**Project 1**

UNO

CSC 17a 43950

Nicholas Cardinal

4/27/2015

**Introduction**

Title: UNO!

UNO is a card game.

The objective of UNO is to place all of the cards down that are in your hand.

The game is played against another player and when the size of either the players hand or the

AI’s hand is zero, the game is over.

The game UNO is played by placing a card of the same number or color of the top card

in the discard pile. You begin with a hand of seven cards and are forced to draw when you cannot place the correct card. There is a special action card called a wild that allows you to change the color of the card you may place next. The game is played this way until one player has placed his last card and they become the winner.

It is a fun two player version of UNO that can be played over and over again.

This program is important because it can help children who cannot afford a real deck of UNO cards.

Summary

Project size: About 440 lines.

The number of variable: 16

I used many concepts from the chapters covered so far is class such as structures, pointers to structures, passing in pointers to variables and structures, returning pointers to structures, dynamically allocating memory, copying values of a structure array into another. This project took me about a week and a half working from 3-5 hours per day. I also left lots of room for improvement and look forward to making this program even better.

Description

I began by creating a structure holding most of the functionality of an UNO card including color, number and wild card. I then prompt the user to enter their names into a character array which then is written to a file and then outputted to the user welcoming them to the game of UNO. I then generated a deck of 108 UNO cards to be used during the game.

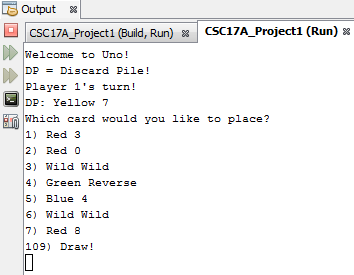
Then begins the first players turn where they are prompted to place a card which runs through

a card checker checking if it’s a valid card to place and if it’s a wild card or not. A card is valid if it is either the same color or number as the card in the discard pile which is originally generated by a function in my main that draws a card from the deck and places it in the discard pile. The player may also play a wild card that allows them to change the color of the card that can next be placed. If the player cannot place a card they are allowed to draw a card from the top of the deck. After player one’s turn player two places a card and the same protocol follows. This concept is repeated until one of the players hand is empty and the game is over.

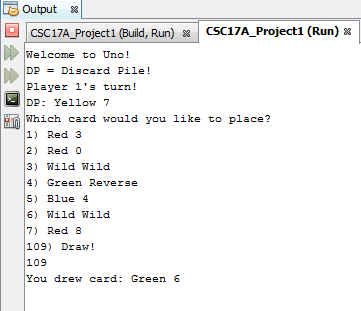
Variables

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Name | Description | Location |
| int | size | =108, hold the size of deck of UNO cards. | Line 41, main. |
| int | handSize | = 7, holds the size of players 1’s hand. | Line 42, main. |
| int | handSize2 | = 7, holds the size of player 2’s hand so it can be manipulated separately from player one’s. | Line 43, main. |
| Card | deck[size] | This structure stores every card value, numbers 1-9 and colors red, blue, green and yellow. Along with wild cards. | Line 45, main. |
| Card \* | deckPtr | = deck, a pointer to my deck structure | Line 44, main. |
| Card | discardPile | A instance of my Card structure to store the attributes of the card after it has been discarded. | Line 48, main. |
| Card \* | dpPtr | = discardPile, a pointer to my discardPile. | Line 49, main. |
| int | turn | Stores whose turn it is. | Line 51, main |
| bool | won | = false, becomes true when one of the players handSize<=0. | Line 60, main. |
| bool | drew | = false, becomes true when player chooses to draw a card, if true, the rest of the “gameTurn” function is skipped. | Line 256, gameTurn. |
| bool | accepted | = false, becomes true when the card the player attempted to place matches either the color or number of the card in the discard pile, or if the card being placed is a wild. | Line 257, gameTurn. |
| Card \* | newHand | The value of this instance of the Card structure is passed back from the drawCard function. | Line 258, gameTurn |
| Card | cardPlaced | Stores the attributes of the card the player is attempting to place and then checks it against the card in the discardPile. | Line 259, gameTurn. |
| Card \* | cpPtr | A pointer to the cardPlaced instance of the Card structure. | Line 260, gameTurn. |
| Card \*  fstream  Player | tempHnd  statsFile  p & g | Dynamically allocated hand that reassigns/copies all values of the players hand to it then adds one element and places the card they drew in that last element.  Fstream declaration used to open the a file to store the stats.  Instance of the player class to write and read the players names to and from the file. | Line 340, drawCard.  Line 418, fileIO. |

