Project 1

<Uno Game>

CSC-17C
Rachel Scherer
Date: 10/31/18

Introduction

Uno is a simple card game with a custom set of cards.

Each player draws seven cards from a deck of 108. Aside from “wild”, and “wild draw 4” cards, each deck contains cards with both a color (red, yellow, green, blue) and value (0-9, reverse, skip, and draw 2). The objective of the game is to get rid of your own cards by taking turns placing a card of a matching color and/or value to the card in the center.

“Wild” cards allow you to change the color at play. “Reverse” switches the order from clockwise to counterclockwise. “Skip” skips the next player’s turn, “Draw 2” forces the next player to draw 2 and skip their turn, and “Wild Draw 4” allows you to change the color at play while also causing the next player to draw 4 and skip their turn.

At the end of the game, the points are tallied using the all player’s card’s numeric values, or 20 points for the special cards and 50 points for the wild cards. These points all go to the player who got rid of their cards first.

Summary

Project size: 1105 Lines Total

- **765 lines of code**
- **265 commented lines**
  - **29 shared with code**
- **104 blank lines**

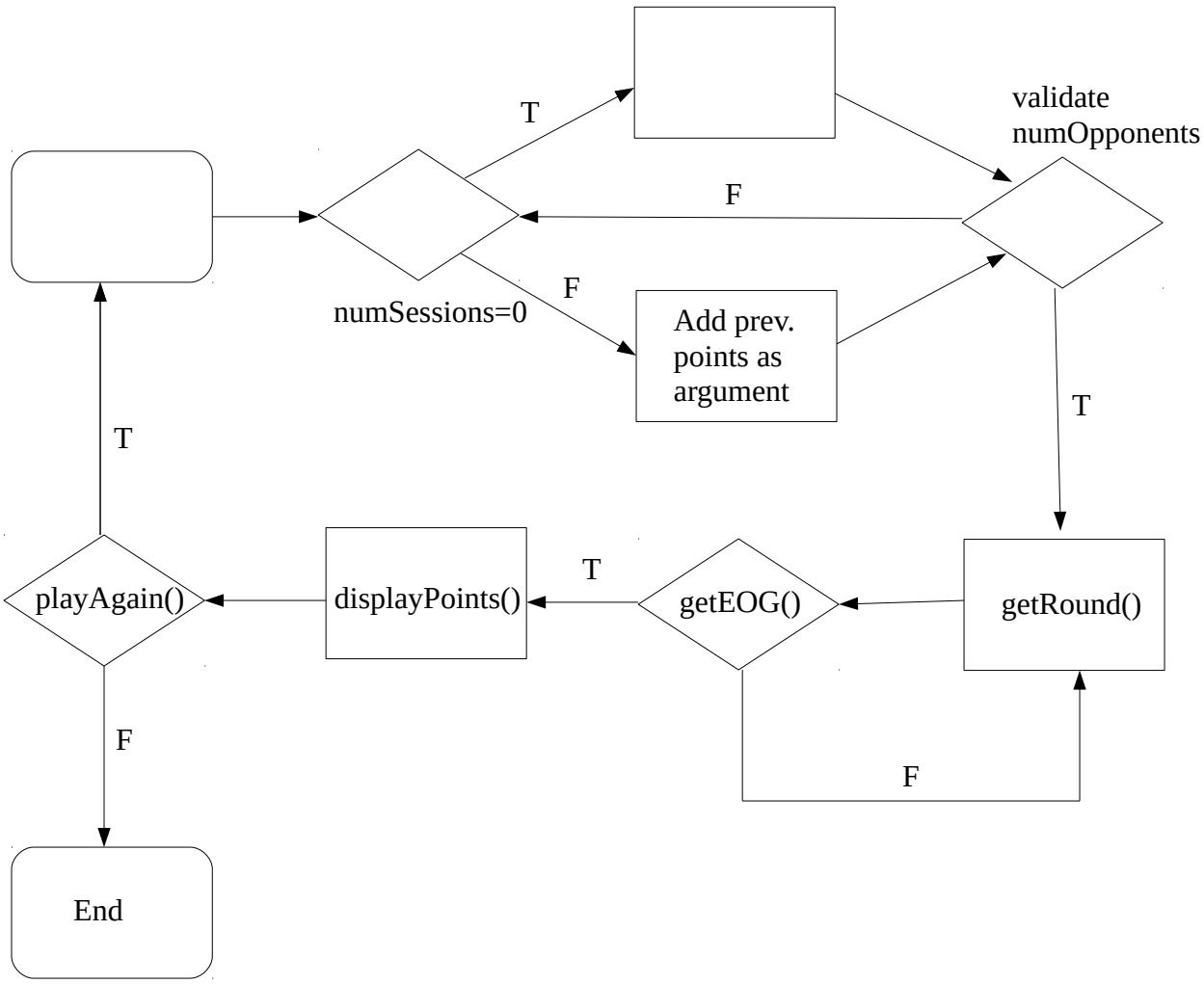
This project utilizes object-oriented design and the C++ STL to run an Uno clone in the console. This project was a challenge as prior to two weeks ago, I had never used the STL. I found myself researching and implementing complex (and not so complex) solutions to the inevitable errors that come with learning new realms of the language (which for me included iterators, nested containers, algorithms, and usage of typedef). Considering it only took about one week, I am proud of what I was able to accomplish in that time.

The only feature that is currently missing from this project is the namesake feature, the need to call Uno after having only one card left (and the need to call out the other players as they have one card left). Based on my research, this requires multithreading which is something I am excited to learn and come back to this project with.

