Jeffrey Wentz
06.08.15
CS 362 Final Paper

Dominion Debugging

Imagine, you've been given a jumble of words that have a vague similarity to C code. Pointers astray, methods as stable as Tacoma Narrows Bridge, and compiler error lists denser than Linux Kernel documentation. There are rumors Linus Torvalds himself won't even attempt this dangerous of a feat. This is not a work of fiction, no. Please join me as I am about to convey to you the very real process I went through with testing/debugging the files known as Dominion.

What I thought was a playfully light Aprils Fools joke quickly turned into a dark reality. I was given a collection of files containing many parts of the machine that is Dominion, most predominantly being dominion.c. The first step in the process was to familiarize myself with the files. Figure out which controlled what features so I could begin to understand how the gears turned. For this process we were using GitHub, a version control website where our changes would be posted for the whole class to see and in some situations, write over our files. Quickly those who didn't know how to use Git properly were spotted. Already being familiar with Git myself though, there was a very small learning curve.

Diving into dominion.c, I first re-factored the cards adventurer, council_room, feast, gardens, and mine to have their own functions. It was important to keep track of the changes I was making so that if any other bugs were to come up in later implementations I would have an idea of where to start. These dependencies where recorded in assorted .txt files.

Following re-factoring, my next task was to write four unit tests and four card tests. This was my first real taste of writing test cases. I quickly learned what a pain it is writing all these manual tests. Sure, it's helpful for very specific parts of the files and for me personally, being the one who is well acquainted with dominion, but it would not be quite as helpful for big picture or for those unacquainted. To my surprise, these test cases taught me quite a bit of bugs in the Dominion code running from card errors to overarching structure of the code. These were dealt with incrementally and as they were encountered they were dealt with. I struggled with understanding gcov and how it was supposed to cover the code. During initial runs with it, I was only covering a small amount of the overall code which I learned was common for certain cards or functions.

Through and through, at this point I was still going strong with hope in my heart that the end would be near. No major issues had come up yet and GitHub was working flawlessly. It seemed those who were using Git and those who didn't were content with SVN. Oh how I was wrong. When attempting to submit a portion of assignment 2 a user had reverted thousands of lines of code when he mistakenly tried to clean files. Luckily, GitHub proved it's use and when this was discovered it was able to be reverted. I can only hope they learned their lesson.

Assignment 3 rolled by and it was time to write my first random test generator. It seemed daunting at first, but once I got a better understanding of how it worked it proved to be much more useful than writing specific test cases. The random test cases allowed me to cycle through most parts of the code quickly and run hundreds of random tests back to back, making bugs a lot easier to detect and fix. I did forget to add Makefile targets regrettably. I also learned that when editing the Makefile on my machine that my tabs are actually detected as a set number of spaces. Makefile's being the picky beasts that they are, did not respond to this well. To repair this, I had to copy the existing tabs and paste them wherever I needed a new target.

Assignment 4 proved to be by far the most difficult task assigned to us. I wrote a complete test for dominion. Assigning 2-4 bots to play against each other. The process ended up being easier than I initially thought as most of the functionality already existed in the contained files. All I needed to do

was determine the correct order of calls on those functions to create an entire game. This process also dug up quite a few bugs as the random testing always does. Specifically with my feast card spitting out an error where it thought the card was too expensive for the player to buy. I also had to run my tests against another classmate to compare the difference between the two. This was pretty insightful to see the difference between the two implementations.

Looking back at the process, I feel like I have learned a lot about testing. I think giving us this poor excuse for a program called Dominion as our testing subject was a fantastic idea. It forced me to look through unfamiliar code and debug something not written by me which was a first for me. It taught me the importance of random testing and why writing manual test for each card or function, while useful in it's own respect, is not going to always be a viable option with so much functionality.

Dominion used to be one of my favorite games to play. I loved all the expansions and diversity the game has to offer. It was unlike any game I had ever played before and I will remember those times fondly. As for now, I don't see myself partaking in anything related to Dominion for a very long time. That is a beast that I would like to store away in a cupboard and forget about for the next decade or two. Perhaps one day I will dust it off again and form my own implementation of the game. Until then, goodbye, Dominion. You won't be missed.