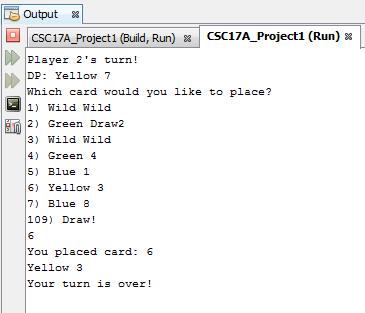
Screen shots



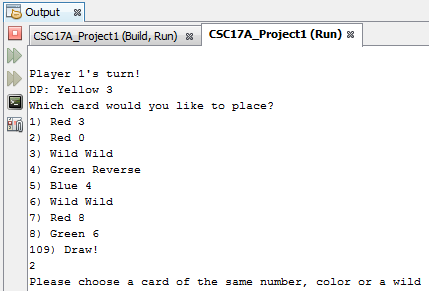
* At this point the deck of UNO cards has been generated and shuffled randomly.
* Also, both players hands have been assigned with random cards from the deck and those cards are removed from the deck.
* The top card from the deck has been placed in the discard pile and it is now the first players turn to place a card.

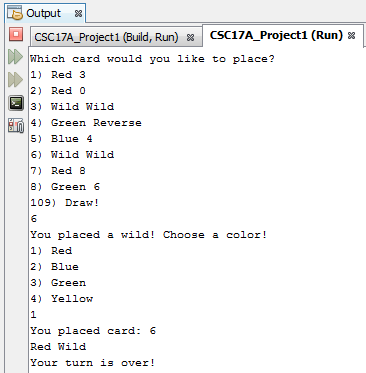
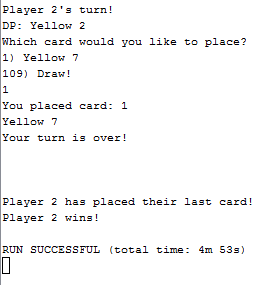


* Since player 1 could not place a card they have chosen to draw a card from the deck by entering “109.”
* This is done through the “drawCard” function that dynamically allocated an array of structures with a size one more than that of the hand passed in and copies all attributes of the hand passed in and adds one element that stores the new card that was drew off the top of the deck.

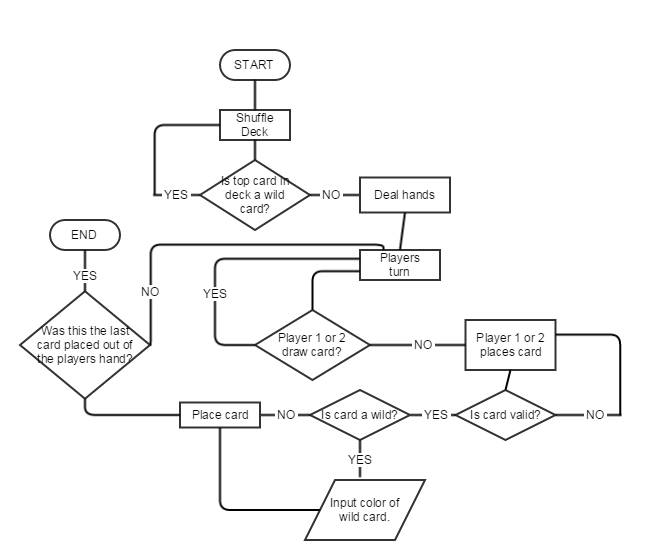


* This shot shows player 2 placing a yellow card which then passes through a card checker and compares the color and number of the card in the discard pile with the card placed by the player if it is accepted then the card is added to the discard pile.



* Here player one tries to place a Red 0 when the card in the discard pile is a Yellow 3, the user is then prompted that they attempted to place an invalid card and is prompted for input again.
* This shot shows the functionality of the WILD card, player one placed a wild card and then is prompted to choose the color they would like to change the discard pile to.
* Player one chose the color Red and the color of the Wild is then changed to Red.
* Once one of the players plays their last card the game is over.