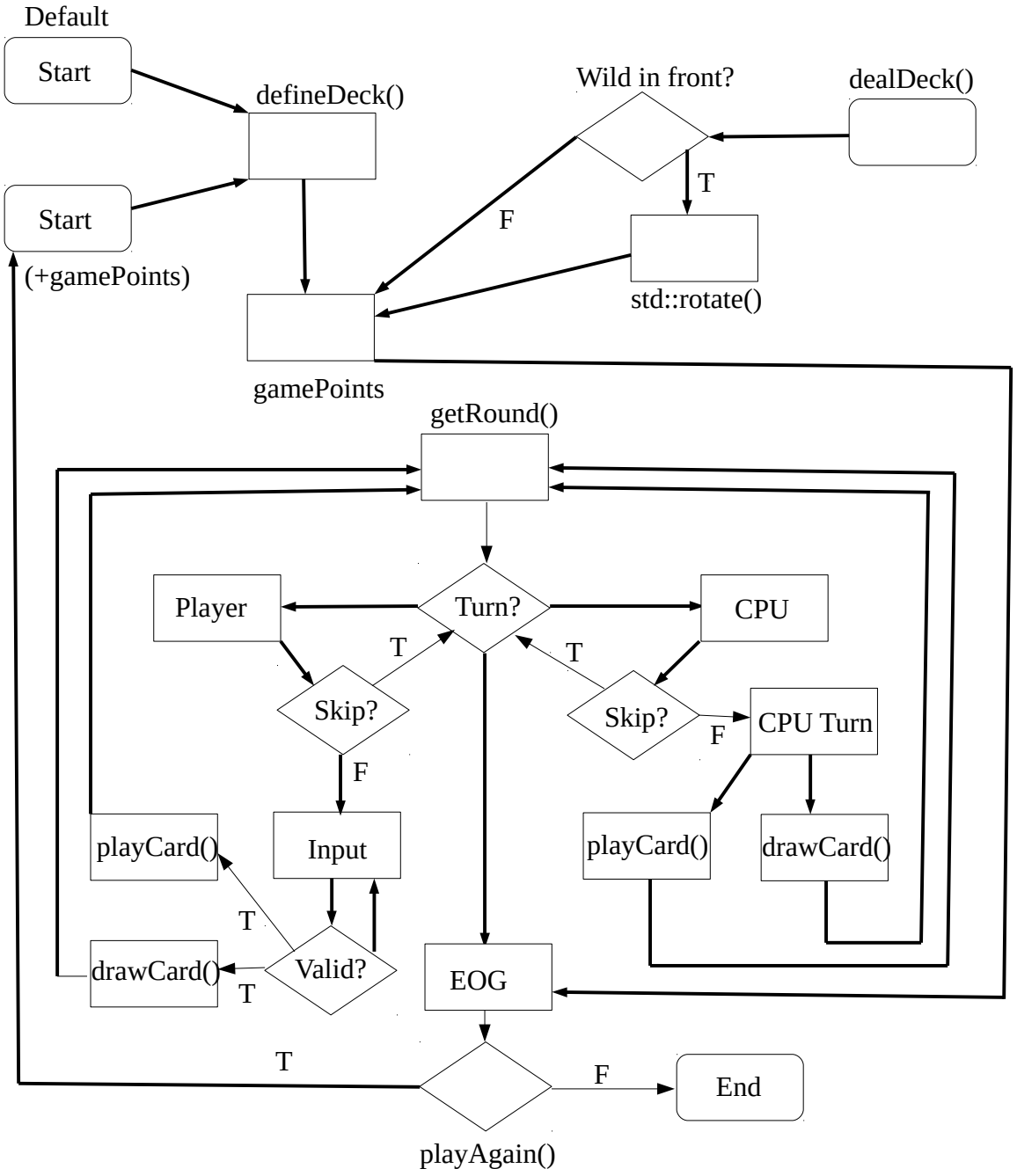
Description

This project was created as a means of getting more comfortable with the STL, which I am!

Flow Chart (main.cpp)



Flow Chart (Game.cpp)



Variables

Type	Variable Name	Description	Location
integer	numSess	Game counter	main()
vector<Game>	sess	Holds current iteration of game	Main()
String	numOpp	Holds number of opponents from user	Main()
deque<pair<int,string>>*	plrHnd	Holds each player's set of cards	Game.cpp
deque<pair<int,string>>	deck	Holds the rest of the cards	Game.cpp
vector<int>	gamePts	Total number of points earned by each player across multiple games	Game.h, Game.cpp
boolean	eog	End of game flag	Game.h, Game.cpp
boolean	skip	Skip turn flag	Game.h, Game.cpp
boolean	reverse	Reverse flag	Game.h, Game.cpp
boolean	wd4Mtch	Wild Draw 4 bluff flag	Game.h, Game.cpp
integer	numOpp	Holds number of Opponents	Game.h, Game.cpp
integer	numDrwn	Holds total number of cards drawn	Game.h, Game.cpp
integer	winner	Holds numeric value of winner	Game.h, Game.cpp
string	wColor	Holds selected color during Wild and Wild Draw 4 plays	Game.h, Game.cpp
Integer	count	Used to make sure we don't deal more cards than we have	dealDeck()
Integer	oppInt	Holds int value of user input when setting numOpponents.	setNumOpponents()
string	input	Used to validate user input	getRound()
integer	turn	Keeps track of whose turn it is	getRound()
string	name	Holds name associated with a given card.	displayCard()
integer	iInt	Holds int value of user input when validating card	checkCardValidity()
string	wInput	Holdes user input when deciding to bluff on Wild Draw 4.	CheckCardValidity(), getCardColorOrder()
multimap<int,int>	srtCnt	Holds number of iterations of each color card in a hand.	ComputerTurn(), getCardColorOrder()
map<int,int>::reverse_iterator	itr	Used to iterate backwards through countHelper	computerTurn()
map<int,int>	count	Initializes values before they're copied to countHelper.	getCardColorOrder()
map<int,int>::iterator	itr	Used to move from count to countHelper.	getCardColorOrder()
Integer	chance	Holds chance of computer calling your bluff	playCard()
Integer	rndCPU	Calls on random CPU to be the one to call your bluff,	playCard()
String	input	Used to validate user input.	playAgain()
string	cInput	Used to validate user input.	playCard()

Cross Reference for Project 1

You are to fill-in with where located in code

Chapter	Section	Topic	Where Line #"s	Pts	Notes
2	2	cout	main.cpp 30		
	3	libraries	main.cpp 9-10	8	iostream, iomanip, cmath, cstdlib, fstream, string, ctime
	4	variables/literals	main.cpp, 18		No variables in global area, failed project!
	5	Identifiers	main.cpp, 17		
	6	Integers	main.cpp, 18	3	
	7	Characters	game.cpp, 303	3	
	8	Strings	main.cpp, 31	3	
	9	Floats No Doubles	N/A (Not needed)	3	Using doubles will fail the project, floats OK!
	10	Bools	game.h, 37	4	
	11	Sizeof *****			
	12	Variables 7 characters or less	Throughout		All variables <= 7 characters
	13	Scope ***** No Global Variables			
	14	Arithmetic operators	game.cpp, 112		
	15	Comments 20%+	Throughout	5	Model as pseudo code
	16	Named Constants	N/A (Not needed)		All Local, only Conversions/Physics/Math in Global area
	17	Programming Style ***** Emulate			Emulate style in book/in class repositiory
3	1	cin	main.cpp, 37		
	2	Math Expression	game.cpp, 112		
	3	Mixing data types ****			
	4	Overflow/Underflow ****			
	5	Type Casting	game.cpp, 159	4	
	6	Multiple assignment *****			
	7	Formatting output	Throughout	4	
	8	Strings	main.cpp, 31	3	
	9	Math Library	N/A (Not needed)	4	All libraries included have to be used
	10	Hand tracing ******			
4	1	Relational Operators	game.cpp, 112		
	2	if	game.cpp, 112-115	4	Independent if
	4	If-else	main.cpp, 21-29	4	
	5	Nesting	Everywhere	4	
	6	If-else-if	game.cpp, 533-548	4	
	7	Flags *****			
	8	Logical operators	game.cpp, 336	4	
	11	Validating user input	main.cpp, 38	4	
	13	Conditional Operator	game.cpp, 241	4	
	14	Switch	game.cpp, 254-268	4	
5	1	Increment/Decrement	game.cpp, 54	4	
	2	While	game.cpp, 125-128	4	
	5	Do-while	main.cpp, 19-50	4	
	6	For loop	game.cpp, 54-57	4	
	11	Files input/output both	N/A (Not needed)	8	
	12	No breaks in loops ******			Failed Project if included
****** Not required to show			Total	100	

