Arthur Liou
CS362

Assignment 2

1) Make sure your changes from the Assignment 2 branch are merged into Master. Create a new branch from Master for Assignment-3 called "youronid-assignment-3 "

– Done. Branch is "931989226-assignment-3".

2) 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)
unittest1.c = initializeGame() – Valid Start
unittest2.c = initializeGame() – Invalid Starts
unittest3.c = getCost() - multiple Cards
unittest4.c = isGameOver()

3) Write unit tests for four Dominion cards implemented in dominion.c. 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)
cardtest1.c = Smithy
cardtest2.c = Adventurer
cardtest3.c = Village
cardtest4.c = Remodel

Writing Your Results

Bugs

***Copied from the 4 Bugs I Created in Assignment 2
Format: Card – Bug Description (Line #). Bold is mandatory
1. Smithy – Draw 4 instead of 3 (655)
2. Adventurer – Excluded Gold from Treasure Drawn (674-675)
3. Villager – Added 3 Actions instead of 2 (695)
4. Garden – Allowed a Player to Draw a Card (731-733, 861-862)
***
Bugs found during testing: Smithy draws 4 cards, Smithy – deck count of 0 instead of expected 2 (have 5 in hand, draw 3, expected 2). Adventurer – incorrect coin count, incorrect tnumber of treasure cards added, state change for other player. Village – didn't draw card, too many actions. Remodel – did not discard 2

Unit Testing: Code Coverage / gcov

Prompt: Discuss code coverage (statement, branch, boundary, etc) and their implications.

Coverage: Copied from unittestresults.out. In the course of my unit tests, it was a mix of coverages, looking through a handful of statements and branches, the boundaries of some statements. Implications to come further down below (in italics).

Results from running unittestresults / unittestresults.out

```
unittest1.c:
Unit Test 1: Starting a game of Dominion.
Unit Test 1: Test Complete - SUCCESS.
File 'dominion.c'
Lines executed:16.56% of 646
dominion.c:creating 'dominion.c.gcov'

unittest2.c:
Unit Test 2: initializeGame() function Specifics and Edge Cases.
Unit Test 2: Test Complete - SUCCESS - Number of Players Cannot be 1.
File 'dominion.c'
Lines executed:16.72% of 646
dominion.c:creating 'dominion.c.gcov'

unittest3.c:
Unit Test 3: getCost() Function on various cards.
Unit Test 3: Test Complete - SUCCESS.
File 'dominion.c'
Lines executed:17.65% of 646
dominion.c:creating 'dominion.c.gcov'

unittest4.c:
Unit Test 4: Checking isGameOver().
isGameOver Test 1 - Province
Unit Test 4: Province - Test Complete - SUCCESS.
isGameOver Test 1 - 3 Empty Stacks
Unit Test 4: 3 Stacks - Test Complete - SUCCESS.
File 'dominion.c'
Lines executed:19.35% of 646
dominion.c:creating 'dominion.c.gcov'

cardtest1.c:
Card Test 1: smithy
Card Test 1A: Receive Exactly 3 Cards. hand count = 8, expected = 7
Card Test 1B: Remove 3 Cards from own Deck. deck count = 0, expected = 2
Card Test 1C: No State Change for Other Players = 1
Card Test 1D: No State Change for Victory Card Piles and Kingdom Card Piles = 0

 >>>>> Card Test 1 Complete - smithy <<<<<

File 'dominion.c'
Lines executed:26.78% of 646
```

```
dominion.c:creating 'dominion.c.gcov'

cardtest2.c:
Card Test 2: adventurer
Card Test 2A: Receive at least 2 Coins. Coin count = 4, minimum expected = 6
Card Test 2B: Hand Gets +2 Treasure Cards. Hand count = 9, expected = 7
Card Test 2C: No State Change for Other Players = 0
Card Test 2D: No State Change for Victory Card Piles and Kingdom Card Piles = 0

 >>>>> Card Test 2 Complete - adventurer <<<<<

File 'dominion.c'
Lines executed:31.42% of 646
dominion.c:creating 'dominion.c.gcov'

cardtest3.c:
Card Test 3: village
Card Test 3A: +1 Card. hand count = 5, expected = 6
Card Test 3B: +2 Actions. action count = 4, expected = 3
Card Test 3C: -1 Deck. deck count = 4, expected = 4
Card Test 3D: No State Change for Other Players = 1
Card Test 3F: No State Change for Victory Card Piles and Kingdom Card Piles = 0

 >>>>> Card Test 3 Complete - village <<<<<

File 'dominion.c'
Lines executed:31.42% of 646
dominion.c:creating 'dominion.c.gcov'

cardtest4.c:
Card Test 4: remodel
Card Test 4A: Hand Count -2. hand count = 5, expected = 3
Card Test 4B: Bought a Village. village supply count = 9, expected = 9
Card Test 4C: No State Change for Other Player = 1
Card Test 4D: No State Change for Victory Card Piles and other Kingdom Card Piles = 0

 >>>>> Card Test 4 Complete - remodel <<<<<

File 'dominion.c'
Lines executed:35.60% of 646
dominion.c:creating 'dominion.c.gcov'

******End unittestresults.out
```

