**PROJECT 1**

**<Go Fish>**

**Link to Github Repository for proof of continuous development:**

[**https://github.com/k7ndr4/C-Game-Project-2**](https://github.com/k7ndr4/C-Game-Project-2)

**CIS-5 #47589**

**Kendra Devlahovicht**

**December 09, 2024**

**Introduction:**

Title: Go Fish

The objective of the game is to get as many 'books' as possible. You gain a 'book' by obtaining 4 of the same type of card. (i.e 4 Jacks = 1 book) The cards used to gain a 'book' will be removed from your hand. Each turn you will ask your opponent for a type of card. If they have this type of card, they must give you all cards of that type. (i.e You ask for 7's, and the opponent has 3 7's. They must give you all 3, and vice versa. Face cards are asked for in ascending order after 10 with the exception of Ace which is 1. (Ace =1, Jack = 11, Queen = 12, King = 13). If you ask for a number below 1, the game will assume you asked for 1's. If you ask for a number above 13, the game will assume you asked for 13's. The game will end once the entire game deck has run out of cards to fish from. The winner will be determined by who has more books. Points are determined by (the rank of your card \* the amount of books you have \* the current turn). Points will be displayed at the end of the game.

**Summary:**

Project Size: ~670 lines.

Project Files: main.cpp, GoFish.h, GoFish.cpp, Deck.h, Deck.cpp

This project is a full demonstration of what I am currently capable of doing using C++. This project contains use of functions, arrays, vectors, pointers, references, and low-level search & sort algorithms.

This project took me around 3 days to complete. I think given the size of the project; I am satisfied with my ability to code such a game. I always enjoyed playing Go Fish with my cousins & siblings growing up, so it was fun conceptually putting together this game into code and seeing it come to life.

This project taught me that it never hurts to have your code scalable for future development as long as you don’t overengineer it and spend too much time on a feature that will be used maybe once or twice. I geared my code towards a very object-oriented approach since I am looking to code games in the future and heard its good practice in that field.

**Description:**

The main purpose of this game is to highlight my proficiency with C++ thus far by using my entire knowledge of C++ and its capabilities. The game can be used as a form of entertainment for those who enjoy a simple RNG game, or those that want to try and build strategies against the AI.

**Flow Chart:**

A screenshot of a computer flowchart

Description automatically generated

**Pseudo Code:**

Include Libraries & use namespace std;

Int main

Initialize Game Deck, Player Deck, Enemy Deck, and Pile

Initialize a GoFish class //ESSENTIALLY THE GAME MANAGER

GoFish.MainMenu() //START THE MAIN MENU OF THE GAME

Void MainMenu() //IN GoFish.cpp

Prompt menu choices

Case 1: Start Game

Display game start

Reset Decks & Stats just incase this is the same session

Clear all decks

Reset all gamesession points to 0

Shuffle the gamedeck

Randomly deal 7 cards to each player from the GameDeck

While(GameDeck still has cards)

If the turn is even, the enemy goes, if not, the player goes

Player asks for card

Input is validated (must be a number 1-13)

If the enemy has the card, call a function that takes all cards of that type from the enemy deck and places it into the player deck.

Display how many cards were taken and check if either player made 4 of a kind, if so then add a book to the corresponding player

If they did: keep track of this variable in the GoFish class

Output the new book score

Check who has more books

Determine winner and log stats into logs.txt

Increase sessiongameCount and keep track of player & enemy points

Case 2: Display Rules and go back to main menu

Case 3: Exit(0);

FOR DECKS

Each Deck is created empty, and has a function built in that fills it with random cards.

Cards are made by the member struct ‘Card’ inside the ‘Deck’ class.

Loop is made to create cards 1-13 and give them corresponding suites up until there is 52 cards.