Psuedocode – main.cpp

```
Int main
    Vector of “Games” sess
    Int number of sess is 0
    Do
        If sess is 0
            call the default constructor of Game
        Else
            Pass the points from the previous game as an argument to Game
        Do
            Get number of opponents from user
        While input is valid
        Do
            Get Round
        While EOF flag is false
        Display Points
    While PlayAgain is true
```

Psuedocode – Game.cpp

```
Default constructor
    Set seed for random number
    call defineDeck()
    set wColor to ""
    set eog to false;
    set skip to false;
    set reverse to false;
    set gamePts vector to have 0 elements.

Constructor with gamePts passed as argument
    Set seed for random number
    call defineDeck()
    set wColor to ""
    set eog to false;
    set skip to false;
    set reverse to false;
    set gamePts vector equal to argument

Destructor
    Clear deck vector
    Clear each element of plrHnd array
    Delete plrHand array
    Set it equal to null

Get EOG
    Return EOG

Define Deck
    Push Red 0 to front of deck
    Push Yellow 0 to front of deck
    Push Green 0 to front of deck
    Push Red 0 to front of deck
    For I from 0 to 12
        Push two Red Is to front of deck
        Push two Yellow Is to front of deck
        Push two Green Is to front of deck
        Push two Blue Is to front of deck
    For I from 0 to 4
        Push “13” Draw 4 to front of deck
        Push “14” Draw 4 Wild to front of deck
    Shuffle the cards
```

```
Deal Deck
    Set int count to 0
    Iterate from beginning to end of deck, incrementing count
        If count is equal to number of players * 7
            break
        Push current iterator to front of plrHnd[count mod numOpp plus 1]
        Pop deck front
    While front of deck is Wild or Wild Draw 4
        Rotate front card to back of deck
    set numDrwn to 7 * numOpp+1
    For I from 0 to numOpp
        If size of gamePts is less than I
            Push gamePts back with 0
    While size of gamePts is greater than numOpp
        pop gamePts back

Set numOpp
    If the first character of input is not a digit
        Print error message, return false.
    Cast input to oppInt
    If oppInt is less than 1 or 14
        Print error message, return false
    set numOpp to oppInt
    Set plrHnd to array of size oppInt + 1 containing deque of pairs
    Call dealDeck()

Get Round
    If skip flag is set to false
        Display top of deck
        If wColor is not blank
            Display wColor //Used for wild and wild draw 4 cards
        For I from 0 to size of player 0's deck
            Display card I
        Print Draw Card option
        string input
        Do
            Get input from user
            If input is not between 0 and number of cards
                Print error
        While checkCardValidity() is false
    else
        Print that player's turn was skipped.
        Set skip flag to false.
    Int turn
    If reverse flag is true
        Set turn to numOpp
    Else
        Set turn to 0
    While turn is between 0 and numOpp inclusive
        If EOG flag is true
            return
        Call computerTurn(turn)
        If Reverse is true, increment turn. Otherwise, decrement turn.
    Print line

Display Card
    Set name to ""
    Switch numeric value of card
        Case 10
            Set name to color value of card, then " Skip". Break.
        Case 11
            Set name to color value of card, then " Reverse". Break.
```

```
        Case 12
            Set name to color value of card, then " Draw 2". Break.
        Case 13
            Set name Wild Card. Break.
        Case 14
            Set name Wild Draw 4. Break.
        Default
            Set name to color value of card, then numeric value of card.
    Return name
```

Get color

```
    Transform input to all lowercase
    If input isn't "red", "yellow", "green", or "blue"
        print error
        return false
    return true
```

Check Card Validity

```
    If first digit of input is d
        print draw card
        call drawCard(0)
        return true
    If first digit of input is h
        print how to play
        return false
    If first digit is not a number
        print error
        return false
    cast input to iInt
    If iInt is less than one or number of cards
        print error
        return false
    If the selection is a wild card
        print line
        call playCard
        return true
    Set wd4Mtch to false
    If the selection is a wild draw 4 card
        For I from 0 to size of player 0's deck
            if deck's color matches by color, number, or the color matches wColor
                set wd4Match to true
        if wd4Match is true
            string input
            do
                print warning and ask for y or n
                get input from user
                if the first character of input is not y or n
                    print error
            while input is not y or n
            if input is n
                return false
            else
                print line
                call playCard()
                return true
        else
            print line
            call playCard()
            return true
    if deck's color doesn't match by color, number, and the color doesn't match wColor
        print error
        return false
```

```
    else
        call playCard()
        return true
```

Computer Turn

```
    If skip is true
        skip is false
        Print that computer's turn was skipped
        return
    set multimap srtCnt to value returned from getCardColorOrder
    reverse map iterator itr
    if srtCnt's size is greater than 0
        For itr from srtCount's beginning to end
            If itr's second value is 0
                For I from 0 to size of cpu's hand
                    If the card is red and the value matches the deck
                        call playCard()
                        return
            If itr's second value is 1
                For I from 0 to size of cpu's hand
                    If the card is yellow and the value matches the deck
                        call playCard()
                        return
            If itr's second value is 2
                For I from 0 to size of cpu's hand
                    If the card is green and the value matches the deck
                        call playCard()
                        return
            If itr's second value is 3
                For I from 0 to size of cpu's hand
                    If the card is blue and the value matches the deck
                        call playCard()
                        return
    For I from 0 to size of cpu's hand
        if the color matches the deck's color or wColor
            call playCard()
            return
    For I from - to size of cpu's hand
        if cpu has a wild card or a wild draw 4
            call playCard()
            return
    print that computer drew a card
    call drawCard()
```

Get Card Color Order

```
    Map of two integers - Count
    For I from 0 to 4
        insert I,0 into count
    for I from 0 to size of computer's hand
        if the color is red
            increment count[0]
        if the color is yellow
            increment count[1]
        if the color is green
            increment count[2]
        if the color is blue
            increment count[3]
    For I from 0 to 4
        If deck's color is red and count[i] is less than count[0], or
        if deck's color is yellow and count[i] is less than count[1], or
        If deck's color is green and count[i] is less than count[2], or
        if deck's color is blue and count[i] is less than count[3]
```

```
            delete count[i]
    multimap of two integers srtCnt
    map iterator itr
    for iterator from beginning to end of count
        insert itr's second value into srtCnt's first, and itr's first value into strCnt's second
    return srtCnt
```

