Project 1
<UNO>

CIS-5 44187
Name: Seth Tyler
Date: 6/2/17

Introduction

Title: UNO

This game is based on the classic card game of UNO. The concept is mutual with another classic card game named Crazy 8's. In the real game, the objective of the game is to be the first person to completely rid of their cards.

Omissions

Because this game is a stripped down and modified version of the physical card game, there are a few concepts in this program that are absent from the real card game. This includes the wild card system, the ability to choose when you would like to draw a card, the concept of saying "UNO" when one card is left, and the removal of the reverse card, since only 2 players will play and the functionality would be inline with the skip card.

Ruleset

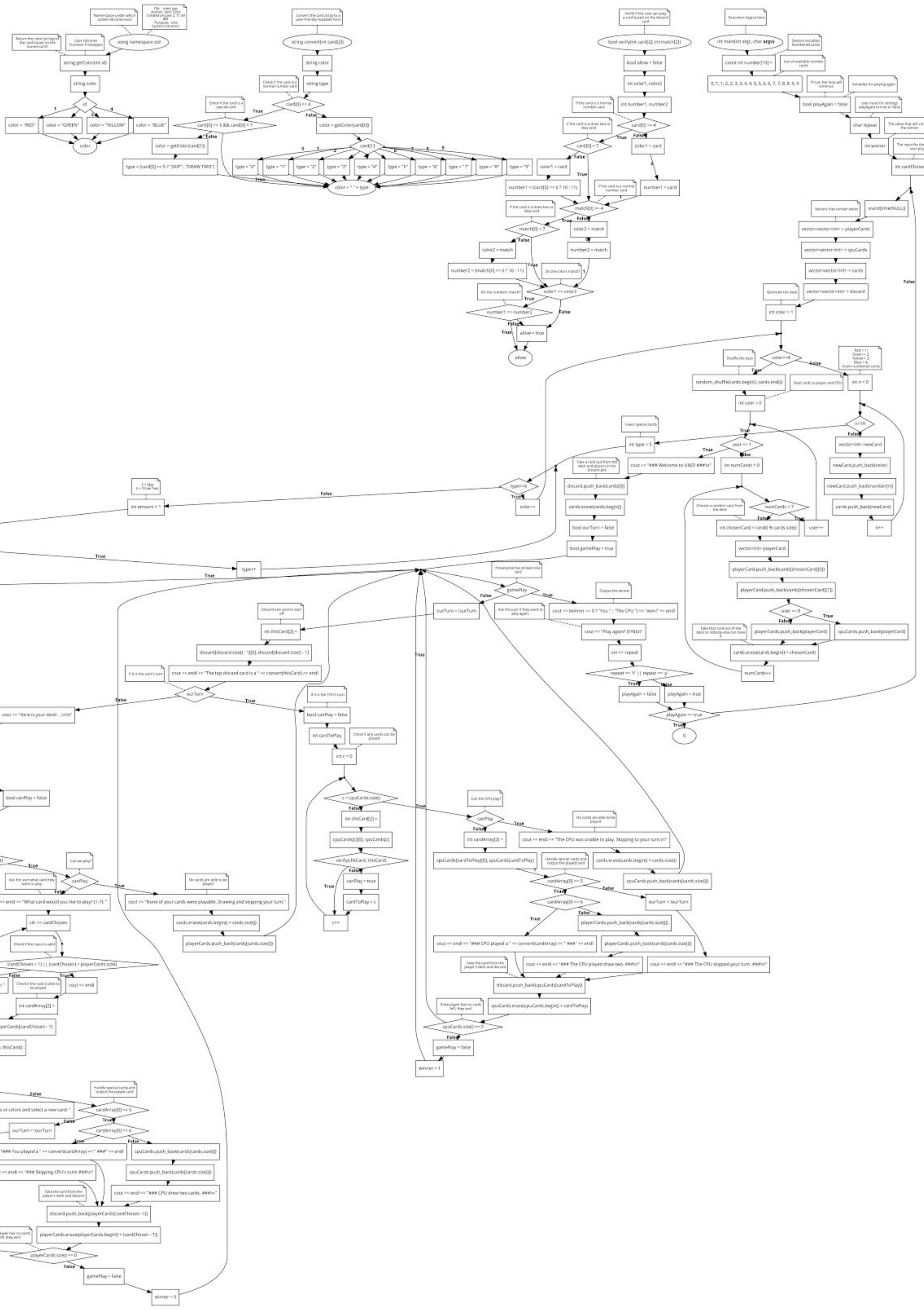
The program will lay down a discard card. The player will always be the first person to have a turn. They will choose to place a card that corresponds to the discard card's color OR number. If the discard card is a skip or draw two card, the player will have to place a matching color or a skip/draw two card. If no cards are able to be played, the program will automatically detect this and skip the player's turn, while also drawing a card for them. If the CPU places a draw two card, two cards will be added to the player's deck, and vice versa for the player.

