Elaine Alasagas Documentation
CS362 – Assignment 4

Github repository: https://github.com/ewalasagas/CS362-F2019/tree/alasagae-assignment-4/alasagae-assignment-4

4. Documentation:

I developed my random-testing strategies very simply in order to obtain code coverage and ensure the code does not run into seg-faults when unable to assert when the function does not execute. Nothing was changed from the original code from Assignment-3. For all randomtestcards, for each test scenario, the function was called 500 times (n = 500) and tested against initialization for 2 players and 4 players to cover the different play-type scenarios with the same set of kingdom cards as in my unit-tests for that specific function.

For the randomtestcard1 for the baron_ref function, I tested used two random integer generators, one between 0 < choice1 < 2, and -2147483647 < random_num_pos < 2147483647. At first, I had tried to test for positive values and negative vales where choice1 was either 0, 1, or 2 specifically, and testing for 0 < 2147483647 and -2147483647 < 0, however it was easier to run a test for max to min. For each 2 player and 4 player mode, I tested the choice1 where it was at a random_num_pos, and player p was already initialized and when choice1 was between 0 and 1, and the currentPlayer was the random_num_pos. No errors were detected, which should not occur, as the function should either exit or return -1 for all wrong inputs.

For randomcardtest2, for the minimion_ref function to run, a player had to have at least 1 card in their hand and another to possibly discard or not. I added 2 cards (randomly) to a players hand each time before the minion_ref function was called. The random card was between 0 < 25, as I did not want to test out-of-bounds randomization as this prevented game play and from the function from running if the card did not exist. I tested the choice1 and choice2 for randomization, as choice 1 indicates your action (either +2 coins or discard your hand.) Both choices were randomized between 0 and 3 – this ensured coverage would occur.

For randomcardtest3, for the tribute_ref function this was quite difficult to test randomized and inputs as well as code coverage without altering the tribute_ref function. This was quite challenging to execute because I wanted to test more out-of-bounds, but I had a difficult time in trying to catch or "ignore" when the function had unplayable inputs. Additionally, I needed to fill each opponent's deck in order to execute the bonuses. Therefore, I added a function that randomly filled each player's deck with in-bounds cards (i.e. 0 < 25 range.) For the initialization of 2 players, I only called the tribute cards and left the nextPlayer as 1, because it would be obvious. For the initialization for players, I set bounds from 1-3 (0 being the current player) as randomization of inputs could only occur in either current player or next player. The function needed to execute in order to capture code-coverage, so I was not able to test out-of-bounds integers for inputs.

In comparison to my unit-tests, randomg testing was easier and more simple to run and code in comparison to unit testing as the expected results are not as specific as they are in unit-testing. Additionally, by controlling the range, it was easier to predict that the function would run. However, random-testing had higher coverage of each function in comparison to unit-testing. I would say using

both unit testing and random testing are ideal in detection of faults, however I found unit-testing to be more specific to code coverage/feature and random testing better for compiler/system issues.

**CODE COVERAGE RESULTS:**

**BARON_REF FUNCTION -- RANDOMCARDTEST1.C**

Function 'baron_ref'
Lines executed:100.00% of 30
Branches executed:100.00% of 18
Taken at least once:100.00% of 18
Calls executed:100.00% of 9

**MINION_REF FUNCTION – RANDOMCARDTEST2.C**

Function 'minion_ref'
Lines executed:100.00% of 18
Branches executed:100.00% of 18
Taken at least once:100.00% of 18
Calls executed:100.00% of 6

**TRIBUTE_REF FUNCTION – RANDOMCARDTEST3.C**

Function 'tribute_ref'
Lines executed:75.00% of 32
Branches executed:93.33% of 30
Taken at least once:76.67% of 30
Calls executed:50.00% of 2