**Project 1**

**UNO Card Game**

**CIS-17C**

**47065**

**April 26, 2023**

**David Guadian**

**1 Introduction**

The card game UNO has been around since 1992 it was been the heart of many parties and family game nights. UNO is a game that can be enjoyed no matter what age and is why I decided to program the game of UNO. Also UNO being one of the games I most have experience playing is the reason I choose this project to work on.

|  |  |
| --- | --- |
| Number of Classes | 6 |
| Number of comments | 66 |
| Number of Lines | 1088 |

This project took around 40 hours with an estimated 5 hours planning out the project and the rest programming and working through issues within the project.

Project Locations: <https://github.com/dguad54/CIS17C-Project1>

**2 Approach to Development**

**2.1 Concepts**

The core concepts of this program are in the classes. Using the object oriented approach for a card game like UNO were there are different objects involved with the program for example the deck, individual cards, and player. The UNO program also utilizes containers including stack, list, maps, and sets. When the game is taking place different actions have to take place like shuffling and running the game. To effectively achieve these task different STL algorithms have to be used including for each, swap, and sort.

**2.2 Version Control**

Version Control

Version 1.0.0

Date: April 15, 2023

Changes: Started the main functionality from the UNO game including basic Card, Deck, and Player classes that would be used for the program. All classes were done on the main.cpp with the goal of making the UNO program work.

Author: David Guadian

Version 1.1.0

Date: April 20, 2023

Changes: Switched all classes into separate class cpp and header files. Added algorithms needed for sorting, for each and swap for different methods in the deck class.

Version 2.0.0

Date: April 26, 2023

Changes: Fixed most errors in the code and implemented the containers needed for the project. Added a displaycard and displayStack classes in order to efficiently be able to show the user a uno set of cards to better understand the way the game is played.

**3 Game Rules**

The game can be played with 2-4 players, and the objective is to be the first player to get rid of all your cards.

To start the game, each player is dealt 7 cards from the deck, and the remaining cards are placed in a draw pile. The top card of the draw pile is turned over to begin a discard pile.

On each turn, a player must match the top card of the discard pile either by number, color, or symbol. If a player cannot play a card, they must draw a card from the draw pile. If they still cannot play, their turn ends. However, if the card drawn can be played, the player can immediately play that card.

Special cards in UNO include:

* Draw Two: The next player must draw two cards from the draw pile and miss their turn.
* Reverse: The direction of play is reversed.
* Skip: The next player is skipped and loses their turn.
* Wild: The player can choose any color to continue play.
* Wild Draw Four: The player can choose any color to continue play, and the next player must draw four cards from the draw pile and miss their turn.

The first player to get rid of all their cards wins the game.

**4 Description of Code**

**4.1 Organization**

**4.1.2 Menus:**

User is greeted with a meu allowing for the user to view the rules of the game, general advice and the choice to exit the menu. This was intended to make new players have a way to understand the game of UNO and know how to play. While input checking the range of values they input for the switch.

The next menu allows players to view different card sets in order to be familiar with the cards they will be playing with. There are different forms of access for a user to view the deck. The first being a view with a stack in which a container stack is used to display seven cards to the user. This stack is randomly generated. The next form of viewing the cards is using the same stack container but now using the sort algorithm the colors are ranked my ascii value and numbers are compared to sort the cards in number order and color going red as the lowest then green, blue, and ending with yellow. The last menu option for the user is a set of cards in which a set container is used to display a whole deck of cards.

**4.1.2 Game Logic:**

To start the game user inputs the amount of players that will be playing the game using a while loop that checks if the input is over 1 player and under 4 ensures the game will run smoothly. Next the deck is created from the deck class then shuffle using the random\_shuffle algorithm in the shuffle method from the stl library.

A pointer array of class player is created based on the number of players the user has inputted and using the draw method from class deck a card is taken from the deck with the method draw() which decrements one card from the deck class and adds a card to the object of type player. This process is repeated with the use of a for loop which gives 7 cards to each player from the player array making it a double for loop in use.

To show that the player has the proper cards a for loop is used to display the cards in hand from the object player. After this step the game starts in a while loop that ends when a player has a value of cards = 0;

To start each player turn a branch of if statements are used to force a player to draw two cards if the previous player has put down a plus two card or a plus four. The next while loop is what action the user wants to take. If they have no proper cards they input -1 to draw a card and move on to the next turn. If they choose to play a card from their hand they input the index of the card. If the cards match the card is removed from the hand by using the .erase function then added to the discard pile. If the user presses the index for a wild card the card is removed from their hand and they are able to choose the color with an input of type string. After their turn the index of cards held by player is checked with ->get\_size() if the index of cards is equal to 0 the game ends and the player wins.

