Erin Alltop
CS362 – Winter 2018
Assignment 5

Refactored Code

For this assignment, I first took the refactored dominion.c code from one of my teammates and added it to its own folder, boesejDominion under my projects/alltope directory. I then ran all my card and unit tests on the code. Jonathan and I only shared bugs in Adventurer and Smithy card functions, and those were the only cards that I found bugs in.

Bug-Reports

==============================

Title: Player deck counts are incorrect after running Smithy Card
Class: Medium Bug
Date: 2-24-18
Reported By: Erin Alltop
Email: alltope@oregonstate.edu
Product: Dominion Version: boesej refactored code (from Assignment 2)
Platform: Linux
Browser: Flip School Server
URL: N/A
Is it reproducible: Yes

Description
===========
Summary: Deck count for the player is incorrect after calling Smithy Card function
I created an instance of the Dominion game and called the Smithy Card function.
I then ran test code to check various counts. I found that the deck count was showing an incorrect amount of cards in the current player's deck, indicating that one card was "lost."
This error happens on every run of the Smithy Card.

Steps to Produce/Reproduce

Any time the Smithy Card is called this bug occurs.

Expected Results

The deck count should be the original deck count plus 3

Actual Results

The deck count showed the original plus 2

Workarounds

No known workarounds at this time

Attachments

Test code output for relevant bug:

```
TEST 2: Three cards are drawn from the current player's deck
deck count = 1, expected = 2
TEST 2 FAILED!
```

==============================

Title: Player hand and deck counts incorrect after running Adventurer Card
Class: Serious Bug
Date: 2-24-18
Reported By: Erin Alltop
Email: alltope@oregonstate.edu
Product: Dominion Version: boesej refactored code (from Assignment 2)
Platform: Linux
Browser: Flip School Server
URL: N/A
Is it reproducible: Usually

Description
===========
Summary: Hand and Deck counts for current player are incorrect after running Adventurer Card function
I created an instance of the Dominion game and called the Adventurer Card function.
I then ran test code to check various counts. I found that both the hand and deck counts for the current player were not as expected.
This error usually happens on every run of the Adventurer Card, but if the player's hand has the amount of treasure cards needed without reshuffling it might not occur.

Steps to Produce/Reproduce

Any time the Adventurer Card function is called this bug occurs

Expected Results

Hand count is expected to be increased by two cards after two cards are drawn.
Deck count is expected to be reduced by two.

Actual Results

Hand count was three less than expected
Deck count was three less than expected

Workarounds

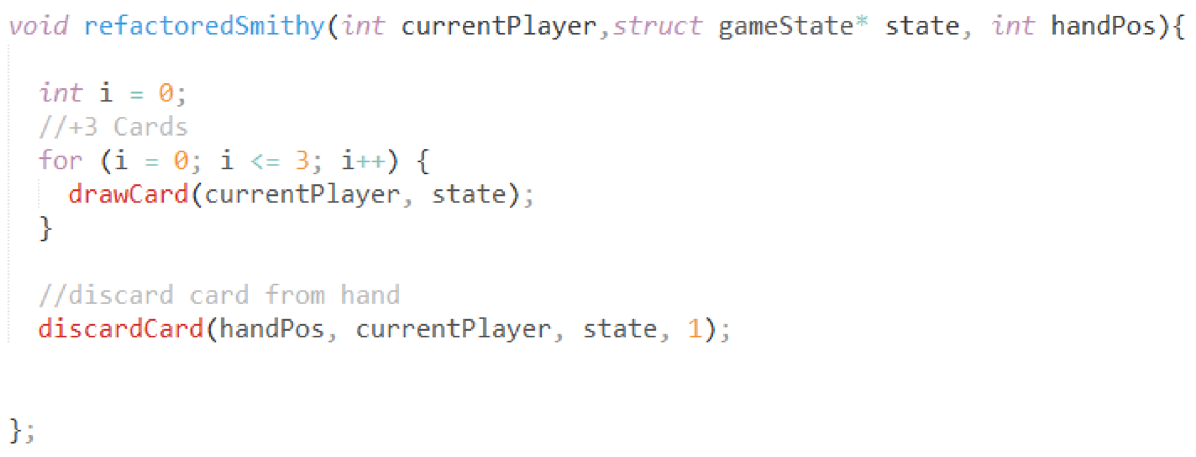
No known workaround at this time

Attachments

Test code for relevant bug:

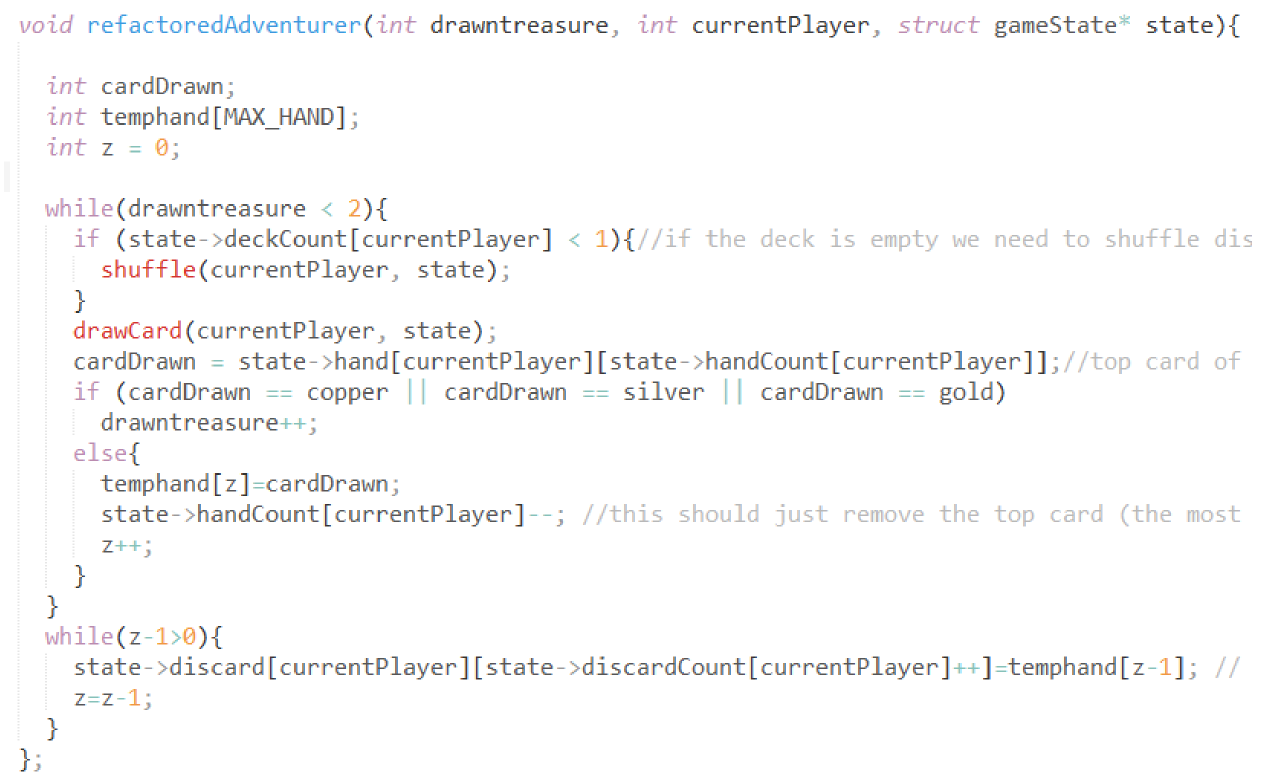
```
TEST 1: Current player receives 2 cards.
hand count = 4, expected = 7
TEST 1 FAILED!

TEST 2: Two cards are drawn from the current player's deck.
deck count = 0, expected = 3
TEST 2 FAILED!
```

I thought that Jonathan did a great job in making his bugs for these two cards. The bug in the Smithy Card caught me off guard until I did a careful review of his code. If I was not testing this function explicitly it would have taken me considerably longer to find the bug. My test cases did pick up the error, but my name for the test "Three cards are drawn from the current player's deck" gives away my original intent for the test and the difference in his bug. The cards were indeed drawn from the current player's deck, but the deck count was incorrect because the function was trashing the card instead of discarding it back to his deck. Note the trash flag in discardCard:



```
void refactoredSmithy(int currentPlayer,struct gameState* state, int handPos){

  int i = 0;
  //+3 Cards
  for (i = 0; i <= 3; i++) {
    drawCard(currentPlayer, state);
  }

  //discard card from hand
  discardCard(handPos, currentPlayer, state, 1);


};
```