Implications: The implications of the results of my gcov is that my unit tests cover only a handful of the dominion file. Anywhere from very simple unit tests covering ~17-20% of the file to the card tests that covered up to 31%. Total gcoverage was at 35%. Considering the granularity of the functions and cards we were testing with these 8, I'm a little surprised at such high a number as I expected something more sub-25%. So, looking forward, if I wanted to get greater coverage and/or focus on statement or bramnch coverage, I would want more leeway and specificity in terms of the unittest 1-4 (see next section below for more details). I appreciate that for the card tests, they covered nearly 1/3 of the test file.

What parts of code are not covered: To simplify this part of my response, all functions and cards not invoked in any of these unit tests.

As I will mention below under "Unit Testing Efforts", I would want better categorization of each test file. Ex. a unitTest file for all aspects of a function. This kind of categorization and focus on creating a greater number of unit tests around that category would take a much more massive effort due to the scale I was thinking of. Like 10-15 tests per file, a qualified QA team who can "peer program/QA" and identify the correct types of use cases and test files to include. However, since we're just learning the basics of testing, I'm assuming that you (the staff) are assuming we just have fundamentals down. IE. Learn how to walk before shooting for the moon.

For more of my thoughts, impressions, and conclusions, see note below.

Unit Testing Efforts:

My 4 Unit Tests Cover very simple functions of the application. initializeGame(), getCosts(), isGameOver(). As I worked in implementing these unit tests, I began wondering about how many asserts, or equivalent, I should be using. For example, when I was implementing a unit Test for initializeGame, I realized there were numerous aspects of the function I could unit test on: correct number of players, edge cases for invalid number of players, (in)correct number of provinces, number of kingdom cards, number of coins, making sure two of the kingdom cards were not the same kind, etc.

After doing some research (ie google "How many asserts should a unit test have?"), best practice was to have one assert per unit test, which is normal, but then I had to reconcile that the staff wanted a singular assert/unit test per file (hence the 1, 2, 3, 4), so I resolved to break up my original intent for unit testing for initializeGame() and instead took two of the simpler aspects to unit test and then chose two other functions. I tried to not dive too deep into those, as in the card tests, I would be able to write a more comprehensive set of unit tests for each card, rather than a 1x per function.

My 4 Card Tests cover a more comprehensive overlook of the card functions, such as mentioned by the professor on these pages and my comments for the function unit tests above:

- https://oregonstate.instructure.com/courses/1706563/pages/assignment-3-testing-something-vs-testing-for-business-rules?module_item_id=18416504 / cardtest4.c
- For example, the page specifically mentions the business rules that we would need to test for on the smithy card.

***Copied from Assignment 3 notes from instructor
Let us first start enumerating the **basic requirements of smithy**.
1. Current player should receive exactly 3 cards.

