CS362-004

Assignment-3:

The primary goal of this assignment is to learn how to create unit tests.

Note:

- This is NOT part of the team project. Do it on your own!
- Make all your changes on your master branch, then create your new branch.
- You are required to submit a **unit test** not a **random test generator**. **50% off** for submitting a random test generator!

Assignment Details:

1- Write unit tests for **four** functions (not card implementations, and not cardEffect) in **dominion.c** "the refactored code you created for assignment-2". Check these tests in as **unittest1.c**,**unittest2.c**, **unittest3.c**, and **unittest4.c**. At least two of these functions should be more **than 5 lines of code**. (**20 points**)

2- Write unit tests for **four** Dominion cards implemented in **dominion.c** "the refactored code you created for assignment-2". *Write these tests so that they work whether a card is implemented inside cardEffect or in its own function*. These tests should be checked in as **cardtest1.c**, **cardtest2.c**, **cardtest3.c**, and **cardtest4.c**. It is mandatory to test **smithy** and **adventurer card**. (**20 points**)

Writing your results:

- Execute your unit tests and describe any bugs you find in a section named **Bugs.** (**10 points**)
- Use gcov to measure code coverage for all of these tests. Report your findings by discussing your tests' coverages (statement, branch, boundary, etc.), and describe their implications for the tests in a section called **Unit Testing**. I want you to look at the dominion code coverage and find out what parts of your code are not covered so that in future you can improve your test suite. (**30 points**).
- Discuss your unit testing efforts in a section called **Unit Testing Efforts**. (**10 points**)
- Add a rule in **Makefile** that will generate and execute all of these tests, and append complete testing results (including % coverage) into a file called **unittestresults.out**. The rule should be named **unittestresults.out** and should depend on all your test code as well as the dominion code. The .out files contain the output of your running tests and coverage information. Basically .out file should act as a proof that your tests run correctly and you collected coverage information correctly. (**10 points**)

Helpful hint:

For the unit tests, you can pick the size but I suggest trying to produce enough unit tests that it is

plausible some developer would view this as an"ok" suite for making sure dominion isn't broken.
To avoid making it hard to collect coverage when a test fails, use your own asserttrue function instead of the standard C assert (which basically crashes the code and fails to collect coverage). Your assert can also print out more information, and maybe have an option that controls whether it exits the program or not. In these and other tests, I find it helpful to make all tests print "TEST SUCCESSFULLY COMPLETED" or some other message if and only if the entire test passes, and usually (this isn't always possible for crashing bugs) print "TEST FAILED" for a failure. This makes it easy to process failing and passing tests.

Submission instructions:

- **Canvas**
  - **Assignment-3.pdf** that contains three sections: **Bugs, Unit Testing, and Unit Testing Efforts**.

- **The class github repository**
  - Submit your complete dominion code under **projects/your-onid/dominion.**
  - Create a new **branch** of your repository called "**youronid-assignment-3**" contains your final submission. This branch must be created before the due date to receive credit.

** Add a comment in **Canvas** and give the URL for your fork (under Assignment-3).