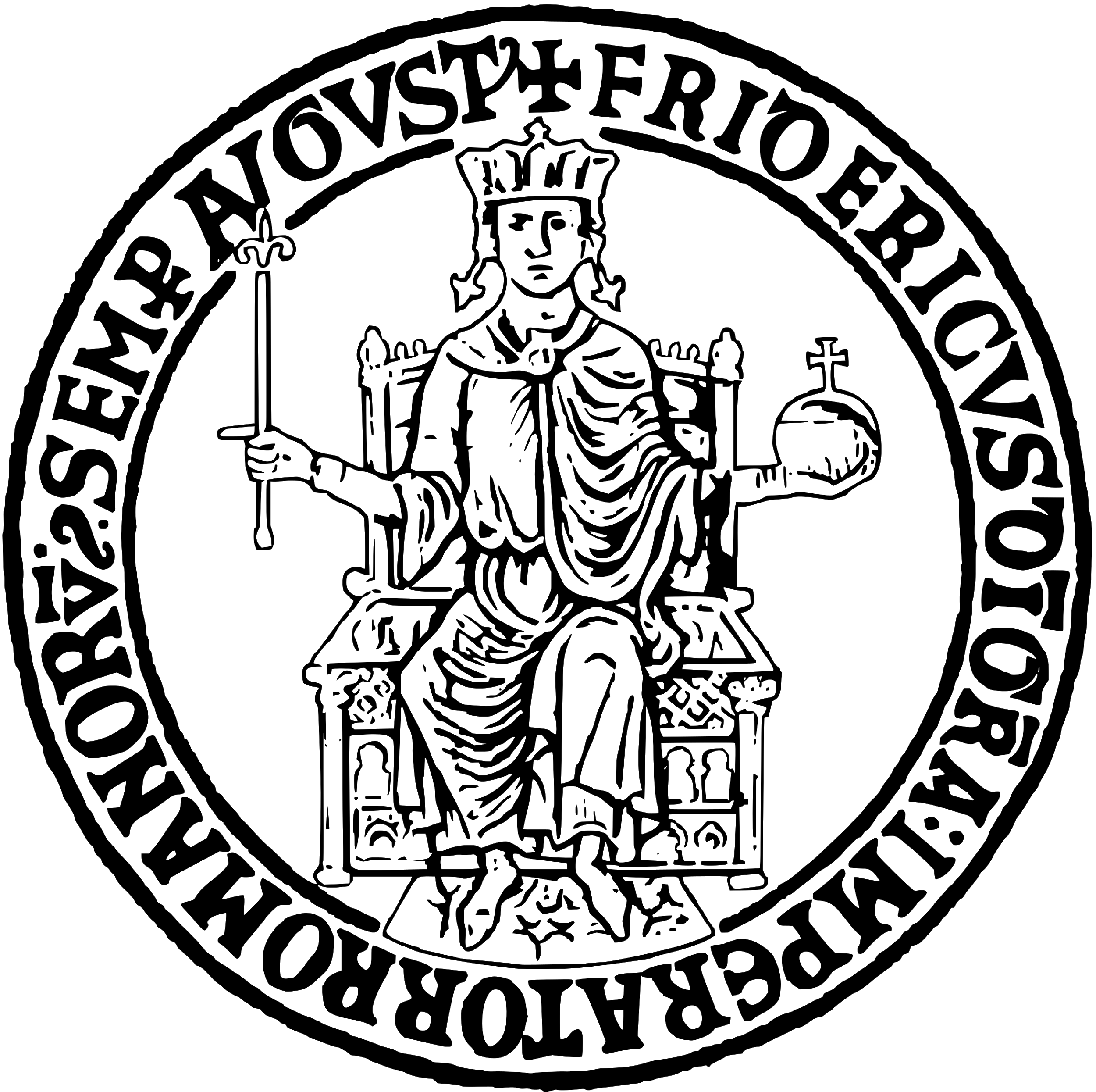
UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL’INFORMAZIONE



Sunto

Il sistema consentirà a più client di partecipare a delle sessioni di gioco collegandosi ad un server.

Docenti

Alberto Finzi

Carmine Testa Dmytro Lozyak

N86002676 N86002756

Progettazione e sviluppo di un sistema su piattaforma unix

Corso di laurea in Informatica Anno Accademico 2020/2021

*Indice*

1. **DESCRIZIONE PROGETTO** Pag. 2
   1. Introduzione Pag. 2
   2. Descrizione dettagliata Pag. 2
2. **MANUALE D’USO**  Pag. 3
   1. Introduzione al manuale d’uso Pag. 3
   2. Compilazione ed esecuzione Pag. 3
      1. Compilazione client Pag. 3
      2. Compilazione server Pag. 3
      3. Esecuzione client Pag. 3
      4. Esecuzione server Pag. 3
   3. Schermata di benvenuto Pag. 4
   4. Login e registrazione Pag. 5
      1. Schermata registrazione e login Pag. 5
   5. Chiusura programma Pag. 6
   6. Schermata di gioco Pag. 6
   7. Movimenti di gioco e conquiste Pag. 7
   8. Terminazione partita Pag. 8
   9. Comunicazione client-server Pag. 9
   10. Considerazione finali Pag. 10

*Capitolo 1*

Descrizione progetto

## **Introduzione**

Realizzare un sistema client-server che consenta a più utenti di giocare ad un gioco di conquista di territori. Si utilizzi il linguaggio C su piattaforma UNIX. I processi dovranno comunicare tramite socket TCP.

## **1.2 Descrizione dettagliata**

Il server manterrà una rappresentazione dell’ambiente che rappresenta un insieme di territori da conquistare. L’ambiente sia rappresentato da una matrice in cui gli utenti si potranno spostare di un passo alla volta nelle quattro direzioni: S, N, E, O. Le caselle saranno libere o di proprietà di un utente. Ogni utente, una volta connesso al server, potrà partecipare alla conquista; il server comunicherà all’utente il punto di partenza con coordinata (x,y) e il numero di territori da conquistare (stabilito dal server ed uguale per tutti i giocatori). La locazione di partenza sarà anche la prima conquista dell’utente. Dopo ogni passo l’utente riceverà l’informazione sull’effetto proprio movimento: se lo spostamento porterà su di una locazione libera questa diventerà di proprietà dell’utente; se la locazione di arrivo appartiene già ad un altro utente sarà possibile conquistarla con un lancio di dadi simulato (2 estrazione di numeri random da 1 a 6, uno per l’attacco l’altro per la difesa, in caso di pareggio vince la difesa). Quando un utente avrà conquistato il numero stabilito di territori o alla scadenza di un limite di tempo fissato, il server notificherà agli utenti la fine della sessione e il vincitore per poi generare una nuova sessione. Se la sessione scade prima della conquista del numero predefinito di territori il vincitore sarà colui che ne ha conquistati di più. Per accedere al servizio ogni utente dovrà prima registrarsi al sito indicando password e nickname. Non c'è un limite a priori al numero di utenti che si possono collegare con il server. Il client consentirà all'utente di collegarsi ad un server di comunicazione, indicando tramite riga di comando il nome o l'indirizzo IP di tale server e la porta da utilizzare. Una volta collegato ad un server l'utente potrà: registrarsi come nuovo utente o accedere al servizio come utente registrato. Il servizio permetterà all’utente di: spostarsi di una posizione, vedere la lista degli utenti collegati, vedere i territori degli altri utenti in gioco, vedere il tempo mancante, disconnettersi. Il server dovrà supportare tutte le funzionalità descritte nella sezione relativa al client. All'avvio del server, sarà possibile specificare tramite riga di comando la porta TCP sulla quale mettersi in ascolto. Il server sarà di tipo concorrente, ovvero in grado di servire più client simultaneamente. Durante il suo regolare funzionamento, il server effettuerà il logging delle attività principali in un file apposito. Ad esempio, memorizzando la data e l'ora di connessione dei client e il loro nome simbolico (se disponibile, altrimenti l'indirizzo IP) e la data e l'ora delle conquiste.

*Capitolo 2*

Manuale d’uso

## **2.1 Introduzione al manuale d’uso**

In questo capitolo verranno mostrati gli scenari tipici durante le varie fasi di gioco e verranno successivamente spiegate per garantire il corretto utilizzo del sistema. Verrà inoltre indicata la sintassi corretta per compilare ed eseguire il codice sorgente.

## **2.2 Compilazione ed esecuzione**

Di seguito è riportata la sintassi per compilare

# **2.4.1 Compilazione client**

gcc-o client client.c checkInput.h checkInput.c lib/inputReader.h lib/inputReader.c

# **2.4.1 Compilazione server**

gcc -o server server.c file.h file.c listUser/array.h listUser/array.c listUser/list.h listUser/list.c checkInput.h checkInput.c lib/inputReader.h lib/inputReader.c -lpthread

# **2.4.1 Esecuzione client**

./client 127.0.0.1 18000

# **2.4.1 Esecuzione server**

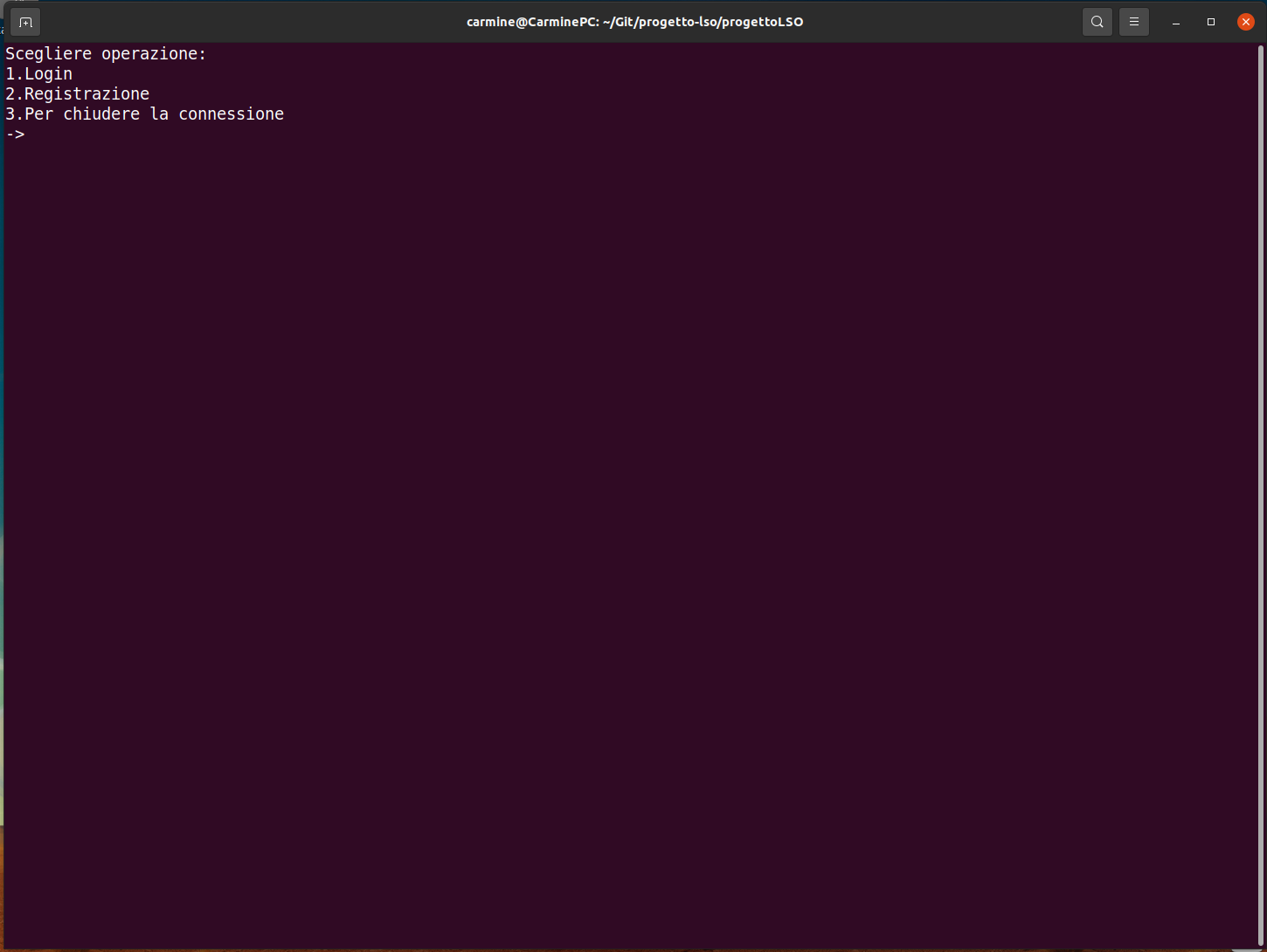
./server 18000

*Capitolo 2 Manuale d’uso*

# **2.3** **Schermata di benvenuto**

Questa è la prima schermata che compare dal lato client una volta avviata la connessione col server. In questa schermata verranno mostrate le 3 azione che l’utente potrà fare e cioè avrà la possibilità di:

1. Effettuare il login
2. Effettuare la registrazione
3. Chiudere la connessione



**Nota**

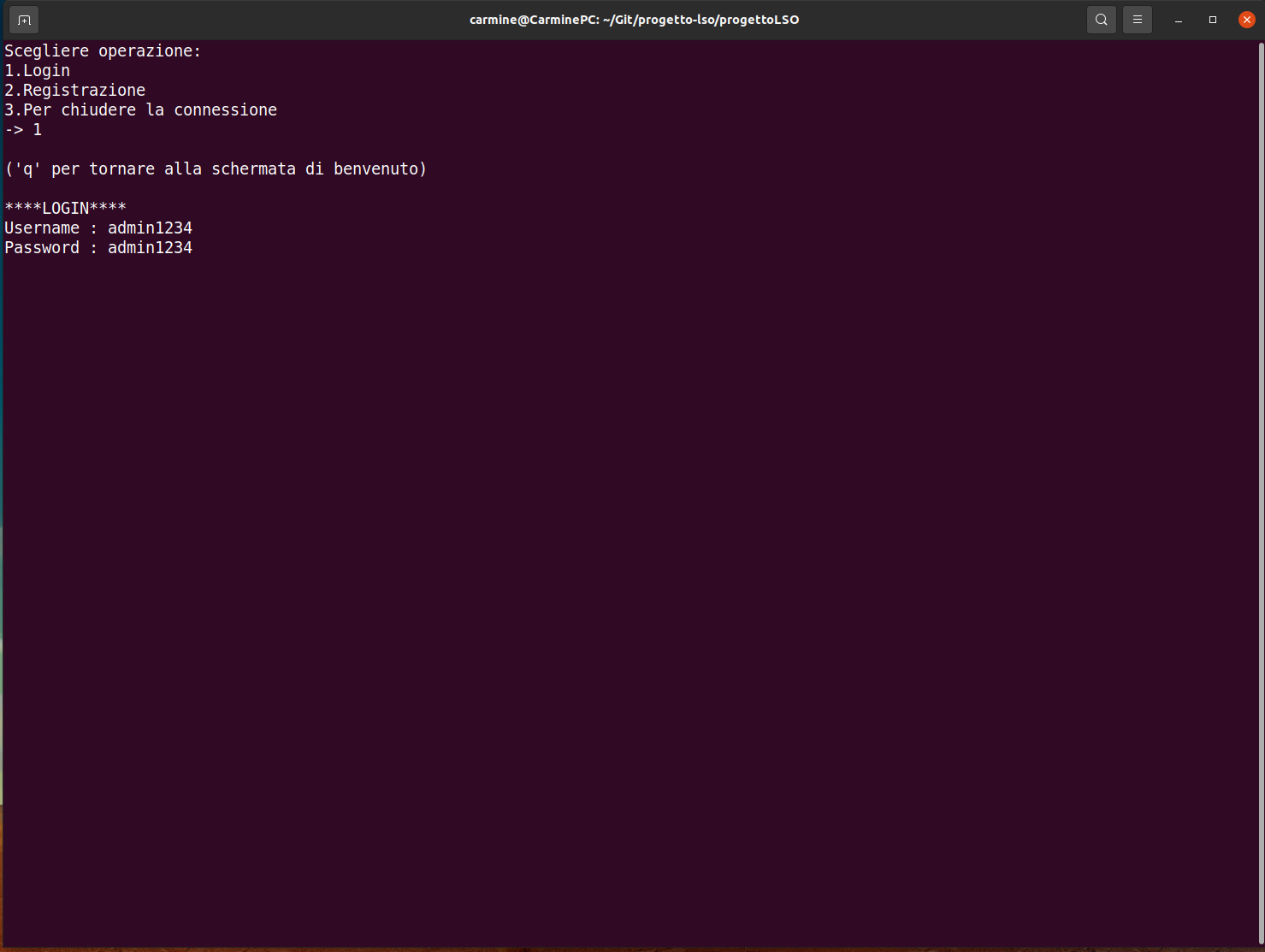
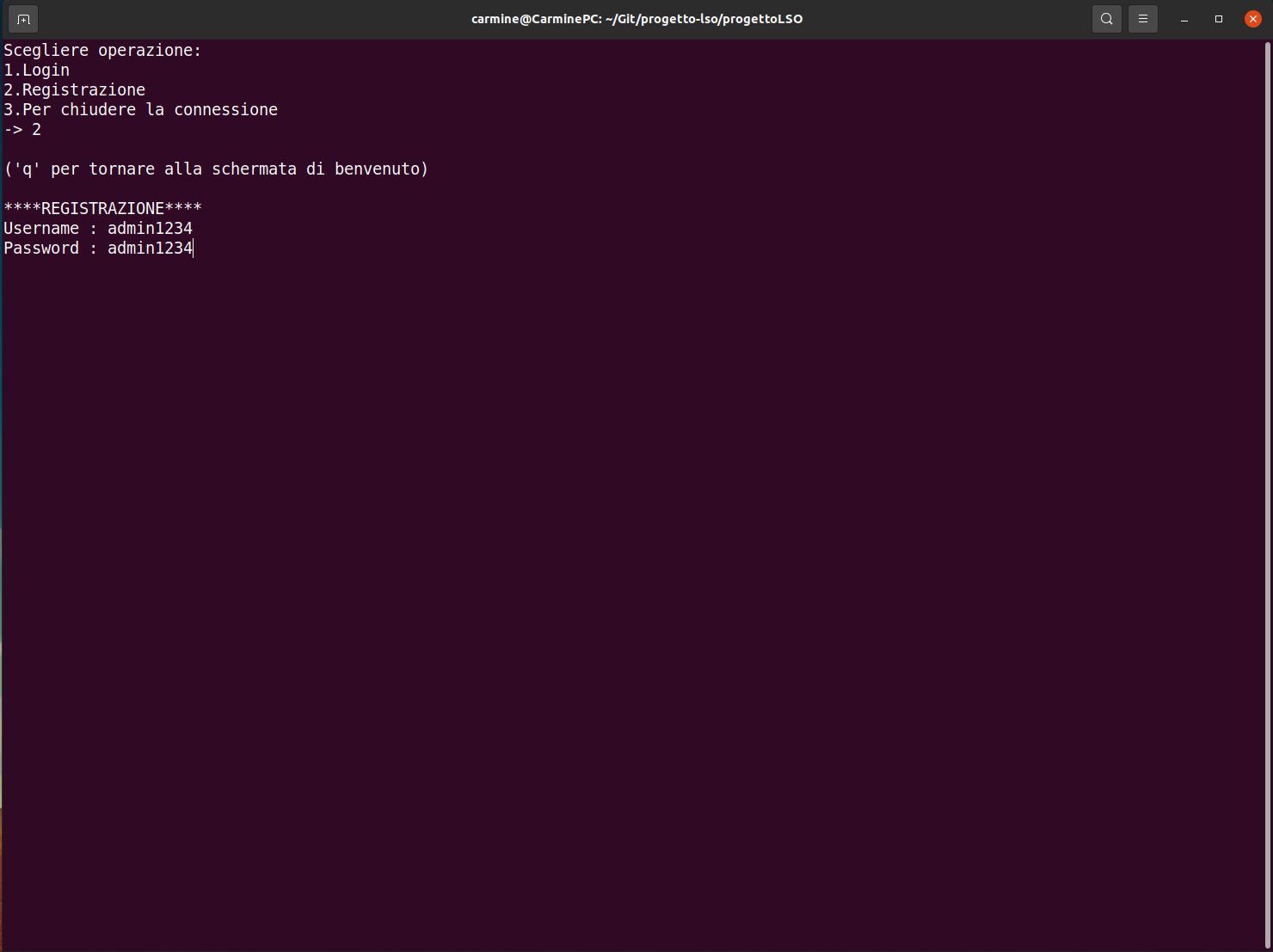
Se viene inviata una qualsiasi scelta diversa da 1 2 o 3 il sistema chiederà all’utente di reinserire una scelta corretta.

*Capitolo 2 Manuale d’uso*

# **2.4 Login e registrazione**

In questo paragrafo verranno mostrate le schermate di login e registrazione, per comodità verranno mostrate insieme, e poiché queste ultime sono molto simili tra di loro se non per qualche minima differenza, non ci saranno perdite di informazioni. Username dev’essere di almeno 4 caratteri mentre la password minimo 6.

# **2.4.1 Schermata registrazione/login**

Durante la schermata di login o di registrazione, in qualsiasi momento con l’invio del carattere q è possibile tornare alla schermata di benvenuto per annullare l’operazione in corso. 

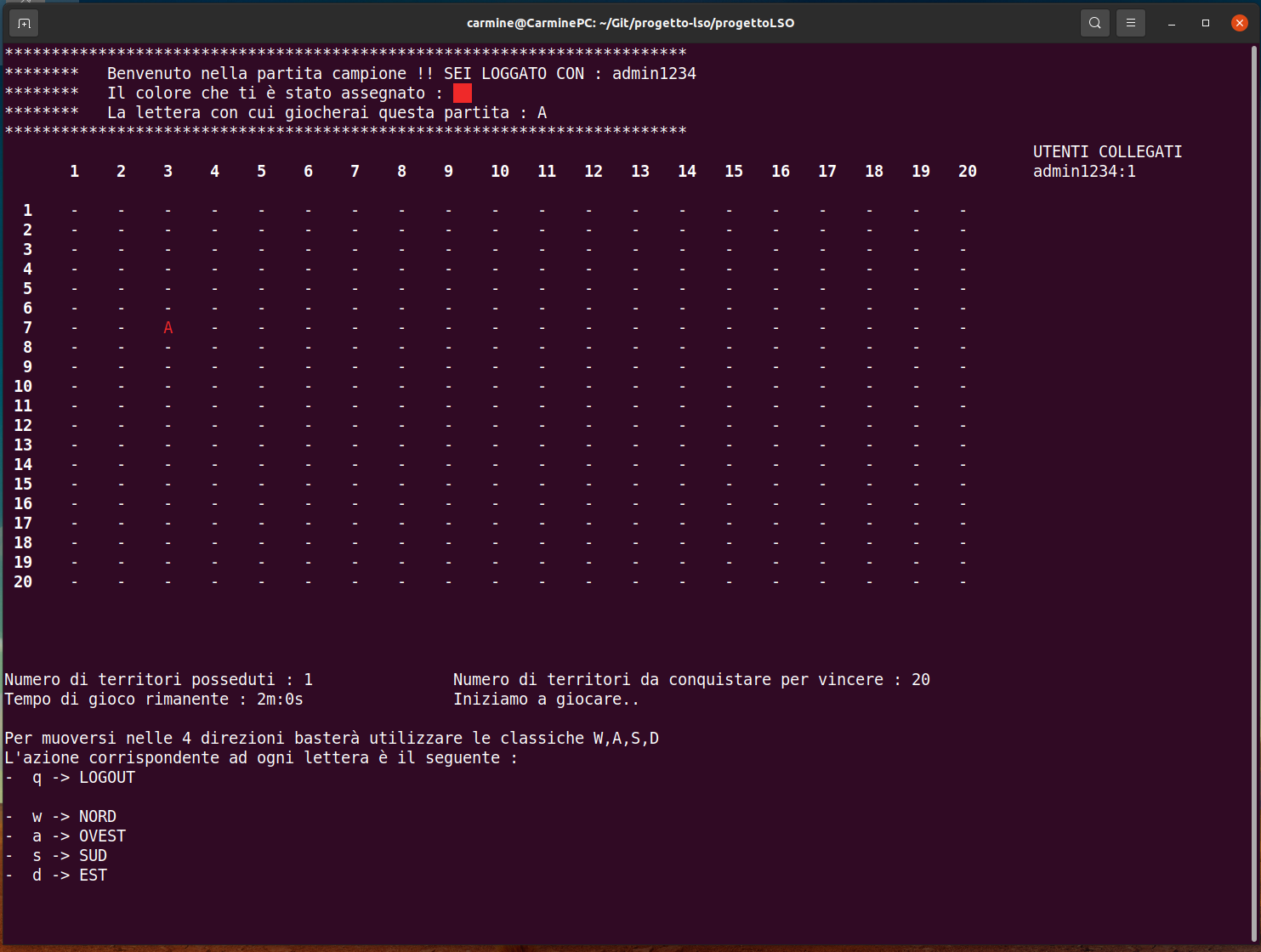
*Capitolo 2 Manuale d’uso*

# **2.5 Chiusura programma**

Con la scelta dell’opzione 3 nella schermata di benvenuto, verrà chiusa la connessione con il server e terminato il programma.

# **2.6 Schermata di gioco**

Effettuato il login si parteciperà alla partita, ad ogni giocatore verrà assegnata una lettera e un colore. I territori conquistati saranno colorati con il proprio colore. Al lato è mostrata una lista con gli utenti collegati alla partita e rispettivi territori conquistati. Durante qualsiasi fase di gioco sarà possibile tramite il tasto ‘q’ effettuare il logout.

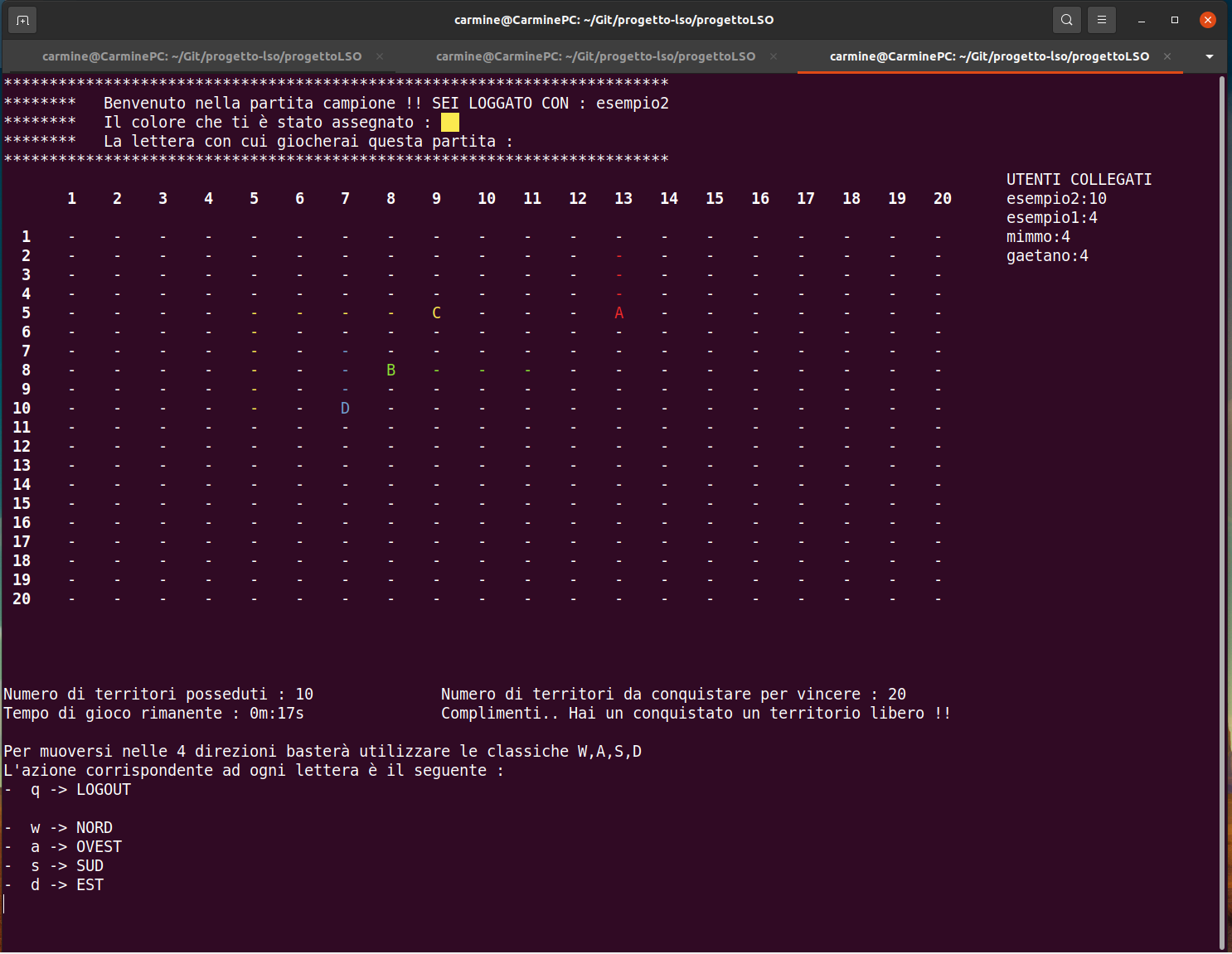


*Capitolo 2 Manuale d’uso*

# **2.7 Movimenti di gioco e conquiste**

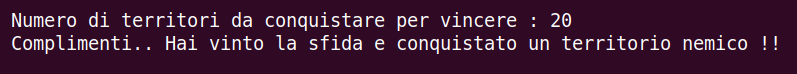
Per effettuare qualsiasi movimento nelle 4 direzioni sarà sufficiente inviare i caratteri “w,a,s,d” sul terminale. La conquista di un territorio avviene muovendosi al di sopra di esso e possono verificarsi due cose in base al tipo di territorio da conquistare:

1. Se il territorio in questione è “Libero” quindi di nessun proprietario. Una volta sopra al territorio quest’ultimo sarà immediatamente contrassegnato come di proprietà del giocatore che ci si trova sopra.
2. Se invece il territorio è di proprietà di qualcun altro… in quest’occasione bisogna lottare. La lotta avviene tramite un lancio di dadi simulato dal server e nel caso di successo verrà consentita la mossa con un messaggio che indichi il successo della conquista altrimenti la mossa verrà negata con un messaggio di notifica della vittoria della difesa.

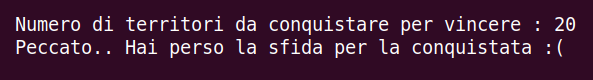


*Capitolo 2 Manuale d’uso*

Il messaggio mostrato per indicare la vittoria della lotta e di conseguenza la conquista di un territorio è il seguente.



