Project 1

The complex game of

Poker!

CIS -17c 40369

Name: Paul Thai

Date: 04/26/2020

**Introduction:**

“Life is not always a matter of holding good cards, but sometimes, playing a poor hand well.”

― Jack London

Hello! Welcome to the game of poker, Texas Holdem! This project is about a poker game that I have longed to learn how to play but I never got the change to since my family does not gamble and I do not want to get addicted to it. When I was in high-school, I was invited to a casino-themed birthday day and out of all the games they had, this was the only game that I did not know how to play. Now, I have the opportunity to create this game through code and teach myself how to play it!

Although there are still some minors and bugs that I have yet to finish debugging, I have spent more than 60 hours on this project because of many reasons. First, I didn’t know how to play Texas Holdem. As a result, I had to watch many videos and look up many examples how to play the game. If I don’t know the rules of the game, there is no way that I could have programmed it so reading and making sure I knew the rules was important; the process which I researched took a fairly long time as well. Second, the implementations off the check list is very difficult since most of the code was finished or near finished before the check-off sheet was released, as such, there wasn’t enough time for me to finish debugging all the problems and implement all the additional requirements. Third, I like to comment while I program so my future-self can look back and remember what my present-self is doing.

Roughly, there are 1430 lines of codes and comments. In the sixth version of the project, there are four header files and four source files with the addition of the main file. There is a card class which creates the cards that is used to compare in the hand class. The hand class takes in the cards and determines what type of hands the cards create. Ex. pair, two pair, flush, straight flush and so on. Then there is a player class, that accepts the cards as they are dealt out to them. The dealer class will then get the player, allow them to pet, create and set the pot, and give them the ability to call, raise, or fold. It also determines the winner for each round and will reset once the round is over.

**Approach to development:**

This project, I thought about changing up the way I develop. Usually, I start in main and them work out to the classes, but in this case, I decided to work on each class individually and then testing them out as I go along. In the first version I started out with the basic – a deck of cards. Any card game cannot function without a card class, so I created the card class the gets the number of cards, the suits, and then the faces. I was researching online how to incorporate actual card like figures and came upon this website, htmlsymbols.xyz/Unicode, and was able to add the card pictures with its face name and suit. After finishing the card class, I went and build the hand class. Now, the hand class will take in the hands the players has and check how strong it is. It is organized in terms of hand strength, so the high card is the lowest and straight flush is the highest. I did consider adding the royal straight flush, but since the chances of getting one are so low, I opted out on it, for now. Once the hand class was finish and tested. I went on to the player class and created the status of the player as well as many setters and getter functions. I thought that the players don’t really do much the poker besides calling, raising, or folding, so I thought maybe we can get the responses from the players through the dealer class and put most of the work there. As such, the player class, add cards to their hand and confirm the bets as well as their agreement to continue playing or sit out. Now here is where the pain begins, the dealer class. The dealer class does a lot of things: shuffle, set and gets bet, get players, gets the deck, create the small and big blind, create the betting rounds, resetting the rounds and determining the winner. Although I have ran my code more than 100 times to test out the dealer class, there is always something wrong with it. Granted it might be me who is missing something in between, but the dealer class was created last because it incorporated everything in the other classes. Overall, I did think it was a lot harder than this way then it is to work from completed main class and then expand it to classes. However, I did learn a lot from this, and I am looking forward to making further changes in the next project.

**Game Rules:**

Texas Hold'em Implementations Draft and Setup.

(Everything will be moved clockwise)

1. Everyone needs to take turns being the dealer. At the end of the code, move the person who was initially the dealer to the person on the right.

2. Create the betting system.

a. There are two betting systems, the big and the small blind.

1. The small blind will be to the left of the dealer and then the big blind will be to the left of the person with the small blind.

2. Ex. If the person who has the small blind bids 5, then the persons who has the big blind has to bid twice as much or more. It can’t be less than the person who bided in the small blind.

3. Dealing the Cards.

a. Let’s do three players.

