Brian Laccone
CS 362
5/27/18

Assignment-5

Bug-Reports

Three bugs were found during the testing of my teammate's, Jeff Loung, dominion program. Both the bugs were found in the refractor adventurer and smithy card functions. Adventurer and Smithy were the only refactored cards that our programs had in common. All of the other tests unit and randomized tests were done for cards and functions that were untouched by my teammate.

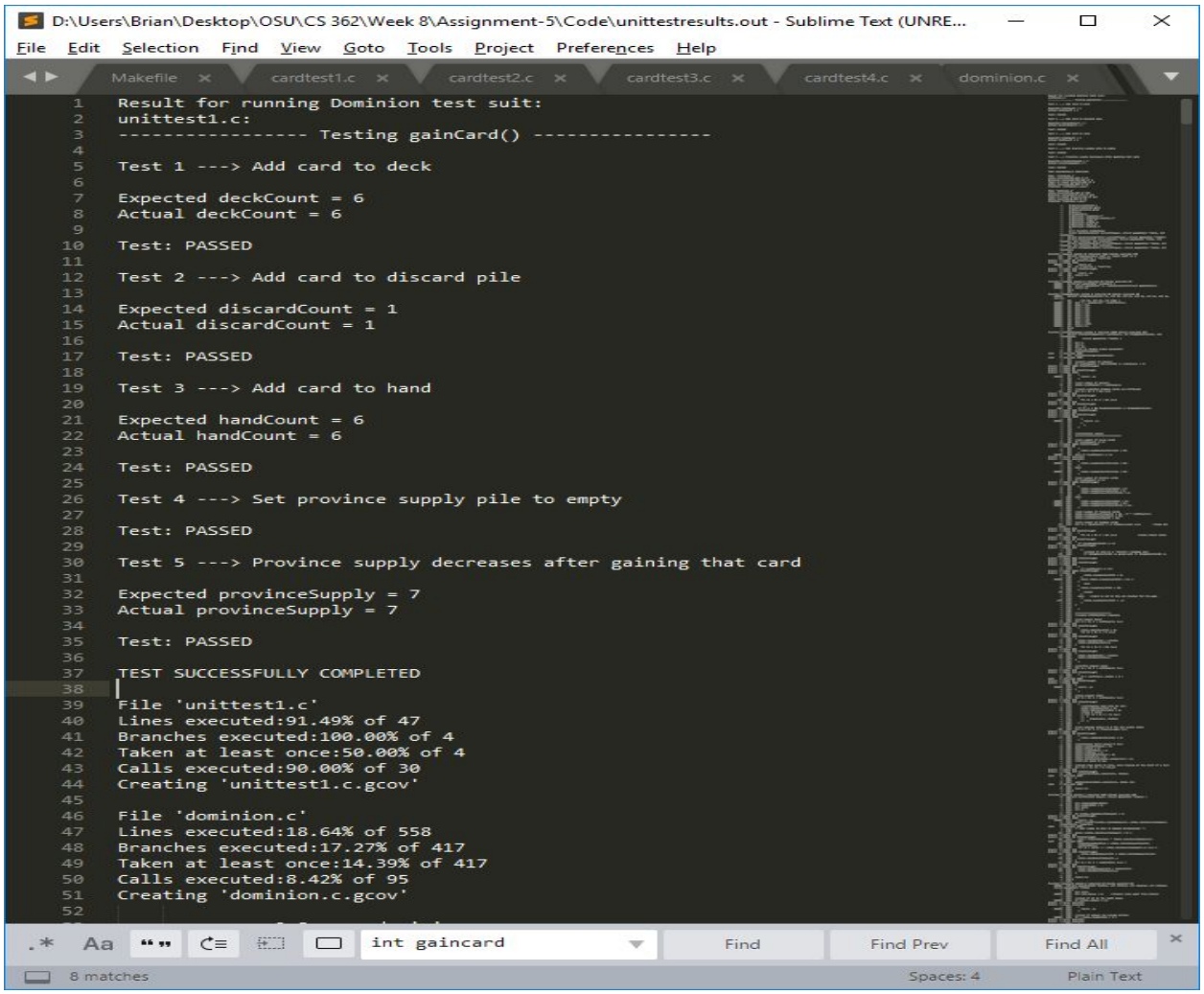
1-2. AdventurerCard() - Both cardtest1.c and randomtestadventurer.c discovered that the hand counts before and after this function was called did not match up. randomtestadventurer.c also discovered that the total card count was wrong as well. The hand count was always 1 less more than it should have been. This suggests that either adventurer was not being discarded, the function was drawing more than two cards, or the way the cards are being counted is wrong. I first discovered the hand count bug when I tested cardtest1.c. The hand count was always 7 instead of 6. I first discovered the total card count bug when I ran the randomtestadventurer.c test. The count was off everytime no matter how many iterations I ran.

3. SmithyCard() - The third bug was discovered in both cardtest2.c and randomtest2.c. These tests discovered that the hand counts were wrong before and after SmithyCard was called. The hand count was always 1 more than what the count should have been. I first discovered this bug during the cardtest2.c and I confirmed it again when the randomtest2.c test ran. The randomtest2.c was about a 99% failure rate. The reason it was not 100 is because of the random values for the deck count. Occasionally the deck would only have three cards in it so the hand count would have been correct then.

Test Report

unittest1.c

My first unit test covered the function gainCard(). The test received 100% statement and branch coverage for this function. Unfortunately my overall coverage of dominion.c was only 18.64% with 17.27% branch coverage. I could have feed the function several different states to try and test what it would do with better defined states instead of just the same state every call to it. I made sure that I hit every branch by feeding the function supply counts that were empty as well as all three “toFlag” branch options. There were no problem with this function as it was not refactored by my teammate and passed all the tests.



D:\Users\Brian\Desktop\OSU\CS 362\Week 4\Assignment 3\unittestresults.out - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

dominion.c | rngs.c | Makefile | unittestresults.out

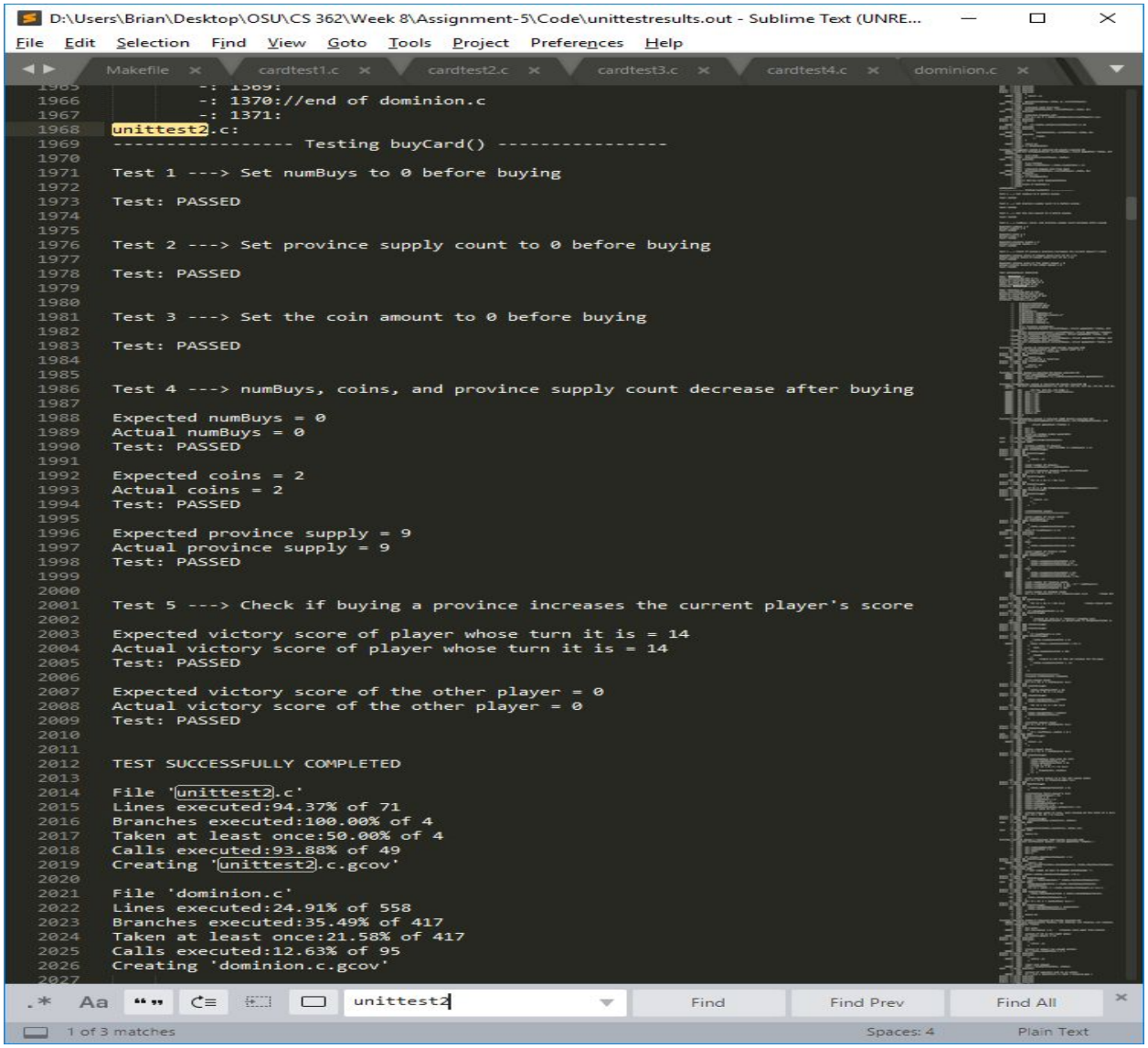
```
        -: 1350:
function gainCard called 5 returned 100% blocks executed 100%
        5: 1351:int gainCard(int supplyPos, struct gameState *state, int toFlag, int player)
        -: 1352:{
        -: 1353:  //Note: supplyPos is enum of choosen card
        -: 1354:
        -: 1355:  //check if supply pile is empty (0) or card is not used in game (-1)
        5: 1356:  if ( supplyCount(supplyPos, state) < 1 )
call    0 returned 100%
branch  1 taken 20% (fallthrough)
branch  2 taken 80%
        -: 1357:    {
        1: 1358:      return -1;
        -: 1359:    }
        -: 1360:
        -: 1361:  //added card for [whoseTurn] current player:
        -: 1362:  // toFlag = 0 : add to discard
        -: 1363:  // toFlag = 1 : add to deck
        -: 1364:  // toFlag = 2 : add to hand
        -: 1365:
        4: 1366:  if (toFlag == 1)
branch  0 taken 25% (fallthrough)
branch  1 taken 75%
        -: 1367:    {
        1: 1368:      state->deck[ player ][ state->deckCount[player] ] = supplyPos;
        1: 1369:      state->deckCount[player]++;
        -: 1370:    }
        3: 1371:  else if (toFlag == 2)
branch  0 taken 33% (fallthrough)
branch  1 taken 67%
        -: 1372:    {
        1: 1373:      state->hand[ player ][ state->handCount[player] ] = supplyPos;
        1: 1374:      state->handCount[player]++;
        -: 1375:    }
        -: 1376:  else
        -: 1377:    {
        2: 1378:      state->discard[player][ state->discardCount[player] ] = supplyPos;
        2: 1379:      state->discardCount[player]++;
        -: 1380:    }
        -: 1381:
        -: 1382:  //decrease number in supply pile
        4: 1383:  state->supplyCount[supplyPos]--;
        -: 1384:
        4: 1385:  return 0;
        -: 1386:}
        -: 1387:
function updateCoins called 1 returned 100% blocks executed 82%
```