Summary

Project Size: 344 lines total (including blank lines)
Number of Functions: 3

This project aims to include all sections from chapters 1-8 in the book, though not every concept may be illustrated in this program. Total work time estimated to be about 15 hours.

Flow Chart



Psuedo Code

```
// Libraries

// getColor function
      // return "RED" for 1;
      // return "GREEN" for 2;
      // return "YELLOW" for 3;
      // return "BLUE" for 4;

// convert function
      // if the card is a normal number card
            // use the getColor function to grab the color and also return the
number
      // else if the card is a special card
            // obtain the color and return if it is a skip or draw two

// verify function
      // if the card being played is a normal number card
            // set variables color1 and number1
      // else if it is a special card
            // set variables color1 and number1
      // if the discard card is a normal number card
            // set variables color2 and number2
      // else if it is a special card
            // set variables color2 and number2

      // check if color or number variables match

      // return allow

// int main

      // initialize variables
      // do loop
            // set random seed
            // initialize vectors for card decks

            // for loop, cycle through colors
                  // for loop, cycle through numbers
                        // add to the deck

                  // for loop, cycle through special cards
                        // for loop, cycle through types
                              // add to the deck

            // shuffle the cards vector
            // for loop, cycle between player and cpu
                  // for loop, cycle 7 times for each card
                        // deal a random card

            // Output "Welcome to UNO!"

            // Take out an element from the cards vector and place it in the
discard vector

            // while loop for gamePlay being true
                  // switch turns with ourTurn variable

                  // create an array for the discard card to output what it is
in readable form

                  // check if it is the player's turn
                        // output the player's deck (use a for loop to cycle
through the playerCards vector)

                        // verify if any of the player's cards can be played
with another for loop
                              // call verify function and set canPlay to true
if true

                        // if canPlay
                              // Output asking what card they want to play,
accept input

                              // while loop to verify input, cycle if invalid
input

                              // create an array for the card the player chose
                              // call verify to check if it is a valid card

                              // while loop until validCard is true

                              // if skip card
                                    // skip cpu's turn
                              // if draw two card
                                    // add two card's to cpuCard's vector

                              // add a card to the discard vector
                              // take a card from the playerCards vector

                              // if playerCards size is 0, they have no cards
and they win.
                                    // set gamePlay to false and winner to 0
                        // if not canPlay
                                // take a card from the cards vector, push it
into the playerCard's vector and go to the CPU's turn

                // if it is the cpu's turn
                        // for loop checking if any cards are able to be
played, then set cardToPlay if there is a card

                        // if canPlay

                                // if skip card
                                        // skip cpu's turn
                                // if draw two card
                                        // add two card's to cpuCard's vector
                                // else
                                        // output what card they played

                                // add a card to the discard vector
                                // take a card from the cpuCards vector

                                // if cpuCards size is 0, they have no cards and
they win.
                                        // set gamePlay to false and winner to 1

                        // if not canPlay
                                // take a card from the cards vector, push it
into the cpuCard's vector and go to the player's turn



                // output the winner based on winner variable being 0 or 1

                // ask to play again

                // if input is Y or Y, set playAgain to true
```

