Jason Lim

CS 362

2/27/19

Assignment 4

**Assignment-4:**

The primary goal of this assignment is to learn how to write a random tester.

**Note:**

* • This is NOT part of the team project. Please do it on your own.
* • Submit your entire dominion folder with all new files to your repository under projects/youronid/dominion.
* • Create a new branch of your repository called “youronid-assignment-4”
* • Also, submit a zip file of your repository code.

Use the refactored code you created for assignment-2 to create a new branch for Assignment-4 called “**youronid-assignment-4**”.

Branch has been created.

2. Write an automated random test generator for three Dominion cards, one of them being the adventurer card, and at least one being a card you wrote unit tests for in Assignment-3. Check these testers in as randomtestcard1.c, randomtestcard2.c, and randomtestadventurer.c. (45 points)

For randomtestcard1.c I chose to write a test for Smithy and for randomtestcard2.c I used Council Room. Then I tested the mandatory adventurer card with a random test.

3. Submit a pdf file to Canvas called Assignment-4.pdf, that contains the following sections:

• **Random Testing**: write up the development of your random testers, including improvements in coverage that you gained from random testing. (15 points)

I developed the random test much like how I did the Assignment 3 unit testing and card tests. The difference was that the random tests used randomization as explained in the modules and further testing to make sure there is enough branch coverage. The improvements in randomtestadventurer compared to the cardtest for adventurer was that this time around I had a random deck size and a random hand and seeded a time value so that the randomization would occur. I did this in line 17-18 and also I used a random card and a random kingdom card value. These values were so that we could create copper, silver, gold, or other random kingdom cards based on what value was returned. My adventurer random test then checks to make sure there are not too many cards or too few cards based on the coin count for treasure cards. If two coins were not added to the hand then that means there is a bug in the code. I had more improvements gained from the random testing because we were testing multiple times rather than one time for a static unit test.

For randomtestcard1 I used the smithy card because this was the easiest for me to understand since it is a simple functionality of drawing +3 cards. I also had a random size of the deck and size of the hand similar to adventurer but without the random kingdom cards. I checked to make sure that the initial hand count and after executing smithy matched. Smithy will overall provide +2 into your hand after executing. The random test would do this with a random hand and a random deck count so it would test all possible scenarios unlike my previous test in Assignment 3 where I had a card test or unit test of one specific scenario where I chose one fixed amount in my hand. The loop also allows us to do multiple tests as per line 28 where we have 1000 cases where we can test these random test results. Overall there is much more coverage and improvements using the random tests.

For randomtestcard2 I chose the council room card. This card is also one of the cards I used in Assignment 3 and a simple one to test since it adds 4 cards to the hand and allows 1 more buy. Before in assignment 3 I tested the buys but in this randomtestcard2 I test the handcount. Using the similar way we set up Smithy I randomized the size of the deck and the hand and made sure that the total hand count during the game after the council room was executed was correct. I augmented how many tests were successful and how many failed in all of my random tests and then printed out how many were failures, successes, and how many tests overall were executed. This is an improvement due the number of tests I did in line 15 (5000) than the unit tests in assignment 3.

• **Code Coverage**: discuss how much of adventurer and the other cards’ code you managed to cover. Was there code you failed to cover? Why? For at least one card, make your tester achieve 100% statement and branch coverage, and document this and list how long the test must run to achieve this level of coverage. It shouldn’t take more than five minutes to achieve the coverage goal (on a reasonable machine, e.g. flip). (15 points)

I was able to cover 75% of adventurer and 100% of testcard1 which was smithy and 100% of testcard2 which was council room. It took 1500000 runs for 75% of adventurer and 1500000 for 100% for smithy and 5000 for 100% for council room. All of these did not take more than five minutes to achieve the coverage goal on flip. There were no codes were I did not fail to cover however for adventurer it took too long to cover 100% so I left it at 1500000 runs. The test for council room took 2-3 secs to complete and it achieved 100% coverage goal after using gcov on it. For Smithy randomtestcard1 it took roughly 2 mins to cover 100% coverage.

• **Unit vs Random**: compare your coverage to that of your unit tests that you created in assignment-3 and discuss how the tests differ in ability to detect faults. Which tests had higher coverage – unit or random? Which tests had better fault detection capability? Be detailed and thorough. (15 points)

The random test had higher coverage than unit testing on a case by case basis. I would say that if one knows how to test, the unit tests have quicker and faster fault detection capability however since random testing usually has higher coverage it has greater overall flexibility for catching the overall picture of faults. However for testing edge cases, random testing may miss the edge cases. For edge case testing unit testing such as in Assignment 3 might be more useful.

There are many downsides to random testing however. Many of the random tests in this week Assignment 4 were harder to debug as I had to trace all the way up and there is just too many tests being done at once. Also it is hard to know what is the expected answer since the input is random I am not sure how the expected output should look like whereas in a unit test we know what the expected output is and what we are testing for. I think that the unit tests for general sanity check assertions are better at detecting faults quickly even though random testing has higher coverage.

Since there are many inputs and output ranges with random tests, unit tests are better since using effective values in unit tests offers less complexity with the same amount of goal in testing.

However there are also many pros of random testing since it can provide an extensive and greater insight into undetermined edge cases. It provides integration testing and creates a whole lot of test cases all at once. Additionally, sometimes the random test provides ways to find outlier values due to the coverage that we can further isolate using unit tests so a combination of random testing and unit testing would be integral to a tester’s arsenal.

4. Add rules to the Makefile to produce randomtestcard1.out, randomtestcard2.out, and randomtestadventurer.out, including coverage results (10 points)

I was able to put the rules to produce the .out files and coverage results.

1. Identify the method under test

2. Identify all the dependencies (parameters)

3. Write code to generate random inputs for the chosen method

4. Invoke the method (execute the method)

5. Check if stopping criterion (like time or number of loops) is not satisfied, if not go back to step 2.

If the input is a primitive data type, generate a random primitive value, etc.

If the input is an array, create an array and initialize it with some random values, etc.

Try to “stay random” but shift the probability space (e.g., if you choose int numCoppersInDeck = rand() % 20 and int numAdventurersToPlay = rand() % 10. There must be a logical reason in the code specification for choosing 20 and 10.

Note on inputs:

You also need to improve your oracles (step 5) (i.e., assertions "if/print in our case") until you feel that all the problems that should be caught are caught! Make sure your Oracle helps you develop a ‘rock solid’ random generator and not just a random generator.