Project 1

## <UNO>

**CIS 17C**

**Author: Adam Zavala**

**Due: 5/18/2025**

### Introduction:

Title: UNO

The game I have chosen for my project is UNO. This game was chosen because this game involves various types of card management such as decks, discard piles, and player hands which made stl containers a perfect fit alongside using stl algorithms like find() to search for matching cards. To keep things relatively simple, this version of UNO is the original version and thus has no new cards or house rules, like stacking draw 2s.

### Approach to Development:

Before starting, I wanted to make sure I was able to utilize the required amount of stl containers and algorithms. I started by making a list of everything needed for an UNO game, which includes the deck, players, their respective hands, a discard pile, and a way to check if their cards match. I decided to utilize the stl list container for the deck, stl stack container for the discard pile, stl map container for players, stl queue to handle turn orders, and stl find() for card comparisons.

The first two versions involved creating and shuffling the UNO deck which has both normal and special cards. The normal cards are assigned a color and number while special cards are assigned color and name like “Reverse”. Wild cards have no color until a player plays that card and declares a color of their choice. Version three then implements a game loop, player turns, special card conditions, and win conditions.

### Game Rules

This version of UNO follows the original game rules where there are 76 normal cards, 8 skip cards, 8 draw cards, 8 reverse cards, 4 wild cards, and 4 wild draw 4 cards. The goal of this game is to reduce your hand size to zero. If you get down to only one card in your hand, you shout “UNO”. When the game starts, everyone draws 7 cards from the deck and then the top card of the deck is put to the discard pile for players to start matching. For example, if a red 8 was drawn, the first player can try to match with either a red card or a card with the number 8 on it, with exceptions to wild cards which can be played on any card currently at the top of the discard pile. On your turn you have a choice between playing a card to match or drawing. You can only draw once and if you don’t have any playable cards, you simply end your turn.

Special cards like skips, or draw 2 have special conditions if played. Skip cards forces the next player to skip their turn, Draw 2 makes the next player draw 2 cards and skip their turns, reverse cards reverse the turn order, wild cards can be played on any card and allow the player to choose the color they want, and wild draw 4 cards make the next player draw 4 cards, skip their turn, and the player who played the wild draw 4 also chooses their color.

### Description of Code

This c++ project utilizes object oriented programming and the standard template library for containers, algorithms, and iterators. There is a card class which handles individual card information, a deck class which creates a list of cards from the Card class and shuffles them, and the Game class which handles the game loop and all the gameplay events. The main.cpp file sets the game to start and the amount of cpu players wanted.

Checkoff Sheet1.Container classes (Where in code did you put each of these Concepts and how were they used?

1.Sequences (At least 1)

**1.list (Deck.h line 10, used to handle deck of UNO cards)**

2.slist

3.bit\_vector

2.Associative Containers (At least 2)

1.set

**2.map (Deck.h line 11, used to contain players)**

3.hash

3.Container adaptors (At least 2)

**1.stack (Game.cpp line 51, stores all discarded cards and displays top card)**

**2.queue (Game.cpp line 54, keeps track of player turn order)**

3.priority\_queue

2.Iterators

1.Concepts (Describe the iterators utilized for each Container)

1.Trivial Iterator

2.Input Iterator

3.Output Iterator

4.Forward Iterator

**5.Bidirectional Iterator (Game.cpp line 41, used to traverse player hand to display it for gameplay)**

6.Random Access Iterator

3.Algorithms (Choose at least 1 from each category)

1.Non-mutating algorithms

1.for\_each

**2.find (Game.cpp line 352, used to compare card with specific names like “Skip”)**

3.count

4.equal

5.search

2.Mutating algorithms

1.copy

2.Swap

3.Transform

4.Replace

5.fill

6.Remove

7.Random\_Shuffle

3.Organization

1.Sort

2.Binary search

3.merge

4.inplace\_merge

5.Minimum and maximum

### Code:

#### Main.cpp

/\*

Author: Adam Zavala

Purpose: Uno game for Project 1 of CS 17C Project that utilizes c++ stl.

Version 1: Sets up cards and deck which utilizes lists from stl. Will turn to its own deck and card class in next version.

Version 2: Card and Deck class created.

Players are created as structures which maps will be utilized to hold all player objects and assign them a unique ID.

Draw function added.

Created stack container for discard pile.

Using Queue container for player turn orders.

Version 3:

Game Class created.

