Brandon Ellis

Testing Dominion

Through this course I got many opportunities to expand on my knowledge of testing from Software Engineering 1. This went from reviewing unit testing that had been covered in the previous class to learning and implementing random testing and the generation of a simulated user.

When this course started I was able to ease my way in with the refactoring assignment, in which I correctly refactored sea_hag, smithy, and village while introducing errors into the refactored versions of great_hall and salvager. But it was the second assignment that really helped me get my feet on the ground. In that assignment being able to take my prior experience of writing unit tests in Java and apply it to this assignment helped me first apply something I was already familiar with to this class, but also become familiar with how a similar concept was done in C. Though a decent amount of time on that assignment was spent struggling with the Makefile I was eventually able to complete unit tests for the functions getCost, numHandCards, drawCard, and buyCard and tests for the cards smithy, village, embargo, and finally adventurer. Through all of this I was only able to get a code coverage in dominion of around 20%. Looking back I think this was because I failed to test outliers in the cards and simulate them being used by players in different scenarios. If I had approached this more like I did assignment 4 and simulated a player using this cards it would have been a more verbose test, but then it may have exceeded the realm of simply being a unit test.

From there we went to assignment 3, which I think I ultimately gained the most from. While writing a test that handled one specific thing like a unit test, but stress tested it with random occurrences was more challenging, in the end it seemed the most effective form of testing I learned. In this assignment I tested adventurer, smithy, and embargo, this was able to push my code coverage up into the 30% range. This leaves the last assignment, simulating a played game.

Assignment 4 was definitely the most time intensive of all of the assignments, and though I did learn quite a bit from it and though it did achieve a coverage percentage in the high 30’s, I felt like it lacked in strength in comparison to the random tester of assignment 3. Assignment 3 had a much more narrow scope as was just testing one element similar to the unit tests, but the random simulation of it seemed quite verbose. In contrast, while the simulated game of assignment 4 did cover a lot of code and did cover many different aspects of the code because of its randomness, it felt significantly more shallow. I believe this was because the code of assignment 4 ended up testing how I thought the game should go, not how it could go. The idea of a simulated run of a program is one that I very much like, but I believe that its biggest downfall is that the program can only be as random as the programer makes it.

Testing Classmates

Longmane

To test the reliability of their code and the coverage that they had achieved I first cloned their distro. From there I was able to run their Makefile and go assignment by assignment to first see how their tests ran and then to determine their code coverage. I am choosing to omit the classmates and the next's refactoring results as I did not see the point in referencing them in the testing of their code. The first test that I looked at was their unit test, which ran fine and resulted in a test coverage percentage in the high 20's. I looked at the code for several of the function and card tests to see how they worked and what they were testing for. Overall it looked like they had, more or less, the same issue I identified in my unit tests above, the tests ensured that the cards themselves and their core functionality was working but may have omitted some of the outliers that could occur. The other test that I ran was on their random testing suite. In assignment their code again functioned properly and after inspecting their code the functions seemed to handle a larger range of occurrences. While this is the nature of random testing, it did seem like these functions were better at catching outliers that may occur. This assignment was able to get their cover up to the mid 30's and even get specific functions to 100% if the tester was allowed to run several thousand times. Overall the random tests seemed more reliable than the unit tests but both did cover enough of their respective function to validate them as functional.

Nguyalex

To test the reliability of this student's code and the coverage that their tests had I again cloned their distro and ran through their different assignments. The first thing that I noticed was that their unit testing achieved percentages in the mid 80's, I eventually realized that this was individual to the function in question instead of dominion.c itself. Because of this percentage it seemed like, and I later confirmed by looking at their unit tests, that they had done a much better job than I had of ensuring that any outlier in were tested. For their random testing the program again functioned properly and was able to achieve a coverage in the 90's, increasing when I allowed it to run for quite a while. This seemed fairly standard as the random testing put more pressure on the functions and resulted in more of the lines being called. And finally I tested their assignment 3 test, the simulation of users playing a game of dominion. This time their code coverage was in terms of dominion.c, which they achieved a percentage in the low to mid 40's, similar to what I was able to get on my simulation. Their program did determine winners and was able to simulate multiple games successfully. Overall I felt that their random tester was the most reliable of the three, with the unit tests close behind. I was impressed with how much they were able to cover. The one test I would not rely on was the simulated game, though this is of no fault to nguyalex, as stated above I believe that the simulations are too shallow. They test what we the programer expects the user to do within the bounds of what we give them. Nguyalex did a good job in their creation of a simulated game, as I believe I did as well, but neither was detailed enough to result in the even half of code covered.