The Adventurer Card bug has an even subtler affect on the game. My tests did catch this bug but it did not give a simple output as to why. After reviewing Jonathan's code I could see that he implemented a bug by having the incorrect card drawn in the cardDrawn assignment:



```
void refactoredAdventurer(int drawntreasure, int currentPlayer, struct gameState* state){

  int cardDrawn;
  int temphand[MAX_HAND];
  int z = 0;

  while(drawntreasure < 2){
    if (state->deckCount[currentPlayer] < 1){//if the deck is empty we need to shuffle dis
      shuffle(currentPlayer, state);
    }
    drawCard(currentPlayer, state);
    cardDrawn = state->hand[currentPlayer][state->handCount[currentPlayer]];//top card of
    if (cardDrawn == copper || cardDrawn == silver || cardDrawn == gold)
      drawntreasure++;
    else{
      temphand[z]=cardDrawn;
      state->handCount[currentPlayer]--; //this should just remove the top card (the most
      z++;
    }
  }
  while(z-1>0){
    state->discard[currentPlayer][state->discardCount[currentPlayer]++]=temphand[z-1]; //
    z=z-1;
  }
};
```

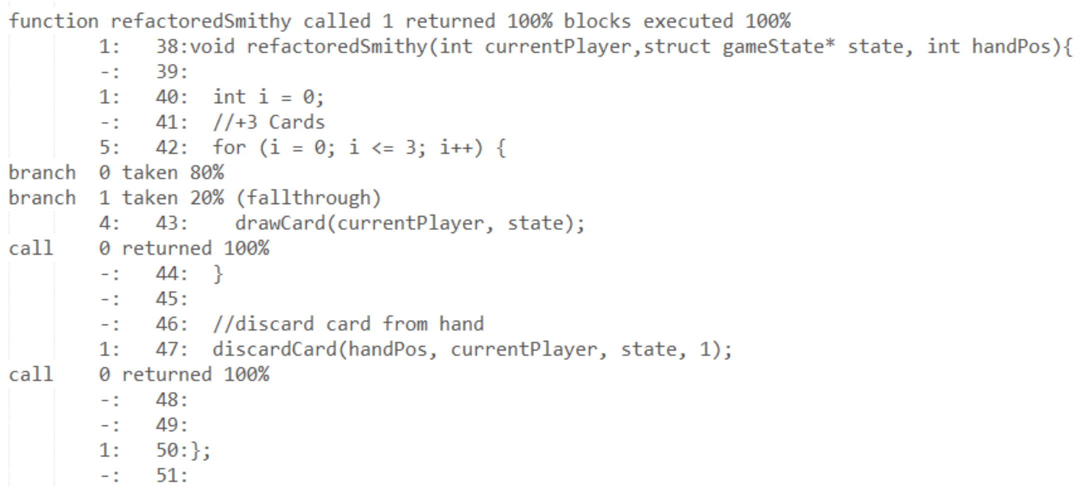
My testing code indicated that both the hand and deck counts were three less than expected. Going off of the testing results alone, my first thought was that his code was potentially not drawing any cards but discarding as normal, but that wouldn't explain why the deck count was incorrect. I was only able to find the bug after a careful review of his refactored code. The way that this bug works with the rest of the code is not as clear as the Smithy Card. I believe that the incorrect counts must occur when the shuffle function is used, which throws off the counts, but I am not 100% certain.

Test Report

I ran all of the tests that I had created for cards and unit tests on Jonathan's code. While I had a bit of trouble making up the test cases throughout the first weeks of the class, testing against someone else's code made me have a much greater appreciation for a good testing suite. The fact that I did not have to refactor my own testing code to catch the bugs in Adventurer and Smithy cards made me feel confident in my testing suite overall.

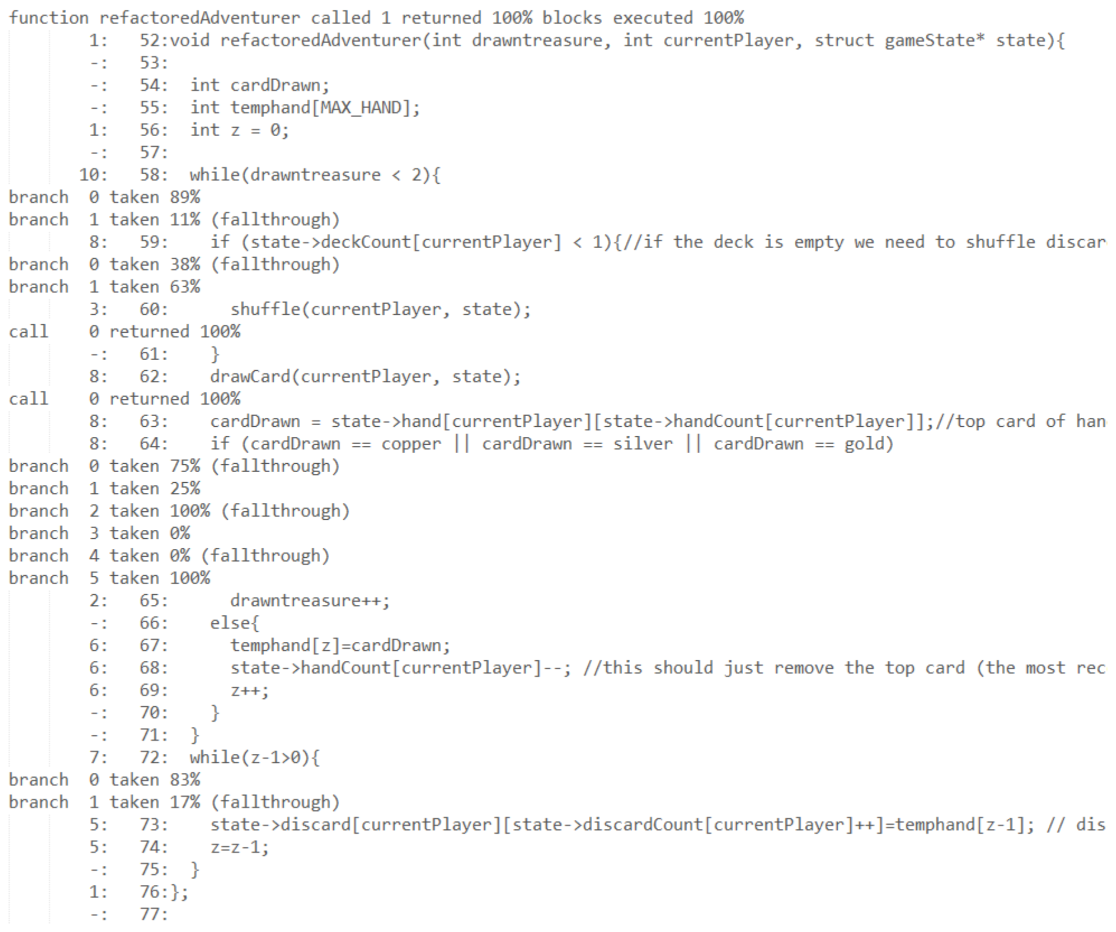
Coverage Information –

Smithy Card Test Coverage – 100% Branch (cardtest1)



