DREW WOLFE | ONID: WOLFEDR
CS 362-400 WINTER 2018

ASSIGNMENT: 4, RANDOM TESTER

CODE UNDER TEST:
Smithy
Council Room
Adventurer

Random Testing

Someone mentioned in the class Slack workspace that the testDrawCard.c provides a reasonable foundation or framework for the assignment, so I started with that. For each test I wrote a method that duplicates the game state, essentially encapsulating before and after states, so that specific attributes of the game state structure could be manipulated and subsequently compared. I then wrote the main() functions capable of generating randomized input and essentially running the targeted card through thousands of game states. Of course, this is a generalized summary of my approach. Each card included specific and unique challenges that have contributed to further hair-loss to my already Jason-Statham-like-bald-head (i.e. the "unique challenges" were frustrating).

CUT | smithy_card() defined within dominion.c
The primary challenges for most, if not all cards, arose from the game state attributes controlled by the card-specific methods (obviously, as they are not generalized). The following game state struct attributes and their dependencies required individualized management:

- **whoseTurn**: My goal was to compare the same player with two different game states. The first games state had to be modified manually and the second had to be modified by the cardEffect. In order to do this, I needed to ensure that the whoseTurn variable had been set to the randomly selected player.
- **playedCardCount**: I kept causing a seg fault because I had not initially bounded the randomized playedCardCount variable. It seems it would occasionally attempt to access memory outside the deck array. The problem of course was that this didn't happen every time, so it wasn't a particularly easy issue to rundown (I mean, I had to trace the code, which was more annoying than difficult). Ultimately I bound the variable and didn't have the issue moving forward.

CUT | council_room_card() defined within dominion.c

- **Uniform Decks**: The Council Room Card gives the holding-player +4 cards, but it also gives every other player at the table +1 card, which triggers all other players to call the drawCard function. As a result, it seemed to make the most sense to control the game state of all players, setting the same hand, discard count, and starting deck (which wasn't easy, but seemed better than the alternative).
- **numBuys**: The Council Room Card increments the numBuys value, so to control the expected outcome, I set it to one (1).
- **playedCardCount** and **whoseTurn**: Both variables required generally the same modifications used to manipulate the smithy_card()

CUT | adventurer_card() defined within dominion.c

- **Meeting Loop Exit Criteria**: The biggest issue with this card was ensuring that the deck and hand meet the criteria of exiting the loop, that is to say, the player has to cycle through the hand and/or deck in order to find three (3) Treasure Cards. I bounded the randomized deck

and hand count and ensured that there were never less than three (3) treasure cards within the combined player's hand and deck.
- **discardCount**: As part of the player searching for the treasure within his hand and eventually within the larger deck, the player must continually discard non-treasure cards. While there may be another way to control for this, I simply set the discard count to zero (0) to prevent the discardCount from growing larger than the limit of the deck array.
- **whoseTurn**: Very similar to the Council Room Card and Smithy Card, the whoseTurn variable needed to manually manipulated in order to control the cardEffect outcome.

Code Coverage

```
********************************** GCOV OUTPUT **********************************
File 'randomtestcard2.c'
Lines executed:78.72% of 94

********************************** GCOV OUTPUT **********************************
File 'randomtestcard1.c'
Lines executed:78.13% of 64

********************************** GCOV OUTPUT **********************************
File 'randomtestadventurer.c'
Lines executed:81.11% of 90
```

I generally expected the coverage to be better than ~80%. Surprisingly the best coverage came from the one card I least expected. The Adventurer Card has an empty deck condition that seems pretty impossible to test randomly. Outside of that, I had really figured with the range of variables and the number of test cycles (25,000), I would have had greater coverage, but alas, that wasn't the case. I suppose I could have increased the test cycles; it took less than a minute for each to run, and with an up-to five (5) minute limit, I suspect the coverage would increase with more game state scenarios.

Unit vs Random

The unit tests had a much better ability to detect and convey specific faults. I believe that was largely as a result of building targeted tests with very specific error messages. The random testing has about 30% greater coverage, which is pretty fantastic, but, likely a fault of how I built the tests, the tests don't do as great a job exposing errors. The act of writing the unit tests seemed to force me to think more critically about how to test a method or function, while the random tests seem to force me to think about how to build a test that wouldn't break the compiler… Generally I've come away with the feeling that unit tests provide "precision" while random tests provide "horsepower". I should probably figure out how to merge the two approaches.