INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS SANTA FE

UNO++

Students	IDs
Pablo Terán Ríos	A01421434
Mauricio Peón García	A01024162
Romeo Varela Nagore	A01020736

Course name: Advanced Programming

Teacher: Gilberto Echeverría Furió

Delivery Date: 9 December 2019

Index

Problem

The problem we encountered is that sometimes you don't have cards to play UNO with your friends. But what if you were able to play UNO without cards just by downloading a simple program capable of running in all possible operating systems.

Our game is capable of handling up to 4 players to play while connecting to a server on the same internet connection. The game has the simple functionalities of UNO handling number cards going from 0 to 9 and special cards like Reverse, represented by an R, Skip or Jump turn represented by a J, +2 card which gives the next in turn 2 cards, change Color wildcard, which is represented by a C, and finally, the +4 change color wildcard.

Functions

Server

void usage(char * program);
If the server is not initiated correctly, the function will display how to use it and close the program.

void waitForConnections(int server_fd);
There is a poll waiting for the clients to connect. When 2 players connect, a timer starts to start the game.

void game(map<int, pair<string, int> > &players, vector <int> &clients);
Handles the game once all players connected or the timer ended.

void setupHandlers();
Blocks all signals from the program except Cntrl + C.

void onInterrupt(int signal);
If a interruption is detected, change a variable to close the program.

void setPlayer(int client, map<int, pair<string, int> > &players, int i);
Every time a player connects, sets it’s variables in a map.

void sendFirstHands(vector <int> &clients);
Creates the first hand for each player and sends it to each player.

void confirmations(vector <int> &clients);
Confirms all users have received a message that was sent.

pair<int, int> gameConfirmations(map<int, pair<string, int> > &players, int who, vector <int> &clients, pair<int, int> &current);
Handles each turn and confirms all users have either received the current card and have played when it's their turn.

void sendNames(map<int, pair<string, int> > &players, vector <int> &clients);
Sends the names of each player to the client to know who he is playing against.

void turnHandle (map<int, pair<string, int> > &players, vector<int> &clients, int who, pair<int, int> &current);
The game function call this function every turn.

void sendTurn(string s, vector <int> &clients);
Sends the current state of the game to each player.

string createString(map<int, pair<string, int> > &players, vector<int> &clients, int who, pair<int, int> &current);
Creates the string that is sent to the players including the current cards.

string createRequest(map<int, pair<string, int> > &players, vector<int> &clients, int who, pair<int, int> &current);
Creates the string that is sent to the players whenever someone asks for another card.

void finishGame(vector<int> &clients, string winer);
If a player card counter reaches 0, he wins and a message is sent to all the players.

Client

void usage(char * program);
Explanation to the user of the parameters required to run the program

void attendRequest(int connection_fd);
Handle all client request for the game

void initGame(char *buffer, vector<pair<int,int> > *hand, int *turn);
Save initial cards and player's turn

void saveCards(char *buffer, vector<pair<int,int> > *hand);
Save player cards

void dealWithTurn(char *buffer, pair<int, int> *current_card ,int * current_turn, const int my_turn, const char total_players, vector<pair<string,int> > *players_names, vector<pair<int,int> > *hand);
Save all data related with the current turn

void savePlayersName(char *buffer, vector<pair<string,int> > *players_names, int *total_players);
Save the name of all players

void sendConfirmation(char *buffer, int connection_fd);
Confirmation message that will be send to the server

int makeMove(int connection_fd ,int option, vector<pair<int,int> > *hand);
Send card, selected by the player ,to the server

void printMyCards(vector<pair<int,int> > hand);
Print player cards in console

void setupHandlers();
Blocks all signals from the program except Cntrl + C.

void onInterrupt(int signal);
If a interruption is detected, change a variable to close the program.

Other

bool verifyCard(pair <int,int> * card,pair <int,int> * deck_card);
Check if the card given by the player can be played.

bool wild_card();
Get wild card probability for the UNO game.

bool winner(vector<pair <int, int> > * hand);
Check if a player won.

void vectorToString(vector<pair <int, int> > * hand, char * buffer);
Convert vector data in a long string.

pair <int, int> get_card();
Get a UNO game card.

string getColor(pair <int,int> * card);
Get a card color given two integers.

vector<pair <int, int> > player_hand();
Save initial hand.

void deleteCardAtPosition(vector<pair <int, int> > * hand,int position);
Delete a card from a hand at a given position.

void renderWindow(vector<pair <int, int> > * hand, pair <int,int> * current_card,vector<pair <string, int> > * players,int turn, sf::RenderWindow * window);
Function that renders the game.

