My experience in testing the game of Dominion was very fun and interesting. I've never heard of this of this game and thought I was going to have a terrible time in this class but it was the complete opposite. It was fun watching the entire class learn and enjoy this game together which made the learning experience that much better. Playing tutorials and learning the game was easy as well since it was easily accessible via Google. When I understood the game and learned the card effects it made testing dominion.c much easier.

The first assignment taught me two of Agans principle: understanding the system and making it fail. In order to test dominion, I needed to understand what dominion is and how it worked. If I didn't learn dominion then I would't know what to test for. It also The second part was to refactor cards and intentionally add bugs into them. This shows how easily bugs are introduced and that bugs can be very small but cause very big problems.

The second assignment was to create unit tests for four cards which is the divide and conquer principle. Splitting up the process and testing individual cards is easier than testing the entire dominion code at once. Unit testing individual cards lets me know if the card is working the way it was intended to and/or if it is working incorrectly. It also shows if the function is consistent when tested multiple times. This assignment showed me that this dominion implementation is extremely broken despite being a complete compiling code.

The next assignment was to create random test for two cards. This is similar to the second assignment but instead of inputting the same value every test, this time it sends random input values. Random testing gives better test results as it covers multiple cases. It makes sense to do unit testing before random testing. Unit testing shows us if the code gives consistent results, if it is inconsistent with just a single value then there is no way it would be consistent with varying values. After fixing the code, the random testing would show if the code is giving correct output.

The last assignment was to test the entire game of dominion with random amount of players to see if the game kept track of values such as money, points, amount of cards in hand, deck, and discard, and other values. These values were printed every turn and the score at the end of the game. This test also generated random sets of cards to be used each game to test consistency and make sure there is as much coverage as possible.

I tested rodrijos dominion using the tests made in the second, third, and fourth assignment. All card tests passed. The adventurer card sometimes gave more than 2 treasure cards. The coverage was about 26% with the test from assignment 3 and about 36% with assignment 4 test. The amount of money in play increases by way too much every game because the adventurer card doesn't correctly give treasure cards and smithy also gives too many or too little cards. This game of dominion is not reliable enough to be played.

I did the same tests to nguyenhai dominion code. All of the cards passed most of the time and occasionally had a couple of fails. Unit testing gave 23% coverage. The minion card allowed both options to +2 coins and +4 cards randomly. Most of the times it was used correctly though. The last test had a 64% coverage. When the gardens card was used, it would add up all of the cards in play and not only the cards from one player. This gave all players a lot of points and have a false winner. This code is not reliable to be used.