Luke Morrison
931-802-349
CS 362
Winter 2016

TEST REPORT: dominion.c

Introduction

This test report will go over the analysis of the Dominion code of Mohan Alarifi (alarifim) and Junshi Jia (jiajun). Both of their dominion.c source file have been added to my own repository under the name dominion_alarifim.c and dominion_jiajun.c. We put both of these files through the testers we have have written for class assignments.

Classmate #1: Mohan Alarifi (alarifim)

I renamed Mohan's Dominion source code to dominion.c and ran it through my own random tester. The first thing I noticed that it immediately got caught in an infinite loop. This was remedied by commenting out the while loop and the corresponding closing bracket in the feast card in cardEffect(). This is the only change I have made to the source code.

Coverage

The coverage for Mohan's Dominion source code was quite extensive according to results from GCOV after 100 games:

```
File 'dominion.c'
Lines executed:92.94% of 581
dominion.c:creating 'dominion.c.gcov'
```

This information was aquired by running Mohan's Dominion code through the random tester written for assignment #4. It can be found in my dominion directory under the name testdominion (Type “make testdominion” to compile).

Code Status and Reliability

As mentioned earlier, for my random test to be able to run successfully on this Dominion code, I had to go in and manually comment out an infinite loop to get results. This alone should be an indicator that the Dominion source code needs a lot more testing.

Furthermore, I have done an in depth testing of this code when doing my inspection (Part 3.c) and have gathered a good insight of how well this code performs. Just by looking at the adventurer card, one or two bugs could still be found and remedied even though it already has been looked over in great detail. Now if we included all the other cards, and all the other miscellaneous functions, it is no surprise that this Dominion code would contain a large amount of faults.

Some other bugs that my tester has picked up include players running out of all cards (deck, hand and discard), all sorts of failures when the deck is empty and a card is played, failures to discard from a free hand, cards discarding themselves before their effect is over and more. Fortunately, I did not run into any segfaults or any other disastrous rutimes errors such as bus errors. It seems the majority of the bugs is logical, while syntactical and memory management is performing correctly.

Still, I do not believe that this code would be in acceptable shape after it being tested that much. More testing is required to significantly improve reliability, and should be done was this Dominion source code to submitted for any real life application.

Classmate #2: Junshi Jia (jiajun)

I renamed Mohan's Dominion source code to dominion.c and ran it through my own random tester. Again, I noticed that it immediately got caught in an infinite loop. I applied the same fix as before by commenting out the while loop and the corresponding closing bracket in the feast card in cardEffect(). This is the only change I have made to the source code, and was able to run it for 100 games with no problems at all after that.

Coverage

As with Mohan's Dominion source code, Junshi's source code showed good coverage after testing with 100 games:

```
File 'dominion.c'
Lines executed:91.36% of 579
dominion.c:creating 'dominion.c.gcov'
```

This information was aquired by running Mohan's Dominion code through the random tester written for assignment #4. It can be found in my dominion directory under the name testdominion (Type "make testdominion" to compile).

Code Status and Reliability

Junshi's code showed many faults that were also present in Mohan's code, although some of the more common ones seemed to be fixed. Unable to tell how reliable the code was through inspection (since all the more common faults had been fixed), I ran this Dominion source file through some of the other random testers I had written. This tested the Steward card and the Adventurer card.

Again, with the adventurer card, there were some bugs. The biggest one was that the hand count isn't correct after the card is used. This is probably due to the same bug where the card does not ever discard itself after being used. Otherwise, unlike in Mohan's code, the process of going through the deck to find treasure cards seemed to work correctly and yield normal results.

After testing the Steward function as well, I am a little less certain of the reliability of the code. There were a lot of errors that I was unable to explain, such as the first 20 cards in the discard pile being incorrect after playing the card (although the discard pile was around 300 or 400 cards in size). Maybe there are some issues with the discard pile becoming too large as the player buys more cards. Let's say a player starts with an already large deck, close to the 500 card limit, and then keeps buying more cards, there could easily be 300 cards in the deck and 300 in the discard. This would create some undefined errors when the shuffle() function is called for example.

Overall, this code seemed more reliable at first, but after some more in-depth testing, it may not be that much more reliable than the previous one. There are just too many cards to test and a lot of bugs found whenever a single card is tested. We cannot assume that the other cards will be bug-free, so this code should really undergo more testing.

Conclusion

To reiterate, both of the Dominion codes analyzed in this test report need to be tested more thoroughly. The reliability of these source files have barely increased since they have been supplied to us, and they were littered with bugs from the start. The truth is that many of these bugs even seem to have been purposely put in place because they are in the most uncommon of places. Moreover, the implementation barely feels complete, and uses some unorthodox twists and turns to make some functions work. With only a couple of random testers, it is impossible to test this code enough to call reliable by any means. At the end of the day, it remains still full of faults and would require some full time testers to change that fact.