Name: Zihan Xia

UCLA ID Number: 205838465

Notable obstacles:

1. It’s been a long time since I took cs31 and wrote code in C++ so I am very unfamiliar with a lot of the grammar, and it was a bit of work to get comfortable with the syntax again.
2. I haven’t played blackjack before so it took some time for me to understand the rule, but playing the online game is helpful, also I found out the project itself does not require us to understand the game inside and out.

Test Data:

**int** main() {

//Testing the setSuit and setFace functions

Card b;

// Set the suit and face using the setSuit and setFace methods

b.setSuit(Suit::DIAMONDS);

b.setFace(Face::JACK);

// Verify the changes using the getSuit and getFace methods

cout << "Suit: " << **static\_cast**<**int**>(b.getSuit()) << endl; // Should print 2 (DIAMONDS)

cout << "Face: " << **static\_cast**<**int**>(b.getFace()) << endl; // Should print 10 (JACK)

Card c;

Card c1;

Card c2;

Hand h( c1, c2);

Card ace( Suit::CLUBS, Face::ACE );

Card two( Suit::CLUBS, Face::DEUCE );

Card three( Suit::CLUBS, Face::THREE );

Card four( Suit::CLUBS, Face::FOUR );

Card five( Suit::CLUBS, Face::FIVE );

Card six( Suit::CLUBS, Face::SIX );

Card seven( Suit::CLUBS, Face::SEVEN );

Card eight( Suit::CLUBS, Face::EIGHT );

Card nine( Suit::CLUBS, Face::NINE );

Card ten( Suit::CLUBS, Face::TEN );

Card jack( Suit::CLUBS, Face::JACK );

Card queen( Suit::CLUBS, Face::QUEEN );

Card king( Suit::CLUBS, Face::KING );

// two aces...

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

//6,6, reason:testing the row when we have two six in hand

Hand h2(six, six);

assert( h2.evaluateHand(two) == Choice::SPLITHIT );

assert( h2.evaluateHand(three) == Choice::SPLIT );

//9,

Hand h3(four, five);

//testing two cards that sums to 9, with two different cases of dealer’s card

assert(h3.evaluateHand(three) == Choice::DOUBLEHIT);

assert(h3.evaluateHand(seven) == Choice::HIT);

//8

Hand h4(two, six);

//according to the strategy card, we should hit when the cards in our hand sums to eight regardless of the value of the dealer’s card

assert(h4.evaluateHand(seven) == Choice::HIT);

//10

Hand h5(two, eight);

assert (h5.evaluateHand(seven) == Choice::DOUBLEHIT);

assert(h5.evaluateHand(ten) == Choice::HIT);

assert(h5.evaluateHand(ace) == Choice::HIT);

//13, testing all cases when the hand sums to 13

Hand h6(six, seven);

assert (h6.evaluateHand(ace) == Choice::HIT);

assert(h6.evaluateHand(four) == Choice::STAND );

//11

Hand h7(two, nine);

assert(h7.evaluateHand(four) == Choice::DOUBLEHIT );

assert(h7.evaluateHand(five) == Choice::DOUBLEHIT );

assert(h7.evaluateHand(seven) == Choice::DOUBLEHIT );

//15

Hand h8(jack, five);

assert (h8.evaluateHand(ten) == Choice::SURRENDERHIT );

assert(h8.evaluateHand(five) == Choice::STAND );

assert(h8.evaluateHand(nine) == Choice::HIT);

//12

Hand h10(queen, two);

assert(h10.evaluateHand(five) == Choice::STAND );

assert(h10.evaluateHand(nine) == Choice::HIT);

//A, 7, testing three different cases with the soft card A,7

Hand h11(ace, seven);

assert(h11.evaluateHand(two) == Choice::STAND );

assert(h11.evaluateHand(five) == Choice::DOUBLESTAND );

assert(h11.evaluateHand(nine) == Choice::HIT);

//2,2, testing all cases with the pair card of 2,2

Hand h1( two, two );

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

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

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::HIT );

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

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

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

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

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

//4,4, testing all cases with the pair card 4,4

Hand h9( four, four );

assert( h9.evaluateHand(two) == Choice::HIT );

assert( h9.evaluateHand(three) == Choice::HIT );

assert( h9.evaluateHand(four) == Choice::HIT );

assert( h9.evaluateHand(five) == Choice::SPLITHIT );

assert( h9.evaluateHand(six) == Choice::SPLITHIT);

assert( h9.evaluateHand(seven) == Choice::HIT );

assert( h9.evaluateHand(eight) == Choice::HIT );

assert( h9.evaluateHand(nine) == Choice::HIT );

assert( h9.evaluateHand(ten) == Choice::HIT );

assert( h9.evaluateHand(jack) == Choice::HIT );

assert( h9.evaluateHand(queen) == Choice::HIT );

assert( h9.evaluateHand(king) == Choice::HIT );

**return**( 0 );

}