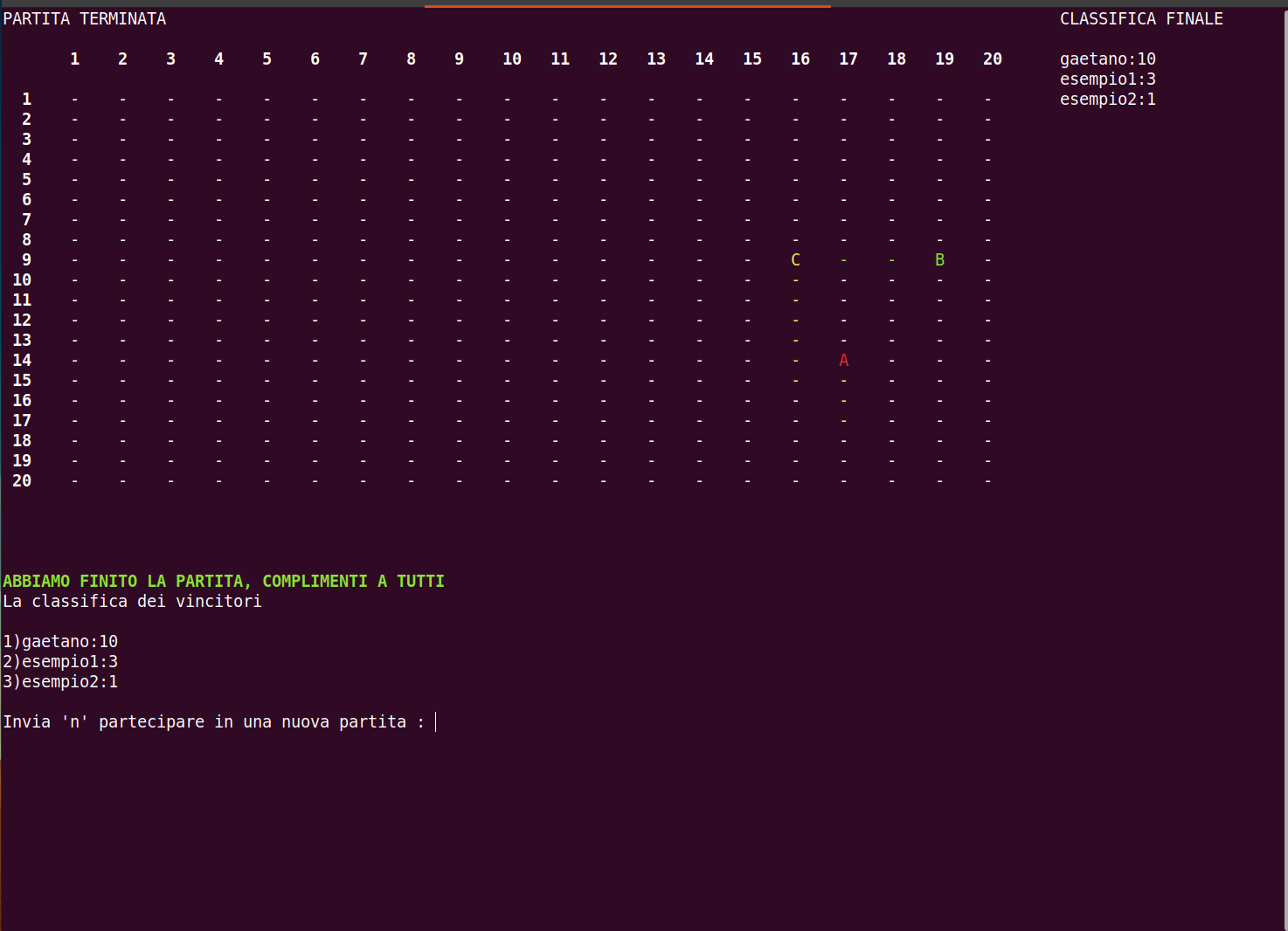
In caso di sconfitta durante la conquista di un territorio verrà mostrato questo messaggio.



# **2.8 Terminazione partita**

Dopo una qualsiasi mossa è possibile che la sessione di gioco termini. Potrebbe terminare perché qualche utente ha raggiunto il numero territori necessari per vincere oppure perché tempo è scaduto.

In entrambi i casi all’utente verrà visualizzata la schermata di fine partita in cui verrà proposta la classifica dei vincitori. Con n sarà possibile partecipare ad una nuova partita.



*Capitolo 2 Manuale d’uso*

# **2.9 Comunicazione client-server**

Per consentire una corretta comunicazione tra client e server si è scelto di adottare una strategia di comunicazione ovvero, tramite dei micro-messaggi di conferma. Il client e il server quando inviano un messaggio si mettono in stato di read, aspettando che la controparte gli invii la conferma di ricezione del messaggio.

# **2.10 Considerazioni finali**

Immagine che contiene testo

Descrizione generata automaticamente

Una cosa interessante e piacevole è stata la possibilità   
di utilizzare chiamate di sistema all’interno del   
codice C. In questo modo (come si può osservare  
dalla figura al lato) mediante l’utilizzo della chiamata  
di sistema è stato facile controllare se l’utente   
attualmente è già registrato. Questo è stato possibile  
grazie anche all’utilizzo della fork con conseguente  
creazione di un processo figlio e combinata con la   
funzione pipe che permette la comunicazione tra  
processo padre e processo figlio.   
Da notare anche il reindirizzamento dell’output   
della funzione “grep” sulla pipe, consentendo,  
al processo padre di leggere il risultato conseguito  
della grep.

*Capitolo 2 Manuale d’uso*

In questo frammento di codice abbiamo gestito il thread che si occupa ad ogni inizio partita di contare il tempo di gioco. Si può osservare l’utilizzo della condition variable con la chiamata di sistema “pthread\_mutex\_wait” che si occupa di mettere il thread in uno stato di attesa fino a quando la partita non è cominciata. Tramite l’attributo “partitaInCorso” conosciamo lo stato della partita attuale, con il valore 1 sappiamo che è iniziata la partita. Scaduto il tempo il thread si occuperà di impostare lo stato di gioco della partita a 2 (Partita terminata) per poi rilasciare il mutex.

Immagine che contiene testo

Descrizione generata automaticamente

//CLIENT.C

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <netdb.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <signal.h>

#include <unistd.h>

#include <errno.h>

#include <arpa/inet.h>

#include <ctype.h>

#include "checkInput.h"

//FUNZIONI LOGIN-REGISTRAZIONE-BENVENUTO

int gestioneBenvenuto(int , int );

int gestioneUsernamePassowrd(int );

//FUNZIONI PARTITA

int startGame(int);

int update(int , char [20][20] , char [26][100] , char\* , int\*);

int recuperaScelta(char\* );

//FUNZIONI INIZIALIZZAZIONI VARIABILI

int initGame(int,char[20][20] , int\*, char\* , char [26][100] , char\*, int\*);

int initMappa(int , char[20][20]);

int initTimer(int , char\*);

int initListUsers(int , char[26][100]);

int initUserColor(int , int\*);

int initUserChar(int, char\*);

int initTerritoriNecessari(int , int\*);

void initColoriLettere();

int initTerritoriPosseduti(int , int\* );

//FUNZIONI DI STAMPA

void stampaVincitore(char [26][100]);

void stampaFraseFinale(int);

void stampaMappa(char [20][20],int,char,char[26][100]);

void stampaLegenda(int , char );

void stampaStatistichePartita(int , char [],int,char\*);

void stampaScelte();

//UTILITY

int conferma(int);

int coloraCarattere(int, char);

char sostituisciCasella(int , int , char[20][20], char);

//VAR GLOBALI

int arrayColori[26];

char arrayCaratteri[26];

char usernameGlobale [30];

//Per compilare...

//gcc -o client client.c checkInput.h checkInput.c lib/inputReader.h lib/inputReader.c

//./client 79.42.150.62 18000

int main(int argc, char \*argv[]){

int sd\_client = 0, n = 0, m = 0, retBenvenuto = 0 , retGame = 0;

int scelta;

char recvBuff[1024] ;

struct sockaddr\_in server\_addr;

char \*ip\_adress;

int porta\_ingresso;

char msgBenvenuto[] = "Scegliere operazione:\n1.Login\n2.Registrazione\n3.Per chiudere la connessione\n\0";

ssize\_t lenBenvenuto = strlen(msgBenvenuto);

if (argc != 3){

write(STDERR\_FILENO, "Numero insufficiente di parametri\n", 34);

exit(-1);

}

signal(SIGINT,SIG\_IGN);

ip\_adress = argv[1]; //INIZIALIZZO L'IP A CUI CONNETTERMI

porta\_ingresso = atoi(argv[2]); //INIZIALIZZO LA PORTA A CUI CONNETTERMI

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(porta\_ingresso);

inet\_aton(ip\_adress, &server\_addr.sin\_addr);

sd\_client = socket(AF\_INET, SOCK\_STREAM, 0);

if (sd\_client < 0)

perror("socket"), exit(-1);

if (connect(sd\_client, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0){

perror("connect"), exit(-1);

}

system("clear");

do{

retGame = 0 ;

retBenvenuto = 0;

//STAMPO SU CONSOLE IL MESSAGIO DI BENVENUTO

if ((n = write(STDOUT\_FILENO, msgBenvenuto, lenBenvenuto)) < 0){

perror("write benvenuto");

return -1;

}

//LEGGO DA STDIN ED INVIO LA PRIMA RISPOSTA AL SERVER

/\*\*

\* 1 : LOGIN

\* 2 : REGISTRAZIONE

\* 3 : CHIUDERE PROGRAMMA

\*/

//LEGGO LA PRIMA RISPOSTA

scelta = doSceltaIntError("-> ", 3, "Errore scelta argomenti riprovare...\n");

//CONVERTO LA PRIMA RISPOSTA IN UNA STRINGA

sprintf(recvBuff, "%d", scelta);

//INVIO LA PRIMA RISPOSTA AL SERVER

if ((n = write(sd\_client, recvBuff, 1)) < 0){

perror("write prima rispostaServer");

return -1;

}

retBenvenuto = gestioneBenvenuto(scelta, sd\_client);

if(retBenvenuto == -1){

close(sd\_client);

return -1 ;

}

if(retBenvenuto == 1){//LOGIN OK

retGame = startGame(sd\_client);

}

if(retGame == -1){

close(sd\_client);

return -1 ;

}

} while (retGame == 2 || retBenvenuto == 2);//TORNO ALLA SCHERMATA DI BENVENUTO

close(sd\_client);

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FUNZIONI LOGIN-REGISTRAZIONI-BENVENUTO \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*SI OCCUPA DELLA GESTIONE DEL BENVENUTO DOPO CHE L'UTENTE HA EFFETTUATO UNA SCELTA CON IL

\* PARAMETRO SCELTA DOVE 1 INDICA IL LOGIN 2 INDICA LA REGISTRAZIONE E 3 INDICA LA CHIUSURA

\* DEL PROGRAMMA. LA FUNZIONE RITORNA 1 SE IL LOGIN E' ANDATO A BUON FINE, 2 NEL CASO SI

\* VOGLIA RIPETERE LA SCHERMATA DI BENVENUTO, 3 SE SI VUOLE CHIUDERE IL PROGRAMMA E -1 IN CASO

\* DI ERRORE

\*/

int gestioneBenvenuto(int scelta, int sd\_client){

int esito = 0;

switch (scelta){

case 1: //LOGIN

fprintf(stdout,"\n('q' per tornare alla schermata di benvenuto)\n\n");

fprintf(stdout,"\*\*\*\*LOGIN\*\*\*\*\n");

do{

esito = gestioneUsernamePassowrd(sd\_client);

switch (esito){

case 0:

fprintf(stdout, "Errore Nome utente o Password non corretti\n\n");//RIPETO IL LOGIN

break;

case 1 :

fprintf(stdout, "Login OK\n\n");

return 1;

case 2 :

return 2; //TORNO ALLA SCHERMATA DI BENVENUTO

case 3 :

fprintf(stdout, "Utente già loggato :)\n\n");//RIPETO IL LOGIN

break ;

default:

fprintf(stdout, "Errore Login rispostaServer : %d\n",esito);//ESCO

return -1;

}

} while (esito == 0 || esito == 3);

break;

case 2: //REGISTRAZIONE

fprintf(stdout,"\n('q' per tornare alla schermata di benvenuto)\n\n");

fprintf(stdout,"\*\*\*\*REGISTRAZIONE\*\*\*\*\n");

do{

esito = gestioneUsernamePassowrd(sd\_client);

switch (esito){

case 0:

fprintf(stdout, "Errore nome utente già registrato\n\n");//RIPETO LA REGISTRAZIONE

break;

case 1 :

system("clear");

fprintf(stdout, "Registrazione OK\n\n");

return 2; //TORNO ALLA SCHERMATA DI BENVENUTO

case 2 :

return 2; //TORNO ALLA SCHERMATA DI BENVENTUO

default:

fprintf(stdout, "Errore registrazione rispostaServer : %d\n",esito);//ESCO

return -1;

}

} while (esito == 0);

break;

case 3 : //CHIUSURA PROGRAMMA

return 3;

default :

fprintf(stdout,"Terminazione di default\n");

return -1 ;

}

return 1;

}

/\*\*

\* GESTIONE DEL RECUPERO DATI NEL CASO DI UN LOGIN O UNA REGISTRAZIONE

\* USERNAME MINIMO 4 CARATTERI MENTRE LA PASSWORD 6 CARATTERI

\* RITORNA

\* -1 IN CASO DI ERRORE

\* 2 PER TORNARE ALLA SCHERMATA DI BENVENUTO (CON 'q' SI PUO TORNARE ALLA SCHERMATA DI BENVENUTO)

\* 0 PER DATI ERRATI

\* OPPURE UNA RISPOSTA DAL SERVER

\*/

int gestioneUsernamePassowrd(int sd\_client){

int n = 0;

int m = 0;

char username[50];

char password[50];

char quit[] = "q";

char localBuf[256];

int rispostaServer = 0;

//CONTROLLO DELL'USERNAME DEV'ESSERE 4 CARATTERI

do{

write(STDOUT\_FILENO, "Username : ", 11);

if ((n = read(STDIN\_FILENO, username, 36)) < 0){

perror("read username");

return -1;

}

username[n-1]='\0';//SOSTITUISCO LO \N CON \0

if(n < 5 || strcmp(quit,username) == 0){

if(strcmp(quit,username) == 0){

write(sd\_client,"q\0",2); //INVIO AL SERVER LA Q PREMUTA

system("clear"); //PULISCO LA SCHERMATA

printf("Rilevato 'q' , torno al benvenuto\n\n");

return 2; //TORNO ALLA SCHERMATA DI BENVENUTO

}

else

fprintf(stdout,"Errore nome utente troppo corto... \n");

}

}while (n<5);

//CONTROLLO DELLA PASSWORD DEV'ESSERE 6 CARATTERI

do{

write(STDOUT\_FILENO, "Password : ", 11);

if ((m = read(STDIN\_FILENO, password, 36)) < 0){

perror("read password");

return -1;

}

password[m-1] = '\0'; //SOSTITUISCO LO \N CON \0

if(m < 7 || strcmp(quit,password) == 0){

if(strcmp(quit,password) == 0){

write(sd\_client,"q\0",2); //INVIO AL SERVER LA Q PREMUTA

system("clear"); //PULISCO LA SCHERMATA

printf("Rilevato 'q' , torno al benvenuto\n\n");

return 2; //TORNO ALLA SCHERMATA DI BENVENUTO

}

else

fprintf(stdout,"Errore password troppo corta... \n");

}

}while (m < 7);

//INVIO USERNAME AL SERVER

if (write(sd\_client, username, n) < 0){

perror("write username");

return -1;

}

strcpy(usernameGlobale,username);

//LEGGO LA CONFERMA DI RICEZIONE DEL NOME UTENTE DA PARTE DEL SERVER

if (read(sd\_client, localBuf, 128) < 0){

perror("Read rispostaServer al login/registrazione username ricevuto");

return -1;

}

//INVIO LA PASSWORD AL SERVER

if (write(sd\_client, password, m) < 0){

perror("write password");

return -1;

}

//LEGGO LA RISPOSTA DEL SERVER DOPO L'INVIO DEI DATI

/\*\*

\* 0 : UTENTE GIÀ REGISTRATO OPPURE DATI INCORRETTI

\* 1 : LOGIN O REGISTRAZIONI ANDATE A BUON FINE

\* -1 : ERRORE DI COMUNICAZIONE

\* 3 : UTENTE GIÀ LOGGATO

\*/

//LEGGO LA RISPOSTA

if ((n = read(sd\_client, localBuf, 128)) < 0){

perror("Read rispostaServer al login/registrazione");

return -1;

