Jason Ye
6/5/16
CS 362
Dr. Alex Groce

Dominion Test Report

Dominion Testing:

Throughout the whole process of testing the game, it was extremely difficult to follow and read through all the codes from the Dominion game file. The files was complex and in such a tangled control structure. Once I started researching on the game and the program, I gradually understand the basic and main structure of the game.

For the first few assignments, I was encountered with unit testing along. I did not have much difficulties doing unit tests, because I have learned and worked with unit testing in Software Engineering I. Once I understand how each card effect works, it was simple enough for me to implement unit tests for each card. Although not every card tests have full coverages, I was able to manage about at least 60% of test coverages.

Difficulties and problem started to occur after assignment 2. I have problem implementing random test generator for the Dominion cards. The fact that I have never done any random testing had put me off-guard. I realized random tests were different than unit tests, so I had to analyze the basic structure of generating randomness for my code. For the first time, my random tests for the card adventure got a coverage of a hundred percent. Although random testing gave me hard time, I knew that it is more efficient than unit tests.

Differential testing was very difficult for me to write, because I have never done any work with that time of testing. I did not get much success out of my differential tests due to the lack of knowledge I had. After a while, I had to find help from a teacher's assistant and friends to complete my assignment. When I reached to the point where I had to compare my code to one of a classmate, he had a similar idea and code coverages as mine. I realized that most student had difficulties for assignment 4.

Code Coverage for Melloc:

Most of my card tests, unit tests, and random tests were all had at least 50 percent coverage. As I picked and compared one of my classmate, Melloc's, unit test and card test, my test coverage and his were about the same. We share similar idea but did it in a total different ways. We had an average of 52 percent test coverages on our unit tests, and an average of 49 percent on card tests.

As for my random tests, it covered about at least 50 percent of the cards, but I had to work on it even more. Surprisingly, my random testing for adventure card had a full coverage. I implemented a thousand time for number of test on this card in order to verify the correct test cases. For my other two cards, I implemented random testing on Cutpurse and Smithy. Both of

those cards required a lot more work than I expected, because I had to run and cover at least 40% of testing. In the end, Smithy and Cutpurse had coverages above 50 percent. Since we had similar unit and card test coverages, it was not reliable for me to analyze. The major differences we had were testing on the dominion game. I have ran my test code on his game, and the result printed out a noticable differences. This show that I can rely on his game for my tests, because we had different result. Since my testing was barely passing the coverage requirement, my coverage was at the minimum of 61 percents while his testing covered about 70 percents. Neither of our code was broken and all my tests were passing on his game.

Code Coverage for Leed:
Leed's card tests are pretty similar to my card tests. For our card tests, we both used the cards Village, Smithy, and Great Hall. The only card test that is different than mine is the adventure card. As I compared our three identical card tests, we had about the same coverages. Our idea is similar but had different implementation. It is very reliable for me to compare my card tests with his card tests.

As for our comparison of unit tests, we had very different testing. My unit tests covered the functions playCard, endTurn, and supplyCount, while his unit tests covered the functions scoreFor, shuffle, and initializeGame. Our only similar unit test is testing for whoseTurn function. I could only rely on whoseTurn function, since he was testing for it, too. His implementation and idea of testing whoseTurn is similar to mine. There were only a slight differences of adding extra test cases, but our code coverage are about the same.

I have ran my random testing on his whole dominion game, and it show the result of only 40 percent coverage. It surely did not cover a whole lot of the game, but it was nice to see that my test was not perfect. I also ran his random testing into my dominion game, and it covered about 65 percent. I could totally rely on his test codes, because my testing was not efficient enough. There are many weak points in my dominion game testing. It is surely impressive to see how other people do their testing on the game.

Conclusion:
For the past 10 weeks, I have learned so much from simple Test-Driven Development to using complicated testing tool such as mutation and tarantula. My experience with testing dominion was fun and challenging. I had hard time writing up valid test cases due to the fact that I did not understand how the game work for a long time. To be honest, Dominion is a tough game to test especially to someone as inexperience as me. The hardest assignment for this class was the random testing for the entire game. Having above 60 percent of coverages in random testing was the most challenging part of this class. Although I had difficulties working on the assignments, I manage to completed them with helps from my peers and teacher's assistants. Overall, I learned a lot about the usage of testing tools and from experimenting the Dominion game.