Flow chart



**Pseudo Code**

*Start*

*Create and shuffle the deck*

*Hand out seven cards to both players*

*Prompt the user to place a card*

*If player has a valid card to be placed*

*If card is a wild*

*Prompt user for color change, place card.*

*Else*

*Place Card*

*Else*

*Call the drawCard() function.*

*If the size of either players hand is <=0*

*Won = true, output winner, game over.*

*Else*

*Next players turn.*

**Concepts**

**Memory allocation:**

Beginning with the declaration on line 330, I dynamically allocate a new array of structures. This is done in my drawCard function; the purpose is to create a temp hand for the player that is one size bigger than the hand passed in the function. Increasing the size of the array by one allows for me to add a card from the top of the deck to the players hand. This temp hand is then passed back and reassigned to the players hand with the new card they have drawn from the deck.

**Function with structures:**

You can find this concept nearly every function I used in my program. For example, in my resetDeck function I use a for loop to assign all attributes of an UNO deck to 108 cards in the deck instance of my structure, I assign attributes such as color, number and whether the card is a wild or not.

**Pointers with arrays and arrays of structures:**

Once again these concepts can be found in nearly every function in my program. For example, starting with their declarations in lines 55&56 both players hands are generated with 7 unique cards, each card is given a color, number, action and a bool value of wild. These hands are used all across the program, the attributes are compared to that of the discard pile which is an instance of the structure where a card is placed if it has the same number or color as the card currently in the discard pile. All instances of the structure are passed in and out of functions by utilizing pointers. In line 287 I return a pointer to the player’s hand where it is then reassigned in main to the original hand after a card is placed or drawn.

**Use of character arrays and reading and writing to binary files:**

In my fileIO function on line 412, I utilize a structure containing a character array; I use this array to read in the name of each player. I then open a file for output in binary and output the players name to the file. I then read back the named and welcome the players to the game of UNO.

**Program:**

#include <cstdlib>

#include <iostream>

#include <cstring>

#include <ctime>

#include <fstream>

using namespace std;

enum Action { blank, DRAW2, DRAW4, WILD, SKIP, REVERSE };

//User Libraries

#include "Card.h"

#include "Player.h"

//Functions

void changeActionCard(int &, int &, int &, string &); //Adds the action cards to un-shuffled deck.

void shuffleDeck(Card \*, int &); //Shuffles deck and returns the pointer to the deck.

void resetDeck(Card \*, int&);//Sets deck back to default.

void outputDeck(Card \*, int);//Outputs deck.

Card\* fillHand(Card \*, Card \*, int&);//Fills players hands with random cards from the deck.

void outputHand(Card \*, int); //Displays the players hand after it is randomly picked from the deck.

void gameBegin(Card \*, int &, Card \*, int);//Places the first card of the shuffled deck down.

Card\* gameTurn(Card \*, int &, Card \*, int &, Card \*, int &, Card \*, int &, int &);//Handles all actions of players turn.

bool checkCard(Card \*, Card \*);//Checks to see if the card placed is valid

int cardPlace(Card \*, int); //Outputs players hand and reads in their choice.

Card\* drawCard(Card \*, int &, Card \*, int &); //Draws a card from the deck.

void checkAction(Card \*, int &, Card \*, int &, Card \*, int &);//Checks what action card was placed.

void removeCard(Card \*, int &, int &, int &);//Removes the card chosen from the players hand.

void dpOut(Card \*, int);//Outputs the discard pile.

void pOrD(bool, Card \*, int, Card \*, int); //Checks whether a card is drawn or placed;

void fileIO(); //Writes the players name to a file and then reads them out of the file.

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

srand(time(0));

int size = 108;

int handSize = 7;

int handSize2 = 7;

Card \*deckPtr;//Pointer to the deck.