.* Aa "" int gainCard Find Find Prev Find All

8 matches Spaces: 4 Plain Text

unittest2.c

My unittest2.c file contained the unit test for the buyCard() function. The file says that I had returned 100% and had 100% blocks executed. I had 100% branch coverage, which I find interesting because I didn't hit any of the DEBUG branches. DEBUG was meant to be a visual representation of the sequence of code in the buyCard() function. I didn't hit the print command in any of the if (DEBUG) {} statements. My guess is that gcov has a built in feature to skip DEBUG branches. My statement coverage of the whole dominion.c file was 24.91% and my branch coverage was 35.49%. There were no problem with this function as it was not refactored by my teammate and passed all the tests.



D:\Users\Brian\Desktop\OSU\CS 362\Week 4\Assignment 3\Code\unittestresults.out - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

dominion.c | unittestresults.out

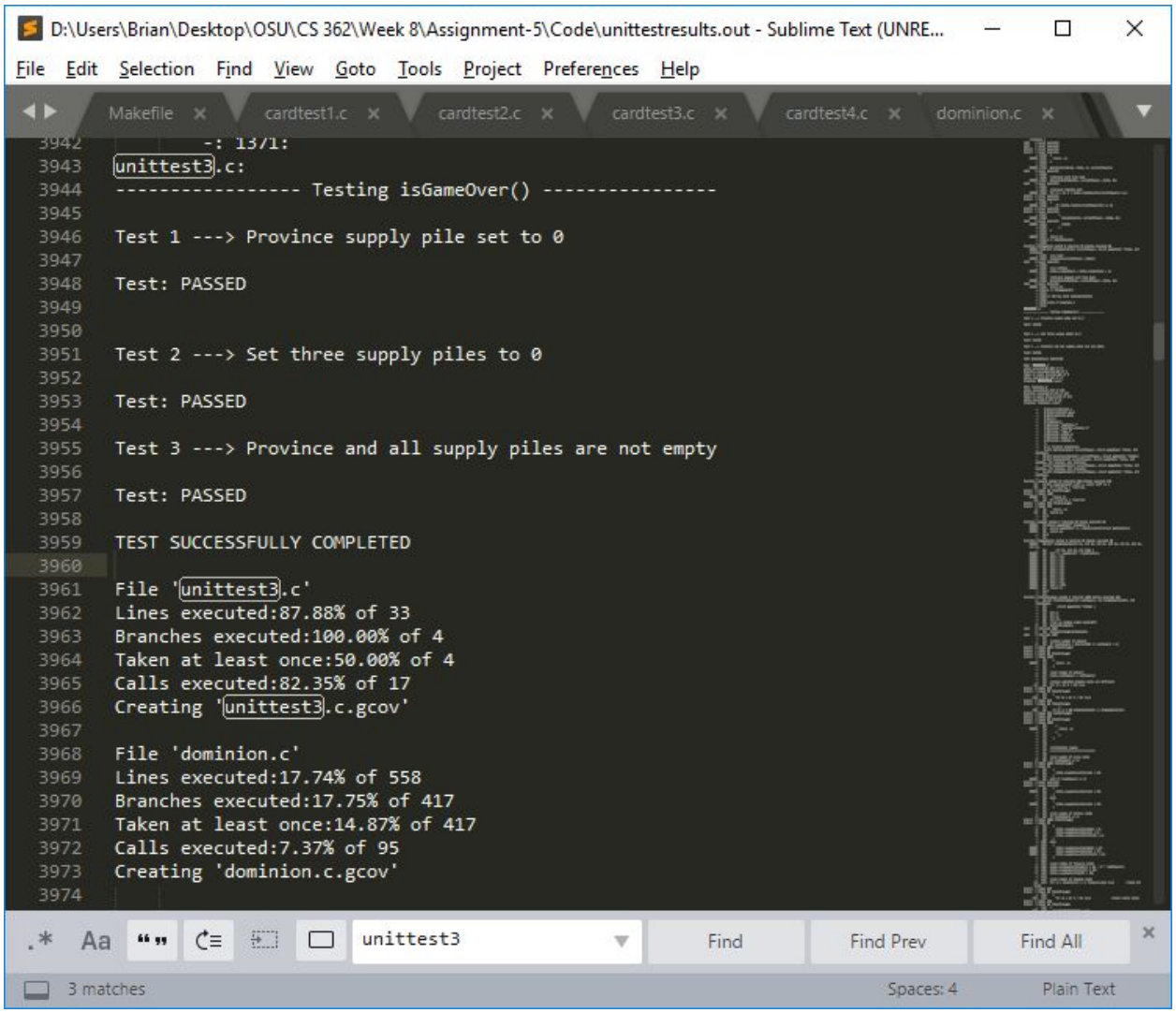
```
        -:  271:
function buyCard called 5 returned 100% blocks executed 100%
        5:  272:int buyCard(int supplyPos, struct gameState *state) {
        -:  273:  int who;
        -:  274:  if (DEBUG){
        -:  275:    printf("Entering buyCard...\n");
        -:  276:  }
        -:  277:
        -:  278:  // I don't know what to do about the phase thing.
        -:  279:
        5:  280:  who = state->whoseTurn;
        -:  281:
        5:  282:  if (state->numBuys < 1){
branch  0 taken 20% (fallthrough)
branch  1 taken 80%
        -:  283:    if (DEBUG)
        -:  284:      printf("You do not have any buys left\n");
        1:  285:    return -1;
        4:  286:  } else if (supplyCount(supplyPos, state) <1){
call    0 returned 100%
branch  1 taken 25% (fallthrough)
branch  2 taken 75%
        -:  287:    if (DEBUG)
        -:  288:      printf("There are not any of that type of card left\n");
        1:  289:    return -1;
        3:  290:  } else if (state->coins < getCost(supplyPos)){
call    0 returned 100%
branch  1 taken 33% (fallthrough)
branch  2 taken 67%
        -:  291:    if (DEBUG)
        -:  292:      printf("You do not have enough money to buy that. You have %d coins.\n",
          state->coins);
        1:  293:    return -1;
        -:  294:  } else {
        2:  295:    state->phase=1;
        -:  296:    //state->supplyCount[supplyPos]--;
        2:  297:    gainCard(supplyPos, state, 0, who); //card goes in discard, this might be
          wrong.. (2 means goes into hand, 0 goes into discard)
call    0 returned 100%
        -:  298:
        2:  299:    state->coins = (state->coins) - (getCost(supplyPos));
call    0 returned 100%
        2:  300:    state->numBuys--;
        -:  301:    if (DEBUG)
        -:  302:      printf("You bought card number %d for %d coins. You now have %d buys and %d
          coins.\n", supplyPos, getCost(supplyPos), state->numBuys, state->coins);
        -:  303:  }
        -:  304:
        -:  305:  //state->discard[who][state->discardCount[who]] = supplyPos;
        -:  306:  //state->discardCount[who]++;
        -:  307:
        2:  308:  return 0;
        -:  309:}
        -:  310:
function numHandCards called 0 returned 0% blocks executed 0%
```

.* Aa int buycard Find Find Prev Find All

8 matches Spaces: 4 Plain Text

unittest3.c

My unittest3.c file contains the tests for the isGameOver() function. This test got 100% statement coverage and 100% branch coverage for the isGameOver() function. Luckily, this was a very simple function and only took me 3 calls of it to cover 100%. My coverage off dominion.c was 17.74% statement and 17.75% branch coverage. There were no problem with this function as it was not refactored by my teammate and passed all the tests.



