Test Report

- CS 362 Final Project #4

Jiongcheng Luo

June 5, 2016

Introduction

Dominion was a game that I have not played with before I attended in this class, the Dominion program looked extremely complicated to me at the very beginning, and I felt puzzling when we were supposed to test and debug this big program. However, I realized this was always the situation for a software developer, which always have to test other's program. Therefore, it's important to know how to test a program in smart ways and by efficient tools.

My Testing Process

I experienced in testing two of classmates ("jiangzh" and "houw")' dominion files by using my own testers. I used the combination of unit tests, random tests and game generators for the testing. I assumed I have no any information about what and where the bugs were in both classmates' files. The goal of the testing was to estimate the reliability of their Dominion code. Since I have limited test cases and testing tools for the test, my tested functions in unit tests are "updateCoins()", "isGameOver()", "drawCard()" and "numHandCards()", my tested cards in unit tests are card "smithy", card "adventurer", card "great_hall" and card "steward", my tested cards in random tests are card "smithy", card "adventurer", and card "steward", so

my estimation would be based on the output result and coverage information (line executed and branches executed) of these 4 functions and 5 cards. Then I would have my own analysis and estimation about the reliability for both Dominion codes based on all the testing results, I would use percentage to indicate my estimation to the reliability of the codes. My test steps basically included:

1. Make a copy of the original Dominion files from each of the classmate's repository and saved into two different directories.
2. I replaced the both classmates' original Makefiles with my own Makefile, so that I can output results that I would need for estimation.
3. Compile my Makefile separately for both programs, then I would get 5 output files, 1 of them consisted all unit tests cards and card tests, 3 of them were random tests for three different cards and the other one was for the entire game generator)
4. Compare the coverage information from different testers. For example, card “adventurer” has been tested by both unit test and random test, so I could compare the two test results and see the differencees between two coverage information. However, the four functions (“updateCoins()”, “isGameOver()”, “drawCard()” and “numHandCards()”) have been only tested by the unit test, thereby I would compare the results with a correct version of the “dominion.c” file.
5. And since all tester and the whole game generator could output coverage information for the entire “dominion.c” file, so that I could estimate an overall performance and reliability of the Dominion code.

Test Result of jiangzh's Dominion code

My function unit tests showed all four tested functions “TEST SUCCESSFULLY COMPLETED” and with “Lines executed:100.00%”, that indicates there were no bugs found in any of these functions. This information is also same as the original Dominion files.

Throughout my card unit tests, the result showed that the card “adventurer” in this Dominion has error with message “Lines executed:85.71% of 21”, then I compared this with the result of my random test for “adventurer”, which showed “Lines executed:89.47% of 38”, so I assume that there is at least one syntax or value error in the code, The other three card test results all showed TEST SUCCESSFULLY with 100% line executed, and random tests of card “steward” and “smithy” showed above 90% coverage. Therefore, the reliability of all card functions may have 90% coverage.

For the coverage of the entire “dominion.c” file, my unit tests showed coverage between 16% ~ 21%, this test for original file showed %16 ~ 23%; and card unit tests showed between 20% ~ 21%, random test showed between 10% ~ 15%, and random test for original file showed 12% to 15%; finally the whole game generator showed:

- Lines executed:92.86% of 560
- Branches executed:97.60% of 417
- Taken at least once:92.81% of 417
- Calls executed:95.79% of 95

In overall, I think this Dominion code showed pretty great performance, most tested functions were acting good since they all had similar result to the original program, but there

might be errors in some of the card functions, therefore the reliability of the overall program is between 85% to 90%.

Test Result of houw's Dominion code

Unit tests showed all four tested functions “TEST SUCCESSFULLY COMPLETED” and with “Lines executed:100.00%”, that indicates there were no bugs found in any of these functions. This information is also same as the original Dominion files.

In card unit tests, card “adventurer” showed “Lines executed:85.71% of 21” and card “great_hall” showed “Lines executed:88.89% of 18”, others show 100%; then I checked the random test result for “adventurer”, showed Lines executed:92.11% of 38, and other two tested both showed above 97%. Therefore, except card “adventurer” and “great_hall”, other card functions are working great.

For the coverage information of the entire “dominion.c file”, my function unit tests showed 15% ~ 18%, and card function unit tests showed
18% ~ 21%. Random tests showed 11% ~ 13% which was a little off from the result of the original program, and the coverage information of the entire game generator showed:

- Lines executed:92.27% of 569
- Branches executed:96.63% of 415
- Taken at least once:91.57% of 415
- Calls executed:97.00% of 100

In overall, I think this Dominion performs good, but jiangzh's performance showed little better than this one, and I estimate this performance is between 84% ~ 88%. I considered the reason of this difference is that houw's adjusted bugs made larger portion of effects to the entire program, so that the overall testing coverage got affected.

Conclusion

Although there is no such a perfect way to show how the reliability of a program is, a great testing method or tool is powerful and necessary in software development. After I testing the programs from other two persons, I found that my testers might not be good enough to detect bugs. Since a Dominion program consists around 10 different files with thousands lines of codes, my testers are limited by just testing small portion of the entire program, and I think the actual reliability is hardly up to 90% if using a more detective tester. However, I learned things like unit tests, random tests, mutation test and more in this class and I believe these knowledges will be useful in my future work of software development.