Adam Puckette
CS 362
Spring 2015
932204993

Final Test Report

Let me begin by saying that the dominion.c file provided to us this term has been very fun to poke through. I see plenty of bad code, but most of it is code that just doesn't work or isn't feature complete. Dominion was a different beast. Just how the code manages to cling together in a zombie-like approximation of function amazes me. I learned a fair amount more about coding just by deciphering it. Just figuring out where to begin was a challenge.

To be honest, I have not and probably will never play Dominion. I have seen it played, and through the code I now have a pretty good understanding of how it works, but I never got the chance to actually try the game – I was too busy testing it. Assignment #1 was easy enough once I actually got to the coding, since after CS 361 I felt like some kind of refactoring King. The only real challenge was getting an environment set up enough to actually start work. That's pretty standard for college, though. Every professor/person has their preferred programming style/language/IDE/editor. Adapting to changing styles is nothing new, and it's an invaluable skill for a programmer. That said, the abomination that is cardEffect() took a bit of getting used to. I mean c'mon! Three arguments named choice# and a 600+ line switch statement! The last minute requirements called for two bugs to be put into the program, which I found to be a refreshing change of pace. That said, I suck at intentionally introducing bugs in strange code. I obviously need more practice.

Assignment #2 was a bit easier since I had gotten the environment set up already, and again, CS 361 had me more than prepared to write a few unit tests. Part 2 was easy enough, since what works to test one card can serve as a pretty good base to test the others. I had to format cardEffect() to my standards so I could read it. Part 1 was the real challenge, since all the functions I was testing were radically different. InitializeGame() proved difficult to get coverage of due to the multitude of different failure conditions. Speaking of coverage, this was my first time using gcov. It's a pretty easy tool to use as coverage tools go, even if it doesn't have the shiniest interface. I used it to root out all the corners of initializeGame() I was missing. I managed to find and root out all the bugs I had introduced in the first assignment, which I considered to be a personal failure on my part.

Assignment #3 wasn't as hard as I thought it would be, though I admit to doing a fairly poor job. The random tests I wrote ended up being little more than smoke tests. I found it was a bit more difficult to get coverage of some of the more unlikely outcomes, since it required biasing the random output towards certain outcomes.

Assignment #4 called for complete random games of dominion, which required a lot more coding than the previous assignments. Since all the card functions require vastly different inputs, I had to make do with fewer cards and have the players randomly switch between the different card-centric strategies. Randomizing the kingdom cards was the hardest part, but after modifying a Fisher-Yates shuffle to shuffle between two decks, one being filled with -1's and the other a complete set of all possible kingdom cards, I managed to create a random setup that would avoid duplicates. The one thing

I would do to improve the design is make it so cards not in the kingdom cards list cannot be purchased or played. I used colliell's dominion implementation to test for differences in output. I didn't expect to find any, since we haven't been required to change much of dominion's code. Much to my surprise, when I used an online diff checker on both .out files, I found a few extra lines of output in colliell's output. Upon further investigation I found out that colliell had her code set to debug mode.

The current state of my dominion code is fairly poor, but I'd like to think it's a lot more readable at least. I spent quite a bit of time just getting it formatted to a state that I can read easily. I was only able to achieve 40% code coverage with my tests, but the functions I did test were very well covered, so the problem wasn't in the test I wrote – I just didn't write enough of them to achieve total coverage. Since we weren't told to actually fix the bugs we found, I mostly just left dominion.c alone, and from the state of most of my classmate's code it seems I was not the only one to do this. Colliell's code was pretty much identical to my own, with the exception of the debug statements and formatting. Aside from the refactored cardEffect() cases, colliell's code is still a nightmare to read thanks to the lack of proper indentation and spacing. As for reliability, her code is much the same as mine. Which is to say, terrible.

I've worked with liawb (Bryan) before, and I have to admit to being a little surprised to see his code. I suppose I expected some sort of polished, gleaming, Godlike dominion implementation. Instead his code is much the same: unformatted and chock full of the usual bugs. He also messed up the refactoring such that the return values are not returned through cardEffect(). I must say I expected better of Bryan. Aside from the aforementioned bug, his code is just as functional as mine. Which is to say, not very.

Overall, I learned a lot this term, and I think I can safely say that if I never see code as inconsistent as dominion ever again, it will be too soon. Still, it was good to learn about all the other forms of testing out there after my CS 361 class focused entirely on eXtreme programming to the detriment of all the other forms and methods out there.