D:\Users\Brian\Desktop\OSU\CS 362\Week 4\Assignment 3\Code\unittestresults.out - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

dominion.c | unittestresults.out

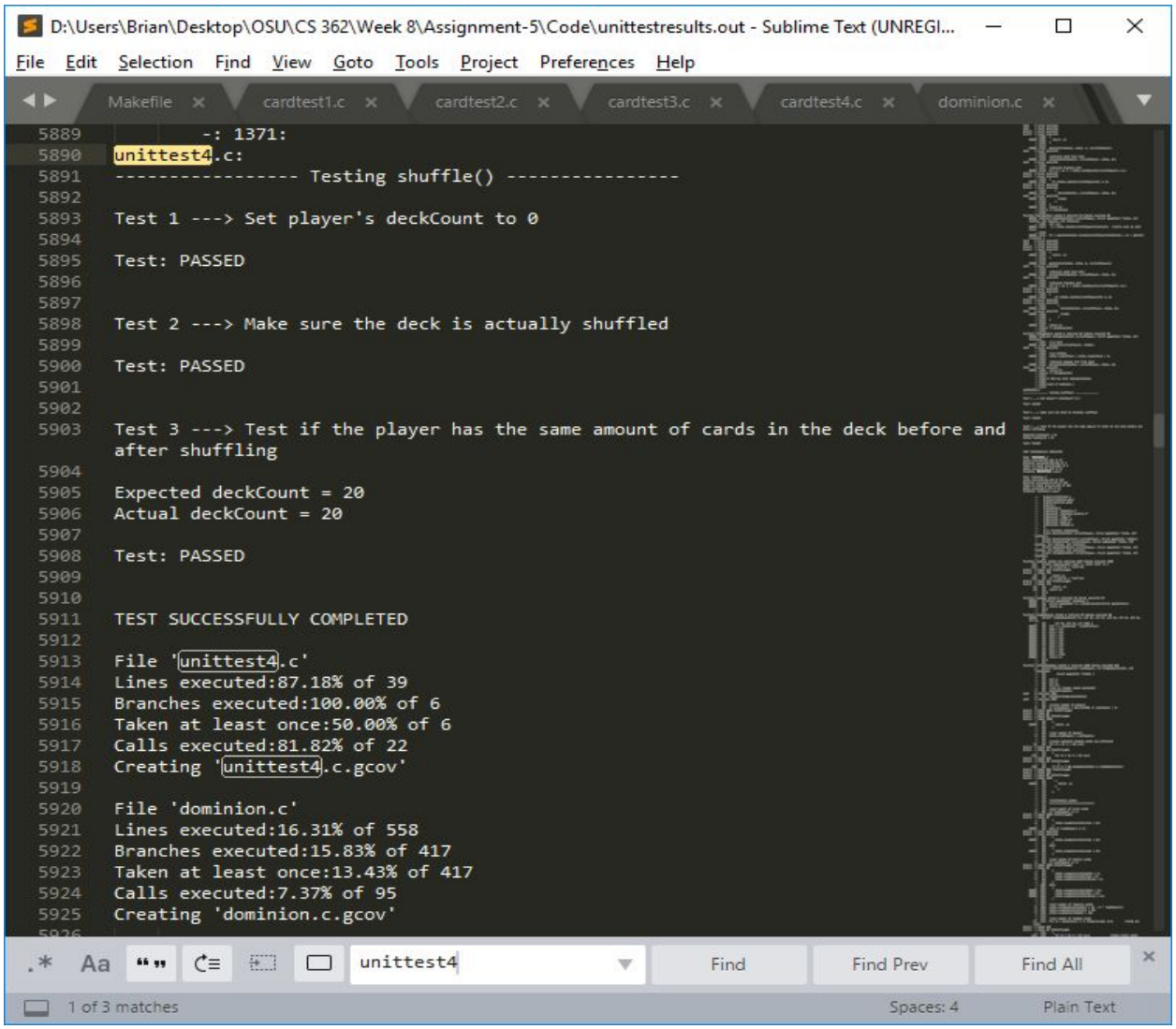
```
        -:  388:}
        -:  389:
function isGameOver called 3 returned 100% blocks executed 100%
        3:  390:int isGameOver(struct gameState *state) {
        -:  391:  int i;
        -:  392:  int j;
        -:  393:
        -:  394:  //if stack of Province cards is empty, the game ends
        3:  395:  if (state->supplyCount[province] == 0)
branch  0 taken 33% (fallthrough)
branch  1 taken 67%
        -:  396:    {
        1:  397:      return 1;
        -:  398:    }
        -:  399:
        -:  400:  //if three supply pile are at 0, the game ends
        2:  401:  j = 0;
       52:  402:  for (i = 0; i < 25; i++)
branch  0 taken 96%
branch  1 taken 4% (fallthrough)
        -:  403:    {
       50:  404:      if (state->supplyCount[i] == 0)
branch  0 taken 6% (fallthrough)
branch  1 taken 94%
        -:  405:      {
        3:  406:        j++;
        -:  407:      }
        -:  408:    }
        2:  409:  if ( j >= 3)
branch  0 taken 50% (fallthrough)
branch  1 taken 50%
        -:  410:    {
        1:  411:      return 1;
        -:  412:    }
        -:  413:
        1:  414:  return 0;
        -:  415:}
        -:  416:
function scoreFor called 0 returned 0% blocks executed 0%
```

.* Aa “” int isgame Find Find Prev Find All

8 matches Spaces: 4 Plain Text

unittest4.c

My unittest4.c file was created to test the shuffle() file. The test received 100% statement and 100% branch coverage for the function. Unfortunately, it only received a 16.31% statement and 15.83% branch coverage for the whole dominion.c file. There were no problem with this function as it was not refactored by my teammate and passed all the tests.



D:\Users\Brian\Desktop\OSU\CS 362\Week 4\Assignment 3\Code\unittestresults.out - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

dominion.c | unittestresults.out

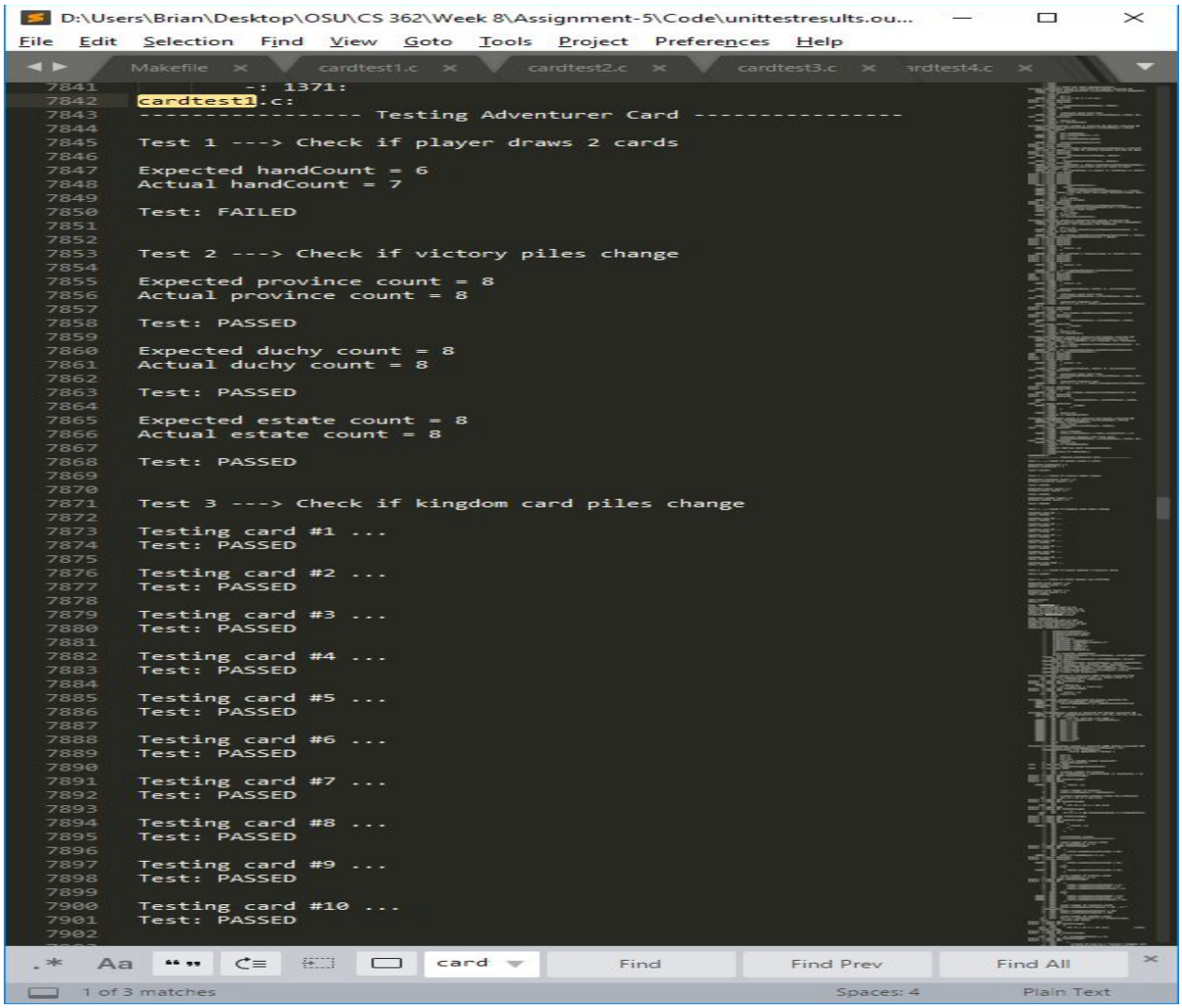
```
function shuffle called 5 returned 100% blocks executed 100%
        5:  201:int shuffle(int player, struct gameState *state) {
        -:  202:
        -:  203:
        -:  204:  int newDeck[MAX_DECK];
        5:  205:  int newDeckPos = 0;
        -:  206:  int card;
        -:  207:  int i;
        -:  208:
        5:  209:  if (state->deckCount[player] < 1)
branch  0 taken 20% (fallthrough)
branch  1 taken 80%
        1:  210:    return -1;
        4:  211:  qsort ((void*)(state->deck[player]), state->deckCount[player], sizeof(int),
            compare);
call    0 returned 100%
        -:  212:  /* SORT CARDS IN DECK TO ENSURE DETERMINISM! */
        -:  213:
       68:  214:  while (state->deckCount[player] > 0) {
branch  0 taken 94%
branch  1 taken 6% (fallthrough)
       60:  215:    card = floor(Random() * state->deckCount[player]);
call    0 returned 100%
       60:  216:    newDeck[newDeckPos] = state->deck[player][card];
       60:  217:    newDeckPos++;
      295:  218:    for (i = card; i < state->deckCount[player]-1; i++) {
branch  0 taken 80%
branch  1 taken 20% (fallthrough)
      235:  219:      state->deck[player][i] = state->deck[player][i+1];
        -:  220:    }
       60:  221:    state->deckCount[player]--;
        -:  222:  }
       64:  223:  for (i = 0; i < newDeckPos; i++) {
branch  0 taken 94%
branch  1 taken 6% (fallthrough)
       60:  224:    state->deck[player][i] = newDeck[i];
       60:  225:    state->deckCount[player]++;
        -:  226:  }
        -:  227:
        4:  228:  return 0;
        -:  229:}
        -:  230:
function playCard called 0 returned 0% blocks executed 0%
```

