For my first test I tested a random classmate’s dominion.c file. The ONID of the user randomly chosen was sudarsar(Ramchuran Sudarsanam). Their Dominion.c was downloaded and processed through each of the tests I had previously created to test my own dominion code, the first of which was the code for each unit test. These unit tests tested 4 different functions within the dominion.c code.

The first unit test tested the getcost function. The test polled the getcost function for each card integer ID, then compared it to the suspected value. This test passed flawlessly as each cost associated with each card was returned correctly.

The second unit test tested the kingdomcard function. The purpose of this function is to accept ten cards and generate a new array containing one of each of these ten cards in each of its elements. For this test 10 random numbers were generated and were placed in an initial array filling it fully. Then the same numbers were sent into the kingdomcard function along with an empty array. Then each element of the initially created array and the array passed to the function are compared. In this instance when testing Ramchuran’s dominion.c code the function worked perfectly as expected.

The third unit test tests the compare function. The tester runs a number of tests to tests the function. It runs a for loop for a series of onehundred tests. In each iteration, it generates two random integer pointers and compares them internally thereby determining the expected output. Then it passes the two integer pointers to the compare function and compares its output to the expected ouput. In all 100 cases the given dominion.c code tested flawlessly.

The fourth unit test tests the newgame function. The unit test tests the newgame function by equating an uninitialized gamestate to the newgame function. It then tests the initialization by sending the now ininitialized gamestate to the initializegame function and seeing if the initializegame function returns the proper integer flag. In this case, the given dominion.c code tested flawlessly.

The next set of tests I ran were the four card tests. The first of these tests tested the outpost card. The test ininitialized a newgame with a static set of parameters. After this set of commands an assertion was made to test that the outpost flag was not yet set. Then after this assertion, the cardeffect function was executed with the paramter of the outpost card. Then, after all this the assertion was made to test that the outpost flag was then set. When testing the given dominion.c code, the test passed flawlessly.

The second card test tested was the smithy card. The test ininitialized a newgame with a static set of parameters. After this set of commands an assertion was made to test that the current hand size was the default 5 cards. Then after this assertion, the cardeffect function was executed with the paramter of the smithy card. Then, after all this the assertion was made to test that the hand size had increased to 8. When testing the given dominion.c code, the test passed flawlessly.

The third card test tested was the council room card. The test ininitialized a newgame with a static set of parameters. After this set of commands a looping assertion was made to test that the current hand size of each other player was 0, and a singular assertion for the current player was made to show that their hand size was 5. Then after this assertion, the cardeffect function was executed with the paramter of the council_room card. Then, after all a looping assertion was made to test that the current hand size of each other player was 1, and a singular assertion for the current player was made to show that their hand size was 8. Then the assertion was made to show that the number of buys for the current player had increased by one to two. When testing the given dominion.c code, the test passed flawlessly.

The fourth and final card test tests the gardens card. As before, the test ininitialized a newgame with a static set of parameters. The card effect was tested and its output was asserted to show that card effect ran properly and that there were no errors. When run against Ramchuran's dominion.c code I found no errors and that all the tests passed.

The next set of tests ran against the code was the series of random tests suites I had created for the third assignment. The first random test was a test for the salvager card. This test starts by ininitializing a newgame with a static set of parameters. Then it populates a hand with one of every card. Then it finds the initial money for the player. Then it calculates the expected money for after the salvager card has been played if discarding a selected random card. It then calls the cardeffect function passing the parameters of the salvager card and the chosen random card. It then finds the final money for the player and asserts it against the expected money. After which, it also checks the size of the hand, and discard pile. When running this test against the dominon.c file, all assertions were met except the fact that the discard pile size did not increase from 0 to 1.

The second random test tested the gardens card and the scorefor function when a garden card was possessed by the player. To begin the test ininitializied a newgame with a static set of parameters. Then it resizes the deck to a random size and populates the entire deck with copper cards. Then it replaces the hand with a single garden card and calls the card effect of gardens. After asserting that the card effect function exited properly, it calls the scorefor function and asserts this against the expected score of the current deck. For this test, the returned score was not equal to the expected score. This is due to the fact that the scorefor function only counts curse cards towards the total size of the deck.

The last random test tests the adventurer card. To begin the test initialized a newgame with a static set of parameters. It then asserts the initial hind size to be 5. Then it populates the deck to be all estates aside from two coppers. Then it calls the cardeffect function with the adventurer parameter. It then asserts that the final hand size, money, and discarded card pile size are all the correct size. When run against the dominion.c code all tests passed.

Similarly, when these tests were run on another random classmate's dominion.c code, namely the code of chani(Isaac Chan) it found similar results in most facets. In most tests we found the exact same bugs and passing tests as before with the other dominion.c file, aside from one test. This difference was the appearance of a new failing test, more specifically the failing of the second card test. This test was the test for smithy. In this test we found that the player did not draw their expected three cards, but instead drew only two cards, thus failing the test.

Overall the testing of both dominion.c files has been spotty at best, however the tests have proven to reveal a few bugs for each file and are therefore quite successful. The coverage in either case is rather poor since only a select group of functions and card effects are covered, but each are covered quite nicely. All in all, the tests proved successful and the newfound bugs can be fixed in no time.