**4.1.3 Classes:**

***Card:***

The card class has two private member variables: number and color, which represent the value and color of the card, respectively. The number variable can range from 0 to 14, where 10 represents "Draw 2", 11 represents "Skip", 12 represents "Reverse", 13 represents "Wild", and 14 represents "Draw 4 Wild". The color variable represents the color of the card and can be one of five colors: red, green, blue, yellow, or wild.

The card class has two constructors: a default constructor that initializes number to 0 and color to wild, and a parameterized constructor that takes an integer num and a COLOR enumeration col as arguments and initializes number and color with these values, respectively.

The card class also defines two operator overloads: operator== and operator!=. The operator== overload compares two cards and returns true if they have the same number or color or if either of them has a wild color. The operator!= overload returns the opposite of operator==.

Finally, the code defines an overloaded << operator that allows the card class to be output to an ostream. The << operator first outputs the card's number value, using a switch statement to convert the special values (10-14) to their corresponding names. It then outputs the card's color value, using a switch statement to convert the COLOR enumeration to a string representation.

***Deck:***

This code defines a **deck** class that represents a deck of cards used in a card game. The deck consists of cards with different colors and numbers. The deck is implemented as an array of **card** objects. The class provides methods for creating a deck, shuffling the deck, drawing a card from the deck, adding a card to the deck, and printing the deck.

The **deck** class contains a private member variable **ptr\_deck** which is a pointer to an array of **card** objects. The class also has a private member variable **size** which keeps track of the number of cards in the deck.

The **deck** class provides a default constructor that initializes the **ptr\_deck** and **size** variables. The constructor also initializes an array **card\_counts** with the count of cards of each number in the deck.

The class provides a copy constructor and an assignment operator for creating a copy of a **deck** object.

The **create()** method creates a new deck by adding cards to the **ptr\_deck** array. The cards are added to the array in a specific order based on their number and color.

The **shuffle()** method shuffles the cards in the deck. The method uses the **std::random\_device** class to generate a random number generator and the **std::mt19937** class to generate random numbers. The method then uses the **std::swap()** function to swap the cards in the deck.

The **draw()** method removes a card from the top of the deck and returns it. If the deck is empty, the method returns a default-constructed card.

The **add\_card()** method adds a card to the bottom of the deck. If the deck is full, the method returns -1.

The **print\_deck()** method prints the cards in the deck along with their count.

The **clear()** method deallocates the memory allocated to the **ptr\_deck** array.

The **get\_size()** method returns the number of cards in the deck.

The **print\_values()** method prints the value of each card in the deck.

***Player:***

This code defines the implementation of a class named player which represents a player in a card game. The class has a default constructor, a copy constructor, an overloaded assignment operator, a destructor, and several member functions to manipulate the player's hand of cards.

The default constructor initializes an object of the player class with an empty hand of cards. The size data member is set to 0.

The copy constructor creates a new player object with the same contents as another player object passed as a parameter.

The overloaded assignment operator assigns the contents of one player object to another player object. It first checks if the two objects are not the same, then clears the existing contents of the object being assigned to and copies the contents of the other object to it.

The destructor frees the memory allocated by the player object.

The hand\_add() function adds a new card to the player's hand by pushing it to the front of the hand list and incrementing the size data member.

The hand\_remove() function removes a card from the player's hand at a given position and returns a copy of the removed card. If the position is out of range, it returns an empty card.

The copy() function copies the contents of another player object to the current object.

The clear() function removes all the cards from the player's hand and sets the size data member to 0.

The print() function prints the cards in the player's hand to the console, each card preceded by its index in the hand.

The get\_size() function returns the number of cards in the player's hand.

The peek() function returns a copy of a card in the player's hand at a given position without removing it. If the position is out of range, it returns an empty card.

***DisplayCard:***

This code defines a class named displaycard, which represents a card in a card game. The class has a constructor that takes in a color, an action, and an optional number for the card, and a default constructor that initializes the card's properties to empty values or 0.

The class also has a destructor which is empty, indicating that there is no dynamic memory allocation within the class that needs to be freed upon destruction.

There are three member functions which return the color, action, and number of the card, respectively. These functions are marked as const to indicate that they do not modify the state of the displaycard object.

The purpose of the displaycard class is to serve as a base class for displaying cards without having to change the == operator used in the other classes for comparing.

