CS 362 Final Report Zhenggan Zheng
932101499

My first experience testing dominion was in assignment 2 when I wrote unit tests to test functions and card implementations in dominion.c. Since this was my experience testing the code, the test I wrote were not very sufficient. However, I was still able to get around 42.88% code coverage for all my tests combined. The functions I tested were: supplyCount, isGameOver, gainCard, and endTurn. The cards I tested were: Smithy, Great Hall, Outpost, and Embargo. One of the issues with my first unit tests were that they were not very thorough. They would only check to make sure that one or two criteria were met and it would pass the function. For example, for the test I wrote to test the gainCard function, I only tested to see that the size of the hand was increased and did not check to see if the deck or supply was decreased. Nor did I check to make sure it was actually the correct card that was added to the hand. I also did not check if the function can add cards to the deck or the discard pile. Another example is when I tested. These things will eventually cause problems later as I found out when I did mutation testing that my tests were very leniently passing functions that were actually failing. However, I did also write some good tests, for example my test for isGameOver tested to make sure that the game can end for either supplyCount of 3 things is 0 or if supplyCount of province is 0.

For assignment 3's random testing, I tested Adventurer, Smithy, and Great Hall. For these three I allowed the user to enter a seed and it would run the card 1000 times with random parameters generated by the seed of the user. With this method I was able to achieve 100% coverage for each card tested. However, the cards that I chose to test were relatively simple to implement and that is probably the reason for the high coverage. I believe that random testing provides much better results than unit testing. This is because even on a simple function such as smithy, unit testing will always pass it. However, even for simple functions such as smithy that should theoretically pass every time will still fail occasionally when running the function enough times with random testing. Random testing helps ensure that all scenarios are covered and if your function can pass a good, high quality and thorough session of unit testing, it is most likely a reliable function. That is of course if your test is actually written correctly. This reminds me of one of Agan's Principles: "Do not trust your tests or debugger." This is because there has been multiple times where my test will fail my function and I will think that there is something wrong with the function when in fact it is the test itself that is failing. One example is when I did random testing on Smithy and I wrote: If number of cards in hand is +3 of the original hand size then Smithy passes. However, this is incorrect since the card Smithy gets discarded and and the hand size is in fact only +2 the original hand size. Another thing I failed to do in my random tests is to check to make sure that the card that is being played is actually discarded. What I should have done is iterate through the hand of the current player and keep track of their hand and after every run of the test iterate through the current player hand once more and make sure that the card that was played is now discarded.

For assignment 4, I wrote a random tester to play a full game of dominion. For this I put all the possible cards into an array and then shuffled that array and spit out 10 of them to be the kingdom cards that were going to be played that game. Then I wrote an AI player that would randomly buy and play cards. The AI was much improved upon during the project.

For the project, I tested the dominion implementation of Rhea Mae Edwards and Kathryn Maule. First I ran unit testing of Rhea's Ambassador implementation and found out that it was not being called at all. This caused it to fail all test cases. However, it seems there isn't anything wrong with the actual function. I also ran random testing of her Village implementation and found out that it does not grant +2 actions to the player and actually grants +3. After that I used my game simulator to run a full game of dominion on Rhea's code. The results were that it was able to complete a full game with a clear winner and loser, however, the actual accuracy of the game is questionable. This is because her implementation is never actually able to play the cards that they purchase. The players would buy the cards but would never play them. I suspect that this is because the gainCard function never actually puts the cards that they buy into their respective discard pile. Due to this, the players are buying and cards but are never getting them. However, the game can still end since the cards are still disappearing from supply pile. I came to this conclusion by modifying Rhea's source code and changing the error values and found out that the playCard function never returns 0 and instead returns -3 (I added this to Rhea's code). This means that it never receives a card between adventurer and treasure_map. I also tested Kathryn Maule's code using the same game simulator and get the same result as Rhea's code. Since this is the case, both of their implementations are equally unreliable. Although Kathryn's implementation has less buggy card implementations compared to Rhea's. Unfortunately since the cards are never played, it does not matter. However, modifying the gainCard function should be an easy fix and after that is fixed, Kathryn's code should be more reliable due to more correctly implemented cards.

This has been my experience testing and debuggy dominion.c. Overall it has been a very enjoyable experience and has made me a better programmer by giving me better tools to discover bugs in code. I feel like I have already been using a lot of the concepts learned in this class but taking this class further solidifies it and allows me to see debugging and programming in general in a different light.