NAME: LUAN SONGJIAN
COURSE: CS 362
ONID: luans@onid.oregonstate.edu

TESTING REPORT

Summary

During learning in CS 362, there are four assignments about testing of game Dominion. In general, some assignments are hard and some assignments are easy. I did not hear about Dominion before so that I did not know how to design testing code to test all relative source files about Dominion sometimes. Through testing those files, I think all bugs are caused by complex structure and too many lines of implementing code in dominion.c. When I saw each line of code in that file, I found that it was very difficult to read and find out some important code.

Assignments

For all assignments, they are categorized by Unit and Card Test, Random Test, and Whole Game Test. In the first assignment, it was the easiest one because we just needed to create five individual functions for implementations of five different cards. However, the unique problem was that the implementation of targeting code was too complex so that I might cost much time to look for boundary from cases in switch(). However, when I did those test, sometimes the testing file could not execute and print out useful information to me. Most time it always shown segmentation fault.

❖ Unit and Card Test

In the assignment 2, it requested us to create four unit tests and four card tests to look for possible bugs in any four implementations of cards in dominion.c. It was very easy to write code for each implementation because there was a sample code supported in directory of project, but I almost could not find out bugs and execute smoothly during testing. Sometimes, those testing files shown 100% of passing rate and it would be aborted since unappropriated conditions added into function assert(). Especially, in my four implementation code of Card Test, there are three testing code can prove targeting source code correctly without any error information, such as segmentation fault.

❖ Random Test

In the assignment 3, it requested us to do two testing code for one of implementation of dominion and card adventure. For this game, I did not know too many things about this game and so I had to look for helps from my friends. After I known mechanism of this game, I try to define ten times for my maximum number of random test because I did not want to let my testing code to print out a lot of results into files *.out. In my ten rounds for random test, it did not execute smoothly at most time, instead of showing part of result and segmentation fault to exit entire testing. Therefore, I had to execute my executable testing program continuously so that it can show important information and write it into corresponding *.out files. According to my experience of entire random testing, it could prove that there were too many bugs existing in allocation of memory to relative variables.

- **Game Test**

In the assignment 4, it requested us to implement a testing code to test entire game. I thought that it was the hardest testing assignment because the game should not run smoothly because of many bugs. In this assignment, it needed to show some different status of several players and what kinds of cards they handled and did not handle. Therefore, I also might assign some random numbers to each variables, including random player numbers, handling counts, and so on. In this way, the testing code can assign some random numbers to each variable and model a real time of playing Dominion. Random testing is the best way to find out bugs existing entire programs. Moreover, I also called those main functions in dominion.c to run entire game, such as initialGame(), shuffle(), and so on. I needed to call some created functions to print out results on screen in terminal. In this time, I did not need to limit maximum number for this test. Basically, my testing code could execute smoothly and show some unusual data. Those data was not my expected because they were very different from actual results. Also, when one player had no cards, the game could keep running. Sometimes, there were some negative numbers shown in terminal.

- **Coverage**

Apart from assignment 1, other three testing assignments requested coverages as part of testing results. I was a freshman of Linux and so I did not know many commands in shell. When I heard about command gcov at first, I did not know how to use it to create coverage information for my testing programs. After searching relative information in Google and trying to use it, I had known and mastered this command. According to my all coverages of all testing programs, they always kept at very low level, usually from 20% to 30%. Especially in unit test and card test, the coverage could be less than 20%. There were only one or two times that kept more than 40%. I thought those coverages were very enough to prove there were too many bugs in dominion.c.

Final Project Test

In the part one of final project, it requests us to test and find out three bugs from classmates' code. For me, to find out more than three bugs in the implementations of the same person is difficult. Therefore, I want to look for three different persons' code and write unit test and random test for each person to find out three different bugs. It is easy way because I do not need to cost much time to look for more than three bugs in the same program. Basically, all bugs are the same since each person just revise only a little code in each source code. After executing and testing targeting files, they are printing out similar bugs as results. In addition, the coverages are very low, usually most of them maintain at level of 5%.

Conclusion

In conclusion, all assignments, including final project, are challenge for me because I need to do all things that I have never attempted before. Through doing those unit test, random test, and integrated test, I have understand what a project testing includes, how to choose adaptable type of test to test a project, and how to find and analyze bug information for targeting project. Moreover, learning how to test a project will improve my skill of making code for my programming career.