Test Report

Vasile Grejuc
June 4, 2016

Before taking this class I had very little testing and debugging experience. I wrote some very basic tests in software engineering 1, and some in data structures. In assignment two for this class, I had make four unit tests and four tests for my dominion cards. This was my first time actually writing unit tests. The way I saw testing at the beginning of this course was just writing code to check that the program is actually doing what it's supposed to do. But I was seeing it on a very broad level. The only types of coverage I ever used was line coverage. I did my unit tests on gameOver, initialize, buyCard, and getCost. I also tested steward, outpost, smithy, and village. In my tests, I would pass inputs I needed to make sure that every line of the function was covered. I used a setting in my text editor that would measure the number of lines covered and give me a percentage of coverage. When my line coverage got to 100%, then I assumed everything was tested and complete. I didn't know about all the different types of coverage, such as graph coverage, logic coverage, input space partitioning, and syntax-based coverage.

My unit tests in assignment two covered under 27% of code based on my results from gcov. And they were not catching many bugs, even though there definitely were bugs in the program. When I moved onto assignment three, I knew more about covering all the different paths and nodes. This helped me write my random tests. I wrote random tests for adventurer, steward, and outpost. I wrote my random tests by making sure that there were tests for all of the lines in the function. And when I initialized the game I would randomize the seed, how many players were in, and the choices. I also shuffled the deck and would give the players random hand sizes. I tried to cover as many combinations as possible. I put the random tests in a loop that ran 1000 times and then totaled the number of tests passed and number failed. This gave me a much better idea of how buggy the code was than when I wrote my unit tests. My coverage also increased when using random testers. I was getting around 33%.

For assignment four, I created a random tester that played the entire game by itself. I made a design similar to playdom, but I added my own rules. I used a similar setup to the one in my random tests by randomizing the seed, numbers of players, shuffling the deck. But the decisions the players made were not random. I set it up so that for the first four turns, the game would try to gain as much money as possible and not buy any provinces or duchy's. After four turns the players would aim for gaining as many points as possible. At the end of the game, all of the player's points were added up and the winners were determined. I managed to get 63.08% coverage from my random test game. But sometimes it would not complete the game due to a segmentation fault or a giant loop.

The dominion code I received at the beginning was very broken. I noticed my game would get segmentation faults often, and many of the behaviors were incorrect. I did not actually try to fix much of the code while doing the assignments. I mostly just tested everything and located the bugs. So the code I had was not very reliable. For assignment four I had to compare my code to a working copy. After testing my copy and the working copy side by side, the working copy would get better coverage, and would play to the end of the game. As I mentioned earlier, my copy would not always make it to the end of the game. For part one of the final project, I looked for bugs in rotithoj's repo. His code was very similar to mine in reliability. There were some bugs that I was able to find just by looking at the code and not even running a tester on it. But I ran the testers and I found two bugs in the village card and one in smithy card. My unit tests were able to find the errors very easily, I didn't even need to use my random testers in this part. Testing other people code was something I had never done before this class. And I did notice minor changes in overall coverage. The less buggy code would usually have better coverage because it was able to get through the game without hitting an infinite loop or having a segmentation fault.

Some of the most important things I learned in this class were the different types of testing techniques, and the different ways of measuring coverage. In addition to this, I also found David Agans' rules for debugging to be useful. Even though his rules are simple, they are often not followed. This leads to a lot of wasted time, bugs, and stress. I think understand the system, quit thinking and look, change one thing at a time, keep an audit trail, and if you didn't fix it, it ain't fixed, are the rules that applied the most to me this term. When I first began looking through the code, I didn't understand some of the game rules and card functionalities well enough. This will make it impossible to find most problems. After understanding the game and knowing that there are problems, the thing that I did was start looking into it my code to actually see what the problems were rather than just imagining what I thought they were. Changing one thing at a time is very important when trying to fix code, especially if you don't know exactly what is causing the problem. Keeping an audit trail let me know exactly what I changed and in what order. I would do this sometimes by making a copy of my working code, and placing it in another folder. If I made a bad change and I couldn't remember exactly what I did, I could revert back to my working copy. This was especially useful when I would get a segmentation fault and not know what caused it. The last rule that helped me was that if I didn't fix the problem then the problem was still there. When I was testing my code with the random tester, sometimes the bugs would show up and other times they wouldn't.