Play Card

```
    If size of player's hand is one
        Call placeCard()
        Set winner to playerNum
        Set EOG flag to true
        return
    int chance is 0
    int rndCPU is 0
    wColor is ""
    if plrNum is 0
        If they play a wild or wild draw 4
            string cInput
            do
                Ask what color they'd like it changed to
                get cInput from user
            while getColor returns false
            If cInput is red
                wColor is red
            else if cInput is yellow
                wColor is yellow
            else if cInput is green
                wColor is green
            else if cInput is blue
                wColor is blue
            If they play a wild draw 4 and wd4Match is true
                chance is equal to rand() mod size of player's hand - 1
                If chance is one
                    rndCPU is equal to rand() mode numOpp + 1
                    call drawCard() twice
                    Print that rndCPU called player's bluff.
                Else
                    call placeCard()
            Else
                call placeCard()
        Else
            call placeCard()
    Else
        call placeCard()
```

Place Card

```
    Rotate deck
    Push card played to front of deck
    Erase that card from player's hand
    if playerNum is 0
        print that you played that card
    Else
        print that computer played that card
        If computer plays a wild or wild draw 4
            vector of pairs(int,string) count
            push back 0, red
            push back 0, yellow
            push back 0, green
            push back 0, blue
            for I from 0 to size of computer's hand
                if the color is red
```

```
                    increment count[0]
                if the color is yellow
                    increment count[1]
                if the color is green
                    increment count[2]
                if the color is blue
                    increment count[3]
                Sort count
                Set wColor is count[3]’s color
                print wColor
    if it’s a wild draw 4
        Set skip to true
        If reverse is false
            If plrNum is numOpp
                Print that you drew 4 cards
            else
                print that next computer drew 4 cards
            for I from 0 to 4
                if plyrNum is numOpp
                    call drawCard(0)
                else
                    call drawCard(plrNum+1)
        else
            if plrNum is 0
                print that previous computer drew 4 cards
            else if plrNum is 1
                print that you drew 4 cards
            else
                print that last computer drew 4 cards
            for I from 0 to 3
                if plrNum is 0
                    call drawCard(numOpp)
                else
                    call drawCard(plrNum - 1)
    else if skip was played
        set skip to true
    else if reverse was played
        if reverse is true
            set reverse to false
        else
            set reverse to true
    else if draw 2 was played
        set skip to true
        if reverse is false
            if plrNum is numOpp
                print that you drew two cards
                call drawCard() twice
            else
                print that computer drew two cards
                call drawCard() twice
        else
            if plrNum is 0
                print that last computer drew two cards
                call drawCard() twice
            else if plyNum is 1
                print that you drew two cards
                call drawCard() twice
            else
                print that previous computer drew two cards
                call drawCard() twice
```

Display Points

```
    if winner is 0
        print that player won
    else
        print that computer won
    for I from 0 to numOpp inclusive
        for j from 0 to size of hand
            If value of card is less than ten
                Add value to gamePts
            If card is skip, draw 2, or reverse
                Add 20 to gamePts
            else
                Add 50 to gamePts
    Display players points
    For I from 0 to numOpp
        Display computer points
```

Play Again

```
string input
do
    Ask user if they'd like to play again
    Get input from user
    If input is not y or n
        Print error
While input is not y or n
if input is y
    return true
else
    return false
```

Get gamePts

```
    return gamePts
```

Contents of main.cpp

```
 /*
 * File:   main.cpp
 * Author: Rachel Scherer
 * Created on October 21, 2018, 1:48 PM
 * Purpose: Uno Clone
 */

//System Libraries
#include <iostream>
#include <vector>

//User Libraries
#include "Game.h"

int main(int argc, char** argv)
{
    std::vector<Game> sess;
    int numSess = 0;
    do{
        //If this is the very first game, call the default constructor.
        if(numSess == 0)
        {
            sess.emplace_back();
        }
        //Otherwise, pass the points earned from the previous game as an argument to the next one.
        else
        {
            sess.emplace_back(sess[numSess-1].getGamePts());
        }
        std::cout << "Welcome to Uno!\n\n";
        std::string numOpp;

        //Get the number of opponents from the user, and validate using the getNumOpponents member
function.
        do
        {
            std::cout << "How many computers would you like to play with? ";
            std::cin >> numOpp;
        }while(!sess[numSess].setNumOpp(numOpp));

        //Loop through each round until the EOG flag is called.
        do
        {
            sess[numSess].getRound();
        }while(!sess[numSess].getEOG());

        //If the game is finished, display the points.
        sess[numSess].displayPoints();

        //If the user wants to play again, loop back to top.
    }while(sess[numSess++].playAgain());

    return 0;
}
```

Contents of Game.h

```
/*
 * File:   Game.h
 * Author: Rachel Scherer
 * Created on October 22, 2018, 8:54 PM
 * Purpose: Uno Clone
 */

#include <cstdlib>
#include <iostream>
#include <utility>
#include <string>
#include <algorithm>
#include <deque>
#include <ctime>
#include <map>
#include <time.h>
#include <vector>

#ifndef GAME_H
#define GAME_H

class Game
{
private:
    //Typedefs
    typedef std::pair<int, std::string> card;
    typedef std::multimap<int,int> mmInt;

    //Deques/pairs
    std::deque<card>* plrHnd;                       //Holds cards dealt to each player
    std::deque<card> deck;                          //Holds rest of deck

    //Vectors
    std::vector<int> gamePts;

    //Bool flags
    bool eog;                                       //End of game flag
    bool skip;                                      //"Skip" card flag
    bool reverse;                                   //"Reverse" card flag
    bool wd4Mtch;                                   //"Wild Draw 4" flag used for computers to call
bluff on player

    //Ints
    int numOpp;                                     //Number of opponents
    int numDrwn;                                    //Number of cards drawn (used to shuffle cards)
    int winner;                                     //Holds which player won game

    //Strings
    std::string wColor;                             //Holds color set by Wild/Wild Draw 4 cards

    //Member functions
    void defineDeck();                              //Initialize each card and shuffle them
    void dealDeck();                                //Deal the cards
    bool checkCardValidity(std::string);            //Validates user input when checking card
    bool getColor(std::string&);                    //Validates user input when choosing wild card
    void computerTurn(int);                         //Handle opponent computer logic
    std::multimap<int,int> getCardColorOrder(int);  //Helps computer decide which card to choose
    std::string displayCard(card);                  //Display card
```

```
    void playCard(int,int);                         //Get special inputs for certain cards, organizes
placeCard()
    void placeCard(int,int);                        //Adds card from player hand to deck
    void drawCard(int);                             //Draw Card

public:
    //Constuctors
    Game();                                         //Default Constructor
    Game(std::vector<int>);                         //Constructor using points earned from different
game as an argument

    //Destructors
    ~Game();                                        //Deletes dynamically allocated data

    //Mutators
    bool setNumOpp(std::string);                    //Sets numOpponents

    //Accessors
    bool getEOG();                                  //Returns end of game flag
    std::vector<int> getGamePts();                  //Returns game points

    //Helper functions
    bool playAgain();                               //Asks user if they want to play again
    void getRound();                                //Gets one complete round of uno
    void displayPoints();                           //Displays points earned in game
};

#endif /* GAME_H */
```