\*/

//System Libraries

#include <iostream>

using namespace std;

//User Libraries

//#include "Game.h"

#include "Game.h"

//Function Prototypes

int getPlayerCount();

int main(){

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

int playerCount = getPlayerCount();

Game game(playerCount);

game.dealHand();

game.gameLoop();

cout << "Thanks for playing!" << endl;

return 0;

}

//Prompts user for amount of players in game and returns that amount

int getPlayerCount(){

int n;

cout << "Please enter total amount of UNO players (Max = 6): ";

cin >> n;

if(n<2)n=2;

if(n>6)n=6;

//TODO: User validation

return n;

}

#### Card.h

#include <string>

using namespace std;

#ifndef CARD\_H

#define CARD\_H

class Card{

private:

int value;

string color, name;

bool isSpecial = false;

public:

Card();

Card(string, int);

Card(string, int, bool);

Card(string, string, int, bool);

string getName(){return name;}

void setColor(string);

string getColor(){return color;}

int getValue(){return value;}

bool special(){return isSpecial;}

};

#endif // CARD\_H

#### Card.cpp

#include "Card.h"

#include <iostream>

#include <string>

#include <bits/stdc++.h>

using namespace std;

Card::Card(){

}

Card::Card(string color, int value){

this->color = color;

this->value = value;

name = color + " " + to\_string(value);

}

Card::Card(string name, int value, bool isSpecial){

this->name = name;

this->value = value;

this->isSpecial = isSpecial;

color = "Black";

}

Card::Card(string name, string color, int value, bool isSpecial){

this->color = color;

this->name = color + " " + name;

this->value = value;

this->isSpecial = isSpecial;

}

void Card::setColor(string c){

color = c;

}

#### Deck.h

#include "Card.h"

#include <list>

#include <string>

using namespace std;

class Deck{

private:

list<Card> deck;

int drawCount; //to help keep count of when to shuffle

void setDeck();

void setSpecials();

Card setCard(string, int);

Card setCard(string, int, bool);

Card setCard(string, string, int, bool);

public:

Deck();

void printDeck();

void printCard(Card);

void shuffleDeck();

void pop\_back(){deck.pop\_back();}

void push(Card card){deck.push\_back(card);}

Card getTopCard(){return deck.back();}

int getSize(){return deck.size();}

};

#### Deck.cpp

#include "Deck.h"

#include <iostream>

#include <ctime>

using namespace std;

Deck::Deck(){

setDeck();

setSpecials();

shuffleDeck();

}

void Deck::setDeck(){

//19 each color for normal cards with numbers on it

//each get 1 set from 0-9, and another from 1-9, 19 total

int n = 40;

for(int z=0;z<2;z++){

for(int i=0;i<n-(z\*4);i++){

int mod = 10;

if(z==1){

mod = 9;

}

int value = i % mod;

string color;

int c = i / mod;

switch(c){

case 1:

color = "Red";

break;

case 2:

color = "Blue";

break;

case 3:

color = "Yellow";

break;

case 0:

color = "Green";

break;

}

if(z==1){

value ++;

}

deck.push\_back(setCard(color, value));

}

}

}

void Deck::setSpecials(){

string color, name;

//loop 4 times for the 4 special types of cards: skip, reverse, wild, & wild +4

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

//8 each of special types, except for wild which only has 4 for base wild and 4 for wild +4

for(int z=0;z<8;z++){

if(i==3){

if(z%2==0){

name = "Wild";

}

else{

name = "Wild Draw +4";

}

deck.push\_back(setCard(name, z%2, true));

}

else{

switch(i){

case 0:

name = "Skip";

break;

case 1:

name = "Reverse";

break;

case 2:

name = "Draw +2";

break;

}

int c = z % 4;

switch(c){

case 1:

color = "Red";

break;

case 2:

color = "Blue";

break;

case 3:

color = "Yellow";

break;

case 0:

color = "Green";

break;

}

deck.push\_back(setCard(name, color, i, true));

}

}

}

}

//mash shuffle variation that moves front and back card into rand position inside deck

void Deck::shuffleDeck(){

//insert new card into deck, for shuffling later

srand(time(0));

list<Card>::iterator it = deck.begin();

Card temp;

int n, pos;

n = 400;

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

//new card position based on ran num generator

pos = rand() % 40;

advance(it, pos);

//rotates from moving back or front card based on loop iteration being even or odd

if(i%2==0){

temp = deck.back();

deck.emplace(it, temp);

deck.pop\_back();

}

else{

temp = deck.front();

deck.emplace(it, temp);

deck.pop\_front();

}

}

}

//useful function for debug purposes

void Deck::printDeck(){

for(auto i = deck.begin();i!=deck.end();i++){

//printCard(\*i);

string n = i->getName();

cout << n << endl;

