Benny Zhao
Alex Groce
CS 362

Test Report

The experience of testing and debugging dominion in this class has been a valuable and fun experience to me. In this class, I have learned a lot about writing test cases to test other code, what it means to write an effective test case, how to improve code coverage, what are bugs, what are errors, different methods and processes to testing code by well known software engineers.

Each assignment and project in this class has taught me what it means to test other people's code and how to randomly test other programmer's code. Also Dr. Groce's method of teaching has been very enthusiastic which made the class more interesting to look forward. His history of being a software-testing engineer at NASA has given him credibility, which motivates us as students to look at software testing at a different totally new perspective in this world.

The first assignment we had was refactoring the dominion source code and implementing some of the cards into their function. This assignment was very straightforward in instruction wise but actually getting accustomed to the code and knowing what was going on took a while. Also the fact that we had to introduce a bug within the code was very interesting because at some point even I forgot where the bug that I introduced myself was regardless that there was probably a bug in the code initially.

When it came to writing unit tests and card tests, the unit tests were a bit more challenging to write that the card tests. Probably due to the fact that the card tests were more specific on what functions we wanted to target while the unit testing seemed more of a "where do we want to start testing" kind of situation. After a while I noticed that my tests weren't as detailed as they should've been because in class we learned that a good test should be able to give enough information for the programmer to find the bug. Even though my tests weren't as sufficient, they still gave me a good 51-percentage coverage of the dominion code.

Proceeding on the next objectives we had to complete was write a random tester. I found this part to be more challenging than it should have been. But even though it was difficult, I liked how the fact that random testing was the more efficient way to detect a bug and hit more test cases. Running random tests seemed to test the dominion code for more different types inputs, but this didn't mean the coverage would increase because coverage has to do with how many lines were tested essentially. One problem that occurred while random testing adventurer was that it kept segmentation faulting when the random tester was run. This increased the difficulty in trying to find the bug that was in adventurer.

As we got to the fourth assignment where we had to create a test of a full game of dominion and basically test it against another student's dominion to see the differences that occurred. The difficulty for this assignment was to figure out how exactly did I want dominion to run and output it's gameResults so that I can actually tell what is going on within the execution. To figure out the logic of the game I had to use knowledge from the previous assignments on random testing and looking at the playdom as an example. Also

since we only played dominion in class once, I had to look up and research how different scenarios would be played out in the dominion game. In the end, I got differences between my dominion game and another student's, but I still didn't feel as if the random test was sufficient enough in testing the different cases that could be played out in dominion. Initially I had set the random seed to be very high, but not so high that the game would give weird outputs. Then as I had my random tester, I wanted to test against my own code first and diff those two outputs so to double-check if my tester was working properly.

Lastly the final project had us combine all our previous knowledge and use our unit and random testers to test other students' code. After I chose which dominion files I wanted to test on, I looked back on my unit and random testers and made some changes to them because I felt the need to see more information on the outputs I was getting. As I made the changes to my testers I could see what bugs were lurking within other students' dominion files. It was very interesting to see the bugs that came up and one that seemed to be popular was the adventurer bug that didn't discard correctly and how it drew more cards than it needed. After testing other student's code, I had to read bug reports that were about my dominion file and seemingly it was bugs similar to the ones I found. One of the bugs that I corrected was how smithy would discard the third card after it was drawn and this didn't seem correct as I remember smithy is drawing three cards and then at end of turn the hand is discarded as a new hand is drawn.

The coverage for my code was around 29% for my remodel random test and around 28% coverage for my adventurer random test. Then for total coverage on my dominion.c was around 50%, which seemed decent since there are different seeds and different ways to choose the cards in game.

When testing the other two classmates' dominion code, I had to change their new named dominion.c file in my directory to just dominion.c in order to test it.

Looking at the hai's dominion code that I tested on, I found three bugs using my unit tests and random adventurer test. One being the common adventurer discard bug and one that discarded the third card of smithy effect. Then fahlmant's dominion code didn't seem to have problems with the cards that my tester tested. And it seemed that the coverage for random test adventurer was around 24%.

Overall, even though each of their dominion codes weren't severely wrong, I wouldn't say that their code is reliable because there are test cases that I might not have covered yet that could be buggy. Also it seems like their code didn't change significantly from the initial source code, so the possibility of bugs still existing in the code is likely.