int shuffle | Find | Find Prev | Find All

8 matches | Spaces: 4 | Plain Text

cardtest1.c

My cardtest1.c was created to test the adventurer card. The test only had 73% statement coverage and missed a few branches. The first branch that it missed was the shuffle call. I didn't give the player and empty deck to test whether the shuffle call worked as intended. The next branch I missed contained a problem where the cards that were drawn were only treasure cards. This was the test's biggest coverage failure. It didn't randomize the cards in the deck nor did it have decks where no treasure cards existed. This test lacked boundary tests as well. Next time I would definitely put certain cards in the deck and test it that way. My dominion statement coverage was 19.89% and the branch coverage was 23.26%. There was a failure in one test. The hand count was one more than it should have been.



D:\Users\Brian\Desktop\OSU\CS 362\Week 8\Assignment-5\Code\unittestresults.ou...

File Edit Selection Find View Goto Tools Project Preferences Help

Makefile | cardtest1.c | cardtest2.c | cardtest3.c | ardtest4.c

```
Test: PASSED


Test 4 ---> Check if player gained 2 treasure cards

Test: PASSED


Test 5 ---> Check if other player was affected

Expected deck count = 10
Actual deck count = 10
Test: PASSED

Expected hand count = 0
Actual hand count = 0
Test: PASSED


TEST FAILED
FAILS: 1

File 'cardtest1.c'
Lines executed:98.46% of 65
Branches executed:100.00% of 22
Taken at least once:77.27% of 22
Calls executed:97.50% of 40
Creating 'cardtest1.c.gcov'

File 'dominion.c'
Lines executed:19.89% of 558
Branches executed:23.26% of 417
Taken at least once:14.87% of 417
Calls executed:10.53% of 95
Creating 'dominion.c.gcov'
```

.* Aa card Find Find Prev Find All

1 of 3 matches Spaces: 4 Plain Text

D:\Users\Brian\Desktop\OSU\CS 362\Week 8\Assignment-5\Code\unittestresults.out - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Makefile | cardtest1.c | cardtest2.c | cardtest3.c | cardtest4.c | dominion.c | stadventurer.out

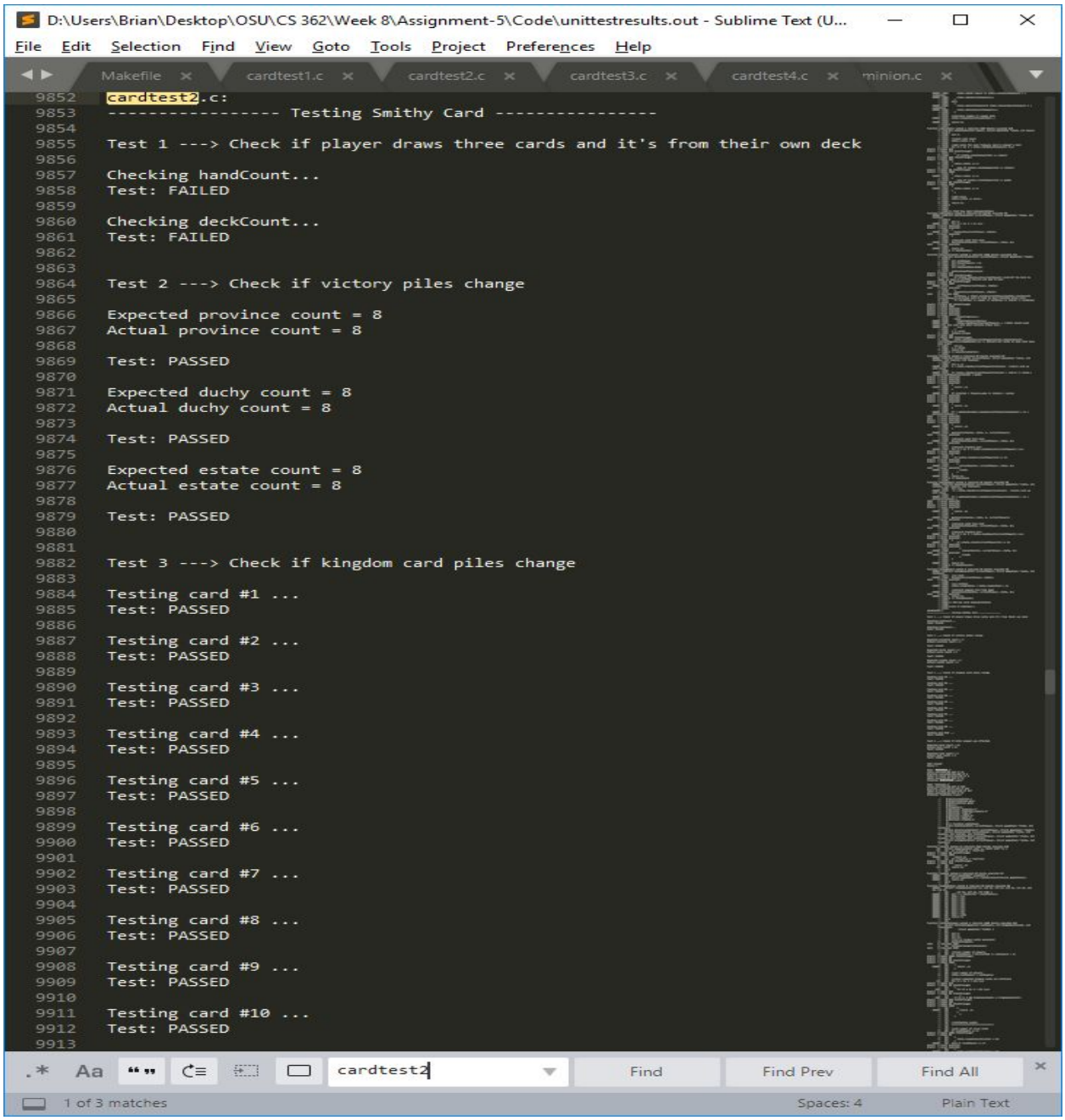
```
        -: 1263:
function AdventurerCard called 1 returned 100% blocks executed 73%
        1: 1264:int AdventurerCard(int currentPlayer, struct gameState *state)
        -: 1265:{
        -: 1266:  int cardDrawn;
        1: 1267:  int drawntreasure = 0;
        1: 1268:  int z = 1;
        -: 1269:  int temphand[MAX_HAND];
        -: 1270:
        4: 1271:  while(drawntreasure<2){
branch  0 taken 67%
branch  1 taken 33% (fallthrough)
        2: 1272:    if (state->deckCount[currentPlayer] <1){//if the deck is empty we need to shuffle discard and add to deck
branch  0 taken 0% (fallthrough)
branch  1 taken 100%
    #####: 1273:      shuffle(currentPlayer, state);
call    0 never executed
        -: 1274:    }
        2: 1275:    drawCard(currentPlayer, state);
call    0 returned 100%
        2: 1276:    cardDrawn = state->hand[currentPlayer][state->handCount[currentPlayer]-1];//top card of hand is most recently drawn card.
        2: 1277:    if (cardDrawn == copper || cardDrawn == silver || cardDrawn == gold)
branch  0 taken 0% (fallthrough)
branch  1 taken 100%
branch  2 never executed
branch  3 never executed
branch  4 never executed
branch  5 never executed
        2: 1278:      drawntreasure++;
        -: 1279:    else{
    #####: 1280:      temphand[z]=cardDrawn;
    #####: 1281:      state->handCount[currentPlayer]--; //this should just remove the top card (the most recently drawn one).
    #####: 1282:      z++;
        -: 1283:    }
        -: 1284:  } // while
        3: 1285:  while(z-1>=0){
branch  0 taken 50%
branch  1 taken 50% (fallthrough)
        1: 1286:    state->discard[currentPlayer][state->discardCount[currentPlayer]++]=temphand[z-1]; // discard all cards in play that have been drawn
        1: 1287:    z=z-1;
        -: 1288:  } // while
        1: 1289:  return 0;
        -: 1290:} // adventurerCard();
        -: 1291:
function MineCard called 0 returned 0% blocks executed 0%
```

.* Aa "" cardtest Find Find Prev Find All

43 lines, 1857 characters selected Spaces: 4 Plain Text

cardtest2.c

My cardtest2.c file was created to test the smithy card. My test received 100% statement and 100% branch coverage. The overall dominion.c coverage was only 20.43% statement and 23.26% branch coverage. One of the tests in cardtest2.c failed. The hand count and deck count were different from what they should have been.