Deck has the option to be shuffled after

**Major Variables:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable Name** | **Description** | **Location** |
| **Const Integer** | \_FOUROFAKIND | Number of same card types to achieve a ‘book’ | GoFish.h |
|  | \_PLAYER\_HANDSIZE | Size of the starting deck of player&enemy |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **Unsigned Integer** | \_turn | Keeps track of game turns |  |
|  | \_playerBooks, \_enemybooks | Keeps track of how many books each player has |  |
|  |  |  |  |
| **Integer** | \_SessionGameCount | Counts amount of games played in one session |  |
|  |  |  |  |
|  |  |  |  |
| **Integer 1D Array** | \_SessionPlayerPoints[], \_SessionEnemyPoints[] | Keeps track of how many points each player scored across games in a session |  |
|  |  |  |  |
| **Integer 2D Array** | \_SessionWtoL[][] | Keeps track of the wins to losses for each player. [0][] is the player, [1][] is the enemy |  |
|  |  |  |  |
| **Std::Vector<Deck::Card>** | CardDeck | The variable inside Deck.h that is responsible for holding all the cards in that deck. Overload operators for the class have been made | Deck.h |
|  |  |  |  |
| **Float** | \_playerPoints, \_enemyPoints | Keeps track of enemy and player points during a game. | GoFish.h |
|  |  |  |  |
| **Char** | Input{}; | Keeps track of what card the player asked for during a turn | GoFish.cpp |
|  |  |  |  |
|  |  |  |  |
| **Bool** | \_gameStarted | Keeps track if the game has started or not |  |
|  |  |  |  |
| **Deck\*** | GameDeck, PlayerDeck, EnemyDeck, Pile | Decks used to play the game | Int main() |
|  |  |  |  |
| **Static Vars** | \_SUITES[] | Const inline 1D Array that holds the suites names to be created for cards. |  |
|  |  |  |  |
| **GoFish** | GameManager | Keeps track of all decks used to play the game and is responsible for the game loop |  |
|  |  |  |  |
|  |  |  |  |
| **ofstream** | \_GameLogs | Keeps track of all game stats and records it in ‘logs.txt’ | GoFish.h |

**Program**

**main.cpp**

/\*

\* File: main.cpp

\* Author: Kendra

\*

\* Created on December 9, 2024, 3:16 PM

\*/

#include <cstdlib>

#include <iomanip>

#include <iostream>

#include <random>

#include <cmath>

#include <string>

#include <fstream>

#include <ctime>

#include "Deck.h"

#include "GoFish.h"

using namespace std;

int main(int argc, char\*\* argv) {

srand(time(0));

//MAKE THE GAME DECK

Deck\* GameDeck = new Deck(true);

//GameDeck.PrintDeck();

//PLAYER DECK, ENEMY DECK, & PILE

Deck\* PlayerDeck = new Deck;

Deck\* EnemyDeck = new Deck;

Deck\* Pile = new Deck;

//GAME MANAGER

GoFish GameManager(GameDeck, PlayerDeck, EnemyDeck, Pile);

//START THE MAIN MENU

GameManager.MainMenu();

return 0;

}

**Deck.h**

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/cppFiles/file.h to edit this template

\*/

/\*

\* File: Deck.h

\* Author: Kendra

\*

\* Created on December 8, 2024, 4:46 AM

\*/

#ifndef DECK\_H

#define DECK\_H

#include <vector>

#include <string>

#include <random>

#include <cmath>

#include <cstdlib>

#include <iostream>

#include <algorithm>

class Deck{

public:

struct Card{

public:

Card(){

}

Card(int num, std::string suite){

\_number = num;

\_suite = suite;

}

bool operator==(const Deck::Card& card2){

if( card2.\_number == this->\_number && card2.\_suite == this->\_suite ) return true;

return false;

}

bool operator==(const int num){

if(num == this->\_number) return true;

return false;

}

bool operator>(const Deck::Card& card2){

return this->\_number > card2.\_number;

}

bool operator>(const int num){

return this->\_number > num;

}

bool operator<(const Deck::Card& card2){

return this->\_number < card2.\_number;

}

bool operator<(const int num){

return this->\_number < num;

}

bool operator>=(const Deck::Card& card2){

return this->\_number >= card2.\_number;

}

bool operator>=(const int num){

return this->\_number >= num;

}

bool operator<=(const Deck::Card& card2){

return this->\_number <= card2.\_number;

}

bool operator<=(const int num){

return this->\_number <= num;

}

inline std::string name(){ return numToVal() + \_suite; }

inline std::string suite() { return \_suite; }

inline int num() { return \_number; }

private:

int \_number{}; //PHYSICAL CARD NUMBER, ANYTHING ABOVE 10 CORRELATES TO THE FACE CARDS. (i.e JOKER = 11, QUEEN = 12, KING = 13}

std::string \_suite{};

inline std::string numToVal(){

if(\_number == 1) return "Ace";

if(\_number <=10) return std::to\_string(\_number);

switch(\_number){

case 11: return "Jack";

case 12: return "Queen";

case 13: return "King";

default: return "Jack";

}

}

};

Deck(int size){

CardDeck.reserve(size);

}

Deck(){

CardDeck.reserve(52);