Program

```
/*
 * File:   main.cpp
 * Author: Seth Tyler
 * Created on June 2, 11:54 AM
 * Purpose:  Uno
 */

//System Libraries
#include <iostream>  //Input - Output Library
#include <iomanip>
#include <cmath>
#include <string>
#include <fstream>
#include <vector>
#include <stdlib.h>
#include <ctime>
#include<limits>
#include <algorithm>

using namespace std; //Name-space under which system libraries exist

//User Libraries

//Function Prototypes
string getColor(int id);
string getColor(int id = 1) { // Return the color (string) of the card based on
the numerical ID
    string color;
    switch (static_cast<int>(id)) {
        case 1:
            color = "RED";
            break;
        case 2:
            color = "GREEN";
            break;
        case 3:
            color = "YELLOW";
            break;
        case 4:
            color = "BLUE";
            break;
    }
    return color;
}
```

```
string convert(int card[2]) { // Convert the card array to a user-friendly
readable form
    string color;
    string type;

    if (card[0] <= 4) { // Check if the card is a normal number card
        color = getColor(card[0]);
        switch (card[1]) {
            case 0:
                type = "0";
                break;
            case 1:
                type = "1";
                break;
            case 2:
                type = "2";
                break;
            case 3:
                type = "3";
                break;
            case 4:
                type = "4";
                break;
            case 5:
                type = "5";
                break;
            case 6:
                type = "6";
                break;
            case 7:
                type = "7";
                break;
            case 8:
                type = "8";
                break;
            case 9:
                type = "9";
                break;
        }
    } else if (card[0] >= 5 && card[0] < 7) { // Check if the card is a special
card
        color = getColor(card[1]);
        type = (card[0] == 5 ? "SKIP" : "DRAW TWO");
    }
    return (color + " " + type);
}
```

```
bool verify(int card[2], int match[2]) { // Verify if the user can play a card
based on the discard card
    bool allow = false;
    int color1, color2;
    int number1, number2;

    if (card[0] <= 4) { // If the card is a normal number card
        color1 = card[0];
        number1 = card[1];
    } else if (card[0] < 7) { // If the card is a draw two or skip card
        color1 = card[1];
        number1 = (card[0] == 6 ? 10 : 11);
    }

    if (match[0] <= 4) { // If the card is a normal number card
        color2 = match[0];
        number2 = match[1];
    } else if (match[0] < 7) { // If the card is a draw two or skip card
        color2 = match[1];
        number2 = (match[0] == 6 ? 10 : 11);
    }

    if (color1 == color2) { // Do the colors match?
        allow = true;
    } else if (number1 == number2) { // Do the numbers match?
        allow = true;
    }

    return allow;
}

//Execution begins here
int main(int argc, char** argv) {

    // Declare variables

    // Numbered cards
    const int number[19] = {0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8,
9, 9}; // List of available number cards

    // Variables for playing again
    bool playAgain = false; // if true, the loop will continue
    char repeat; // user input for settings playAgain to true or false
    int winner; // The value that will contain the winner
    int cardChosen; // The input for the card the user plays

    do {
        srand(time(NULL));
        // Vectors that contain decks
        vector<vector<int> > playerCards;
        vector<vector<int> > cpuCards;
        vector<vector<int> > cards;
        vector<vector<int> > discard;

        // Generate the deck
        for (int color = 1; color<=4; color++) {
            /*
             Red = 1;
             Green = 2;
             Yellow = 3;
             Blue = 4;
             */

            // Insert numbered cards
            for (int n = 0; n<19; n++) {
                vector<int> newCard;
                newCard.push_back(color);
                newCard.push_back(number[n]);
                cards.push_back(newCard);
            }

            // Insert special cards
            for (int type = 5; type<=6; type++) {
                /*
                 * 5 = Skip
                 * 6 = Draw Two
                 */
                for (int amount = 1; amount<=2; amount++) {
                    vector<int> specialCard;
                    specialCard.push_back(type);
                    specialCard.push_back(color);
                    cards.push_back(specialCard);
                }
            }
        }

        random_shuffle(cards.begin(), cards.end()); // Shuffle the deck

        // Deal cards to player and CPU
        for (int user = 0; user <= 1; user++) {
            for (int numCards = 0; numCards < 7; numCards++) {
                int chosenCard = rand() % cards.size(); // Choose a random card
from the deck
                vector<int> playerCard;
                playerCard.push_back(cards[chosenCard][0]);
                playerCard.push_back(cards[chosenCard][1]);
                if (user == 0) {
                    playerCards.push_back(playerCard);
                } else {
                    cpuCards.push_back(playerCard);
                }
                cards.erase(cards.begin() + chosenCard); // Take that card out
of the deck so nobody else can have it
            }
        }

        cout << "### Welcome to UNO! ###\n";

        // Take a card out from the deck and place it in the discard pile
        discard.push_back(cards[0]);
        cards.erase(cards.begin());

        bool ourTurn = false;
        bool gamePlay = true;

        while (gamePlay) { // If everyone has at least one card
            ourTurn = !ourTurn;

            // Discard one card to start off
            int thisCard[2] = {discard[discard.size() - 1][0],
