Khoa Nguyen
CS 362
02-04-2018

Assignment 3

I. Bugs

unitTest1.c: gainCard()
-No bugs found.

unitTest2.c: isGameOver()
-No bugs found.

unitTest3.c: fullDeckCount()
-No bugs found.

unitTest4.c: getCost()
-No bugs found. I took the costs from http://dominion.diehrstraits.com, which is referenced in dominion.h to have been used by the developer.

cardTest1.c: cardEffect() for smithy
-The function does not correctly increase handCount of the player by 2 after drawing 3 cards and discarded smithy.
-The function does not correctly decrease deckCount of the player by 3 after drawing 3 cards.
=>The above 2 bugs seem to be caused by the for loop in smithyEffect(), which is called by cardEffect(), not properly iterating.

cardTest2.c: cardEffect() for adventurer
-The function does not correctly increase handCount of the player by 1 after drawing 2 cards and discarded adventurer.
-The function does not correctly place adventurer into the played pile
-The function does not correctly increase playedCardCount by 1
-The function does not correctly decrease player's deckCount by 2
=>The above 4 bugs seem to be caused by: 1- the while loop in adventurerEffect(), which is called by cardEffect(), not properly iterating; and 2- the adventurer card is not discarded after play.
-Also, I have found another bug that will be discussed below in Unit Testings - cardtest2 section below.

cardTest3.c: cardEffect() for steward
-No bugs found.

cardTest4.c: cardEffect() for great_hall
-No bugs found.

II. Unit Testings

unittest1 - gainCard()

gainCard() coverage
Lines executed:100.00% of 15
Branches executed:100.00% of 6
Taken at least once:100.00% of 6
No calls

unittest2 - isGameOver()

isGameOver() coverage
Lines executed:100.00% of 12
Branches executed:100.00% of 8
Taken at least once:100.00% of 8
No calls

unittest3 - fullDeckCount()

fullDeckCount() coverage
Lines executed:100.00% of 11
Branches executed:100.00% of 12
Taken at least once:100.00% of 12
No calls

unittest4 - getCost()

getCost() coverage
Lines executed:100.00% of 30
Branches executed:100.00% of 28
Taken at least once:100.00% of 28
No calls

Findings: My unit tests have total coverage of all tested functions and thus their results are a very good indicator of correct implementation. That said, I may have not covered all logics associated with the tested function and thus may have missed some important logics that the function may lack to execute correctly according to the business requirement.

cardtest1: cardEffect() calling smithyEffect()

smithyEffect() coverage
Lines executed:100.00% of 5

Branches executed:100.00% of 2
Taken at least once:100.00% of 2
No calls

cardtest2: cardEffect() calling adventurerEffect()

adventurerEffect() coverage
Lines executed:86.67% of 15
Branches executed:100.00% of 12
Taken at least once:75.00% of 12
No calls

Findings: This test has a reasonably high coverage of the FUT as all branches were covered but there were a couple lines that were not covered. When I inspected which lines were not covered, it appeared that my test didn't check the case when the player has 0 cards on their deck to draw. This is one of the scenarios I have actually written in the test for smithy but eventually commented it out as I thought that this scenario would better be suited for a unit test for the drawCard() function. I rechecked the code for adventurerEffect() and realized there is a major bug here as the shuffle() function was called within the branch which covers this empty-deck scenario instead of the drawCard(). So, I found this bug not by reading the result of my tests but by reading the code coverage. After I have added the scenario as Test Case 2, the Lines executed became 100% as expected (see below)

adventurerEffect() coverage
Lines executed:100.00% of 15
Branches executed:100.00% of 12
Taken at least once:83.33% of 12
No calls

cardtest3(): cardEffect() case steward

cardEffect() coverage
Lines executed:10.31% of 262
Branches executed:15.08% of 179
Taken at least once:5.03% of 179
No calls

Findings: it was harder to find the exact coverage stats for this test as the logics was implemented as a case within the cardEffect() function instead of a refactored function as in smithyEffect() and adventurerEffect() above. After this, I learned that having factored code will better serve me when it comes to coverage testing.

