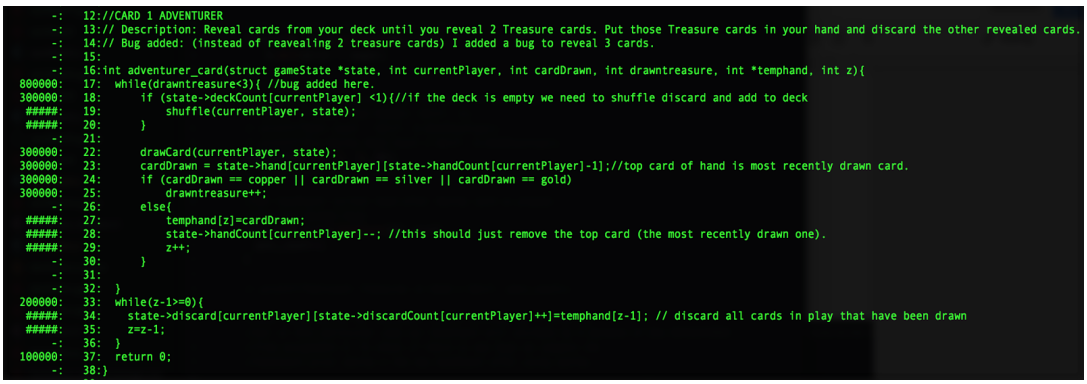
Juan Daccarett

Random Testing

Adventurer

- Random Testing
    - For adventurer I randomized multiple variables to increase the chances of finding errors. I started randomizing game states such as **player**, **whoseturn** and the **player's hand** at every test iteration (one hundred thousand test iterations). Once game states are randomized I keep count of the treasure cards in the player's hand before and after the adventure card is played. With the pre and post treasure card count I can now test if the correct amount of the treasure cards are being put into the player's hand. Because I added a refactored bug to the function adventurer I am testing if adventurer is adding 3 cards to our deck . Analyzing the code coverage below for the adventurer function I can see that I can construct a test that can check when the deck is empty in line 19. Since I hardcoded the deck to have all coppers we never cover line # 27 - 30. This can be a simple fix by randomizing the cards in the game state deck.
- Code Coverage
    - 

```
    -:   12://CARD 1 ADVENTURER
    -:   13:// Description: Reveal cards from your deck until you reveal 2 Treasure cards. Put those Treasure cards in your hand and discard the other revealed cards.
    -:   14:// Bug added: (instead of reavealing 2 treasure cards) I added a bug to reveal 3 cards.
    -:   15:
    -:   16:int adventurer_card(struct gameState *state, int currentPlayer, int cardDrawn, int drawntreasure, int *temphand, int z){
800000:   17:  while(drawntreasure<3){ //bug added here.
300000:   18:      if (state->deckCount[currentPlayer] <1){//if the deck is empty we need to shuffle discard and add to deck
 #####:   19:          shuffle(currentPlayer, state);
 #####:   20:      }
    -:   21:
300000:   22:      drawCard(currentPlayer, state);
300000:   23:      cardDrawn = state->hand[currentPlayer][state->handCount[currentPlayer]-1];//top card of hand is most recently drawn card.
300000:   24:      if (cardDrawn == copper || cardDrawn == silver || cardDrawn == gold)
300000:   25:          drawntreasure++;
    -:   26:      else{
 #####:   27:          temphand[z]=cardDrawn;
 #####:   28:          state->handCount[currentPlayer]--; //this should just remove the top card (the most recently drawn one).
 #####:   29:          z++;
    -:   30:      }
    -:   31:
    -:   32:  }
200000:   33:  while(z-1>=0){
 #####:   34:    state->discard[currentPlayer][state->discardCount[currentPlayer]++]=temphand[z-1]; // discard all cards in play that have been drawn
 #####:   35:    z=z-1;
    -:   36:  }
100000:   37:  return 0;
    -:   38:}
```

