Glen Nicol

CS362

Alex Groce

11 June 2012

Final Test Report

This report is an addition to the previous test reports specifically aimed at the previous tests including a random tester aimed to test the execution of the main functions in dominion, as well as a test written specifically to test for the corner cases of buyCard. The details of those will be copy and pasted from the previous reports and appended to the end of this final report focused on my third and final test involving CBMC and the mine card implementation.

**Test Strategy:** Model Checking

**Test Details:**

The test is based on the CBMC model checker and uses a lot of assumes to handle a correct game state is used for the model checking process. The mine card trashes a treasure and draws another treasure from the supply piles.

In order to test this I used asserts to check for the hand count of treasures stays the same because one is trashed and another immediately replaces it with a larger value coin card hence the coins for that hand increase as well. I also checked for the supplyCount of the gained card decreases by 1. The discardCount should also increase by one. The deckCount should not change.

**Surgery:**

In order to save time and processing complexity I reduced the MAX\_HAND, and MAX\_DECK to 5 dominion.h. I did not have to make any modifications to the code for CBMC to run quickly.

**Command to run CBMC:** cbmc MineTest.c dominion.c --unwind 5

make testdom will remove old oldput and run the above command

**Important Test code:**

\_\_CPROVER\_assume(p == G0.whoseTurn);

\_\_CPROVER\_assume(G0.handCount[p] > 0);

\_\_CPROVER\_assume(G0.handCount[p] < MAX\_HAND);

\_\_CPROVER\_assume(G0.numActions > 0);

\_\_CPROVER\_assume(G0.phase == 0);

\_\_CPROVER\_assume(G0.deckCount[p] >= 0);

\_\_CPROVER\_assume(G0.deckCount[p] < MAX\_DECK);

\_\_CPROVER\_assume(G0.discardCount[p] >= 0);

\_\_CPROVER\_assume(G0.discardCount[p] < MAX\_DECK);

\_\_CPROVER\_assume(G0.deckCount[p] + G0.discardCount[p] < MAX\_DECK);

\_\_CPROVER\_assume(minePos >= 0);

\_\_CPROVER\_assume(minePos < G0.handCount[p]);

\_\_CPROVER\_assume(G0.hand[p][minePos] == mine);

\_\_CPROVER\_assume(tPos >= 0);

\_\_CPROVER\_assume(tPos < G0.handCount[p]);

\_\_CPROVER\_assume(G0.hand[p][tPos] >= copper && G0.hand[p][tPos] < gold);

//code to choose correct treasure option

c = G1.hand[p][tPos];

int tChooser;

switch (c){

case copper : tChooser = silver;break;

case silver : tChooser = gold;break;

default : break;

}

res = playCard(minePos, tPos, tChooser, input3, &G1);

////////////////////////////////////////////////////////////////

for (i = 0; i < G1.handCount[p]; i++) {// Count the number of treasures in the hand

c = G1.hand[p][i];

tch2 += ((c == copper) || (c == silver) || (c == gold));

}

assert(res == 0);

assert(G0.discardCount[p]+1 == G1.discardCount[p]);

assert(G0.deckCount[p] == G1.deckCount[p]);

assert(G0.supplyCount[tChooser] == G1.supplyCount[tChooser] +1);

assert(tch2 == tch);

assert(gc0 < gc1);

**Results:**

The mine implementation is correct and was successfully verified. I did not check other peoples code because it is all the same.

**Random Tester**

**Testing strategy:** Coverage testing

By calling the 4 major functions, InitializeGame, BuyCard, PlayCard, and Endturn, many of the other functions will be subsequently called and can be tested in a more realistic fashion.

This will be done by simulating games by simply calling these functions a specified number of times per game with multiple game states based on a random seed with various player numbers.

By using grep or similar I can find where cards are played and possible errors are reported in the large test.out file. In order to make sure invalid cards are not introduced I will check for them after each playCard and BuyCard call. Furthermore, a visual inspection will also likely reveal errors that are too hard to come up with checks inside the code.

**Increasing coverage:**

My initial test was based off of the one given in class. And got an initial coverage of around 29% percent. By looking at the gcov output I found that the end game code was never being called. So I added checks for this in my BuyCard function and in the test as well to catch these conditions after a playCard call. Coverage went up to 43%. Then I realized that the majority of dominion.c is in the cardEffect function for all the specialty cards. I examined the gcov output again and found that when playCard was being called it was never executing the cardEffect function. CardEffect is a very large part of dominion.c. By tuning the test I was able to get correct random options for playCard that successfully entered cardEffect. After this the coverage jumped up to 75 %.