Contents of Game.cpp

```
/*
 * File:   Game.cpp
 * Author: Rachel Scherer
 * Created on October 22, 2018, 8:54 PM
 * Purpose: Uno Clone
 */

//User Libraries
#include "Game.h"

/**
 * Default constructor. Initializes variables to default values.
 */
Game::Game()
{
    //Seed random numbers
    std::srand(static_cast<unsigned int>(time(0)));

    //Define the deck of cards
    defineDeck();

    //Initialize variables and flags.
    wColor = "";
    eog = false;
    skip = false;
    reverse = false;
    gamePts = {0};
}

/**
```

```
 * Same as default constructor, but passes in previous game points earned as an argument
 * @param gamePts
 */
Game::Game(std::vector<int> gamePts)
{
    std::srand(std::time(0));
    defineDeck();
    wColor = "";
    eog = false;
    skip = false;
    reverse = false;
    this->gamePts = gamePts;
}

/**
 * Delete the dynamically allocated data
 */
Game::~Game()
{
    //Clear deck
    deck.clear();

    //Delete pointer array of deques
    for(int i=0;i<plrHnd->size();i++)
    {
        plrHnd[i].clear();
    }
    delete[] plrHnd;
    plrHnd = NULL;
}

/**
 * Return end of game flag
 */
bool Game::getEOG(){
    return eog;
}

/**
 * Initalizes each card then shuffles them
 */
void Game::defineDeck()
{
    //Define the cards using a deque of pairs
    deck.push_front(std::make_pair(0,"Red"));
    deck.push_front(std::make_pair(0,"Yellow"));
    deck.push_front(std::make_pair(0,"Green"));
    deck.push_front(std::make_pair(0,"Blue"));
    for(int i=1;i<=12;i++)
    {
        deck.push_front(std::make_pair(i,"Red"));
        deck.push_front(std::make_pair(i,"Red"));
        deck.push_front(std::make_pair(i,"Yellow"));
        deck.push_front(std::make_pair(i,"Yellow"));
        deck.push_front(std::make_pair(i,"Green"));
        deck.push_front(std::make_pair(i,"Green"));
        deck.push_front(std::make_pair(i,"Blue"));
        deck.push_front(std::make_pair(i,"Blue"));
    }
    for(int i=0;i<4;i++)
    {
        deck.push_front(std::make_pair(13,"Wild"));
```

```
        deck.push_front(std::make_pair(14,"WildD4"));
    }

    //Shuffle the cards
    std::random_shuffle(deck.begin()+1,deck.end());
}

/**
 * Deals the deck of cards amongst the players
 */
void Game::dealDeck()
{
    //Count makes sure we don't deal more cards than we have
    int count = 0;

    //Iterate through the deck
    for(std::deque<card>::const_iterator iter = deck.begin(); iter != deck.end();++iter,count++)
    {
        //Safety measure
        if(count==(numOpp+1)*7)
        {
            break;
        }

        //Push iterated value to plrHnd
        plrHnd[count%(numOpp+1)].push_front(std::make_pair(iter->first,iter->second));

        //Pop from deck
        deck.pop_front();
    }

    //Make sure the front value of deck isn't a "Wild" or "Wild Draw 4" by moving it to the back.
    while(deck[0].first == 13 || deck[0].first == 14)
    {
        std::rotate(deck.begin(),deck.end()-1,deck.end());
    }

    //Update cards drawn so we know if they need to be shuffled
    numDrwn = 7*(numOpp+1);

    //Since different matches can have different numbers of players, make sure that the gamePts
    //passed in from previous game match current number of players.
    for(int i=0;i<=numOpp;i++)
    {
        if(gamePts.size() <= i)
        {
            gamePts.push_back(0);
        }
    }
    while(gamePts.size() > numOpp+1)
    {
        gamePts.pop_back();
    }
}

/**
 * Validate user input for number of opponents and return true if successful.
 * @param numOpp
 */
bool Game::setNumOpp(std::string numOpp)
{
    if(!isdigit(numOpp[0]))
```

```
    {
        std::cout << "Error! Must enter a number.";
        return false;
    }
    int oppInt = stoi(numOpp);
    if(oppInt < 1 || oppInt > 14)
    {
        std::cout << "Error! Must enter a number between 1-14. ";
        return false;
    }
    this->numOpp = oppInt;
    plrHnd = new std::deque<card>[oppInt+1];
    dealDeck();
    return true;
}

/**
 * Goes through one complete round of uno.
 */
void Game::getRound()
{
    //Make sure turn wasn't skipped
    if(!skip)
    {
        //Display top of deck, which is the previous player's card was discarded
        std::cout << "\nCenter card: " << displayCard(deck[0]);

        //In the case of a wild card, display the color set by the previous player
        if(wColor != "")
        {
            std::cout << " (" << wColor << ")";
        }

        //Display your hand alongside a number to select the cards by.
        std::cout << "\n\nYour hand:\n";
        for(int i=0;i<plrHnd[0].size();i++)
        {
            std::cout << i+1 << ". " << displayCard(plrHnd[0][i]) << "\n";
        }

        //Include an option to draw your card
        std::cout << "(D). Draw Card\n";

        //Get user input and validate it.
        std::string input;
        do
        {
            std::cout << "\nWhich card would you like to play? (enter 1 - " << plrHnd[0].size() << " to
pick your card, D to draw, or H to hear the rules!) ";
            std::cin >> input;
        }while(!checkCardValidity(input));
    }
    else
    {
        //If your turn was skipped, display it.
        std::cout << "Your turn was skipped.";

        //Set flag to false so next computer's turn isn't skipped as well.
        skip = false;
    }

    //Keep track of turn
```

```
    int turn;

    //Set turn depending on which direction we're approaching loop from.
    if(reverse)
    {
        turn = numOpp;
    }
    else
    {
        turn = 1;
    }

    while(turn > 0 && turn <= numOpp)
    {
        //Make sure the winning card wasn't just played.
        if(eog)
        {
            return;
        }
        std::cout << "\n";

        //Get computer turn
        computerTurn(turn);

        //Depending on if reverse is set, choose whether to move "clockwise" or "counterclockwise" by
incrementing or decrementing
        turn = (reverse)? turn+1:turn-1;

    }
    std::cout << "\n";
}

/**
 * Return string representing card passed as parameter
 * @param currentCard
 */
std::string Game::displayCard(card currentCard)
{
    std::string name = "";
    switch(currentCard.first)
    {
        case 10:
            name = currentCard.second + " Skip"; break;
        case 11:
            name = currentCard.second + " Reverse"; break;
        case 12:
            name = currentCard.second + " Draw 2"; break;
        case 13:
            name = "Wild Card"; break;
        case 14:
            name = "Wild Draw 4"; break;
        default:
            name = currentCard.second + " " + std::to_string(currentCard.first);
    }
    return name;
}

/**
 * Validate user input for selecting Wild card color and return true if successful.
 * @param input
 */
bool Game::getColor(std::string& input)
```

```
{
    //Make input all lowercase
    std::transform(input.begin(), input.end(), input.begin(), ::tolower);

    //Compare input against acceptable values.
    if(input != "red" && input != "blue" && input != "green" && input != "yellow")