Since the stats was not immediately available, I had to look at the gcov output for dominion.c and found out that my test for steward card did have total coverage for this card function.

cardtest4(): cardEffect() case great_hall

cardEffect() coverage
Lines executed:10.31% of 262
Branches executed:15.08% of 179
Taken at least once:5.03% of 179
No calls

Findings: similar to cardTest3() it was harder to find the exact coverage stats for this test as the logics was implemented as a case within the cardEffect() function instead of a refactored function as in smithyEffect() and adventurerEffect() above.

Since the stats was not immediately available, I had to look at the gcov output for dominion.c and found out that my test for great_hall card did have total coverage for this card function.

III. Unit Testing Efforts

For all unit tests and card tests, I first inspected the implementation of the function in dominion.c and also looked at dominion.h and playdom.c occasionally for better understanding. Then I wrote my idea about what test scenarios I should have, their respective input and expected output. Then I implemented the test and double check if they perform as expected to find possible bugs. Moreover, when any of my tests failed, meaning there was a potential bug, I attempted to fix the bug in dominion.c to see if my tests became succeeded to rule out the possibility that my test implementation was actually buggy. That said, I understand that I may have not covered all scenarios in my tests and bugs may still exist undetected.

Specifics to each test are discussed below:

unitTest1.c: gainCard()
-Test case 1 and test case 2 is to check if the function performs as expected when the supply pile is empty or card is not used in game.
-Then there are three scenarios depending on the player's choice whether to add to discard, deck or hand. Each of these became one of my test case.
-I made sure that the supplyCount, handCount, deckCount, discardCount were updated according to the player's choice and that the correct card was added to the correct place. Also, I checked if all other player's handCount/deckCount/discardCount was not changed. Finally, I checked for the return value of the function.

unitTest2.c: isGameOver()
-I came up with different scenarios that this function may see and made sure to include some boundary cases

unitTest3.c: fullDeckCount()

-I wanted to check if this function could do correct count for more than 1 specific card and thus I developed 2 test cases, each of which do a full deck count of a different card.

-For each test case, I randomized handCount, deckCount, and discardCount 10 times and have the function run to get the full count to compare with my manual counts to make sure they matched.

unitTest4.c: getCost()

-This was a fairly straightforward test. I read dominion.h to find the source of the costs of the cards, which was http://dominion.diehrstraits.com. Then I basically made sure that the getCost() is returning matching costs to what is documented on the website.

cardTest1.c: cardEffect() for smithy

-As this card effect involves drawing cards, I thought we needed to check what happen when the player has enough card on their deck, and when they do not.

-The function should give the same result under the two scenarios as drawCard() should just transfer cards from the player's discard pile back to their deck.

-I made sure the handCount of the player is correct after the card is played. The smithy card is properly placed into the played pile. Also the function should increase playedCardCount and update deckCount correctly. Finally, other players' handCount and deckCount should not be affected.

cardTest2.c: cardEffect() for adventurer

-As this card effect involves drawing cards, I thought we needed to check what happen when the player has enough card on their deck, and when they do not.

-The function should give the same result under the two scenarios as drawCard() should just transfer cards from the player's discard pile back to their deck.

-I made sure the handCount of the player is correct after the card is played. The adventurer card is properly placed into the played pile. Also the function should increase playedCardCount and update deckCount correctly. Then, other players' handCount and deckCount should not be affected.

-Last but not least, the last 2 cards in the player's hand should be treasures.

cardTest3.c: cardEffect() for steward

-There are 3 scenarios based on 3 choices that player can make. For the first scenario when the player chooses to draw 2 cards, there are 2 test cases to test in both cases when the player has enough cards on hand and when they don't.

-On each test cases, the number of coins, player's handCount, playedCardCount, and deckCount are checked if they are updated correctly. Also the steward card should be added into the played pile.

-Then, I check other players' handCount and deckCount are not affected.

cardTest4.c: cardEffect() for great_hall

-Similar to above, this card effect involes player drawing cards so the 2 test cases check the behavior when player has enough cards on deck, and when they don't have enough cards on deck

-In each test case, I check the player's handCount, playedCardCount, deckCount are updated correctly. Also other players' handCount and deckCount are not affected. The great_hall card is correctly put into the played pile.

-Then I also check if the number of actions has been increased by 1.