Nolan Thompson

CS362

Final Project

When I started in this class I thought I had ways to debug code. Placing random write statements everywhere and hoping I could figure stuff out by which writes worked and which ones didn't. Essentially it only worked for large bugs that would make the program crash if they weren't fixed. I found those if I was lucky. This class has taught me of a multitude of ways to find a multitude of small, medium, and even large bugs. It was entertaining to break already broken code more and challenging your classmates to figure out what bugs you put in. Finding my classmates bugs forced me to learn the material to a much higher extent. Plus we got to play Dominion in class, which was awesome.

The first assignment started the term off simple, recreate five dominion cards as a separate function to be called from main, rather than something that is just always working inside of main. This is where we started hiding bugs. They were simple at first, which made it easy to find them. Two functions were bugged, and the breaking began.

The second assignment asked us to create tests that would showcase the bugs that were in the functions. When I first tried to make the tests, I wasn't very successful since I hadn't ever tried this type of debugging before. However it started to make more sense and quickly became easier. I tested the code coverage of my tests and they weren't super high, but at 25%, it was a start. The longer the term ran, the better I became at creating unit tests, and my code coverage started to climb.

The third assignment was to create a random test generator for two dominion cards. This proved to be similarly difficult to the unit tests. Not knowing how to set up base cases and original values, the first time was rough, but once I figured out how to do it, it became much easier to replicate. These random test generators are good to check random variables that could happen in any game, such as evening the amount of coins, the number of players, or different hand sizes. The variables had to all be random or each card wouldn't always be tested under a different scenario.

Unfortunately random tests often don’t cover special cases. Like drawing from an empty pile or discarding when a player has no cards to discard. This is where unit tests, from the second assignment, come in, to check for the special cases of bugs. There will always be special cases unit tests are needed for.

The fourth assignment was creating a random tester that plays the complete game of dominion with random amounts of players (2-4) and a random set of kingdom cards. The hardest part for me in this assignment was the random set of kingdom cards implementation. However it's clear why it's so important. Random amounts of players and random amounts of cards, many more differing scenarios can be created. This can always potentially result in finding special cases, and then the bugs that happen in those special cases. A random tester for the entire game wasn’t super difficult to make, but making it quick was important, and that’s where the challenge was. A way used to make it more efficient was/is to test the points and score of each player when a round finished. When doing this, its important to also look for hand size and deck size of each pile. While also looking into the coin count. However, this can be improved even further by adding a plethora of differing scenarios where players could choose what cards they could randomly buy. This causes them to have more random choices, including the action they chose when they had a turn. This was one of the hardest assignments for me personally, because there were so many scenarios that I couldn’t figure out how to test. I ended up having to look for help online, which helped my understanding a lot.

For the final project I worked with my classmates Cooper Gale and Dillon Odle. Although I didn’t test all the different iterations of their code, I tested the latest ones using both unit tests and random tests to look through both of their entire dominion game. These unit tests and random tests were used on changed functions to test what other people had changed. While also looking at the code they made and trying to find the bugs hidden inside.

After running the unit tests and documenting the results, I asked to make sure that I had actually found the bugs that they described. Unfortunately I happened to miss one in Coopers code when I looked over it with my texts, but other than that I feel I was pretty successful in catching bugs. This could have been from a multitude of things. A random test not testing one special case, or a unit test failing to cover all the possibilities. However, a promising thing was that when I tested the code, the dominion game itself was pretty reliable. Although the game wasn’t played

COMPLETELY correctly, it wouldn't core-dump or seg-fault, an order was set for turns and it was followed, cards were drawn mostly correct, and most everything worked except for the bugs I found (and didn't find). As a group we chose to inspect each others code via the online moderator/organizer. Unfortunately, since we know each other pretty well, there wasn't much formality that was followed. We mainly just teased each other while we worked about random stuff. Although it worked surprisingly well. It was smooth and a solid amount of bugs were found in each others code.

This class was a very well taught class, bringing together learning with a rather enjoyable card game I had never heard of before. I learned how important debugging is to creating a successful program, and the multitude of proper debugging techniques that can be applied to real code. I would never test for bugs before unless the compiler told me there was an error. If I encountered a bug through playing it, I would fix it, but I never had the ability to such wildly test for bugs. Now I can apply this knowledge to future CS classes, and potentially future jobs.