Tanner Cecchetti
CS 362
Spring 2016
June 5th, 2016

Testing Dominion

Overview

I chose two fellow students at random that have public "cs362sp16_*" repositories on GitHub. I grabbed each of their `dominion.c` files and placed them into my own `dominion` directory. Using their implementation, I executed all tests that I had previously created through the course.

My suite consists of the following: unit tests for game functions and card effects, random tests for card effects, and a game generator. A summary of each section of testing is included below, as well as a final conclusion and summary of all results.

Unit testing (game functions)

My unit tests cover the following functions: `whoseTurn`, `handCard`, `fullDeckCount`, and `updateCoins`. In total, they check 13 different assertions relating to the functionality of each method, and, additionally, print out the code coverage for each function under test.

leed:
Code passed most unit tests and all had 100% coverage. The only assertions that failed are related to bounds checking. For example, the function `handCard` will happily reach out of bounds if given an invalid player or card index. Similarly, `whoseTurn` doesn't validate if the current player is within the bounds of 0 and `numPlayers`. These are perhaps "nitpicks," but it could help prevent errors in the future if these functions had the ability to catch runtime errors.

lantowp:
Code passed most unit tests and all had 100% coverage. In fact, the results of running against their `dominion.c` are exactly equal to those of leed's. It would be nice to have some runtime bounds checking on these functions, but it is certainly not necessary, and all core functionality appears to be working properly.

Unit testing (cards)

My unit tests also cover the following cards: Cutpurse, Sea Hag, and Smithy. Each unit tests plays the card and checks the gamestate to validate that it has been properly modified, following from what the card is intended to do. In total, they validate a total of 14 assertions.

leed:
Code passed most assertions, but failed a few that are critical to the operation of the card. Cutpurse failed to increment the player's coin count by 2, and Sea Hag seems to be doing some strange things with the gamestate.

lantowp:
Exact same result as was found when running against leed's dominion.c (see above).

Random testing

My random tests cover the following three cards: Cutpurse, Council Room, and Adventurer. The basic idea of all these tests is to set the gamestate into a random state (meaning random number of players, random decks, random Kingdom cards, random hands, etc.) and then playing the card under test. Each test then attempts to validate that the proper effect has been applied to the gamestate. Each test is repeated approximately 100,000 times to test the card's effect under various different conditions, since a (mostly) unique gamestate will be generated upon each execution.

leed:
Cutpurse has a couple of issues. First, coins is not incremented by 2. This was previously detected by the unit test, but appears here once more. Additionally, when looking at the line coverage, we can see that some statements within the function are never executed. Specifically, this clause:

```
  4910449: 1155:                if (j == state->handCount[i])
        -: 1156:                  {
    #####: 1157:                    for (k = 0; k < state->handCount[i]; k++)
        -: 1158:                     {
        -: 1159:                       if (DEBUG)
        -: 1160:                         printf("Player %d reveals card number %d\n", i,
state->hand[i][k]);
    #####: 1161:                     }
    #####: 1162:                   break;
        -: 1163:                  }
```

Council Room has 100% coverage but reports an assertion error. It appears that in all causes it fails to increment numBuys as it should.

Random testing Adventurer causes a segmentation fault.

lantowp:
Cutpurse has the exact same issue as with leed's dominion.c (see above).

Council Room completely passes this portion of my testing. All assertions are satisfied and it has 100% line coverage.

Adventurer completely passes this portion of my testing. All assertions are satisfied and it has 100% line coverage.

Game generator

Similar to the random tester above, the game generator creates a random game state (random players, kingdom cards, etc.). However, instead of creating random decks, hands, etc. it attempts to play through a game by making valid moves/choices. This results in a much more "holistic" test, since every card will be executed if the test is ran enough times.

leed:
Running my game generator for 20 iterations with a seed of 42 resulted in an infinite loop. This is a very critical bug because it is "game breaking" for a user. This prevented code coverage statistics from being generated (had to kill process). If I bumped the iterations to 5, the tester executed properly, and resulted in 73.44% code coverage.

lantowp:
Same as above. Under the same conditions, an infinite loop was generated. Unlike above, however, 5 iterations still resulted in an infinite loop. If I took it all the way down to 1 iteration, the tester executed properly, and resulted in 52.15% coverage.

Conclusions

Below is a summary of my findings for each of their dominion.c implementations. This combines the results of all individual sections above.

leed:
At a high level, I believe leed's dominion.c is somewhat unreliable. It wasn't very hard to cause a segmentation fault, which I believe to be an extremely critical bug. On top of that, an infinite loop was found when attempting to play through an entire game. Lastly, random testing exposed a few issues with card effects, some of which were more serious than others (such as Sea Hag). It didn't seem to me like a whole lot had been fixed versus the original version of dominion.c.

lantowp:

At a high level, I believe lantowp's dominion.c to be somewhat reliable. The biggest issue I found with their version of dominion.c is an infinite loop when playing a full game. In addition, I found a few minor errors that were caught by the card unit tests. On the plus side, segmentation faults had been fixed. I really only found one "game breaker" (the infinite loop), though some of the card effects sometimes had pretty dire effects as well (such as Sea Hag).