Class: Advanced Programming

Mauricio Peón García A01024162

Romeo Varela Nagore A01020736

Pablo Terán Ríos A01421434

Teacher: Gilberto Echeverría Furió

Delivery Date: 16/October/2019

# UNO++

The problem we are going to solve is the creation of a digital UNO game created for 2 players up to 4 players. The game will use typical UNO rules like: clockwise way, action cards and an infinite pile of cards. For more information visit: https://www.unorules.com/


There will be a server which 4 clients will be able to connect and play a game of UNO. The client will wait for the clients to connect and manage the game by dealing with the plays of each client and deal cards to each player. Each client will only see their hands and the current play from each client and send their play to the server.

There will be a dynamic array that contains all the users or players; The turns will always start clockwise and in the case a reverse card is played, the game will now be counter clockwise.

Each hand will be a dynamic matrix of size Nx2 which there will be N number/cards the player will have and the second row will be the color of the card.

Row 1:

- 0 - 9 will use their own number.
- 10 will be the +2.
- 11 will be the skip turn.

- 12 will be the reverse.
- -1 will be the wildcard

Row 2:

- 0 will be black color used for wildcards.
- 1 will be the color red.
- 2 will be the color blue.
- 3 will be the color green.
- 4 will be the color yellow.

The server will manage turns using a pointer in the player array created at the connection of the clients. And send the current card in play and the player the turn belongs to.

Once received, the client will see if the turn is his own, if so it will be able to play a card from his own hand matrix. If no option is available, the client can send another card and wait for the player to reply if it's playable or not.

- Dynamic memory: We are going to have a dynamic array that contains all the players. It will also be used in the players' array of cards because it can be infinite. We still don't know if we will use vectors or our dynamic arrays to solve this problem, but we think vectors will be very useful to just push a new player or a new card.
- Pointers: We need pointers to know the turns of the users, also if one player jumps the next one. All the arrays are also pointers and we will use them with the players and the cards.
- Process creation: Every client is a new process and the server is a process too.
- Inter process communication: The server and the client will send the following data:
    - The player's turn
    - The player's cards
    - The decision of the first player. To start the game or to wait for more players.
    - The decision of the player. To start a new game or to close.
    - Who won the game

Threads: We need threads to share data between players. (This point will be considered as extra)

**Communication Protocol**

| | Server | Client |
|---|---|---|
| **The client start the game** | | **Start** |
| **The server send a player number to the client** | **OK:Player1** | |
| **The first player send if he wants to wait for another player or start the game** | | **START or WAIT** |
| **The server send to the client if a player join to the game** | **NEW:number of players** | |
| **When the game starts, the server send the initial cards to each player** | **CARDS:1,3,2,3,2,3,2** | |
| **The player send a card to play when it's his turn. If it's a wildcard the color also is send it.** | | **CARD:1**<br>**CARD:0 red** |
| **If the player doesn't have any card to put then the server sends a new card to him** | **NEWCARD:3** | |
| **The server send a message if the card isn't valid** | **BAD** | |
| **The server send the player's card to the other players** | **CARD:2** | |
| **The server send to the client who won the game** | **WON:player1** | |

**Libraries**

Some of the libraries we are going to use are:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <netdb.h>

#include <errno.h>

#include <time.h>

#include <arpa/inet.h>

#include <vector>

#include <iostream>

#include <SFML/Audio.hpp>

#include <SFML/Graphics.hpp>

These are some of them, but in the progress we can learn some new ones to adapt them to our game.

The SFML will be implemented to get a graphic interface for the game.

The game will be done in C++, so we can manage classes and some better libraries that can help us build the game in a more efficient way and with some cool graphics, for this, we plan to use so many topics seen in class and some extras.

We know there can be some changes in the process of development but this is the general background we are basing in.