**Tester Code important parts:**

for (i = 0; i < G.handCount[G.whoseTurn]; i++) {

card = G.hand[G.whoseTurn][i];

cardNumToName(card, cardName); // allow for card name to be easily read in output

printf ("%s ", cardName);

if ((card < 0) || (card > treasure\_map)) {

printf ("\n INVALID CARD IN HAND.\n");

exit(-1);

}

}

printf (": ");

op = floor(Random() \* 3);

card = (floor(Random() \* G.handCount[G.whoseTurn])); // Choose random card from hand

val = floor(Random() \* ((treasure\_map+2)- 1)) ; Choose random card in the current game

switch(op) {

case 0:

printf ("> ENDTURN: ");

r = endTurn(&G);

break;

case 1:

cardNumToName(val, cardName);

if(TEST\_DETAILS)

printf ("> BUY %d%s $%d/%d: ",val, cardName, G.coins, getCost(val));

else printf ("> BUY %s: ", cardName);

r = buyCard(val, &G);

check\_decks(&G);

break;

case 2:

cardNumToName( handCard(card, &G), cardName);

printf ("> PLAY %s: ", cardName);

r = playCard(card, floor(Random() \* G.handCount[G.whoseTurn]), floor(Random() \* ((treasure\_map+2)-1)) , floor(Random() \* G.handCount[G.whoseTurn]), &G);

//r = playCard(card,-1,-1,-1, &G);

check\_decks(&G);

break;

}

printf ("\tError code: %d\n", r);

if(isGameOver(&G)){

printf("End Game conditions met ");

break;

}

**Tuned PlayCard**

r = playCard(card, floor(Random() \* G.handCount[G.whoseTurn]), floor(Random() \* ((treasure\_map+2)-1)) , floor(Random() \* G.handCount[G.whoseTurn]), &G);

The first random number chooses a random card out of the players hand, the second chooses a random out of the current available cards in the game. And the last parameter does the same as the first random parameter. In the case of **Feast** where cards must be taken out of the deck, to avoid an infinite loop a couple lines must be added to choose another random card.

**Steps for grader:**

Use the code in the bigtest directory of my tester-code directory. DomTest.c is the source for the test.

If you are using another dominion.c file other than mine **you must a**dd this line to the first two if/else if within the Feast card effect to avoid an infinite Loop. Around line 670.

choice1 = floor(Random() \*((treasure\_map)-1));

In my Testing I used 5000 Games that are defined in my bigTest.c file. However, I shrunk that down to only 500 for submission. I figured that it would make the files much smaller and easier to grade.

Calling “make testdom” will compile the code and then output the test results into test.out. No other action is required.

**Bugs Caught In my dominion.c**

**1.** When adventurer is played handCount DeckCount DiscardCount offsets come out to be nonzero indicating a card has gone missing.

<52,60> STATE: 1's turn, DECK:3, DISCARDS: 5, BUYS: 1, cards in hand 5: Copper Estate Adventurer Copper Estate : > PLAY Adventurer: Error code: 0

<52,61> STATE: 1's turn, DECK:1, DISCARDS: 5, BUYS: 1, cards in hand 6: Copper Estate Copper Copper Estate Copper : > PLAY Estate: Error code: -1

Deck decreases by 2, discard is unchanged, hand increases 1. offset -1.

<57,319> STATE: 1's turn, DECK:11, DISCARDS: 5, BUYS: 1, cards in hand 5: Estate Baron Adventurer Curse Copper : > PLAY Adventurer:

added card 0 back into discard

added card 1 back into discard

added card 2 back into discard Error code: 0

<57,320> STATE: 1's turn, DECK:6, DISCARDS: 8, BUYS: 1, cards in hand 6: Estate Baron Copper Curse Copper Silver : > PLAY Baron: Error code: -1

Offset -1

**FIX:** Shuffle requires the discard pile to be moved to deck then shuffle can be called. I also added breaks and continues to the shuffle code to ensure a deckCount of 0 was not used in lower code resulting in a -1.