}

rispostaServer = atoi(localBuf);

return rispostaServer;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FUNZIONI PARTITA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*RITORNA

\* 2 PER INDICARE CHE SI VUOLE TORNARE ALLA SCHERMATA DI BENVENUTO

\* -1 PER UN ERRORE

\* 1 TERMINAZIONE OK MA NON PREVISTA

\*/

int startGame(int sd\_client){

char \* letteraUtente = malloc(sizeof(char)); //LETTERA UTENTE

int \* coloreUtente = malloc(sizeof(int)); //COLORE UTENTE

char mappa[20][20]; //MAPPA DI GIOCO

char \* scelta = malloc(sizeof(char)); //MOVIMENTO SCELTO DALL' UTENTE

int \*territoriPosseduti = malloc(sizeof(int)); //N TERRITORI POSSEDUTI

int \*territoriDaConquistare = malloc(sizeof(int)); //N TERRITORI PER VINCERE

int rispostaServer = 0 ; //FLAG RISPOSTA SERVER

int retInitGame = 0 , retRecuperoScelta = 0 ; //VAR DI CONTROLLO LOCALE

char buf[128]; //BUFFER LOCALE

char userUtenti [26][100]; //LISTA UTENTI IN GIOCO

char \* time = malloc(sizeof(char) \* 128); //TEMPO DI GIOCO RIMANENTE

char \* conseguenzaUltimaAzione = malloc(sizeof(char) \* 128); //DESCRIZIONE DELLE CONSEGUENZE LEGATE ALL'ULTIMA AZIONE

initColoriLettere(); //INIZIALIZZO COLORI E LETTERE DEGLI AVVERSARI

system("clear");

do{

\*territoriPosseduti = 1 ; //AD INIZIO PARTITA HAI 1 TERRITORIO

retInitGame = initGame(sd\_client,mappa,coloreUtente,letteraUtente,userUtenti,time,territoriDaConquistare);

strcpy(conseguenzaUltimaAzione,"Iniziamo a giocare..");

if(retInitGame<0){

fprintf(stdout,"Errore initGame\n");

return -1;

}

else if(retInitGame == 2){//IMPOSSIBILE ACCEDERE.. FINITE LE LETTERE DISPONIBILI

return 2 ; //TORNO ALLA SCHERMATA DI BENVENUTO

}

else if(retInitGame == 1){//INIZIALIZZAZIONE OK

do{

//SCHERMATA DI GIOCO

stampaLegenda(\*coloreUtente,\*letteraUtente);

stampaMappa(mappa,\*coloreUtente,\*letteraUtente,userUtenti);

stampaStatistichePartita(\*territoriPosseduti,time,\*territoriDaConquistare,conseguenzaUltimaAzione);

stampaScelte();

retRecuperoScelta = recuperaScelta(scelta); //PRENDO LA SCELTA DELL'UTENTE E LA SALVO IN SCELTA

switch (retRecuperoScelta){

case 1://LETTERRA INSERITA CORRETTAMENTE

if(strcmp(scelta,"q") == 0){//RILEVATO Q FACCIO LOGOUT

if(write(sd\_client,scelta,2)<=0){

perror("Errore write scelta 'q'");

return -1;

}

printf("Rilevato q\n\n");

system("clear");

return 2; //TORNO ALLA SCHERMATA DI BENVENUTO

}

else{

//INVIA AL SERVER LA RISPOSTA E AGGIORNA LA POSIZIONE

if(write(sd\_client,scelta,2)<=0){//INVIO IL MOVIMENTO

perror("Errore write movimento");

return -1;

}

if(read(sd\_client,buf,128)<=0){//LEGGO IL FLAG DEL SERVER

perror("Errore read rispostaServer al movimento");

return -1;

}

conferma(sd\_client);

rispostaServer = atoi(buf);

system("clear");

update(sd\_client,mappa,userUtenti,time,territoriPosseduti);

switch (rispostaServer){

case 0: //movimento bloccato perchè fuori mappa o altro utente già occupa

strcpy(conseguenzaUltimaAzione, "Movimento bloccato, out of range o scontro con altro player");

break;

case 1: //movimento accettato su territorio di nessuno, posseduti +1

strcpy(conseguenzaUltimaAzione, "Complimenti.. Hai un conquistato un territorio libero !!");

break;

case 2: //movimento sul territorio nemico e vinto la conquista

strcpy(conseguenzaUltimaAzione, "Complimenti.. Hai vinto la sfida e conquistato un territorio nemico !!");

break;

case 3: //tentativo di conquista senza successo, rimani sul posto

strcpy(conseguenzaUltimaAzione, "Peccato.. Hai perso la sfida per la conquistata :(");

break;

case 4: //movimento sul proprio territorio

strcpy(conseguenzaUltimaAzione, "Movimento sul proprio territorio");

break;

case 5: //fine partita per tempo o arrivo ai territori necessari

//conferma(sd\_client);

//read(sd\_client,buf,128);//leggo il vincitore

fprintf(stdout, "PARTITA TERMINATA%93s%s\n\n", "", "CLASSIFICA FINALE");

stampaMappa(mappa, \*coloreUtente, \*letteraUtente, userUtenti);

stampaVincitore(userUtenti);

stampaFraseFinale(sd\_client);

system("clear");

retRecuperoScelta = 2;//RICOMINCIO UNA NUOVA PARTITA

break;

default:

fprintf(stdout, "Errore, Risposta del server sconosciuta : %d\n", rispostaServer);

return -1;

break;

}//FINE SWITCH RISPOSTA SERVER

}//FINE ELSE CASE 1

break;//FINE CASE 1

case 0: //LETTERA SBAGLIATA RIPETO LA SCELTA

strcpy(conseguenzaUltimaAzione,"Mi sembra una mossa illegale... RIPROVA");

system("clear");

break;

case -1: //ESCO PER QUALCHE ERRORE

return -1;

break;

default:

printf("Errore recupero scelta, ret sconosciuto : %d\n",retInitGame);

return -1;

break;

}

}while (retRecuperoScelta == 0 || retRecuperoScelta == 1);//RIPETO LA MOSSA

}

} while (retRecuperoScelta == 2);//RICOMINCIO LA PARTITA

free(letteraUtente);free(coloreUtente);free(scelta);

free(territoriPosseduti);free(territoriDaConquistare);

free(time);free(conseguenzaUltimaAzione);

return 1 ;

}

//LEGGE DAL SERVER E AGGIORNA LA MAPPA DI GIOCO, LA LISTA UTENTI CONNESSI E IL TEMPO DI GIOCO

/\*\* RESTITUISCE

\* 1 TUTTO OK

\* -1 ERRORE

\*/

int update(int sd\_client, char map[20][20] , char listaUser[26][100] , char\* time , int\* territoriPosseduti){

char localbuf[1024];

if(initMappa(sd\_client,map) == -1)

return -1;

if(conferma(sd\_client) == -1)

return -1;

if(initListUsers(sd\_client,listaUser) == -1)

return -1;

if(conferma(sd\_client) == -1)

return -1;

if(initTimer(sd\_client,time) == -1)

return -1;

if(conferma(sd\_client) == -1)

return -1;

if(initTerritoriPosseduti(sd\_client,territoriPosseduti) == -1)

return -1;

return 1 ;

}

//LEGGE DA STDIN IL MOVIMENTO DELL'UTENTE E RESTITUSCE

/\*\* 0 SE HA INSERITO UN CARATTERE SBAGLIATO

\* 1 SE HA INSERITO UN CARATTERE VALIDO

\* -1 PER UN ERRORE

\* \*/

int recuperaScelta(char\* carattere){

int n = 0 ;

n = read(STDIN\_FILENO,carattere,128);

\*(carattere+n-1) = '\0';

if ( n < 0){

fprintf(stdout,"Errore lettura rispostaServer\n");

return -1; //ERRORE

}

if (strlen(carattere) > 1 || \*carattere != 'w' && \*carattere != 'a' && \*carattere != 's' && \*carattere != 'd' && \*carattere!= 'q' ){

printf("Errore carattere\n");

return 0; //CARATTERE SBAGLIATO

}

return 1 ; //TUTTO OK

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INIZIALIZZAZIONI \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* INIZIALIZZA LE VARIABILI PER INIZIARE UNA NUOVA PARTITA

\* RESTITUSCE

\* 2 PER INDICARE CHE SONO FINITE LE LETTERE E NON È POSSIBILE GIOCARE

\* 1 TUTTO OK

\* -1 ERRORE

\*/

int initGame(int sd\_client,char map[20][20] , int\* colore , char\* lettera , char listaUser[26][100], char\* time , int\* terreDaConquistare){

if(initUserChar(sd\_client,lettera) == -1)

return -1;

if(strcmp(lettera,"0") == 0){

system("clear");

printf("Lettere momentaneamente non disponibili per giocare... Riprovare tra qualche minuto\n\n");

return 2 ; //TORNO ALLA SCHERMATA DI BENVENUTO

}

if(conferma(sd\_client) == -1)

return -1;

if(initUserColor(sd\_client,colore) == -1)

return -1;

if(conferma(sd\_client) == -1)

return -1;

if(initMappa(sd\_client,map) == -1)

return -1;

if(initListUsers(sd\_client,listaUser) == -1)

return -1;

if(conferma(sd\_client) == -1)

return -1;

if(initTimer(sd\_client,time) == -1)

return -1;

if(conferma(sd\_client) == -1)

return -1;

if(initTerritoriNecessari(sd\_client,terreDaConquistare) == -1)

return -1;

return 1;

}

/\*INIZIALIZZA L'ARRAY DEI CARATTERI E DEI COLORI PER STAMPA GLI AVVERSARI COLORATI\*/

void initColoriLettere()

{

int j= 21;

for (int i = 0; i < 26; i++)

{

arrayCaratteri[i] = i + 65;

if (i < 6)

arrayColori[i] = i+9;

else{

arrayColori[i] = j;

j += 10;

}

}

}

/\*INIZIALIZZO LA MAPPA DI GIOCO\*/

int initMappa(int sd\_client, char mappa[20][20]){

char\* localbuf = malloc(sizeof(char)\*512);

int i = 0 , j = 0 ;

if(read(sd\_client,localbuf,400)<=0){//LEGGO LA MAPPA DI GIOCO

perror("read mappa");

return -1;

}

for(int count = 0 ; count < 400 ; count++){//SALVO LA MAPPA DI GIOCO

mappa[i][j] = localbuf[count] ;

j++;

if(j==20){

i++;

j=0;

}

}

free(localbuf);

return 1;

}

/\*INIZIALIZZO IL TIMER DI GIOCO\*/

int initTimer(int sd\_client, char \*timer){

char\* localbuf = malloc(sizeof(char)\*512);

if(read(sd\_client,localbuf,512)<=0){//LEGGO IL TIMER DI GIOCO

perror("read timer");

return -1;

}

strcpy(timer,localbuf);

free(localbuf);

return 1 ;

}

/\*INIZIALIZZO LA LISTA DEGLI UTENTI IN GIOCO\*/

int initListUsers(int sd\_client, char listaUser[26][100]){

char\* localbuf = malloc(sizeof(char)\*1024);

int i = 0 , j = 0 , t = 0 ;

if(read(sd\_client,localbuf,1024)<=0){//LEGGO LA LISTA DI UTENTI IN GIOCO

perror("read lista utenti");

return -1;

}

while(localbuf[i] != '\0' ){

if(localbuf[i] != '-'){

listaUser[j][t] = localbuf[i];//COPIO IL CARATTERE

i++;

t++;

}

else if(localbuf[i] == '-'){

i++; //SALTO IL TRATTINO

listaUser[j][t] = '\0'; //CHIUDO UNA RIGA CON \0

j++; //AVANZO DI UNA RIGA

t = 0 ;

}

}

if(localbuf[i] == '\0'){

listaUser[j][t] = '\0';

}

while(++j<26){

listaUser[j][0] = '\0';

}

free(localbuf);

return 1 ;

}

/\*INIZIALIZZO IL COLORE DELL'UTENTE\*/

int initUserColor(int sd\_client, int\* colore){

char\* localbuf = malloc(sizeof(char)\*512);

int n = 0 ;

if((n = read(sd\_client,localbuf,512))<=0){//LEGGO IL COLORE CHE MI È STATO ASSEGNATO

perror("Errore lettura colore assegnato");

return -1;

}

\*colore = atoi(localbuf);

free(localbuf);

return 1;

}

/\*INIZIALIZZO LA LETTERA DELL'UTENTE\*/

int initUserChar(int sd\_client, char\* carattere){

char\* localbuf = malloc(sizeof(char)\*512);

if(read(sd\_client,localbuf,512)<=0){//LEGGO LA LETTERA CHE MI È STATA ASSEGNATA

perror("Errore lettura lettera assegnata");

return -1;

}

\*carattere = localbuf[0];

\*(carattere + 1) = '\0';

free(localbuf);

return 1 ;

}

/\*INIZIALIZZO IL NUMERO DI TERRITORI NECESSARI PER VINCERE\*/

int initTerritoriNecessari(int sd\_client, int\* territoriNecessari){

char\* localbuf = malloc(sizeof(char)\*512);

if(read(sd\_client,localbuf,512)<=0){//LEGGO IL NUMERO DI TERRE DA CONQUISTARE

perror("read territori da conquista");

return -1;

}

\*territoriNecessari=atoi(localbuf);

free(localbuf);

return 1 ;

}

int initTerritoriPosseduti(int sd\_client, int\* territoriPosseduti){

char\* localbuf = malloc(sizeof(char)\*512);

if(read(sd\_client,localbuf,512)<=0){//LEGGO IL NUMERO DI TERRE POSSEDUTE

perror("read territori da conquista");

return -1;

}

\*territoriPosseduti=atoi(localbuf);

free(localbuf);

return 1 ;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* STAMPE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*STAMPA FINALE COL VINCITORE\*/

void stampaVincitore(char listaVincitori[26][100]){

int i = 0 ;

fprintf(stdout,"\033[1;38;5;10m");

fprintf(stdout,"ABBIAMO FINITO LA PARTITA, COMPLIMENTI A TUTTI\n");

fprintf(stdout,"\033[0m");

fprintf(stdout,"La classifica dei vincitori \n\n");

while(listaVincitori[i][0] != '\0' && i < 26){

fprintf(stdout,"%d)%s\n",i+1,listaVincitori[i]);

i++;

}

fprintf(stdout,"\n");

}

/\*STAMPA DEL MSG DI FINE PARTITA CON ATTESA DI UN INPUT PER L'INIZIO DI UNA NUOVA PARTITA\*/

void stampaFraseFinale(int sd\_client){

char buf[256];

char \* msg =malloc(sizeof(char) \* 128);

int n = 0 ;

strcpy(msg,"Invia 'n' partecipare in una nuova partita : ");

do{

write(STDOUT\_FILENO,msg,strlen(msg));

n = read(STDIN\_FILENO,buf,256);

buf[n-1] = '\0';

fprintf(stdout,"\n");

}while(strcmp(buf,"n") != 0);

conferma(sd\_client);

free(msg);

}

/\*STAMPA DEL MSG DI INTRO E LEGENDA\*/

void stampaLegenda(int colore , char carattere){

fprintf(stdout,"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

fprintf(stdout,"\*\*\*\*\*\*\*\* Benvenuto nella partita campione !! SEI LOGGATO CON : %s\n",usernameGlobale);

fprintf(stdout,"\*\*\*\*\*\*\*\* Il colore che ti è stato assegnato : \033[48;5;%dm \033[0m\n",colore);

fprintf(stdout,"\*\*\*\*\*\*\*\* La lettera con cui giocherai questa partita : %c\n",carattere);

fprintf(stdout,"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n%126s\n","UTENTI COLLEGATI");

