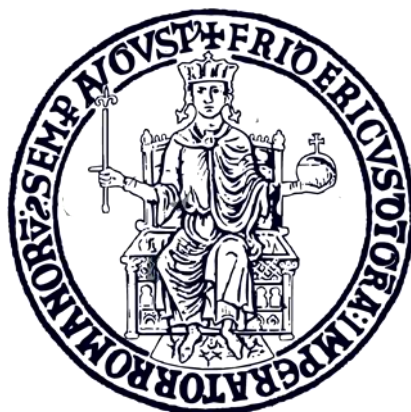


Università degli Studi di Napoli Federico II



Corso di Laurea in Informatica
Insegnamento di Laboratorio di Sistemi Operativi
Anno accademico 2017/2018

Autore:

Carlo Mennella

Matricola N86001552

Docente :

Alberto Finzi

INDICE

1. Manuale d'uso dell'applicazione

1.1 Esecuzione dei sorgenti

1.2 Guida all'applicativo

1.3 Analisi Client

2. Strutture dati comunicazione client server

3. Dettagli Implementativi

4. Appendice

4.1 Server.c

4.2 Client.c

1. Manuale d'uso.

L'applicativo è formato da due file sorgente:

- **server.c**
- **client.c**

Il server svolgerà tutte le funzioni tipiche di un server:

- Gestisce i client che si connettono ad esso per richiedergli dei servizi
- Gestisce gli strumenti di comunicazione

In particolare i servizi forniti da parte del server al client sono i seguenti:

- Login alla piattaforma
- Registrazione alla piattaforma
- Spostamento in una delle 4 direzioni (Nord, Sud, Est, Ovest)
- Logout
- Visualizzazione degli utenti collegati
- Visualizzazione degli ostacoli incontrati da tutti gli utenti partecipanti alla partita
- Posizione dei soldati alleati con relativa vita rimanente
- Stampa della mappa di gioco

Il client permetterà all'utente di usufruire dei servizi offerti dal server.

1.1 Compilazione dei sorgente:

Per il server lanceremo il seguente comando nella shell:

gcc server.c -o server -pthread

Per il client:

gcc client.c -o client -pthread

1.2 Esecuzione:

Dopo aver compilato eseguiamo il server digitando sempre nel terminale

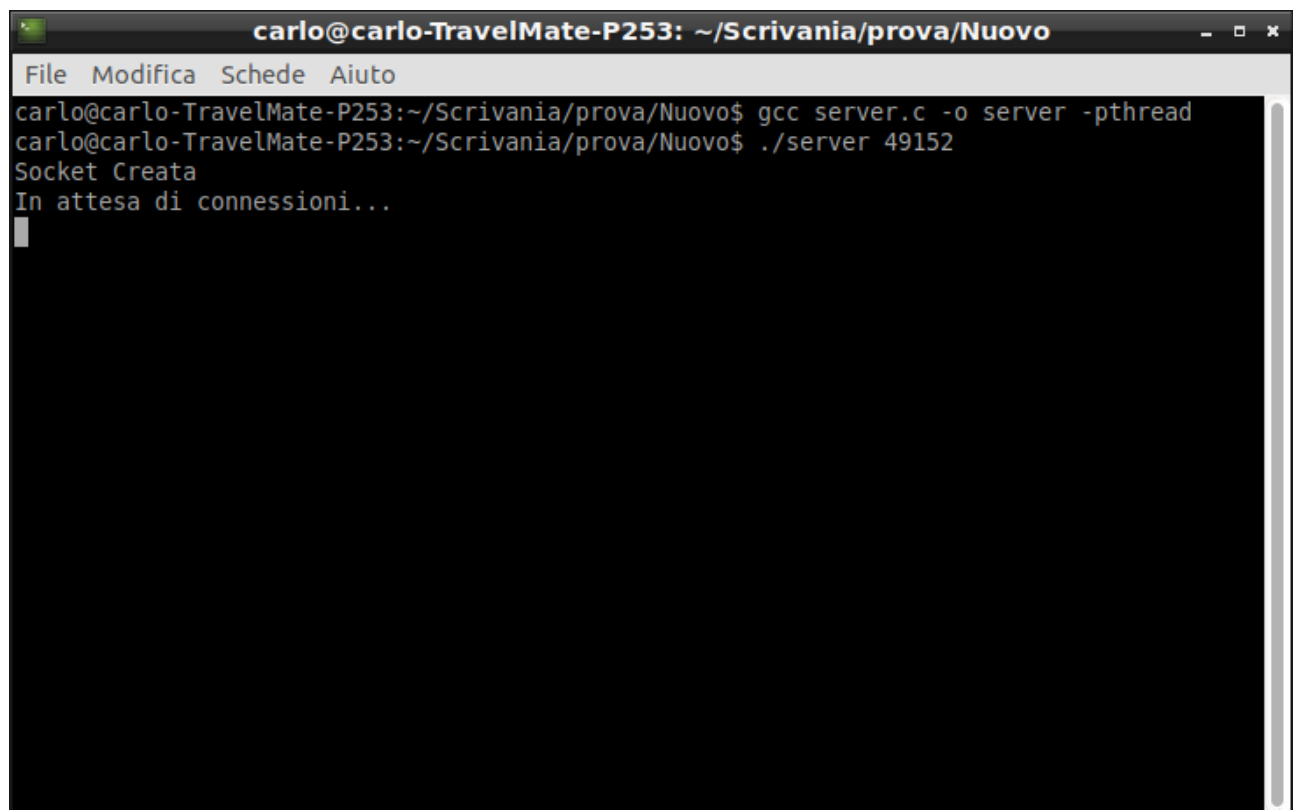
`./server <porta>`

La porta è un valore alfanumerico, su cui il server si pone in ascolto.

Ricordiamo agli utenti che non tutte le porte sono uguali, in particolare:

- Le porte comprese tra 0 e 1023 sono riservate
- Le porte comprese tra 1024 e 49151 contengono dei servizi registrati
- Le porte comprese tra 49152 e 65535 sono porte libere

Se l'esecuzione andrà a buon fine ci verrà mostrata la seguente schermata:

A screenshot of a terminal window titled "carlo@carlo-TravelMate-P253: ~/Scrivania/prova/Nuovo". The terminal shows the following commands and output:

```
carlo@carlo-TravelMate-P253:~/Scrivania/prova/Nuovo$ gcc server.c -o server -pthread
carlo@carlo-TravelMate-P253:~/Scrivania/prova/Nuovo$ ./server 49152
Socket Creata
In attesa di connessioni...
```

The terminal has a menu bar with "File", "Modifica", "Schede", and "Aiuto". The output shows the successful compilation of the server and its execution on port 49152, where it is now waiting for connections.

Nel caso in cui la porta sia utilizzata verrà stampato un messaggio di errore:

Bind Fallita: Address already in use

Successivamente lanciamo in un nuovo terminale il client

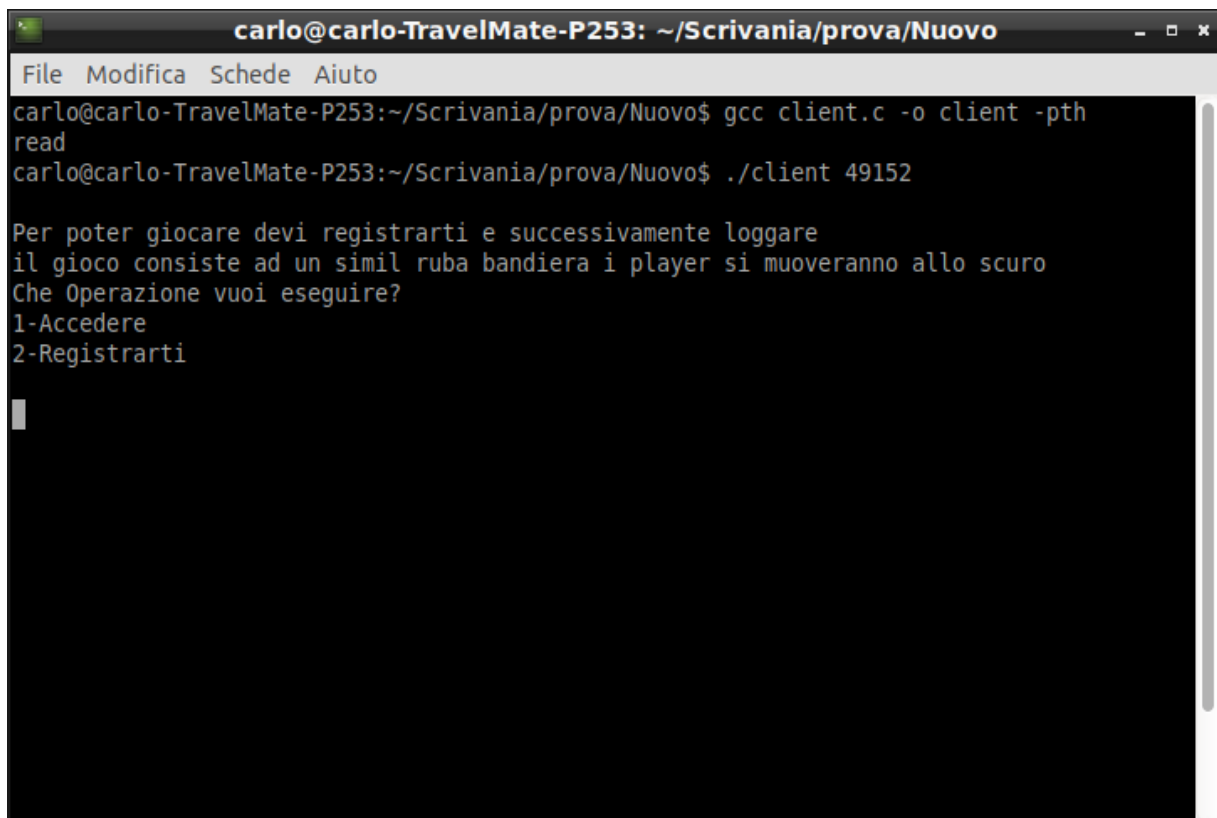
`./client <porta> <address>`

In <porta> inseriamo il valore della porta su cui il server si è posto in ascolto

Ovvero quella scelta nell'esecuzione del server.

In <address> inseriamo l'indirizzo del nostro server, tuttavia essendo il server locale, possiamo omettere questo parametro, che verrà settato automaticamente a 127.0.0.1

Se l'esecuzione andrà a buon fine ci verrà mostrata la schermata:



```
carlo@carlo-TravelMate-P253: ~/Scrivania/prova/Nuovo
File Modifica Schede Aiuto
carlo@carlo-TravelMate-P253:~/Scrivania/prova/Nuovo$ gcc client.c -o client -pth
read
carlo@carlo-TravelMate-P253:~/Scrivania/prova/Nuovo$ ./client 49152

Per poter giocare devi registrarti e successivamente loggare
il gioco consiste ad un simil ruba bandiera i player si muoveranno allo scuro
Che Operazione vuoi eseguire?
1-Accedere
2-Registrarti
█
```

1.3 Analisi Client

Le scelte possibili all'avvio del client sono 2:

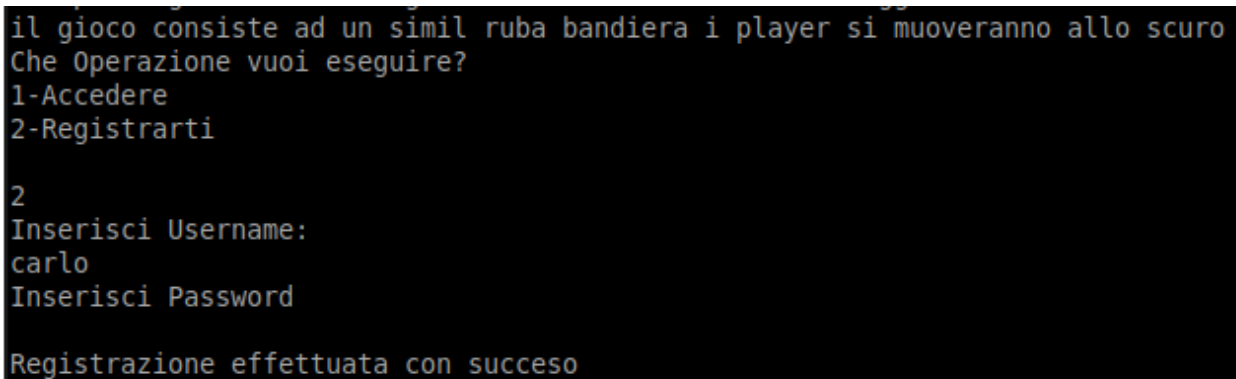
- Accedere
- Registrarsi

1.3.1 Registrazione:

Digitando 2 viene richiesto di inserire un username, se questo è già presente all'interno della piattaforma, viene richiesto un nuovo username da inserire.

Successivamente viene richiesto l'inserimento della password. (Non visibile)

Se la registrazione viene effettuata con successo ci verrà mostrata la seguente schermata



```
il gioco consiste ad un simil ruba bandiera i player si muoveranno allo scuro
Che Operazione vuoi eseguire?
1-Accedere
2-Registrarti

2
Inserisci Username:
carlo
Inserisci Password

Registrazione effettuata con successo
```

1.3.2 Accesso:

Digitando 1 viene richiesto di inserire l'username e successivamente la password.