Card deck[size];//Holds the deck.

deckPtr = deck;//Assigns pointer to the deck.

int dpSize = 1;

Card discardPile[dpSize];

Card \*dpPtr;

dpPtr = discardPile;

int turn = 1;

fileIO();

resetDeck(deckPtr, size);

shuffleDeck(deckPtr, size);

Card \*hand = new Card[handSize];

Card \*hand2 = new Card[handSize2];

cout << "Welcome to Uno!" << endl;

cout << "DP = Discard Pile!" << endl;

hand = fillHand(hand, deckPtr, size);

hand2 = fillHand(hand2, deckPtr, size);

gameBegin(deckPtr, size, dpPtr, dpSize);

bool won = false;

while(won == false){

cout << "Player 1's turn!" << endl;

dpOut(discardPile, turn);

hand = gameTurn(hand, handSize, hand2, handSize2, dpPtr, dpSize, deckPtr, size, turn);

if(handSize2 == 0 || handSize == 0){

turn = 1;

won == true;

break;

}

cout << "Player 2's turn!" << endl;

dpOut(discardPile, turn);

hand2 = gameTurn(hand2, handSize2, hand, handSize, dpPtr, dpSize, deckPtr, size, turn);

if(handSize2 == 0 || handSize == 0){

turn = 2;

won == true;

break;

}

}

cout << "Player " << turn << " has placed their last card!" << endl;

cout << "Player " << turn << " wins!" << endl;

delete [] hand;

delete [] hand2;

return 0;

}

void changeActionCard(int &aCrdTyp, int &cNum, int &aCard, string &aCrdStr){

aCrdTyp++;

if(aCrdTyp >= 0 && aCrdTyp <= 2){//Set action card to reverse

cNum = -1;

aCard = REVERSE;

aCrdStr = "Reverse";

}

else if(aCrdTyp > 2 && aCrdTyp <= 4){//Set action card to Draw2

aCard = DRAW2;

cNum = -1;

aCrdStr = "Draw2";

}

else{//Set action card to Skip

aCard = SKIP;

cNum = -1;

aCrdStr = "Skip";

}

}

void shuffleDeck(Card \*deck, int &size){

int shuffle = 0;

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

shuffle = rand() % 108 + 0;

swap(deck[i],deck[shuffle]);

}

}

void resetDeck(Card \*deck, int &size){

int cNum = 1; //Holds a value 1-9 that is to be stored in .number

string cColor;//Holds the string color(red,green,blue,yellow) & stores in .color

int aCard;//Holds the action card(draw4,wild,reverse,draw2,skip) & stores in .action

int aCrdTyp = 0;//After storing 2 action cards the action card is changed

string aCrdStr;

for(int i=0;i<size;i++){//Loops in order to change the color of the card stored in .color

if(i < 24){//If parameters met set card color to Blue

aCard = 0;

cColor = "Blue";

if(i >= 18){//If i is > 18 the color begin placing action cards.

changeActionCard(aCrdTyp, cNum, aCard, aCrdStr);

}

}

else if(i > 24 && i < 48){//If parameters met set card color to Green

if(aCrdTyp > 6){

aCrdTyp = 0;

}

aCard = 0;

cColor = "Green";

if(i >= 42){//If i is > 42 the color begin placing action cards.

changeActionCard(aCrdTyp, cNum, aCard, aCrdStr);

}

}

else if(i > 48 && i < 72){//If parameters met set card color to Red

if(aCrdTyp > 6){

aCrdTyp = 0;

}

aCard = 0;

cColor = "Red";

if(i >= 66){//If i is > 66 the color begin placing action cards.

changeActionCard(aCrdTyp, cNum, aCard, aCrdStr);

}

}

else if(i > 72 && i < 96){//If parameters met set card color to Yellow

if(aCrdTyp > 6){

aCrdTyp = 0;

}

aCard = 0;

cColor = "Yellow";

if(i >= 90){//If i is > 90 the color begin placing action cards.

changeActionCard(aCrdTyp, cNum, aCard, aCrdStr);

}

}

deck[i].number = cNum;

deck[i].color = cColor;

deck[i].action = aCard;

deck[i].strAction = aCrdStr;

deck[i].wild = false;

cNum++; //Increases from 1 - 9 then resets if cNum = 10;

int count = 0;//Stores the number of times the cNum variable has been reset.

if(cNum == 10){

cNum = 1;

count++;

}

}

//Adds the missing 9's to the deck.

deck[97].number = 9;

deck[97].color = "Green";

deck[97].wild = false;

deck[98].number = 9;

deck[98].color = "Red";

deck[98].wild = false;

deck[99].number = 9;

deck[99].color = "Yellow";

deck[99].wild = false;

for(int i=100;i<108;i++){//Adds wilds to the deck.

deck[i].action = WILD;

deck[i].number = -1;

deck[i].wild = true;

deck[i].color = "Wild";

deck[i].strAction = "Wild";