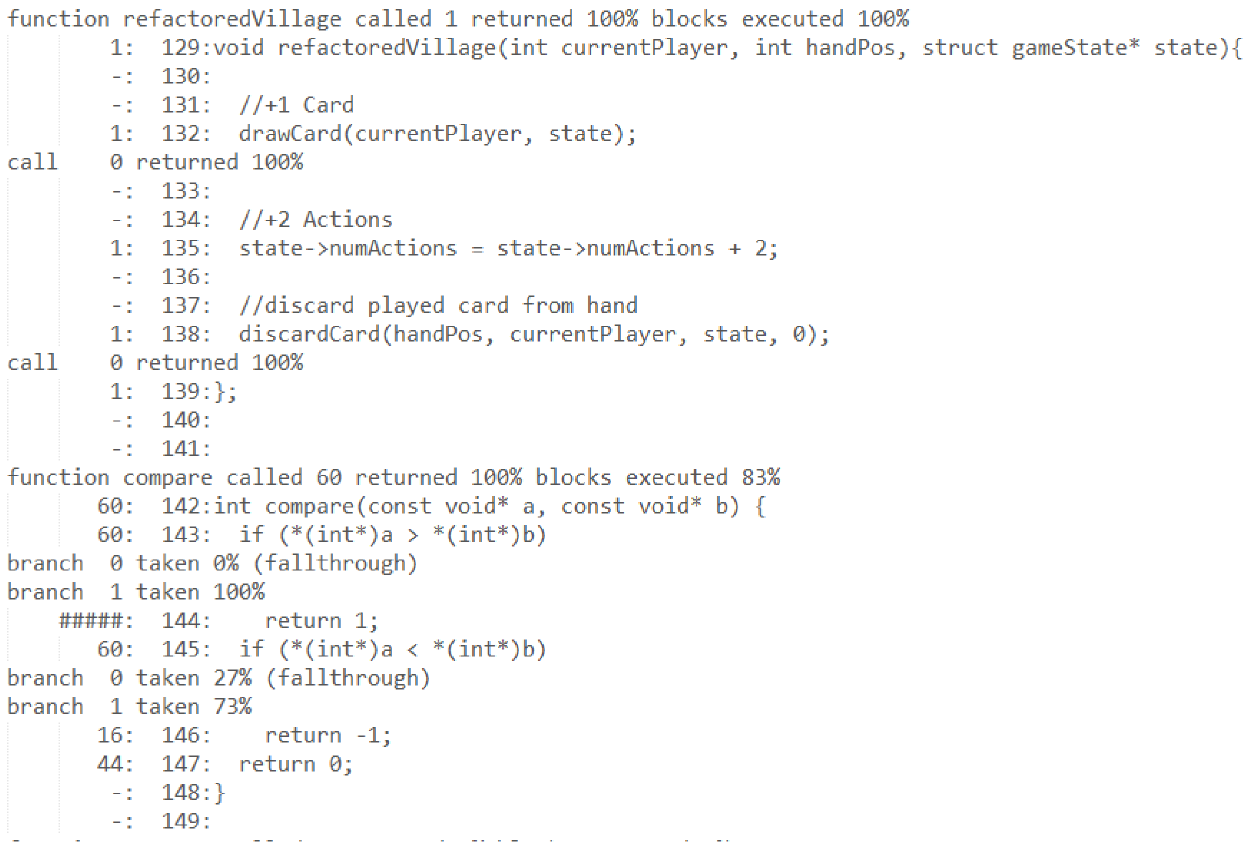
```
function refactoredSmithy called 1 returned 100% blocks executed 100%
        1:   38:void refactoredSmithy(int currentPlayer,struct gameState* state, int handPos){
        -:   39:
        1:   40:  int i = 0;
        -:   41:  //+3 Cards
        5:   42:  for (i = 0; i <= 3; i++) {
branch  0 taken 80%
branch  1 taken 20% (fallthrough)
        4:   43:    drawCard(currentPlayer, state);
call    0 returned 100%
        -:   44:  }
        -:   45:
        -:   46:  //discard card from hand
        1:   47:  discardCard(handPos, currentPlayer, state, 1);
call    0 returned 100%
        -:   48:
        -:   49:
        1:   50:};
        -:   51:
```

Adventurer Card Test Coverage – 100% Branch (cardtest2)



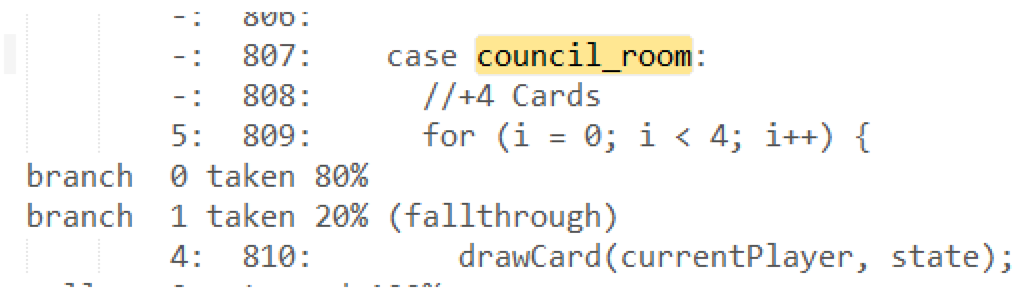
```
function refactoredAdventurer called 1 returned 100% blocks executed 100%
        1:   52:void refactoredAdventurer(int drawntreasure, int currentPlayer, struct gameState* state){
        -:   53:
        -:   54:  int cardDrawn;
        -:   55:  int temphand[MAX_HAND];
        1:   56:  int z = 0;
        -:   57:
       10:   58:  while(drawntreasure < 2){
branch  0 taken 89%
branch  1 taken 11% (fallthrough)
        8:   59:    if (state->deckCount[currentPlayer] < 1){//if the deck is empty we need to shuffle discar
branch  0 taken 38% (fallthrough)
branch  1 taken 63%
        3:   60:      shuffle(currentPlayer, state);
call    0 returned 100%
        -:   61:    }
        8:   62:    drawCard(currentPlayer, state);
call    0 returned 100%
        8:   63:    cardDrawn = state->hand[currentPlayer][state->handCount[currentPlayer]];//top card of han
        8:   64:    if (cardDrawn == copper || cardDrawn == silver || cardDrawn == gold)
branch  0 taken 75% (fallthrough)
branch  1 taken 25%
branch  2 taken 100% (fallthrough)
branch  3 taken 0%
branch  4 taken 0% (fallthrough)
branch  5 taken 100%
        2:   65:      drawntreasure++;
        -:   66:    else{
        6:   67:      temphand[z]=cardDrawn;
        6:   68:      state->handCount[currentPlayer]--; //this should just remove the top card (the most rec
        6:   69:      z++;
        -:   70:    }
        -:   71:  }
        7:   72:  while(z-1>0){
branch  0 taken 83%
branch  1 taken 17% (fallthrough)
        5:   73:    state->discard[currentPlayer][state->discardCount[currentPlayer]++]=temphand[z-1]; // dis
        5:   74:    z=z-1;
        -:   75:  }
        1:   76:};
        -:   77:
```

Village Card Test Coverage – 100% Branch (cardtest3)



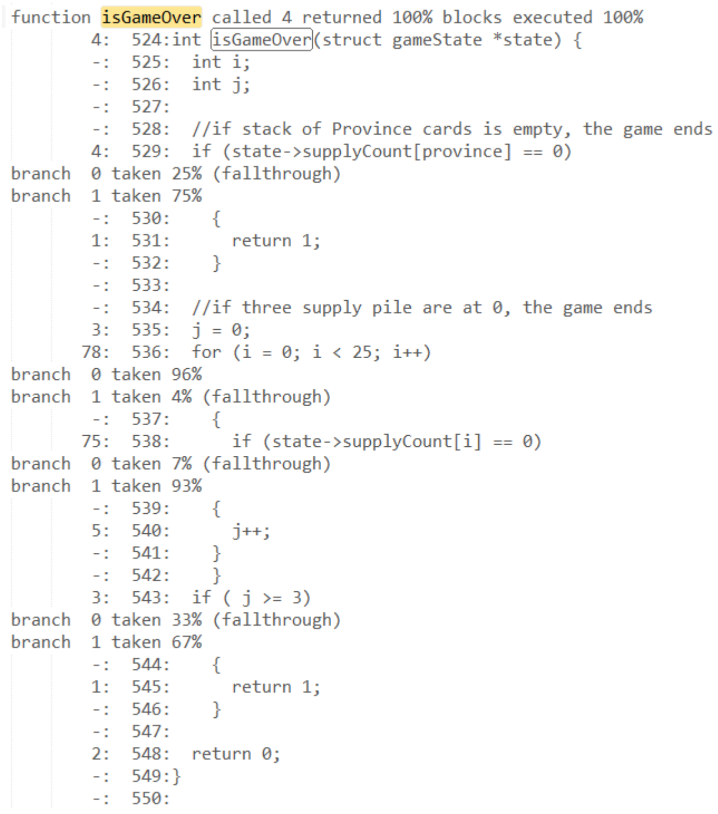
```
function refactoredVillage called 1 returned 100% blocks executed 100%
        1:  129:void refactoredVillage(int currentPlayer, int handPos, struct gameState* state){
        -:  130:
        -:  131:  //+1 Card
        1:  132:  drawCard(currentPlayer, state);
call    0 returned 100%
        -:  133:
        -:  134:  //+2 Actions
        1:  135:  state->numActions = state->numActions + 2;
        -:  136:
        -:  137:  //discard played card from hand
        1:  138:  discardCard(handPos, currentPlayer, state, 0);
call    0 returned 100%
        1:  139:};
        -:  140:
        -:  141:
function compare called 60 returned 100% blocks executed 83%
       60:  142:int compare(const void* a, const void* b) {
       60:  143:  if (*(int*)a > *(int*)b)
branch  0 taken 0% (fallthrough)
branch  1 taken 100%
    #####:  144:    return 1;
       60:  145:  if (*(int*)a < *(int*)b)
branch  0 taken 27% (fallthrough)
branch  1 taken 73%
       16:  146:    return -1;
       44:  147:  return 0;
        -:  148:}
        -:  149:
```