Smithy

- Random Testing
    - For Smithy I randomized multiple variables to increase the chances of finding errors. I was able to get 100% statement and branch coverage with one hundred thousand iterations of random variables. I can lower the test case iterations to five thousand and still yield one hundred percent coverage. Because I introduced a bug in my smithy card to add 4 cards to our hand instead of the 3. I checked where the difference between post and pre is 3 and not 2. Since we discard Smithy after its use. Therefore the hand Count should be 8 which means we added 4 cards to our hand Count after smithy use. Fortunately after our multiple tests smithy card function was able to increase our hand size correctly without any errors.
    - Code Coverage
    - 

```
      -:   41:// CARD 2 SMITHY
      -:   42:// Description: its ability to increase your handsize by drawing 3 cards
      -:   43:// Bug added: instead of increasing your handsize by 3 we will incease it to 4.
      -:   44:
      -:   45:
      -:   46:int smithy_card(int currentPlayer, struct gameState *state, int handPos, int i){
      -:   47:  //+3 Cards
1000000:   48:  for (i = 0; i < 4; i++){ //Bug added to add 4 cards to handsize.
 400000:   49:    drawCard(currentPlayer, state);
 400000:   50:  }
      -:   51:
      -:   52:  //discard card from hand
 100000:   53:  discardCard(handPos, currentPlayer, state, 0);
 100000:   54:  return 0;
      -:   55:}
      -:   56:
      -:   57://*****************************************************************************************//
```

Cutpurse

- Random Testing
    - For Cutpurse I randomized multiple variables to increase the chances of finding errors. I proceeded to save a before and after playing cutpurse game state coin

amount of the player. This was helpful to be able to test if the cutpurse was actually updating our coin state to +4 coins (with refactored bug). The test seemed to fail to update the right amount of coins until the 14th test which updated the coins correctly. Looking at the code coverage I can construct test cases to test lines 133 to 141 which deals with the logic to reveal the opponent's cards. Another place that can help up find the bug can be found in update Coins function. The reason I think update Coins holds the bug is because in assignment 2 one of my unit test was updated Coins which failed my tests.

- Code Coverage
-

```
    -:  116:// CARD 5 cutpurse
    -:  117:// Description:It is a terminal Silver, since it gives +2 coins when played, but it also makes other players discard Coppers.
    -:  118:// Bug added: instead of +2 coins it gives you +4 coins.
    -:  119:
    -:  120:int cutpurse_card(int currentPlayer, struct gameState *state, int i, int j, int k, int handPos){
    -:  121:
   16:  122:  updateCoins(currentPlayer, state, 4);
    -:  123:
   96:  124:  for (i = 0; i < state->numPlayers; i++){
   32:  125:      if (i != currentPlayer){
    -:  126:
   32:  127:        for (j = 0; j < state->handCount[i]; j++){
   10:  128:          if (state->hand[i][j] == copper){
   10:  129:              discardCard(j, i, state, 0);
   10:  130:              break;
    -:  131:          }
    -:  132:
#####:  133:          if (j == state->handCount[i]){
    -:  134:
#####:  135:            for (k = 0; k < state->handCount[i]; k++){
    -:  136:              if (DEBUG)
    -:  137:              printf("Player %d reveals card number %d\n", i, state->hand[i][k]);
#####:  138:            }
#####:  139:            break;
    -:  140:          }
#####:  141:        }
   16:  142:      }
    -:  143:
   32:  144:  }
    -:  145:  //discard played card from hand
   16:  146:  discardCard(handPos, currentPlayer, state, 0);
    -:  147:
   16:  148:  return 0;
```

Unit vs Random

Looking at my code coverage for my previous unit test I am starting to see how limited my test where compared to the random testing we are doing this week. Most of my line coverage of my unit test does not exceed hitting each line more than 10 times. Although I am having the same amount of coverage percentage for both adventurer and smithy. Compared to unit testing for adventurer and smithy I am hitting each line 50k times more with random testing in contrast to the 30 times max in unit testing. Adding the variability of random game states in random testing that could potentially expose bugs that could have never been discovered in unit testing. I now understand why the lecturer stated that when it comes to testing “test harder to smarted”. With today's computing power we can get away with testing a million random values with very little time that could potentially expose bugs that unit test could not otherwise.