4. Player's options.

a. Fold - If hand is not good, it’s like forfeiting for the turn.

b. Call - Hand is decent so the player will match the price in the big blind. If blind = 10 then call 10.

c. Raise - Now you are confident, so you have to double the current bid of the previous bid (minimum)

Now everything will continue clockwise. If the other player's bid is less than the current max bid, they

will have a chance to match the highest bid or fold, call or even raise the pot even higher.

d. Now, everyone has the same amount in their bidding hand and the 1st round of betting ends.

All the money will be going into the pot.

5. The Flop

a. Three cards will be set onto the table and the players will try to match the two cards that they have,

with the three cards on the table to make the strongest hand.

6. 2nd round of betting

a. Starting from the left of the dealer, everyone will start their 2nd round of betting. This will repeat

Step 4 again.

7. Checking

a. Don't want to commit anything to the pot, and you don’t want to add anything either, so you check, and the turn then goes to the next person.

8. Betting Continues

a. In this scenario, since the first-person checks, but the 2nd person doesn't. What will happen is that

since the small blind was 5 and the higher blind was 10, then the person has to bid at least 10.

b. However in this case, the person that betted bet 200 chips and now it continues around to the other

players. The other players have the option to do things in step 4.

9. The turn

a. After the flop ends, there is a turn phase where an additional card is added to the flop or onto the middle of the table.

10. 3rd round of betting.

a. Player next to the dealer checks his hand and, in this case, he is very confident so he will raise by 200 chips.

b. The other player that is still with him is confident as well, so he raises it to 500 chips.

c. Both players have the option to continue step 4.

d. Then, one player calls the bet and then all the money will the go into the pot.

11. The River

a. The fifth card is now placed onto the table after the betting cycle ends.

b. The player will do their best to create the best hand out of the ones that they have and the ones on the table.

12. 4th round of betting

a. The remaining player can do step 4 again and see if the other will drop out or not.

13. Turning the cards over

a. Both players will turn the cards over and the winning player will get the all the things in the pot!

**A screenshot of a social media post

Description automatically generatedSample Input/output**

**A screenshot of a cell phone

Description automatically generated**

**Check Off List:**

|  |  |
| --- | --- |
| Check Off | Location and File Line# |
| Sequences |  |
| List: | “Player.h” file Line 30 |
| Associative Containers |  |
| Set: | “Hand.cpp” file line 72 |
| Map: | “Hand.h” file Line 27 |
|  |  |
| Container Adapters: |  |
| Stack: | No Current Implementations |
| Queue: | “Player.cpp” file Line 20 |
|  |  |
| Iterators |  |
| Forward Iterator: | “Hand.cpp” file Line 84 |
| Input Iterator: | “Hand.cpp” file line 197 |
|  |  |
| Algorithms |  |
| Non-mutating: Find - | “Hand.cpp” file Line 203 |
| Mutating Algo: Swap - | “Dealer.cpp” file Line 170 |
| Organization: Sort - | No direct implications, only use cstdlib  For randomizing the cards. |
|  |  |
| Others |  |
| Pair: | “Hand.cpp” file Line 152 |
|  |  |

**Pseudo Code:**

***Code is running-***

*Displays greeting-*

*Ask the player how many people are going to be playing today*

***Pre-Flop***

*Deal out 2 cards for the players*

***Game-***

*Blinds –*

*The greater blind is created by one player.*

*The Small Blind is created by the other player*

*Both Player has to call to match the total of the big blind or raise.*

*-if raised, the bet must be at least double of the bid blind.*

***First round of Bidding –***

*Both players must have the same amount as the highest as the big blind*

*-if someone does not, they will be asked to call, raise or fold.*

***The Flop –***

*Three cards will be dealt to the table for the players to compare*

***Second round of Bidding –***

*The players are each given the chance to check and then call, raise or fold.*

*-if called, the pot remains the same*

*-if raised, the amount has to be at least double of the previous raise.*

*-if fold, the player will be removed from the round.*

***The Turn –***

*Another Card is put on the table in addition to the flop.*

***Third Round of Bidding –***

*The players are each given the chance to check and then call, raise or fold.*

*-if called, the pot remains the same*

*-if raised, the amount has to be at least double of the previous raise.*

*-if fold, the player will be removed from the round.*

***The River –***

*Another Card is put on the table with the flop and the turn.*

***Fourth Round of Bidding***

*The players are each given the chance to check and then call, raise or fold.*

*-if called, the pot remains the same*

*-if raised, the amount has to be at least double of the previous raise.*

*-if fold, the player will be removed from the round.*

***Deciding outcome-***

*Now the players must make the strongest hand by using the two cards that they have and the cards on the table*

