Michael Oatman

CS 362

Test Report

Overall, the experience of testing the dominion code was very unpleasant because the code is very poorly written and overly complicated, making it very difficult to understand and fix when problems arise. Multiple dominion.c files that have been tested and fixed separately were checked for coverage to get an idea of the reliability of the code and those trials are discussed in this document along with the coverage for my dominion.c file. Testing for coverage is a good way of checking the overall reliability of a program because it gives a general idea of the moving parts involved in the program and how the different conditional statements in the program can affect the overall outcome. If a program has more coverage it is more likely to perform better than a program with less coverage, due to more moving parts and conditional statements in the latter program. In this document, the files being tested for coverage are dominion.c (which belongs to me, oatmanm), dominion1.c (which belongs to hansemik), and dominion2.c (which belongs to amidong). The coverage for each file is stored in testdominion.out, testdominion1.out, and testdominion2.out, respectively. The game results for each game are stored in gameResults.out, gameResults1.out, and gameResults2.out, respectively.

For the first coverage test in dominion.c, we can see a total coverage of 68.09% out of 586 lines of code. When looking through the gcov file we can see a couple reasons why the coverage is as low as it is. A few functions such as getWinners(), adventurer card, mine card, and remodel card, all were not run at all, along with the chunks of code that were card implementations inside of the cardEffect() function. The cards inside card effect that were not run include great hall, minion, tribute, and outpost. These cards that were not run all can account for a very significant chunk of the program as a whole, and if they were not run at all then the coverage is understandably going to drop. When examining the code more closely, it appears that the program is very poorly written and disorganized, making it hard to grasp what is going on. Multiple bugs are found in the file as well. A big bug that was found during the testing process of this implementation of dominion was a bug where a player seemed to disappear from the game as the game progressed. When running testdominion, which played full games of dominion and printed the players' moves, I noticed that the game would finish with less players than it began with. For example, a 2 player game would be initialized, but by the end of the game only 1 player existed in the game. When looking through the moves in the game, there didn't seem to be a pattern of when a player would be dropped from the game, and sometimes it wouldn't happen at all. This makes the code very unreliable, and very difficult to debug because it is a problem with the game state and contents held by the game state. This means that virtually anything in the program can account for this bug.

For the second coverage test in dominion1.c, the total coverage of the test is seen to be 55.83% of 575 lines of code. Functions that did not run throughout the test include getWinners(), mine card, tribute card, baron card, and cards inside the cardEffect() function. The cards that were not executed in the cardEffect() function include adventurer, council room, smithy, great

hall, minion, steward, cut purse, outpost, and sea hag. Again, these functions and cards represent a large chunk of the dominion1.c file, which drags the coverage down. In this program, many similar bugs were found between mine and hansemik's dominion files. This includes the discard bug. Not a lot of bugs were found in this program because it seemed as if many of the problems that were run into by other programs were fixed or the test was more compatible with the code because it ran smoother. This is an indication that coverage might not be the best way to test a program as a whole because this file had the least amount of coverage but consistently operated more reliably than the other two implementations of dominion.

For the final coverage test in dominion2.c, the total coverage of the test can be observed to be 55.48% out of 575 lines of code. Functions that did not run in the test include getWinners(), great hall, and multiple cards throughout the cardEffect() function. These cards include adventurer, mine, village, baron, minion, steward, tribute, cut purse, salvager, and treasure map. Many bugs were found in this program that were similar to the previous two implementations of dominion. The first bug major bug noticed was problems involving the discard function that is responsible for discarding a card from the player's hand and placing it in the discard file. This problem is seen when playing cards and discarding them after playing them, because a few of them, such as the council room card, do not end up in the discard pile right away after the card is played, which caused many errors in the test I tried to run on the code. The reason why the cards do not end up in the discard pile immediately is because they go to the played card pile first, which is a variable held inside the game state. Eventually the cards make it from the game state to the discard pile but not until after going through the played card medium. I disagree with this implementation because it makes the code much more confusing and more difficult to test. Instead, the discard function should be written to move cards directly from the players hand to the discard pile, and the trash flag can still be there to remove the card from the game completely. I do not see the need for the middle step of the played card pile, and all it does is increase the complexity of the program and adds more moving parts, which makes it more likely for a bug to occur. It is also strange, that although the other two programs have the same discard function, they do not appear to have this bug, which points away from the discard function as the problem, but something else involving the discard function.

In conclusion, coverage is a good place to start when debugging code, and also a good tool along the way to look at when debugging, but doesn't say everything about a program. As a whole, the dominion.c file was very difficult to debug because it is very poorly organized and written, so it very difficult to understand. Because it is very difficult to understand the program, it makes it even more difficult to debug and fix it.