Sam Lichlyter

For starters let me say I *really* did not enjoy testing this implementation of dominion.c, and I don't think that was because it was specifically dominion. I think it was because the code behaved in remarkably unexpected ways. Which I guess on one hand made it easy to find bugs, but on the other I wish there weren't a flood of them. I would have much more preferred a code base which only consisted of just a few bugs then go more into detail about how to find small bugs that would come out maybe only in mutation testing or one of the other more advanced forms of testing.

With that being said understand that some of (read: most of) the bugs in this implementation of dominion were rather laborious to fix. So I assume most of the people in the class, including me, didn't bother to fix any more bugs than they had to, which if this was your job that (hopefully) wouldn't be the case, but it's not, so here we are. The two people who's code I tested came out to work just about the same as my implementation of dominion, given the reasons above I think I'm safe to assume they also didn't fix many bugs in their implementation of dominion.

I tested "merdlera" and "foulgerd"s dominion implementations. For a baseline, the coverage I got on my implementation was 52.87%.

When I tested "merdlera"'s dominion code I ran my full game tester against it and I got a code coverage of 52.53% which is comparable to mine meaning we had almost the same implementations. This gave me a good idea of how improved the code had been, or rather how it hadn't been improved. From here I ran my unit and card tests against it to get a more detailed view of their implementation. I discovered the functions I tested against (the smithy, gardens, great hall, outpost cards as well as the initialize game, whose turn, end turn, and buy card functions) all came out the same as when I tested my own code. I compared the results from the unit tests I ran on their dominion to the ones I ran on my implementation, this told me again, our code was very similar. Given the similarity of my dominion implementation and theirs from the unit and card tests I am able to conclude that their code is just as reliable as my code is, at least for the functions and cards I tested for. From there I got a more abstract view by comparing their code coverage and full game tests I wrote, to conclude that even in a more general sense our code is very similar. Which confirms our code has the same reliability. And given the explanation I gave at the beginning, my code is very similar to the original code we were given. Which now should give you a solid idea of how our code stands up and its reliability. I would say in general neither of our code is incredibly reliable per the true rules of dominion, but compared to the original implementation we were given, it behaves nearly identically. This also tells me the bugs which "merdlera" included were not very significant.

I had the same exact thought process with "foulgerd"'s dominion code as I did with "merdlera"'s. When I ran my full game tester against their code I got a code coverage of 56.15% which again is comparable to both my code and "merdlera"'s. I ran my unit and card tests against their code and got the same results I did before. Following my thought process above I concluded from the unit tests our code was similar for the exact functions I was testing against. And for the full game tester I found out in general our code was nearly the same. "foulgerd"'s dominion implementation was just as reliable as "merdlera"'s which was just as reliable as mine, which in turn was just as reliable as the original code base we were given at the beginning of the term.

I also ran my random tests against both "merdlera" and "foulgerd"'s code. Both of them passed my random card tests and "foulgerd"'s passed my random test adventurer, but "merlera"'s didn't. This led me to the bug I found earlier in the first part of the project, their first bug was in adventurer. When I ran my random tester against "merdlera"'s adventurer implementation it actually seg faulted. This confirmed that it was an out of bounds error which is what he was producing in one of the bugs he implemented for the first assignment.

Going back to the point I started with, I did not enjoy this because the code was too broken to start with. Nearly everyone's code I looked at performed the same as mine telling me the same bugs existed in my code as did theirs. This, I feel at liberty to assume, is because of people's apathy to fix this monstrosity of an implementation. I found this true in this project as well as the previous assignments we went through when we had to compared code.

Overall I feel you should have a pretty clear idea of my sentiment toward this code base, but I do believe a learned a few valuable things by testing this code. I learned that some bugs that you think are small and may not affect too much down the line actually causes significant damages. For example "merdlera"'s adventurer bug, they probably didn't expect that in some rare occurrences this simple switch of a minus to a plus could cause the entire application to crash. I also learned that no matter how hard you try you probably won't fix or even find all the bugs, and the more you try the more bugs you introduce to the entire system. Lastly I learned there is no simple, single solution to testing, you more or less need to try all the variants and see what you can find.