}

Deck(bool create){

if(create) CreateRandomDeck(\_SUITES);

}

Deck& operator +=(Deck::Card card){

CardDeck.push\_back(card);

return \*this;

}

Deck& operator -=(Deck::Card card){

//FIND IF CARD PRESENT, IF IT IS DELETE IT FROM CardDeck

if( std::find(CardDeck.begin(), CardDeck.end(), card) != CardDeck.end()){

CardDeck.erase(std::find(CardDeck.begin(), CardDeck.end(), card));

}

return \*this;

}

//THE DECK ITSELF

std::vector<Deck::Card> CardDeck{};

//GETTERS

inline int Size(){ return CardDeck.size(); }

//PUBLIC FUNCS

void CreateRandomDeck();

void CreateRandomDeck(const std::string SUITES[]);

void PrintDeck(int format);

void bublSrt();

void selSrt();

int linSrch(int val);

void Shuffle();

void Sort(int type);

Deck::Card GetRandomCard();

Deck::Card ReturnAndRemove(int num);

private:

inline const static std::string \_SUITES[4] = {

" of Spades",

" of Hearts",

" of Clubs",

" of Diamonds"

};

};

#endif /\* DECK\_H \*/

**Deck.cpp**

#include "Deck.h"

//PUBLIC FUNCS

void Deck::CreateRandomDeck(){

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

for(int j = 1; j <= 13; ++j){

CardDeck.emplace\_back(j, \_SUITES[i]);

}

}

}

void Deck::CreateRandomDeck(const std::string SUITES[]){

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

for(int j = 1; j <= 13; ++j){

CardDeck.emplace\_back(j, SUITES[i]);

}

}

}

void Deck::PrintDeck(int format = 1){

switch(format){

case 1:

for(Deck::Card &card: CardDeck){

std::cout << card.name() << '\n';

}

break;

case 2:

for(Deck::Card &card: CardDeck){

std::cout << card.name() << ", ";

}

}

}

void Deck::bublSrt(){

for(int max = CardDeck.size() - 1; max > 0; max --){

for(int index = 0; index < max; index++){

if(CardDeck[index] > CardDeck[index + 1])

std::swap(CardDeck[index], CardDeck[index + 1]);

}

}

}

void Deck::selSrt(){

int minIndex, minValue;

for(int i = 0; i < (CardDeck.size() - 1); i++){

minIndex = i;

minValue = CardDeck[i].num();

for(int index = i + 1; index < CardDeck.size(); index++){

if(CardDeck[index].num() < minValue){

minValue = CardDeck[index].num();

minIndex = index;

}

}

std::swap(CardDeck[minIndex],CardDeck[i]);

}

}

int Deck::linSrch(int val){

int index = -1;

for(int i = 0; i < CardDeck.size(); i++){

if(CardDeck.at(i) == val){

index = i;

return index;

}

}

return index;

}

void Deck::Shuffle(){

std::shuffle(CardDeck.begin(), CardDeck.end(), std::default\_random\_engine(time(0)));

}

void Deck::Sort(int type = 0){

//std::sort(CardDeck.begin(), CardDeck.end());

if(type == 0)

bublSrt();

else

selSrt();

}

Deck::Card Deck::GetRandomCard(){

return CardDeck.at(rand()% CardDeck.size());

}

Deck::Card Deck::ReturnAndRemove(int num){

if(linSrch(num) != -1){

Deck::Card card = CardDeck.at(std::distance(CardDeck.begin(), std::find(CardDeck.begin(), CardDeck.end(), num)));

if(std::find(CardDeck.begin(), CardDeck.end(), card) != CardDeck.end())

CardDeck.erase(std::find(CardDeck.begin(), CardDeck.end(), card));

return card;

}

Deck::Card errorCard;

return errorCard;

}

**GoFish.h**

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/cppFiles/file.h to edit this template

\*/

/\*

\* File: GoFish.h

\* Author: Kendra

\*

\* Created on December 8, 2024, 5:35 AM

\*/

#ifndef GOFISH\_H

#define GOFISH\_H

#include "Deck.h"

#include <chrono>

#include <thread>

#include <fstream>

#include <ctime>

class GoFish{

public:

GoFish(Deck\* GameDeck, Deck\* PlayerDeck, Deck\* EnemyDeck, Deck\* Pile){

\_GameDeck = GameDeck;

\_PlayerDeck = PlayerDeck;

\_EnemyDeck = EnemyDeck;

\_Pile = Pile;

