Testing dominion was a bit of a pain. It was hard because of a few reasons,
One: It wasn't really a realiably even semi working piece of code
Two: It was a large program and so it was difficult to pinpoint where things could be going wrong.

With that being said however; if I was working out in the real world dealing with software much older than me. Than it's what I get and I better be ready to write tests that would make sure software that could be deployed out to millions of people works the way it is supposed to. In all of my testing I've achieved over 95% code coverage the reason is because there were statements that were not executed because they didn't meet the flow criteria. However this isn't a great representation of how the program is being operated as a whole. I felt as though while dominion was a relatively easy to grasp game. It still had a lot of smaller parts to it that needed to interact with each other and so the tests that I performed I think poorly reflected how dominion operated as a whole
Out of everything, I believe my game generator was probably the best written assignment I had. This is because by the time I got to this assignment I was pretty comfortable with the game and therefore could write code better. The problem is that the testdominion game didn't exactly test if the whole game worked correctly and more was just seeing if it executed at all. This left me with tests that I had to use from early on that were weak because of my inexperience with the program.
This was frustrating because I was pretty interested in testing but felt that we could have written tests
for a smaller program than I would have better grasp on how to implement unit tests better.
What to test on
instead of just return values.