}

}

Card Deck::setCard(string color, int value){

Card temp(color, value);

return temp;

}

Card Deck::setCard(string name, int value, bool isSpecial){

Card temp(name, value, isSpecial);

return temp;

}

Card Deck::setCard(string name, string color, int value, bool isSpecial){

Card temp(name, color, value, isSpecial);

return temp;

}

#### Game.h

#include "Deck.h"

#include "Player.h"

#include <map>

#include <stack>

using namespace std;

class Game{

private:

Deck deck;

map<int, Player> players;

int playerCount;

void drawCards(Player&, int);

void printHand(Player);

bool playerTurn(Player&, stack<Card>&);

bool npcTurn(Player&, stack<Card>&, int);

void setWildColor(Card &, string);

void setWildColor(Card &);

void playCard(Player &, Card);

int getNextPlayer(int, int, bool);

bool checkMatch(Card, Card);

bool checkWild(Card);

bool checkSkip(Card);

bool checkReverse(Card);

int checkDraw(Card);

bool checkWin(Player, int);

list<Card>::iterator findPlayableCard(Player&, Card);

void playerConfirmation();

void reshuffle(stack<Card>&);

public:

Game(int);

void dealHand();

void gameLoop();

};

#### Game.cpp

#include "Game.h"

#include <iostream>

#include <cstring>

#include <ctime>

#include <queue>

#include <stack>

#include <algorithm>

#include <bits/stdc++.h>

using namespace std;

Game::Game(int n){

playerCount = n;

for(int i=0;i<playerCount-1;i++){

players[i] = Player();

}

}

//Only to be called at very start of new Uno game

//Each player recieves 7 cards per Uno rules

void Game::dealHand(){

int n = playerCount;

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

drawCards(players[i], 7);

}

}

//function that handles drawing cards. can change based on game.

void Game::drawCards(Player &p, int drawCount){

int n = drawCount;

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

p.hand.push\_back(deck.getTopCard());

deck.pop\_back();

p.handSize ++;

}

if(deck.getSize() < 20){

cout << "You guys decide it's time to reshuffle the deck!" << endl;

}

}

//Prints player hand when called

void Game::printHand(Player p){

int count = 1;

cout << "\nPlayer Hand (" << p.handSize << " cards):\n\n";

for(auto i=p.hand.begin();i!=p.hand.end();i++){

//calls getName function from card class which returns the card name as string

cout << count << ": " << i->getName() << endl;

count ++;

}

cout << endl;

}

//Handles player turn orders and card actions that have been played

void Game::gameLoop(){

//Discard pile utilizes stack container because we really only need to see the "top" card of this stack

stack<Card> discardPile;

//Queue's first in, first out behaviour makes it easy to handle turn orders

//Will start off in clockwise rotation, and then reverse order if a reverse card is played

queue<int> turnOrder;

int n = playerCount;

int winner;

int count = 0; //used only for simulating turns up to a certain count

//game flags

bool win = false;

bool quit = false;

bool reverse = false;

bool skip;

bool played; //keeps track if a player was able to play a card or not

//before the game loop officially starts, we take the top card of the deck and put it into discard to start the game

discardPile.push(deck.getTopCard());

deck.pop\_back();

cout << "First card is: " << discardPile.top().getName() << endl;

playerConfirmation();

if(checkWild(discardPile.top())==true){

setWildColor(discardPile.top());

}

//first player will eventually be decided at random

//player 0 is us

int next = 0;

turnOrder.push(0);

int mod = 1;

//start the loop, and only end it when a winner has been declared

while(win==false&&quit==false){

//start by resetting states from previous loops

skip = false;

mod = 1;

//check if deck needs to be reshuffled

if(deck.getSize()<25){

cout << "The deck seems to be getting low, time for a reshuffle!" << endl;

reshuffle(discardPile);

playerConfirmation();

}

//determinds if current turn belongs to you or npc player

if(turnOrder.front()==0){

played = playerTurn(players[next], discardPile);

}

else{

played = npcTurn(players[next], discardPile, turnOrder.front());

playerConfirmation();

}

win = checkWin(players[next], turnOrder.front());

if(win==true){

break;

}

turnOrder.pop();

//check flags if card was played

if(played == true){

skip = checkSkip(discardPile.top());

if(checkReverse(discardPile.top())==true){

//if reverse was played, go the other way

cout << "Reverse played! Turn order is reversed!\n";

reverse = !reverse;