*Lowest -> Highest (Card Strength)*

1. *High-Card* 
   1. *One card that is the highest out of the bunch*
2. *One-pair*
   1. *There are two cards that have the same face value*
3. *Two-pair*
   1. *There are two sets of two cards that have the same face value*
4. *Three-of-a-kind*
   1. *There are three same faces from three different suits*
5. *Straight*
   1. *There are five cards that are in ascending chronological order*
6. *Flush*
   1. *There are 5 cards that have all of the same suit*
7. *Full House*
   1. *There is a pair and a three of a kind*
8. *Four-of-a-kind*
   1. *There are all four faces from all four suits*
9. *Straight Flush* 
   1. *5 cards from the same suit but is ascending in chronological order.*
10. *Royal Straight Flush*
    1. *The rumored hand that follows the straight flush but with the order of 10->jack->queen->king->ace*

**Reference**:

1. <https://www.htmlsymbols.xyz/unicode/U+1F0D1>
2. Doctor Lehr’s Lecture and Examples

**GitHub:**

1. <https://github.com/Lyfae/CIS---17c>

A close up of text on a white surface

Description automatically generated**Flow Chart:**

A picture containing text, map, parking, lot

Description automatically generated

A close up of a map

Description automatically generated

A map of a lot of text

Description automatically generated**Classes:**

**CPP File: Cards**

#include <cstdlib>

#include <iostream>

#include <iomanip>

#include <string>

//user libraries

#include "Cards.h"

using namespace std;

//constructor for cards

Cards::Cards(int n) {

//true

if (n >=0 && n < 52) {

//set values

this->cNum = n;

this->setPlayCard();

this->setSuit();

this->setFace();

} else { //precaution if it doesn't match

this->cNum = -1;

this->cFace = -1;

this->cSuit = -1;

this->faceName = "Invalid";

this->playCard = "Invalid";

this->suitName = "Invalid";

}

}

//setter functions

//set suit

void Cards::setSuit() {

//set values of clubs

if (this->cNum < 13) {

this->cSuit = 0;

this->suitName = "Clubs";

} else if (this->cNum < 26) { //set values for diamonds

this->cSuit = 1;

this->suitName = "Diamonds";

} else if (this->cNum < 39) { //set the values for hearts

this->cSuit = 2;

this->suitName = "Hearts";

} else if (this->cNum < 52) { //set the value for spades

this->cSuit = 3;

this->suitName = "Spades";

} else { //if neither matches

this->cSuit = -1;

this->suitName = "Invalid Card";

}

}

//set face

void Cards::setFace() {

//setting value of the cards

//setting the card face equal to the card number

if (this->cNum % 13 + 1 == 1) {

this->cFace = 14;

} else {

this->cFace = this->cNum % 13+1;

}

switch (this->cFace) {

case 2:

this->faceName = "Two";

break;

case 3:

this->faceName = "Three";

break;

case 4:

this->faceName = "Four";

break;

case 5:

this->faceName = "Five";

break;

case 6:

this->faceName = "Six";

break;

case 7:

this->faceName = "Seven";

break;

case 8:

this->faceName = "Eight";

break;

case 9:

this->faceName = "Nine";

break;

case 10:

this->faceName = "Ten";

break;

case 11:

this->faceName = "Jack";

break;

case 12:

this->faceName = "Queen";

break;

case 13:

this->faceName = "King";

break;

case 14:

this->faceName = "Ace";

break;

default:

this->faceName = "Not applicable";

}

}

//set play card

//the code string to the cards = pictures

void Cards::setPlayCard() {

string c[52] =

{"\U0001F0D1", "\U0001F0D2", "\U0001F0D3", "\U0001F0D4", "\U0001F0D5"

, "\U0001F0D6", "\U0001F0D7", "\U0001F0D8", "\U0001F0D9", "\U0001F0DA", "\U0001F0DB"

, "\U0001F0DD", "\U0001F0DE", "\U0001F0C1", "\U0001F0C2", "\U0001F0C3", "\U0001F0C4"