Se i due campi inseriti risultano essere corretti ed è la prima volta che il giocatore entra in partita, si procederà con l'inizializzazione di esso.

Viene richiesto l'esercito di appartenenza:

- 1 - Blu
- 2 – Rosso

```
carlo@carlo-TravelMate-P253: ~/Scrivania/prova/Nuovo
File Modifica Schede Aiuto

Inserisci l'esercito di appartenenza.
1 - BLU
2 - ROSSO

1
Attendere altri giocatori ...
```

Il server attenderà almeno un giocatore per esercito

Quando ci sono 2 giocatori di squadre diverse il server chiederà il posizionamento

1.3.3 Inizializzazione Giocatore:

```
carlo@carlo-TravelMate-P253: ~/Scrivania/prova/Nuovo
File Modifica Schede Aiuto

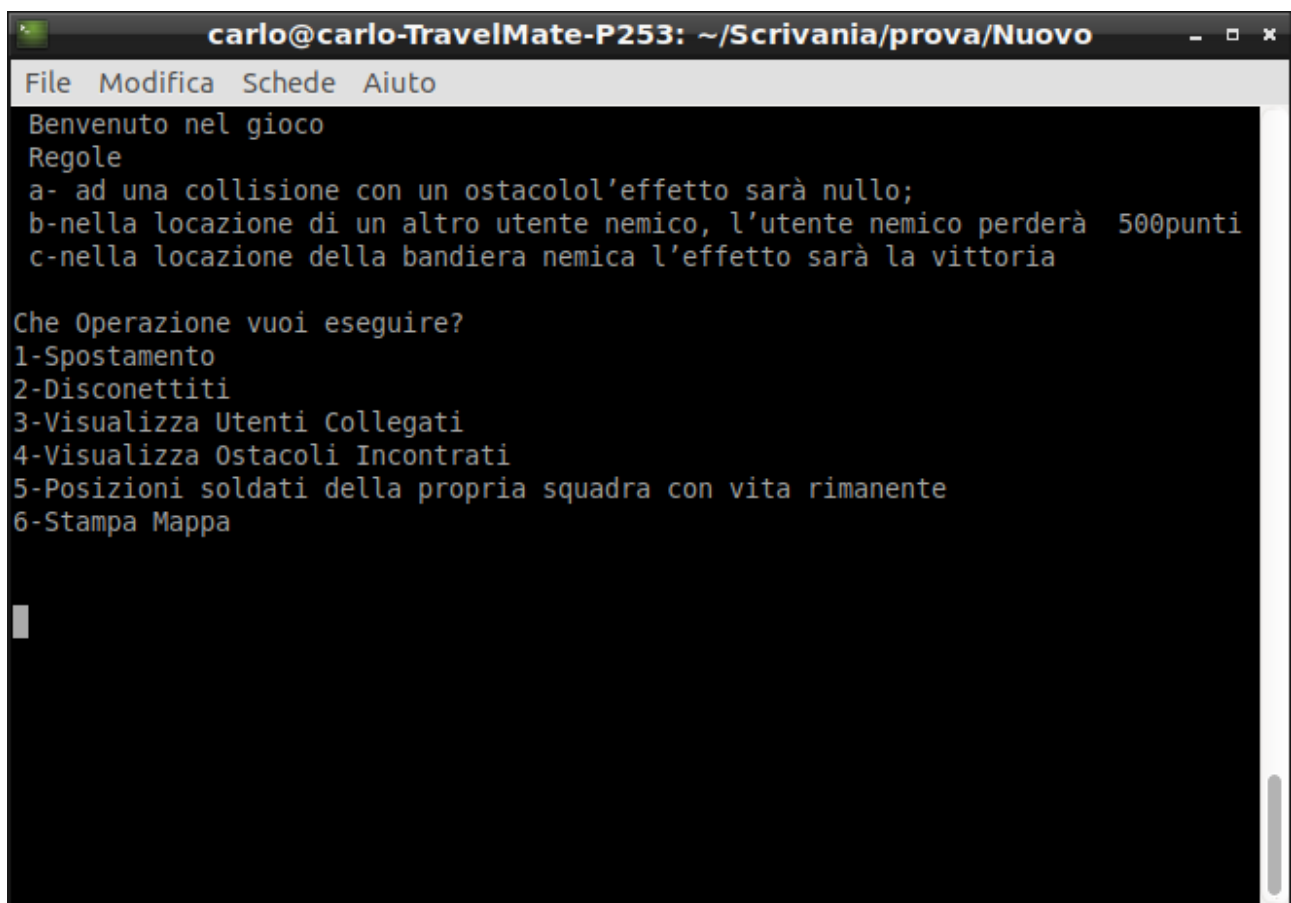
Inserisci l'esercito di appartenenza.
1 - BLU
2 - ROSSO

2
Attendere altri giocatori ...

Mappa Della Partita:
x 0 1 2 3 4
0 - - - - -
1 - - - - -
2 - - - - -
3 - - - - -
4 - - - - -

Inserisci Riga dove posizionare il giocatore
3
Inserisci Colonna dove posizionare il giocatore
2
```