}

/\*STAMPA DELLE STATISTICHE DELLA PARTITA IN CORSO E DELLE AZIONI DELL'UTENTE\*/

void stampaStatistichePartita(int posseduti, char tempo[128],int territoriDaConquistare, char\* ultimaAzione){

fprintf(stdout,"Numero di territori posseduti : %d \t\tNumero di territori da conquistare per vincere : %d\n",posseduti,territoriDaConquistare);

fprintf(stdout,"Tempo di gioco rimanente : %s \t\t%s\n\n",tempo,ultimaAzione);

}

/\*STAMPA DELLA DESCRIZIONI DELLE MOSSE POSSIBILI E DELLA MODALITÀ DI LOGOUT\*/

void stampaScelte(){

fprintf(stdout,"Per muoversi nelle 4 direzioni basterà utilizzare le classiche W,A,S,D\n");

fprintf(stdout,"L'azione corrispondente ad ogni lettera è il seguente :\n");

fprintf(stdout,"- q -> LOGOUT\n\n");

fprintf(stdout,"- w -> NORD\n");

fprintf(stdout,"- a -> OVEST\n");

fprintf(stdout,"- s -> SUD\n");

fprintf(stdout,"- d -> EST\n");

}

//STAMPA LA MAPPA DI GIOCO CON LA LISTA UTENTI ACCANTO

void stampaMappa(char mappa[20][20],int coloreU, char carattereU, char utenti[26][100]){

int cond = 0 ;

int k = 0;

printf("\033[1m");

printf(" ");

for (int k = 0 ; k < 10 ; k ++){

printf(" ");

printf("%d",k+1);;

printf(" ");

}

for (int k = 10 ; k < 20 ; k ++){

printf(" %d",k+1);

printf(" ");

}

printf("\033[0m");

printf(" %s\n",utenti[0]);

printf("%-110s%s\n","",utenti[1]);

for ( int i = 0 ; i < 20 ; i++){

for (int j = 0; j < 20 ; j++){

if(j == 0){

printf("\033[1m");

if(i<9)

printf(" %d ",i+1);

else

printf(" %d ",i+1);

printf("\033[0m");

}

printf(" ");

if(mappa[i][j]==carattereU){

coloraCarattere(coloreU,carattereU);

}

else if(mappa[i][j]==tolower(carattereU)){

coloraCarattere(coloreU,'-');

}

else if(mappa[i][j] == '-')

printf("%c",mappa[i][j]);

else {

k = 0 ;

cond = 0 ;

while( cond == 0 && k < 26 ){

if(mappa[i][j] == arrayCaratteri[k] /\*&& arrayCaratteri[k] != carattereU && arrayColori[k] != coloreU\*/){

coloraCarattere(arrayColori[k],arrayCaratteri[k]);

cond = 1 ;

}

else if(mappa[i][j] == tolower(arrayCaratteri[k]) /\*&& arrayCaratteri[k] != carattereU && arrayColori[k] != coloreU\*/){

coloraCarattere(arrayColori[k],'-');

cond = 1 ;

}

k++;

}

}

printf(" ");

}

printf(" %s\n",utenti[i+2]);

}

printf("%-110s%s\n","",utenti[22]);

printf("%-110s%s\n","",utenti[23]);

printf("%-110s%s\n","",utenti[24]);

printf("%-110s%s\n","",utenti[25]);

}

/\*\*\*\*\*\* UTILITY \*\*\*\*\*\*/

int conferma(int sd\_client){

if(write(sd\_client,"1\0",2)<=0){//CONFERMA

perror("Errore invio conferma");

return -1;

}

return 1 ;

}

//SOSTITUISCE LA POSIZIONE X Y DELLA MAPPA CON "CARATTERE" E RESTITUSCE IL CARATTERE

//CHE C'ERA PRIMA DI ESSERE SOSTIUITO O '0' IN CASO DI ERRORE

char sostituisciCasella(int x, int y , char mappa [20][20], char carattere){

char ret ;

if((x < 0 || x > 20)||(y < 0 || y > 20)){

fprintf(stdout,("Errore x o y sballati\n"));

return 0;

}

else{

ret = mappa[x][y];

mappa[x][y] = carattere;

}

return ret;

}

//STAMPA IL CARATTERE COLORATO

int coloraCarattere (int colore , char carattere){

char tmp[30] ;

if(colore < 0 || colore > 255){

fprintf(stdout,"Colore non corretto\n");

return -1;

}

else{

sprintf(tmp ,"\033[38;5;%dm",colore);

fprintf(stdout,"%s",tmp);

fprintf(stdout,"%c",carattere);

fprintf(stdout,"\033[0m");

}

return 1 ;

}

//SERVER.C

#include <stdlib.h>

#include <stdio.h>

#include <stdio\_ext.h>

#include <string.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <fcntl.h>

#include <signal.h>

#include <sys/wait.h>

#include <errno.h>

#include <ctype.h>

#include <time.h>

#include <pthread.h>

#include <sys/socket.h>

#include <sys/un.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include "file.h"

#include "listUser/list.h"

#include "listUser/array.h"

#include "checkInput.h"

#define \_\_USE\_GNU

int err\_th = -1;

char ok[2] = "1\0";

char error[2] = "0\0";

char msLogged[2] = "3\0";

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t cond = PTHREAD\_COND\_INITIALIZER;

pthread\_mutex\_t mutexListaUser = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_mutex\_t mutexListaGame = PTHREAD\_MUTEX\_INITIALIZER;

Game infGioco;

/\*

\*Codice per compilare

\*gcc -o server2 server2.c file.h file.c listUser/array.h listUser/array.c listUser/list.h listUser/list.c checkInput.h checkInput.c lib/inputReader.h lib/inputReader.c -lpthread

\*gcc -w server.c file.h file.c list.h list.c checkInput.h checkInput.c lib/inputReader.h lib/inputReader.c -lpthread

\*/

User head;

/\*

\*Gestisco rottura della pipe con terminazione del thread

\*/

void gestioneSIGPIPE(int tid);

void \*gestisci(void \*);

void \*gestisciTempo(void \*val);

/\*

\*Inizializza la lettera e il colore del nuovo utente e li invia al client

\*ritorna la lettera assegnata

\*/

char initColoreLetteraUtente(User localUser);

/\*

\*mando la mappa al client

\*ritorna 1 nel caso di successo

\*/

int sendMappa(User localUser, char mappa[20][20]);

/\*

\*manda la lista di utente al client sotto forma di una stringa

\*primoGiocatore:n-secondoGiocatore:m-terzoGiocatore ...

\*/

int sendGiocatori(User localUser, UserIG \*listaUtentiConnessi);

/\*

\*mando il tempo rimanente alla chiusura della partita all'user

\*/

int sendTime(User localUser, unsigned int tempo\_gioco);

/\*

\*mando il numero di territori da conquistare

\*/

int sendTerritoriDaConquistare(User localUser, int numeroDiTerritoriDaConquistare);

/\*

\*gestisce tutto riguardante utente loggato

\*/

void prePartita(User localUser);

/\*

\*imita la nuova partita

\*/

int nuovaPartita(User localUser);

/\*

\*esegue logout del utente, controllando se e' presente nella lista di giocatori e rimuovendolo

\*controllando se ha dei possedimenti nella mappa, cancella tutte le sue conquiste dalla mappa

\*/

void LogOut(User localUser);

/\*

\*setta i valori di localUser con i valori di default

\*/

void esciDallaPartita(User localUser);

/\*

\*Funzione di procedura, esegue:

1)Cerca lettera e colore disponibile (initCOloreLetteraUtente)

2)Assegna il colore e lettere al utente e lo comunica al client (initCOloreLetteraUtente)

3)Assegna la posizione (x,y ) iniziale disponibile al utente getPosizoneInizialeCodificata()

4)Salva utente nella matrice di rappresentazione mappa[20][20]

5)invia la mappa al client se ha trovato posizioni iniale disponibile, altrimenti manda un messaggio di errrore

\*RETURN: ritona 1 nel caso in cui ha effettuato tutto correttamente -1 altrimenti

\*/

int inizializzaUtenteNelGioco(User localUser);

/\*

\*controlla se può attuale User può effettuare il movimento "aswd".

\*controllo e' suddiviso in:

1)controllo se il movimento non porta fuori dalla mappa

2)controllo se il movimento non porta sulla posizione nella quale attualmente e' presente un'altro utente

3)controllo se non e' stata soddisfatta la condizione fine gioco (un utente ha conquistato il numero di territori richiesti o e' scaduto il tempo)

\*ritorna 1 nel caso il movimento non e' permesso 0 altrimenti

\*ritorna -1 nel caso di errore della lettura di movimento

\*/

int isBlocked(User localUser, int newX, int newY);

/\*

\*[Prerequisito]:la nuova casella non e' occupata da un'altro giocatore (isBlocked)

\*controlla se la nuova casella non e' sotto proprieta' di un'altro utente

\*ritorna 1 nel caso e' posseduto da un'altro utente 0 altrimenti

\*ritorna -1 nel caso di errore della lettura di movimento

\*/

int isOwnedByEnemy(User localUser, int newX, int newY);

/\*

\*ritorna 1 se il territorio su cui si vuole fare il passaggio e' gia' in possesso all'attuale utente

\*ritorna 0 altrimenti

\*ritorna -1 nel caso di errore della lettura di movimento

\*/

int isMine(User localUser, int newX, int newY);

/\*

\*[Prerequisito]:deve esiste la disponibilita' per effettuare questo movimento(altro utente, limite mappa), funzione non esegue nessun controllo

\*Eseguo movimento verso la nuova casella, nella vecchia posizione lascio tolower(localUser->letteraAssegnata),

\*nella nuova posizione scrivo localUser->letteraAssegnata

\*il numero di conquiste rimane invariato

\*ritorna 1 nel caso in cui il procedimento e' andato a buon fine, 0 altrimenti

\*/

int eseguiMovimentoSenzaConquista(User localUser, int newX, int newY);

/\*

\*Prerequisito]:deve esiste la disponibilita' per effettuare questo movimento(altro utente, limite mappa), funzione non esegue nessun controllo

\*funzione richiama un'altra funzione eseguiMovimentoSenzaConquista(..) e aumenta il numero di conquiste del localUser

\*NOTA BENE: se user ha raggiunto obiettivo, funzione impostera partitaInCorso = 0

\*ritorna 1 nel caso in cui il procedimento e' andato a buon fine, 0 altrimenti

\*/

int eseguiMovimentoConConquista(User localUser, int newX, int newY);

/\*

\*imita due lanci di dadi, uno per attacco un'altro per la difesa. Se attacco > diffesa funzione restituisce 1, 0 altrimenti

\*/

int riuscitoConquistareTerritorio();

/\*

\*ritona il valore flag da 0 a 5

\*/

int eseguiComando(User localUser, int newX, int newY);

/\*

\*Manda mappa, lista utenti, tempo rimamente e nel caso anche vincitore della partita a client

\*ritorna 1 se ha ricevuto flag = 5

\*/

int sendUpdateToUser(User localUser, int flag);

int sendTerritoriPosseduti(User localUser, int nTerritoriPosseduti);

int sendError(int sd\_client,User\* headUser, Game\* headGame, User localUser);

int confermaWrite(int sd\_client,User\* headUser, Game\* headGame, User localUser);

int confermaRead(int sd\_client,User\* headUser, Game\* headGame, User localUser);

void cancellaUserDalServer(User\* headUser, Game\* headGame, User localUser);

int main(int argc, char \*argv[])

{

int sd\_server, sd\_client, porta\_ingresso;

struct sockaddr\_in server\_addr, client\_addr;

pthread\_t tid;

socklen\_t client\_len;

char \*ip\_adress;

if (argc != 2)

{

perror("argc"), exit(-1);

}

porta\_ingresso = atoi(argv[1]);

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(porta\_ingresso);

server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY); //specifico adress che voglio accettare, in questo caso accetto un address qualsiasi

sd\_server = socket(PF\_INET, SOCK\_STREAM, 0);

if (sd\_server < 0)

{

perror("socket"), exit(-1);

}

bind(sd\_server, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

listen(sd\_server, 100);

while (1)

{

client\_len = sizeof(client\_addr);

if ((sd\_client = accept(sd\_server, (struct sockaddr \*)&client\_addr, &client\_len)) < 0)

perror("accept"), exit(-1);

ip\_adress = inet\_ntoa(client\_addr.sin\_addr);

log\_write\_message("Accesso:\n");

log\_write\_access(NULL, ip\_adress);

write(STDOUT\_FILENO, "Nuovo accesso",13);

Client client = initNodeClient(ip\_adress, sd\_client);

pthread\_create(&tid, NULL, gestisci, (void \*)client);

}

return 0;

}

void \*gestisciTempo(void \*val)

{

Game tmp = val;

pthread\_mutex\_lock(&mutex);

while (tmp->partitaInCorso != 1)

{

pthread\_cond\_wait(&cond, &mutex);

}

printf("La partita id:%d e' iniziata\n", tmp->idGame);

while (tmp->tempo\_gioco > 0)

{

sleep(1);

tmp->tempo\_gioco--;

if (tmp->partitaInCorso != 1)

{

break;

}

}

if (tmp->tempo\_gioco == 0)

{

tmp->partitaInCorso = 2;

printf("Il tempo del gioco della parita id:%d e' terminato\n", tmp->idGame);

}

pthread\_mutex\_unlock(&mutex);

}

void gestioneSIGPIPE(int tid)

{

printf("Uccido thread: %ld\n", pthread\_self());

pthread\_cancel(pthread\_self());

}

void \*gestisci(void \*sd)

{

signal(SIGPIPE, gestioneSIGPIPE);

int esito = 0 ;

Client localClient = sd;

User localUser = NULL;

char buf[1024];

int choice\_client;

int n, condUsicta = 0;

char userName[36], password[36];

int nuovaOperazione = 0; // 0 se deve scegliere di nuovo utente, altrimenti il numero dell'operazione da effettuare

// Mi sono appena connesso con client

do

{

//Aspetto utente che sceglie operazione da eseguire

if (nuovaOperazione == 0 && (n = read(localClient->sd\_client, buf, 1)) <= 0)

perror("read benvenuto"), pthread\_exit(&err\_th);

choice\_client = atoi(buf);

switch (choice\_client)

{

case 1:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ...LOGIN... \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* RICEVO USERNAME \*/

if ((n = read(localClient->sd\_client, userName, 512)) <= 0){

perror("read username login");

pthread\_exit(&err\_th);

}

if (strcmp(userName, "q") == 0)

{

nuovaOperazione = 0;

break;

}

confermaWrite(localClient->sd\_client, &head, &infGioco, localUser);

/\* RICEVO PASSWORD \*/

if ((n = read(localClient->sd\_client, password, 36)) <= 0)

perror("read password login"), pthread\_exit(&err\_th);

if (strcmp(password, "q") == 0)

{

nuovaOperazione = 0;

break;

}

pthread\_mutex\_lock(&mutexListaUser);

esito = isLogged(head, userName);

pthread\_mutex\_unlock(&mutexListaUser);

if (checkCredenziali(userName, password) && !esito)

{

printf("Utente loggato:%s\n", userName);

confermaWrite(localClient->sd\_client, &head, &infGioco, localUser);

nuovaOperazione = 0;

//Login effettuato con successo, inizializzo il nuovo utente

localUser = initNodeUser(userName, 0, 0, localClient->adress, 0, 0, localClient->sd\_client);

pthread\_mutex\_lock(&mutexListaUser);

head = appendNodeUser(head, localUser);

pthread\_mutex\_unlock(&mutexListaUser);

/\* Scrivo nel log file evento utente loggato \*/

log\_write\_message("Accesso:\n");

log\_write\_access(localUser->username, localUser->address);

prePartita(localUser);

//A questo punto utente entra nella partita, gli devo mandare la matrice con tutti i dati

