Jong Park

Professor Alex Groce

CS 362 Software Engineering II

Final Project – Testing Report

Working with Dominion Code

I learned a lot from working with dominion code. For starters, playing the game of dominion. I've never even heard of the game until I signed up for this class and this game is more addicting than I initially imagined. It took me a while to refactor the dominion codes since I've never played it before and once I started, I saw a lot of bugs throughout the code. For most part, the code looked straight-forward so I did not notice the bugs at first. The most confusing part about working with the dominion code was the dominion_helper.h file needing to be included in most of the files. Since Makefile does not contain dominon_helper.h, I had compiling errors when I was compiling the testing case.

For the unit testing, I tested initializeGame, numHandCards, supplyCount, and endTurn functions. They were simple enough to test since they're one of the basic functions of the game and they're straight forward. The harder part was the card testing. For those, I used gardens, cutpurse, smithy, and steward. They were the difficult part since I was still learning to play the game and I wasn't sure what their effects would be. For all these files, I reported in unittesting.out but I'm not too sure if I tested them correctly since I wasn't sure what their effect was supposed to do. At assignment 3 I was still learning to play the game and I tested adventurer, smithy and gardens. It seemed simple enough and I had refactored my testing.h I've created in assignment2 to be used in assignement 3's random testing cases.

In most code coverage, I was getting average of 60% or 100% coverage. Then I remembered Agan's principle: "Coverage-based techniques focus on what gets tested." It was hard to tell if 100% code coverage meant that the code itself was not being covered so gcov could not detect it as complete coverage or not. I increased the number of testing and it was giving me different results and after fixing the code, it was averaging to 80% code coverage. I wasn't sure where it was missing the coverage. I kept editing small part of the code here and there until I had 92% coverage of the code. I decided to wait and ask for peer for help since more people means more coverage.

For testing other people's code, I used my classmates, Kathryn Maule's (maulek) and Rhea Mae Edwards' (edwardrh) codes. Since Kathryn and I worked on the assignment 1 together, our code started out similar but I could see that she had fixed her code more. Her code's coverage was better than mine and she seemed to have fixed all the errors in the code (or at least that's what it looked like). Her code was mostly 80%- 100% code coverage with 7.2% in her cards.h function. I ran the test couple more times and there were times when cards.h was 100% covered. Weird as it seems, I moved on to Rhea's code. Her code's coverage was

similar to mine; either mostly 60% or 100%. It seemed like her code was mostly untouched like mine.

Most fun I had working with dominion code was when I was making my own testdominion.c file. I was able to take player.c and refactor it so that it is actually working game of dominion file. There were many times when the game tester would throw segmentation fault or infinite loop for no reason. I assumed it was because I used not working copy of dominion.c file. It took me a while to get a working dominion.c file from someone before I could actually test my testdominion.c file to work. After getting a working dominion.c file, it was easy to test my testdominion.c file. I used bash script called diffdominion to test my dominion to my classmate's code and output to diffdom.txt and diffdom2.txt respectively. Then I used the linux diff tool to see the difference between the two text file and output them to difftext file. If the output were same, it would display "TEST SUCCESS" and if it wasn't it would display "TEST FAILED." For most part, the test would fail and I was back to fixing the code and then I realized, I wanted the test to fail since our codes are different.

The worst part about working with dominion code, other than dominion_helper.h file, was not knowing the proper rules to the game. It took me a while to figure out how to play the game and even with tutorial, it seemed like the game itself was broken somewhat. When I ran the playdom file to do a test run, the numbers would sporadically jump up and down and I wasn't sure whether if that number represented score, remaining cards, discarded cards, or money at hand. I also had hard time implementing the correct card functions since the game could only explain certain cards at the time and only way to explain different cards was to get to the next level where different feature was explained.

In overall, working with a buggy code was very different experience than writing my own code from scratch or working off a skeleton code the teacher would give us. There was times, especially the cardEffect function, where I wanted to write my own code based on what the card's effect was and what value it should return. Regardless, it was fun working with a code I did not write and taking time to fix one thing at a time. I might take some time in the summer and write the code from scratch because this is very fun game and it's a shame that it does not work as it should. From this course, I learned a lot more about fixing code, refactoring code, code coverage, and proper way to debug a code. I also learned what a mutant code was and different debugging tools other than gdb. This class turned out to be more fun than I initially expected.