1.3.4 Menù di gioco:

A screenshot of a terminal window titled "carlo@carlo-TravelMate-P253: ~/Scrivania/prova/Nuovo". The window has a menu bar with "File", "Modifica", "Schede", and "Aiuto". The text in the terminal is as follows:

```
Benvenuto nel gioco
Regole
a- ad una collisione con un ostacolol'effetto sarà nullo;
b-nella locazione di un altro utente nemico, l'utente nemico perderà 500punti
c-nella locazione della bandiera nemica l'effetto sarà la vittoria

Che Operazione vuoi eseguire?
1-Spostamento
2-Disconettiti
3-Visualizza Utenti Collegati
4-Visualizza Ostacoli Incontrati
5-Posizioni soldati della propria squadra con vita rimanente
6-Stampa Mappa
```

Le possibili operazioni previste dall'applicativo sono rappresentate nell'immagine sopra illustrata.

Spostamento:

Digitando 1 è possibile spostarsi all'interno della mappa in una delle 4 direzioni:

- Nord
- Sud
- Est
- Ovest

Le possibili conseguenze di uno spostamento sono:

-collisione con un ostacolo: l'effetto sarà nullo, l'utente perde 1 punto vita e rimane nella posizione precedente.

-collisione con utente di una squadra avversaria: L'utente perde 1 punto vita e l'avversario perde 500 punti vita, inoltre per garantire l'integrità del gioco, l'utente avversario viene temporaneamente eliminato dalla mappa e verranno salvate le informazioni riguardanti la sua precedente posizione.

Tale implementazione è stata pensata per impedire eliminazioni scorrette degli avversari con mosse del tipo Nord - Sud - Est - Ovest o viceversa.

-posizione vuota: l'utente si sposta nella nuova posizione e perde 1 punto vita.

