CS 362 Software Engineering II
Hyunwook Shin

Assignment 4

- Documentation
  - **A. Random Testing**
    - For this assignment we were asked to use random tests to find bugs in our code. Before moving on, I knew for random testing that the random inputs the range should be valid for the test to be effective so for each of the functions being tested: baron, minion and tribute I looked at all the parameters to make sure all the inputs I was entering were valid and the biggest one for me was the state of the game, the number of players and seed. These are the main variables needed to initialize the game, so switching these parameters would test the limits and boundaries of the functions themselves. So to randomize these inputs I used the rand() function to generate random numbers and use the % operator to divide by a number to get a number in the valid range. The unit test from assignment 3 were used but with the randomness generated by the rand() function with specific inputs. Along with this random testing requires a larger sample size for a wider range of variables to be utilized during the testing phase so most all three random testing had 30 iterations done. This greater increased the sample size as well as the number of random inputs that were tested for each function.
  - **B. Code Coverage**
    - I managed to obtain 26% coverage for the baron card which is not near the target of 100% code coverage. I believe the limited range of the inputs were a factor in the lower coverage I think I needed to find better ways to randomize the inputs for the parameters for each of the functions. If more variables were randomized the tests would tests more of the code and the coverage would be increased. Also and increase in the sample size would also increase the code coverage. Along with that I did want to note that this is 26% of 572 lines which is a little confusing. I'm not sure if the lines of code include all of the code that is in the switch cardEffect function, which would mean that 26% is actually much higher code coverage since the baron card function is much smaller than 572 lines of code. So given that, this coverage percentage is a little misleading. I think the random testing covers more of the code than the statements say. Either way the increase of a wider range of valid inputs for the parameters would still increase the coverage numbers and also increasing the number of the sample size will also increase code coverage. With more testing done or a greater sample size for the random testing more and more of the code will be covered during this process. So if there was more time, increasing the number of tests from 30 iterations to 100 or 1000 iterations could

greatly increase the code coverage. Although I didn't hit my mark of 100% code coverage I believe with more time and some increase in the design of the code coverage could hit 100%. This might take 10-15 minutes, but I believe refactoring my code would help achieve the targeted code coverage much sooner than this time amount.

C. Unit vs. Random Testing

- When I compare the coverage numbers from assignment 3 vs assignment 4 the coverage percentage is higher for the random testing. For example assignment the baron coverage increased from 19% to 26% in coverage percentage. 21% to 24% for the minion card function and 19% to 33% coverage for the tribute card function. All the 3 card functions increased which I attribute to the number of tests that are run by the random testing and also the amount of variation in the input in random testing. Because exhaustive testing is impossible to complete in many real world scenarios random testing is a better alternative that can test for many valid inputs and can be run a number of times. Also the reason why random testing is better for testing numerous different scenarios is the amount of testing that can be done with random inputs that are valid. With a unit test a specific set of inputs are decided and that is tested, which limits the code coverage because only targeted sections of the code can be covered. Whereas the random testing allows for more robust testing of the code and increases the code coverage, which is seen in all 3 card functions.