Testing procedure

I ran 'testdominion.c' on each file for multiple tests with the same random seeds. 'testdominion.c' chooses a random number of players between 2 and 4 and a random set of 10 Kingdom cards from the 19 cards implemented. It uses these values and the random seed to initialize the game.

The main code is based on playdom.c and consists of a while loop that constantly checks for game over conditions between turns. For each turn the tester iterates through the player's hand and tries to play each card with randomized choices. It then purchases in the following order: Gold, silver, random card, Province.

My first version of the tester tried to purchase a random card before anything else, but I discovered that some seeds caused the game to be gold starved and last an exponentially long time. I then tried to do the random card purchasing last, but the game did not last long enough to get good coverage (only 40-45% coverage no matter how many tests were run).

My Code

Coverage

Running 1 test: 60.59% of 576

Running 10 tests: 83.33% of 576

Running 100 tests: 87.67% of 576

Running 1000 tests: 88.02% of 576

After running 1000 tests I looked at the .gcov file to see which areas of code were being ignored so I could increase my test coverage.

Function	Fix
newGame()	Call newGame to create my gameState instead of creating it in the tester
kingdomCards()	Pass my kingdom Cards into the function to be packaged
getWinners()	Call getWinners() at the end of each game

After implementing these changes coverage after running 1000 tests increased to 97.57% of 576.

Results

When doing running testdominion.c I found that some seeds caused an infinite loop to occur. The first thing I did was narrow down the cause of the issue by commenting out parts of the code until the issue no longer occured. This showed that the inifnite loop was happening when I was trying to play one of the cards in my hand.

Knowing that it was probably a card problem I noted down the seeds that caused the infinite loop and the seeds that ran to completion so that I could compare the Kingdom cards that were chosen in each iteration. I found out that Feast was a common card to all the failing test cases and was not present when the testing was successful.

Turning on DEBUG showed that the loop was printing:

> "That card is too expensive!
> Coins: 5 < 8"

Looking at the Feast code shows that if the user submits a card choice that is too expensive, Feast will never break the loop and will keep trying to buy that card. I added a return -1 to the if statement so that if the player chooses a card that is too expensive the card will not do anything and allow the player to retry. (Returning -1 tells playcard not to decrease actions).

Further testing showed another bug causing an infinite loop printing “None of that card left, sorry!” I searched for the error statement and found that the offending card was Feast again. I employed the same fix described above so that if a player tries to buy a card that is not available Feast will fail and not change the gamestate, allowing the player to try again (Return -1).

Classmate 1: zakrevsm

Coverage

I chose to remove Feast from play to avoid the errors described in my own code above. This decreased the coverage slightly.

Running 100 tests: 93.46% of 581

Results

The code did not crash with over 90% coverage. I believe this code is reliable.

Classmate 2: alzamilb

Results

Core dump, unreliable.

Classmate 3: amidong

Coverage

I chose to remove Feast from play to avoid the errors described in my own code above. This decreased the coverage slightly.

Running 100 tests: 93.74% of 575

Results

The code did not crash with over 90% coverage. I believe this code is reliable.

Classmate 4: boringk

Coverage

I chose to remove Feast from play to avoid the errors described in my own code above. This decreased the coverage slightly.

Running 100 tests: 93.62% of 580

Results

The code did not crash with over 90% coverage. I believe this code is reliable.

Overall Experience

Being an ECE major this class was very different from what I am used to. Instead of endless equations and circuit analyses I had a lot of fun coding and trying to figure out what the assignments actually meant. I felt that the class was a lot of fun and I actually learned more than I was expecting to. Not knowing Python made the lectures difficult to understand, but it was cool to see what was possible with the higher level languages. I really enjoyed learning how to use gcov and seeing the graphs of coverage information. That is definitely a tool I want to try to use in the future. I have a hard time finally accepting that projects, especially coding projects, are 'done.' This is mainly do to the fact that I could never be sure that some obscure bug wouldn't pop up in some random part of the code. Gcov allows me to visualize exactly what is running and what isn't so I can better test all aspects of a program.

I also think I might be able to apply the principles of random testing to my hardware troubleshooting too. For example I just finished a project that occasionally decided to stop working whenever power was cycled. If I had a programmable power supply I could randomly turn off power to it to simulate random power failures. Then if I found an issue I could iteratively go through each function and turn off the power at a specified time, aka unit testing. Overall I'm glad I took this class even though it does not directly apply to my major. I believe the skills learned are valuable to ay discipline and the university should consider making a similar course required for engineering disciplines as well.