}

else

{

if (checkCredenziali(userName, password)){// se mi trovo nel else e utente esiste nel db => utente gia' connesso

if((write(localClient->sd\_client, msLogged, 2))<=0){

perror("write checkCredinziali");

cancellaUserDalServer(&head, &infGioco, localUser);

}

}

else{

if((write(localClient->sd\_client, error, 2))<=0){

perror("write checkCredinziali");

cancellaUserDalServer(&head, &infGioco, localUser);

}

}

nuovaOperazione = 1; //Resto sulla schermata login e aspetto username

}

//pulisco le variabili

memset(userName, 0, sizeof(userName));

memset(password, 0, sizeof(password));

break;

case 2:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ...REGISTRAZIONE... \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* RICEVO USERNAME \*/

if ((n = read(localClient->sd\_client, userName, 36)) <= 0)

perror("read username registrazione"), pthread\_exit(&err\_th);

if (strcmp(userName, "q") == 0)

{

nuovaOperazione = 0;

break;

}

// invio la conferma

confermaWrite(localClient->sd\_client, &head, &infGioco, localUser);

/\* RICEVO PASSWORD \*/

if ((n = read(localClient->sd\_client, password, 36)) <= 0)

perror("read passowrd registrazione"), pthread\_exit(&err\_th);

if (strcmp(password, "q") == 0)

{

nuovaOperazione = 0;

break;

};

if (!userExist(userName))

{

printf("Utente:%s e' stato registrato\n", userName);

//Utente non esiste nel db, lo registro e mando il segnale al client

registraUser(userName, password);

confermaWrite(localClient->sd\_client, &head, &infGioco, localUser);

nuovaOperazione = 0;

/\* Scrivo nel log file evento utente registrato \*/

log\_write\_message("Utente registrato:\n");

log\_write\_access(userName, localClient->adress);

//Ritorno alla schermata di login

}

else

{

//Utente e' gia' registrato

sendError(localClient->sd\_client, &head, &infGioco, localUser);

//write(localClient->sd\_client, error, 2);

nuovaOperazione = 2;

//Resto sulla schermata login e non faccio niente

}

//pulisco le variabili

memset(userName, 0, sizeof(userName));

memset(password, 0, sizeof(password));

break;

case 3:

/\*\*\* USITA DAL SERVER \*\*\*/

printf("Chiudo connessione con il client:%s\n", localClient->adress);

close(localClient->sd\_client);

free(localClient);

condUsicta = 1;

break;

}

} while (condUsicta != 1);

return 0;

}

void prePartita(User localUser)

{

Game tmp;

int ret;

pthread\_t tid;

do

{

pthread\_mutex\_lock(&mutexListaGame);

tmp = findGameDisponibile(infGioco);

pthread\_mutex\_unlock(&mutexListaGame);

if (tmp != NULL)

{

localUser->game = tmp;

}

else

{

localUser->game = initNodeGame(120, 20);

pthread\_mutex\_lock(&mutexListaGame);

infGioco = appendNodeGame(infGioco, localUser->game);

pthread\_mutex\_unlock(&mutexListaGame);

pthread\_create(&tid, NULL, gestisciTempo, (void \*)localUser->game);

}

UserIG tmpUser = initNodeUserIG(localUser->username);

pthread\_mutex\_lock(&mutexListaGame);

localUser->game->utenti = appendNodeUserIG(localUser->game->utenti, tmpUser);

pthread\_mutex\_unlock(&mutexListaGame);

ret = nuovaPartita(localUser);

} while (ret != 0);

}

int nuovaPartita(User localUser)

{

int ret = 0;

int size\_buf = 2;

char \*buf = malloc(sizeof(char) \* size\_buf);

char carr;

int n, choice\_client, flag = -1;

if (inizializzaUtenteNelGioco(localUser) != 1)

{

pthread\_exit(&err\_th);

}

//si può' giocare

do

{

int newX = localUser->x, newY = localUser->y;

if ((n = read(localUser->sd\_client, buf, 2)) <= 0)

{

perror("read scelta mossa");

cancellaUserDalServer(&head, &infGioco, localUser);

}

carr = \*buf;

if (carr == 'q')

{

LogOut(localUser);

free(buf);

return 0;

}

else if (carr == 'w')

{

newX = localUser->x - 1;

flag = eseguiComando(localUser, newX, newY);

}

else if (carr == 's')

{

newX = localUser->x + 1;

flag = eseguiComando(localUser, newX, newY);

}

else if (carr == 'a')

{

newY = localUser->y - 1;

flag = eseguiComando(localUser, newX, newY);

}

else if (carr == 'd')

{

newY = localUser->y + 1;

flag = eseguiComando(localUser, newX, newY);

}

else

{

break;

}

//MANDO I DATI AGGIORNATI

sendUpdateToUser(localUser, flag);

if(flag == 5)

return 1;

} while (1);

memset(buf, 0, size\_buf);

free(buf);

return ret;

}

int inizializzaUtenteNelGioco(User localUser)

{

char letteraUser;

int posizioneInizialeCodificata, x, y, indice;

startGameTimer(localUser->game, &mutex, &cond);

indice = getIndiceDispForLetteraColore(localUser->game->arrayDisp);

posizioneInizialeCodificata = getPosizoneInizialeCodificata(localUser->game->mappa);

if (posizioneInizialeCodificata != -1 && indice != -1)

{

letteraUser = initColoreLetteraUtente(localUser);

x = (posizioneInizialeCodificata / 20);

y = (posizioneInizialeCodificata % 20);

localUser->game->mappa[x][y] = letteraUser;

localUser->game->count++;

localUser->x = x;

localUser->y = y;

localUser->nTerritoriPosseduti = 1;

incrementaNumTerritoriPosseduti(localUser->game->utenti, localUser->username);

pthread\_mutex\_lock(&mutexListaGame);

sendMappa(localUser, localUser->game->mappa);

sendGiocatori(localUser, &(localUser->game->utenti));

pthread\_mutex\_unlock(&mutexListaGame);

confermaRead(localUser->sd\_client, &head, &infGioco, localUser);

sendTime(localUser, localUser->game->tempo\_gioco);

confermaRead(localUser->sd\_client, &head, &infGioco, localUser);

sendTerritoriDaConquistare(localUser, localUser->game->numeroDiTerritoriDaConquistare);

return 1;

}

else

{

sendError(localUser->sd\_client, &head, &infGioco, localUser);

return -1;

}

}

char initColoreLetteraUtente(User localUser)

{

int indiceDisp, coloreUser, letti, scritti;

char coloreUserChar[128], letteraUser[128], buf[128];

//recupero primo indice disponibile dall'array di colori/lettere

indiceDisp = getIndiceDispForLetteraColore(localUser->game->arrayDisp);

// prendo i-esima lettera

letteraUser[0] = localUser->game->arrayLettera[indiceDisp];

letteraUser[1] = '\0';

localUser->letteraAssegnata = localUser->game->arrayLettera[indiceDisp];

//prendo i-esimo colore

coloreUser = localUser->game->arrayColori[indiceDisp];

localUser->coloreAssegnato = localUser->game->arrayColori[indiceDisp];

sprintf(coloreUserChar, "%d", coloreUser);

coloreUserChar[strlen(coloreUserChar)] = '\0';

// rendo i-esimo colore/lettera non disponibile

localUser->game->arrayDisp[indiceDisp] = 1;

if((scritti=write(localUser->sd\_client, letteraUser, 128)) <= 0){

perror("write invia lettera");

cancellaUserDalServer(&head, &infGioco, localUser);

}

confermaRead(localUser->sd\_client, &head, &infGioco, localUser);

if((write(localUser->sd\_client, coloreUserChar, 128)) <= 0){

perror("write invia colore");

cancellaUserDalServer(&head, &infGioco, localUser);

}

confermaRead(localUser->sd\_client, &head, &infGioco, localUser);

//ritorna lettera del utente

return letteraUser[0];

}

void LogOut(User localUser)

{

rimozioneUtenteDallaMappa(localUser);

//rendo disponibile nella matrice delle lettere e colori

localUser->game->arrayDisp[(int)localUser->letteraAssegnata - 65] = 0;

pthread\_mutex\_lock(&mutexListaGame);

localUser->game->utenti = removeNodeUserIGbyUsername(localUser->game->utenti, localUser->username);

localUser->game->count--;

if (localUser->game->count == 0 && localUser->game->partitaInCorso == 2)

{

localUser->game->tempo\_gioco = 0; // per terminare il thread che sta decrementando il tempo e tiene mutex

infGioco = removeNodeGameById(infGioco, localUser->game->idGame);

}

pthread\_mutex\_unlock(&mutexListaGame);

// controllo che si può' evitare, per adesso lo lascio

log\_write\_logout(localUser->username, localUser->address);

// rimuovo utente dalla lista

pthread\_mutex\_lock(&mutexListaUser);

head = removeUserByUsername(head, localUser->username);

pthread\_mutex\_unlock(&mutexListaUser);

}

void esciDallaPartita(User localUser)

{

// rimuovo utente dalla lista

localUser->coloreAssegnato = 0;

localUser->letteraAssegnata = 0;

localUser->nTerritoriPosseduti = 0;

localUser->x = 0;

localUser->y = 0;

pthread\_mutex\_lock(&mutexListaGame);

localUser->game->count--;

if (localUser->game->count == 0)

{

infGioco = removeNodeGameById(infGioco, localUser->game->idGame);

}

localUser->game = NULL;

pthread\_mutex\_unlock(&mutexListaGame);

}

int eseguiComando(User localUser, int newX, int newY)

{

int flag;

pthread\_mutex\_lock(&mutexListaUser);

pthread\_mutex\_lock(&mutexListaGame);

if (!(isBlocked(localUser, newX, newY)))

{

if (isOwnedByEnemy(localUser, newX, newY))

{ //

if (riuscitoConquistareTerritorio())

{

flag = 2;

char letteraGiocatoreInDifesa = localUser->game->mappa[newX][newY];

char \*usernameDifesa; // riferimento ad un stringa dinamica interna alla funzione

usernameDifesa = getUsernameByLetteraNelGioco(head, letteraGiocatoreInDifesa, localUser->game->idGame);

eseguiMovimentoConConquista(localUser, newX, newY);

updateUserTerritoriByLettera(head, letteraGiocatoreInDifesa, -1);

decrementaNumTerritoriPosseduti(localUser->game->utenti, usernameDifesa);

log\_write\_conquista(localUser->username, newX, newY);

free(usernameDifesa);

}

else

{

flag = 3; //resto sul posto

}

}

else if (isMine(localUser, newX, newY))

{

flag = 4;

eseguiMovimentoSenzaConquista(localUser, newX, newY);

}

else

{ // casella e' libera

flag = 1;

eseguiMovimentoConConquista(localUser, newX, newY);

log\_write\_conquista(localUser->username, newX, newY);

}

}

else

{

flag = 0;

}

if (localUser->game->partitaInCorso == 2)

flag = 5;

// close mutex

pthread\_mutex\_unlock(&mutexListaGame);

pthread\_mutex\_unlock(&mutexListaUser);

return flag;

}

int sendUpdateToUser(User localUser, int flag)

{

int size\_flagBuf = 2;

char \*flagBuf = malloc(sizeof(char) \* size\_flagBuf);

/\* flag 0, movimento bloccato,

flag 1 movimento sul nuovo territorio +1 conq,

flag 2 movimento conquista del territorio nemico

flag 3 movimento sul territorio nemico senza la conquista, rimani sullo stesso posto

flag 4 movimento sul proprio territorio

flag 5 e' finito il gioco invio la notifica al utente con i risultati

\*/

sprintf(flagBuf, "%d", flag);

if((write(localUser->sd\_client, flagBuf, size\_flagBuf))<=0){

perror("write flag");

cancellaUserDalServer(&head, &infGioco, localUser);

}

confermaRead(localUser->sd\_client, &head, &infGioco, localUser);

pthread\_mutex\_lock(&mutexListaGame);

sendMappa(localUser, localUser->game->mappa);

confermaRead(localUser->sd\_client, &head, &infGioco, localUser);

sendGiocatori(localUser, &(localUser->game->utenti));

confermaRead(localUser->sd\_client, &head, &infGioco, localUser);

sendTime(localUser, localUser->game->tempo\_gioco);

pthread\_mutex\_unlock(&mutexListaGame);

confermaRead(localUser->sd\_client, &head, &infGioco, localUser);

pthread\_mutex\_lock(&mutexListaUser);

sendTerritoriPosseduti(localUser, localUser->nTerritoriPosseduti);

pthread\_mutex\_unlock(&mutexListaUser);

if (flag == 5)

{

confermaRead(localUser->sd\_client, &head, &infGioco, localUser);

esciDallaPartita(localUser);

}

free(flagBuf);

}

int sendError(int sd\_client, User\* headUser, Game\* headGame, User localUser)

{

if (write(sd\_client, "0\0", 2) <= 0)

{

perror("Errore invio errore");

cancellaUserDalServer(headUser, headGame, localUser);

return -1;

}

return 1;

}

int confermaWrite(int sd\_client, User\* headUser, Game\* headGame, User localUser)

{

if (write(sd\_client, "1\0", 2) <= 0)

{

perror("Errore invio conferma");

cancellaUserDalServer(headUser, headGame, localUser);

return -1;

}

return 1;

}

int confermaRead(int sd\_client, User\* headUser, Game\* headGame, User localUser)

{

char \*buffer = malloc(sizeof(char) \* 32);

if (read(sd\_client, buffer, 2) <= 0)

{

perror("Errore ricezione conferma");

free(buffer);

cancellaUserDalServer(headUser, headGame, localUser);

return -1;

}

return 1;

free(buffer);

}

int sendMappa(User localUser, char mappa[20][20])

{

char buf[800];

int k = 0;

for (int i = 0; i < 20; i++)

{

for (int j = 0; j < 20; j++)

{

buf[k++] = mappa[i][j];

}

}

buf[k] = '\0';

if((write(localUser->sd\_client, buf, strlen(buf)))<=0){

perror("write sendMappa");

cancellaUserDalServer(&head, &infGioco, localUser);

}

return 1;

}

int sendGiocatori(User localUser, UserIG \*listaUtentiConnessi)

{

char \*buffer;

// buffer = getConnectedUserInString(head);

// N.B qualsiasi cosa passo e' una copia di riferimento

MergeSort((listaUtentiConnessi));

buffer = getConnectedUsersInGame(\*listaUtentiConnessi);

// INVIO LISTA DI GIOCATORI AL CLIENT

if((write(localUser->sd\_client, buffer, strlen(buffer) + 1))<=0){

perror("write sendGiocatori");

cancellaUserDalServer(&head, &infGioco, localUser);

}

free(buffer);

}

int sendTime(User localUser, unsigned int tempo\_gioco)

{

int tempo\_rimanente = tempo\_gioco;

int minuti = 0, secondi = 0;

char \*tmpMin = malloc(sizeof(char) \* 4);

char \*tmpSec = malloc(sizeof(char) \* 3);

char \*tempoString = malloc(sizeof(char) \* 32);

while (tempo\_rimanente != 0)

{

if (tempo\_rimanente >= 60)

{

minuti++;

tempo\_rimanente -= 60;

}

else

{

secondi = tempo\_rimanente;

tempo\_rimanente = 0;

}

}

sprintf(tmpMin, "%d", minuti);

sprintf(tmpSec, "%d", secondi);

tempoString = concatenation(tmpMin, concatenation("m:", (concatenation(tmpSec, "s"))));

// INVIO IL TEMPO "Xm:Ys" al client

if((write(localUser->sd\_client, tempoString, 32))<=0){

perror("write sendTime");

cancellaUserDalServer(&head, &infGioco, localUser);

}

}

int sendTerritoriDaConquistare(User localUser, int numeroDiTerritoriDaConquistare)

{

int size\_tmpNumeroTerritoriDaConquistare = 4;

char \*tmpNumeroTerritoriDaConquistare = malloc(sizeof(char) \* size\_tmpNumeroTerritoriDaConquistare);

sprintf(tmpNumeroTerritoriDaConquistare, "%d", numeroDiTerritoriDaConquistare);