, "\U0001F0C5", "\U0001F0C6", "\U0001F0C7", "\U0001F0C8", "\U0001F0C9", "\U0001F0CA"

, "\U0001F0CB", "\U0001F0CD", "\U0001F0CE", "\U0001F0B1", "\U0001F0B2", "\U0001F0B3"

, "\U0001F0B4", "\U0001F0B5", "\U0001F0B6", "\U0001F0B7", "\U0001F0B8", "\U0001F0B9"

, "\U0001F0BA", "\U0001F0BB", "\U0001F0BD", "\U0001F0BE", "\U0001F0A1", "\U0001F0A2"

, "\U0001F0A3", "\U0001F0A4", "\U0001F0A5", "\U0001F0A6", "\U0001F0A7", "\U0001F0A8"

, "\U0001F0A9", "\U0001F0DA", "\U0001F0AB", "\U0001F0AD", "\U0001F0AE"};

//print out the cards

//if cards match to the value being called

for (int i = 0; i < 52; i++) {

if (this->cNum == i) {

//value of card equal to the actual card

this->playCard = c[i];

}

}

}

//output

void Cards::output() {

//output the playing card

cout << this->Cards::gPlayCard() << " ";

//output face name

cout << this->Cards::gFaceName() << " ";

//output the suits

cout << this->Cards::gSuitName() << endl;

}

**CPP File: Hand**

//libraries

#include <iostream>

#include <iomanip>

#include <string>

#include <set>

#include <map>

#include <iterator>

#include <algorithm>

#include <list>

//user libraries

#include "Hand.h"

using namespace std;

//constructor

Hand::Hand(){

this->Hand::setConfirm();

this->Hand::setPot();

}

//getting the name of the hands

string Hand::getHandName() {

switch (this->Hand::getHand()) {

case 0:

this->handName = "High-Card";

break;

case 1:

this->handName = "One-Pair";

break;

case 2:

this->handName = "Two-Pair";

break;

case 3:

this->handName = "Three-of-a-Kind";

break;

case 4:

this->handName = "Straight";

break;

case 5:

this->handName = "Flush";

break;

case 6:

this->handName = "Full-house";

break;

case 7:

this->handName = "Four-of-a-kind";

break;

case 8:

this->handName = "Straight-Flush";

break;

default:

this->handName = "Invalid";

}

return this->handName;

}

//check the cards if they order 5 in a row

void Hand::checkStraight() {

this->pS = 0;

//set faces / declare variables

set<int, greater<int>> faceSet;

int count = 0;

//get face

for (auto& i : this->Hand::getCards()) {

faceSet.insert(i.getFace());

}

//start

set<int>::iterator a = faceSet.begin();

//checking the cards

while (a != faceSet.end() && count != 4) {

if (\*a - \*faceSet.upper\_bound(\*a) == 1) {

count++;

} else {

count = 0;

}

++a;

}

if (count == 4 && \*a + 4 != 14) {

this->pS = \*a + 4;

} else if (count == 4 && \*a + 4 == 14) {

this->pS = 15;

}

if (\*a == 14) {

while (a != faceSet.end() && count != 3) {

if (\*a - \*faceSet.upper\_bound(\*a) == 1) {

count++;

} else {

count = 0;

}

++a;

}

}

if (count == 3 && \*a + 3 == 5) {

this->pS = 14;

}

faceSet.clear();

}

//check for flush by determining the suits

void Hand::setSuits() {

//initialize

this->pf = 0;

multiset<int> setSuit;

//enhanced for loop to get the suits

for (auto& i :this->Hand::getCards()) {

setSuit.insert(i.getSuit());

}

//loop through

for (int i = 0; i < 4; i++) {

if (setSuit.count(i) > 4) {

this->suits.insert(pair<int, int>(setSuit.count(i), i));

this->pf = 1;

}

}

//clear the set

setSuit.clear();

}

//checking for pairs

void Hand::setFaces() {

//declare and initialize variables

this->pP = 0;

multiset<int> setFace;

//enhanced for loop to get the face

for (auto& i : Hand::getCards()) {

setFace.insert(i.getFace());