D:\Users\Brian\Desktop\OSU\CS 362\Week 8\Assignment-5\Code\unittestresults.out - Sublime Text (U...

File Edit Selection Find View Goto Tools Project Preferences Help

Makefile | cardtest1.c | cardtest2.c | cardtest3.c | cardtest4.c | minion.c

```
Test: PASSED


Test 4 ---> Check if other player was affected

Expected deck count = 10
Actual deck count = 10
Test: PASSED

Expected hand count = 0
Actual hand count = 0
Test: PASSED


TEST FAILED
FAILS: 2

File 'cardtest2.c'
Lines executed:98.15% of 54
Branches executed:100.00% of 6
Taken at least once:83.33% of 6
Calls executed:97.44% of 39
Creating 'cardtest2.c.gcov'

File 'dominion.c'
Lines executed:20.43% of 558
Branches executed:23.26% of 417
Taken at least once:14.63% of 417
Calls executed:11.58% of 95
Creating 'dominion.c.gcov'
```

.* Aa cardtest2 Find Find Prev Find All

1 of 3 matches Spaces: 4 Plain Text

D:\Users\Brian\Desktop\OSU\CS 362\Week 8\Assignment-5\Code\unittestresults.out - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Makefile | cardtest1.c | cardtest2.c | cardtest3.c | cardtest4.c | dominion.c | rer.c

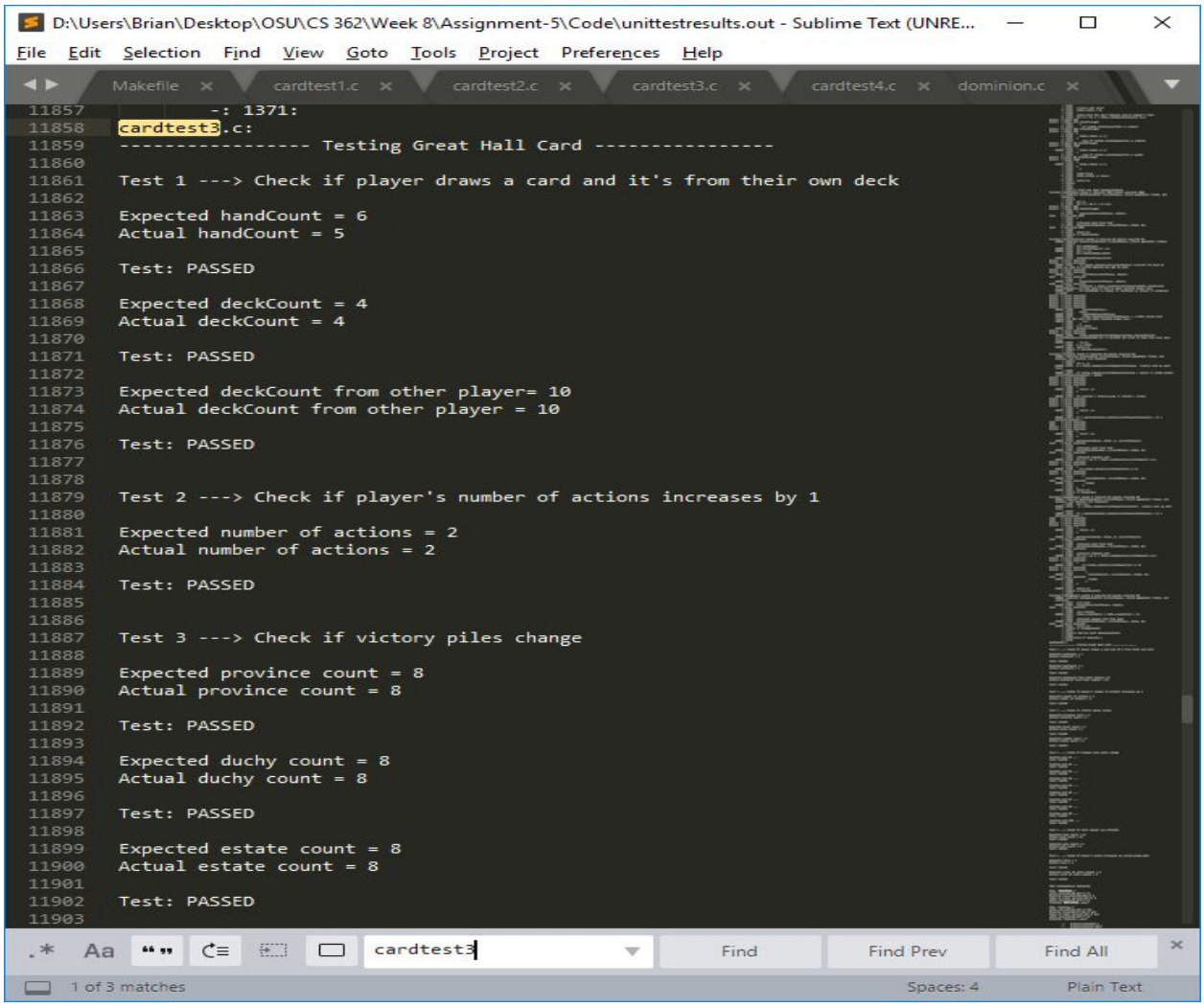
```
        -: 1249:// START New Card Implementations
function SmithyCard called 1 returned 100% blocks executed 100%
        1: 1250:int SmithyCard(int currentPlayer, struct gameState *state, int handPos)
        -: 1251:{
        -: 1252:  int i;
        5: 1253:  for (i = 0; i < 4; i++)
branch  0 taken 80%
branch  1 taken 20% (fallthrough)
        -: 1254:  {
        4: 1255:    drawCard(currentPlayer, state);
call    0 returned 100%
        -: 1256:  }
        -: 1257:
        -: 1258:  //discard card from hand
        1: 1259:  discardCard(handPos, currentPlayer, state, 0);
call    0 returned 100%
        -: 1260:
        1: 1261:  return 0;
        -: 1262:} // SmithyCard()
        -: 1263:
function AdventurerCard called 0 returned 0% blocks executed 0%
    #####: 1264:int AdventurerCard(int currentPlayer, struct gameState *state)
```

.* Aa cardtest3 Find Find Prev Find All

3 matches Spaces: 4 Plain Text

cardtest3.c

My cardtest3.c was created to test the great hall card. Unfortunately, my teammate did not refactor this card or make a separate function for this card so I was not given a definitive statement coverage. Although, it is a very simple card so after inspecting the gcov, it appears that the test received 100% statement and 100% branch coverage. The dominion.c coverage was 22.22% statement coverage and 27.10% branch coverage. There were no problem with this function as it was not refactored by my teammate and passed all the tests.



D:\Users\Brian\Desktop\OSU\CS 362\Week 8\Assignment-5\Code\unittestresults.out - Sublime Text (UNRE...

File Edit Selection Find View Goto Tools Project Preferences Help

Makefile | cardtest1.c | cardtest2.c | cardtest3.c | cardtest4.c | dominion.c

```
Test 4 ---> Check if kingdom card piles change

Testing card #1 ...
Test: PASSED

Testing card #2 ...
Test: PASSED

Testing card #3 ...
Test: PASSED

Testing card #4 ...
Test: PASSED

Testing card #5 ...
Test: PASSED

Testing card #6 ...
Test: PASSED

Testing card #7 ...
Test: PASSED

Testing card #8 ...
Test: PASSED

Testing card #9 ...
Test: PASSED

Testing card #10 ...
Test: PASSED


Test 5 ---> Check if other player was affected

Expected deck count = 10
Actual deck count = 10
Test: PASSED

Expected hand count = 0
Actual hand count = 0
Test: PASSED


Test 6 ---> Check if player's score increases by having great_hall

Expected score = 2
Actual score = 1

Test: PASSED

Expected score of other player = 0
Actual score of other player = 0

Test: PASSED


TEST SUCCESSFULLY COMPLETED

File 'cardtest3.c'
Lines executed:94.59% of 74
Branches executed:100.00% of 6
Taken at least once:66.67% of 6
Calls executed:95.52% of 67
Creating 'cardtest3.c.gcov'

File 'dominion.c'
Lines executed:22.22% of 558
Branches executed:27.10% of 417
Taken at least once:16.79% of 417
Calls executed:10.53% of 95
Creating 'dominion.c.gcov'
```

cardtest3 | Find | Find Prev | Find All

1 of 3 matches | Spaces: 4 | Plain Text

D:\Users\Brian\Desktop\OSU\CS 362\Week 8\Assignment-5\Code\unittestresults.out - Sublime Text (UNRE...