std::vector<sf::RectangleShape> createRectangles(vector<pair <int, int> > * hand);
Function that creates the rectangles of the cards.

sf::Color getRectangleColor(int color);
Helper funtion that returns the inside color of the card.

sf::RectangleShape createCurrentRectangle(pair <int,int> *current_card);
Function that creates the rectangle of the current card.

pair<int,int> getCoords(int player);
Function that indicates the coordinates to place players names.

pair<int,int> getTurnCoords(int turn);
Function that indicates the coordinates to place arrows turn.

string getArrowTurn(int turn);
Function that crates the arrow.

string getSpecialCard(string number);
Function thet changes the id of the card to a more understandable rule.

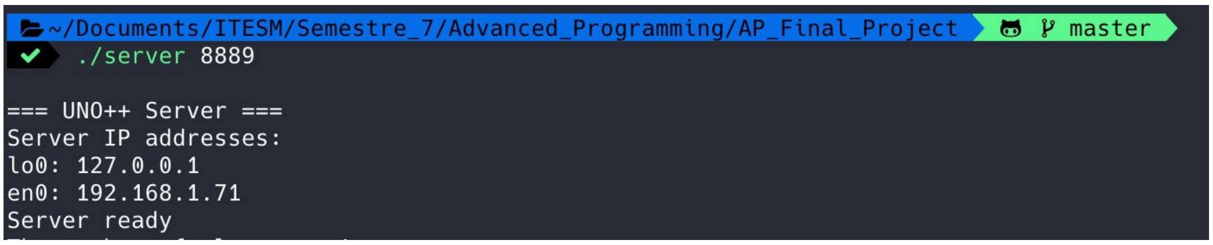
Step by Step Instructions

In order to be able to run the program, the computer must be capable of running the SFML graphics and compile a makefile using the make command. Once the compilation is done, you are able to start the game with the following instructions.

Server

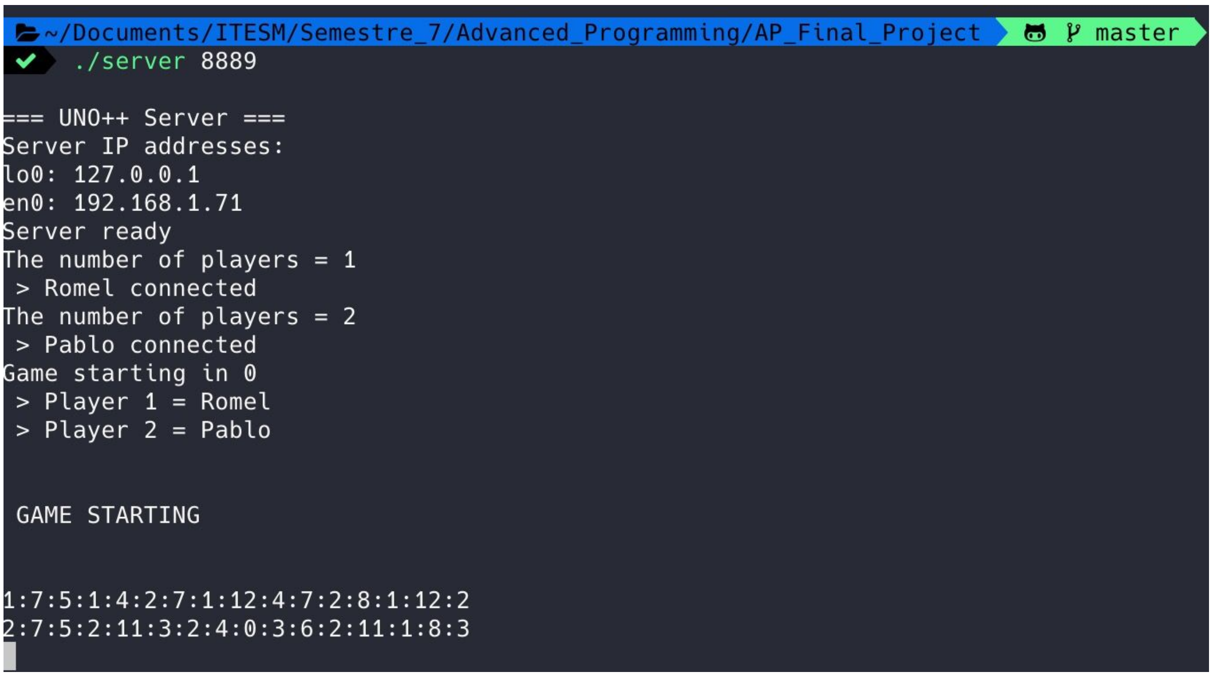
The server is only required to host the game and the program cannot play as a server/user. If the administrator want to play, a new run of the program client needs to be used.