Council Room Card Test Coverage – 100% Branch (cardtest4)



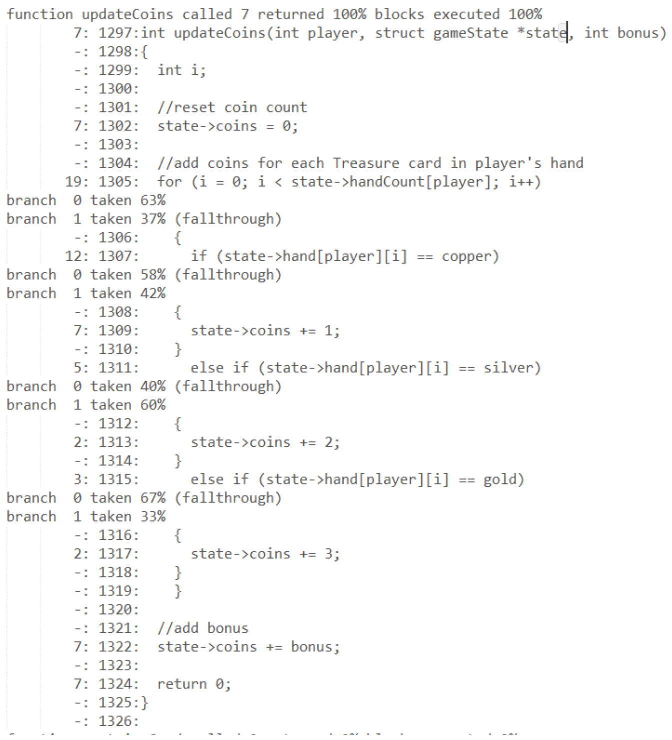
```
        -:  806:
        -:  807:    case council_room:
        -:  808:      //+4 Cards
        5:  809:      for (i = 0; i < 4; i++) {
branch  0 taken 80%
branch  1 taken 20% (fallthrough)
        4:  810:        drawCard(currentPlayer, state);
```

isGameOver Function Unit Test Coverage – 100% Branch (unittest1)



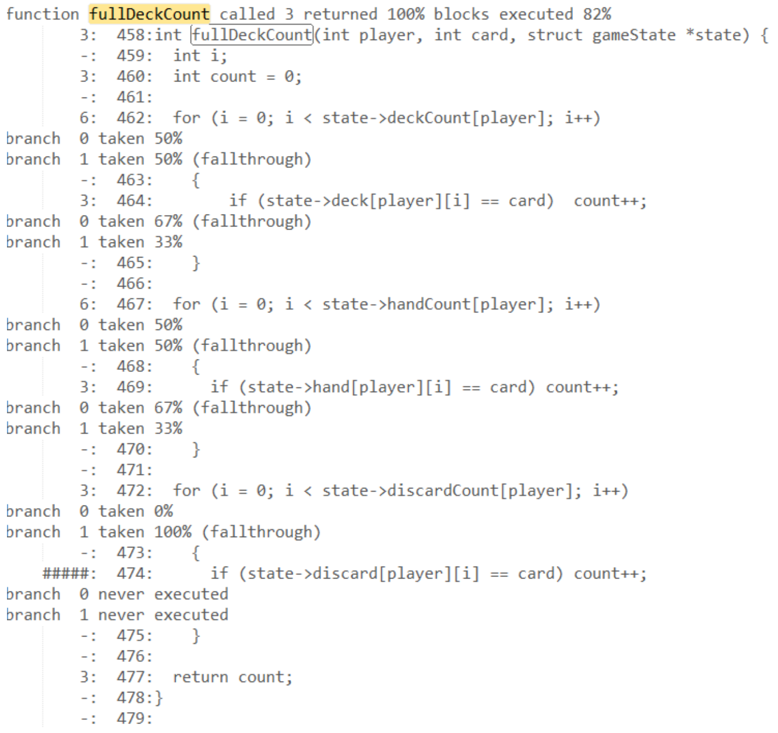
```
function isGameOver called 4 returned 100% blocks executed 100%
        4:  524:int isGameOver(struct gameState *state) {
        -:  525:  int i;
        -:  526:  int j;
        -:  527:
        -:  528:  //if stack of Province cards is empty, the game ends
        4:  529:  if (state->supplyCount[province] == 0)
branch  0 taken 25% (fallthrough)
branch  1 taken 75%
        -:  530:    {
        1:  531:      return 1;
        -:  532:    }
        -:  533:
        -:  534:  //if three supply pile are at 0, the game ends
        3:  535:  j = 0;
       78:  536:  for (i = 0; i < 25; i++)
branch  0 taken 96%
branch  1 taken 4% (fallthrough)
        -:  537:    {
       75:  538:      if (state->supplyCount[i] == 0)
branch  0 taken 7% (fallthrough)
branch  1 taken 93%
        -:  539:    {
        5:  540:      j++;
        -:  541:    }
        -:  542:    }
        3:  543:  if ( j >= 3)
branch  0 taken 33% (fallthrough)
branch  1 taken 67%
        -:  544:    {
        1:  545:      return 1;
        -:  546:    }
        -:  547:
        2:  548:  return 0;
        -:  549:}
        -:  550:
```

updateCoins Function Unit Test Coverage – 100% Branch (unittest2)



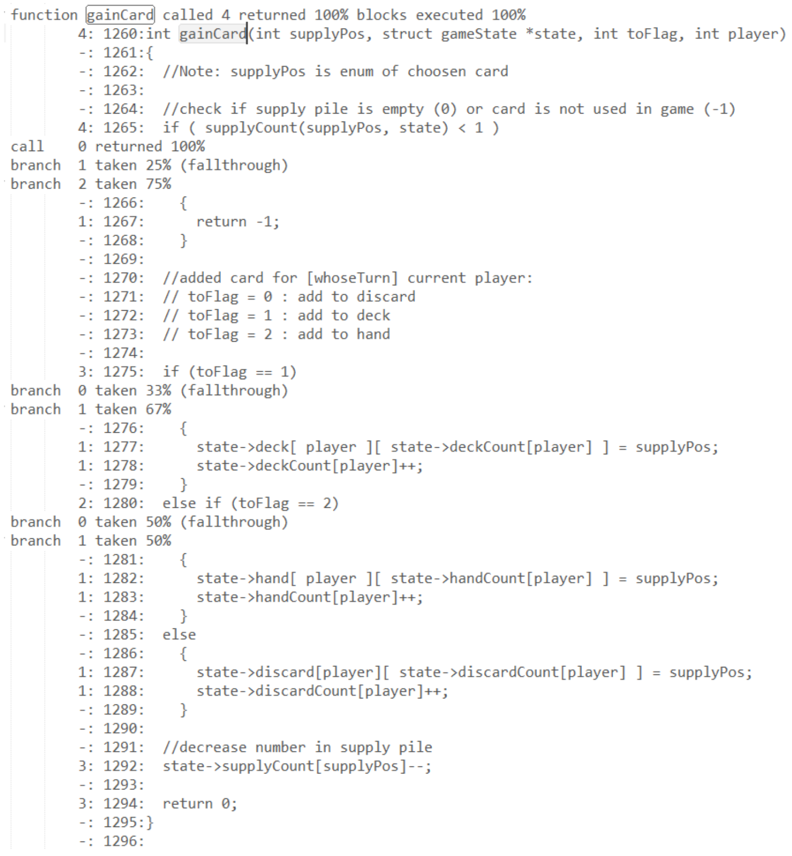
```
function updateCoins called 7 returned 100% blocks executed 100%
        7: 1297:int updateCoins(int player, struct gameState *state, int bonus)
        -: 1298:{
        -: 1299:  int i;
        -: 1300:
        -: 1301:  //reset coin count
        7: 1302:  state->coins = 0;
        -: 1303:
        -: 1304:  //add coins for each Treasure card in player's hand
       19: 1305:  for (i = 0; i < state->handCount[player]; i++)
branch  0 taken 63%
branch  1 taken 37% (fallthrough)
        -: 1306:    {
       12: 1307:      if (state->hand[player][i] == copper)
branch  0 taken 58% (fallthrough)
branch  1 taken 42%
        -: 1308:    {
        7: 1309:      state->coins += 1;
        -: 1310:    }
        5: 1311:      else if (state->hand[player][i] == silver)
branch  0 taken 40% (fallthrough)
branch  1 taken 60%
        -: 1312:    {
        2: 1313:      state->coins += 2;
        -: 1314:    }
        3: 1315:      else if (state->hand[player][i] == gold)
branch  0 taken 67% (fallthrough)
branch  1 taken 33%
        -: 1316:    {
        2: 1317:      state->coins += 3;
        -: 1318:    }
        -: 1319:    }
        -: 1320:
        -: 1321:  //add bonus
        7: 1322:  state->coins += bonus;
        -: 1323:
        7: 1324:  return 0;
        -: 1325:}
        -: 1326:
```

fullDeckCount Function Unit Test Coverage – 100% Branch (unittest3)



