William Leslie

CS 362 – Spring 2012

5/17/12

Test Report 2

For test report 2, I fixed many of the problems in test report 1. testdom now works and outputs to test.out. I also fixed my buyCard test and added Prof. Groce’s drawCard test. Both these tests use random testing. This new implementation of the buyCard test is similar to Prof. Groce’s drawCard test with the difference that it will use asserts rather than comparing the memory of the actual game state with the ideal. In the test file, the main function uses a random number generator to make a random card to test, choose a random player, add a random amount of coins, add a random amount of buys, and set the supply count of the card in question:

card = floor(Random() \* (treasure\_map+1));

p = floor(Random() \* 4);

G.whoseTurn = p;

G.coins = floor(Random() \* 10);

G.numBuys = floor(Random() \* 5);

G.discardCount[p] = floor(Random() \* 5);

G.supplyCount[card] = floor(Random() \* 10);

To show what is happening within the test, the statement “attempting to buy to buy card (card#) with (coins#) coins” before calling the actual test function. The function checkBuyCard takes the parameters of the player number, the card number, and the game state. On this game state buyCard is run and the following asserts are checked:

assert (r == 0);

assert(G.numBuys == buys-1);

assert(G.supplyCount[card] == supply-1);

assert(G.discardCount[p] == discard+1);

The variable r being 0 would indicate that the buyCard function did not fail to complete. After buyCard is complete, the number of buys and the supply of the particular card decreases. The number of discarded cards increases. After these asserts there is a switch case for every type of card, when that card is chosen, it makes sure that the number of coins after the buy is equal to the number of coins before minus the cost of the card. This test can run up to 2000 times.

I first ran the test on my own code. The draw card test ran 2000 times and takes 3 seconds to completely run its course. While the test seems to work fine, the function itself seems a little buggy. The program spits out the number of cards in a player’s hand. The test indicates that in most games a player has well over a hundred cards in a hand and the maximum is around 500 cards. I am no dominion expert, but this seems slightly off. After this, the buyCard test runs. According to the log, the game attempted to buy card number 3 with only 1 coin. The log also indicts that the buyCard function is run once entered before the program states that the number of buys in the program does not decrement; no other outputs are written to the file after this point. Running gcov on dominion.c after the test indicates that 7.96% of the 553 lines of code have executed. I ran this on all other dominion.c files in my group and achieved similar results.

After this test did not quite work, I decided just to look at my fellow teammates’ code and see if I could find any problems. I started with ellingsn’s buyCard function. I found that this function was barely even touched: it only adds the card in question to the discard pile of the current player. It does not take into account the cost of the card or the amount of money that player has on hand. From what I can see, though, there is not much that can cause a bug, other than the fact it is simply not completely implemented. nguyenta’s implementation is almost identical to the previous one.

I then moved to omaraa’s code. It takes into account the fact that one cannot take a card that has no supply and that a player with insufficient funds cannot buy the card. Other than that it just adds the card to the discard pile. It does not remove any money from the player’s hand. This one, too, is not implemented quite correctly. taylodav, tangke, and nicolgl’s implementations both seem to take everything into account.

I also hand inspected my adventurer implementation. This implementation starts with position 0 in the deck and counts up until 2 treasure cards are found. Otherwise the cards are placed in the discard pile. While this code is not quite implemented correctly, since the cards are put aside before being put in the discard, it also has a potential bug. If there are not 2 treasure cards in the deck, for whatever reason, this function could result in an infinite loop. I did not bother to check my buyCard by hand, since I have not changed the implementation since the last test report.

The automated test I wrote does not seem to work particularly well. Given more time I would have completely rewritten it; instead I tried to fix it, but this did not work out. Hand inspection worked better, though I saw a lot more incorrect implementations than code that can cause any sort of program breaking bug, assuming functions associated with it are also implemented correctly. As far as tools go, gcov seems to work well, though my test case makes it hard to tell.