Alex Nguyen

CS362

Alex Groce

6 June 2016

Final Test Report

Introduction

Testing dominion.c was a bit daunting at first. Initially, I had never played the game and have never seen the code. Thankfully, Groce allowed us to take the code base by piecemeal via refactoring, unit tests, card tests then finally a full game test via assignments. I wasn't familiar with Dominion so I did some research through the Android app and through YouTube. Through my research, it looks like each card could affect the state of the player, the state of other players via card counts and discards and overall game state via points. Also there is some synergy between all the cards but those were a bit harder to understand. After skimming the code, it looks the concept of the game is fairly modularized with the data structure being mostly integers to represent states, cards, and more. However, the code was very large and was mostly in one file. I believe the code could've been refactored deeply with object-oriented programming.

Coverage

Initially, my dominion tester only reached roughly 45% of the domino code. I implemented a preferred card choice that limited the AI's possible card choices but allowed them to further progress in the game. This action somewhat improved my coverage by a couple of percent. From there, I implemented more aggressive and larger looping to really randomize the inputs. This improved the coverage to roughly 55% but the benefits from more loops stopped there.

Reliability

I feel my code is only reliable if I tested it with unit tests and card tests. The scope of my random tester, testdominion.c, is so randomized that I feel that it could cause gaps in testing my code. I feel that some of the more expensive cards do not execute as often as they should which could bring up issues in the future. This brings a bit of uncertainty to myself that the only way to truly test something is through unit and feature testing.

Testing Other's code

I have decided to test Ramcharan's and Dennis' code.

Upon testing sudasar's (Ramcharan Sudarsanam) code, a bug that I found that his Council Room card was acting strange upon a unit test of mine. The card was gaining four cards appropriately but it was not increasing the buy counter by one. This was one card that Ramcharan did not test in his testing but we found it

with my unit test. Upon further investing his code, I found his tests to be very thorough. His coverage was very high at 94%, a lot higher than my 55%.

The second bug I found from Sudasar's code was that the card, Remodel, was not discarding itself into the discard pile. The card effect was causing it to generate a card into the hand but not discarding Remodel thus technically gaining the player a card for no cost. I found this out by manual testing cards chosen at random. I made sure their card operation matches their execution. The process was slow but it showed that bugs like this could exist without crashing the application.

All in all, I would say that Ram's code is fine and works without major segfaults or crashes. It feels functional and runs for the majority of the time I used it. His code coverage is very strong but there are some logic flaws that could be addressed in the future.

Leed's (Dennis Lee) code was somewhat reliable. During my random tests on his dominion.c code, it would segfault and crash randomly. Upon further investigation via divide and conquer, there was an execution branch that would happen when the hand would play Remodel with an empty hand. This would trigger to lookup the cost of a null pointer to the expected discarded card. Although very specific it completed most of my testing and performed expectantly. All of Leed's and mine unit and card tests passed as expected. I believe that his coverage was thorough and I probably found an unexpected execution branch by luck. He probably would've found the same bug in the near future.

Leed's code coverage was about average at roughly 58%. However, it did look like Leed's coverage was augmented by some custom tests, which I thought was very admirable. He created a customHand, compare and a few other validator functions to help the code base perform well. Although this increased his code base it seems like it was used to check items for existence before utilizing them.

Thoughts and Conclusion

In conclusion, the process of testing was a lot more difficult than I thought it would be. Coming up with inspiration and ideas just for tests was probably the most difficult part. Once I had an idea and a goal, writing the test was quite quick and painless. It would nice to have a testing flowchart guideline introduced to me. Part of me did not want to introduce redundant tests or overlapping tests.

There were points where I were stuck with very simple tests and wanted to test something deeper. There are many articles of resolving bugs but articles to search for bugs are a bit harder to find. After looking at other student's dominion tests, I copied some of them for my own but I also tried to understand their reasoning/thinking.

Refactoring code in a large unknown code base was a bit scary since I wasn't completely sure that I could've broken something in CardEffects or not. At first, using an IDE seemed like the most logical tool to achieve a 'proper' refactor but it was not too bad. After completing that assignment and the unit tests one, I can see why test-drive-development is so common and helpful in the long run. Having the unit tests early in our debugging allowed us to check even for only 5 cards, that they work. If we had a complete test suite for the entire dominion card I think that would really help our game tester at the end of the term.