Dillon Odle

CS 362

6/9/2015

Dominion Test Report

In this document, I will describe my experience with testing of the dominion game with both my dominion.c file and the dominion.c file used by my peers.

In assignment 1 I familiarized myself with the dominion rules (as I had not played the game before this testing occurred), then refactored 5 cards (village, feast, remodel, smithy, and great hall) into their own functions from the cardEffect function that these cards were previously found in. While there wasn't really any testing that happened at this point in the testing experience, giving myself a base knowledge of how the game works helped me later design the tests that I used for the rest of the term. Also, as this class is about the process of testing itself rather than finding bugs in our dominion.c file, introducing a bug into both the smithy and village cards gave me a way to make sure my tests were working as intended in later assignments.

In assignment 2 I performed the first actual testing of the term, creating unit tests for the numHandCards, whoseTurn, isGameOver, and drawCard functions. In addition to these unit tests for non-card implementation I also created unit tests for the minion, great hall, smithy, and village cards. Through these tests, I found the bugs I had programmed into my smithy and village cards (+4 cards instead of 3 for smithy and +3 actions instead of +2 actions for the village card) in the previous step. Additionally, I found a bug I didn't program into my village card implementation which was that playing the card failed to give the user an additional card in hand. My unit tests all failed to show any bugs for the functions that I chose for this assignment, which could indicate that these functions worked correctly with the parameters I set for my unit tests or it could indicate that my unit tests were poorly designed and would need better development for future work on debugging the dominion implementation. I measured the coverage of the dominion.c file of all my card and non-card unit tests and had a coverage of 21%. While this was not a high amount of coverage I assumed it was reasonable given that I only tested 8 functions in the entire dominion.c file which contained over 30 such functions. This coverage was also consistent with coverage that I obtained using gcov in other assignments throughout the course.

In assignment 3 I wrote random test generators for adventurer and village cards. I appended the results of these tests to two files in my repository. I designed these tests to have a random number of players from 1-4, to have a random number of cards in deck for each player on each turn, to have a random number of discarded cards for each player on each turn, and to have a random handcount for each player on each turn. All of this randomness was done in an attempt to increase the chances of encountering a bug while running these tests on the two functions in question. The randomness element allowed me to see that my village card failed to give +1 card on every test and failed to give +2 actions on every test which, while I could see that on my previously built card test for this function, this random test allowed me to see that this failing condition occurs under a wide variety of conditions. My random adventurer tests found 3 failed tests in a total of 50 tests performed, but didn't give me any direction as to what these tests failed for. If I were to try to localize these bugs, I would print more

aspects of the game state during these tests to see where exactly the bug occurs, but knowing that there are bugs in the card was useful itself too. Both tests had 24% coverage, which was consistent with other tests.

In assignment 4 I built a random tester that could play a full game of dominion. This tester had 2, 3, or 4 players (randomly) with each player implementing a different strategy. Printing out the gamestate information using this tester was a beneficial testing feature since it allowed me to discover possible sources of bugs better than I would be able to if I were to just play a game of dominion and interpret the results after these tests were complete. Additionally I used diff to measure the difference between my tests running on my code and tests running on a classmate's code (Cooper Gale, galec in the repository). When I attempted to print the difference between the two files, nothing printed out which indicated to me that our implementations of all tested functions in the game were the same. While this is not incredibly useful for testing purposes finding differences between these two implementations would have greatly helped to find bugs in either implementation and thus diff was a useful tool to use in my testing. Both of my random testers had line coverage of around 24%, which was consistent with other measures of coverage found in testing so far.

In the final project, I started by performing my unit and random tests to test the code written my classmates Cooper Gale (galec) and Nolan Thompson (thomnol). Through this, I localized two bugs in Cooper's code and one bug in Nolan's code. I thought it was interesting that I could take these tests that I designed and use them to be useful with another person's code. After I wrote bug reports for each of these bugs in their code, I used a debugger to isolate the +3 cards in hand bug that I had in the smithy function. By isolating this bug using this method, I've isolated this same bug using unit tests, random tests, and debugging which was great because it shows that there are multiple different approaches that can lead to isolation of the same bugs. After I identified this bug, I performed a code review on Cooper Gale's dominion implementation.

This term has taught me a wide variety of testing techniques that I can use in subsequent classes and work situations. In fact, in my other classes I've already been using some of the techniques to find bugs in my code. Overall I found the information presented to me to be very useful and I am excited to continue to use this information in the future.