// INVIO NUMERO DI TERRITORI DA CONQUISTARE

if((write(localUser->sd\_client, tmpNumeroTerritoriDaConquistare, size\_tmpNumeroTerritoriDaConquistare))<=0){

perror("write sendTerritoriDaConquistare");

cancellaUserDalServer(&head, &infGioco, localUser);

}

}

int sendTerritoriPosseduti(User localUser, int nTerritoriPosseduti)

{

char \*localBuf = malloc(sizeof(char) \* 32);

sprintf(localBuf, "%d", nTerritoriPosseduti);

if (write(localUser->sd\_client, localBuf, 32) <= 0)

{

perror("write sendTerritoriPosseduti");

free(localBuf);

cancellaUserDalServer(&head, &infGioco, localUser);

return -1;

}

free(localBuf);

return 1;

}

int isBlocked(User localUser, int newX, int newY)

{

if (newX > 19 || newY > 19 || newX < 0 || newY < 0)

{

return 1;

}

// se il valore della nuova casella e' una lettera maiuscola allora e' un'altro giocatore

// non c'e' il bisogno di discriminare attuale utente perché utente può' trovarsi solo in un posto allo stesso tempo

// se utente attuale ha fatto un passo e' sicuro che non trovera' se stesso.

if (65 <= localUser->game->mappa[newX][newY] && localUser->game->mappa[newX][newY] <= 90)

{

return 1;

}

// controllo se non e' stata soddisfata la condizione del fine gioco

// adesso le condizioni di uscita sono: 0 partita non init, 1 partita in corso, 2 partita terminata

if (localUser->game->partitaInCorso != 1)

{

// dovrei mandare un messaggio al client dicendo che e' finita la partita'

return 1;

}

//se nessuno dei primi if e' stato soddisfatto => ho la posibilita' di fare il movimento

return 0;

}

int isOwnedByEnemy(User localUser, int newX, int newY)

{

if (97 <= localUser->game->mappa[newX][newY] && localUser->game->mappa[newX][newY] <= 122 && localUser->game->mappa[newX][newY] != tolower(localUser->letteraAssegnata))

{

return 1;

}

// se if non e' verificato allora il territorio non e' posseduto da altri utenti

// puo' essere libero o appartenere all'attuale utente

return 0;

}

int isMine(User localUser, int newX, int newY)

{

if (97 <= localUser->game->mappa[newX][newY] && localUser->game->mappa[newX][newY] <= 122 && localUser->game->mappa[newX][newY] == tolower(localUser->letteraAssegnata))

{

return 1;

}

// il territorio su quale voglio fare il passaggio non e' di mia proprieta'

return 0;

}

int eseguiMovimentoSenzaConquista(User localUser, int newX, int newY)

{

localUser->game->mappa[newX][newY] = localUser->letteraAssegnata;

localUser->game->mappa[localUser->x][localUser->y] = tolower(localUser->letteraAssegnata);

localUser->x = newX;

localUser->y = newY;

}

int eseguiMovimentoConConquista(User localUser, int newX, int newY)

{

eseguiMovimentoSenzaConquista(localUser, newX, newY);

incrementaNumTerritoriPosseduti(localUser->game->utenti, localUser->username);

localUser->nTerritoriPosseduti++;

// printAllUserIG(localUser->game->utenti);

if (localUser->nTerritoriPosseduti >= localUser->game->numeroDiTerritoriDaConquistare)

{

localUser->game->partitaInCorso = 2;

strcpy(localUser->game->vincitore, localUser->username);

//localUser->game->vincitore = localUser->username;

}

}

int riuscitoConquistareTerritorio()

{

srand(time(NULL));

int attacco = 1 + rand() % 6;

int difesa = 1 + rand() % 6;

if (attacco > difesa)

return 1;

else

return 0;

}

void cancellaUserDalServer(User\* headUser, Game\* headGame, User localUser){

User tmpUser = \*headUser;

Game tmpGame = \*headGame;

int err\_th = -1 ;

if(localUser == NULL){

pthread\_mutex\_unlock(&mutexListaGame);

pthread\_mutex\_unlock(&mutexListaUser);

pthread\_exit(&err\_th);

}

if(localUser->username == NULL){

pthread\_mutex\_unlock(&mutexListaGame);

pthread\_mutex\_unlock(&mutexListaUser);

pthread\_exit(&err\_th);

}

if(localUser->game == NULL){

pthread\_mutex\_unlock(&mutexListaGame);

pthread\_mutex\_unlock(&mutexListaUser);

pthread\_exit(&err\_th);

}

rimozioneUtenteDallaMappa(localUser);

localUser->game->arrayDisp[(int)localUser->letteraAssegnata - 65] = 0;

localUser->game->utenti = removeNodeUserIGbyUsername(localUser->game->utenti, localUser->username);

localUser->game->count--;

if(localUser->game->count == 0 && localUser->game->partitaInCorso == 2){

infGioco = removeNodeGameById(infGioco,localUser->game->idGame);

printAllGameInfo(infGioco);

}

close(localUser->sd\_client);

head = removeUserByUsername(head,localUser->username);

pthread\_mutex\_unlock(&mutexListaGame);

pthread\_mutex\_unlock(&mutexListaUser);

pthread\_exit(&err\_th);

}

//FILE.C

#include "file.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <errno.h>

#include <ctype.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <x86\_64-linux-gnu/bits/fcntl-linux.h> //contiene le modalita di apertura per la funzione open()

#include <time.h>

#include "checkInput.h"

#include "pthread.h"

#define WRITE 1

#define READ 0

pthread\_mutex\_t mutexFileUtenti = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_mutex\_t mutexFileLog = PTHREAD\_MUTEX\_INITIALIZER;

// compilazione: gcc -w file.c checkInput.h checkInput.c lib/inputReader.h lib/inputReader.c -lpthread

/\*

\*Scrive nel file log.txt nome e ip (se disponibili)

\*ritorna 1 se inserimento e' avvenuto con succcesso -1 altrimenti

\*Non controlla se i dati passati sono corretti ma solo se sono != NULL

\*/

int log\_write\_access(char \*nome, char \*ip)

{

pthread\_mutex\_lock(&mutexFileLog);

int logfile;

char \*userMessaggio = "User name: ";

char \*addressMessaggio = "Ip: ";

char dataMessaggio[30];

time\_t data;

if (nome == NULL && ip == NULL)

return -1;

if ((logfile = open("log.txt", O\_WRONLY | O\_CREAT | O\_APPEND)) < 0)

{

perror("open log\_write\_access");

return -1;

}

if (nome != NULL)

{

write(logfile, userMessaggio, strlen(userMessaggio));

write(logfile, nome, strlen(nome));

write(logfile, "\n", 1);

}

if (ip != NULL)

{

write(logfile, addressMessaggio, strlen(addressMessaggio));

write(logfile, ip, strlen(ip));

write(logfile, "\n", 1);

}

// GiornoSettimana Mese giornoMese ora:min:sec anno

time(&data);

ctime\_r(&data, dataMessaggio); //salvo il tempo attuale nella stringa locale dataMessaggio

write(logfile, dataMessaggio, strlen(dataMessaggio)); // contiene gia' \n

close(logfile);

pthread\_mutex\_unlock(&mutexFileLog);

return 1;

}

int log\_write\_conquista(char \*username, int x, int y)

{

pthread\_mutex\_lock(&mutexFileLog);

int logfile;

char \*userMessaggio = "username: ";

char \*coordinateMessaggio = "coordinate: ";

char \*conquistaMessaggio = "Conquista:";

char \*xString = malloc(sizeof(char) \* 3);

char \*yString = malloc(sizeof(char) \* 3);

char dataMessaggio[30];

time\_t data;

if (username == NULL)

return -1;

if ((logfile = open("log.txt", O\_WRONLY | O\_CREAT | O\_APPEND)) < 0)

{

perror("open log\_write\_conquista");

return -1;

}

sprintf(xString, "%d", x);

sprintf(yString, "%d", y);

write(logfile, conquistaMessaggio, strlen(conquistaMessaggio));

write(logfile, "\n", 1);

// username: nomeUser

write(logfile, userMessaggio, strlen(userMessaggio));

write(logfile, username, strlen(username));

write(logfile, "\n", 1);

// coordinate: x:y

write(logfile, coordinateMessaggio, strlen(coordinateMessaggio));

write(logfile, xString, strlen(xString));

write(logfile, ":", 1);

write(logfile, yString, strlen(yString));

write(logfile, "\n", 1);

// GiornoSettimana Mese giornoMese ora:min:sec anno

time(&data);

ctime\_r(&data, dataMessaggio); //salvo il tempo attuale nella stringa locale dataMessaggio

write(logfile, dataMessaggio, strlen(dataMessaggio)); // contiene gia' \n

close(logfile);

free(xString);

free(yString);

pthread\_mutex\_unlock(&mutexFileLog);

return 1;

}

int log\_write\_logout(char\* username, char \*ip){

log\_write\_message("Logout\n"); // ha gia' mutex

log\_write\_access(username, ip); // ha gia' mutex

}

/\*

\*Scrive un messagio personalizzato nel file log.txt SENZA \n alla fine

\*pensato per essere usato insieme(prima) alle altre funzioni come log\_write\_access(..) ecc

\*ritorna 1 nel caso di successo e -1 altrimenti

\*/

int log\_write\_message(char \*messaggio)

{

pthread\_mutex\_lock(&mutexFileLog);

int logfile;

if (messaggio == NULL)

{

return -1;

}

if ((logfile = open("log.txt", O\_WRONLY | O\_CREAT | O\_APPEND)) < 0)

{

perror("open log\_write\_message");

return -1;

}

write(logfile, messaggio, strlen(messaggio));

close(logfile);

pthread\_mutex\_unlock(&mutexFileLog);

return 1;

}

int checkCredenziali(char \*username, char \*password)

{

pthread\_mutex\_lock(&mutexFileUtenti);

int p1[2];

char buf[2];

char \*query;

char \*qUser;

char \*qPasw;

int elementiTrovati;

qUser = concatenation("@username: ", username);

qPasw = concatenation(" @password: ", password);

query = concatenation(qUser, qPasw);

pid\_t childID;

if (pipe(p1) < 0)

{

perror("pipe");

exit(0);

}

childID = fork();

if (childID < 0)

{

perror("fork");

exit(0);

}

else if (childID == 0)

{

close(p1[0]);

dup2(p1[1], STDOUT\_FILENO);

close(p1[1]);

execlp("grep", "grep", "-c", query, "utenti.txt", (char \*)NULL);

perror("exec");

exit(0);

}

else

{

close(p1[1]);

dup2(p1[0], STDIN\_FILENO);

close(p1[0]);

read(STDIN\_FILENO, buf, 2);

elementiTrovati = atoi(buf);

free(qUser);

free(qPasw);

free(query);

pthread\_mutex\_unlock(&mutexFileUtenti);

return elementiTrovati;

}

}

int userExist(char \*username)

{

pthread\_mutex\_lock(&mutexFileUtenti);

int p1[2];

char buf[2];

char \*query;

int elementiTrovati = 0;

query = concatenation("@username: ", concatenation(username, " @"));

pid\_t childID;

if (pipe(p1) < 0)

{

perror("pipe");

exit(0);

}

childID = fork();

if (childID < 0)

{

perror("fork");

exit(0);

}

else if (childID == 0)

{

close(p1[0]);

dup2(p1[1], STDOUT\_FILENO);

close(p1[1]);

execlp("grep", "grep", "-c", query, "utenti.txt", (char \*)NULL);

perror("exec");

exit(0);

}

else

{

close(p1[1]);

dup2(p1[0], STDIN\_FILENO);

close(p1[0]);

read(STDIN\_FILENO, buf, 1);

elementiTrovati = atoi(buf);

free(query);

pthread\_mutex\_unlock(&mutexFileUtenti);

return elementiTrovati;

}

}

int write\_message\_inFile(char \*messaggio, char \*nameFile)

{

int tipoMutex = 0;

if(strcmp(nameFile, "utenti.txt") == 0){

tipoMutex = 1;

}

if(tipoMutex == 0){

pthread\_mutex\_lock(&mutexFileLog);

}else{

pthread\_mutex\_lock(&mutexFileUtenti);

}

int logfile;

if (messaggio == NULL)

{

return -1;

}

if ((logfile = open(nameFile, O\_WRONLY | O\_CREAT | O\_APPEND)) < 0)

{

perror("open write\_message\_inFile");

return -1;

}

write(logfile, messaggio, strlen(messaggio));

close(logfile);

if(tipoMutex == 0){

pthread\_mutex\_unlock(&mutexFileLog);

}else{

pthread\_mutex\_unlock(&mutexFileUtenti);

}

return 1;

}

int registraUser(char \*username, char \*password)

{

write\_message\_inFile(concatenation("@username: ", username), "utenti.txt");

write\_message\_inFile(" ", "utenti.txt");

write\_message\_inFile(concatenation("@password: ", password), "utenti.txt");

write\_message\_inFile(" \n", "utenti.txt");

return 1;

}

/\*

int main(int argc, char const \*argv[])

{

//log\_write\_conquista("dima", 10, 10);

//checkCredenziali("mimmo", "esempio");

userExist("mimmo");

return 0;

}

\*/

//CHECKINPUT.C

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <ctype.h>

#include "./lib/inputReader.h"

int controllaLunghezza(char\* stringa, int dimensione\_esatta, int dimensione\_min, int dimensione\_max){

int dimensione\_stringa = strlen(stringa);

if(dimensione\_esatta != 0){

if( dimensione\_stringa == dimensione\_esatta)

return 1;

else

return 0;

}

else if(dimensione\_min != 0){

if(dimensione\_stringa >= dimensione\_min){

if(dimensione\_stringa <= dimensione\_max)

return 1;

else

return 0;

}

else

return 0;

}

else{

printf("Errore controllo parametri funzione controllaLunghezza");

return 0;

}

}

int doSceltaInt(char\* messaggio, int valore\_max){

int input\_locale;

do{

printf("%s", messaggio);

}while(!getPositive(&input\_locale) || input\_locale == 0 || input\_locale > valore\_max);

return input\_locale;

}

int doSceltaIntError(char\* messaggio, int valore\_max,char \* error){

int input\_locale;

int i=0;

do{

if(i>0)

printf("%s",error);

i++;

printf("%s", messaggio);

}while(!getPositive(&input\_locale) || input\_locale == 0 || input\_locale > valore\_max);

return input\_locale;

}

int doSceltaIntZero(char\* messaggio, int valore\_max){

int input\_locale;

do{

printf("%s", messaggio);

}while(!getPositive(&input\_locale) || input\_locale > valore\_max);

return input\_locale;

}

double doSceltaDoubleZero(char\* messaggio, int valore\_max){

double input\_locale=-1;

do{

printf("%s", messaggio);

}while(!getPositiveDouble(&input\_locale) || input\_locale > valore\_max);

// printf("AAA %f\n",input\_locale);

return input\_locale;

}

int doSceltaIntZeroError(char\* messaggio, int valore\_max, char\* error){

int input\_locale;

int i = 0;

do{

if(i>0)

printf("%s",error);

i++;

printf("%s", messaggio);

}while(!getPositive(&input\_locale) || input\_locale > valore\_max);

return input\_locale;

}

char\* doSceltaString(char\* messaggio, int dimensione\_esatta, int dimensione\_min, int dimensione\_max){

char buff[20];

//int\* pt = &buff;

char\* pt = buff;

do{

printf("%s", messaggio);

scanf("%s", buff);

}while(!controllaLunghezza(buff, dimensione\_esatta, dimensione\_min, dimensione\_max));

return pt;

}

char\* doSceltaStringError(char\* messaggio, char\* error, int dimensione\_esatta, int dimensione\_min, int dimensione\_max){

char buff[20];