}

if(checkDraw(discardPile.top())>0){

int temp = (next + 1)%n;

temp = getNextPlayer(next, mod, reverse);

drawCards(players[temp], checkDraw(discardPile.top()));

}

playerConfirmation();

}

//this loop section implements special card actions like reverses/skips/wilds if applicable

//if a skip is played, mod = 2, meaning, that we jump to the second guy who is next

if(skip == true){

mod = 2;

}

next = getNextPlayer(next, mod, reverse);

turnOrder.push(next);

}

if(next == 0){

cout << "You win!" << endl;

}

else{

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

}

}

bool Game::checkMatch(Card playerCard, Card card){

//return true if player's card is wild, which can be played on anything

if(playerCard.getName()=="Wild"||playerCard.getName()=="Wild Draw +4"){

return true;

}

if(card.special()==false){

if(playerCard.special()==true){

return playerCard.getColor()==card.getColor();

}

else{

if(playerCard.getColor()==card.getColor()||playerCard.getValue()==card.getValue()){

return true;

}

else{

return false;

}

}

}

else{

if(playerCard.getColor()==card.getColor()||playerCard.getValue()==card.getValue()){

return true;

}

return false;

}

}

//handles player turn, returns true if they were able to play a valid card

//returns false if they were only able to draw

bool Game::playerTurn(Player &p, stack<Card> &discardPile){

int input, cInd;

list<Card> playable;

Card card = discardPile.top();

cout << "\nYour turn: " << endl;

cout << "Card to Match: " << endl << card.getName() << endl;

printHand(p);

do{

//look for playable cards in hand and if none are available, automatically draw a card

auto it = findPlayableCard(p, card);

if(it==p.hand.end()){

cout << "No playable cards...\n You are forced to draw a card" << endl;

playerConfirmation();

drawCards(p, 1);

cout << "You drew a " << p.hand.back().getName() << "!";

playerConfirmation();

//checks again if you have a playable card, if not, your turn is over

it = findPlayableCard(p, card);

if(it==p.hand.end()){

cout << "No playable cards drawn... You end your turn." << endl;

return false;

}

input = 2; //sets input to 2 to account for that player already drew a card at this point

//If there are playable cards, we exit from loop to decide if we want to play a card or skip

}

else{

cout << "Enter 1 to play a card from hand" << endl;

cout << "Enter 2 to draw a card from the deck" << endl;

cin >> input;

if(input==2){

drawCards(p, 1);

}

}

}while(!(input == 1 || input == 2));

cout << "\nYour turn: " << endl;

cout << "Card to Match: " << endl << card.getName() << endl;

printHand(p);

if(input == 2){

cout << "Enter 1 to play a card from hand" << endl;

cout << "Enter 2 to skip turn" << endl;

cin >> input;

}

if(input == 1){

cout << "Enter the corresponding number next to your card to play it" << endl;

cin >> cInd;

while(checkMatch(\*next(p.hand.begin(), cInd-1), card)==false){

cout << "Unable to play that" << endl;

cout << "Enter the corresponding number next to your card to play it" << endl;

cin >> cInd;

}

cout << "You played: " << next(p.hand.begin(), cInd-1)->getName() << endl << endl;

discardPile.push(\*next(p.hand.begin(), cInd-1));

auto it = p.hand.begin();

advance(it, cInd-1);

p.hand.erase(it);

p.handSize--;

//wildcard actions

if(checkWild(discardPile.top())==true){

int c;

do{

cout << "Select the color for the wild card: " << endl << endl;

cout << "Enter 1 for Red\nEnter 2 for Blue\n";

cout << "Enter 3 for Yellow\nEnter 4 for Green" << endl;

cin >> c;

}while(c<1||c>4);

switch(c){

case(1):

setWildColor(discardPile.top(), "Red");

break;

case(2):

setWildColor(discardPile.top(), "Blue");

break;

case(3):

setWildColor(discardPile.top(), "Yellow");

break;

case(4):

setWildColor(discardPile.top(), "Green");

break;

}

}

return true;

}

else{

return false;

}

return true;

}

bool Game::npcTurn(Player &p, stack<Card> &discardPile, int playerID){

Card card = discardPile.top();

cout << "\nOpponent " << playerID << "'s turn" << endl;

cout << "Cards in their hands: " << p.handSize << endl;

cout << "Card to Match: " << endl << card.getName() << endl;

if(checkWild(card)==true){

cout << "Wild card color set to: " << card.getColor() << endl;

}

auto it = findPlayableCard(p, card);

//forces npc to draw if no cards are playable

if(it==p.hand.end()){

cout << "No playable cards...\nOpponent " << playerID << " is forced to draw a card" << endl;

drawCards(p, 1);