I also found that the built in function, discardCard() does not actually put the card into the discard pile which is why the offset was always -1. Everytime I was discarding the adventurer card it was actually being trashed because the code was not complete. Adding these two lines of code in the trashflag < 1 condition fixes this.

state->discard[currentPlayer][state->discardCount[currentPlayer]] = state->hand[currentPlayer][handPos];

state->discardCount[currentPlayer]++;

**2.** Multiple Cards are not discarded upon play.

* 1. Adventurer played successfully, except not discarded. Detected on <10,504><10,626> <10,806><11,1396>**FIX:**  add discardCard(handPos, currentPlayer, state, 0); before returns.
  2. Feast is not trashed on play, Detected on <13,657>

**FIX:** Add code to trash after gaining card.

if(state->hand[currentPlayer][i] == feast)

x = i;

}

discardCard(x,currentPlayer, state, -1);// trash feast

* 1. Baron is not discarded on successful play. <13,928>**FIX**: add discardCard(handPos, currentPlayer, state, 0); before return 0;

1. Baron doesn't gain an estate when the player has none in his/her hand. <1,1055> But does on <2,950>
2. Gardens are not implemented. Will add code to trash them on play.

**REPORT:** I sent an email to the whole class informing them of the discarding bugs in feast and baron. The code should not have these in their code in this final report. Here are the stats to after that bug report was sent.

**Bugs Caught**

**Ellingsn: Same bugs as entry 2 from my code above. Adventurer is not Implemented at all. Adding code to trash adventurer when played. These were not fixed from the previous report.**

**Lines executed:63.53% of 573 –** changed from previous report.

**Branches executed:56.44% of 427**

**Taken at least once:49.18% of 427**

**Calls executed:60.91% of 110**

**Omaraa:** Same bugs as 2.1-3 from my section. Detected lines are not the same. These errors were not fixed from the previous report.

**Lines executed:62.64% of 530**

**Branches executed:55.23% of 411**

**Taken at least once:48.42% of 411**

**Calls executed:63.37% of 101**

**Taylodav:** Tester halts for unknown reason on this second run of my tester. There isn't an invalid card detected, it just halts mid printf. To see the original test report see report 2.

**Lesliew:** Adventurer was never called in a play roll. I tried changing the test to change the seed but even with 1000 games it never came up. I cannot test if Adventurer is discarded after play, but from code inspection I can tell that it was not. Baron and Feast suffer then same bug due to it being in the master file. No errors were fixed from the previous report.

**Lines executed:57.33% of 546**

**Branches executed:51.56% of 417**

**Taken at least once:42.93% of 417**

**Calls executed:47.37% of 114**

**Tangke:** Code compiles this time around and Feast and baron discard bugs were correctly fixed. Adventurer is not discarded.

**Lines executed:63.93% of 549**

**Branches executed:56.50% of 423**

**Taken at least once:49.88% of 423**

**Calls executed:62.75% of 102**

**Nguyenta:** Same as 2.1-3 from my section. Bugs were not fixed.

**Lines executed:77.78% of 513**

**Branches executed:74.31% of 397**

**Taken at least once:64.99% of 397**

**Calls executed:83.52% of 91**

**Shearini:** Same bugs as 2.1-3 from my section. Detected lines are not the same. Bugs were not fixed from last report.

**Lines executed:79.56% of 548**

**Branches executed:77.51% of 409**

**Taken at least once:69.19% of 409**

**Calls executed:88.89% of 99**

**Mcconnjo:** The test did not complete because an invalid card was detected. The test print out is covered in detail in my second report. The bug was not fixed from that same report even after I personally emailed a detailed bug report.

**Lines executed:60.98% of 528**

**Branches executed:56.97% of 409**

**Taken at least once:46.70% of 409**

**Calls executed:46.24% of 93**

**RESULTS of Random Tester:**

This tester was in my opinion the most effective at catching errors. It would have been more effective if I had applied delta debugging to the test cases. It was particularly good at catching errors that involved moving cards around between piles. This is evident in the bugs caught involving baron and feast not getting discarded in the primary dominion.c file we were given as well as my implementation of adventurer.

**BuyCard Test**

The results of my first test were much less significant and also should be caught from the random tester. This is because it was a test that was specifically written to catch the corner cases of buycard in a specified fashion. There was no random inputs or state. Because of the limited nature I will not include it's details again in this final report.