File Edit Selection Find View Goto Tools Project Preferences Help

Makefile | cardtest1.c | cardtest2.c | cardtest3.c | cardtest4.c | dominion.c

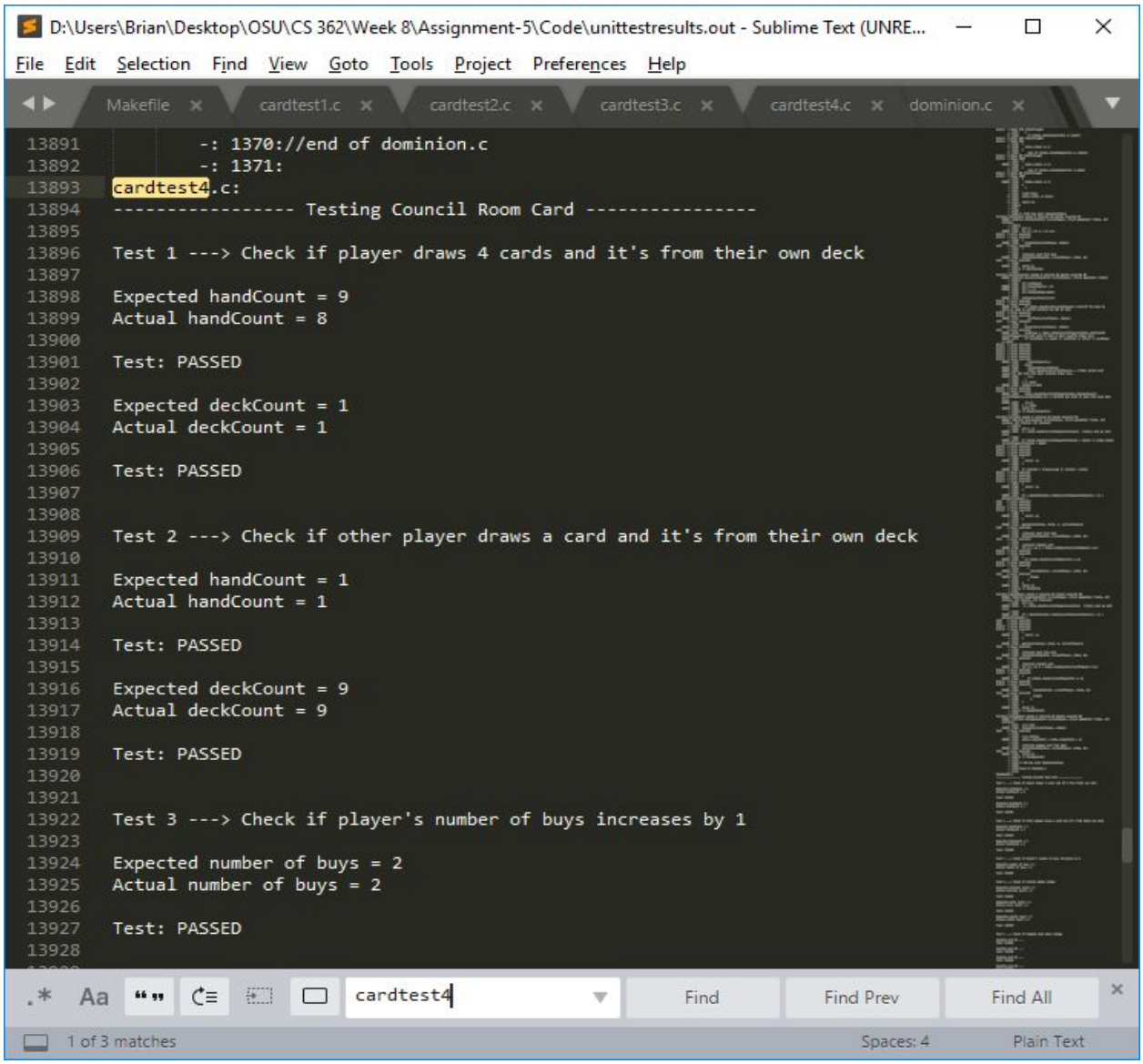
```
        -:  818:
    #####:  819:        return 0;
        -:  820:
        -:  821:    case great_hall:
        -:  822:        //+1 Card
        1:  823:        drawCard(currentPlayer, state);
call    0 returned 100%
        -:  824:
        -:  825:        //+1 Actions
        1:  826:        state->numActions++;
        -:  827:
        -:  828:        //discard card from hand
        1:  829:        discardCard(handPos, currentPlayer, state, 0);
call    0 returned 100%
        1:  830:        return 0;
        -:  831:
        -:  832:    case minion:
```

cardtest4 | Find | Find Prev | Find All

3 matches | Spaces: 4 | Plain Text

cardtest4.c

My cardtest4.c was created to test the council room card. Unfortunately, my teammate did not refactor this card or make a separate function for this card so I was not given a definitive statement coverage. Although, after inspecting the gcov, it appears that the test received 100% statement and 100% branch coverage. The overall coverage of dominion.c was 20.79% statement coverage and 24.22% branch coverage. There were no problem with this function as it was not refactored by my teammate and passed all the tests.



D:\Users\Brian\Desktop\OSU\CS 362\Week 8\Assignment-5\Code\un...

File Edit Selection Find View Goto Tools Project Preferences Help

Makefile | cardtest1.c | cardtest2.c | cardtest3.c

```
Test 4 ---> Check if victory piles change

Expected province count = 8
Actual province count = 8

Test: PASSED

Expected duchy count = 8
Actual duchy count = 8

Test: PASSED

Expected estate count = 8
Actual estate count = 8

Test: PASSED


Test 5 ---> Check if kingdom card piles change

Testing card #1 ...
Test: PASSED

Testing card #2 ...
Test: PASSED

Testing card #3 ...
Test: PASSED

Testing card #4 ...
Test: PASSED

Testing card #5 ...
Test: PASSED

Testing card #6 ...
Test: PASSED

Testing card #7 ...
Test: PASSED

Testing card #8 ...
Test: PASSED

Testing card #9 ...
Test: PASSED

Testing card #10 ...
Test: PASSED


TEST SUCCESSFULLY COMPLETED

File 'cardtest4.c'
Lines executed:93.44% of 61
Branches executed:100.00% of 6
Taken at least once:66.67% of 6
Calls executed:93.48% of 46
Creating 'cardtest4.c.gcov'

File 'dominion.c'
Lines executed:20.79% of 558
Branches executed:24.22% of 417
Taken at least once:15.59% of 417
Calls executed:11.58% of 95
Creating 'dominion.c.gcov'
```

.* Aa Find Find Prev Find All

1 of 3 matches Spaces: 4 Plain Text

D:\Users\Brian\Desktop\OSU\CS 362\Week 8\Assignment-5\Code\unittestresults.out - Sublime Tex...

File Edit Selection Find View Goto Tools Project Preferences Help

Makefile | cardtest1.c | cardtest2.c | cardtest3.c | cardtest4.c | n.c

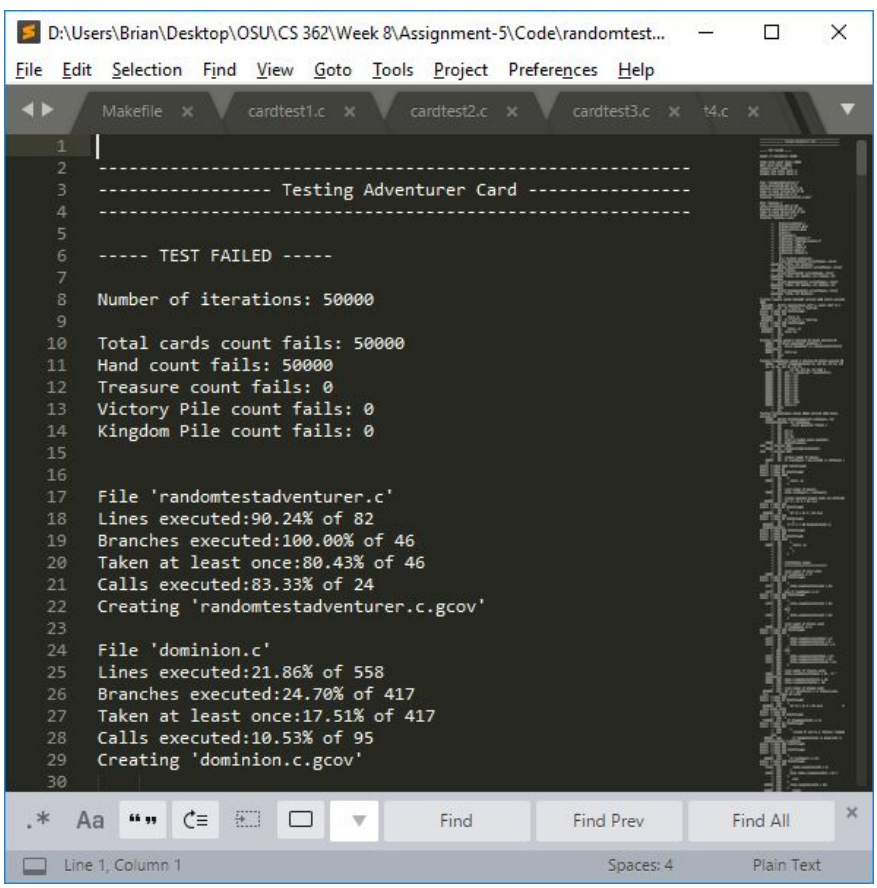
```
call    0 never executed
        -:  676:
        -:  677:    case council_room:
        -:  678:        //+4 Cards
        5:  679:        for (i = 0; i < 4; i++)
branch  0 taken 80%
branch  1 taken 20% (fallthrough)
        -:  680:        {
        4:  681:            drawCard(currentPlayer, state);
call    0 returned 100%
        -:  682:        }
        -:  683:
        -:  684:        //+1 Buy
        1:  685:        state->numBuys++;
        -:  686:
        -:  687:        //Each other player draws a card
        3:  688:        for (i = 0; i < state->numPlayers; i++)
branch  0 taken 67%
branch  1 taken 33% (fallthrough)
        -:  689:        {
        2:  690:            if ( i != currentPlayer )
branch  0 taken 50% (fallthrough)
branch  1 taken 50%
        -:  691:            {
        1:  692:                drawCard(i, state);
call    0 returned 100%
        -:  693:            }
        -:  694:        }
        -:  695:
        -:  696:        //put played card in played card pile
        1:  697:        discardCard(handPos, currentPlayer, state, 0);
call    0 returned 100%
        -:  698:
        1:  699:        return 0;
        -:  700:
        -:  701:    case feast:
```