2. 3 cards should come from his own pile.
3. No state change should occur for other players.
4. No state change should occur to the victory card piles and kingdom card piles.
These are the basic requirements that your test must fulfill in order to receive a full grade.
***

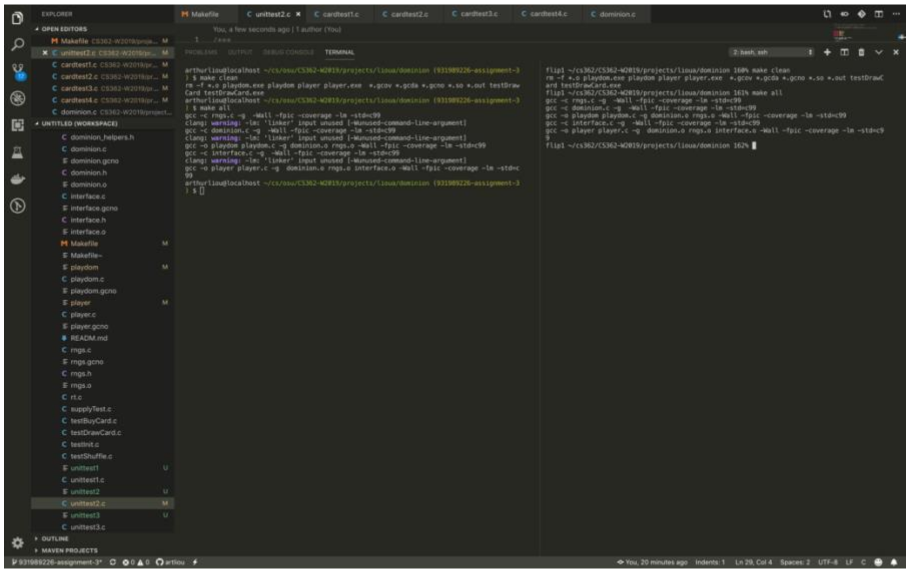
Thus, for my cardtest1.c, I checked off and fulfilled these four requirements. I based my other three cards tests off this template, so for the most part, they should look relatively similar. The differences would be any changes to game state depending on the card being testing. I tested all 4 cards I microserviced into separate functions, Smithy, Adventurer, Village, Remodel

Each cardtest invokes cardEffect(), so they fulfill this requirement: “Write these tests so that they work whether a card is implemented inside cardEffect or in its own function.”