\_GameLogs.open("logs.txt",std::fstream::app);

}

~GoFish(){

delete \_GameDeck;

delete \_PlayerDeck;

delete \_EnemyDeck;

delete \_Pile;

LogSessionStats();

\_GameLogs.close();

}

//CHECKS THE WIN CONDITION

bool CheckFour(Deck\* deck, int num);

//CHECKS WIN CONDITION

bool CheckFour(Deck\* deck);

//LOG SESSION STATS

void LogSessionStats();

//FIND A CARD OBJECT, FIND IT FROM A DECK. ADD IT TO 'to' DECK, REMOVE CARD FROM 'from' DECK

void TakeFromDeck(Deck\* from, Deck\* to, Deck::Card card, int check);

//FIND A CARD NUMBER, KEEP TAKING FROM THE DECK UNTIL IT IS NO LONGER PRESENT

bool WhileTakeFromDeck(Deck\* from, Deck\* to, int num);

//GAMEPLAY TURN

void PromptTurn(int turn, int& checkNum);

void Reset();

void PrintBooks();

void DisplayHand(Deck\* deck);

void DisplayRules();

void DisplayVictory();

void MainMenu();

//GAMEPLAY LOOP

void StartGame();

private:

//YMD

time\_t now = time(0);

char\* date = ctime(&now);

//GAME RULES

const int \_FOUROFKIND = 4;

const int \_PLAYER\_HANDSIZE = 7;

//GAME DECKS

Deck\* \_GameDeck = nullptr;

Deck\* \_PlayerDeck = nullptr;

Deck\* \_EnemyDeck = nullptr;

Deck\* \_Pile = nullptr;

//GAME STATS

unsigned int \_turn = 1; //CURRENT TURN

unsigned int \_playerBooks = 0; //BOOKS ARE THE # OF 4 OF A KINDS

unsigned int \_enemyBooks = 0;

float \_playerPoints = 0;

float \_enemyPoints = 0;

bool \_gameStarted = false;

//GAME LOGS

std::ofstream \_GameLogs{};

int \_SessionWtoL[2][1] = {{0}, {0}}; //SESSIONS WINS TO LOSSES ([0][] = PLAYER, [1][] = ENEMY)

int \_SessionGameCount = 0; //NUMBER OF GAMES PLAYED IN THIS SESSION

int \_SessionPlayerPoints[10]; //SESSION POINTS, INDEX INDICATES GAME

int \_SessionEnemyPoints[10];

};

#endif /\* GOFISH\_H \*/

**GoFish.cpp**

#include "GoFish.h"

//CHECKS THE WIN CONDITION

bool GoFish::CheckFour(Deck\* deck, int num){

if( std::count(deck->CardDeck.begin(), deck->CardDeck.end(), num) == \_FOUROFKIND){ return true; }

return false;

}

//CHECKS WIN CONDITION

bool GoFish::CheckFour(Deck\* deck){

//CHECKS FOR 4 OF ALL NUMBERS

for(int i = 1; i <= 13; i++){

if( std::count(deck->CardDeck.begin(), deck->CardDeck.end(), i) == \_FOUROFKIND){

return true;

}

}

return false;

}

void GoFish::LogSessionStats(){

if(\_SessionGameCount == 0) return;

\_GameLogs << "Session Stats:\nPlayer Wins: " << \_SessionWtoL[0][0] << '\n';

for(int i = 0; i < \_SessionGameCount; ++i){

\_GameLogs << "Game " << i+1 << ", you had: " << \_SessionPlayerPoints[i] << " points. The enemy had: " << \_SessionEnemyPoints[i] << " points.\n";

}

}

//FIND A CARD OBJECT, FIND IT FROM A DECK. ADD IT TO 'to' DECK, REMOVE CARD FROM 'from' DECK

void GoFish::TakeFromDeck(Deck\* from, Deck\* to, Deck::Card card, int check = 1){

if(check == 1){

if( std::find(from->CardDeck.begin(), from->CardDeck.end(), card) != from->CardDeck.end()){

\*to += card;

\*from -= card;

}

}

else{

\*to += card;

\*from -= card;

}

}

//FIND A CARD NUMBER, KEEP TAKING FROM THE DECK UNTIL IT IS NO LONGER PRESENT

bool GoFish::WhileTakeFromDeck(Deck\* from, Deck\* to, int num){

auto itr = std::find(from->CardDeck.begin(), from->CardDeck.end(), num);

//GO FISH!

if(itr == from->CardDeck.end()){

return false;