}

}

void outputDeck(Card \*deck, int size){

cout << "Default deck " << endl;

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

if(deck[i].number < 0){

cout << i << "Card Value: " << deck[i].strAction;

}

else{

cout << i <<"Card Value: " << deck[i].number;

}

cout << " Color: " << deck[i].color;

cout << endl;

}

}

Card\* fillHand(Card \*hand, Card \*deck, int &size){

int randCrd;

for(int i=0;i<7;i++){

randCrd = rand() % size+0;

hand[i].action = deck[randCrd].action;

hand[i].color = deck[randCrd].color;

hand[i].number = deck[randCrd].number;

hand[i].wild = deck[randCrd].wild;

hand[i].strAction = deck[randCrd].strAction;

swap(deck[randCrd],deck[size-1]);

size--;

}

return hand;

}

void outputHand(Card \*hand, int handSize){

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

if(hand[i].number < 0){

cout << i+1 << ") " << hand[i].color << " " << hand[i].strAction << endl;

}

else{

cout << i+1 << ") " << hand[i].color << " " << hand[i].number << endl;

}

}

}

void gameBegin(Card \*deck, int &size, Card \*discardPile, int dpSize){

int randCrd = rand()% size+0;

if(deck[randCrd].wild == true){

gameBegin(deck, size, discardPile, dpSize);

}

else{

discardPile[0].action = deck[randCrd].action;

discardPile[0].color = deck[randCrd].color;

discardPile[0].number = deck[randCrd].number;

discardPile[0].wild = deck[randCrd].wild;

discardPile[0].strAction = deck[randCrd].strAction;

swap(deck[randCrd], deck[size-1]);

size--;

}

}

Card\* gameTurn(Card \*hand, int &handSize, Card \*hand2, int &hand2Size, Card \*discardPile, int &dpSize, Card \*deck, int &deckSize, int &turn){

int choice = cardPlace(hand, handSize); //Stores the choice of the users card.

bool drew = false;

bool accepted = false;

Card \*newHand;

Card cardPlaced[1]; //Stores the card that the user attempts to place.

Card \*cpPtr;

cpPtr = cardPlaced;

if(choice == 109){

newHand = drawCard(hand, handSize, deck, deckSize);

drew = true;

}

if(choice < 0){

choice = cardPlace(hand, handSize);

}

if(drew == false){

cpPtr[0].action = hand[choice-1].action;

cpPtr[0].color = hand[choice-1].color;

cpPtr[0].number = hand[choice-1].number;

cpPtr[0].strAction = hand[choice-1].strAction;

cpPtr[0].wild = hand[choice-1].wild;

accepted = checkCard(cpPtr, discardPile);

}

while(accepted == false && drew == false){

cpPtr[0].action = hand[choice-1].action;

cpPtr[0].color = hand[choice-1].color;

cpPtr[0].number = hand[choice-1].number;

cpPtr[0].strAction = hand[choice-1].strAction;

cpPtr[0].wild = hand[choice-1].wild;

accepted = checkCard(cpPtr, discardPile);

if(accepted == false){

choice = cardPlace(hand, handSize);

}

}

if(drew == false){

checkAction(hand, handSize, deck, deckSize, discardPile, turn);

}

pOrD(drew, newHand, handSize, discardPile, choice);

if(drew == true){

return newHand;

}

else{

removeCard(hand, handSize, choice, turn);

return hand;

}

}

bool checkCard(Card \*cardPlaced, Card \*discardPile){

if(cardPlaced[0].color == discardPile[0].color ||

cardPlaced[0].number == discardPile[0].number ||

cardPlaced[0].wild == true){

discardPile[0].action = cardPlaced[0].action;

discardPile[0].color = cardPlaced[0].color;

discardPile[0].number = cardPlaced[0].number;

discardPile[0].strAction = cardPlaced[0].strAction;

return true;

}

else{

cout << "Please choose a card of the same number, color or a wild" << endl;

return false;

}

}

int cardPlace(Card \*hand, int handSize){

int choice;

cout << "Which card would you like to place?" << endl;

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

if(hand[i].number < 0){

cout << i+1 << ") " << hand[i].color << " " << hand[i].strAction << endl;

}

else{

cout << i+1 << ") " << hand[i].color << " " << hand[i].number << endl;

}

}

cout << "109) Draw!" << endl;

cin >> choice; //FIX SO IT DOESN'T ACCEPT CHARS

if(choice == 109){

return choice;

}

if(choice < 1 || choice > handSize){

return -1;

}

return choice;

}

Card\* drawCard(Card \*hand, int &handSize, Card \*deck, int &deckSize){

Card \*tempHnd = new Card[handSize+1]; //Stores the hand passed in and adds one element.

