Mark Andrews

Test Report

Testing is something I'm not entirely new to. I've made a few testers but mainly for TA purposes to test for cheating students. I have also used diff testing for assignments to make sure my output format was perfectly correct to meet standards for the class. However, even with this background there was a lot out there that I had no idea existed. Coverage test tools and mutations are something that I think are amazing, and I'm very happy to have gotten to know them and touch them slightly. However, there is a lot more I have to say about testing.

Testing I feel can vary a lot depending on what you are testing, and this is where I slightly dislike testing. Testing Dominion, or more specifically this version, was not all that entertaining. There was SO much wrong with it, and the structure of it was so bad that it made testing it less about learning to test things well or get good at testing, but more a task in learning and understanding terribly written code that is still broken. The coverage alone on testDom and most people's random testers definitely shows this to be true. Honestly I'd have rather programed my own dominion and had to run testing as I went to verify it was working correctly.

I'm saddened by the lack of mathematical rigor used in testing as many mathematicians and original computer developers used math to support their programs and processes. I've read a lot on mathematical proofing of programs, as a math major, and while we didn't use it I still feel it holds good value in testing, even if it may not be "practical" as deemed by the same people whom get coffee as their computer *hopefully* finds the problem.

Anyways, coverage is an amazing tool as it completely eliminates things. The fact that you can't test what you didn't cover is perfect to narrow down what you are testing. While this doesn't help since the lines covered can still be good or bad It's still an amazing tool in un the process of testing coupled with other methods. Tarantula is a beautiful system, however I feel it can be improved. A possible tarantula improvement would be to look at the percent of failures that occurred in a function and look for that percentage in the lines covered. There should a be a correlation in this, because if a function fails 60% of the time, and the lines are not all covered equally, then we know it's not a logical statement and it is then a line that was only covered 60% of the time the function was called. This would be a lot of fun to create and test.

Back to the point of dominion. This game in itself is a very good game for testing purposes as it’s one of the more complex types of card games out there. It takes a lot of strategy and some card affects are complex in terms of incremental amounts. In testing it became vital to play dominion a lot to get familiar with it in order to test things easier. Sadly I almost feel the creators of the dominion program we tested had never played dominion in person as it was organized so terribly.

Coverage of dominion was something that was hard to gauge exactly, because depending on the kingdom card selection some sections of code pertaining to cards not included would always be left out. If every card was the same amount of lines we could then notice that even with a perfect tester you would never get 100% coverage. This is another nail in using coverage as a testing element itself. As stated before coverage is a good tool to couple with other testing methods.

In the end the best testing method of differential testing almost feels theoretical. If we are making something new, but wanting to test it perfectly, then we must test it to what it should be, but that doesn’t exist as we are making it, such as a philosopher’s proof of god not existing. This leads to what I believe are flaws in people’s methods of creating code. Understanding the system is far beyond the first step in testing as it should be the first step in anything. Because the card game exists of dominion if someone has a good understanding of it, and they print game state after every action in the game, someone that is well informed on the system could spot most all flaws and errors that occur. Note that this is even without differential testing as it doesn’t actually rely on the card game existing. Understanding the system of what is to be created before creating is the true key to create something new and know how to test it without having it finished.

In testing others dominion code I wasn’t surprised by anything I saw really, except for the large large attempts to make since of such a broken system. While it’s not too bad of a system I still highly disagree with its layout. Besides this I’m not sure if I understand testing well enough to say if what they did was good testing, or good work ethic in their attempt to fix the program. I feel this only detracted from my ability to properly test their code and my own code. While someone else’s random tester may have found more failures or covered more code I can’t tell if their tester is better than mine or also giving false reports. It also doesn’t solve the issue knowing and understanding the system to fix it. I say this because the order that using and giving actions is done is not consistent through the code. This detracts from having any tester for actions that works for the entire system.

This leads to my last issue in my understanding of testing, and that’s if the first step fails in understanding and having a good system then changing one thing at a time can sometimes give less failures while not having fixed the actual issue. It’s like part of a system not getting power so you hookup a generator to it instead of fixing the actual power to the system.

I liked testing though, and learned a lot from this class. I have more questions now than when I came into the class. As for this dominion version I feel it needs to be scrapped and re-designed in order to make since. But that’s if we were actually testing it and trying to make a fully functional dominion that works.