Anita Chow

931-733-076

Final Report of Dominion Testing for CS362

My experience testing Dominion was fairly interesting since I had next to no experience using debugging tools other than print lines. I've always had trouble understanding other people's code, something about the fact that I didn't experience the general thought process that went into creating the function makes it hard for me to quickly grasp what is being done in any semi-complicated function. This class was a good learning experience in deciphering code and improving my static analysis skills while learning about different and more advanced debugging tools. I found gcov a handy little tool that I did not know about going into this class (among other tools). I like being able to know just how much my testing was actually affecting the code or how much of the code is made up of a function. It did feel a little frustrating trying to find more ways to make to increase my code coverage. I ended up using it as little progress indicator for how well my testing was going and found it was a good motivator to keep adding more functionality to my test to reach that mythical 100% coverage. I do wish that we had used a larger variety of debugging tools. I remember during one of the seminars, the speaker was going over the different debugging tools he used and we only had a small idea what some of them were. More demos showing the different tools would have been useful because going into the final project, I had no idea what Tarantula was.

For my classmate's implementation, Khue Tran's code her unit and card tests got about 21% code coverage which I would consider quite good for simple unit and card testing. The cards she tested were Smithy, Council Room, Mine and Stewards. The unit tests were compare, supplyCount, fullDeckCount and gainCard. I'm not really a fan of the unit testing method but it is useful for testing small snippets of code temporarily to make sure that in a very basic case it performs as expected. In Khue's case, she gives a very specific set of cards to each players so it sort of overrides some of the functionality of the game initialization but does do a good job of insuring that the outcome will be constant. Also it allows some later customization, albeit not very friendly or quick refactoring of code but it could be done at a later point to do more special test case testing. Her testdominion.out covered about 60% of all tests which is quite good. The coverage is significantly better than the unittesting. Also the scoring system would randomly jump values seemingly without explanation. More in-depth investigation would be needed but the fault is mostly due to that specific dominion.c code since it doesn't happen with my dominion.c implementation and I did make several changed along the way to fix the scoring system being buggy. Also implementing a way to properly perform multiple action cards in a row. A lot of action cards give you the ability to use more cards but creating a way to intelligently pick more as well as choosing the correct choices is probably outside the scope of student possibility while taking this course. Also the bug of several cards causing hangs would be fixed by proper choice values being give to them. Overall I would consider her code adequately reliable, for the scope of this class I would consider her code great.

For another classmate's implementation, Alec Haagenson's code his unit and card tests got about 21% as well for coverage, which again is not terrible for basic unit and card testing. The cards he tested were adventurer, smithy, council room and great hall. And the unit tests he chose were fulldeckcount, numhand, initializegame, isgameover. I found his tests to be much more focused on the outcome of hands played such as the resulting value of hand counts compared to initial hand counts. A different approach to unit testing but still valid, might be prone to unforeseen situations popping up. His

testdominion.out covered about 54% of code. His testdominion code also had the same issue of the scoring system being sporadically changing values despite no cards being bought that would have affected it. The issue it again most likely that specific dominion.c implementation since it works fine with my implementation. Again a function that would dynamically pick action cards would be great but probably not implementable in the time of this class. Also the bug of several cards causing hangs would be fixed by proper choice values being given to them if this was implemented correctly. Overall for I would consider his code adequately reliable, for the scope of this class I would consider his code fine.

In conclusion, I found this class an interesting experience and would definitely be helpful later on since I plan on going the computer engineering route versus other electrical engineering routes. My thoughts on the class is that I like that fact that we took one piece of code and worked on improving it the entire term. It always irked me the fact that after we finished an assignment in more computer science classes we would never even talk or look at the assignment again. I do wish there was a either a bigger push to change the base code or don't let anyone change the base dominion code. An error I ran while doing assignment 4 was that when comparing my code results running with my dominion.c with other people's dominion.c there would be a huge amount of results in my diff output. After a lot of frustrating hours, I finally figured out why it was running weirdly because I made changed to the original dominion.c to improve the scoring and drawing while there's did not leading to a huge difference in how the scoring was done. Overall this class was fun, but just like any other class I'm glad it's over!