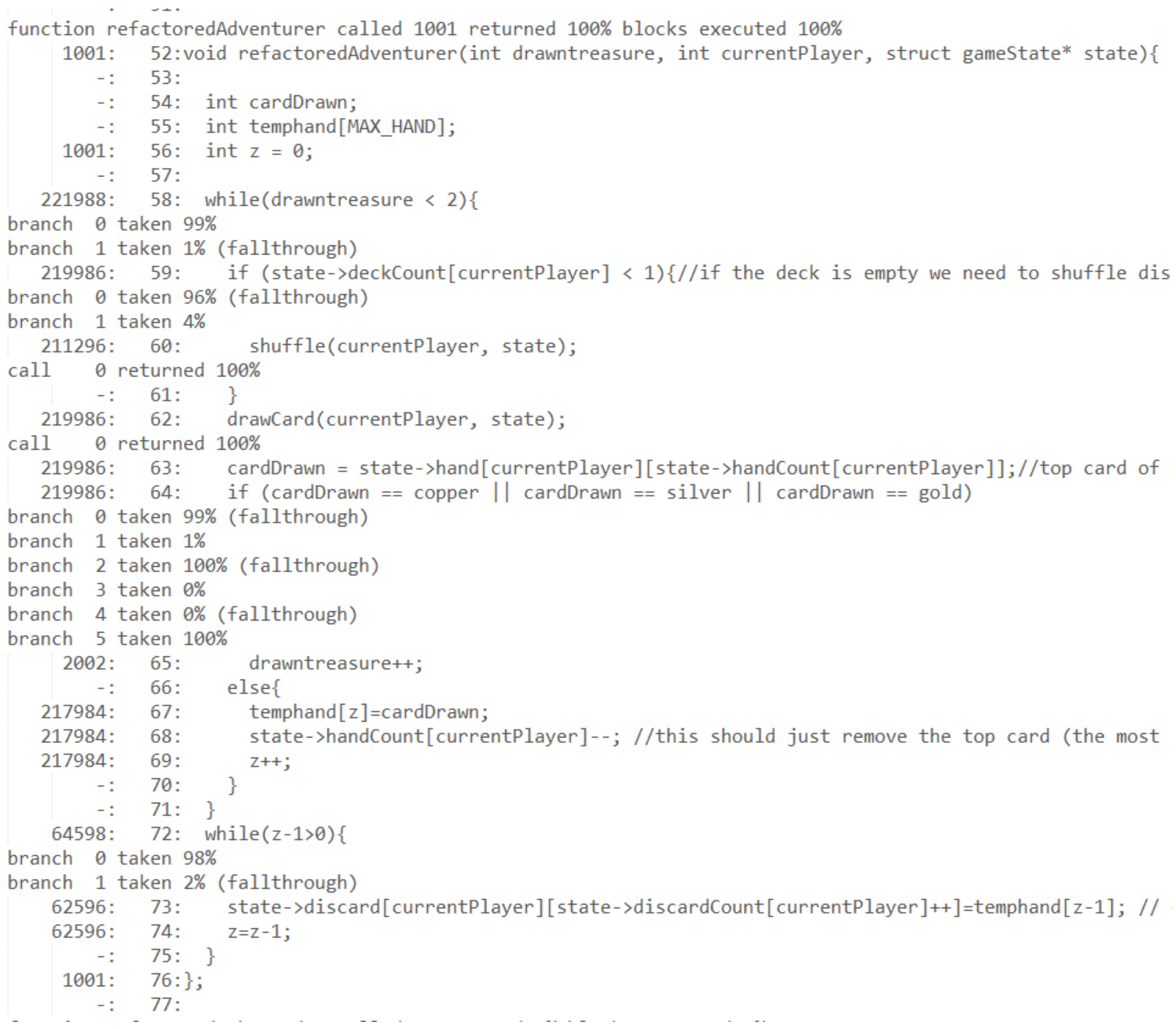
```
function fullDeckCount called 3 returned 100% blocks executed 82%
        3:  458:int fullDeckCount(int player, int card, struct gameState *state) {
        -:  459:  int i;
        3:  460:  int count = 0;
        -:  461:
        6:  462:  for (i = 0; i < state->deckCount[player]; i++)
branch  0 taken 50%
branch  1 taken 50% (fallthrough)
        -:  463:    {
        3:  464:        if (state->deck[player][i] == card)  count++;
branch  0 taken 67% (fallthrough)
branch  1 taken 33%
        -:  465:    }
        -:  466:
        6:  467:  for (i = 0; i < state->handCount[player]; i++)
branch  0 taken 50%
branch  1 taken 50% (fallthrough)
        -:  468:    {
        3:  469:      if (state->hand[player][i] == card) count++;
branch  0 taken 67% (fallthrough)
branch  1 taken 33%
        -:  470:    }
        -:  471:
        3:  472:  for (i = 0; i < state->discardCount[player]; i++)
branch  0 taken 0%
branch  1 taken 100% (fallthrough)
        -:  473:    {
    #####:  474:      if (state->discard[player][i] == card) count++;
branch  0 never executed
branch  1 never executed
        -:  475:    }
        -:  476:
        3:  477:  return count;
        -:  478:}
        -:  479:
```

gainCard Function Unit Test Coverage – 100% Branch (unittest4)



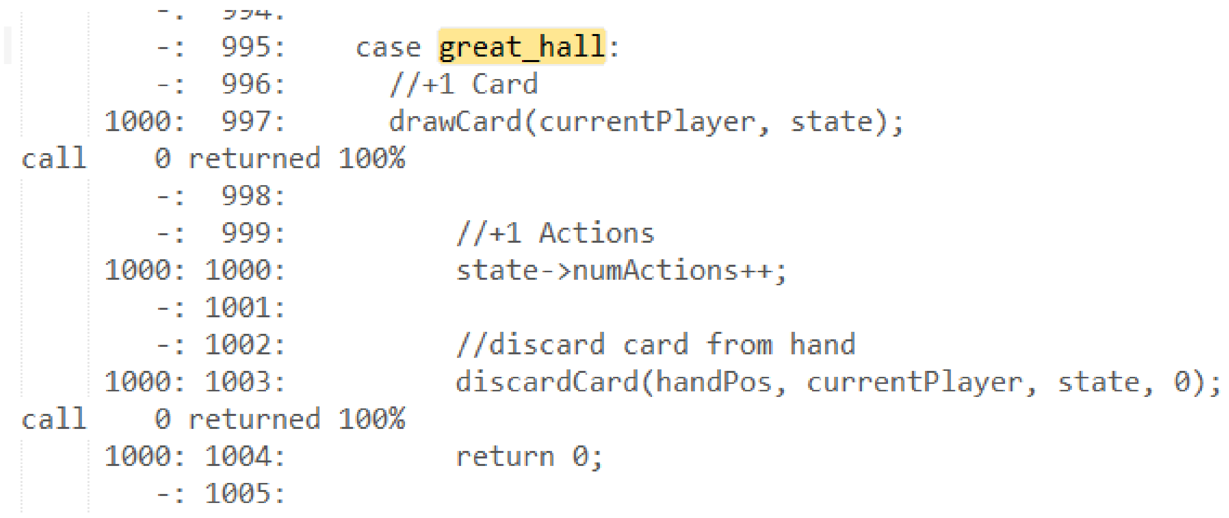
```
function gainCard called 4 returned 100% blocks executed 100%
        4: 1260:int gainCard(int supplyPos, struct gameState *state, int toFlag, int player)
        -: 1261:{
        -: 1262:  //Note: supplyPos is enum of choosen card
        -: 1263:
        -: 1264:  //check if supply pile is empty (0) or card is not used in game (-1)
        4: 1265:  if ( supplyCount(supplyPos, state) < 1 )
call    0 returned 100%
branch  1 taken 25% (fallthrough)
branch  2 taken 75%
        -: 1266:    {
        1: 1267:      return -1;
        -: 1268:    }
        -: 1269:
        -: 1270:  //added card for [whoseTurn] current player:
        -: 1271:  // toFlag = 0 : add to discard
        -: 1272:  // toFlag = 1 : add to deck
        -: 1273:  // toFlag = 2 : add to hand
        -: 1274:
        3: 1275:  if (toFlag == 1)
branch  0 taken 33% (fallthrough)
branch  1 taken 67%
        -: 1276:    {
        1: 1277:      state->deck[ player ][ state->deckCount[player] ] = supplyPos;
        1: 1278:      state->deckCount[player]++;
        -: 1279:    }
        2: 1280:  else if (toFlag == 2)
branch  0 taken 50% (fallthrough)
branch  1 taken 50%
        -: 1281:    {
        1: 1282:      state->hand[ player ][ state->handCount[player] ] = supplyPos;
        1: 1283:      state->handCount[player]++;
        -: 1284:    }
        -: 1285:  else
        -: 1286:    {
        1: 1287:      state->discard[player][ state->discardCount[player] ] = supplyPos;
        1: 1288:      state->discardCount[player]++;
        -: 1289:    }
        -: 1290:
        -: 1291:  //decrease number in supply pile
        3: 1292:  state->supplyCount[supplyPos]--;
        -: 1293:
        3: 1294:  return 0;
        -: 1295:}
        -: 1296:
```

