Erik Watterson
onid: watterse

# Dominion Report

**My experience with the assignments**

Overall, from what I can tell as an inexperienced software tester, is that the dominion code could not be wholly reliable as a means to play the game Dominion. My evidence for this conclusion is based on several bugs that I've found within the dominion code, the outputs that I've received while testing individual cards, and the code setup I had to perform while setting up my full game random test suite. Though you could play several games of dominion relatively fine, I don't believe that it outways the overall problems. Below is my reasoning and evidence as to why I feel this way.

The first thing thing that I looked at while debugging my dominion code was unit testing several of my dominion cards. What I had found while doing these unit tests is that the struct state variable discard_count was never used by any of the Dominion code functions. The way I found this was by attempting to use the cardEffect() function which contains the discardCard() function which should keep track of the cards that I've discarded. So I decided to try and find any instance of "discard_count" in the dominion.c code. I didn't find a single one. This implies that any functionality that intends to interact with global card counts, or anything relating to the discard pile, needs to re-evaluate the size of the discard pile every single time that it wants to interact with it. This problem could of been easily avoided if inside of the discardCard() function, it was added that discardCount[current_player]++.

In the third assignment, we were tasked to produce randomized tests for single cards. Before I even started testing, I had analyzed a file that was left in the SVN that was named randomtestadventurer.c which was named after the file I needed to create. When I started analyzing it, there was an issue where if you ran the same gamestate multiple times, it would seg fault. After manipulating the file for a while, I couldn't find a way to fix the issue within that file, so I started from scratch to see if I could somehow reproduce it. After getting a working test going, I found that I could not reproduce the error and I wasn't able to trace the issue back to it's source. Though my implementation worked, the previous implementation did show that there was very few conditionals that checked for error handling and this would be a prime example as to why error handling should always be implemented.

For the last assignment, when I was assigned to construct a fully random test suite it was apparent how poorly implemented the structuring of the code is. Though this is subjective, I'm under the impression that the gamestate struct variables should all be interactable by only function calls to dominion.c. I have this opinion mainly so that more error checking can occur for the programmer, and this leaves less room for error when the programmer intends to use the calls. So, by reducing the number of directly accessible variables cleaner implementations using the code can be made. An example of this is would be trying to access a variable such as gamestate.playedCards[]. Since it's an array, the ease of causing an out of bounds memory error increases significantly when the programmer interacts with it directly.

**My experience with my classmates code**

Setting aside my the issues I've run across in my own dominion implementation, below is my evaluation of 2 of my classmates dominion implementations. This includes an evaluation of the codes bugs, coverage, and reliability.

Padraig Gillen:

For Padraig's code, both the bugs that had been found were bugs that seg faulted the code. Meaning that if you were to try and construct a game using the code, there is a good chance the game would prematurely end. Both bugs switch the hand position iterators with the player iterators which means that if your hand count is larger than your player count, you'll seg fault. Since the test was ended prematurely each time, a true code coverage was unobtainable. Again, this isn't reliable and I wouldn't consider it playable.

Elya  Christman:

For Elya's code, none of the issues I came across seg faulted the code so I'd say that from that aspect the code is reliable enough to enjoy a game of dominion. The issue I did see is that she had a switch statement clause return an error every time the adventurer card was ran. If this was implemented into a way to actually play dominion, the likely case would be that the user would be told that the card failed to be used, but you would still see the cards effect in your gamestate. It may be a slight inconvenience  but it is a playable bug. Other than that single bug, the code coverage for the implementation reached an overall of 50.35% coverage from gcov for 50  tests. Most of the locations that it didn't hit were functions that were intentionally left out of the test suite due to either their simplicity and there were some cases where the cards weren't covered due to random chance. Overall however, it didn't seg fault, there was only one erroneous find and the game was finishable, so I rate it reliable.

**Conclusion**

To reiterate, the Dominion implementation could be used to play a simple game of dominion, but I wouldn't expect to see the correct results of the game even with the original build that hadn't been tampered with by the students of this class. I think that code if poorly implemented and there is too much of a risk for even the original build to end prematurely. Though I haven't seen it do so, some of the loose array calls made me nervous going forward. With that said, I think it was a perfect obstacle course to learn how to test properly.