}

//insert card

for (int i = 2; i < 15; i++) {

if (setFace.count(i) > 1) {

this->faces.insert(pair<int, int>(setFace.count(i), i));

}

}

map<int, int>::iterator pos;

if (this->faces.empty()) {

this->pP = 0;

} else if (this->faces.size() == 1) {

pos = this->faces.begin();

if (pos->first == 2) {

this->pP = 1;

} else if (pos->first == 3) {

this->pP = 3;

} else if (pos->first == 4) {

this->pP = 7;

}

} else if (this->faces.size() > 1) {

pos = this->faces.begin();

if (pos->first == 2) {

this->pP = 2;

} else if (pos->first == 3) {

this->pP = 6;

}

}

setFace.clear();

}

int Hand::getHand() {

this->checkStraight();

this->Hand::setSuits();

this->Hand::setFaces();

if (this->pS != 0 && this->pf != 0) {

this->pSF = 1;

this->handType = 8;

set<int, greater<int>> psf; //poker-straight-flush

for (auto& i : this->Hand::getCards()) {

if (i.getSuit() == this->pSF) {

psf.insert(i.getFace());

}

}

if (this->pS == 15) {

for (int i = 14; i > 9; i--) {

if (psf.find(i) == psf.end()) {

this->pSF = 0;

}

}

} else if (this->pS == 15) {

for (int i = 5; i > 1; i--) {

if (psf.find(i) == psf.end()) {

this->pSF = 0;

}

}

} else {

for (int i = this->pS; i> this->pS - 5; i--) {

if (psf.find(i) == psf.end()) {

this->pSF = 0;

}

}

}

if (this->pSF == 0) {

this->handType = 5;

}

}

//setting the hand type according to the cards

else if (this->Hand::pP == 7) {

this->handType = 7;

} else if (this->pP = 6) {

this->handType = 6;

} else if (this->pf != 0) {

this->handType = 5;

} else if (this->pS != 0) {

this->handType = 4;

} else if (this->pP == 3) {

this->handType = 3;

} else if (this->pP == 2) {

this->handType = 2;

} else if (this->pP == 1) {

this->handType = 1;

} else {

this->handType = 0;

}

return this->handType;

}

**CPP File: Player**

//libraries

#include <list>

//user libraries

#include "Player.h"

//constructor

Player::Player(){

this->setPlayerTotal();

}

//add cards

void Player::addCards(Cards c){

this->pCards.push\_back(c);

}

**CPP File: Dealer**

//libraries

#include <cstdlib>

#include <iostream>

#include <string>

//user libraries

#include "Dealer.h"

using namespace std;

//constructor

Dealer::Dealer(int n) {

//declare and initialize the variables

int replay = 1;

this->setnPlayers(n);

this->DOC();

do {

//start of the game.

this->setIBet(10); //initialize the starting bet to 10 (testing)

this->setBlinds(); //make sure blinds are associated with the right person

this->resetPot(); //clear the pot to avoid leaking mem

this->shuffle(); //shuffle the deck

this->preflop(); //deal out 2 cards

//starting the bet

//first round of betting

int betValue = roundOneBet();

cout << "The first round of betting has ended. We will continue to the flop." << endl << endl;

//second round of betting.

int num = 0;

int count = 1;

int nP = this->nPlayers;

int x = 1;

//flop

cout<<"The Flop"<<endl;

this->flop();

cout << "hit2" << endl;

while (num < nP) {

do {

betValue = this->roundTwoBet((count + this->smallB) % nP, betValue);

if ((count + this->bigB) % nP == this->bigB) x = 0;

if (this->Dealer::getNumAct() == 1) {

x = 0;

}

count++;

num++;

} while (betValue - this->player[(count + this->smallB) % nP].potTot() || x);

}

cout << "The Second Round of betting is done, proceeding to the turn." << endl << endl;

//the turn

this->turn();

//third round of betting

count = 0;

x = 1;

num = 0;

while (num < nP) {

do {

int aP = this->Dealer::getNumAct();

betValue = this->roundThreeBet((count + this->bigB) % nP, betValue);

count++;

if ((count + this->bigB) % nP == this->bigB) x = 0;

if (this->Dealer::getNumAct() == 1) {

x = 0;

}

num++;