1) Run the ./server {port number} command.



```
~/Documents/ITESM/Semestre_7/Advanced_Programming/AP_Final_Project   master
✓  ./server 8889

=== UNO++ Server ===
Server IP addresses:
lo0: 127.0.0.1
en0: 192.168.1.71
Server ready
```

2) Once 2 or more players are connected, the game has a 10 seconds timer that will start the game if no new connections are detected.



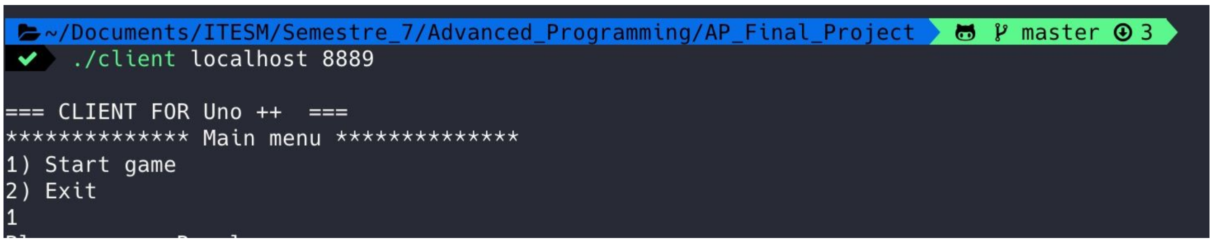
```
~/Documents/ITESM/Semestre_7/Advanced_Programming/AP_Final_Project   master
✓  ./server 8889

=== UNO++ Server ===
Server IP addresses:
lo0: 127.0.0.1
en0: 192.168.1.71
Server ready
The number of players = 1
 > Romel connected
The number of players = 2
 > Pablo connected
Game starting in 0
 > Player 1 = Romel
 > Player 2 = Pablo


 GAME STARTING


1:7:5:1:4:2:7:1:12:4:7:2:8:1:12:2
2:7:5:2:11:3:2:4:0:3:6:2:11:1:8:3
```

3) If the program reaches 4 users, the game will start automatically.
4) Let the program run on the background.

Client

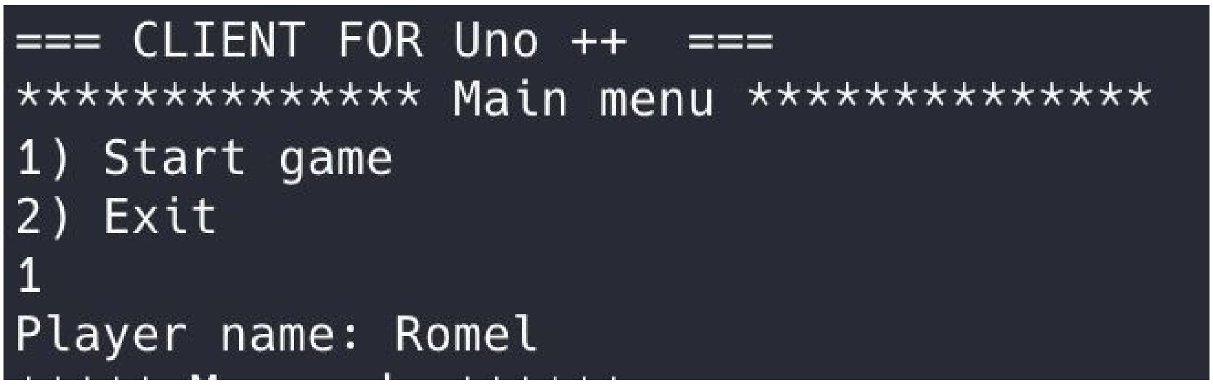
All the plays and the validation messages are shown in the terminal, the graphics are just a visual representation of the current game state.

1) Run the ./client {server ip} {port number} command.