DisplaySet:

This code defines a class named displaySet, which has a single member function named displayCardSet(). The displayCardSet() function takes in a set of displaycard objects as its argument and prints the contents of the set to the console.

The displaySet class has a default constructor and a destructor, both of which are empty and do not perform any actions.

The displayCardSet() function uses a range-based for loop to iterate over each element in the input set of displaycard objects. For each card in the set, the function checks the color, number, and action of the card and prints them to the console in a formatted string. If the color, number, or action of the card is empty, that property is not printed to the console.

Overall, this code provides a simple way to display the contents of a set of displaycard objects. It could be used in a card game program to display the current state of the deck or the cards in a player's hand.

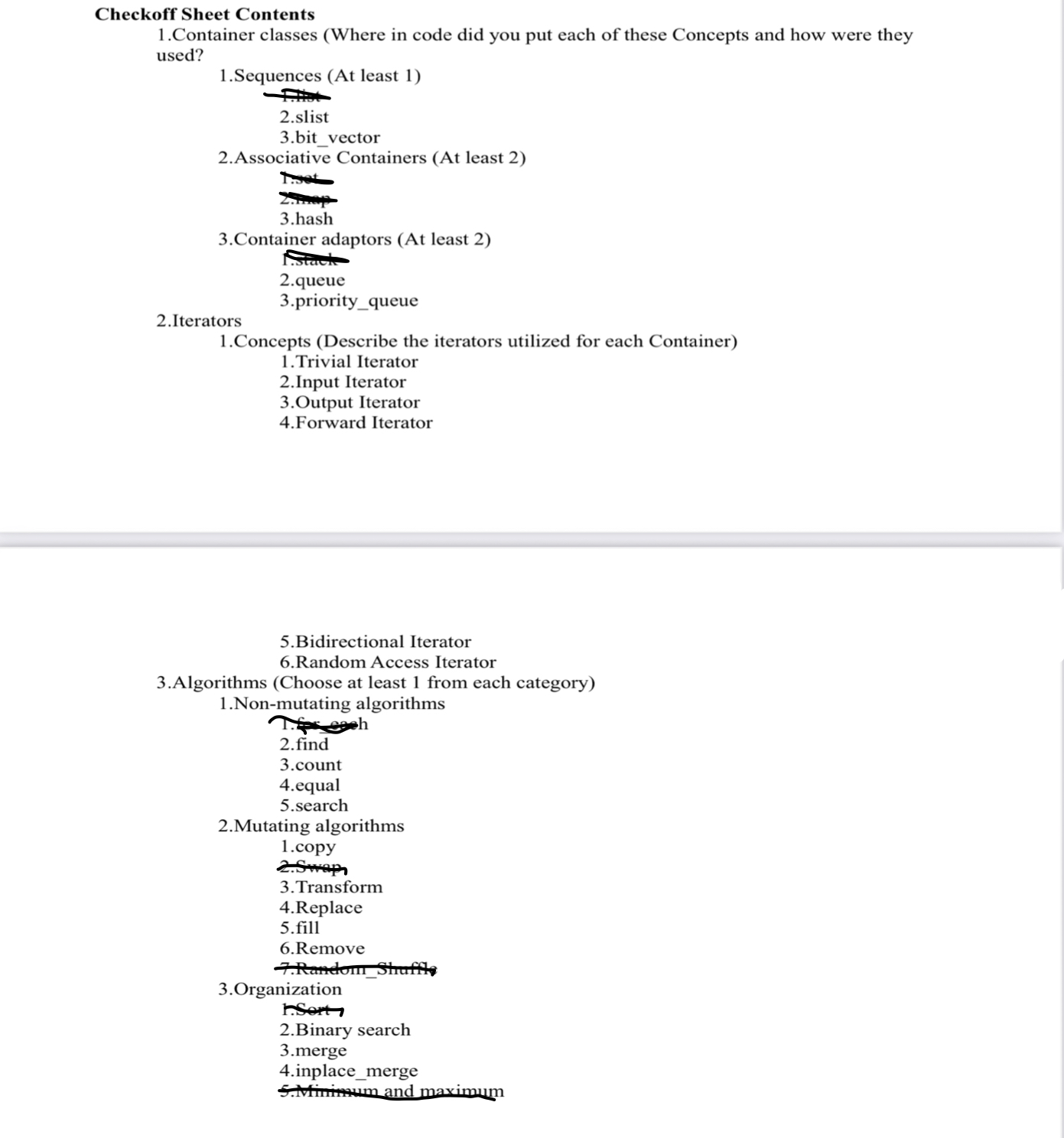
***DisplayStack:***

This code defines a class called displayStack that contains two member functions: displayRandomCards and displaySortedStacks.

