Glen Nicol

CS 362

Alex Groce

1 May 2012

Test Report 1

Buy Card testing

Run make clean; make testdom. Then run ./writeTest to create test.out.

**Affected data structures:** supplycount, discard

**Affected fields:** coins, numBuys, discardCount

**Obvious Failure Case:** Able to purchase with

Not enough coins

No more buys

No more cards of specified type

**Check for:**

supplyCount going negative

Able to buy card with any of the obvious cases

Discard getting correct value

**Tests:** return number of detected errors.

**SupplyTest** assumes there is enough buys and coins for a player to buy out all of the supply cards. The rationale is to simply test that each each card is successfully gained into the discard pile and that the correct number of coins and buys are subtracted. It also checks to make sure that all cards can be successfully bought. Source code is below. I consider this to be the most strenuous test I performed on BuyCard.

int supplyHolder = dom->supplyCount[cardNumTest], i, errors = 0;

memcpy(check, dom, sizeof(struct gameState));

supplyHolder = dom->supplyCount[cardNumTest];

for(i = 0; i < supplyHolder + neg ;i++){// +neg to check for negative supplyCount case

//set ideal final conditions

if(i < supplyHolder){// only when count is >= 0

check->discard[0][check->discardCount[0]] = cardNumTest;

check->coins -= getCost(cardNumTest);

check->supplyCount[cardNumTest]--;

check->numBuys--;

check->discardCount[0]++;

}

buyCard(cardNumTest,dom);

if(NOISE){

//printf("%d / %d\n", i+1, supplyHolder);

}

}

if(assertions(cardNumTest, dom, check))

errors++;

return errors;

**CoinTest** assumed there is enough buys for the test to run. The game state should not change during this test unless it is applied to a curse or copper card, because they are free.

**NumBuyTest** assumes there is enough coins for the test to run. The test sets the number of buys to 1. Then checks for a valid buy event with the correct number of buys subtracted(1). Then checks the gameState for errors. Then attempts to buy another to see if the number of buys goes negative. Again checks for changes and for brevity attempts to buy one more time and again checks for changes.

**Assertions** is a method that checks for all of the above assumed fields and data structures for correct values. And then also does a memcmp to detect other unexpected changes. It does not highlight those changes but does make a vague comment that something is wrong. There are two modes, if STRICT is set then the method uses asserts that will halt upon error. If it is not set then it will print out the difference between the ideal value and the actual.

**Applied Tests to Group Code**

**ellingsn**

SupplyTest revealed that buyCard was not not changing coins, numBuys, or discardCount correctly. It was correctly moving the cards to the discard pile. Upon looking at the code Nothing looked wrong. However, after some trial and error, I found that (state->numBuys)--; somehow does not work as expected. Removing the parenthesis fixed the code and passed the tests.

Gcov output: 22.65% of 543

300: 250:int buyCard(int supplyPos, struct gameState \*state) {

-: 251: if(DEBUG){

300: 252: printf("Entering buyCard...\n");

-: 253: }

-: 254:

-: 255: // Nothing is ever changing the phase, but if it did:

-: 256: // if (state->phase != 1) { } else

300: 257: if (state->numBuys < 1) {

32: 258: printf("No more buys remaining\n");

268: 259: } else if (state->supplyCount[supplyPos] < 1){

16: 260: printf("Supply exhausted\n");

252: 261: } else if (getCost(supplyPos) > state->coins){

14: 262: printf("Not enough coins\n");

-: 263: } else {

238: 264: (state->numBuys)--;

238: 265: state->supplyCount[supplyPos]--;

238: 266: state->coins -= getCost(supplyPos);

-: 267: //// something with money... or... do you even discard money for each purchase?

-: 268: //// Maybe I'll leave this exciting little design flaw -- I mean bug -- as

-: 269: //// an exercise for the reader XD

238: 270: int who = state->whoseTurn;

238: 271: state->discard[who][state->discardCount[who]] = supplyPos;

238: 272: state->discardCount[who]++;

-: 273: }

-: 274:

300: 275: return 0;

-: 276:}

**lesliew**

Coin tests reported incorrect coin values by -1 for all supply cards except curses which were off by -3; I found that the buyCard function was using the updateCoins function inside and overriding my tests coin overrides for tests. After temporarily commenting out that line and replacing it with a simple assignment to the coins field based on cost, the coins reported correct values. I could change my test to account for the actually moving cards out of he hand and using updateCoins but it seemed like it was an unnecessary step because once the player is in the buy phase they cants get any more cards to increase their coins. However numBuys was still 1 higher than needed for cards that were free to buy, curses and copper. I found that the inequality controlling whether or not a card could be bought did not allow for cards that cost 0 coins to be bought because it only a greater than conditional. Adding an = to the inequality fixed this.

NumBuyTest came back false, and I could tell that buy card was not checking for enough buy cards because on the second and third attempts to buy a card it should not succeed, but the test reported the coin value being off by multiples of 1 and 2 of the card cost signifying the card was being purchased when it shouldn't be. The code has no checks for numBuy.

There was an anomaly, when attempting to purchase a duchy coins are off by a much larger number than all the other cards.

When I commented out the code for moving coins used to buy previous cards into the discard the anomaly disappeared and the code passed the numBuyTest.

Finally, running supplyTest reported an error on each supply stack showing that one too many cards were being bought than should have been. The test specifically tests for this condition for when the stack is empty. The obvious error is that the suppplyCount variable is not checked for less than or equall to 0 in buyCard.

Assuming the manual moving of coin cards into the discard is not needed my test has successfully caught all the errors in this implementation. And the fixes are described above.

Gcov output: 24.59% of 545

300: 250:int buyCard(int supplyPos, struct gameState \*state) {

-: 251: if(DEBUG){

300: 252: printf("Entering buyCard...\n");

-: 253: }

-: 254:

-: 255: // THIS IMPLEMENTATION IS NOT WORKING, BUT LETS YOU PLAY. FIX IT

300: 256: int owed = getCost(supplyPos);

300: 257: int who = whoseTurn(state);

-: 258: //updateCoins(who, state , 0);

300: 259: if(state->supplyCount[supplyPos] <= 0)

16: 260: return -1;

284: 261: if(state->numBuys <= 0)

32: 262: return -1;

252: 263: if(state->coins >= owed){

238: 264: state->numBuys--;

238: 265: gainCard(supplyPos, state, 0, who);

238: 266: int i = 0;

-: 267: /\*while(owed > 0){

-: 268: if (handCard(i,state) == copper){

-: 269: discardCard(i, who, state, 0);

-: 270: owed--;

-: 271: }

-: 272: if (handCard(i,state) == silver){

-: 273: discardCard(i, who, state, 0);

-: 274: owed = owed - 2;

-: 275: }

-: 276: if (handCard(i,state) == gold){

-: 277: discardCard(i, who, state, 0);

-: 278: owed = owed - 3;

-: 279: }

-: 280: i++;

-: 281: }

-: 282:\*/ //updateCoins(who,state,0);

238: 283: state->coins -= getCost(supplyPos);

-: 284: }

252: 285: return 0;

-: 286:}