Garrett Amidon

CS362

Testing

Throughout this term, testing dominion has proven to be extremely beneficial in my debugging and coding skills. To start, when I run my tests on Oatmanm's dominion.c and Maddens' dominion.c file the results are shown below:

Unittest1:

Maddens: 0.86% of 581

Oatmanm: 0.85% of 586

Mine: 0.87% of 575

Unit1: Checking the function supplyCount and making sure the right number of a certain card is returned correctly. Considering this function isn't a very big part of the game, which is why the percentage is so low.

Unittest2:

Maddens: 0.52% of 581

Oatmanm: 0.51% of 586

Mine: 0.52% of 575

Unit2: Checking the compare function. It tests all 3 cases (greater than, less than, and equal to) and makes sure the correct number is returned. Again this function isn't used very much throughout the game, which again is why the percentage is so low.

Unittest3:

Maddens: 2.24% of 581

Oatmanm: 2.23% of 586

Mine: 2.26% of 575

Unit3: Checking the kingdomCards function. It asserts to make sure that all the integers that are passed are place in the correct spot of the array passed with it. This function is used more frequently than the other 2 functions from above, so therefore the percentage is higher.

Unittest4:

Maddens: 0.86% of 581

Oatmanm: 0.85% of 586

Mine: 0.87% of 575

Unit4: Checking the whoseTurn function. It sets the state turn to player one then runs the function. It then asserts to make sure that the whoseTurn function returned 1. Whoever wrote this code wrote this function, but they never really used the function and just called directly to state-whoseTurn, which is why the percentage is so low.

Cardtest1:

Maddens: 16.01% of 581

Oatmanm: 15.87% of 586

Mine: 16.17% of 575

Card1: Checking the gardens call through cardEffect. Considering this card only returns -1, the function really only checks to make sure that -1 is returned and that none of the values inside of the gamestate are changed. Considering gardens is higher up on the switch statement in cardEffect and also doesn't have much functionality, it doesn't take too much of the coverage.

Cardtest2:

Maddens: 20.31% of 581

Oatmanm: 20.31% of 586

Mine: This is where I intentionally put one of my errors, return 1 instead of 0.

Card2: Checking the great_hall call through cardEffect. The test checks the gamestate's variables before and after the card is called. After the great hall is called, it should increase the number of actions and return 0. Considering there is more to the great hall then there is with gardens, there is more coverage.

Cardtest3:

Maddens: 20.14% of 581

Oatmanm: 20.14% of 586

Mine: 20.35% of 575

Card3: Checking the outpost call through cardEffect. The test checks the gamestate's variables before and after the card is called. After the outpost is called, it should increase the number of outposts played, decrease the number of cards in the hand and return 0. Considering there is more to the outpost and is about the same length as great hall and more going on then there is with gardens, there is more coverage.

Cardtest4:

Maddens: As found earlier, there is a bug inside of the village call. The number of actions should increase by 2 but instead decreases by 1.

Oatmanm: 20.31% of 586

Mine: 20.52% of 575

Card4: Checking the village call through cardEffect. The test checks the gamestate's variables before and after the card is called. After the village is called, it should increase the number of actions by 2 and return 0. Considering it is about the same length as great hall and outpost and there is more going on then there is with gardens, there is more coverage.

Update so far:

Up to this point, these tests are checking very basic calls and effects on dominion. For the code to be somewhat reliable, it would seem that all of these tests should pass. Considering Maddens' village has a bug in it, I would have to say his dominion.c file is not very reliable. As for Oatmanm's dominion.c, there are no errors yet, so of the tests ran, the file seems to be reliable. Besides the errors that were found, it seems like all the percentages are in the same ball park with a little difference because the different amount of line numbers. For examples, I have fewer lines then the other 2 students dominion.c file. So when I run the coverage, we are checking the same number of lines, but since my code is shorter, it gets a greater percentage.

RandomTestAdventurer:

Maddens: 24.96% of 581, ran 25 times.

Oatmanm: Seg faults on first run.

Mine: 24.00% of 575

RandomTestCard:

Maddens: 18.93% of 581, ran 25 times.

Oatmanm: 18.43% of 586, ran 25 times.

Mine: 19.13% of 575

TestDominion:

Maddens: 71.77% of 581, ran 50 times, broke 23 times.

Oatmanm: 75.26% of 586, ran 50 times, broke 27 times.

Mine: 77.91% of 575, ran 50 times, broke 21 times.

From running randomtestadventurer, there seems to be a very crucial error in Oatmanm's adventurer cardeffect, and when running with random seed and settings, it seg faults. Yet, it tested for more coverage on the testdominion.c. I concluded that neither of these dominion.c files seems to be the most reliable. Considering the settings are random each time the testdominion test is ran, I can't narrow down which cards are causing the error in each students dominion.c file. If I had to choose whose dominion file seems to be more reliable I would go with Madden's. I concluded this because although when running the whole dominion test, they got about 4% higher than Maddens, it also broke more. The breaks were only ran when the gamestate lost track of whose turn it was. Although my code did cover more of Oatmanm 's dominion file, he still has that crucial error in adventurer. Whereas Madden's has a numActions error with village, which whould make his turn run short, it wouldn't cause the program to break completely. Considering I made the changes in my dominion.c file, it was easier to make my testdominion.c according to my dominion.c file. So therefore, the coverage should be greater on mine then theirs. But the coverage numbers are pretty close to mine, which shows that the test was written well and the main differences are going to be how we refactored the code. For instance, if they refactored a card effect and that card isn't one of the ones that get played, the coverage will be less on theirs.