Testreport - Isaac Stallcup

Testing dominion was no picnic. The code is riddled with omissions, errors and functions that do not perform their intended function (here's looking at you, updateCoins). The number of errors, and terrible code design (hundred line switch statement) of dominion.c added further challenges. However, through unit testing, random testing, game generation and bug localization I was able to find and fix some of the bugs in dominion.c.

To begin with I wrote unit and card tests for dominion.c. The unit tests met with high success, testing individual functions initializeGame, buyCard, supplyCount, and whoseTurn. The function initializeGame was selected because it is extremely important, being called in virtually every test and implementation in dominion.c. Errors in initializeGame would then need to be fixed for anything else to work. Luckily, the function did not fail.

buyCard was tested because it is also an integral part of playing the game, though at the time gainCard would have been a wiser choice. buyCard is rarely used, while gainCard is a much more direct way of getting cards into a player's deck/hand. However, buyCard had no issues with my unit tests.

supplyCount was the next function to test, as it involved a game end condition, and all was well with it. Finally, whoseTurn was tested to ensure that the game would be able to tell whose turn it was in order to not make illegal game moves.

These tests are available as unittest1.c, unittest2.c, unittest3.c, and unittest4.c.

Four cards were also tested; these were village, smithy, minion and cutpurse. The cards were tested by asserting that, after they had been played, various factors that were the result of these cards had indeed occurred.

These card tests were met with much less success than the unit tests, partially because they tested more things and partially because the card's effects travelled into buggy code. For example, minion and cutpurse both failed to correctly give the player more coins. At this point, I was not familiar enough with the dominion.c code to fix these issues, but my card tests made me aware of them. Furthermore, cutpurse failed to force opponents to discard coppers, though again I was too new to dominion.c to fix these issues. Code coverage at this stage was about 35% of lines for all unit and card tests; not terrible but not great either. This was mainly because vast swaths of dominion.c were never run, as they were cards not tested or examined.

Next I wrote random testers to test three cards: smithy, baron and adventurer. These random testers all had the same general structure. For a random number of testing instances, each instance generated a new game state, added a random number of cards to player 0's deck, and then shuffled their hand into their deck and drew a new hand. From that situation, the card to be tested was added then played from the hand of player 0.

After the card was played, an oracle function checks to see whether certain assert statements (again based on the expected results from each card) were true. These assert statements caught several bugs; however due to the chaotic nature of random testing I had little idea what bugs the random tester was catching. Code coverage at this stage for random testers was again around 34% of lines; slightly worse than the unit testing, and still not great. One encouraging side note was that I achieved 100% line and branch coverage for my random test of smithy.

These tests are available as randomtestcard1.c, randomtestcard2.c, and randomtestadventurer.c.

The next step was comprehensive random testing. I chose Thai's repository, located at https://github.com/cs362sp16/cs362sp16_thaia.git. With the seed 42, the testing failed; in particular, the diff of the two files (found as diff.out) is over 5000 lines long. In testdominion.c, I printed the hands and discards of each of the players each turn they had, contributing to the long diff. Furthermore, in the course of this assignment I made improvements to my dominion.c file. The more changes we made, the less likely it was that our implementations of dominion.c would behave similarly when put through the comprehensive random testing.

The comprehensive random tester played a random game of dominion by first determining a random number of players and a random set of kingdom cards to work with, then by taking a random number of turns with those players until the game ended. As a result, errors early on in the game propagated, further increasing the differences between athai's code and mine.

Running 20 differently-seeded versions of this test gave a code coverage of 79.38%, much much higher than any other type of testing. However the results it gave are substantially less helpful (especially in their raw form) than the other types of testing.

This test is available as testdominion.c.

Finally, I used a series of bash scripts and a C program to implement the tarantula algorithm of bug localization, in an effort to track down certain bugs. By this time I had already fixed several bugs out of frustration; most notably a bug in updateCoins that was the source of several bugs found in the card tests. Tarantula parses gcov files generated by running the random tests, which I selected to give a good sample size for tarantula but were not as massively difficult to interpret as the differential testing/complete game testing files. Tarantula found a bug in the gainCard section of baron (where you choose to gain an estate) with 68% suspiciousness (random seed 3). It also found a bug in the adventurer card functionality of putting treasure cards in your hand with 70% suspiciousness (random seed 3 again).

Finally, in order to satisfy the requirements of the final project I again tried to find and document 3 bugs in athai's code; in doing so, I ran my card and random tests as well as running the tarantula algorithm, and found several bugs similar in nature to ones I had encountered before.

I firmly believe that the only thing that could improve the reliability of dominion.c would be a complete rewrite, drawing on the existing material for inspiration and guidance but adopting none of the poor decisions its original author made. This is because some of the key components of dominion.c are fundamentally flawed; cardEffect and gainCard are two examples. Both functions are called constantly in much of the process of playing the game, are buggy and contribute to further bugs by process of propagation. I also examiend the repository of github user kannabanana, ONID kannas. Her code was about the same as mine, having similar minor errors.

One factor contributing to my feelings on dominion.c was that most every line of code my classmates or I added detracted from the problem; the essential problem was the inner workings of dominion.c.

Also, the formatting was not unlike the formatting of code written by a 35-year-old blind dog.