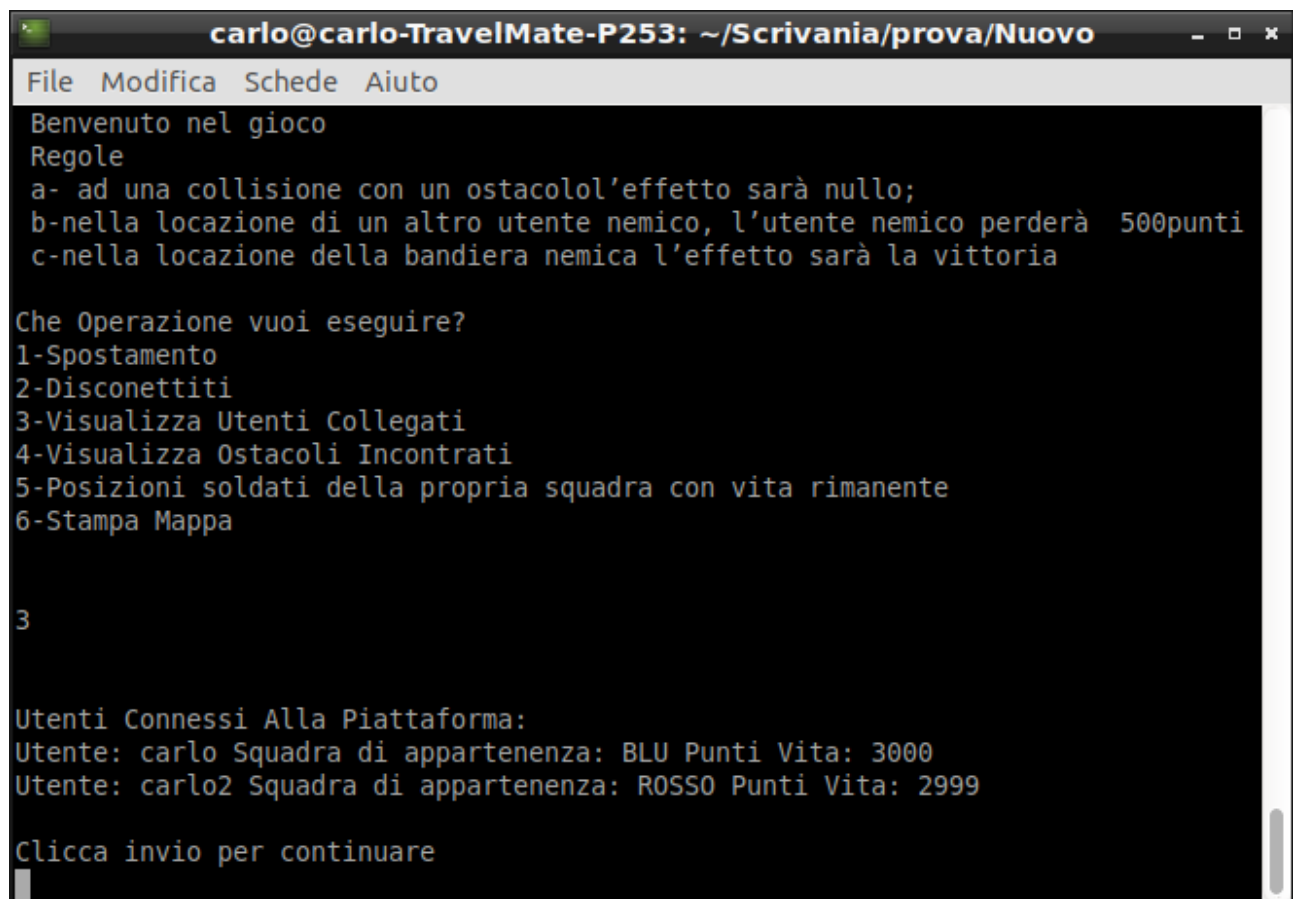
-bandiera avversaria: l'utente vince la partita e tutti i client vengono notificati del risultato della partita.

Disconnessione:

L'utente viene disconnesso e viene visualizzata la pagina iniziale.

Visualizza utenti collegati:

Viene stampato a video la lista degli utenti collegati alla piattaforma con squadra di appartenenza e punti vita.



```
carlo@carlo-TravelMate-P253: ~/Scrivania/prova/Nuovo
File Modifica Schede Aiuto
Benvenuto nel gioco
Regole
a- ad una collisione con un ostacolol'effetto sarà nullo;
b-nella locazione di un altro utente nemico, l'utente nemico perderà 500punti
c-nella locazione della bandiera nemica l'effetto sarà la vittoria

Che Operazione vuoi eseguire?
1-Spostamento
2-Disconnettiti
3-Visualizza Utenti Collegati
4-Visualizza Ostacoli Incontrati
5-Posizioni soldati della propria squadra con vita rimanente
6-Stampa Mappa

3

Utenti Connessi Alla Piattaforma:
Utente: carlo Squadra di appartenenza: BLU Punti Vita: 3000
Utente: carlo2 Squadra di appartenenza: ROSSO Punti Vita: 2999

Clicca invio per continuare
```

Visualizza ostacoli incontrati:

Viene stampata a video una pseudo rappresentazione della mappa di gioco, in cui con una x vengono rappresentati gli ostacoli incontrati dagli utenti durante la partita.

```
carlo@carlo-TravelMate-P253: ~/Scrivania/prova/Nuovo
File Modifica Schede Aiuto

Che Operazione vuoi eseguire?
1-Spostamento
2-Disconnettiti
3-Visualizza Utenti Collegati
4-Visualizza Ostacoli Incontrati
5-Posizioni soldati della propria squadra con vita rimanente
6-Stampa Mappa

4

Stampo la mappa degli ostacoli incontrati:
-x: Rappresenta posizione in cui e' stato incontrato un ostacolo
x 0 1 2 3 4
0 - - - - -
1 - - - - -
2 - - - - -
3 - - - - x
4 - - x - -

Clicca invio per continuare
```

Posizione soldati della propria squadra:

Viene stampato a video la posizione dei soldati alleati con i relativi punti vita

Stampa Mappa:

Viene stampata la mappa di gioco con una o che rappresenta la posizione attuale del giocatore.

```
carlo@carlo-TravelMate-P253: ~/Scrivania/prova/Nuovo
File Modifica Schede Aiuto
c-nella locazione della bandiera nemica l'effetto sarà la vittoria

Che Operazione vuoi eseguire?
1-Spostamento
2-Disconnettiti
3-Visualizza Utenti Collegati
4-Visualizza Ostacoli Incontrati
5-Posizioni soldati della propria squadra con vita rimanente
6-Stampa Mappa

6

Mappa Della Partita:
x 0 1 2 3 4
0 - - - - -
1 - - - - -
2 - - 0 - -
3 - - - - -
4 - - - - -

Clicca invio per continuare
```

Vittoria di un giocatore:

Una volta che un giocatore prende una bandiera avversaria la partita termina con la vittoria del giocatore in questione, il quale viene disconnesso. Per creare una nuova partita gli altri giocatori dovranno effettuare la disconnessione manuale, a quel punto sono pronti per giocare una nuova partita.

2. Strutture dati comunicazione client server

Per la comunicazione tra client e server è stata utilizzata una struttura dati di tipo TCP attraverso una AF_INET + SOCK_STREAM, che fornisce una connessione sequenziale, affidabile e full-duplex.

Per ogni client che accede alla piattaforma il server crea un thread per la sua gestione. Tale thread si mette in attesa di eventuali richieste da parte del client attraverso una lettura ciclica della sua socket di connessione.

Inoltre verranno memorizzate in una apposita struttura dati le informazioni riguardanti il numero di socket del client connesso, al fine di poter notificare correttamente tutti i client quando una partita sarà terminata.

Lo scambio di informazioni tra client e server avviene attraverso le primitive:

- **int send (int sock, char *buffer, int len, int flags)**
attraverso cui il client invia un messaggio al server, indicando la socket tramite la quale si svolge la comunicazione con il server, il messaggio da inviare, la dimensione del messaggio in byte, ed il valore del flag che viene solitamente posto a 0
- **int recv (int sock, char *buffer, int len, int flags)**
attraverso cui il server riceve il messaggio indicando la socket con cui si svolge la comunicazione con il client, la variabile in cui viene posto il messaggio in arrivo, la dimensione in byte del messaggio e il flag che viene solitamente posto a 0

Inoltre il server è di tipo concorrente ed è in grado di gestire più client simultaneamente.

3. Dettagli Implementativi

Vengono di seguito descritti alcuni dettagli implementativi dell'applicativo.

La concorrenza del server viene realizzata attraverso la creazione di un thread per ogni richiesta ricevuta da un client, da come si può evincere dal seguente frammento di codice:

```
while(1){
    leggibile = localtime(&rawtime);
    n=sizeof(client);

    if((client_sock=accept(socket_desc,(struct sockaddr *)&client[visits],&n))< 0){
        perror("accept failed\n");
        exit(1);
    }

    sockd[visits]=client_sock;
    visits++;

    log=fopen("log.txt","a");

    inet_ntop(AF_INET,&client[visits].sin_addr,str,INET_ADDRSTRLEN);
    if(visits%2==0){
        fprintf(log,"%d/%s/%d %d:%d:%d Nuova connessione da client %2s \n",leggibile-
>tm_mday,mesi[leggibile->tm_mon],leggibile->tm_year + 1900,leggibile-
>tm_hour,leggibile->tm_min,leggibile->tm_sec,inet_ntoa(client[visits-1].sin_addr));
    }

    fclose(log);

    if(pthread_create(&tid,NULL,(void *)richiesta,&client_sock)){
        perror("Impossibile avviare un thread");
        return 0;
    }
    pthread_detach(tid);

    if(client_sock<0){
        perror("Accettazione fallita");
        return 1;
    }
}
```

Per la creazione della mappa di gioco sono state utilizzate diverse accortezze:

La mappa viene generata in maniera random ad ogni partita e vengono inseriti una percentuale di muri pari ad un numero compreso tra il 10% e il 20% della cardinalità della mappa.

Inoltre per evitare situazioni di stallo durante la generazione random della mappa di gioco, è stata creata una funzione:

controlloPercorso(struct mappa **a,int riga, int colonna)

Che prima dell'inserimento di un muro all'interno della mappa controlla se questo possa creare successivamente una posizione di stallo.