discard[discard.size() - 1][1]};
            cout << endl << "The top discard card is a " << convert(thisCard)
<< endl;

            if (ourTurn) { // If it is the user's turn

                cout << "Here is your deck! ...\n\n";

                // Display the player's deck of cards
                for (int c = 0; c < playerCards.size(); c++) {
                    int showCard[2] = {playerCards[c][0], playerCards[c][1]};
                    cout << (c + 1) << ". " << convert(showCard) << endl;
                }

                bool canPlay = false;

                // Verify if any of the player's cards can be played
                for (int c = 0; c < playerCards.size(); c++) {
                    int chkCard[2] = {playerCards[c][0], playerCards[c][1]};
                    if (verify(chkCard, thisCard)) {
                        canPlay = true;
                    }
                }
                // Can we play?
                if (canPlay) {

                    // Ask the user what card they want to play
                    cout << endl << "What card would you like to play? (1-7):
";
                    cin >> cardChosen;

                    // Check if the input is valid
                    while ((cardChosen < 1) || (cardChosen) >
playerCards.size()) {
                        cout << endl << "This card is invalid. Please choose
another one: ";
                        cin >> cardChosen;
                    }
                    cout << endl;

                    // Check if the card is able to be played
                    int cardArray[2] = {playerCards[cardChosen - 1][0],
playerCards[cardChosen - 1][1]};
                    bool validCard = (verify(cardArray, thisCard));
                    while (!validCard) {
                        cout << endl << "This card cannot be played. Try to
match the numbers or colors and select a new card: ";
                        cin >> cardChosen;
                        int cardArray[2] = {playerCards[cardChosen - 1][0],
playerCards[cardChosen - 1][1]};
                        validCard = (verify(cardArray, thisCard));
                    }

                    // Handle special cards and output the played card
                    if (cardArray[0] == 5) {
                        ourTurn = !ourTurn;
                        cout << endl << "### Skipping CPU's turn! ###\n";
                    } else if (cardArray[0] == 6) {
                        cpuCards.push_back(cards[cards.size()]);
                        cpuCards.push_back(cards[cards.size()]);
                        cout << endl << "### CPU drew two cards. ###\n";
                    } else {
                        cout << endl << "### You played a " <<
convert(cardArray) << " ###" << endl;
                    }

                    // Take the card from the player's deck and discard
                    discard.push_back(playerCards[cardChosen -1]);
                    playerCards.erase(playerCards.begin() + (cardChosen - 1));
                    // If the player has no cards left, they win!
                    if (playerCards.size() == 0) {
                        gamePlay = false;
                        winner = 0;
                    }

                } else {
                    // No cards are able to be played
                    cout << "None of your cards were playable. Drawing and
skipping your turn.";
                    cards.erase(cards.begin() + cards.size());
                    playerCards.push_back(cards[cards.size()]);
                }
            } else { // If it is the CPU's turn.
                bool canPlay = false;
                int cardToPlay;

                // Check if any cards can be played
                for (int c = 0; c < cpuCards.size(); c++) {
                    int chkCard[2] = {cpuCards[c][0], cpuCards[c][1]};
                    if (verify(chkCard, thisCard)) {
                        canPlay = true;
                        cardToPlay = c;
                    }
                }

                // Can the CPU play?
                if (canPlay) {

                    int cardArray[2] = {cpuCards[cardToPlay][0],
cpuCards[cardToPlay][1]};

                    // Handle special cards and output the played card
                    if (cardArray[0] == 5) {
                        ourTurn = !ourTurn;
                        cout << endl << "### The CPU skipped your turn. ###\n";
                    } else if (cardArray[0] == 6) {
                        playerCards.push_back(cards[cards.size()]);
                        playerCards.push_back(cards[cards.size()]);
                        cout << endl << "### The CPU played draw two. ###\n";
                    } else {
                        cout << endl << "### CPU played a " <<
convert(cardArray) << " ###" << endl;
                    }

                    // Take the card from the player's deck and discard
                    discard.push_back(cpuCards[cardToPlay]);
                    cpuCards.erase(cpuCards.begin() + cardToPlay);
                    // If the player has no cards left, they win!
                    if (cpuCards.size() == 0) {
                        gamePlay = false;
                        winner = 1;
                    }
                } else {
                    // No cards are able to be played
                    cout << endl << "The CPU was unable to play. Skipping to
your turn.\n";
                    cards.erase(cards.begin() + cards.size());
                    cpuCards.push_back(cards[cards.size()]);
                }
            }
        }

        // Output the winner
        cout << (winner == 0 ? "You " : "The CPU ") << "won!" << endl;


        // Ask the user if they want to play again.
        cout << "Play again? (Y/N)\n";
        cin >> repeat;
        if (repeat == 'Y' || repeat == 'y') {
            playAgain = true;
        } else {
            playAgain = false;
        }
    } while (playAgain == true);

    return 0;
}
```