Adventurer Random Test Coverage – 100% Branch (randomtestadventurer)



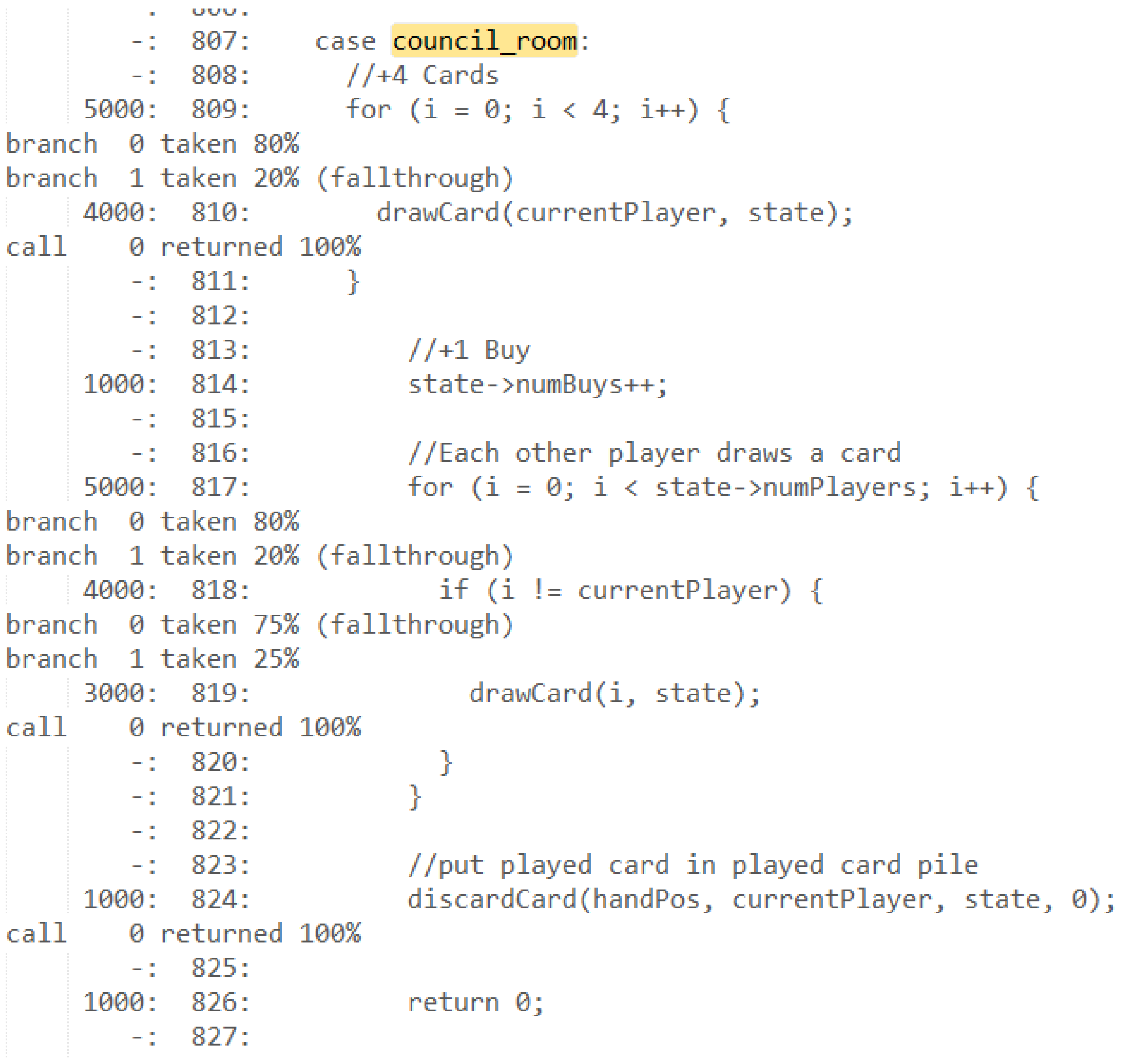
```
function refactoredAdventurer called 1001 returned 100% blocks executed 100%
     1001:   52:void refactoredAdventurer(int drawntreasure, int currentPlayer, struct gameState* state){
        -:   53:
        -:   54:  int cardDrawn;
        -:   55:  int temphand[MAX_HAND];
     1001:   56:  int z = 0;
        -:   57:
   221988:   58:  while(drawntreasure < 2){
branch  0 taken 99%
branch  1 taken 1% (fallthrough)
   219986:   59:    if (state->deckCount[currentPlayer] < 1){//if the deck is empty we need to shuffle dis
branch  0 taken 96% (fallthrough)
branch  1 taken 4%
   211296:   60:      shuffle(currentPlayer, state);
call    0 returned 100%
        -:   61:    }
   219986:   62:    drawCard(currentPlayer, state);
call    0 returned 100%
   219986:   63:    cardDrawn = state->hand[currentPlayer][state->handCount[currentPlayer]];//top card of
   219986:   64:    if (cardDrawn == copper || cardDrawn == silver || cardDrawn == gold)
branch  0 taken 99% (fallthrough)
branch  1 taken 1%
branch  2 taken 100% (fallthrough)
branch  3 taken 0%
branch  4 taken 0% (fallthrough)
branch  5 taken 100%
     2002:   65:      drawntreasure++;
        -:   66:    else{
   217984:   67:      temphand[z]=cardDrawn;
   217984:   68:      state->handCount[currentPlayer]--; //this should just remove the top card (the most
   217984:   69:      z++;
        -:   70:    }
        -:   71:  }
    64598:   72:  while(z-1>0){
branch  0 taken 98%
branch  1 taken 2% (fallthrough)
    62596:   73:    state->discard[currentPlayer][state->discardCount[currentPlayer]++]=temphand[z-1]; //
    62596:   74:    z=z-1;
        -:   75:  }
     1001:   76:};
        -:   77:
```

Great Hall Random Test Coverage – 100% Branch (randomtestcard1)



```
        -:  995:    case great_hall:
        -:  996:      //+1 Card
     1000:  997:      drawCard(currentPlayer, state);
call    0 returned 100%
        -:  998:
        -:  999:          //+1 Actions
     1000: 1000:          state->numActions++;
        -: 1001:
        -: 1002:          //discard card from hand
     1000: 1003:          discardCard(handPos, currentPlayer, state, 0);
call    0 returned 100%
     1000: 1004:          return 0;
        -: 1005:
```

