Dylan Washburne

Test Report

Testing Dominion has been a rather interesting project for me. Historically, I have used just about all of the methods we learned in class in at least a limited capacity, but this class took those methods to the logical extremes.

Beyond just understanding the whole “test for correct values” I had always done previously, I had to learn the most efficient methods to do so, and doing so meant mass automation in most cases. In fact, I’d go so far to say that this class taught me more about processes for automation than it taught me about the entire debugging process.

Allow me to break that statement down. Fundamentally, I have always understood how testing should work. All the methods we learned in class, such as “check at points for correct values,” were things I have known and been doing for a while. I’ve even done stuff as crazy as mutation testing to see what breaks, or more importantly if it can survive. I have always known that you have to get deep into your code to understand the issues. I have always known that you have to run your code a lot.

What I didn’t know, or at least never applied, was the automation of any of those processes. At most I previously had it printf what a variable was at a given moment in time, but that is inefficient and very involved. I would also have to manually work through a program repeatedly to try and force an error to appear, hopefully in a context I could reproduce reliably.

This class changed everything for me. Sure, we started simple by creating tests for individual cards. But that’s just how the process works. By the end, I was automating the entire system to not only test my own game, but also test that of someone else and tell me what went wrong.

The other thing that stood out to me in this class is that we got to test Dominion. Dominion is a game I own and am very familiar with. I have even tried programming it myself previously simply because I was bored one day. Because of that, I know how many moving parts there are that can completely mess up. What I didn’t know was that we weren’t using the base set alone, but cards from a lot of the expansions, slapped together. I was the most shocked that we weren’t working with the Chapel of all cards. The Chapel is incredibly easy to implement, and it’s one of the most powerful cards in the game, yet it was not available.

Chapel rant aside, I know the inner workings of Dominion. I know how every card must be ready to work with every other card. I know how annoying it is to get the Throne Room to work (another card you did not include, by the way). And because I knew all that going in, I feared for myself when making test cases. One incorrect call anywhere in the code can cause the entire game to collapse upon itself.

Luckily, at the end of it all, I believe I have made reliable and responsive tests. The tests do not interfere with the actual workings of the code, and they detect errors with extreme accuracy. I even made a random tester that runs entire games and, over the course of just 2 seeds, has over 80% test coverage.

A good chunk of that is that everything in the code is fully randomized. Player count, Kingdom cards, its attempts to play Copper repeatedly from hand. Nearly everything possible in that given game will be attempted, repeatedly, giving it amazing coverage compared to the playdom included for us.

It is also through the use of my random tester that I have been able to test the code of my classmates. I only know them by their user names, edwardrh and lantowp. Simply running my random tester in their code over 2 seeds broke above 80% test coverage in both directories, and running with 5 seeds broke 95%. In these tests I observed that their programs never crashed under the massive randomized stress from my tester, leading me to the conclusion that their dominion code is incredibly resilient and reliable. In fact, the only issue I actually saw from the testing was that my tester stalled for an extended period of time, a known issue which I believe stems from its random nature, preventing the game from ending if the wrong choices are rolled repeatedly. However, without fail these stalled tests gave 100% coverage of almost every function relevant to that game (implying “of the cards selected,” since it had no way to access non-selected cards).

Honestly, I think a lot of the reliability comes from how good the base dominion code was. Yes, the switch statement was an unholy affront to good practice. That said, the code just *works*. I didn’t have to solve some cryptic puzzle when I first got my directory in order to make it compile, it just compiled and worked. I’m honestly more convinced at this point there were more bugs in the code from students doing their arbitrary bug changes, such as having Smithy under draw, than there were in the default repo.

Which leads me to my final point. This class taught me how to work with code made by others. Yes, I've worked with premade templates in lower level classes, but those were mostly to force me to learn what a struct is. What I had here is my first ever full project repo where I had to dive in and figure out what works. It's certainly not an excessively large directory, but as far as my experience goes, this is the largest thing I've had to start off with. From this I gained incredible knowledge and skill for working with large-scale projects, which are skills I will doubtlessly have to apply for the entirety of my professional career. Until now I only had to work with systems of my own creation, which is not how it works in the real world.

As a result, I feel as though I have gained a lot from this class. It has taught me how to greatly enhance my testing, and how to automate it. I have learned how to deal with large project repos with very little instruction. From this, I can walk away from this class confidently feeling that I learned a lot.