Test Report

Max Dimm

Being someone who has never done any kind of automated testing, this class was a pretty interesting experience for me. It was difficult but a great opportunity to learn. In the start I didn't see the value in writing code to test code because I thought it would be inefficient and didn't see how it would really work. I couldn't see it testing all inputs and didn't see how it would be worth wile. However after going through this class and working through the assignments that the testing code can actually be smarter than the code it is meant to test. Also that writing tests code can save you more time then you think, especially when testing other people's code that you might not be 100% familiar with.

Before taking this class, I was not very savvy to debugging techniques in general. My experience was essentially limited to looking over my code and looking for the missing semi-colon or checking the line the error happened on after trying to compile. I had to learn to use functions such as "print" and "step" within gdb, and how to use gcov when coding my tarantula code when looking for "suspicious" lines. Also learning about saving states, copying them and doing tests on those copies was something that blew my mind at first but made more sense as we went along. I also learned how valuable or necessary random seeds were to the dominion code and how we ended up testing it. I learned that making sudo random seeds was super helpful because it made the bugs more reproducible. This was good because otherwise the testing would have taken much longer. This is somewhat a reference to Agan's second rule, make it fail. This made it so that reliably broke so that I could tell if I had actually fixed the issue when I think I have.

One thing I struggled with was the base dominion code that was given to us was difficult to work with. I know this was largely on purpose because it was our job to navigate it and edit it. The code itself made very little sense, it seemed to have poor formatting, and the coding itself was pretty poor. There were unused variables, repeated code, and control statements that were unneeded. There were some nice aspects to the code though, I was able to take many references from playdom.c and other prebuilt tests that the code came with. These codes gave me an idea of where to start and how to go about creating my own tests which was super helpful. I had a background knowing how dominion was played before coming to this class (I was playing the weekend before you announced it) which helped with the navigation and understanding how the code was working. Had I not had this information before it would have been a bit harder. However I also learned that a couple of our homebrew rules would not fly with the implementation of this code which hurt me in an odd way. The hardest part for me was creating a randomly generated program state and running with it. I was able to do it to a degree that satisfied the assignment and create testdominion.c however I will probably need to revisit that on my own to increase my own knowledge of the subject.

This class definitely improved my C coding skills along with an intro into some python and overall coding. I was also pushed in the testing, debugging and overall problem solving skills which was frustrating at the time but rewarding afterwards. You can kind of see this in the work I submitted, my tests in the beginning did not work as intended and crashed in some cases. I also missed class at one point in the term due to my brother/roommate having an unexpected medical issue which made this class a bit harder in the beginning. With time I was able to recover but it did make an impact at the beginning missing over a week of lectures. I watched the videos you linked us on your github account which helped

a bunch. By the end I felt I was submitting a better final product. My minion_tarantula.c code was leaps and bounds ahead of what I was coding in the beginning.

For the testing, I tested Hotant and Rogersza's codes and wrote about them below.

Classmate #1 Hotant

Me and Hotant has pretty similar code. His code from the start I would say is more reliable than mine (partially due to my first tests being pretty poorly written). His testdominion.c code did not take a seed as a command line argument but rather seeded itself randomly and differently every implementation. This is fine, however it made testing and working with his code pretty difficult as the errors were changing along with the seed. So overall I would say that his code is more reliable, but my tests were able to cover more of the dominion.c code. Due to these seeding differences he covered maybe 51% of the code with 10 seeds, but I was able to cover 67% of the code with the same seeds.

Classmate #2 Rogersza

Rogersza broke completely different functions of the dominion.c code then I did so it was interesting for me to test his code. I would say that Rogersza and Hotant had similar code in the sense that they were both seemingly more reliable than mine. However I would say that my first impression of his code was that the random testers were inferior dude to requiring manual seeding, whereas mine seeded themselves automatically. Also he used assert statements in his code which work, but stop the code when an issue is found. My code tallied in a counter the errors and did not stop. However Rogersza's code did not crash when running testdominion.c which innately makes his code more reliable than mine. Mine would get stuck, loop and freeze which is a big issue. His coverage was better than both mine and Hotant getting near 80% coverage to my sub 70% and Hotant's 51%.