Council Room Random Test Coverage – 100% Branch (randomtestcard2)



```
        -:  807:    case council_room:
        -:  808:      //+4 Cards
     5000:  809:      for (i = 0; i < 4; i++) {
branch  0 taken 80%
branch  1 taken 20% (fallthrough)
     4000:  810:        drawCard(currentPlayer, state);
call    0 returned 100%
        -:  811:      }
        -:  812:
        -:  813:          //+1 Buy
     1000:  814:          state->numBuys++;
        -:  815:
        -:  816:          //Each other player draws a card
     5000:  817:          for (i = 0; i < state->numPlayers; i++) {
branch  0 taken 80%
branch  1 taken 20% (fallthrough)
     4000:  818:            if (i != currentPlayer) {
branch  0 taken 75% (fallthrough)
branch  1 taken 25%
     3000:  819:              drawCard(i, state);
call    0 returned 100%
        -:  820:            }
        -:  821:          }
        -:  822:
        -:  823:          //put played card in played card pile
     1000:  824:          discardCard(handPos, currentPlayer, state, 0);
call    0 returned 100%
        -:  825:
     1000:  826:          return 0;
        -:  827:
```

I believe that Jonathan's Dominion code is reliable in the sense that it should not cause a seg fault or crash the game which is desirable. However, as far as reliability in terms of the creating a correct game state, his code would not produce a correct end game for the players due to the incorrect hand and deck counts that his code creates. His bugs would need to be fixed before we can say for certain (as much as possible) that the game will function correctly for the users.

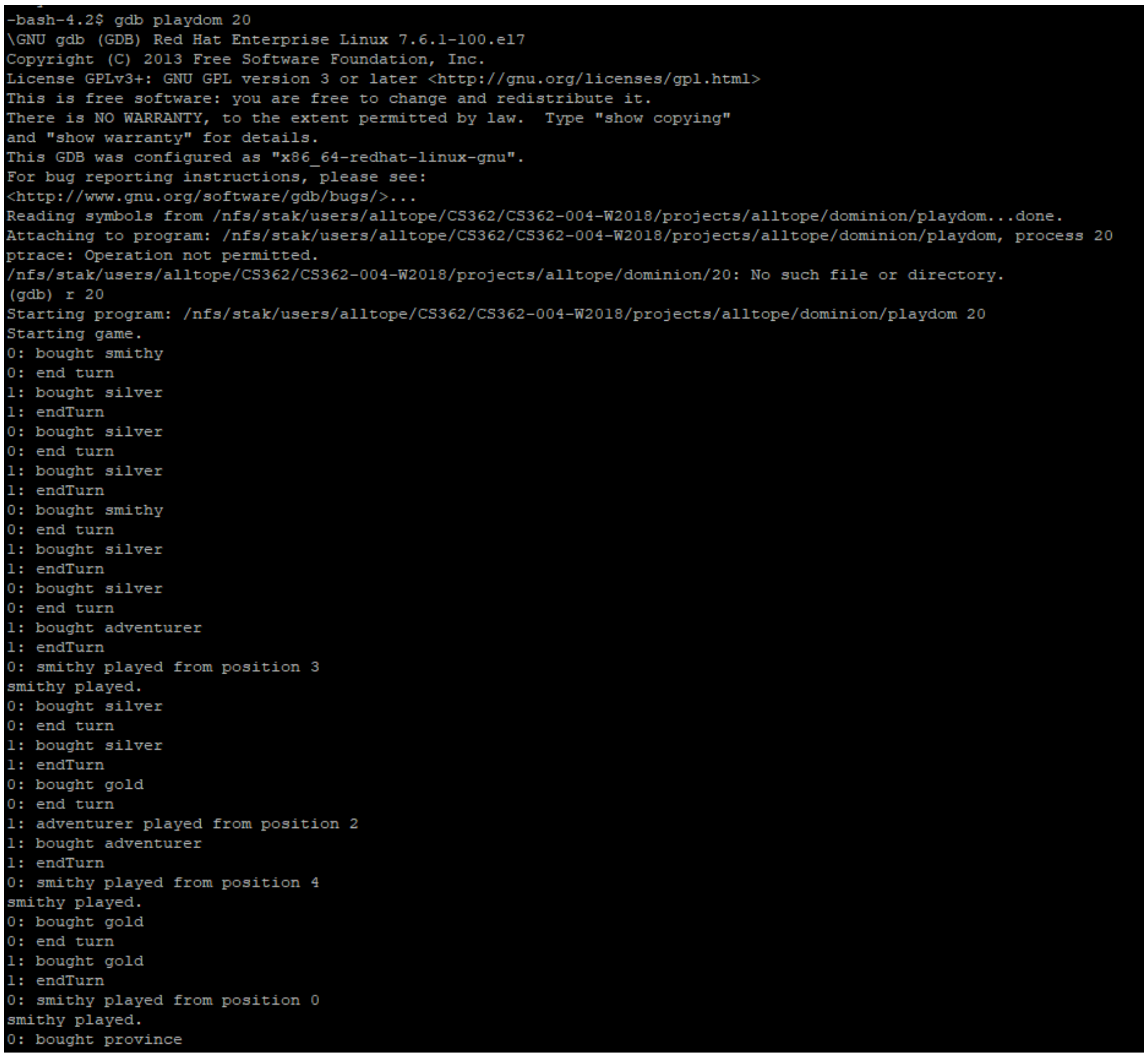
Debugging

Bug report from my own code:

```
=============================

Title: Other player hand count decreased when playing Smithy Card
Class: Serious Bug
Date: 2-24-18
Reported By: Erin Alltop
Email: alltope@oregonstate.edu
Product: Dominion              Version: alltope refactored code (from Assignment 2)
Platform: Linux
Browser:  Flip School Server
URL: N/A
Is it reproducible: Yes

Description
===========
Summary: Hand count of the next player is decreased by one when playing the Smithy Card
I created an instance of the Dominion game and called the Smithy Card function.
I then ran test code to check various counts. I found that the hand count of the next player was
decreased by 1 incorrectly.
This error happens every time the Smithy Card is run

Steps to Produce/Reproduce
--------------------------
Any time the Smithy Card function is called this bug occurs

Expected Results
----------------
Hand counts of other player is unchanged

Actual Results
--------------
Hand count of other player is decreased by 1

Workarounds
-----------
No known workaround at this time

Attachments
-----------
Test code for relevant bug:
```

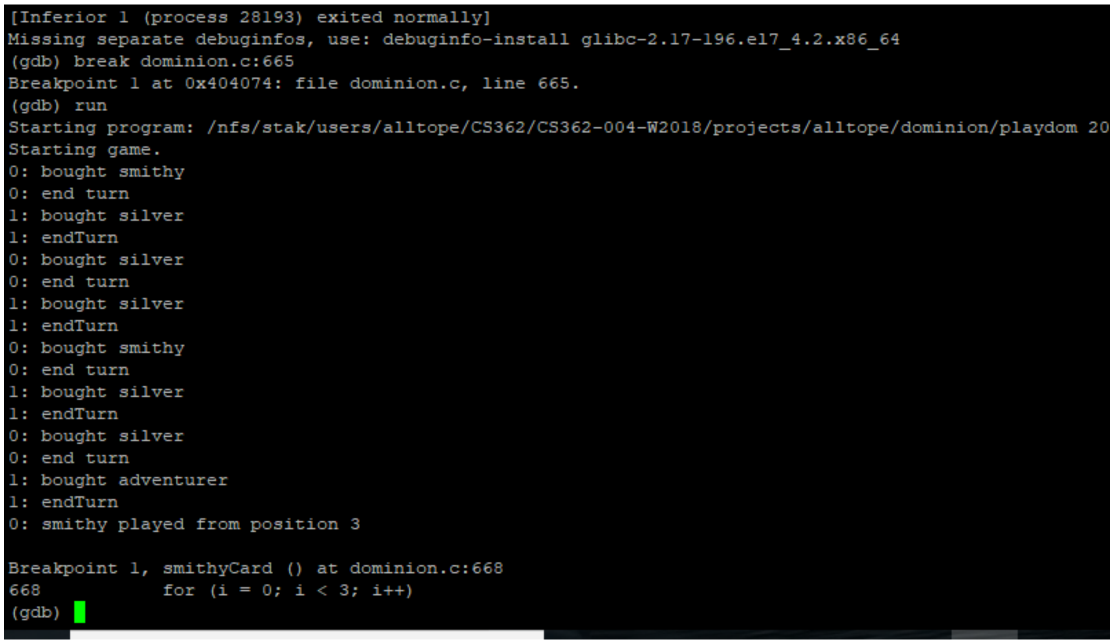
TEST 4: No state change in other players
Hand count expected for other player 1:
Count = -1, expected = 0
TEST 4 FAILED! HAND COUNT CHANGED

To inspect this further, I made a "playdom" state of the game and used the gdb debugger to step through the code. The first thing I did was run an instance of playdom within gdb.