.* Aa cardtest4 Find Find Prev Find All

3 matches Spaces: 4 Plain Text

randomtestadventurer.c

For the adventurer card, I completed 93% statement coverage and completely missed one if statement in the branch coverage. The coverage for the whole dominion.c file was 21.86% statement coverage and 24.70% branch coverage. The reason that I didn't achieve 100% statement and branch coverage for the adventurer card because I failed to hit the if statement which checks to see if the player's deck count is below one and then proceeds to shuffle the cards. I didn't hit this branch because I didn't allow the randomized deck count for the players go below 1 card because I wanted the deck to contain a certain amount of treasure cards to be able to test the card properly. This may have been an oversight and if I were to recreate this test I would allow the randomized deck count to hit zero instead of giving it a minimum amount. The same failures occurred for 50,000 iterations. The failure was that the total cards count was wrong and the hand count was wrong.



D:\Users\Brian\Desktop\OSU\CS 362\Week 8\Assignment-5\Code\randomtestadventurer.out - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Makefile | cardtest1.c | cardtest2.c | cardtest3.c | cardtest4.c | dominion.c | randomtestadventurer.out

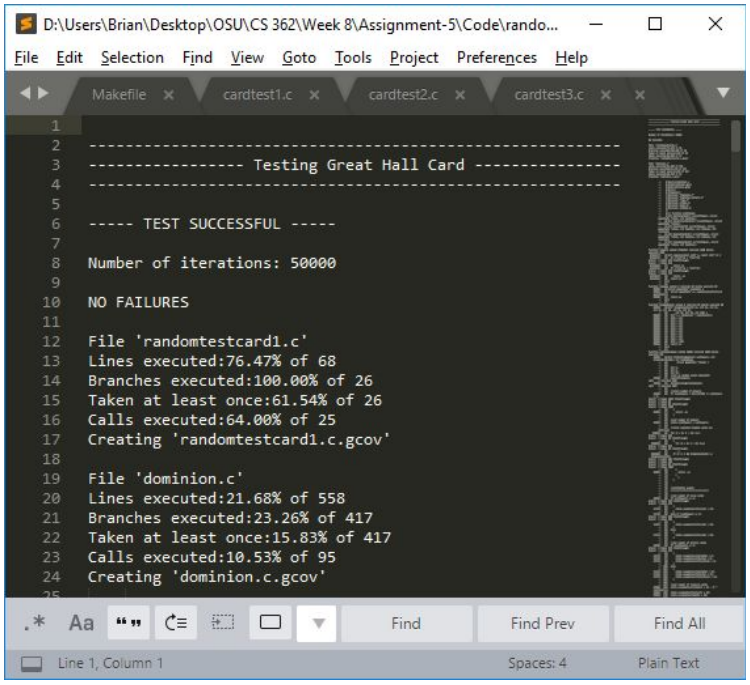
```
        -: 1263:
function AdventurerCard called 50000 returned 100% blocks executed 93%
    50000: 1264:int AdventurerCard(int currentPlayer, struct gameState *state)
        -: 1265:{
        -: 1266:  int cardDrawn;
    50000: 1267:  int drawntreasure = 0;
    50000: 1268:  int z = 1;
        -: 1269:  int temphand[MAX_HAND];
        -: 1270:
  6671321: 1271:  while(drawntreasure<2){
branch  0 taken 99%
branch  1 taken 1% (fallthrough)
  6571321: 1272:    if (state->deckCount[currentPlayer] <1){//if the deck is empty we need to shuffle discard and add to deck
branch  0 taken 0% (fallthrough)
branch  1 taken 100%
    #####: 1273:      shuffle(currentPlayer, state);
call    0 never executed
        -: 1274:    }
  6571321: 1275:    drawCard(currentPlayer, state);
call    0 returned 100%
  6571321: 1276:    cardDrawn = state->hand[currentPlayer][state->handCount[currentPlayer]-1];//top card of hand is most recently drawn card.
  6571321: 1277:    if (cardDrawn == copper || cardDrawn == silver || cardDrawn == gold)
branch  0 taken 99% (fallthrough)
branch  1 taken 1%
branch  2 taken 99% (fallthrough)
branch  3 taken 1%
branch  4 taken 1% (fallthrough)
branch  5 taken 99%
   100000: 1278:      drawntreasure++;
        -: 1279:    else{
  6471321: 1280:      temphand[z]=cardDrawn;
  6471321: 1281:      state->handCount[currentPlayer]--; //this should just remove the top card (the most recently drawn one).
  6471321: 1282:      z++;
        -: 1283:    }
        -: 1284:  } // while
  6621321: 1285:  while(z-1>=0){
branch  0 taken 99%
branch  1 taken 1% (fallthrough)
  6521321: 1286:    state->discard[currentPlayer][state->discardCount[currentPlayer]++]=temphand[z-1]; // discard all cards in play that have been drawn
  6521321: 1287:    z=z-1;
        -: 1288:  } // while
    50000: 1289:  return 0;
        -: 1290:} // adventurerCard();
        -: 1291:
function MineCard called 0 returned 0% blocks executed 0%
    #####: 1292:int MineCard(int currentPlayer, struct gameState *state, int handPos, int choice1, int
```

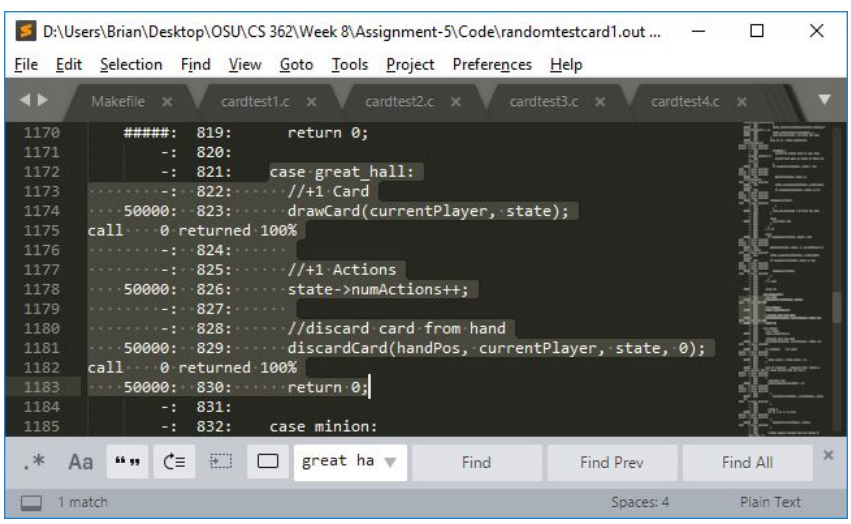
cardtest4 | Find | Find Prev | Find All

43 lines, 1864 characters selected | Spaces: 4 | Plain Text

randomtestcard1.c

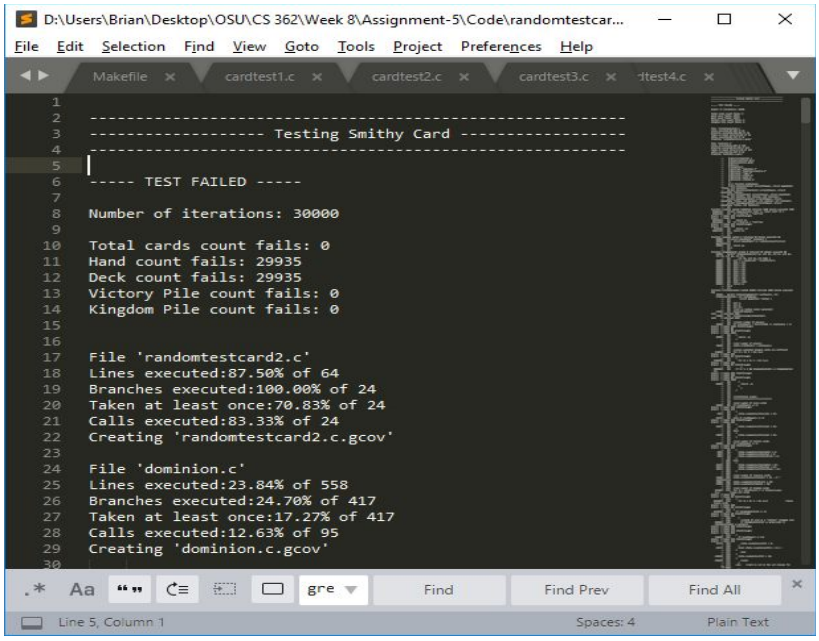
For randomtestcard1.c, I tested the Great Hall card. The test received 100% statement and branch coverage for the Great Hall card function. The Great Hall card is a very basic card function that only has one branch and my teammate didn't refractor this card with a bug or it's own function, so I am not surprised that I achieved 100% statement and branch coverage. It only took 1 iteration to complete 100% coverage. It would have been very hard to not hit 100% coverage for this function. All the tests that I created passed for this function.