//int\* pt = &buff;

char\* pt = buff;

int i = 0;

do{

if(i>0)

printf("%s",error);

i++;

printf("%s", messaggio);

scanf("%s", buff);

}while(!controllaLunghezza(buff, dimensione\_esatta, dimensione\_min, dimensione\_max));

return pt;

}

char \* doSceltaStringZero(char\* messaggio, int dimensione\_esatta, int dimensione\_min, int dimensione\_max){

char buff[20];

//int\* pt = &buff;

char\* pt = buff;

do{

printf("%s", messaggio);

scanf("%s", buff);

if( !strcmp(buff,"0"))

break;

}while(!controllaLunghezza(buff, dimensione\_esatta, dimensione\_min, dimensione\_max));

return pt;

}

char\* doCompare () {

char\* buff;

char\* buff1;

buff = (char\*)malloc(sizeof(char)\*20);

buff1 = (char\*)malloc(sizeof(char)\*20);

do{

printf("Inserisci Password:");

scanf("%s", buff);

printf("Conferma Password:");

scanf("%s", buff1);

}while ( strcmp(buff, buff1) && controllaLunghezza(buff,0, 4, 16));

return buff;

}

char\* strremove(char \*str, const char \*sub) {

char \*p, \*q, \*r;

if ((q = r = strstr(str, sub)) != NULL) {

size\_t len = strlen(sub);

while ((r = strstr(p = r + len, sub)) != NULL) {

memmove(q, p, r - p);

q += r - p;

}

memmove(q, p, strlen(p) + 1);

}

str[strlen(str)-1]='\0';

return str;

}

void printTimeVolo(int minuti){

if ( (minuti / 60) == 1 ) /\* display 'x' hour... \*/

printf("%01d hour",minuti/60);

else /\* display 'x' hours .... \*/

printf("%01d hours",minuti/60);

if ( (minuti % 60) == 1) /\* display 'x' minute... \*/

printf(" %01d minute\n", minuti%60);

else /\* display 'x' minutes....\*/

printf(" %01d minutes\n",minuti%60);

}

char \* concatenation(char \*a, char \*b){

char \*c = malloc(strlen(a)+strlen(b)+2);

strcpy(c,a);

strcat(c,b);

return c;

}

void upperCase(char \* str) {

char \* name;

name = strtok(str,":");

// Convert to upper case

char \*s = name;

while (\*s) {

\*s = toupper((unsigned char) \*s);

s++;

}

}

ARRAY.C

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "array.h"

#include "list.h"

#include <string.h>

#include <ctype.h>

void inizializaMappa(char mappa[20][20])

{

for (int i = 0; i < 20; i++)

{

for (int j = 0; j < 20; j++)

{

mappa[i][j] = '-';

}

}

}

void initColoriLettere(char arrayLettera[26], int arrayColori[26], int arrayDisp[26]){

int j = 21;

for (int i = 0; i < 26; i++)

{

arrayLettera[i] = i + 65;

arrayDisp[i] = 0;

if (i < 6)

arrayColori[i] = i + 9;

else

{

arrayColori[i] = j;

j += 10;

}

}

}

int getIndiceDispForLetteraColore(int arrayDisp[26]){

for (int i = 0; i < 26; i++)

{

if (arrayDisp[i] == 0)

return i;

}

return -1;

}

int getPosizoneInizialeCodificata(char mappa[20][20]){

srand(time(NULL));

int dim;

int array[400];

dim = riempiArrayConPosizioniLibereCodificate(mappa,array);

if (dim == 0)

return -1;

int indice = rand() % (dim);

int x, y;

return array[indice];

}

int riempiArrayConPosizioniLibereCodificate(char mappa[20][20], int \*array)

{

int k = 0;

for (int i = 0; i < 20; i++)

{

for (int j = 0; j < 20; j++)

{

if (mappa[i][j] == '-')

{

\*array = (i \* 20) + j;

array = array + 1;

k++;

}

}

}

return k;

}

void stampaMappa(char mappa[20][20])

{

for (int i = 0; i < 20; i++)

{

for (int j = 0; j < 20; j++)

{

printf(" ");

printf("%c", mappa[i][j]);

printf(" ");

}

printf("\n");

}

}

//LIST.C

#include "list.h"

// gcc -o lista list.c array.h array.c ../checkInput.h ../checkInput.c ../lib/inputReader.h ../lib/inputReader.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FUNZIONI PER LA STRUTTURA User \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

User initNodeUser(char\* username, int x, int y, char\* address, int coloreAssegnato, char letteraAssegnata, int sd\_client){

User L = (User)malloc(sizeof(struct TUser));

L->username = username;

L->x = x;

L->y = y;

L->nTerritoriPosseduti = 0 ;

L->coloreAssegnato = coloreAssegnato ;

L->letteraAssegnata = letteraAssegnata ;

L->address = address ;

L->sd\_client = sd\_client;

L->game = NULL;

L->next = NULL;

return L;

}

User appendNodeUser(User head, User newUser) {

if (head != NULL) {

head->next = appendNodeUser(head->next, newUser);

}

else {

return newUser;

}

return head;

}

User removeUserByUsername(User head, char\* username) {

if (head != NULL) {

if (strcmp(head->username, username)==0) {

User tmp = head->next;

free(head);

return tmp;

}

head->next = removeUserByUsername(head->next, username);

}

return head;

}

void freeUserList(User head) {

if (head != NULL) {

freeUserList(head->next);

free(head);

}

}

void printUserList(User head) {

if (head != NULL) {

printf("username:%s\n", head->username);

printf("x:%d\n", head->x);

printf("y:%d\n", head->y);

printf("Numero territori posseduti:%d\n", head->nTerritoriPosseduti);

printf("colore assegnato:%d\n", head->coloreAssegnato);

printf("lettera assegnata:%c\n", head->letteraAssegnata);

printf("Ip adress:%s\n", head->address);

printf("file sd\_client:%d\n\n", head->sd\_client);

printUserList(head->next);

}

}

int updateUserTerritoriByLettera(User head, char letteraAssegnata, int cambiamento){

User tmp = head;

while(tmp != NULL){

if(tmp->letteraAssegnata != 0 && tolower(tmp->letteraAssegnata) == tolower(letteraAssegnata)){

tmp->nTerritoriPosseduti = tmp->nTerritoriPosseduti + cambiamento;

return 1;

}

tmp = tmp->next;

}

return 0;

}

int isLogged(User head, char\* username){

User tmp = head;

while(tmp != NULL){

if(tmp->username!=NULL && strcmp(tmp->username, username) == 0){

// trovato username connesso

return 1;

}

tmp = tmp->next;

}

return 0;

}

char\* getUsernameByLetteraNelGioco(User head, char lettera, int idGame){

User tmp = head;

char\* username = malloc(sizeof(char)\*36);

while(tmp){

if(tolower(tmp->letteraAssegnata) == tolower(lettera) && tmp->game->idGame == idGame){

strcpy(username, tmp->username);

break;

}

tmp = tmp->next;

}

return username;

}

void rimozioneUtenteDallaMappa(User localUser){

char letteraUser = tolower(localUser->letteraAssegnata);

for (int i = 0; i < 19; i++)

{

for (int j = 0; j < 19; j++)

{

if (tolower(localUser->game->mappa[i][j]) == letteraUser)

{

localUser->game->mappa[i][j] = '-';

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FINE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FUNZIONI PER LA STRUTTURA Game \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Game initNodeGame(unsigned int tempo\_secodi, int numeroDiTerritoriDaConquistare){

//srand(time(NULL));

Game G = (Game)malloc(sizeof(struct TGame));

G->count = 0;

inizializaMappa(G->mappa);

initColoriLettere(G->arrayLettera,G->arrayColori, G->arrayDisp);

G->idGame = 1 + rand()% RAND\_MAX;

G->partitaInCorso = 0;

G->tempo\_gioco = tempo\_secodi;

G->numeroDiTerritoriDaConquistare = numeroDiTerritoriDaConquistare;

G->vincitore = malloc(sizeof(char)\*64);

G->utenti = NULL;

G->next = NULL;

return G;

}

Game appendNodeGame(Game head, Game G){

if (head != NULL) {

head->next = appendNodeGame(head->next, G);

}

else {

return G;

}

return head;

}

Game removeNodeGameById(Game G, int idGame){

if (G != NULL) {

if (G->idGame == idGame){

Game tmp = G->next;

freeListUserIG(G->utenti);

free(G->vincitore);

free(G);

return tmp;

}

G->next = removeNodeGameById(G->next, idGame);

}

return G;

}

Game findGameDisponibile(Game G){

Game tmp = G;

while(tmp != NULL){

if(tmp->partitaInCorso == 1){

return tmp;

}

tmp = tmp->next;

}

return NULL;

}

void startGameTimer(Game G, pthread\_mutex\_t\* mutex, pthread\_cond\_t\* cond){

if (G->partitaInCorso == 0)

{

pthread\_mutex\_lock(mutex);

G->partitaInCorso = 1;

pthread\_cond\_signal(cond);

pthread\_mutex\_unlock(mutex);

}

}

void printGameInfo(Game G) {

if (G != NULL) {

printf("\n\n\*\*\*\*\* INFORMAZIONI SUL GIOCO \*\*\*\*\*\n");

stampaMappa(G->mappa);

for(int i = 0; i < 26; i++){

printf("%c\t", G->arrayLettera[i]);

}

printf("\n");

for(int i = 0; i < 26; i++){

printf("%d\t", G->arrayColori[i]);

}

printf("\n");

for(int i = 0; i < 26; i++){

printf("%d\t", G->arrayDisp[i]);

}

printf("\n");

printf("[idGame]:%d\n", G->idGame);

printf("[paritaInCorso]:%d\n", G->partitaInCorso);

printf("[numeroGiocato]:%d\n", G->count);

printf("[tempo gioco]:%d\n", G->tempo\_gioco);

printf("[Numero di Territori da conquistare]:%d\n", G->numeroDiTerritoriDaConquistare);

printf("[Vincitore]:%s\n", G->vincitore);

printAllUserIG(G->utenti);

printf("\n\n");

}

}

void printAllGameInfo(Game G) {

if(G != NULL){

printGameInfo(G);

printAllGameInfo(G->next);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FINE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FUNZIONI PER LA STRUTTURA UserIG \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

UserIG initNodeUserIG(char\* username){

UserIG G = (UserIG)malloc(sizeof(struct TUserInsideGame));

G->username = malloc(sizeof(char)\*32);

strcpy(G->username, username);

G->nTerritoriPosseduti = 0;

G->next = NULL;

}

UserIG appendNodeUserIG(UserIG head, UserIG G){

if (head != NULL) {

head->next = appendNodeUserIG(head->next, G);

}

else {

return G;

}

return head;

}

UserIG removeNodeUserIGbyUsername(UserIG head, char\* username){

if (head != NULL) {

if (strcmp(head->username,username)== 0){

UserIG tmp = head->next;

free(head);

return tmp;

}

head->next = removeNodeUserIGbyUsername(head->next, username);

}

return head;

}

void freeListUserIG(UserIG head){

if(head != NULL){

freeListUserIG(head->next);

free(head->username);

free(head);

}

}

void incrementaNumTerritoriPosseduti(UserIG head, char\* username){

UserIG tmp = head;

while(tmp){

if (strcmp(tmp->username,username)== 0){

tmp->nTerritoriPosseduti++;

break;

}

tmp = tmp->next;

}

}

void decrementaNumTerritoriPosseduti(UserIG head, char\* username){

UserIG tmp = head;

while(tmp){

if (strcmp(tmp->username,username)== 0){

tmp->nTerritoriPosseduti--;

break;

}

tmp = tmp->next;

}

}

void MergeSort(UserIG\* headRef){

UserIG head = \*headRef;

UserIG a;

UserIG b;

if((head == NULL) || (head->next == NULL)){

return;

}

FrontBackSplit(head, &a, &b);

MergeSort(&a);

MergeSort(&b);

\*headRef = ordinaPerNumeroTerritori(a,b);

}

UserIG ordinaPerNumeroTerritori(UserIG a, UserIG b){

UserIG result = NULL;

if(a == NULL)

return (b);

else if(b == NULL)

return (a);

if(a->nTerritoriPosseduti >= b->nTerritoriPosseduti){

result = a;

result->next = ordinaPerNumeroTerritori(a->next, b);

}

else{

result = b;

result->next = ordinaPerNumeroTerritori(a, b->next);

}

return result;

}

void FrontBackSplit(UserIG source, UserIG\* frontRef, UserIG\* backRef){

UserIG fast;

UserIG slow;

slow = source;

fast = source->next;

while(fast != NULL){

fast = fast->next;

if(fast != NULL){

slow = slow->next;

fast = fast->next;

}

}

\*frontRef = source;

\*backRef = slow->next;

slow->next = NULL;

}

void printAllUserIG(UserIG head){

if(head != NULL){

printf("[username]:%s\n", head->username);

printf("[numero territori conquistati]:%d\n", head->nTerritoriPosseduti);

printAllUserIG(head->next);

}

}

char\* getConnectedUsersInGame(UserIG head){

UserIG tmp = head;

int first = 1; // flag per descriminare la forma del messaggio

char\* buf = malloc(sizeof(char)\*1024);

char\* numTerr = malloc(sizeof(char)\*3);

char lettera;

while(tmp != NULL){

if(tmp->username != NULL){

sprintf(numTerr, "%d", tmp->nTerritoriPosseduti);

if(first == 1){

buf = concatenation(tmp->username, concatenation(":", numTerr));

memset(numTerr, 0, 3);

first = 0;

}

else{

buf = concatenation(buf, "-");

buf = concatenation(buf, concatenation(tmp->username, concatenation(":", numTerr)));

memset(numTerr, 0, 3);

}

}

tmp = tmp->next;

}

// user1:10-user2:20-user3:12

return buf;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FINE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FUNZIONI PER LA STRUTTURA Client \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Client initNodeClient(char\* adress, int sd\_client){

Client C = (Client)malloc(sizeof(struct TClient));

C->adress = adress;

C->sd\_client = sd\_client;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FINE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//INPUTREADER.C

#include <stdio.h>

#include <stdlib.h>

#include "inputReader.h"

void clearBuffer(){

char c;

while ((c = getchar()) != '\n' && c != EOF) { };

}

int getPositive(int \*data){

int ok, // ritorno della funzione

i; // indice per scorrere la stringa letta

char buffer[51];

scanf("%50s",buffer);

clearBuffer();

i = 0;

while( buffer[i]>47 && buffer[i]<58 ) i++;

ok = (buffer[i]=='\0' && i>0 );

if(ok) \*data = atoi(buffer);

return ok;

}

int getPositiveDouble(double \*data){

int ok=0; // ritorno della funzione

fflush(stdin);

if(scanf("%lf", data) == 1){

ok=1;

}else

printf("input non valido\n");

return ok;

}

int getFloat(float \*f){

char buffer[51];

int cont, //indice per lo scorrimento dell'input utente

off, //lunghezza minima della stringa (in base alla presenza del punto e/o del segno negativo)

ret;

scanf("%50s",buffer);

clearBuffer();

cont = off = (buffer[0]=='-') ? 1 : 0;

while(buffer[cont]>47 && buffer[cont]<58) cont++;

if(buffer[cont]=='.'){

cont++;

off++;

while(buffer[cont]>47 && buffer[cont]<58) cont++;

}

ret = (buffer[cont]=='\0' && cont>off);

if(ret)

\*f = atof(buffer);

return ret;

}

int getInt(int \*data){

int ok, // ritorno della funzione

i, // indice per scorrere la stringa letta

negative; // indica se il numero è negativo

char buffer[51];

scanf("%50s",buffer);

clearBuffer();

i = negative = (buffer[0]=='-') ? 1 : 0 ;

while( buffer[i]>47 && buffer[i]<58 ) i++;

ok = (buffer[i]=='\0' && i>negative );

if(ok) \*data = atoi(buffer);

return ok;

}