    {
        std::cout << "Error! Must enter a valid color. ";
        return false;
    }
    return true;
}

/**
 * Return true if user input is valid and the card selected can be played
 * @param input
 */
bool Game::checkCardValidity(std::string input)
{
    //If the user input starts with a d, draw a card.
    if(tolower(input[0]) == 'd')
    {
        std::cout << "\nYou drew a card.";
        drawCard(0);
        return true;
    }

    //If the user input starts with an h, read the "How to play" guide.
    if(tolower(input[0]) == 'h')
    {
        std::cout << "\n-----------HOW TO PLAY-----------";
        std::cout << "\n\nThe standard UNO deck has 108 cards consisting of: four colored 'suits' (Red,
Yellow, Green, Blue), Action cards (Draw 2, Reverse, and Skip), Wild cards, and Wild Draw 4 cards.";
        std::cout << "\n\nEach player starts with seven cards, and one card is placed in the center
which forms the discard pile.";
        std::cout << "\n\nEach player takes turns placing a card in the center. This card must be of
either the same color, number/wording of the card in the center, or a 'Wild' card which can be placed
at any time.";
        std::cout << "\n\nThe objective of the game is to get rid of all the cards in your hand.";
        std::cout << "\n\nThe 'Reverse' cards (of any color) mean that the direction of play is
reversed (i.e. clockwise to counter-clockwise).";
        std::cout << "\n\nThe Draw Two cards (of any color) means that the next player must draw two
cards from the draw pile and skip their turn.";
        std::cout << "\n\nThe cards marked Skip (in any color) means that the next player must skip
his/her turn. In a 2 player game, if someone lays a skip on their last turn (UNO) they must go again
and play another number";
        std::cout << "\n\nThe black cards (Wild) means that a player may choose any color (or stick
with a same color) that they wish.";
        std::cout << "\n\nThe black (Wild Draw Four) cards means that a player may choose any color,
but the next player must draw four cards and skip his/her turn.";
        std::cout << "\n\nIf you are unable to play a card, you may draw a card and skip your turn
instead.";
        std::cout << "\n\nAs soon as a player runs out of cards, the numbers are tallied up as follows
and given to the winner:";
        std::cout << "\n0-9:The card's numeric value.";
        std::cout << "\nCount Draw Two, Skip, and Reverse: 20 points.";
        std::cout << "\nWild, Wild Draw 4: 50 points.\n";
        return false;
    }

    //Makes sure user input is a digit otherwise.
    if(!isdigit(input[0]))
```

```
    {
        std::cout << "Error! You must select a valid number.\n";
        return false;
    }

    //If it's a digit, convert to int
    int iInt = stoi(input);

    //Make sure that the input is within the range of selectable cards
    if(iInt < 1 || iInt > plrHnd[0].size())
    {
        std::cout << "Error! you must select a valid number.\n";
        return false;
    }

    //Wild cards can be played at any time, so check for that
    if(plrHnd[0][iInt-1].first == 13)
    {
        std::cout << "\n";
        playCard(0,iInt-1);
        return true;
    }

    //Color match flag is true when the user plays a wild draw 4 and they are able to
    //play other cards. If this is the case, the computer has a chance to call your bluff
    wd4Mtch = false;

    //If the user selects a wild draw 4, check to see if there are other possible cards to play.
    if(plrHnd[0][iInt-1].first == 14)
    {
        for(int i=0;i<plrHnd[0].size();i++)
        {
            if(deck[0].second == plrHnd[0][i].second || plrHnd[0][i].first == 13 ||
               deck[0].first == plrHnd[0][i].first || deck[0].second == wColor)
            {
                //If that's the case, set wd4Mtch to true.
                wd4Mtch = true;
            }
        }
        if(wd4Mtch == true)
        {
            //Give a warning to make sure the user wants to bluff a Wild Draw 4
            std::string wInput;
            do
            {
                std::cout << "Warning: a computer may call your bluff if you use a Wild Draw 4 while
having other cards to play. Continue? (y/n) ";
                std::cin >> wInput;
                if(tolower(wInput[0]) != 'y' && tolower(wInput[0]) != 'n') std::cout << "Error! Invalid
input.\n";
            }while(tolower(wInput[0]) != 'y' && tolower(wInput[0]) != 'n');
            if(wInput == "n")
            {
                //Return back to card selection if the answer is no.
                return false;
            }
            else
            {
                //Otherwise, play the card.
                std::cout << "\n";
                playCard(0,iInt-1);
                return true;
```

```
            }
        }
        else
        {
            //If wd4Mtch is false, play the card as normal.
            std::cout << "\n";
            playCard(0,iInt-1);
            return true;
        }
    }

    //Return false if card selected doesn't match with the card in the center
    if(plrHnd[0][iInt-1].first != deck[0].first &&
       plrHnd[0][iInt-1].second != deck[0].second &&
       plrHnd[0][iInt-1].second != wColor)
    {
        std::cout << "Card must be either the same color or number as the center card\n";
        return false;
    }
    //Otherwise, play the card ad normal.
    else
    {
        playCard(0,iInt-1);
        return true;
    }
}

/**
 * Handle computer decision-making logic
 * @param cpu
 */
void Game::computerTurn(int cpu)
{
    //Make sure computer's turn isn't skipped.
    if(skip)
    {
        skip = false;
        std::cout << "Computer " << cpu << "'s turn was skipped.";
        return;
    }

    //Obtain the order of colors to prioritize. For example,
    //If there are more reds in the computer's hand, look for reds to swap to.
    std::multimap<int,int> srtCnt = getCardColorOrder(cpu);
    std::map<int,int>::reverse_iterator itr;

    //Make sure there are cards to swap to.
    if(srtCnt.size() > 0)
    {
        //Reverse iterate through the cards sorted by color to prioritize searching for that specific
color
        for(itr=srtCnt.rbegin();itr!=srtCnt.rend();++itr)
        {
            //If the color is red, execute code below
            if(itr->second == 0)
            {
                for(int i=0;i<plrHnd[cpu].size();i++)
                {
                    if(plrHnd[cpu][i].second == "Red" && plrHnd[cpu][i].first == deck[0].first)
                    {
                        //If it's a valid card, swap.
                        playCard(cpu,i);
```

```
                    return;
                }
            }
        }
        //If the color is yellow, execute code below
        if(itr->second == 1)
        {
            for(int i=0;i<plrHnd[cpu].size();i++)
            {
                if(plrHnd[cpu][i].second == "Yellow" && plrHnd[cpu][i].first == deck[0].first)
                {