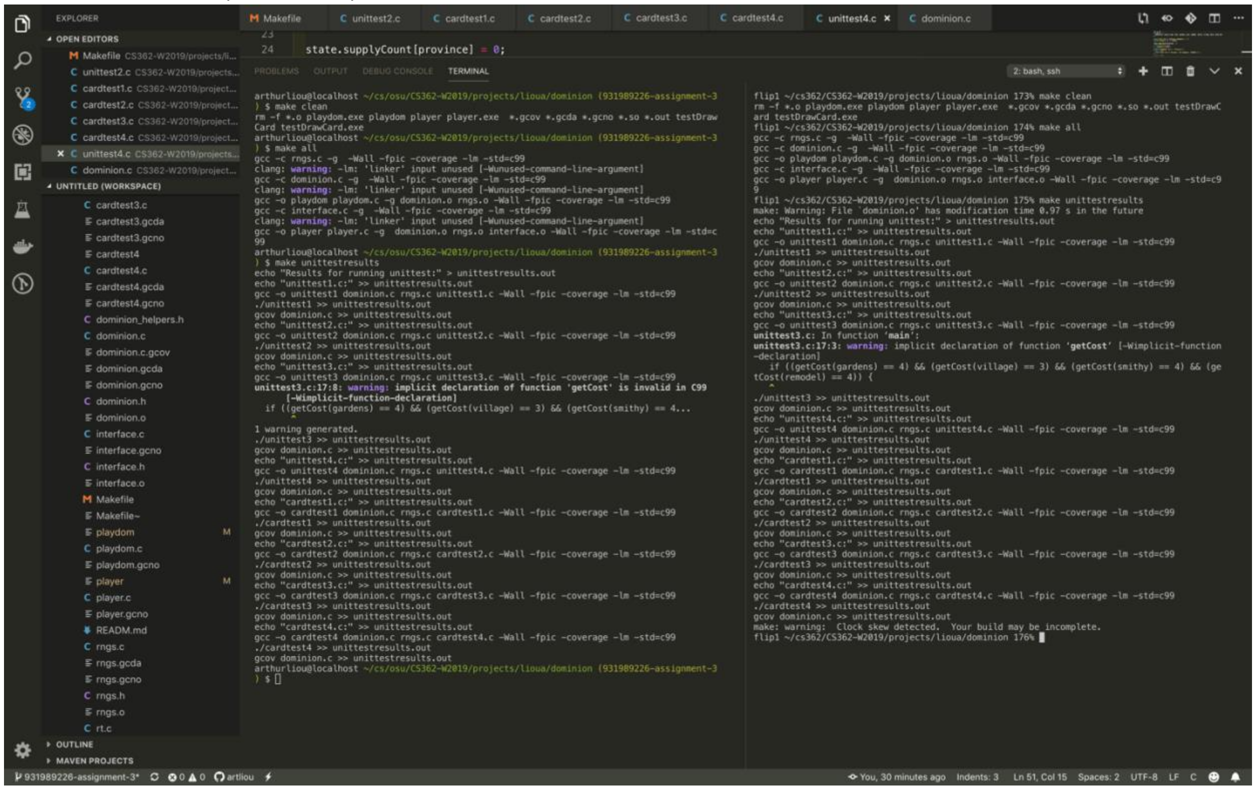
> Add a rule in Makefile (named unittestresults) that will generate and execute all of these tests, and append complete testing results (including % coverage) into a file called unittestresults.out. The rule should be named unittestresults and should depend on all your test code as well as the dominion 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)

Then I created rule in Makefile for “unittestresults”. I used the BST Makefile tests.out and used that template to create my rule for my MakeFile. From there, I would run ‘make unittestresults’ to create unittestresults.out

make unittestresults

To show that my program r
Cmd: Make Clean and Make All

```
arthurliou@localhost ~/cs/osu/CS362-W2019/projects/lioua/dominion (931989226-assignment-3
) $ make clean
rm -f *.o playdom.exe playdom player player.exe  *.gcov *.gcda *.gcno *.so *.out testDraw
Card testDrawCard.exe
arthurliou@localhost ~/cs/osu/CS362-W2019/projects/lioua/dominion (931989226-assignment-3
) $ make all
gcc -c rngs.c -g  -Wall -fpic -coverage -lm -std=c99
clang: warning: -lm: 'linker' input unused [-Wunused-command-line-argument]
gcc -c dominion.c -g  -Wall -fpic -coverage -lm -std=c99
clang: warning: -lm: 'linker' input unused [-Wunused-command-line-argument]
gcc -o playdom playdom.c -g dominion.o rngs.o -Wall -fpic -coverage -lm -std=c99
gcc -c interface.c -g  -Wall -fpic -coverage -lm -std=c99
clang: warning: -lm: 'linker' input unused [-Wunused-command-line-argument]
gcc -o player player.c -g  dominion.o rngs.o interface.o -Wall -fpic -coverage -lm -std=c
99
arthurliou@localhost ~/cs/osu/CS362-W2019/projects/lioua/dominion (931989226-assignment-3
) $ 
```
```
flip1 ~/cs362/CS362-W2019/projects/lioua/dominion 160% make clean
rm -f *.o playdom.exe playdom player player.exe  *.gcov *.gcda *.gcno *.so *.out testDrawC
ard testDrawCard.exe
flip1 ~/cs362/CS362-W2019/projects/lioua/dominion 161% make all
gcc -c rngs.c -g  -Wall -fpic -coverage -lm -std=c99
gcc -c dominion.c -g  -Wall -fpic -coverage -lm -std=c99
gcc -o playdom playdom.c -g dominion.o rngs.o -Wall -fpic -coverage -lm -std=c99
gcc -c interface.c -g  -Wall -fpic -coverage -lm -std=c99
gcc -o player player.c -g  dominion.o rngs.o interface.o -Wall -fpic -coverage -lm -std=c9
9
flip1 ~/cs362/CS362-W2019/projects/lioua/dominion 162% 
```

