Eddie Kovsky
Assignment 3
CS362-W2019

Random Testing

I tested the Adventurer card by using cardEffect() to play the Adventurer card and measure how that changes the game state. Specifically, I exercised this branch in playAdventurer() to see how it changed the total treasure count in the current game:

```
if (cardDrawn == copper || cardDrawn == silver || cardDrawn == gold) {
        drawntreasure++;
```

As for how many iterations are sufficient for the random test, I was unable to determine a scientific answer. 2000 loops seems to be the sweet spot.

Code Coverage

Achieving 100% code coverage for playVillage() and playGreatHall() was fairly straight forward. These two functions can only change the game's card count and number of actions.

The Adventurer card doesn't have a *direct* method to test, but the card's effect on the game's treasure count was easy enough to measure. This brought the test coverage for the Adventurer card to 93.75%. With an additional test to verify the discardCount I expect I could have achieved 100% test coverage for the Adventurer card as well.

Unit vs Random

Unit testing seemed to uncover more bugs than random testing. It's also easier to target unit testing when you have prior knowledge of how the code should work and what can never fail. Random "blackbox" testing can certainly be useful for testing inputs that may not occur to the developer (or worse, inputs that the developer is skipping because they know it will break their code) but it is not particularly rigorous.

One problem that continues to drag down these assignments is the C standard library's lag of useful assert functionality. Crashing the program when a test fails with assert() is not acceptable for our use case, but creating a fully featured replacement could easily be its own course.