Sanjana Rathore

UID: 006216683

Project 1 Report

Notable Obstacles and Coding Process:

The most notable obstacle I faced when programming this project was figuring out how to go about coding the evaluateHand() function. I ended up coding it row by row because it was easier for me to keep track of the test cases that I have coded and organize my code. In my code, the evaluateHand() function can be split into three main parts using a series of if/else statements. First, I started off handling all the soft hands (hands with at least one ace). I did this so I could get the ace pair out of the way before coding the other pair hands and because the soft hands seemed easier to code. Then, I handled all the pairs. I noticed that it would be important to check for pairs before the sums of hands as a hand of four and four should be treated like a pair of 4 and not like a sum of 8. Lastly, I handled all the sums or the hands that added to any value between 8-17. If the test case is not on the strategy card, by default, evaluateHand() will return “stand”.

Testing:

To ease the debugging process, I tested my evaluateHand() function as I coded each row by creating assert statements for each cell in the row to make sure the entire player strategy card works. I organized it by creating a new hand object that matched the hand that was given in the row and a set of assert lines for each row to test the hand for every possible dealer card. Occasionally, I would run into a failed assert statement due to an arithmetic error such as using or (||) instead of (&&), but since I was testing as I went I was able to catch and fix those quickly.

Here is an example of one set of my assert statements:

Hand h1( ace, ace);

assert( h1.evaluateHand(two) == Choice::SPLIT );

assert( h1.evaluateHand(three) == Choice::SPLIT );

assert( h1.evaluateHand(four) == Choice::SPLIT );

assert( h1.evaluateHand(five) == Choice::SPLIT );

assert( h1.evaluateHand(six) == Choice::SPLIT );

assert( h1.evaluateHand(seven) == Choice::SPLIT );

assert( h1.evaluateHand(eight) == Choice::SPLIT );

assert( h1.evaluateHand(nine) == Choice::SPLIT );

assert( h1.evaluateHand(ten) == Choice::SPLIT );

assert( h1.evaluateHand(jack) == Choice::SPLIT );

assert( h1.evaluateHand(queen) == Choice::SPLIT );

assert( h1.evaluateHand(king) == Choice::SPLIT );

assert( h1.evaluateHand(ace) == Choice::SPLIT );

I repeated this for every hand in the player strategy card.

I did not explicitly test the getter and setter methods or the isPair() and isSoft() functions as they are implemented in the evaluateHand() function so I was able to test them via the evaluateHand() function.