The displayRandomCards function generates a specified number of random UNO cards and displays them in the console. It creates an array of possible colors and actions, uses a random number generator to select a random color, action, and number (if applicable) for each card, and adds the resulting displaycard objects to a stack. It then pops each card off the stack and displays its color, number, and/or action in the console.

The displaySortedStacks function generates an array of 5 random UNO cards and sorts them by color, action, and number. It uses a similar process as displayRandomCards to generate the cards and store them in an array. It then uses the std::sort algorithm to sort the cards according to the specified criteria.

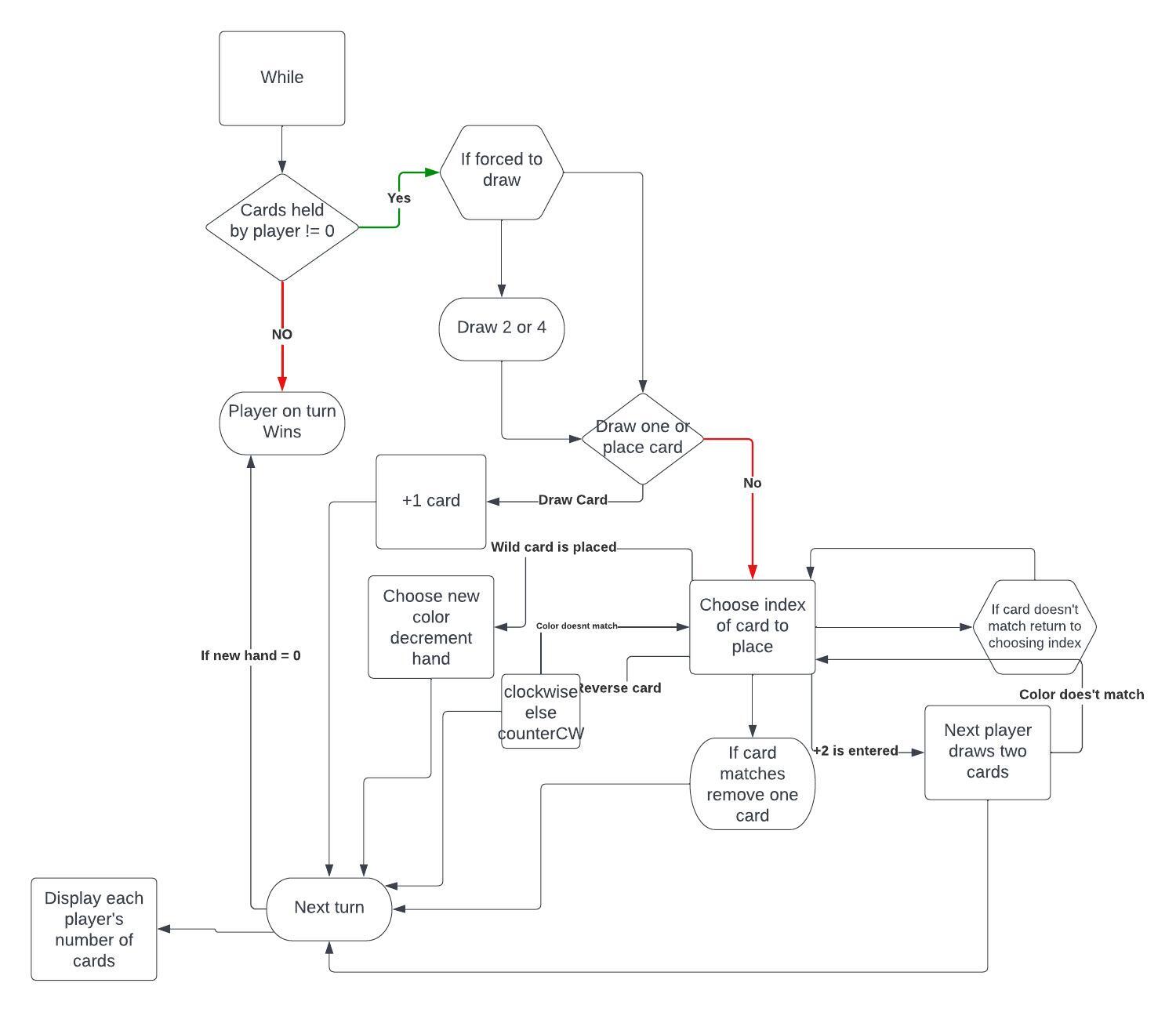
**5 Checkoff Sheet**

****

**6 Documentation of code**

**6.1 Flowchart**

**GameLogic:**

****

**6.2 Pseudo-Code**

*Source code:*

// Declare constants for different game modes

Constants PRINT\_ALL\_PLAYERS, TEMP\_DECK, TURN, and TEST

// Define a function to display an introduction to the game and give users the option to view rules or general advice