//checks again if you have a playable card, if not, your turn is over

it = findPlayableCard(p, card);

if(it==p.hand.end()){

cout << "No playable cards drawn...\nOpponent " << playerID << " ends their turn." << endl;

//end = true;

return false;

}

else{

cout << it->getName() << " is played!" << endl;

discardPile.push(\*it);

p.hand.erase(it);

p.handSize--;

}

}

//actions if npc does have something playable

else{

cout << it->getName() << " is played!" << endl;

discardPile.push(\*it);

p.hand.erase(it);

p.handSize--;

}

if(checkWild(discardPile.top())==true){

setWildColor(discardPile.top());

}

return true;

}

int Game::getNextPlayer(int next, int mod, bool reverse){

int n = playerCount;

if(reverse == false){

next +=mod;

next %= n;

}

else{

if(next == 0){

next = n-mod;

}

else{

if(mod==2&&next==1){

next = n-1;

}

next -=mod;

next %= n;

}

}

return next;

}

list<Card>::iterator Game::findPlayableCard(Player &p, Card card){

//look for playable cards in hand with iterators and find\_if()

auto it = find\_if(p.hand.begin(), p.hand.end(), [&](Card c){

//checks if player needs to match w/ a nonspecial card, ex a Blue 8, or Red 3

if(card.special() == false){

//if current iteration of player hand is non special, compare color and number vals

if(c.special()==false){

return c.getColor() == card.getColor() || c.getValue()==card.getValue();

}

//else just check if its the same color or a wild card

else{

return c.getColor()==card.getColor()||c.getName()=="Wild"||c.getName()=="Wild Draw +4";

}

}

//if the card we are trying to match to is a wild, we simply try to match its chosen color

else if(checkWild(card)){

return c.getColor()==card.getColor()||checkWild(c);

}

//if it is a skip, reverse, or draw 2, match with these conditions

else{

if(c.special()==false){

return c.getColor()==card.getColor();

}

else{

return c.getColor() == card.getColor() || c.getValue()==card.getValue()||checkWild(c);

}

}

});

return it;

}

bool Game::checkWild(Card card){

if(card.getName()=="Wild"||card.getName()=="Wild Draw +4"){

return true;

}

return false;

}

bool Game::checkSkip(Card card){

if (card.getName().find("Skip") != string::npos || card.getName().find("+") != string::npos) {

cout << card.getName() << " causes next player to skip their turn!" << endl;

return true;

}

return false;

}

bool Game::checkReverse(Card card){

if(card.getName().find("Reverse") != string::npos){

return true;

}

return false;

}

int Game::checkDraw(Card card){

if(card.getName().find("Draw +2") != string::npos){

return 2;

}

else if(card.getName().find("Draw +4") != string::npos){

return 4;

}

return 0;

}

bool Game::checkWin(Player p, int playerID){

if(p.handSize==1){

if(playerID==0){

cout << "You shout UNO!" << endl;

}

else{

cout << "Opponent " << playerID << " shouts UNO! Uh oh!" << endl;

}

}

else if(p.handSize==0){

if(playerID==0){

cout << "Your hand is cleared out and you won the game!" << endl;

}

else{

cout << "Gameover, opponent " << playerID << " wins!" << endl;

}

return true;

}

return false;

}

void Game::setWildColor(Card &card, string color){

card.setColor(color);

cout << "Wild card color set to: " << card.getColor() << endl;

}

void Game::setWildColor(Card &card){

srand(time(0));

int n = rand() % 4;

string color;

switch (n){

case(0):

color = "Red";

break;

case(1):

color = "Blue";

break;

case(2):

color = "Yellow";

break;

case(3):

color = "Green";

break;

}

card.setColor(color);

cout << "Wild card color set to: " << card.getColor() << endl;

}

//Used to help player keep track of game state

void Game::playerConfirmation(){

string confirm;

cout << "\n(Enter any key to continue) " << endl;

cin.ignore();

getline(cin, confirm);

}

void Game::reshuffle(stack<Card>&discardPile){

//keeps top card for the discard pile

Card temp = discardPile.top();

discardPile.pop();

/\*

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

deck.push(discardPile.top());

discardPile.pop();

}\*/

while(!discardPile.empty()){

deck.push(discardPile.top());

discardPile.pop();

}

cout << "Deck is at size: " << deck.getSize() << endl;

cout << "Discard pile is at: " << discardPile.size() << endl;

deck.shuffleDeck();

discardPile.push(temp);

}