(Ellen) Reilly Collins
colliell
CS 362
Spring ' 1 5

Test Report

Summary

Overall, I had a pretty good experience testing Dominion throughout this course. It was really interesting and beneficial to see how much I improved over time: as I competed this final project and took a look back at code from my unit/random tests (assignments 2 and 3), I was able to see that some of them weren't really that useful. As I result, I actually came up with ways to make them better and ended up with some fairly good test cases.

I also experienced firsthand how testing shows you what bugs are in your program, specifically testing "harder not smarter." For a few of my dominion testers, I simply had to increase the number of calls to my tester in order to find errors. Furthermore, by following print statements regarding the game state and any errors, I could easily see how rare or common an error was; this helped me localize the fault.

In assignment 4, I spent a lot of time working on the complete Dominion tester, and I ended up with something that I am quite proud of. What I would like to do is improve my tester even more by implementing unit tests for each card, in order to check that the functionality/logic of the game is sound. I think this type of integration testing would really show just how many bugs, both logical and program-crashing, are in dominion.c.

Finally, I wish I had this knowledge about testing in previous programming classes – unit tests would definitely have saved me the frustration of trying to figure out why something wasn't working, and random testing would have saved me the time I spent trying tons of different inputs (as you never know exactly what graders are going to test for!). I look forward to using these skills in future courses, as well as continually improving my software testing knowledge and abilities.

Code Coverage

The coverage percentage for my randomtestadventurer and randomtestcard were about 27 and 26, respectively. Furthermore, all lines of the Adventurer and Great Hall cards were executed for each test case. With this in mind and considering that they are both just testing the functionality of one card, I would say this is very adequate coverage.

My testdominion program's coverage was at about 88 percent the dominion.c gcov information to determine where coverage was missing.

1. During the buy phase, a few of the error-checking lines are never executed: if a player has no more buys, the supply of the card he or she is trying to buy is empty, or he or she does not have enough money for the card chosen to buy, my tester just picks a different card. It would be useful to have coverage of these invalid choices, as I would have been able to see that dominion.c doesn't really have me attempting to "break" my Dominion game in this way, I would be able to find and try to fix any potential bugs that could arise. For example, it would be best to have the gameplay recover from these errors in the player's choice by implementing a while loop so that he or she can continue the turn.

2. I didn't test some of the helper functions (like getting the number of cards in someone's hand) as I just implemented their functionality on my own, but it would be good to make sure they work properly. It would also probably be easy to test these functions and improve coverage in general.
3. I found that part of my Mine and Remodel cards never executed due to some of the logic in my tester: the choices given to the playCard function didn't work out right cost-wise and always returned an error during the action phase. My testdominion program was producing error messages for these cards, and now that I am armed with the coverage information I have essentially found the bugs. This also shows me a weakness of the dominion.c code, as it should be able to recover from these errors and possibly allow the user to choose again, so overall this type of coverage information is very helpful.
4. During the action phase, my Baron, Great Hall, and Steward card testers never execute with the second choice chosen. I would need to randomly pick between choice 1 and 2 while testing in order to check the cards' full functionality. This is definitely good to know, because I had no idea these cards weren't being tested entirely.
5. For some reason, my Treasure Map and Cutpurse are never played. This is also very valuable information given from coverage, as I know my tester should be able to check every card's action phase at some point action. Again, I would not have known this information without looking at the coverage of dominion.c.

Status

My new and improved unit and random tests produce some error messages, so I know that those are working. Although I think my testdominion.c code is pretty thorough, I could certainly make it stronger by integrating unit tests for each card's action and buy phase. After that, I would like to break the program with all sorts of random input to see how well the dominion.c code handles it.

Overall, I'd say coverage running testdominion is quite decent. Looking above at the information above, I am fairly confident I could continue to work on my testdominion.c code until the uncovered cases were accounted for. My tester certainly gave me warning about what wasn't covered, the Mine and Remodel cards always produced error messages; I think this is a good representation of how coverage is related to testing. The Ambassador card also produces an error, and there are instances of players having cards with a -1 value. This demonstrates that my tester is (at least somewhat) doing its job and showing me what doesn't work.

Reliability

hansonm

Using the information from my unit test, I found a couple bugs in the Minion card's functionality as documented in part 1. When I ran my other unit and random testers from assignments 2 and 3, I did not find any errors. Running my testdominion program from assignment 4 on his code took a while, as most of the time the game gets stuck in an infinite loop and the program doesn't finish executing. I found several instances of a player's hand having -1. There were also error messages for the Remodel, Ambassador, and Mine cards. Coverage of his dominion.c with this tester was at approximately 85%. Looking at differences between our dominion.c and the above information, I would think that his code is somewhat reliable as the program is able to compile and run correctly, and the game is able to be played.

However, I wouldn't say it's ready to be released to the public any time soon due to the error messages and the fact that it doesn't run all the way through every time. Not only would these bugs need to be addressed, but further testing would need to be done to ensure the program is robust.

boringk

Using the information from my unit test, I found a bug in the Baron card's functionality as documented in part 1. Again, I did not discover any errors when running my other unit and random testers. Running my testdominion program yielded coverage of dominion.c of around 84%. Though it occasionally had to be executed a few times to avoid an infinite loop, I did not have to do so as many times as I did with hansonm's dominion.c. An error message was produced in the Ambassador and Embargo cards' action phases, and I found several instances of a player's hand having a value of -1 in this implementation as well. Keeping all of this in mind, I would say this code is somewhat reliable as there is gameplay functionality, and more so than the code above; however, it also would need to undergo further testing to be ready for a public release. The error messages would need to be resolved by finding the bugs causing them, and of course the program would need to be able to run every time.