Random Testing

I worked through the development of the testers by starting out with the tester for Adventurer. Initially I had it in a single c file, but, in thinking about the need to run the additional tests for two other cards, I was concerned about rework if I just initiated the other testers from that code. So, I opted to refactor and pull the common functions out into randomtestdriver.c, which contains:

- main, which calls, for each of the cards, functions to setup the gameState, to run the test functions (generally very similar to one another), to run the oracle for each card (some similarities as well as differences from the other oracles), and to output the results
- setupGameState, which just does random initialization of a gameState struct
- outputTestResults which prints out brief text and counts of the errors discovered by the oracle functions, using a struct of int counters
- rand_int, which supports getting a range of integers using the rngs.c randomizer
- dumpGameState, a helper function that I used in debugging

In addition to randomtestdriver.c, I have the randomtestadventure.c, randomtestcard1.c, and randomtestcard2.c modules, which contain a function to execute the test and a function to validate results, as noted above.

Initially, test scenarios were very simple, looking only for the basics of the cards, but, as I thought more deeply about the scenarios, I added additional validations. For example, initially I did not have a validation for Adventurer that checked for two treasure cards in the hand and no additional treasure cards in discard.

I refactored a bit more both before and after starting work on the randomtestcard1.c and randomtestcard2.c modules, and I'm generally pleased with the results of the efforts.

One thing that I struggled with greatly was Segmentation Faults. I was blocked for several hours through the course of this assignment trying to root out the cause of the memory violation.

Code Coverage

I was able to cover 100% of the lines and branches for the three cards I tested: Adventurer, Smithy, and Council Room. All of the tests run very quickly – in seconds – on flip. I generally ran 100 cases per test run, and fully covered the 18 lines and 12 branches of Adventurer, the 6 lines and 2 branches of Smithy, and the 9 lines and 4 branches of Council Room. Very little of the rest of dominion.c was covered, e.g.., only 5.8% of the 207 lines and 12.85% of the branches of cardEffect were covered, which is not surprising given the nature of that function.

I will say that it is now obvious to me why monolithic functions, like cardEffect, are so difficult to test and why we started the quarter with refactoring that function into the discrete functions of assignment 2.

Unit vs. Random

Unfortunately, I had not completely my unit testing homework by the due date. I was again unable to complete Assignment 4 before the due date, but at least I was able (mostly because of the public holiday on 19-Feb) to work through the night to complete the assignment. Because of these two considerations, the coverage I was able to realize in the random testing was significantly greater than the coverage of my unit tests, which were largely incomplete.

Having said that, I will say that I found this random testing experience to be more foreign to my professional experience of testing which has tended to be mostly functional, rather than structural, more at the level of the system than the unit and, while those comments apply both to the structural unit test of the previous assignment and to this random testing assignment, I hasten to add that the unit testing of last assignment was not unfamiliar to me, while the random testing of this assignment (and the lectures that preceded it) was more so, and the random testing more significantly broadened my perspective on testing.

Running the Tests

To run the test, simply execute the following command:

```
$ make randomtest
```

Repository

The source for this project is available at https://github.com/bribru/CS362-004-W2018/tree/bruckneb-assignment-4b/projects/bruckneb/dominion