Andrew Johnson

CS362

Project Paper

# Test Report

To start, it was very interesting learning about the game of Dominion. I've never actually seen or played the game, so learning a new game was fun. Especially one that I could make on the computer. The first couple weeks I was in over my head learning how the game worked and how to properly use the cards. I'd say I was fairly confident in refactoring the functions in Dominion because it was just stripping out code and replacing it with the function and parameters it needs. The tricky part about the refactoring is having the check if the function is doing what it's supposed to and if it's affecting the game like the effect says it should. Some of the easier functions I remember testing where the basic game functions, such as drawCard, kingdomCards, and getCost. While testing these I always was curious one how they worked too. So I always had to go into the functions and understand how they work.

Some of the tricky parts of this class was the different ways of using a makefile. I was very unknowledgeable on how to do anything but compile code with makefiles. So creating .gcov, .out, and other files where intimidating to try to get done. One of the big things I remember from class is even though your test might be great, it might not be looking at the right things your function is doing. So even after all the tests, I had a slight distrust with them.

One of the reasons, is because before this class I wasn't sure even how to create tests in the way we have. The only kind of tests I knew of doing was manually inputting everything. I know this is a kind of unit test, but it's a long a tedious one. I was very confused the first couple of weeks when we started talking about testing. Even the in class examples of unit tests and random tests where confusing. So I had to go to some of my class mates and ask how to even start one for a project like ours. They were helpful and once I was shown one way I started experimenting with other ways. Then I the hang of unit tests and looked up how random tests go. I realized they are essentially unit tests but you just use a bunch a random inputs and see what happens. It got exciting once I started running them.

Although, I'm finding tests to be very useful in just bashing my code with inputs that I may never even think of doing. Before this class I knew about checking the edge cases. I thought that after checking the edge cases then the function or program would work fine. Yet after using random testing and seeing it used in the class demos, I realized that there are many ways anything in the middle of the code could be just as broken as the edge cases. I enjoyed having to go from unit testing and the mind set of focusing on the main parts of how the function works, and then going into random testing where you just let the computer input any silly thing and let it spit anything it doesn't like back out at you. I remember how useful random testing is because it got much more coverage of the code then I code with the unit testing. This again is where unit testing is for focusing on a part of a function. When doing both types of testing I noticed that my unit tests would get about 50-70% coverage. Whereas when I set up my random tests I got up to 80-90% coverage. With random testing, I just put all the bounds in for everything and it seems to give excellent coverage on the code.

One thing that stuck with my about random testing is that there is never 100% when it comes to coverage. I forgot the exact reason why this is a thing, but I do know that if you obtain 100% coverage, you must have a small amount of code. With a large amount of code, you would have to write a test about the same size as the code to get 100% coverage, and with that the code you wrote for the test probably has plenty of bugs itself. So I now the golden area for tests are around 70-90%. I was very excited when I was getting such a high amount of coverage with my random tests. Sadly, I never got the same when it came to testdom. I believe I only got up to 75% or close to 80%. This only strengthens the statement about how large amounts of code won't be able to be covered 100%. I was having trouble getting up to the 70's. After a good brainstorm I realized I wasn't putting all the cards in or giving it a chance to. So in my random test for Dominion I recycled the kingdomCards function and gave it a new deck each time. After that was in place was got the higher coverage I stated earlier.

If I was to pick a classmate's Dominion code that worked phenomenal I'd say leete nailed it on the head. When I was testing through his code it seemed like he had a good understanding of the rules since all of my unit tests cleared the code, along with my random tests. My random tests even achieved up to high 80%. I'd highly trust his code, and possibly choose it over mine. Another students code I was looking at was jiangzh. I remember testing their code and seeing some bugs in were the drawing was off with some of the cards. After testing most of the cards, it seemed like they got the bugs fixed and in order. Then I went on to test more and I was achieving up to 80-90% with theirs as well. It seems like they got a solid Dominion code in their repository to where you'll have a pretty fair game of Dominion.

Overall, this class was pretty stimulating in what we should learn and how we can go about tackling testing. I was overwhelmed with the expectation of knowing some higher level coding terminology, but it wasn't going to hold me back. I'm hoping the assignment feedback is more quick next year. Other than those this class was a great introduction into the life of a debugger. It's not the most fruitful of jobs, but this class helped me realize it's probably one of the most important jobs of programming.