Khue Tran
CS 362
Spring 2015

Dominion Test Report

I had played Dominion a couple of times before taking this class, but that was about five to six years ago so I have no idea how to play Dominion now. It was really difficult at first writing test codes for our dominion.c file because I was struggling with understanding game rules and card effects. A looking around online, I was able to find a basic instruction to Dominion gameplay at http://boardgames.about.com/od/Dominion/a/How-to-Play-Dominion.htm and I also found http://wiki.dominionstrategy.com/index.php/ which was a good site to look up card costs and effects. However, even with those resources, I still wasn't able to get a grasp on how the Dominion game really function so I looked for an online version of the game to test play it myself (the website for dominion online is at https://www.playdominion.com/Dominion/gameClient.html). After playing the game myself but battling a computer player, I understood the game a bit more and felt more comfortable designing a test code for the whole game.

For the assignment where we had to write test codes for individual functions and individual cards, it was a lot easier to implement and find bugs than trying to find errors by playing out a whole dominion game in testdominion.c. With the smaller tests, it is easier to find the bugs present in the code because I can examine the function or card more closely and spot out anything that was incorrectly implemented. Big tests are useful because it covers more of the code so you can test out a big portion instead of small chunks at a time.

Testing My Own Code:

Unit test & card test result from unittestresult.out:

My 4 unit tests covers these functions: compare, supplyCount, fullDeckCount, gainCard.
My 4 card tests covers these cards: Smithy, Council Room, Mine, Stewards.

These tests covered 21.10% of 564 lines from the dominion.c code. The 4 unit tests for the different functions all passed and the tests for Mine and Steward passed also. When the Smithy card is played, the player would draw 4 instead of 3 cards causing an incorrect handCount and the test would print an Assertion Failure message. It then look at the player's deckCount and again would see the incorrect number and throw out another Assertion Failure. The next card that failed in this test is the Council Room card. After it is played, the player draws 5 instead of 4 cards and gain 2 buys instead of 1. The test code checks for the player's handCount, deckCount, and numBuys so when the number it's expecting is incorrect, it throws some Assertion Failures like in the last test.

Full game test from testdominion.out:

In this test, I used a random seed, a random player count, and a random set of 10 Kingdom cards. I have an Action phase, a Buy phase, and I assumed that ending a turn using endTurn() would make sure the player would draw the correct number of cards (if this assumption is wrong, then my testdominion.c code was not written correctly).

The test covered 59.22% of 564 lines from the dominion.c code. The coverage is greatly improved compared to the individual unit tests. The problem here however is that trying to find bugs in the code was a lot harder to write a single test for because the same code has to work with every possible cards and so it was a lot harder to spot out the bugs too when looking at my gameResults.out file. This means that I would need to add more checks in the testdominion.c file if I'm not seeing enough information in the gameResults.out.

Testing Anita Chow's Code:

Unit test & card test result from unittestresult.out:

My 4 unit tests covers these functions: compare, supplyCount, fullDeckCount, gainCard.
My 4 card tests covers these cards: Smithy, Council Room, Mine, Stewards.

These tests covered 21.14% of 563 lines from the dominion.c code. The 4 unit tests for the different functions all passed and the tests for Council Room, Mine, and Steward passed also. This time around, when Smithy is played, the player draws the correct number of cards. The player's handCount is correct, but when checking for the deckCount, the number is 1 less than what it was supposed to be which means there might be something wrong with the implementation of the Smithy card or something else is bad. This only found 1 possible bug and most likely, there are a lot more bugs in the dominion.c code that these tests do not cover.

Full game test from testdominion.out:

In this test, I used a random seed, a random player count, and a random set of 10 Kingdom cards. I have an Action phase, a Buy phase, and I assumed that ending a turn using endTurn() would make sure the player would draw the correct number of cards (if this assumption is wrong, then my testdominion.c code was not written correctly).

The test covered 57.02% of 563 lines from the dominion.c code. In this test, I get Assertion Failure when playing a cards that are not supposed to fail like Treasure Map and Sea Hag. This shows that the dominion.c code is not working correctly but it would be hard to pin-point the actual cause without doing more individual tests on these cards. Also this test does not tell you if a played card is giving the right effects or not. I think this would be a debugging message that I would want to add to the card itself in the dominion.c file so it could be better suited for that specific card or function.

Testing Alec Haagenson's Code:

Unit test & card test result from unittestresult.out:

My 4 unit tests covers these functions: compare, supplyCount, fullDeckCount, gainCard.
My 4 card tests covers these cards: Smithy, Council Room, Mine, Stewards.

These tests covered 21.29% of 559 lines from the dominion.c code. The 4 unit tests for the different functions all passed and the tests for Smithy, Council Room, and Steward passed also.

When Mine is played this time, I get an Assertion Failure when I try to discard a curse card for a gold. If the assertion passes, it means that Mine was played successfully and the player was able to discard a certain card and get gold by default. If it fails, then it would throw an error. For this test, I only received the error when attempting to discard curse for gold and passed it when attempting to discard copper for gold and silver for gold. This means that either the implementation of Mine is bad or my test procedure is wrong.

Full game test from testdominion.out:

In this test, I used a random seed, a random player count, and a random set of 10 Kingdom cards. I have an Action phase, a Buy phase, and I assumed that ending a turn using endTurn() would make sure the player would draw the correct number of cards (if this assumption is wrong, then my testdominion.c code was not written correctly).

The test covered 53.31% of 559 lines from the dominion.c code. In this test, I get Assertion Failure whenever playing the cards Remodel, Ambassador, or Garden. When looking at the source code dominion.c, I can see that those 3 cards are implemented correctly, they just return values different than what my test was expecting so it thinks the card failed to play. This means that my implementation of testdominion.c is not a reliable way to check for failures in dominion.c because it does not tell you if a card effect is incorrect and only checks for returned values of 0, otherwise it would fail.

Conclusion:

Just the difficulty that I was having in writing the test code because I did not understand the game mechanics too much was a big hindrance. It makes me think that a software testing job must be pretty time consuming and difficult if you are thrown different codes to test for bug and debug constantly and you would have to learn in-depth how it works to even start testing it.