} while ((betValue - this->player[(count + this->bigB) % nP].potTot()) || x);

}

cout << "Third Round of betting is done. Proceeding to the river." << endl << endl;

//the river

this->river();

//fourth round of betting

count = 0;

x = 1;

num = 0;

while(num<nP){

do{

int aP = this->Dealer::getNumAct();

betValue = this->roundFourBet((count + this->bigB) % nP, betValue);

count++;

if ((count + this->bigB) % nP == this->bigB) x = 0;

if (this->Dealer::getNumAct() == 1) {

x = 0;

}

num++;

}while ((betValue - this->player[(count + this->bigB) % nP].potTot()) || x);

}

cout << "Fourth Round of betting is done. Proceeding to finding the winner." << endl << endl;

for (int i = 0; i<this->nPlayers; i++) {

int order = (i + this->bigB) % this->nPlayers;

cout << "Player" << order << ": " << this->player[order].getHandName() << endl;

}

this->calcTot();

for (int i = 0; i<this->nPlayers; i++) {

int order = (i + this->bigB) % this->nPlayers;

this->Dealer::pInfo(order);

}

replay = this->reset();

} while (replay != 0);

delete [] player;

}

//Creating the deck of cards

void Dealer::DOC() {

for (int i = 0; i < 52; i++) {

//pass the cards through as new cards

this->c[i] = new Cards(i);

}

}

//setting the number of player

void Dealer::setnPlayers(int num) {

//set the players equal to the number passed in

this->nPlayers = num;

//create the hand according to the players

this->player = new Hand[this->nPlayers];

//set the player name

for (int i = 0; i<this->nPlayers; i++) {

this->player[i].setName("Player: " + to\_string(i));

}

}

//setting the blinds

void Dealer::setBlinds() {

//divide by the players, minus 1 because that person is the dealer

this->bigB = this->getnRounds() % (this->getnPlayers() - 1);

this->smallB = this->bigB + 1;

}

//shuffling the deck of cards

void Dealer::shuffle() {

//set the random seed time

srand(time(0));

//shuffle the cards

for (int i = 0; i < 52; i++) {

int j = i + (rand() % (52 - i));

//swap the shuffled deck with base deck

swap(this->c[i], this->c[j]);

}

}

//display the peron's information

void Dealer::pInfo(int num) {

//determine the player

cout << this->Dealer::player[num].getName() << endl;

//determine where the big blind starts

if (num == this->Dealer::getBigBlind()) {

cout << "Big Blind" << endl;

} else if (num == this->Dealer::getSmallBlind()) {

cout << "Small Blind" << endl;

}

cout << "Here is the total balance: $" << this->Dealer::player[num].getTot() << endl;

for (auto& i : this->player[num].getCards()) {

i.output();

}

cout << endl;

}

//getting the players that are still actively playing

int Dealer::getNumAct() {

int num = 0;

for (int i = 0; i<this->nPlayers; i++) {

if (this->player[i].pStatus()) {

num++;

}

return num;

}

}

//calculate the balance each player has

void Dealer::calcTot() {

int win = this->determineWinner();

cout << "Congratulations! Player" << win << " has won $" << this->grandPot() << endl;

//add the pot to the player's total

this->player[win].addTot(this->grandPot());

//then reset the pot

this->Dealer::resetPot();

}

//determine the winner

int Dealer::determineWinner() {

//initialize and declare variables

int numPlayers = this->getnPlayers();

int winner = 0;

int strongest = 0;

//compare who's in play and their hand

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

if (this->player[i].pStatus() == 1) {

if (this->player[i].getHand() > strongest) {

strongest = this->player[i].getHand();

winner = i;

}

}

}

return winner;

}

//resetting the game

int Dealer::reset() {

int input;

this->nRound();

//reset all the player's Hands

for (int i = 0; i<this->getnPlayers(); i++) {

this->player[i].clearHand();

}

//option to quit

cout << "Game Master if you want to quit press 0" << endl;

cin>>input;

return input;

}

//rules of the game

//pre-flop

void Dealer::preflop() {

//declare and initialize variables

int numCards = 2;

int numPlayers = this->nPlayers;

int bigB = this->bigB; //big blind

int order;