                    playCard(cpu,i);
                    return;
                }
            }
        }
        //If the color is green, execute code below
        if(itr->second == 2)
        {
            for(int i=0;i<plrHnd[cpu].size();i++)
            {
                if(plrHnd[cpu][i].second == "Green" && plrHnd[cpu][i].first == deck[0].first)
                {
                    playCard(cpu,i);
                    return;
                }
            }
        }
        //If the color is blue, execute code below
        if(itr->second == 3)
        {
            for(int i=0;i<plrHnd[cpu].size();i++)
            {
                if(plrHnd[cpu][i].second == "Blue" && plrHnd[cpu][i].first == deck[0].first)
                {
                    playCard(cpu,i);
                    return;
                }
            }
        }
    }
    //If there are no cards to prioritize swapping to, look for a color that matches the color of the
card in the center.
    for(int i=0;i<plrHnd[cpu].size();i++)
    {
        if(plrHnd[cpu][i].second == deck[0].second || plrHnd[cpu][i].second == wColor)
        {
            playCard(cpu,i);
            return;
        }
    }
    //If there are none of the those either, look for Wild/Wild Draw 4 cards to play.
    for(int i=0;i<plrHnd[cpu].size();i++)
    {
        if(plrHnd[cpu][i].first == 13 || plrHnd[cpu][i].first == 14)
        {
            playCard(cpu,i);
            return;
        }
    }
```

```
    //If there's truly nothing to play, draw a card.
    std::cout << "Computer " << cpu << " drew a card.";
    drawCard(cpu);
}

/**
 * Help computer determine which cards to prioritize switching to.
 * @param cpu
 */
std::multimap<int,int> Game::getCardColorOrder(int cpu)
{
    //Represents the computer's cards in a map. The first value will be
    //a number representing the card color, and the second one will be the number
    //of cards that belong to that color in the computer's hand.
    std::map<int,int> count;

    //Initialize the counts to 0
    for(int i=0;i<4;i++)
    {
        count.insert(std::pair<int,int> (i,0));
    }

    //Increment the values at certain keys depending on the color of the card.
    for(int i=0;i<plrHnd[cpu].size();i++)
    {
        if(plrHnd[cpu][i].second == "Red")
        {
            count[0]++;
        }
        else if(plrHnd[cpu][i].second == "Yellow")
        {
            count[1]++;
        }
        else if(plrHnd[cpu][i].second == "Green")
        {
            count[2]++;
        }
        else if(plrHnd[cpu][i].second == "Blue")
        {
            count[3]++;
        }
    }

    //Compare the map values with the number of cards that match the color
    //of the center card, then delete those values if they are less than the number of cards.
    //This is to avoid swapping to a different color when it's advantageous to stay on the current one.
    for(int i=0;i<4;i++)
    {
        if((deck[0].second == "Red" && count[i] < count[0])     ||
           (deck[0].second == "Yellow" && count[i] < count[1])  ||
           (deck[0].second == "Green" && count[i] < count[2])   ||
           (deck[0].second == "Blue" && count[i] < count[3]))
        {
            count.erase(i);
        }
    }

    //Swap the first and second values of the map into a different multimap. This sorts each card by
number of
    //iterations in the first value, and the value representing the color of the card in the second one.
    std::multimap<int,int> srtCnt;
    std::map<int,int>::iterator itr;
```

```
    for(itr=count.begin();itr!=count.end();++itr)
    {
        srtCnt.insert(mmInt::value_type(itr->second,itr->first));
    }

    //Return the sorted multimap.
    return srtCnt;
}

/**
 * Get further validation after card is determined to be playable
 * @param plrNum
 * @param cardNum
 */
void Game::playCard(int plrNum, int cardNum)
{
    //If the player is about to play the last card, set the end of game flag to truel
    if(plrHnd[plrNum].size() == 1)
    {
        placeCard(plrNum,cardNum);
        winner = plrNum;
        eog = true;
        return;
    }

    //Initialize values
    int chance = 0;
    int rndCPU = 0;
    wColor = "";

    //If the user is playing the card (not a computer)...
    if(plrNum == 0)
    {
        //And they play a Wild or a Wild Draw 4...
        if(plrHnd[0][cardNum].first == 13 || plrHnd[0][cardNum] .first == 14)
        {
            //Prompt the user for the color they'd like it changed to. Validate the input.
            std::string cInput;
            do
            {
                std::cout << "What color would you like it changed to? ";
                std::cin >> cInput;
            }while(!getColor(cInput));

            //Set wColor depending on input
            if(cInput == "red")
            {
                wColor = "Red";
            }
            else if(cInput == "yellow")
            {
                wColor = "Yellow";
            }
            else if(cInput == "green")
            {
                wColor = "Green";
            }
            else if(cInput == "blue")
            {
                wColor = "Blue";
            }
```

```
            //If the user is bluffing on a Wild Draw 4...
            if(plrHnd[0][cardNum].first == 14 && wd4Mtch == true)
            {
                //The user's bluff will be called more frequently if they're carrying more cards.
                chance = rand()%(plrHnd[0].size()-1) + 1;

                //If the user's bluff is called, draw 2 cards.
                if(chance == 1)
                {
                    rndCPU = rand()%(numOpp+1)+1;
                    drawCard(plrNum);
                    drawCard(plrNum);
                    std::cout << "Computer " << rndCPU << " called your bluff! You drew two cards.";
                }
                //Else, play cards as normal.
                else
                {
                    placeCard(0,cardNum);
                }
            }
            else
            {
                placeCard(0,cardNum);
            }
        }
        else
        {
            placeCard(0,cardNum);
        }
    }
    else
    //Last function call accounts for if the computer places a card.
    {
        placeCard(plrNum,cardNum);
    }
}

/**
 * Draws a card from the deck into the player's hand.
 * @param plrNum
 */
void Game::drawCard(int plrNum)
{
    //Push into player's hand
    plrHnd[plrNum].push_back(deck.back());

    //Pop from deck
    deck.pop_back();

    //Shuffle cards if all possible cards were drawn.
    numDrwn++;
    if(numDrwn == (deck.size()-(numOpp+1)*7))
    {
        std::random_shuffle(deck.begin()+1,deck.end());
    }
}

/**
 * Adds card to deck from player hand.
 * @param plrNum
 * @param cardNum
 */
```

```
void Game::placeCard(int plrNum, int cardNum)
{
    //Rotates the front of the deck (the center card) to the back.
    std::rotate(deck.begin(),deck.end()-1,deck.end());

    //Pushes the card played to the front.