}

//WHILE THAT NUMBER OF CARD CAN BE FOUND IN A DECK,

int count = 0;

while(itr != from->CardDeck.end()){

//ADD THAT CARD TO THE 'to' DECK, AND REMOVE IT FROM THE 'from' DECK

to->CardDeck.push\_back(from->ReturnAndRemove(num));

//ADVANCE THE ITERATOR

itr = std::find(itr++, from->CardDeck.end(), num);

count++;

}

//DISPLAY WHAT CARD WAS TOOK

std::cout << "\nTook " << count << " " << num << "'s.\n";

return true;

}

//GAMEPLAY TURN

void GoFish::PromptTurn(int turn, int& checkNum){

std::this\_thread::sleep\_for(std::chrono::milliseconds(100));

std::cout << "\n----TURN " << turn << " ----\n";

//SORT BOTH DECKS TO MAKE IT EASIER TO SEARCH THRU

\_PlayerDeck->Sort(0);

\_EnemyDeck->Sort(1);

//TURNS WILL BE DECIDED BASED ON MODULUS 2

//i.e EVEN TURNS WILL BE THE ENEMY TURN, ODD WILL BE PLAYER

//IF THE TURN IS EVEN, (IF IT IS THE PLAYERS TURN)

if(turn % 2 == 0){

//PROMPT THE PLAYER TO TAKE A CARD

DisplayHand(\_PlayerDeck);

std::cout << "\nIt's your turn! Enter a card number that you want to fish for: ";

//USER VALIDATION

while (!(std::cin >> checkNum)) {

std::cout << "Invalid input. Please enter an integer: ";

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

if(checkNum > 13) checkNum = 13;

else if(checkNum < 1) checkNum = 1;

//TAKE THE REQUESTED CARD

if(!WhileTakeFromDeck(\_EnemyDeck, \_PlayerDeck, checkNum)){

//GO FISH

std::cout << "\nGo Fish!\n";

Deck::Card tempCard = \_GameDeck->GetRandomCard();

TakeFromDeck(\_GameDeck, \_PlayerDeck, tempCard, 0);

std::cout << "You received a " << tempCard.name() << '\n';

}

//CHECK FOR BOOKS

if(CheckFour(\_PlayerDeck)){

std::cout << "\nYou have 4 " << checkNum << "'s! These cards will be removed from your hand, and your # of books will go up by 1.\n";

WhileTakeFromDeck(\_PlayerDeck, \_Pile, checkNum);

\_playerBooks++;

\_playerPoints += abs((static\_cast<float>(\_playerBooks)\*checkNum\*\_turn));

PrintBooks();

}

}

//IF THE TURN IS ODD, (IT'S THE ENEMYS TURN)

else{

//ENEMY AI

//WILL ASK YOU FOR A CARD TYPE THEY HAVE, IF THEY HAVE AN EMPTY HAND, THEY WILL ASK YOU FOR ONE OF YOUR CARD TYPES

int enemyFish{};

if(\_EnemyDeck->Size() > 0) enemyFish = \_EnemyDeck->GetRandomCard().num();

else enemyFish = \_PlayerDeck->GetRandomCard().num();

std::cout << "\nYour opponent is asking for " << enemyFish << "'s.";

if(!WhileTakeFromDeck(\_PlayerDeck, \_EnemyDeck, enemyFish)){

//GO FISH

std::cout << "\nOpponent has to Go Fish!\n";

Deck::Card tempCard = \_GameDeck->GetRandomCard();

TakeFromDeck(\_GameDeck, \_EnemyDeck, tempCard, 0);

}

if(CheckFour(\_EnemyDeck)){

std::cout << "\nYour opponent has 4 " << enemyFish << "'s! These cards will be removed from their hand, and their # of books will go up by 1.\n";

WhileTakeFromDeck(\_EnemyDeck, \_Pile, enemyFish);

\_enemyBooks++;

\_enemyPoints += abs((static\_cast<float>(\_enemyBooks)\*enemyFish\*\_turn));

PrintBooks();

}

}

//CHECK FOR BOOKS

}

void GoFish::Reset(){

\_GameDeck->CardDeck.clear();

\_PlayerDeck->CardDeck.clear();

\_EnemyDeck->CardDeck.clear();

\_Pile->CardDeck.clear();

\_playerPoints = 0;

\_enemyPoints = 0;

\_playerBooks = 0;

\_enemyBooks = 0;

\_GameDeck->CreateRandomDeck();

\_GameDeck->Shuffle();

