Keil Boring
6/3/2015
Part 4 Final Project

Initial Thoughts

From the start of this class it was a daunting experience because I had never played or dominion before. I have heard of it and everyone says its a great game. So before I started looking at the code and trying to understand what was going on, I learned how to play the game on dominions website.

Once I knew how to play I started looking at the code mostly dominion.c and dominion.h . I thought it was very clever the way the game stored everything as a int because this removes the confusion of data types throughout the program. Mostly you just have to worry about whose turn it is and if that part of the state is an array. I also liked the way the cards were stored as a enum from curse=0 to treaure_map = 26 to store all the cards because this gave the code a very verbose language and it was really clear on what was trying to be done at all times. From here looking I looked at playdom.c for creating most testing programs after because it showed how the API should be called in a two person game with two cards keeping it very simple. Overall I thought the initial design of this was game was really well done in keeping organized what was going on and where everything was at.

Testing Experience

The first assignment was introducing us on how to create bugs so later on we would have a function that we would know where to find them and learn from this process. We did this by going into dominion.c and creating bugs inside of two cards. This was useful so that we had a better understanding how each card was called as a function and how to use cards with choices. It also showed how bugs can easily be made from a user and how much work it can sometimes be to find them if you don't know what is causing it.

The second assignment had us do unit testing to find bugs. Unit testing is used to target small parts of the source code and trying to force out common bugs such as corner cases. Unit testing is also used to target weird values that you as the programmer think would cause a problem that the program might error out on or cause unwanted behavior. In essence unit testing is used to test chunks of code or functions to see if they behave the way they were intended and return the targeted results. So for our assignment we used unit testing for two things testing four functions and testing four cards. This gave me experience in finding what the edge cases would be for parts of game such as making sure use max number of players and filling a players hand with max number of cards to try and find a bug. I used many for loops to get these values that would then call a separate function comparing the previous game state to the current game state to make sure the correct behavior occurred. If the correct behavior didn't occur it would then assert the error telling the user what value was off.

Third assignment was doing random testing. Random testing is done by randomizing certain input to a particular function This is useful when its hard to think of edge cases or cases to test in general. We used random testing on two cards. Then by giving random inputs for certain states of the game such as number of buys, coins, and cards we have a random tester. This gives a good overall coverage test without having to resort to exhaustive testing where time can become a problem and make test in-feasible.

Lastly our final assignment we wrote a random tester that plays complete games of dominion. This test is less focus about if a card or function has the right behavior but if the overall game works given a random set of cards and players. By printing out key information at each step in the game someone could follow along in real life and play the same game. These test checks if the game states make since while covering many combinations of the game without exhaustive testing. We then compared our output to another classmates. This was useful because it showed how even by doing different things internally by using the same prints of game states our test we were doing close to the same thing.

Test nguyehai code
I tested nguyehai code using my unit test from assignment 2 testing four functions and four cards. When testing all of them passed besides one of the cards and that was council room which got caught in a infinite loop. Next I tested it using my assignment 3 randomtester and that passed (code coverage of only 20.71%). Finally ran it with my assignment 4 and comparing the results between my .out file and seeing if any differences had occurred. Found a bug with Feast card does not decrements numActions correctly and for some reason increase the number of coins in hand by 4.(code coverage of this test was 58.79%). I could increase the coverage by running testdominion.c multiple times for different set of cards to increase code coverage but since I have found two bugs already I stopped here. His code built and ran fine so it does play the game and still gets a result but not all cards behave the way we expected them to so in essence it is some what reliable.

Test hansonm code
I did the same procedures as above when testing hansonm code. All my unit and card tests passed beside cardtest2 which was village because actions didn't match. When running it with my assignment 4 and comparing the .out files I found three bugs. Village card increments number of actions only by one. Remodel does not decrements number of actions. Tribute only gives +1 gold versus the normal +2 gold if a neighbor draws a treasure card. The code coverage I got when running assignment 4 was 72.04% . This code is pretty liable and more so then nguyehai code because there were no infinite loops and the games were still playable. This code has quite a few bugs and would need some more testing and fixing before being released to the public.