# Test Report

SR Kanna

CS444 Spring 2016

**Abstract**

This paper discusses my experiance testing Dominion. It includes details about code coverate, the status and reliablity of my classmates Dominion code. This is a summary of my entire testing process.

## 1 Introduction

This term learning about dominion and creating our own Github repository to test the game. Since the dominion.c file we inherited was filled with errors (testable solutions), we spent most of the term writing tests, measuring code coverage and exploring various testing methods. Various methods in which we wrote tests include writing random tests for functions, random numbers test, delta debugging, tarantula and mutation testing. This report will discuss the various methods and experience I had during the testing of dominion program.
singlespace

## 2 Approach to Testing

In the beginning, I was extremely unfamiliar with dominion, so I did research on the game to understand the functionality. I found understanding the functionality helps a lot during testing, more-so than syntax. Most of the testing, especially the non-automated ones, I approached by understanding the point of test and then testing the functionality. For auto-mated tests, I needed to install the correct software and execute the program. For example, with mutation testing a script would produce the mutations. Whereas tarantula and delta debugging also required me to download the respective tools, but also needed specific code modification like gcov, c files and in the scripts.

## 3 Unittests

For assignment two, we were required to write a series of unit tests for functions in dominion.c, each over five lines of code. The unit tests recorded if the test successfully passed or failed and reported code coverage. I tested the update-Coin function, fullDeckCount,isGameOver, and handCard. For each of these, I looked over the function purpose and parameters to determine the best method of testing these functions. For example, for the update coin function, I allocated with a certain number of coins, tested the update function, added or didnâĂŹt add any coins, and then printed again to test if the number of coins was doing its job. I ran all of my tests and outputted them into a .out file to view test coverage and if they had passed. Almost all of my unittests failed to pass, furthermore the code coverage was extremely poor (around 20 percent). However, to put this in perspective, the code was covering only a fraction of the dominion code (a function), so thereâĂŹs not much to be expected. My outputs from this function showed the various outputs from running the tests and finally if the test passed or failed.

## 4 CardTest

The second part of assignment two, required us to create four tests for function implementations inside dominion, even though only two were tested. Outpostplayed, numhandcards, cardcount, and numActions were all tested though these tests.The two cardtest functions which I ran, both failed and had similar output to the unittests in regards to low code coverage and failed tests. After both sets of tests failed, I began to evaluate more of how efficient of a tester I was. I began to look up more testing tutorials and other people s tests to understand more effective methods of testing.

## 5 Randomtests

The third assignment had us create two randtestcards and a randomadventurer, which tested the dominion cards. Like with the unittests, I created a .out file to see the code coverage the examine the tests. When I compiled the tests, I was able to see much better code coverage (nearly 100 percent in all three) via gcov.

## 6 RandomNumber

In the fourth assignment, we were asked to create a complete game of dominion test, seeded by a random number (we tested 42). This tested my understanding of the game, because in order for the test to fulfill the requirements, it would need to play well. Furthermore, I wrote a mini specification in the form of a python, which compared two working dominion.c implementations. By diffing the files, I was able to see if they were the same or different which determined my pass/fail requirements. Running the script several times with different seeds yielded a failed result, since they two dominion implementations were different, but an upward bound of 200 results in pass. After running code coverage via gcov, I noticed that with a seed around 60-62, code coverage surpassed 60 percent, but the tests were still failing. This meant the requirements were met.

## 7 Project

For our final project, we found errors in other code, debugged and tested our code using delta debugging, tarantula or mutation testing. I choose to do mutation testing, which involved downloading SWI-Prolog and running a script to produce mutations. I noticed if the original dominion.c output and mutated version caused the same result, then the mutation was no longer relevant. I ended up having several difficulties using mutation testing especially with the version of SWI-Prolog and the path variables, so I went ahead and used tarantula. I have a script which finds plausible suspicious areas using gcov. This executes and produces two directories for passed and failed tests and also gives an output of the tarantula script.
Since we were asked to find bugs in our classmates code, I took more time to examine their code bank and the reliability of their code. I would say on average, since we all started from the same broken dominion and didnâĂŹt do any major fixes, most of have similar reliability and code coverage. However, I did notice a few students with significant changes in their dominion implementation, and I was able to learn a lot from their improvements. Their changes, especially in a few of the infinite while loops and broken functions, greatly improved the reliability of the code.

## 8 Conclusion

In the beginning of this course, we inherited a broken dominion.c file from previous classes and throughout the term, we had used various testing methods to understand the best code coverage methods. This report discussed the various methods and experience I had during the testing of dominion program. I covered in depth various methods in which we wrote tests include writing random tests for functions, random numbers test, delta debugging, tarantula and mutation testing. I further review my classmates dominion reliability. I have found my knowledge and approach to testing has significantly changed from the beginning of this course.