//HAND BOTH DECKS '\_PLAYER\_HANDSIZE' RANDOM CARDS EACH

for(int i = 0; i < \_PLAYER\_HANDSIZE; ++i){

TakeFromDeck(\_GameDeck, \_PlayerDeck, \_GameDeck->GetRandomCard(), 0);

TakeFromDeck(\_GameDeck, \_EnemyDeck, \_GameDeck->GetRandomCard(), 0);

}

}

void GoFish::PrintBooks(){

std::cout <<

"Player Books: " << \_playerBooks <<

"\nEnemy Books: " << \_enemyBooks << '\n';

}

void GoFish::DisplayHand(Deck\* deck){

//DISPLAY THE HAND OF THE PLAYER TO THE PLAYER

std::cout << "\nYour current hand:\n";

\_PlayerDeck->PrintDeck(2);

}

void GoFish::DisplayVictory(){

//STORE SESSION POINTS

\_SessionPlayerPoints[\_SessionGameCount] = \_playerPoints;

\_SessionEnemyPoints[\_SessionGameCount] = \_enemyPoints;

if(\_playerBooks > \_enemyBooks){

//player won

std::cout << "\nYOU WON!!!\nCONGRATULATIONS. Points: " << \_playerPoints << "\n\n";

\_SessionWtoL[0][0]++;

}else if(\_playerBooks < \_enemyBooks){

//enemy won

std::cout << "\nYOU LOST!\nBETTER LUCK NEXT TIME. Points: " << \_playerPoints << "\n\n";

\_SessionWtoL[1][0]++;

}else{

//tied

std::cout << "\nTIED GAME!\nRARE OCCURANCE. Points: " << \_playerPoints << "\n\n";

}

\_SessionGameCount++;

}

void GoFish::MainMenu(){

char input{};

do{

std::cout << "\nWelcome to Go Fish!\n1) Start Game\n2) Rules\n3) Exit\n";

std::cin >> input;

switch(input){

case '1':

StartGame();

break;

case '2':

DisplayRules();

break;

case '3':

LogSessionStats();

\_GameLogs.close();

exit(0);

break;

default:

LogSessionStats();

\_GameLogs.close();

exit(0);

break;

}

}while(input != '3');

}

void GoFish::DisplayRules(){

std::cout <<

"\nThe objective of the game is to get as many 'books' as possible.\n" <<

"You gain a 'book' by obtaining 4 of the same type of card. (i.e 4 Jacks = 1 book)\n" <<

"The cards used to gain a 'book' will be removed from your hand.\n" << ""

"Each turn you will ask your opponent for a type of card. If they have this type of card, they must\n" <<

"give you all cards of that type. (i.e You ask for 7's, and the opponent has 3 7's. They must give you all 3, and vice versa.)\n" <<

"Face cards are asked for in ascending order after 10 with the exception of Ace which is 1. (Ace =1, Jack = 11, Queen = 12, King = 13)\n" <<

"If you ask for a number below 1, the game will assume you asked for 1's. If you ask for a number above 13, the game will assume you asked for 13's.\n" <<

"The game will end once the entire game deck has run out of cards to fish from.\n" <<

"The winner will be determined by who has more books. Points are determined by (the rank of your card \* the amount of books you have \* the current turn).\n" <<

"Points will be displayed at the end of the game.\n";

}

//GAMEPLAY LOOP

void GoFish::StartGame(){

std::cout << "\nStarting Game...\n\n";

\_GameLogs << "\nGame Session Started : " << date << std::endl;

//RESET DECKS TO ENSURE NO BUGS

Reset();

//INITIALIZE 'checkNum' WITH -1 SO YOU CANT ACCIDENTLY WIN A BOOK OFF START

int checkNum = -1;

//CHECK FOR GAME CONDITIONS IN WHILE LOOP

while(\_GameDeck->Size() > 0){

PromptTurn(\_turn++, checkNum);

//CHECK FOR BOOKS AFTER DECK IS EMPTY

if(CheckFour(\_PlayerDeck,checkNum)){

std::cout << "\nYou have 4 " << checkNum << "'s! These cards will be removed from your hand, and your # of books will go up by 1.\n";

WhileTakeFromDeck(\_PlayerDeck, \_Pile, checkNum);

\_playerBooks++;

\_playerPoints += (\_playerBooks\*checkNum\*\_turn);

PrintBooks();

}

}

//CHECK WHO WON, DISPLAY VICTORY & RECORD LOGS

DisplayVictory();

}