Logan Wingard

11/1/17

CS_362_400_F2017

Assignment-4

Random Testing

I began by using the code from in lecture ass reference, so I set up a seed and randomly generated the gamestate and number of players. I then used these random gamestates and players to execute the card functions: adverntture() smith() and vil(). It will either break(in the case of adventure) or say how many times out of 2000 it succeeded and how many it failed.

Code Coverage

Because I lost points for not stating this last time:

to run the coverage commands:

```
randomtestadventurer.out
randomtestcard1.out
randomtestcard2.out
```

you must first do run the following commands:

```
make clean (if you haven't already)
make playdom
./playdom 10
make randomtest*.out
```

This will write to the corresponding out file.

For every single one of my random testers, it covered every single line of code in each of the corresponding functions, except for one line in smith() and one line in vil(). The adventure() function covered 100% of its branches. the lines that were not executed in smith and vil were both

```
nextPlayer = 0;
```

This line happens in an if statement where if the next player is out of the array of total players, it sets the next player to player 0.

The total lines executed in the entire dominion.c program is 48.96%

Unit vs Random

Coverage-wise, Random tests covered much more of the code than the unit tests. According to the .out files, random tests covered almost 49% of dominion.c while unit tests only covered around 18%. Unit testing, I have found needs to be more planned out, where you need to make sure you pass values that will cover as many branches of the code as possible, while in random testing, you just let it do its thing and it either succeeds or doesn't. Then you can use a different seed to get completely new random tests. Fault detection definitely goes to Unit tests, as you are able to tell what exact values you passed in that made it fail. Using seeds allows you to get the same exact inputs when random testing to make it easier to detect which values cause a break, but unit testing simplifies this very much so.