To show that my program runs and ‘make unittestresults’ works
Cmd: make clean, make all, make unittestresults



```
state.supplyCount[province] = 0;
```

```
arthurliou@localhost ~/cs/osu/CS362-W2019/projects/lioua/dominion (931989226-assignment-3
) $ make clean
rm -f *.o playdom.exe playdom player player.exe  *.gcov *.gcda *.gcno *.so *.out testDraw
Card testDrawCard.exe
arthurliou@localhost ~/cs/osu/CS362-W2019/projects/lioua/dominion (931989226-assignment-3
) $ make all
gcc -c rngs.c -g  -Wall -fpic -coverage -lm -std=c99
clang: warning: -lm: 'linker' input unused [-Wunused-command-line-argument]
gcc -c dominion.c -g  -Wall -fpic -coverage -lm -std=c99
clang: warning: -lm: 'linker' input unused [-Wunused-command-line-argument]
gcc -o playdom playdom.c -g dominion.o rngs.o -Wall -fpic -coverage -lm -std=c99
gcc -c interface.c -g  -Wall -fpic -coverage -lm -std=c99
clang: warning: -lm: 'linker' input unused [-Wunused-command-line-argument]
gcc -o player player.c -g  dominion.o rngs.o interface.o -Wall -fpic -coverage -lm -std=c
99
arthurliou@localhost ~/cs/osu/CS362-W2019/projects/lioua/dominion (931989226-assignment-3
) $ make unittestresults
echo "Results for running unittest:" > unittestresults.out
echo "unittest1.c:" >> unittestresults.out
gcc -o unittest1 dominion.c rngs.c unittest1.c -Wall -fpic -coverage -lm -std=c99
./unittest1 >> unittestresults.out
gcov dominion.c >> unittestresults.out
echo "unittest2.c:" >> unittestresults.out
gcc -o unittest2 dominion.c rngs.c unittest2.c -Wall -fpic -coverage -lm -std=c99
./unittest2 >> unittestresults.out
gcov dominion.c >> unittestresults.out
echo "unittest3.c:" >> unittestresults.out
gcc -o unittest3 dominion.c rngs.c unittest3.c -Wall -fpic -coverage -lm -std=c99
unittest3.c:17:8: warning: implicit declaration of function 'getCost' is invalid in C99
      [-Wimplicit-function-declaration]
  if ((getCost(gardens) == 4) && (getCost(village) == 3) && (getCost(smithy) == 4...
       ^
1 warning generated.
./unittest3 >> unittestresults.out
gcov dominion.c >> unittestresults.out
echo "unittest4.c:" >> unittestresults.out
gcc -o unittest4 dominion.c rngs.c unittest4.c -Wall -fpic -coverage -lm -std=c99
./unittest4 >> unittestresults.out
gcov dominion.c >> unittestresults.out
echo "cardtest1.c:" >> unittestresults.out
gcc -o cardtest1 dominion.c rngs.c cardtest1.c -Wall -fpic -coverage -lm -std=c99
./cardtest1 >> unittestresults.out
gcov dominion.c >> unittestresults.out
echo "cardtest2.c:" >> unittestresults.out
gcc -o cardtest2 dominion.c rngs.c cardtest2.c -Wall -fpic -coverage -lm -std=c99
./cardtest2 >> unittestresults.out
gcov dominion.c >> unittestresults.out
echo "cardtest3.c:" >> unittestresults.out
gcc -o cardtest3 dominion.c rngs.c cardtest3.c -Wall -fpic -coverage -lm -std=c99
./cardtest3 >> unittestresults.out
gcov dominion.c >> unittestresults.out
echo "cardtest4.c:" >> unittestresults.out
gcc -o cardtest4 dominion.c rngs.c cardtest4.c -Wall -fpic -coverage -lm -std=c99
./cardtest4 >> unittestresults.out
gcov dominion.c >> unittestresults.out
arthurliou@localhost ~/cs/osu/CS362-W2019/projects/lioua/dominion (931989226-assignment-3