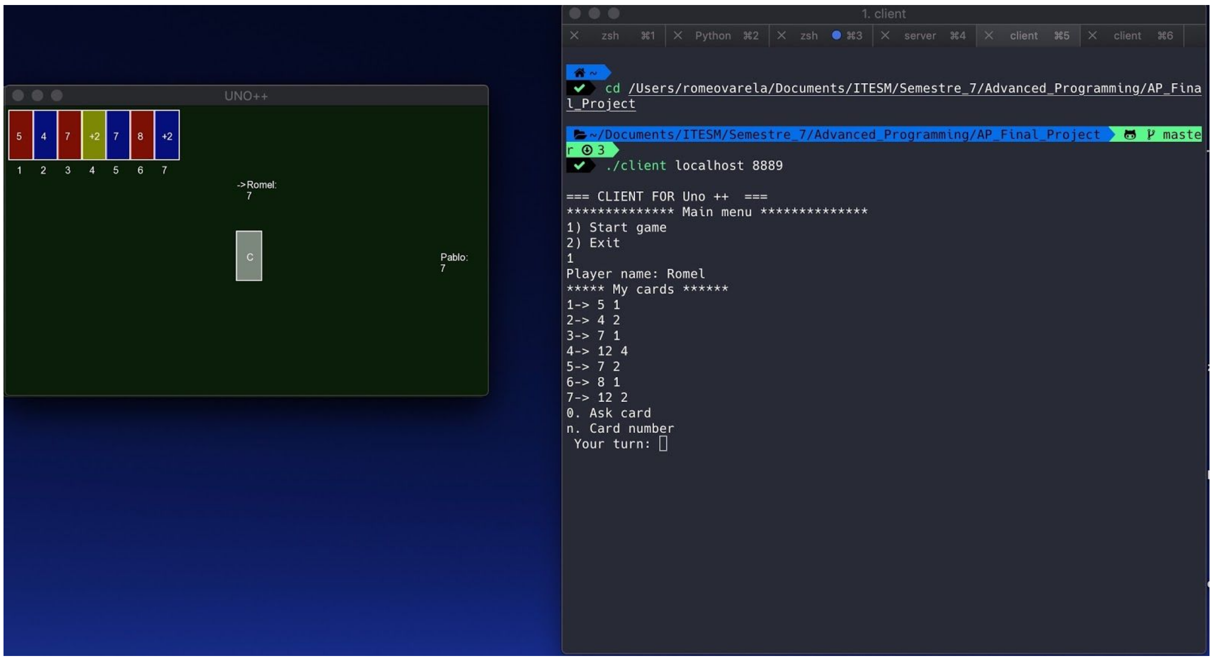
```
~/Documents/ITESM/Semestre_7/Advanced_Programming/AP_Final_Project   master 3
 ./client localhost 8889

=== CLIENT FOR Uno ++  ===
************** Main menu **************
1) Start game
2) Exit
1
```

2) Select the first option to start playing.
3) You will be prompted to enter your name.



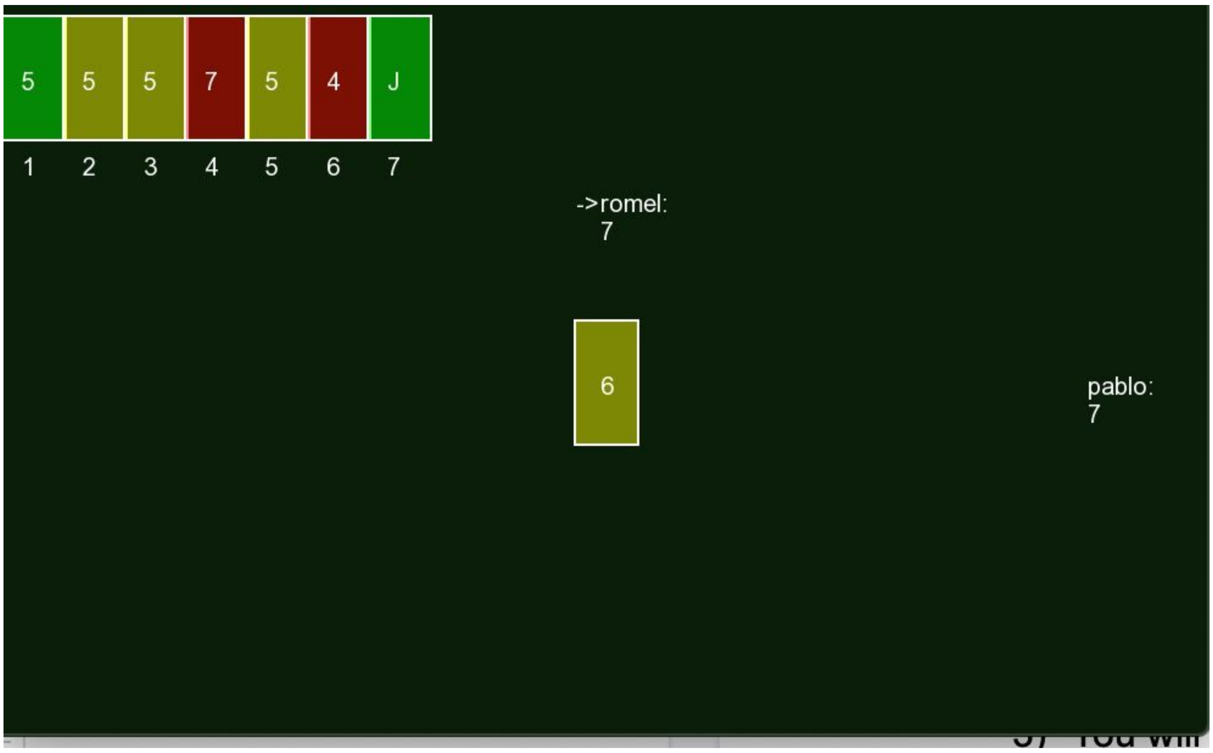
```
=== CLIENT FOR Uno ++  ===
************** Main menu **************
1) Start game
2) Exit
1
Player name: Romel
```

