CS362

Final project

Yuan Yuan

Test Report

In this class, we learned about testing. In assignment 1 to 4, we wrote unit test, random test to test cards and functions in Dominion.c. Also, we wrote a random test generator to test the whole game. We tested the coverage of Dominion.c so that we can see the part that was not executed. By analyzing the results of tests and the coverage of Dominion.c, there were some bugs be found.

In assignment1 and 2, we refactored 5 cards so that these card are implemented in their own functions, and we wrote four unit tests to test four functions. My unit tests tested functions which were initializeGame, playCard, updateCoins, and getCost. Also, we wrote unit test to test four card. We measured the code coverage by using gcov. Most of the tests have coverage which is about 20%. Also these test just covered a few percent of the code. But it makes sense. Because every test only test one function or one card. It is only a fraction of the whole code. But comparing to other classmates' code, my test code coverage is a little bit lower. Some code I tested did not be covered. These code might be only operated under particular situations. In my unittest1 which test the function initializeGame, I found that only the first player get five card drew, other did not. Then, my card test for smithy failed due to a bug existed in the function that implement smithy. According to the rule of smithy, after drawn three card, the played card should be discarded. Therefore, the number of the player's hand card should be 7 if it starts at 5 cards. But the test failed. The result of the card is only 6. The test for minion is also failed. In this test, I used assert to check if the handCount equals to 4. The test failed because the value of the handCount equals to 5. Therefore, by using unit test, we easily found codes that does not make sense.

After unit test, we learned a new method of testing code, which is random testing. In unit test, we only run the test generator once. But in the random test, we run the rest as much as we want. We set the card and number of the player to be randomly generated. So that the test could be able to test most of the possibilities. We wrote random test to test Dominion.c in the third assignment. The total lines executed for most of the test are 20% to 30%. For the card council_room, I made the test achieve 100% statement and branch coverage. I did not spend too much time to achieve 100% branch, I thought it is because my test covers most of the possibilities. In the test for card council_room, there are 255 times failed, and 1745 times succeed. According to this assignment and the experience about writing random test, I realized that the random test is more effective and covers more code than unit test. In the random test, I randomly generated the value of total players, current players, and the handCount for each other. This way helps us to see when and how the data would cause crash. As I mentioned before, the test generator failed to test the card council_room for more than 200 times. This is because we did not successfully pass the new value to handCount().

In the third assignment and previous assignment, we test some cards and functions. In assignment 4, we did some thing even harder than previous assignment. We wrote a random tester that plays the complete game of Dominion. It is easy to achieve 100% to test a card or a function using random test. But it is hard to make the test which is for the whole game to achieve 100%. In my test, my test coverage is about 47%. In my opinion, I think this coverage is not so good. On the one hand, I think my random tester did not cover all of the cards, it only used a few cards. On the other hand, I did not run my tester to try to gain a higher coverage. But when I used my tester to test my dominion.c and one of my classmates' dominion.c, there is no difference between results of the test for my dominion.c and test for my classmate's dominion.c. It shows that my

testdominion.c randomly test dominion.c effectively. In my final project, we are asked to use some kind of debugging tools to test our code. Also, we used the cardtest and unittest which we wrote at assignment 1 to test our classmate's code, and find bugs. I used the code from songyip and the code from houw. Most of our code are the same. But in assignment 1, we are asked to refactor the code so that these card are implemented in their own functions. These parts are slightly different. So do the test generators. In songyip' s code, his test generator is compact. His unit test for the card and function covered almost all of the results for the card and function. Therefore, his test generator is more efficient than mine. I used my testdominion.c to test his code, the coverage is almost the same. In houw's code, I found that his unit test and random test code are very long. His unit test covers a lot of staff. But when I run it, It clearly show the result. He considered a lot of small things, which sometimes I might ignore.

According to this class, I learn how to think like a tester. During this ten weeks, I learned many testing methods which is very useful to my testing. For me, I think one of the most useful tool is gcov. We can check the coverage to analyze our code. We can easily see which part did not be executed, how much times this line been executed. Also, we learned to use mutation method to test code. It lets me know how to improve my test and to make the program work even better. These would be my best experience about testing.