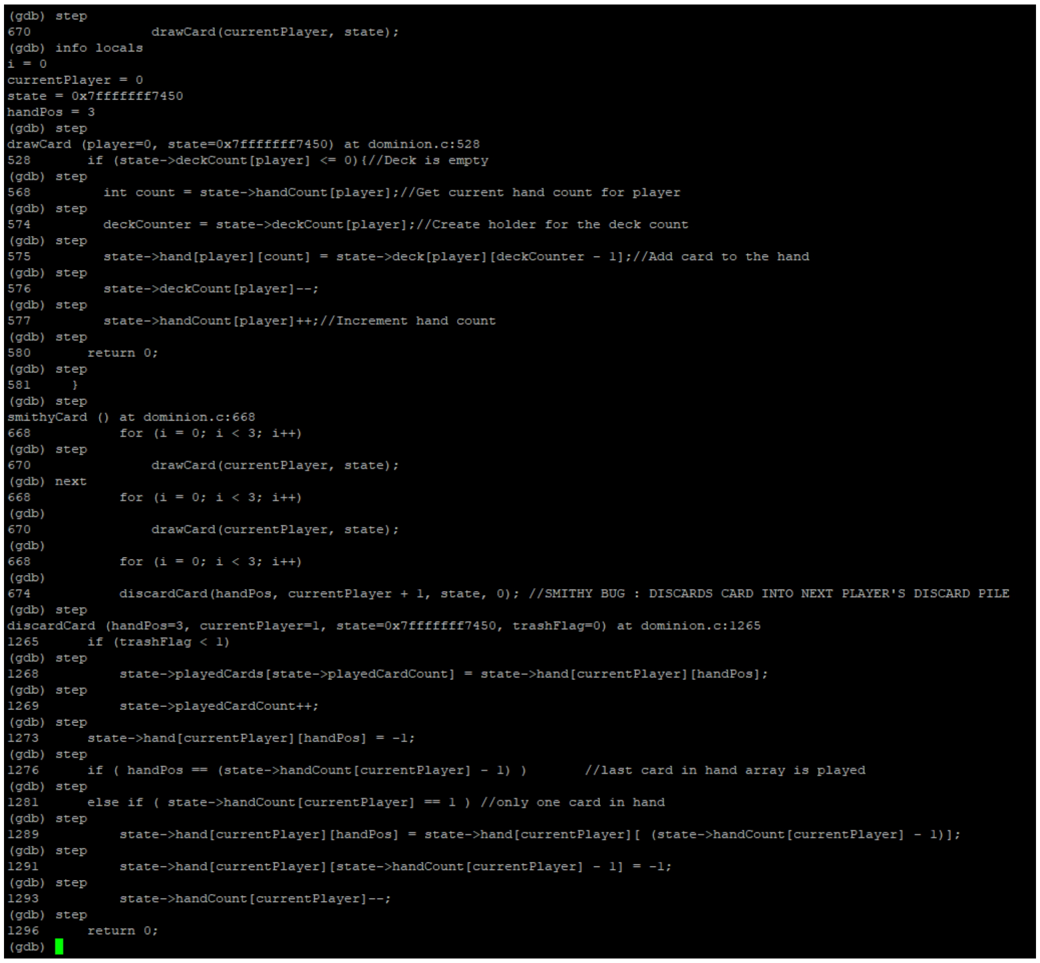
```
-bash-4.2$ gdb playdom 20
\GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-100.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /nfs/stak/users/alltope/CS362/CS362-004-W2018/projects/alltope/dominion/playdom...done.
Attaching to program: /nfs/stak/users/alltope/CS362/CS362-004-W2018/projects/alltope/dominion/playdom, process 20
ptrace: Operation not permitted.
/nfs/stak/users/alltope/CS362/CS362-004-W2018/projects/alltope/dominion/20: No such file or directory.
(gdb) r 20
Starting program: /nfs/stak/users/alltope/CS362/CS362-004-W2018/projects/alltope/dominion/playdom 20
Starting game.
0: bought smithy
0: end turn
1: bought silver
1: endTurn
0: bought silver
0: end turn
1: bought silver
1: endTurn
0: bought smithy
0: end turn
1: bought silver
1: endTurn
0: bought silver
0: end turn
1: bought adventurer
1: endTurn
0: smithy played from position 3
smithy played.
0: bought silver
0: end turn
1: bought silver
1: endTurn
0: bought gold
0: end turn
1: adventurer played from position 2
1: bought adventurer
1: endTurn
0: smithy played from position 4
smithy played.
0: bought gold
0: end turn
1: bought gold
1: endTurn
0: smithy played from position 0
smithy played.
0: bought province
```

This exited normally, but I know there is something amiss in the smithy card, so I set a break point at the smithy card to then step through the code.



```
[Inferior 1 (process 28193) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-196.el7_4.2.x86_64
(gdb) break dominion.c:665
Breakpoint 1 at 0x404074: file dominion.c, line 665.
(gdb) run
Starting program: /nfs/stak/users/alltope/CS362/CS362-004-W2018/projects/alltope/dominion/playdom 20
Starting game.
0: bought smithy
0: end turn
1: bought silver
1: endTurn
0: bought silver
0: end turn
1: bought silver
1: endTurn
0: bought smithy
0: end turn
1: bought silver
1: endTurn
0: bought silver
0: end turn
1: bought adventurer
1: endTurn
0: smithy played from position 3

Breakpoint 1, smithyCard () at dominion.c:668
668             for (i = 0; i < 3; i++)
(gdb)
```

Stepping through the code I can see on line 674 that the wrong player is discarding a card. It discarding currentPlayer + 1’s card instead of the currentPlayer!



```
(gdb) step
670                 drawCard(currentPlayer, state);
(gdb) info locals
i = 0
currentPlayer = 0
state = 0x7ffffff7450
handPos = 3
(gdb) step
drawCard (player=0, state=0x7ffffff7450) at dominion.c:528
528       if (state->deckCount[player] <= 0){//Deck is empty
(gdb) step
568         int count = state->handCount[player];//Get current hand count for player
(gdb) step
574         deckCounter = state->deckCount[player];//Create holder for the deck count
(gdb) step
575         state->hand[player][count] = state->deck[player][deckCounter - 1];//Add card to the hand
(gdb) step
576         state->deckCount[player]--;
(gdb) step
577         state->handCount[player]++;//Increment hand count
(gdb) step
580       return 0;
(gdb) step
581     }
(gdb) step
smithyCard () at dominion.c:668
668           for (i = 0; i < 3; i++)
(gdb) step
670               drawCard(currentPlayer, state);
(gdb) next
668           for (i = 0; i < 3; i++)
(gdb)
670               drawCard(currentPlayer, state);
(gdb)
668           for (i = 0; i < 3; i++)
(gdb)
674           discardCard(handPos, currentPlayer + 1, state, 0); //SMITHY BUG : DISCARDS CARD INTO NEXT PLAYER'S DISCARD PILE
(gdb) step
discardCard (handPos=3, currentPlayer=1, state=0x7ffffff7450, trashFlag=0) at dominion.c:1265
1265      if (trashFlag < 1)
(gdb) step
1268          state->playedCards[state->playedCardCount] = state->hand[currentPlayer][handPos];
(gdb) step
1269          state->playedCardCount++;
(gdb) step
1273      state->hand[currentPlayer][handPos] = -1;
(gdb) step
1276      if ( handPos == (state->handCount[currentPlayer] - 1) )       //last card in hand array is played
(gdb) step
1281      else if ( state->handCount[currentPlayer] == 1 ) //only one card in hand
(gdb) step
1289          state->hand[currentPlayer][handPos] = state->hand[currentPlayer][ (state->handCount[currentPlayer] - 1)];
(gdb) step
1291          state->hand[currentPlayer][state->handCount[currentPlayer] - 1] = -1;
(gdb) step
1293          state->handCount[currentPlayer]--;
(gdb) step
1296      return 0;
(gdb)
```

After using gdb in this way I can find the bug fairly easily and make the correction in the code, which allows the game to perform as intended (or at least more as intended!). Even though the game compiles, plays, and exits normally, this important game-changing bug is able to be found and corrected using the gdb debugger tool.