Function display\_intro() {

// Display a welcome message and menu

Print "Welcome to UNO! Select any information you may need:"

Print "1. View Rules"

Print "2. General Advice"

Print "3. Exit Intro Menu"

// Get the user's choice

Read user\_choice

// Check if the user's choice is invalid

If user\_choice is less than 1 or greater than 3, then:

Print "Invalid input. Please enter a valid number."

Return

// Display the appropriate information based on the user's choice

Switch user\_choice:

Case 1:

Print "Here are the rules of UNO:"

Print "- The goal of UNO is to be the first player to get rid of all of your cards."

Print "- You can play a card if it matches the color or number of the top card of the discard pile."

Print "- If you don't have a playable card, you must draw a card from the draw pile."

Print "- If you draw a Wild card, you can choose any color to play it as."

Print "- If you draw a Draw Four card, you must draw four cards from the draw pile and your turn is skipped."

Case 2:

Print "Here are some general tips for playing UNO:"

Print "- Try to keep track of the cards that have been played."

Print "- If you have a Wild card, save it for when you need it."

Print "- If you have a Draw Four card, don't be afraid to use it."

Case 3:

Print "Have fun!"

// Define a function to display different options to view the deck of cards

Function card\_display():

// Display the menu

Print "Get familiar with UNO cards and Decks"

Print "Choose how you want to you would wish to see the deck"

Print "1. Display a stacked set of cards: "

Print "2. Display a sorted deck: "

Print "3. Display a example set of cards: "

Print "4. Exit deck display menu"

// Get the user's choice

Read input

// Display the appropriate deck based on the user's choice

Switch input:

Case 1:

Display a stack of randomly chosen cards

Case 2:

Display a sorted deck

Case 3:

Display an example set of cards

Case 4:

Print "Exiting deck display menu."

Default:

Print "Invalid input. Please enter a number between 1 and 4."

// Define a function to confirm the current player's turn

Function confirm\_turn(x):

Print "Confirm Player x by typing 'x' and pressing enter: "

// Keep looping until the player's input matches their number

Read temp

While temp is not equal to x:

Read temp

// Define a function to convert a string to a color

Function FromString(str):

If str is equal to "red", return red

If str is equal to "green", return green

If str is equal to "blue", return blue

If str is equal to "yellow", return yellow

Return wild

// Start the main program

Function main():

// Clear the screen and display the introduction and card information

Clear screen

Call display\_intro()

Call card\_display()

// Get the number of players from the user

Read amount\_players

// Create gameDeck and shuffle it

// Create an array of players with amount\_players size

// For each player, deal them 7 cards from gameDeck

// Create a temp\_deck and draw a card from gameDeck to be the first card played

// While card\_flag is 0, draw another card from gameDeck

// If temp\_card.color is not wild, set card\_flag to 1 and played\_card to temp\_card

// Else, add temp\_card to temp\_deck

// Print each player's hand

// Randomly select a player to go first, and confirm their turn

// Force\_draw\_bool to false

// Set turn\_flag to 1

// Set win to 0

while win is 0, do the following:

// Clear the screen

Clear screen

// Get a pointer to the current player

// Display the current player's turn

// If force\_draw\_bool is true, then do the following:

// If played\_card's number is 10, then for i from 0 to 1,

// Do the following: draw a card from gameDeck and add it to curr\_player's hand

// If played\_card's number is 14, then for i from 0 to 3, do the following:

// Draw a card from gameDeck and add it to curr\_player's hand

// Set force\_draw\_bool to false

// Print out the cards remaining for each player

// Print out the played card

// Print out the cards in the current player's hand

// While check\_flag is 0, do the following:

// Ask the current player which card they want to play or if they want to draw a card

// If the player wants to draw a card, then do the following:

// Draw a card from gameDeck and assign it to draw\_temp

// If draw\_temp equals played\_card and draw\_temp's color is not wild, then do the following:

// While play\_draw\_flag is 0, do the following:

// Ask the current player if they want to play the drawn card

// If the player says yes, then set played\_card to draw\_temp, add it to temp\_deck, and set force\_draw\_bool to true

// If its number is between 10 and 14:

// If the player says no, then add draw\_temp to curr\_player's hand set check\_flag to 1

// Else, add draw\_temp to curr\_player's hand and set check\_flag to 1

// If the player wants to play a card, then do the following:

// If the card at the index given by the player is compatible with the played card, then

// Do the following: set played\_card to the card at the index given by the player,

// Remove it from the player's hand, and add it to temp\_deck

// If its number is not between 1 and 9, set check\_flag to 1

// Else, tell the player their card is not compatible with the played card and continue the loop

if curr\_player's hand is empty, then

// Set win to 1

Set win to true

Print the winning player's name and end the game

if the played card is an

if the played card is an 11 and a force draw is active:

If turn flag is equal to 1:

Add 2 to the turn

Else:

Subtract 2 from the turn

Else if the played card is a 12 and a force draw is active:

If the amount of players is equal to 2:

Add 2 to the turn

Else:

If turn flag is equal to 1:

Set turn flag to -1

Subtract 1 from the turn

Else:

Set turn flag to 1

Add 1 to the turn

Else:

If turn flag is equal to 1:

Add 1 to the turn

Else:

Subtract 1 from the turn

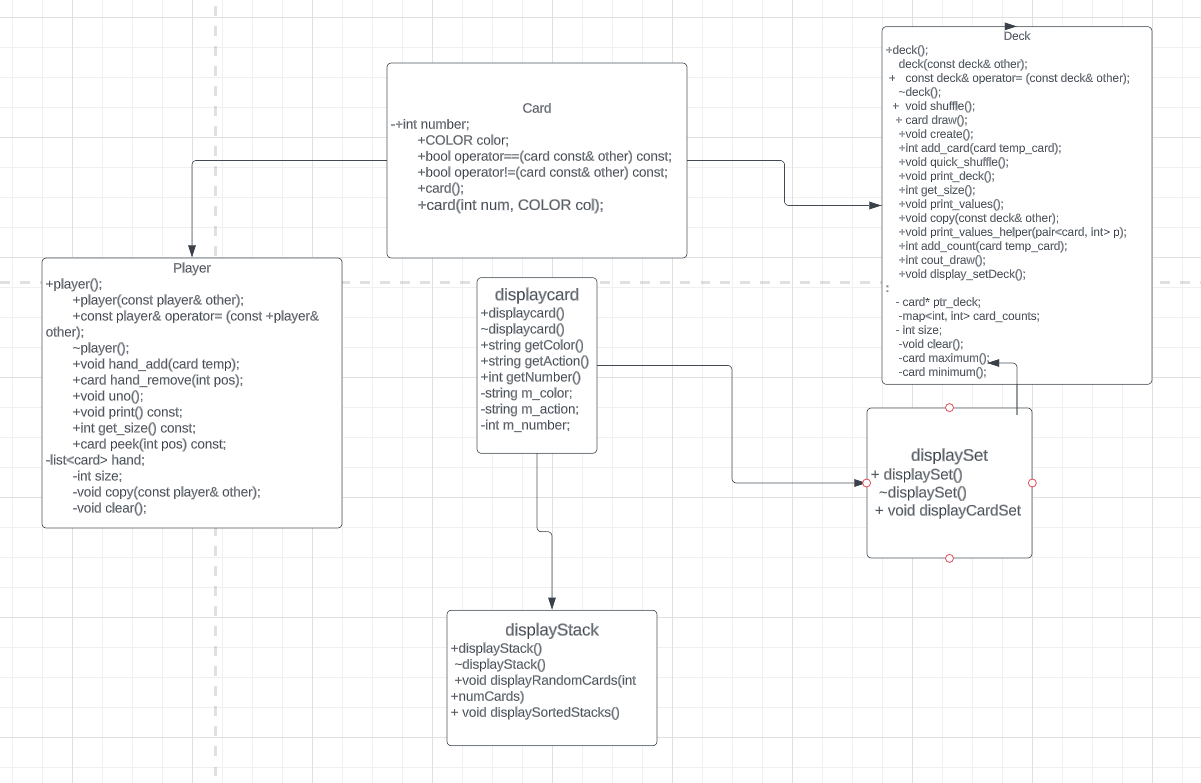
Clear the console screen

Print the remaining cards for each player

Print the temporary card that was played

Confirm whose turn it is

**6.3 UML**

****