4) Wait for the server to start the game once there are enough players.
5) A window will open showing your cards, the current card in the middle and the names of all the players. Indicating who plays with an arrow next to their name.

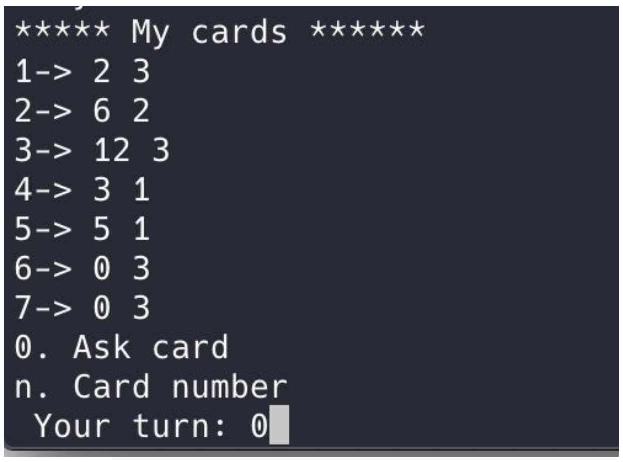


```
~
cd /Users/romeovarela/Documents/ITESM/Semestre_7/Advanced_Programming/AP_Fina
l_Project

~/Documents/ITESM/Semestre_7/Advanced_Programming/AP_Final_Project  master
 3
./client localhost 8889

=== CLIENT FOR Uno ++  ===
************** Main menu **************
1) Start game
2) Exit
1
Player name: Romel
***** My cards ******
1-> 5 1
2-> 4 2
3-> 7 1
4-> 12 4
5-> 7 2
6-> 8 1
7-> 12 2
0. Ask card
n. Card number
 Your turn: 
```

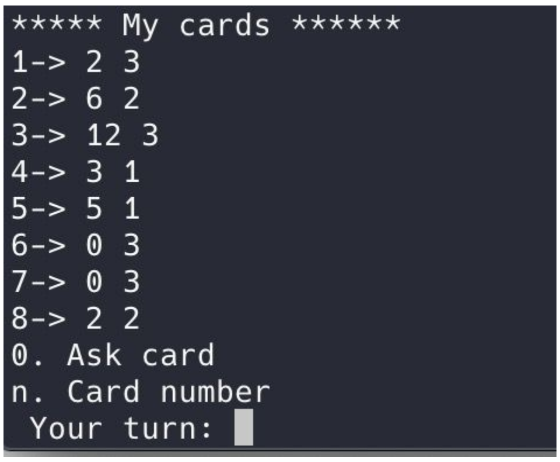
6) Whenever the arrow points to your name, on the terminal, select which card you want to play (all cards have a number below them to show you which one it is)



7) Whenever you are not able to play a card, send the server a 0. It will send a card to your hand and check once again if you are able to play.



```
***** My cards ******
1-> 2 3
2-> 6 2
3-> 12 3
4-> 3 1
5-> 5 1
6-> 0 3
7-> 0 3
0. Ask card
n. Card number
 Your turn: 0
```



```
***** My cards ******
1-> 2 3
2-> 6 2
3-> 12 3
4-> 3 1
5-> 5 1
6-> 0 3
7-> 0 3
8-> 2 2
0. Ask card
n. Card number
 Your turn: 
```

References

https://www.sfml-dev.org/learn.php
http://www.cplusplus.com/reference/
https://en.wikipedia.org/wiki/Uno_(card_game)