    deck.push_front(plrHnd[plrNum][cardNum]);

    //Erases that card from hand.
    plrHnd[plrNum].erase(plrHnd[plrNum].begin()+cardNum);

    //Display which card was played
    if(plrNum == 0)
    {
        std::cout << "\nYou played a " << displayCard(deck[0]);
    }
    else
    {
        std::cout << "Computer " << plrNum << " played a " << displayCard(deck[0]);

        //If the computer plays a Wild or Wild Draw 4, determine which color they choose.
        if(deck[0].first == 13 || deck[0].first == 14)
        {

            //We'll do it through a vector of pairs this time.
            std::vector<std::pair<int,std::string>> count;

            //Initialize values.
            count.push_back(std::make_pair(0, "Red"));
            count.push_back(std::make_pair(0, "Yellow"));
            count.push_back(std::make_pair(0, "Green"));
            count.push_back(std::make_pair(0, "Blue"));

            //Increment depending on cards present in hand.
            for(int i=0;i<plrHnd[plrNum].size();i++)
            {
                if(plrHnd[plrNum][i].second == "Red")
                {
                    count[0].first++;
                }
                if(plrHnd[plrNum][i].second == "Yellow")
                {
                    count[1].first++;
                }
                if(plrHnd[plrNum][i].second == "Green")
                {
                    count[2].first++;
                }
                if(plrHnd[plrNum][i].second == "Blue")
                {
                    count[3].first++;
                }
            }

            //Sort the vector
            std::sort(count.begin(),count.end());

            //Set wColor to the last count (the one now with the greatest value).
            wColor = count[3].second;

            //Display it.
            std::cout << " (" << wColor << ")";
```

```
    }
}

//If it's a Wild Draw 4...
if(deck[0].first == 14)
{
    //Next player's turn is skipped.
    skip = true;

    //Determine who is next by checking for the reverse flag
    if(!reverse)
    {
        //If the last computer played the card and reverse is not set, the player draws the cards.
        if(plrNum == numOpp)
        {
            std::cout << "\nYou drew 4 cards.";
        }
        //Otherwise, the next computer draws the cards.
        else
        {
            std::cout << "\nComputer " << plrNum+1 << " drew 4 cards.";
        }
        for(int i=0;i<4;i++)
        {
            if(plrNum == numOpp)
            {
                drawCard(0);
            }
            else
            {
                drawCard(plrNum + 1);
            }
        }
    }
    //If reverse it set...
    else
    {
        //If the player play the card, the last computer will draw.
        if(plrNum == 0)
        {
            std::cout << "\nComputer " << numOpp << " drew 4 cards.";
        }
        //If the next computer plays the card, the player draws.
        else if(plrNum == 1)
        {
            std::cout << "\nYou drew 4 cards.";
        }
        //If another computer plays the card, the previous computer draws.
        else
        {
            std::cout << "\nComputer " << plrNum-1 << " drew 4 cards.";
        }
        for(int i=0;i<4;i++)
        {
            if(plrNum == 0)
            {
                drawCard(numOpp);
            }
            else
            {
                drawCard(plrNum - 1);
            }
```

```
        }
    }
}
//SKIP CARD - Set skip flag to true.
else if(deck[0].first == 10)
{
    skip = true;
}
//REVERSE CARD - Set reverse flag to true if currently false and vice versa.
else if(deck[0].first == 11)
{
    if(reverse)
    {
        reverse = false;
    }
    else
    {
        reverse = true;
    }
}
//DRAW 2 CARD - Set skip flag to true and call drawCard twice.
else if(deck[0].first == 12)
{
    skip = true;

    //Display who the recipient of the draw 2 is depending on who called it, and whether the
    //reverse flag is set to true or false.
    if(!reverse)
    {
        if(plrNum == numOpp)
        {
            std::cout << "\nYou drew 2 cards.";
            drawCard(0);
            drawCard(0);
        }
        else
        {
            std::cout << "\nComputer " << plrNum+1 << " drew 2 cards.";
            drawCard(plrNum+1);
            drawCard(plrNum+1);
        }
    }
    else
    {
        if(plrNum == 0)
        {
            std::cout << "\nComputer " << numOpp << " drew 2 cards";
            drawCard(numOpp);
            drawCard(numOpp);
        }
        else if (plrNum == 1)
        {
            std::cout << "\nYou drew 2 cards.";
            drawCard(0);
            drawCard(0);
        }
        else
        {
            std::cout << "\nComputer " << plrNum-1 << " drew 2 cards.";
            drawCard(plrNum-1);
            drawCard(plrNum-1);
        }
```

```
        }
    }
}

/**
 * Tallys up the points. This carries over from game to game.
 */
void Game::displayPoints()
{
    //Congratulate the winner
    if(winner == 0)
    {
        std::cout << "\nCongrats! You won :)";
    }
    else
    {
        std::cout << "\nComputer " << winner << " won!";
    }

    //Tally up total points based on card values carried by all players and give them to the winner.
    std::cout << "\n\nTOTAL POINTS\n";

    //Iterate through each player.
    for(int i=0;i<=numOpp;i++)
    {
        //Iterate through each card.
        for(int j=0;j<plrHnd[i].size();j++)
        {
            //Cards 0-9: Add card's numeric value.
            if(plrHnd[i][j].first < 10)
            {
                gamePts[winner] += plrHnd[i][j].first;
            }
            //Cards Reverse, Skip, and Draw 2: Add 20 points.
            else if(plrHnd[i][j].first >= 10 && plrHnd[i][j].first < 13)
            {
                gamePts[winner] += 20;
            }
            //Cards Wild and Wild Draw 4: Add 50 points.
            else
            {
                gamePts[winner] += 50;
            }
        }
    }

    //Display totals.
    std::cout << "Player 1: " << gamePts[0] << " points.\n";
    for(int i=1;i<=numOpp;i++)
    {
        std::cout << "Computer " << i << ": " << gamePts[i] << " points\n";
    }
}

/**
 * Prompt user if they want to play again and validates their input. Returns true if valid.
 */
bool Game::playAgain()
{
    //Loop until correct input is pressed (either a y or n).
    std::string input;
    do{
```

```
        std::cout << "\nWould you like to play again? (y/n) ";
        std::cin >> input;
        if(tolower(input[0])!= 'n' && tolower(input[0] != 'y'))
        {
            std::cout << "Error! Invalid input";
        }
    }while(tolower(input[0])!= 'n' && tolower(input[0]) != 'y');
    std::cout << "\n";

    //If y, play a new game. If n, exit.
    if(tolower(input[0]) == 'y')
    {
        return true;
    }
    else
    {
        return false;
    }
}

/**
 * Returns gamePts. Used to keep track of points from game to game.
 */
std::vector<int> Game::getGamePts()
{
    return gamePts;
}
```