//deal out two cards to the players

for (int j = 0; j < numPlayers; j++) {

order = (j + bigB) % numPlayers;

cout << "Player : " << j << endl;

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

this->player[order].addCards(\*this->c[j + numPlayers \* i]);

this->c[j + i + numPlayers \* i]->output();

}

}

cout << endl;

}

//first round of betting

int Dealer::roundOneBet() {

int raise; //raise the highest blind

int input; //get player's decision

int amount; //how much is going in the pot

int dif; //how much is left

int numPlayers = this->nPlayers;

//making the big blind

this->player[bigB].bet(this->iBet); //take in bet

this->potTot += this->iBet; //make blind = bet

cout << "Player " << this->bigB << ": bet $" << this->iBet << " as the total for Big Blind" << endl;

cout << "Current total in pot: " << this->grandPot() << endl << endl;

amount = this->player[bigB].potTot();

//making the small blind

this->player[smallB].bet(this->iBet / 2);

this->potTot += this->iBet / 2;

cout << "Player " << this->smallB << ": bet $" << this->iBet / 2 << " as the total for Small Blind" << endl;

cout << "Current total in pot: " << this->grandPot() << endl << endl;

;

dif = amount - this->iBet / 2;

//player call, raise or fold

do {

cout << "To call you have to match $" << dif << endl;

cout << "Player" << this->smallB << ": Call - 1, Raise - 2, Fold -3" << endl;

cin >> input;

//set the bet the same

if (input == 1) {

cout << dif << endl;

this->player[smallB].bet(dif);

this->potTot += dif;

cout << "Player#" << smallB << ": $" << dif << endl;

cout << "Pot: $" << this->grandPot() << endl;

//at least double the amount

} else if (input == 2) {

cout << "Amount: ";

do {

cin >> raise;

if (raise <= dif) cout << "Amount should be greater than call Amount\n";

} while (raise <= dif);

this->player[smallB].bet(raise + dif);

this->potTot += (raise + dif);

cout << "Player#" << smallB << ": $" << raise + dif << " into the Pot" << endl;

cout << "Pot: $" << this->grandPot() << endl;

amount = this->player[smallB].potTot();

//withdraw the player from the round

} else if (input == 3) {

this->player[smallB].setAFK();

cout << "Player#" << smallB << ": folded" << endl;

cout << "Pot: $" << this->grandPot() << endl;

} else {

cout << "wrong input" << endl;

}

} while (input < 1 || input > 3);

return amount;

}

//flop

void Dealer::flop() {

//declare and initialize variables

int numCards = 3;

int numPlayers = this->nPlayers;

int bigB = this->bigB;

int order;

//output

//deal out the three cards to the center of the table

for (int i = numPlayers \* 2 + 1; i < numPlayers \* 2 + 1 + numCards; i++) {

//print out card

// this->c[i]->output();

for (int j = 0; j < numPlayers; j++) {

order = (j + bigB) % numPlayers;

this->player[order].addCards(\*this->c[i]);

}

}

}

//round two of betting

int Dealer::roundTwoBet(int num, int amount) {

int input;

do {

if (this->player[num].pStatus()) {

int nP = this->nPlayers;

int raise;

int diff = amount - this->player[num].potTot();

cout << "\nTo call you have to put $" << amount << endl;

cout << "Player" << num << ": call - 1, Raise - 2, Fold -3" << endl;

cin >> input;

if (input == 1) {

this->player[num].bet(diff);

this->potTot += (diff);

cout << "Player#" << num << ": $" << amount << endl;

cout << "Pot: $" << this->grandPot() << endl;

} else if (input == 2) {

cout << "Amount: ";

do {

cin >> raise;

if (raise <= diff) cout << "Amount should be greater than call Amount\n";

} while (raise <= diff);

this->player[num].bet(raise + diff);

this->potTot += (raise + diff);

cout << "Player#" << num << ": $" << raise + diff << endl;

cout << "Pot: $" << this->grandPot() << endl;

amount = this->player[num].potTot();

} else if (input == 3) {

this->player[num].setAFK();

cout << "Player#" << num << ": folded" << endl;

cout << "Pot: $" << this->grandPot() << endl;

} else {

cout << "wrong input" << endl;

}

}

} while (input < 1 || input > 3);

return amount;

