next person. In essence, the adage, "Leave it cleaner than you found it" applies here.

In testing my classmates' dominion code, I found that I had much of the same experience in testing my own code. After running my own tests, including my unit tests, which tested 4 different regular functions in dominion.c, my card tests, which tested 4 different cards in both dominion.c and cardEffects.c (which is a file of my own creation), my random tests, including tests for 3 different dominion cards, and my test dominion function a number of times, which ran a complete simulation of a full game of dominion, I came out with coverage of about 53% on dominion.c. This isn't amazing coverage, but with the time that I had in class to create these tests on a codebase this large, I am pretty proud of the result. When I ran these tests against a class mate's codebase, the classmate being named Jacob Broderick, I got a similar amount of coverage at around 51% of dominion.c. I would attribute this to the fact that Jacob didn't do a lot to change the over all structure of the main dominion code, but instead elected to make small improvements and refactor where he should. I next decided to run my code against a student by the name of Andrew Tolvstad. I knew Andrew from some classes before, and he has a reputation of making some beautiful albeit hard to understand code, and this implementation of dominion was not an exception. I actually had almost an impossible time building his code properly, and it wasn't entirely compatible with my own unit tests as they stood. In this case, I wasn't able to get a proper coverage measure. I really tried my best to test Andrew's code, but he made some extreme changes to the Dominion codebase, to the point where I just couldn't fully understand it without a Rosetta stone of some sort.

As far as my overall experience of testing Dominion went, there's a lot to talk about. I actually went into this class with a minute amount of knowledge of Dominion, having played the

table top version with some roommates. It took not a heavy amount of time to learn what I was missing, and I started work on my unit tests.

The unit tests actually started out pretty difficult, as I had no idea what I was doing going in. I had to figure out how to write proper assertions, and I actually failed to produce correct code on my first assignment. All four of my unit tests and all four of my card tests were just not returning any of the right values that I intended, and I had to move on eventually to work on other assignments. I would eventually figure out later that my assertions were actually returning their opposite result, and I would eventually get every one of my unit tests to work perfectly. It felt good when I finally had functional code that even found the bug I had introduced into the dominion code on the inion card effect.

On the random test generation, once I had a handle on how unit tests worked, I had much of an easier time grasping this concept. One thing I did struggle with was figuring out how to work with the random functions in C, but that is just an example of a regular struggle every computer science major goes through. After that, it was just a matter of figuring out which things to randomly generate about each card effect. Some of this was difficult, and I actually ran into one or two minor problems when working it out, but I eventually ended up with 100% coverage of every card that I tested.

The next testing problem proved to be the hardest: generating a random game of dominion. I had an enormous amount of trouble getting this to work properly, and had to take some shortcuts to get everything working on time and in the right way. I had to make sure that my randomization was correct, and that I wasn't randomizing things that might break the game every time. On top of this, making a differential tester for dominion was a real drag, as I had to start working with the doc bases of other students. I had never really attempted this before, not even in other classes, and making sure my test dominion code worked on theirs the same way it worked on mine, with their debugging information working in tandem, was a true lesson in how one should document your code in a way that others understand it.

When everything worked, though, it was the most satisfying time I had in a computer science class. I feel like, subliminally and more obviously sometimes, this is the CS class that I may have learned the most from. Going into enterprise in the future, I know I will be relied on to test way more than thousands of lines of code on a regular basis, and with the skills I;ve developed in this class with testing as a whole, I think I'm at least somewhat more equipped to take on the wide world of bug hunting!