Jake Smith

Software Engineering II

Final Report

My Experience with Dominion

General Testing Experience

Testing and debugging was a topic I have heard a lot about going through my classes, however it wasn't until this term I wrote my first test for my own program. In previous classes, I generally wrote and compiled my code in pieces, throwing in print statements whenever I was unsure about a certain part of the implementation. Testing and debugging a large prewritten program with buggy code was never something I had to deal with until this class. At first I was surprised by the idea of spending an entire class working with a large amount of very poorly written code. Once we started into the assignments in the course, I quickly got intimidated. But after getting to know the code it became easier and easier. In this class we covered topics from Github basics, refactoring, test design, randomized testing for multiple test cases, and creating code coverage output files from a Makefile command. Ultimately, my experience testing Dominion was a great learning experience, but rough at times.

I was unfamiliar with Dominion so it took me awhile to really understand the code and be able to write tests for it. I found unit testing to be very helpful because they are quick to write and can test a lot of inputs. Where I found it to be less useful is when trying to find errors that occur with game states for example, the hand is empty and you try and play a card. Unit tests are a great way to find errors in a particular part of a program, but not necessarily great for finding errors when parts are combined. My first experience with unit testing was with assignment 2, when writing the tests for the cards it was interesting to dissect what you needed to test and what outputs you would look for and want. I soon realized that the more feedback you give yourself the better and you can never output too much information.

I think random testing is pretty powerful and is probably my favorite type of testing we explored in this class. In assignment 3, I wrote random test generators for Dominion cards. These random test generators looped through the initialization, function call, and test conditions for a defined number of tests. The initialized conditions were randomly set to ensure different test cases are discovered. Assignment 4 also included randomized testing, but was an entire simulated game of Dominion. I created my tester so that it would run multiple full games of dominion, each with a random number of players, kingdom cards, until a winning condition was met. There are many possible issues and bugs when implementing a full game of Dominion, so most of my tests were a series of print statements containing information about the game state at moments in the game. For example, I displayed the selected random variables and kingdom cards, who's current turn, cards in hand (before and after purchases/actions), end of a players turn, total treasure in hand, and ending game score. One problem with random testing is reproducing a bug, on occasion I would find bugs in one of my implementations using random testing but found it hard to reproduce the bug with a unit test because the state that caused the bug was tricky to reproduce in the first place.

Overall I think this class made me a much better tester and a better programmer. In previous classes testing was an afterthought but this class showed me how powerful and important it can be. My initial

tests were pretty weak but I think by the end of this class I can now write much more powerful and comprehensive tests. Testing Dominion introduced me to several strategies to implement when approaching a large document such as this one. It showed me that testing is more complex than I had previously thought. This knowledge is useful when creating any type of program, whether it is a game or something work related. I feel that tackling a big project such as this one was the best way to learn how to test. From testing the implementations of two other students, I learned how important the few changes I made to my own implementation may have been. I only fixed a few bugs while using dominion, but in ways I cannot be sure of, they ended up making my code work in situations where other implementations did not. I found that most students did not change their code significantly, but the small changes they did make may have been more significant than I would have thought.

Testing Mcdoncam:

My code was fairly similar to Mcdoncam's code. I think we both chose a similar way to implement the cards that we did. Before running any of the code, I expected the code coverages to be similar across all versions of Dominion. This is partially due to the fact that we all started with the same Dominion source code, only performing some refactoring of it. I had an overall better coverage of 71% whereas his had 62%, I think this was because he chose to implement some of the cards with shorter effects and I chose to do some of the longer ones. As for reliability it is difficult to say whose is more reliable. Neither of our programs seemed to crash randomly and while testing I did not run into any weird issues. To test Mcdoncam's code all I needed to do was refactor my tests for the functions he implemented.

Testing Hollidac:

Hollidac's code, I noticed, was better written than my own. He passed almost all of my tests, except remodel card. When I tested his remodel card with my random card tester, I noticed that the wrong card was getting removed from the player's hand. That was really the only bug I found while testing his implementations. Overall, I would rate his code higher quality than my own. His code would almost always pass my complete game random tester, but occasionally it would cause an infinite loop. Overall, his dominion was in a working state, but it was still obvious there were many bugs that still need to be fixed to have dominion play correctly.