**Final Report Details**

**Communication With Classmates:**

I used emails to communicate with my classmates. Besides the quick and easy question about dominion or visual studio however I did not receive any feedback on my bug reports from anyone. I alos did not receive any tickets on beaversource reporting on my code. Nor could I find anything in the bug report directory that applied to me or a general announcment.

I sent this bug report to the class list  
  
I found a couple bugs and thought I'd share them with the class.   
  
1.int discardCard(int handPos, int currentPlayer, struct gameState \*state, int trashFlag)

This function never actually puts the card in the discard pile.

FIX: Add the following lines inside the conditional for trashflag < 1   
 state->discard[currentPlayer][state->discardCount[currentPlayer]] = state->hand[currentPlayer][handPos];   
 state->discardCount[currentPlayer]++;   
  
2. Feast is not trashed after use.   
 Before fix:   
  
//Reset Hand   
 for(i = 0; i <= state->handCount[currentPlayer]; i++){   
 state->hand[currentPlayer][i] = temphand[i];   
 temphand[i] = -1;   
 }   
 //Reset Hand   
 return 0;   
  
after **fix:** \*\*'s represent new code

//Reset Hand   
for(i = 0; i <= state->handCount[currentPlayer]; i++){

state->hand[currentPlayer][i] = temphand[i];   
 temphand[i] = -1;   
  
 if(state->hand[currentPlayer][i] == feast)\*\*\*\*\*\*\*

x = i;\*\*\*\*\*\*\*

}\*\*\*\*\*\*   
discardCard(x,currentPlayer, state, 1);// trash feast\*\*\*\*\*\*\*\*   
 //Reset Hand   
 return 0;   
  
3. Baron is not discarded either.   
FIX: add this right before the return for baron.   
discardCard(handPos, currentPlayer, state, 0);

**Revisions:**

All revisions have been covered in the bugs caught section of the random tester or in the above bug report.

**Regression:**

I reran the random tester on everyone's code. Primarily to see if they had fixed the errors I had reported in the email. But also to see if any other obvious errors were present. I did not find any obvious errors that showed up like my original run of the random tester. Unfortunately the only differences in my two runs of the test for each of my group members were rather trivial. Tangke's code originally didn't compile but correctly discarded baron and feast as I described in my emailed report. Ellingsnhad different line coverages then the first run. And Taylodav has an unknown error that halts the test.

**PRE Quality of Code:**

The original code we were given for dominion had a mostly working implementation of dominion. With the exception of the few card effects that haven't been implemented like gardens and adventurer, and buyCard which is crucial to the game playing correctly. The original dominion.c had **1248 LoC.**

I did not find many bugs in the code we were given. So either my testing net was not fine enough or the code is fairly well implemented to begin with.

**POST Quality of Code:**

The bugs I found, including the baron and feast card not discarding after being played and the discard function not trashing card correctly the code we were given is largely the same. My final **LoC** count for dominion.c is **1391**.The biggest change was in my adventurer implementation. This complicated card was not terribly difficult to code originally but took me a awhile to debug and the fix is documented in the random tester section above. My Buy card implementation passed all of my tests and I am quite confident that my contribution to these two sections of code are very close if not completely in line with the dominion specification. To be honest I don't feel like we've done a very good job covering much this term and improving the implementation because I have not received many bug reports from other students or my team.

**REPORT 2**

**Testing strategy: Coverage testing**

By calling the 4 major functions, InitializeGame, BuyCard, PlayCard, and Endturn, many of the other functions will be subsequently called and can be tested in a more realistic fashion.

This will be done by simulating games by simply calling these functions a specified number of times per game with multiple game states based on a random seed with various player numbers.

By using grep or similar I can find where cards are played and possible errors are reported in the large test.out file. In order to make sure invalid cards are not introduced I will check for them after each playCard and BuyCard call. Furthermore, a visual inspection will also likely reveal errors that are too hard to come up with checks inside the code.

Increasing coverage:

My initial test was based off of the one given in class. And got an initial coverage of around 29% percent. By looking at the gcov output I found that the end game code was never being called. So I added checks for this in my BuyCard function and in the test as well to catch these conditions after a playCard call. Coverage went up to 43%. Then I realized that the majority of dominion.c is in the cardEffect function for all the specialty cards. I examined the gcov output again and found that when playCard was being called it was never executing the cardEffect function. CardEffect is a very large part of dominion.c. By tuning the test I was able to get correct random options for playCard that successfully entered cardEffect. After this the coverage jumped up to 75 %.