}

//turn

void Dealer::turn() {

//declare and initialize variables

int numCards = 1;

int numPlayers = this->nPlayers;

int bigB = this->bigB;

int order;

//add another card to the flop

for (int i = numPlayers \* 2 + 5; i < numPlayers \* 2 + 5 + numCards; i++) {

//output

// cout << "The Turn is: " << endl;

// this->c[i]->output();

// cout << endl;

// cout << "On the table: " << endl;

// this->flop();

// this->c[i]->output();

for (int j = 0; j < numPlayers; j++) {

order = (j + bigB) % numPlayers;

this->player[order].addCards(\*this->c[i]);

}

}

}

//third round of betting

int Dealer::roundThreeBet(int num, int amount) {

int nP = this->nPlayers;

int input;

int raise;

int dif;

do {

if (this->player[num].pStatus()) {

dif = amount - this->player[num].potTot();

if (!dif) {

cout << "You can check" << endl;

cout << "Player" << num << ": check - 1, Raise - 2, Fold -3" << endl;

} else {

cout << "To call you have to put $" << dif << endl;

cout << "Player" << num << ": call - 1, Raise - 2, Fold -3" << endl;

}

cin >> input;

if (input == 1) {

this->player[num].bet(dif);

this->potTot += (dif);

cout << "Player#" << num << ": $" << dif << endl;

cout << "Pot: $" << this->grandPot() << endl;

} else if (input == 2) {

cout << "Amount: ";

do {

cin >> raise;

if (raise <= dif) cout << "Amount should be greater than call Amount\n";

} while (raise <= dif);

this->player[num].bet(raise + dif);

this->potTot += (raise + dif);

cout << "Player#" << num << ": $" << raise + dif << endl;

cout << "Pot: $" << this->grandPot() << endl;

amount = this->player[num].potTot();

} else if (input == 3) {

this->player[num].setAFK();

cout << "Player#" << num << ": folded" << endl;

cout << "Pot: $" << this->grandPot() << endl;

} else {

cout << "wrong input" << endl;

}

}

} while (input < 1 || input > 3);

return amount;

}

//river

void Dealer::river() {

//declare and initialize the variables

int numCards = 1;

int numPlayers = this->nPlayers;

int bigB = this->bigB;

int order;

//add one more card from the deck to the turn & flop;

for (int i = numPlayers \* 2 + 7; i < numPlayers \* 2 + 7 + numCards; i++) {

//output

// cout << "The River is:" << endl;

// this->c[i]->output();

// cout << endl;

// this->flop();

// this->turn();

// this->c[i]->output();

for (int j = 0; j < numPlayers; j++) {

order = (j + bigB) % numPlayers;

this->player[order].addCards(\*this->c[i]);

}

}

}

//fourth round of betting

int Dealer::roundFourBet(int num, int amount) {

int nP = this->nPlayers;

int input;

int raise;

int dif;

do {

if (this->player[num].pStatus()) {

dif = amount - this->player[num].potTot();

if (!dif) {

cout << "You can check" << endl;

cout << "Player" << num << ": check - 1, Raise - 2, Fold -3" << endl;

} else {

cout << "To call you have to put $" << dif << endl;

cout << "Player" << num << ": call - 1, Raise - 2, Fold -3" << endl;

}

cin >> input;

if (input == 1) {

this->player[num].bet(dif);

this->potTot += (dif);

cout << "Player#" << num << ": $" << dif << endl;

cout << "Pot: $" << this->grandPot() << endl;

} else if (input == 2) {

cout << "Amount: ";

do {

cin >> raise;

if (raise <= dif) cout << "Amount should be greater than call Amount\n";

} while (raise <= dif);

this->player[num].bet(raise + dif);

this->potTot += (raise + dif);

cout << "Player#" << num << ": $" << raise + dif << endl;

cout << "Pot: $" << this->grandPot() << endl;

amount = this->player[num].potTot();

} else if (input == 3) {

this->player[num].setAFK();

cout << "Player#" << num << ": folded" << endl;

cout << "Pot: $" << this->grandPot() << endl;

} else {

cout << "wrong input" << endl;

}

}

} while (input < 1 || input > 3);

return amount;

}