) $ 
```

```
flip1 ~/cs362/CS362-W2019/projects/lioua/dominion 173% make clean
rm -f *.o playdom.exe playdom player player.exe  *.gcov *.gcda *.gcno *.so *.out testDrawC
ard testDrawCard.exe
flip1 ~/cs362/CS362-W2019/projects/lioua/dominion 174% make all
gcc -c rngs.c -g  -Wall -fpic -coverage -lm -std=c99
gcc -c dominion.c -g  -Wall -fpic -coverage -lm -std=c99
gcc -o playdom playdom.c -g dominion.o rngs.o -Wall -fpic -coverage -lm -std=c99
gcc -c interface.c -g  -Wall -fpic -coverage -lm -std=c99
gcc -o player player.c -g  dominion.o rngs.o interface.o -Wall -fpic -coverage -lm -std=c9
9
flip1 ~/cs362/CS362-W2019/projects/lioua/dominion 175% make unittestresults
make: Warning: File `dominion.o' has modification time 0.97 s in the future
echo "Results for running unittest:" > unittestresults.out
echo "unittest1.c:" >> unittestresults.out
gcc -o unittest1 dominion.c rngs.c unittest1.c -Wall -fpic -coverage -lm -std=c99
./unittest1 >> unittestresults.out
gcov dominion.c >> unittestresults.out
echo "unittest2.c:" >> unittestresults.out
gcc -o unittest2 dominion.c rngs.c unittest2.c -Wall -fpic -coverage -lm -std=c99
./unittest2 >> unittestresults.out
gcov dominion.c >> unittestresults.out
echo "unittest3.c:" >> unittestresults.out
gcc -o unittest3 dominion.c rngs.c unittest3.c -Wall -fpic -coverage -lm -std=c99
unittest3.c: In function ‘main’:
unittest3.c:17:3: warning: implicit declaration of function ‘getCost’ [-Wimplicit-function
-declaration]
   if ((getCost(gardens) == 4) && (getCost(village) == 3) && (getCost(smithy) == 4) && (ge
tCost(remodel) == 4)) {
   ^
./unittest3 >> unittestresults.out
gcov dominion.c >> unittestresults.out
echo "unittest4.c:" >> unittestresults.out
gcc -o unittest4 dominion.c rngs.c unittest4.c -Wall -fpic -coverage -lm -std=c99
./unittest4 >> unittestresults.out
gcov dominion.c >> unittestresults.out
echo "cardtest1.c:" >> unittestresults.out
gcc -o cardtest1 dominion.c rngs.c cardtest1.c -Wall -fpic -coverage -lm -std=c99
./cardtest1 >> unittestresults.out
gcov dominion.c >> unittestresults.out
echo "cardtest2.c:" >> unittestresults.out
gcc -o cardtest2 dominion.c rngs.c cardtest2.c -Wall -fpic -coverage -lm -std=c99
./cardtest2 >> unittestresults.out
gcov dominion.c >> unittestresults.out
echo "cardtest3.c:" >> unittestresults.out
gcc -o cardtest3 dominion.c rngs.c cardtest3.c -Wall -fpic -coverage -lm -std=c99
./cardtest3 >> unittestresults.out
gcov dominion.c >> unittestresults.out
echo "cardtest4.c:" >> unittestresults.out
gcc -o cardtest4 dominion.c rngs.c cardtest4.c -Wall -fpic -coverage -lm -std=c99
./cardtest4 >> unittestresults.out
gcov dominion.c >> unittestresults.out
make: warning:  Clock skew detected.  Your build may be incomplete.
flip1 ~/cs362/CS362-W2019/projects/lioua/dominion 176% 
```

To conclude this assignment and this section on effects, I found this assignment very illuminating for starting to learn testing and gcov. I look forward to HW4 and HW5 when we dive deeper into developing these tests and have greater test coverage for the dominion.c game.