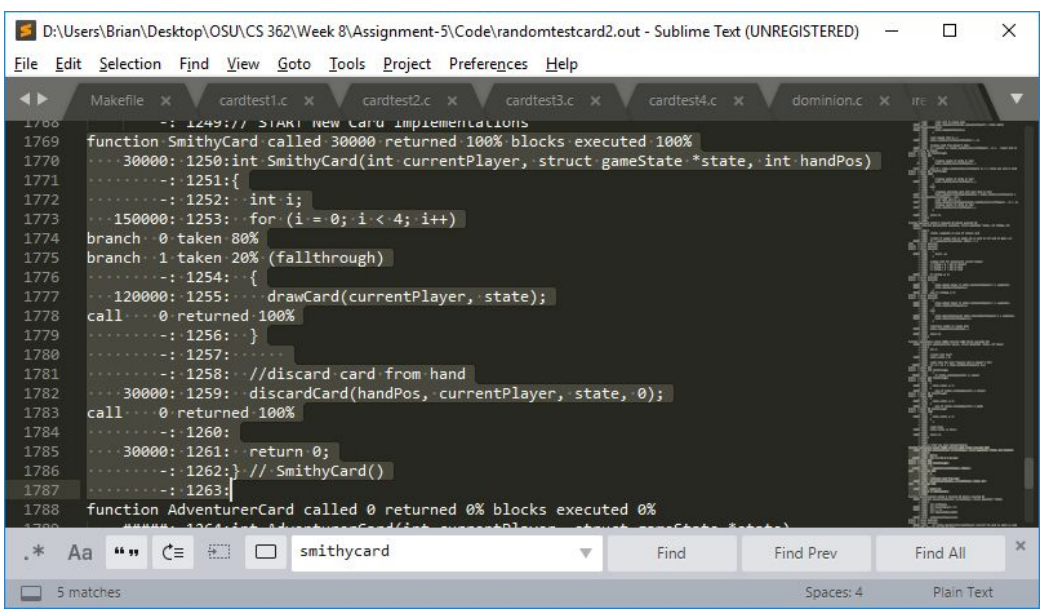




randomtestcard2.c

For randomtestcard2.c, I tested the Smithy card. The test received 100% statement and 100% branch coverage. The overall dominion.c coverage was 23.84% statement and 34.70% branch coverage. 99% of the time the test failed in hand and deck count.

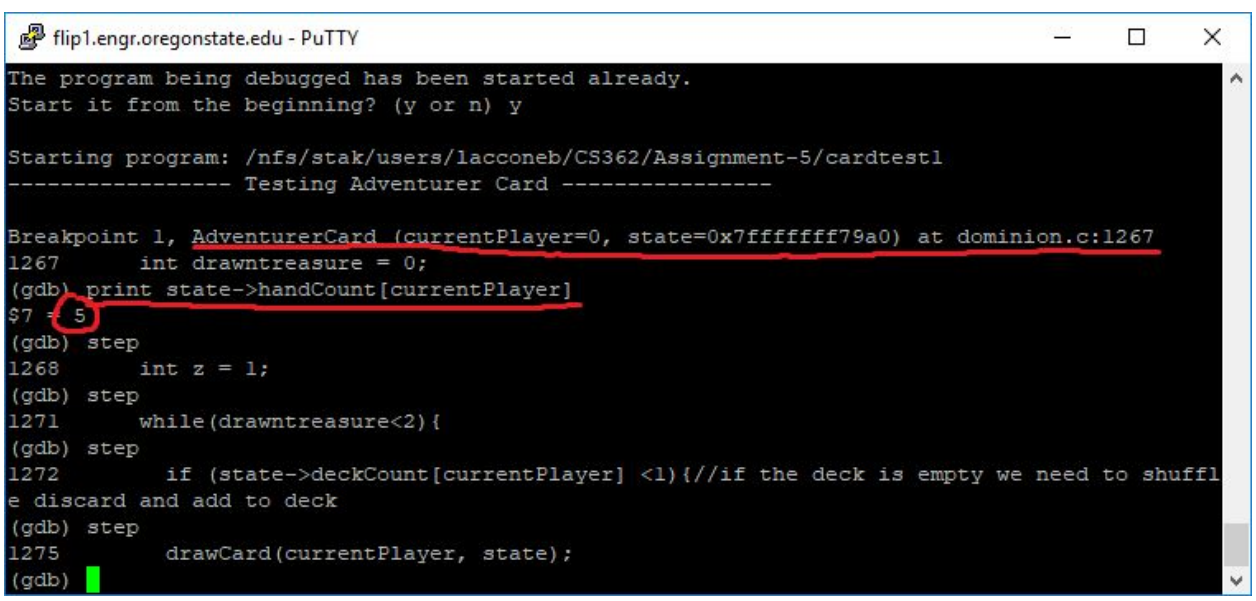




Debugging

In this section I will be going over how I debugged the three bugs that I found while testing my teammate's code.

1. The first bug that I noticed was the hand count difference while testing cardtest1.c (AdventurerCard()). I noticed that the hand count was always 1 higher so I used gbd to debug the problem. In the following pictures I have displayed how I came to realize how the bug was happening. I didn't capture the whole process because it was many lines of stepping through the function. I first made a breakpoint at AdventurerCard in cardtest1.c. I ran gdb on cardtest1 and once it hit the breakpoint I printed out the hand count of the current player. It printed 5, which is what is expected because players start the game with 5 cards in hand. Next, I stepped through the whole function and printed the hand count at certain times throughout the function. It drew 2 treasure cards and hit 7 cards. That hand count stayed the same until the end of the function. After debugging this problem, I quickly realized that the adventurer card, which is in hand, is never being discarded. To fix this bug, I added another parameter to the function called handPos to hold the hand position of the adventurer card and then sent a call to discardCard() to discard the adventurer card. This fixed the bug and the preceding test verified that the hand counts were the same.



flip1.engr.oregonstate.edu - PuTTY

```
(gdb) step
1285        while(z-1>=0){
(gdb) print z
$4 = 1
(gdb) step
1286          state->discard[currentPlayer][state->discardCount[currentPlayer]++]=temphand[z-1
]; // discard all cards in play that have been drawn
(gdb) print state->handCount[currentPlayer]
$5 = 7
(gdb) step
1287          z=z-1;
(gdb) step
1285        while(z-1>=0){
(gdb) step
1289        return 0;
(gdb) step
1290      } // adventurerCard();
(gdb) print state->handCount[currentPlayer]
$6 = 7
(gdb)
```

2. The second bug that I found while testing AdventurerCard() was the total card count problem that randomtestadventurer.c discovered. This test was concluding that the sum of the number of cards in hand, discard pile, and deck where not the same after calling AdventurerCard() than before the function ran. While running through the function with gdb, I made sure to keep track of what branch the code was taking. The first and second card drawn was a treasure card, which means that the function never took the path to lower the hand count and increment int z by one. This is highlighted in the first two pictures in this section. The third picture shows the value of z after the while(drawntreasure<2) loop finished. The value of z was 1, when the value should have been 0 to prevent the function from discarding the non treasure cards that the function drew. I quickly looked up at the value of z and realized that my teammate had initialized int z to equal 1 instead of 0. To fix this problem, I simply initialized int z = 0. After fixing this mistake, both cardtest1.c and randomtestadventurer.c were successful.

flip1.engr.oregonstate.edu - PuTTY

```
583         state->deckCount[player]--;
(gdb) step
584         state->handCount[player]++;//Increment hand count
(gdb) step
587       return 0;
(gdb) step
588     }
(gdb) step
AdventurerCard (currentPlayer=0, state=0x7fffffff79a0) at dominion.c:1276
1276        cardDrawn = state->hand[currentPlayer][state->handCount[currentPlayer]-1];//top
card of hand is most recently drawn card.
(gdb) step
1277        if (cardDrawn == copper || cardDrawn == silver || cardDrawn == gold)
(gdb) print cardDrawn
$3 = 4
(gdb) step
1278          drawntreasure++;
(gdb) step
1271      while(drawntreasure<2){
(gdb)
```

2

flip1.engr.oregonstate.edu - PuTTY

```
(gdb) step
AdventurerCard (currentPlayer=0, state=0x7fffffff79a0) at dominion.c:1276
1276        cardDrawn = state->hand[currentPlayer][state->handCount[currentPlayer]-1];//top
card of hand is most recently drawn card.
(gdb) step
1277        if (cardDrawn == copper || cardDrawn == silver || cardDrawn == gold)
(gdb) print cardDrawn
$3 = 4
(gdb) step
1278          drawntreasure++;
(gdb) step
1271      while(drawntreasure<2){
(gdb) step
1285      while(z-1>=0){
(gdb) print z
$4 = 1
(gdb) step
1286        state->discard[currentPlayer][state->discardCount[currentPlayer]++]=temphand[z-1
]; // discard all cards in play that have been drawn
(gdb)
```

3. The third and last bug that I discovered was found when running cardeffect2.c and randomtestcard2.c. This test discovered that there was a hand count error after running SmithyCard(). The hand count was always one more than it should be except for the rare random test which only initialized the starting deck to have 3 cards in it. The following screenshots illustrate the hand count before and after running SmithyCard(). The hand count was 8 instead of 7. During the for loop I also printed the hand count after each iteration ran and realized that the loop was running 4 times before it stopped. This confirmed that the for loop which was suppose to only draw three cards was drawing 4 instead. To fix this problem I simply changed the for loop to i<3 instead of i<4. This fixed the problem and all subsequent tests were successful.