**Tester Code important parts:**

for (i = 0; i < G.handCount[G.whoseTurn]; i++) {

card = G.hand[G.whoseTurn][i];

cardNumToName(card, cardName); // allow for card name to be easily read in output

printf ("%s ", cardName);

if ((card < 0) || (card > treasure\_map)) {

printf ("\n INVALID CARD IN HAND.\n");

exit(-1);

}

}

printf (": ");

op = floor(Random() \* 3);

card = (floor(Random() \* G.handCount[G.whoseTurn])); // Choose random card from hand

val = floor(Random() \* ((treasure\_map+2)- 1)) ; Choose random card in the current game

switch(op) {

case 0:

printf ("> ENDTURN: ");

r = endTurn(&G);

break;

case 1:

cardNumToName(val, cardName);

if(TEST\_DETAILS)

printf ("> BUY %d%s $%d/%d: ",val, cardName, G.coins, getCost(val));

else printf ("> BUY %s: ", cardName);

r = buyCard(val, &G);

check\_decks(&G);

break;

case 2:

cardNumToName( handCard(card, &G), cardName);

printf ("> PLAY %s: ", cardName);

r = playCard(card, floor(Random() \* G.handCount[G.whoseTurn]), floor(Random() \* ((treasure\_map+2)-1)) , floor(Random() \* G.handCount[G.whoseTurn]), &G);

//r = playCard(card,-1,-1,-1, &G);

check\_decks(&G);

break;

}

printf ("\tError code: %d\n", r);

if(isGameOver(&G)){

printf("End Game conditions met ");

break;

}

**Tuned PlayCard**

r = playCard(card, floor(Random() \* G.handCount[G.whoseTurn]), floor(Random() \* ((treasure\_map+2)-1)) , floor(Random() \* G.handCount[G.whoseTurn]), &G);

The first random number chooses a random card out of the players hand, the second chooses a random out of the current available cards in the game. And the last parameter does the same as the first random parameter. In the case of **Feast** where cards must be taken out of the deck, to avoid an infinite loop a couple lines must be added to choose another random card.

**Steps for grader:**

Use the code in the bigtest directory of my tester-code directory. DomTest.c is the source for the test.

If you are using another dominion.c file other than mine **you must a**dd this line to the first two if/else if within the Feast card effect to avoid an infinite Loop. Around line 670.

choice1 = floor(Random() \*((treasure\_map)-1));

In my Testing I used 5000 Games that are defined in my bigTest.c file. However, I shrunk that down to only 500 for submission. I figured that it would make the files much smaller and easier to grade.

Calling “make testdom” will compile the code and then output the test results into test.out. No other action is required.

**Bugs Caught In my dominion.c**

**1.** When adventurer is played handCount DeckCoun DiscardCount offsets come out to be nonzero indicating a card has gone missing.

<52,60> STATE: 1's turn, DECK:3, DISCARDS: 5, BUYS: 1, cards in hand 5: Copper Estate Adventurer Copper Estate : > PLAY Adventurer: Error code: 0

<52,61> STATE: 1's turn, DECK:1, DISCARDS: 5, BUYS: 1, cards in hand 6: Copper Estate Copper Copper Estate Copper : > PLAY Estate: Error code: -1

Deck decreases by 2, discard is unchanged, hand increases 1. offset -1.

<57,319> STATE: 1's turn, DECK:11, DISCARDS: 5, BUYS: 1, cards in hand 5: Estate Baron Adventurer Curse Copper : > PLAY Adventurer:

added card 0 back into discard

added card 1 back into discard

added card 2 back into discard Error code: 0

<57,320> STATE: 1's turn, DECK:6, DISCARDS: 8, BUYS: 1, cards in hand 6: Estate Baron Copper Curse Copper Silver : > PLAY Baron: Error code: -1

Offset -1

FIX: Shuffle requires the discard pile to be moved to deck then shuffle can be called. I also added breaks and continues to the shuffle code to ensure a deckCount of 0 was not used in lower code resulting in a -1.

