Eddie Christopher Fox

February 7, 2017

CS 362:

Assignment 3 Testing Document

Choice of the Inputs:

Unit Test 1: Testing the getCost() function did not require much input choice. I simply created hard coded arrays of the proper names and costs of every card, making sure the indexes would match up. Then, using a for loop, I called the function manually while passing it every single card, and then asserted it equal to the correct cost of the matching index.

Unit Test 2: Tested the numHandCards() function. Since it returns the state.handCount[currentPlayer], I thought it would be a good idea to create two four loops, one for each player, and run various comparisons, remembering to end the turn in between players to test if the function was able to change player state properly. If it couldn't this would lead to incorrect results. I used similar variables as I did in the card tests, initializing the bare minimum I would need to initialize the game (game state struct, random seed, kingdom array, current player, number of players.) Inside the for loop, I actively changed the value of the hand of the current player, and did assert true's for all three variables with each other in every way: handCount, the loop control variable, and numHandCards. At every stage after the initial assignment, and no matter how it is written, they should be equal.

Unit Test *3:* Did not finish in time.

Unit Test 4: Did not finish in time.

Card Test 1: Smithy card. For this testing, I used the example method of creating two game states, an original and a test, and copying the original. After modification from the tests, we compare the test state to the original with the modifications that should have occurred via alternative variables. Variables initialized were necessary for several purposes.

1. Initializing the game state (such as numPlayers, seed, kingdom array)
2. In order to call cardEffect() (choice1, choice2, choice3, handpos, bonus)
3. To test something relevant within the function by providing baseline modifications to the original state to compare with the tested state later (newCards, extraCoins, discarded, shuffled).

Card Test 2: Adventurer card. For this one, I eased off on variables defined before the game was initialized. I was only doing a single game state for this card, so initializing too much before the game didn't serve a purpose of alternative comparison. I just initialized current player, the kingdom array, the game state, and the seed before calling initialize game. Afterwards, I manually put adventurer in the current players hand and created a deck manually so I could test for specific outcomes. The tests aren't exhaustive, but they can be instructive, and because I set it

up manually, someone can have more control. I set deck count, discard count, played count, and hand count accordingly. After asserting that playAdventurer() worked, I compared several values such as hand count, discard count, and action count to their expected values.

Card Test 3: While mostly like the testing of the adventurer card, I needed to add choice1, choice2, choice3, handpos, and bonus back into the variables declared before the game because I didn't have a refactored playVillage function ready. Everything else is similar to adventurer testing, except I tweaked some of the values, like having 6 cards in the deck instead of 5.

Card Test 4: Testing council room was similar to the others with a bit of difference. I needed handPos but not the other predeclared variables (like choice1, choice2, and bonus) because I already had a refactored playCouncil_Room() function. I decided to start the current player with 1 buy point to see if the card would give them another, and started the other player off with a card so I could test if they would have 2 cards after Council Room was played.

Bugs:

Unit Test 1:

Unit Test 2:

Unit Test 3:

Unit Test 4:

Card Test 1:

Card Test 2:

Card Test 3:

Card Test 4:

Coverage:

Unit Test 1:

Unit Test 2:

Unit Test 3:

Unit Test 4:

Card Test 1:

Card Test 2:

Card Test 3:

Card Test 4: