Arthur Liou
CS362

Assignment 4

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

– Done. Branch is “931989226-assignment-4”.

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)

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)
- 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)
- 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)

Random Testing / Development

Recalling that HW2 was to 1) refactor 5 cards into their own functions, and 2) introduce 4 bugs out of the 5 cards, we do not have any of the test cases we previously created in HW3. In HW3, I wrote unit tests for Smithy, Adventurer, Village, and Remodel. Since we are already doing a randomtestadventurer.c, I want to focus on other two randomtestcard(s) on Smithy and Village since the Smithy/Village combo is one of my favorite strategies in dominion.

- Improvements for Adventurer: Testing 1000 Random Test Cases, combining three checks/sub-test results for whether the test case passes or fails.
- Improvements for Smithy: Testing 1000 random Test Cases, tested for hand count and expected hand count, also total deck count and expected deck count. Started each test case with randomized handcount and deckcount. Max Hand and Max Deck are 500, so the numbers ranges between that.
- Improvements for Village: Testing 1000 random Test Cases, tested for hand count and expected hand count, also total deck count and expected deck count, and numActions. Started each test case with randomized handcount and deckcount. Max Hand and Max Deck are 500, so the numbers ranges between that.

Generally, my random testers coverages’ improvement were significantly better. Previously coverage was sub 35%, while in my random tester coverage, it was 80%+

Code Coverage

- How much of adventurer and other cards code I covered: Adventurer Random Test covered 100% Branches and 90% lines. Random Tests for Smithy and Village were both 100% branches, and 88% and 83% for lines executed respectively.
- Failed to Cover / Why: I failed to run into the edge cases (hence the remaining 12-18%), which would account for why it was such a small percentage of lines not executed.
- 100% Statement/Branch Coverage: Yes! See first bullet point above and following note (also the note/sources section below). I used "gcov -b -f dominion.c >> unittestresults.out" in my included makefile for breakdown coverage of each function, but attached two copies of my unittestresults output. One is named withFFlag for the function coverage of the dominion code. This was very useful to see the -f breakdown coverage, since I could see which parts of my dominion.c code were not covered, which makes sense. Due to unused definition, other cards defined but not played, etc, etc.
    - It was a good question asked by the Student/Original Poster. Otherwise, I would have been super confused on how to get 100% coverage for dominion.c if I could only touch three cards, one of which was Adventurer.
- Document this / list how long the test must run to achieve this level of coverage: The way I see it, I would have to optimize and refactor more test cases to cover all other cards to achieve an 100% coverage of dominion.c code.

Notes/Sources:

- https://piazza.com/class/jpu18p346423vs?cid=177

“”

The coverage is always on the target of the test so in our case, the dominion.c code.
The results you describe are reasonable given that you're targeting a specific area of the code. **And you can provide that portion as your submission.**
There isn't a parm for getting coverage on just one function that I know of, but you can parse out the information. There was a Piazza discussion about that earlier you might want to look at.

Just for something interesting, you might want to look at a gui utility called lcov that gives some nicely formatted output: http://ltp.sourceforge.net/coverage/lcov.php

“”

Unit vs Random

My random test had higher coverage. 1000% branch execution and ~85% Lines executed. For within dominon.c, they haed branch coverage and line execution of around 50%, compared to 30% in my unit tests. My random tests had better fault detection capability because they covered 1000 more test cases than my unit tests, even though they were testing the same card(s).

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

- Done – see attached Makefile and screenshot below