David Taylor  
CS 362: Software Engineering II  
Test Report: #1

**Approach**: My first approach to writing a test report for the first time, I took the approach of doing a type of unit testing. When testing BuyCard, I took 3 approaches of tests. Firstly I tested whether the BuyCard implementation checked whether the player had enough buys in order to make a purchase, whether the player had enough coins, and whether there is enough in the supply count in order to buy the card. So I checked corner cases, when there was 0 buys remaining, when you had infinite or 1000 buys, when you have 0 coins, or infinite, or when there is enough of the supply to buy or not enough. So basically, when the testSuite runs, it will output the attempts to buy cards and tell you if there is anything wrong with the outputs. I did not use any assert statements, nor any random testing. I merely used a bunch of memcmp to check the game state to what it should be. Also, I do have a general testing scenario, which checks to make sure that the player’s coins, numBuys and discardCount are changing more than they should between buys. I will now describe exactly what my 3 types of tests do.  
 *testNumSupply*: this specifically tests that the supply of the card being bought is decrementing properly, and that the right card supply is decrementing properly. Also makes sure that the right number of coins are being subtracted from the player, and that the discard count and order are equivalent to what it should be.  
 *testNumCoins*: This test looks at whether there is enough coins, and specifically looks at corner cases, when there are no coins and the user can still buy cards of cost 0. Testing whether the buying of high cost cards with more coins is done in the other tests.

*testNumBuys*: This looks at the corner cases of buying, when the player only has 1 buy and 0 buys. The other cases of buying is covered in the other tests. It checks to make sure that when the player has ran out of buys it will prevent them from buying again. It does the correct purchase for when the player has plenty and there is plenty of supply remaining.

Finally, my test Suite checks to make sure that the memory allocation of the correct game state is the same as that which is produced by the implemented dominion code. This is merely a memcmp of the correct game state with the expected and correct game state.

The test suite doesn’t explicitly test the coverage of the dominion code, but based of my implementation, and looking at the code implemented by each person’s dominion and what my own test suite code covers, I appear to reach approximately 30-35 lines of code per test run. Many of these lines are hit more than once.

**Tests**: I first ran it on my own code to make sure my tests were correct, then I ran it on others code.  
 *My code*: I ran this on my own code, and noticed that my own buyCard function doesn’t actually decrement the supply properly, nor did increment the players deck count properly. Plus I noticed that it would only attempt to buy a card when the player had buys not equal to zero. So if it was hard coded that the player had less than zero buys, it would still buy it. So I changed it to buy only when it was greater than zero. I also made modifications to the buyCard so it decremented the supply count properly and the coin count properly. I also noticed that my implementation doesn’t like it when there are zero coins. I will look into that, but the reason I believe is I have a conditional that checks on when there is more than zero coins buy something, not when there is zero or more. Now that I have it working on my code, I plan to test it on other students’ code. It gives accurate results about where something is wrong.  
 *lesliew*: When I ran my testing code on lesliew dominion code, I noticed quite a few problems. In my first test of BuyCard Test Coins, I noticed that the count of coins was never equal to that of the test or of the expected value. When buying a card, the buying function doesn’t decrease the coin count of the player by the correct amount. Each time it was off by -1, meaning that they have more coins left over than they should from a buycard. When running my BuyCard Test Buys, I see some more problems. Not only do I again see the problem of the coin count being off, I also see the numBuys being different from the expected value. Not only this, but also the discard count for the player is not what is to be expected Finally, running my third type of test BuyCard Test Supply, the results are similar. The coin count is not decremented properly, the numBuy count and discard count are not changed correctly either. Another problem is that the supply count of the cards of each type are not decreasing properly either. The code must not be decrementing the supply of the each type of card when they are being bought. Looking closer at the implementation, I notice that they only implement buyCard on coin cards, i.e. silver, copper and gold. Therefore, they don’t take into account cards of no value, like curses, and they don’t take into account the other types of cards on the field. In order to fix this, I would recommend looking generally at buying cards, and not buying specific cards. Look specifically at each card’s value, decrement the player’s coin cost by the cost of the card, even if it is zero, decrease the buy count, increase your discard count, and decrease the supply count, and this should solve a lot of the problems that are present.

*Shearini*: When I tried my test suite on this person’s code, there were only very minor faults. I noticed that when I ran my BuyCard Test Coins, the only problem that occurred was the fact that the numBuys left for the player did not match what was expected, it was off by 1. The correct of coins left was correct. Whey I ran the BuyCard Test Buys, the same problem occurred, the number of buys was off by 1. Finally when I ran the BuyCard Supply Test, the same fault occurred, the numBuys of the player was off by the amount of times the attempt of a BuyCard was ran. The supply count was correct though. This implies that the problem lies in the fact that the player is not decrementing the number of buys correctly for a player. Taking a closer look at the implementation of BuyCard, I see the problem. The numBuys of the player is actually never decremented. The easiest fix would be to add one line which decremented the count of the buys for a player. Another problem I noticed as well is that the buyCard never takes into account the possibility of buying a card that has a value of 0, like a curse card. This can be changed by altering the coin check conditional from [if (state->coins < getCost(supplyPos))] to [if (state->coins <= getCost(supplyPos))]. Once these changes are made though, newer faults could occur, so I am not saying that these are all the faults present.

*Ellingsn*: When I ran my testSuite on this person’s code, I noticed a lot of errors. When I ran my BuyCard Test Coins test, I noticed that the player’s coin count is decremented properly. However, the player’s discard count is off by 1. When I ran my BuyCard Test Buys, I realized some more problems. The player’s coin count is not always decremented properly, and the player’s numBuys are not decremented properly. Finally, my BuyCard Test Supply came to the following conclusion. The numbuys is definitely not decremented properly, the player’s coin count is also not decremented properly. Taking a closer look at the implementation, I notice that there was no change to the original code, and so the implementation is not correct at all. An attempt by this student needs to be completed first.

**Plan**: I would like to incorporate more random testing. I would also like to do some coverage testing and use some of the testing tools described in the first few lectures. I would also like to break my testing down even more and add more user friendliness to it, and make it print out results that are more useful, and not just telling the user that they bought a card but it failed, more precisely how it failed, and if possible where it failed.