I also found that the built in function, discardCard() does not actually put the card into the discard pile which is why the offset was always -1. Everytime I was discarding the adventurer card it was actually being trashed because the code was not complete. Adding these two lines of code in the trashflag < 1 condition fixes this.

state->discard[currentPlayer][state->discardCount[currentPlayer]] = state->hand[currentPlayer][handPos];

state->discardCount[currentPlayer]++;

**2.** Multiple Cards are not discarded upon play.

* 1. Adventurer played successfully, except not discarded. Detected on <10,504><10,626> <10,806><11,1396>**FIX:**  add discardCard(handPos, currentPlayer, state, 0); before returns.
  2. Feast is not trashed on play, Detected on <13,657>

**FIX:** Add code to trash after gaining card.

if(state->hand[currentPlayer][i] == feast)

x = i;

}

discardCard(x,currentPlayer, state, -1);// trash feast

* 1. Baron is not discarded on successful play. <13,928>**FIX**: add discardCard(handPos, currentPlayer, state, 0); before return 0;

1. Baron doesn't gain an estate when the player has none in his/her hand. <1,1055> But does on <2,950>
2. Gardens are not implemented. Will add code to trash them on play.

**Bugs Caught**

**Ellingsn:** Same bugs as entry 2 from my code above. Adventurer is not Implemented at all. Adding code to trash adventurer when played.

Lines executed: 79.46% of 516

**Omaraa:** Same bugs as 2.1-3 from my section. Detected lines are not the same.

Lines executed 62.76% of 529

**Taylodav:** Same bugs as 2.1-3 from my section. Detected lines are not the same.

Lines executed:59.10% of 533

**Lesliew:** Adventurer was never called in a play roll. I tried changing the test to change the seed but even with 1000 games it never came up. I cannot test if Adventurer is discarded after play, but from code inspection I can tell that it was not. Baron and Feast suffer then same bug due to it being in the master file.

Lines executed:53.47% of 548

**Tangke:** I did not have to insert the code into the Feast card effect because there was already similar code implemented. However, the code does not compile. I will not spend the time to debug one person's code.

**Nguyenta:** Same as 2.1-3 from my section.

Lines executed:77.86% of 515

**Shearini:** Same bugs as 2.1-3 from my section. Detected lines are not the same.

Lines executed:79.78% of 549

**Mcconnjo:** The test did not complete because an invalid card was detected. The test print out is:

<1,423> STATE: 0's turn, DISCARDS: 30, BUYS: 1, cards in hand 5: Adventurer Gardens Copper Province Province : > PLAY Adventurer: An invalid card# -1 was detected in 0's discard Error code: 0

<1,424> STATE: 0's turn, DISCARDS: 351, BUYS: 1, cards in hand 8: Adventurer Gardens Copper Province Province Copper Copper Copper : > PLAY Copper: An invalid card# -1 was detected in 0's discard Error code: -1

I was able to eliminate the invalid card by moving the decrement counter from a post to a pre decrementor. However, it is still adding far too many to the discard pile.

I also found that the implementation never does the moving of cards correctly.

<4,31> STATE: 0's turn, DECK:50, DISCARDS: 38, BUYS: 1, cards in hand 5: Salvager Copper Adventurer Baron Copper : > PLAY Adventurer: Error code: 0

<4,32> STATE: 0's turn, DECK:38, DISCARDS: 42, BUYS: 1, cards in hand 7: Salvager Copper Adventurer Baron Copper Copper Silver : > PLAY Copper: Error code: -1

The changes are highlighted in yellow. Deck goes down by 12, discard goes up by 4 and hand goes up by 2. 4 + 2 -12 = -6

4,234> STATE: 0's turn, DECK:47, DISCARDS: 55, BUYS: 1, cards in hand 5: Adventurer Feast Sea Hag Gardens Baron : > PLAY Adventurer: Error code: 0

<4,235> STATE: 0's turn, DECK:27, DISCARDS: 63, BUYS: 1, cards in hand 7: Adventurer Feast Sea Hag Gardens Baron Copper Copper : > BUY Gold: Error code: 0

Deck decreases by 20, discard increases by 8 and hand increases by 2. 8+2 -20 = -10. This offset is always off by one half of the number of cards that is taken out of the deck. These cards are never put back into play.

I was unable to find beaversource's ticketing system so I emailed mcconnjo directly and described the bug to him.

Lines executed: 80.19% of 530

**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:}**