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

tempHnd[i].action = hand[i].action;

tempHnd[i].color = hand[i].color;

tempHnd[i].number = hand[i].number;

tempHnd[i].strAction = hand[i].strAction;

tempHnd[i].wild = hand[i].wild;

}

tempHnd[handSize].action = deck[0].action;

tempHnd[handSize].color = deck[0].color;

tempHnd[handSize].number = deck[0].number;

tempHnd[handSize].strAction = deck[0].strAction;

tempHnd[handSize].wild = deck[0].wild;

swap(deck[0],deck[deckSize-1]);

deckSize--;

handSize++;

return tempHnd;

}

void checkAction(Card \*hand, int &handSize, Card \*deck, int &deckSize, Card \*discardPile, int &turn){

if(discardPile[0].action == WILD){

int pColor;

string pColors;

cout << "You placed a wild! Choose a color!" << endl;

cout << "1) Red " << endl << "2) Blue " << endl << "3) Green " << endl

<< "4) Yellow" << endl;

cin >> pColor;

while(pColor < 0 && pColor > 4){

cout << "Please choose a valid number." << endl;

cin >> pColor;

}

if(pColor==1){pColors = "Red";}

else if(pColor==2){pColors = "Blue";}

else if(pColor==3){pColors = "Green";}

else{pColors == "Yellow";}

discardPile[0].color = pColors;

}

else if(discardPile[0].action == SKIP || discardPile[0].action == REVERSE){

if(turn == 1){

turn = 2;

}

else{

turn = 1;

}

}

}

void removeCard(Card \*hand, int &handSize, int &choice, int &turn){

swap(hand[choice-1], hand[handSize-1]);

handSize--;

cout << "Your turn is over!" << endl << endl << endl << endl;

}

void dpOut(Card \*discardPile, int turn){

if(discardPile[0].number < 0){

cout << "DP: " << discardPile[0].color << " " << discardPile[0].strAction << endl;

}

else{

cout << "DP: " << discardPile[0].color << " " << discardPile[0].number << endl;

}

}

void pOrD(bool drew, Card \*newHand, int handSize, Card \*discardPile, int choice){

if(drew == true){

cout << "You drew card: ";

if(newHand[handSize-1].number < 0){

cout << newHand[handSize-1].color << " " << newHand[handSize-1].strAction <<endl<<endl<<endl<<endl;

}

else{

cout << newHand[handSize-1].color << " " << newHand[handSize-1].number <<endl<<endl<<endl<<endl;

}

}

else{

cout << "You placed card: " << choice << endl;

if(discardPile[0].number < 0){

cout << discardPile[0].color << " " << discardPile[0].strAction << endl;

}

else{

cout << discardPile[0].color << " " << discardPile[0].number << endl;

}

}

}

void fileIO(){

Player g;

Player p;

Player p2;

Player g2;

int turn = 1;

fstream statsFile;

const int size = 20;

cout << "Enter your name Player " << turn << endl;

cin.getline(g.name, size, '\n');

statsFile.open("player1.txt", ios::out | ios::binary);

statsFile.write((char\*)&g, sizeof(Player));

statsFile.close();

turn++;

cout << "Enter your name Player " << turn << endl;

cin.getline(p.name, size, '\n');

statsFile.open("player2.txt", ios::out | ios::binary);

statsFile.write((char\*)&p, sizeof(Player));

statsFile.close();

statsFile.open("player1.txt", ios::in | ios::binary);

statsFile.read((char\*)&g2, sizeof(Player));

cout << "Welcome " << g2.name << "!" << endl;

statsFile.close();

statsFile.open("player2.txt", ios::in | ios::binary);

statsFile.read((char\*)&p2, sizeof(Player));

cout << "Welcome " << p2.name << "!" << endl;

statsFile.close();

cout << "HAVE FUN PLAYING UNO!" << endl << endl << endl;

}