Shaan Mathur

Stahl

CS31

March 11, 2016

Project 7 Report

This project was very straightforward in regards to what was expected of us. To call the skeleton a “skeleton” would be an overstatement— the program given to us was already functioning well, and there were just but a few simple functions Professor Stahl wanted us to implement. Even these functions consisted of accessor methods which required one simple return statement. As for obstacles, there were not many. The function that did require some thought was perhaps Player::handcount(), because we needed to know when to count an Ace as a 1 or an 11. After some thought, I realized that in any game you would only count one Ace as an 11, because two Aces counted as a 11 would cause the player to bust. Thus, in my algorithm, I made sure to keep one Ace aside and then evaluate at the end whether to count it as an 11 or 1, depending on what would make it better for the player. If the count so far was less than or equal to 10, then 11 should be counted because the higher the number the better. If it was greater than 10, though, it should be counted as 1 because counting it as 11 would cause a bust.

Test Data:

Person player, dealer;

Deck d;

d.shuffleDeck()

//Tests Player::acceptCard() for 12 indices

for(int i = 0; i < 12; i++){

player.acceptCard(d.dealCard());

}

//Makes sure player can't accept more than 12 cards

//as long as there is space in the array myCards, place Card c into myCards

//if there is not enough space for another card, throw an exception

try{

player.acceptCard(d.dealCard());

}catch(...){

cout << "Success! You can't have more than 12 cards" << endl;

}

//Dealer will act similarly

//Tests Player::getCard() AND Player::operator<<(...)

for(int i = 0; i < 12; i++){

cout << player.getCard(i) << endl;

}

//You cannot have an invalid index, and function should throw an exception if an invalid //index is given

player.getCard(12);

}catch(...){

cout << "Success! You cannot have an invalid index, and function should throw an exception." << endl;

}

Player p;

Card aceOfHearts; //default valued at Ace of Hearts

p.acceptCard(aceOfHearts);

p.acceptCard(aceOfHearts);

//Testing function Player::handcount()

assert(p.handcount() == 12);

assert(p.handcount90 != 22); //handcount() must be able to recognize when it should //count Ace as 11, and as 1

//In this case, the optimum situation would be for a 12;

//Testing function Player::handcount()

p.acceptCard(aceOfHearts);

p.acceptCard(aceOfHearts);

assert(p.handcount() == 14); // 4 Aces in a hand have to equal 14 = (11 + 1 + 1 + 1)

//Testing function Player::handcount()

Player d;

Card two(Card::Face DEUCE);

Card three(Card::Face THREE);

Card jack(Card::Face JACK);

d.acceptCard(two);

d.acceptcard(three);

d.acceptCard(jack);

assert(d.handcount() == 15);

//Testing function Player::hasBlackJack() and Player::handcount()

Player p2;

p2.acceptCard(aceOfHearts); //11

p2.acceptCard(jack); //11 + 10 = 21

assert(p2.hasBlackJack());

//Testing Game::deal();

Player player, dealer;

Game g(player, dealer);

g.deal();

game.display(); //CHECK TO SEE IF BOTH PLAYER AND DEALER HAVE CARDS. //PLAYER SHOULD SHOW TWO CARDS, AND DEALER ONLY SHOWS ONE

//Tests Game::dealerBusted() and Game::playerBusted()

Player bustedp, bustedd;

bustedp.acceptCard(jack);

bustedp.acceptCard(jack);

bustedp.acceptCard(jack); //now player is busted

bustedd.acceptCard(jack);

bustedd.acceptCard(jack);

bustedd.acceptCard(jack); //now dealer is busted

Game newGame(bustedp, bustedd);

assert(newGame.dealerBusted() && newGame.playerBusted());

//Tests Game::dealerPlays() and Game::dealerStands()

for(int i = 0; i < 50; i++){

Player p, d;

Game g(p,d);

g.dealerPlays();

g.display(); //Check each iteration and make sure dealer never falls below 17 in //handcount()

system("PAUSE");

}

Game blackjack(player, dealer);