Per quanto riguarda la notifica del risultato di gioco, questa si è resa possibile salvando ad ogni connessione il numero di socket del client connesso

```
sockd[visits]=client_sock;
```

```
visits++
```

E istanziando un thread per ogni connessione effettuata al server il quale resta in attesa di ricevere un messaggio come si evince dal seguente codice:

```
void mex(int *sock){
    char risposta[70];
    while(1){
        if(flag==10){
            read(*sock,risposta,70);
            if(strcmp(risposta,"") > 10){
                puts(risposta);
                puts("\n\nClicca un tasto per sloggarti\nPotrai Loggarti nuovamente e
partecipare a una nuova partita");
                flag=1;
            }
        }
    }
}
```

E' stata creata una funzione ciclica, che continua ad aspettare un giocatore per squadra, ed è inserita subito dopo la scelta del colore, questo serve per evitare di giocare da soli.

Il file di log user e mappa sono creati in formato .txt per un accesso semplice tramite le funzioni standard C/C++

4. Appendice.

Di seguito i codici del server e client

4.1 Server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <time.h>
#define N 5 //Numero di righe
#define M 5 //Numero di colonne
#define Z N*M+N*2+M //Numero di elementi della mia matrice
int visits=0;
int sockd[220];
char *mesi[] = {"Gennaio", "Febbraio", "Marzo", "Aprile", "Maggio", "Giugno", "Luglio",
"Agosto", "Settembre", "Ottobre", "Novembre", "Dicembre"};
int vittoria=0;
char userVittoria[24];
char vittoriaMessage[1024];
struct mappa **matrice;
struct utenti *top=NULL;
struct utenti *nuovo(struct utenti *top,char *user, char *pw);
struct utenti *nuovomappa(struct utenti *mappa,char *user, char *pw,int color);
struct mappa **nuovaMappa(struct mappa **a,char *user,char *pw,int i, int j);
struct mappa **inizializzaMatrice(struct mappa **a);
int getij(struct utenti *top, char *user, int opzione);
int getcolor(struct utenti *top,char *user);
void setij(struct utenti *top,char *user,int i ,int j);
int int_rand(int min, int max);
int inserisciBandiere(struct mappa **a, int riga, int colonna);
void riempiMatrice(struct mappa **a);
int inizializzaGiocatore(struct utenti *top,char *user, int i);
struct utenti *inserisciColore(struct utenti *top, int color, char *user);
int inserisciSoldato(struct mappa **a,char *user,char *passw,int i , int j, int color,int k);
int ricerca(struct utenti *top,char *user);
void eliminatoFile(char *user);
int controlloPercorso( struct mappa **a,int riga, int colonna);
int login(struct utenti *top,char *user,char*pw);
void deallocaMatrice(struct mappa **a);
void vita(struct utenti *top, char *user,int i);
void eliminaVita(struct utenti *mappa,int color);
struct utenti *elimina(struct utenti *top,char *user);
int spostamento(struct mappa **a, char *user, char *passw, int c);
int start(struct utenti *top);
void disconnect(struct utenti *top,char *user);
void disconnectall(struct utenti *top);
void richiesta(int *socket_desc) ;
void scriviFile(int i);
```

```

void stampaMatrice(struct mappa **a);
void stampa(struct utenti *top);
char *utentiConnessi(struct utenti *top,char *users);
char *alleati(struct utenti *top,char *user,char *stringa);
void stampamatrix(struct utenti *top,char *matrix,char *username,int i1,int j1,int k);
char *stampaostacoli(struct mappa **a,char *matrix);

```

```

void win(char *mex,int sock){
    char numero[20];
    int i;
    for(i=0;i<=visits;i++){

        if(i%2!=0){
            if(sockd[i]!=sock){
                write(sockd[i],mex,120);
            }
        }
    }
}

```

//creo le strutture che mi servono matrice ,utenti

```

struct mappa{
    /*Valore conterrà informazioni riguardanti la partita, in particolare
    *   0 - Posizione Libera.
    *   1 - Posizione Occupata da uno ostacolo.
    *   2 - Posizione Occupata da una bandiera.
    *   3 - Posizione Occupata da un giocatore.
    */
    int valore;

    char *descrizione; // Variabile per maggiori informazioni riguardante il valore
    (es:colore bandiera)
    struct utenti *utente;
    //Eventuale contatore.
    int x;
};

```

//Struttura per memorizzare informazioni su gli utenti registrati alla piattaforma
//l'utente ha una password la vita che serve per restare in gioco il colore della
squadra di appartenenza

```

struct utenti {
    char username[30];
    char password[30];
    int vita;
    int colore; //1 - Blu ; 2 - Rosso ; 0 - Utente registrato ma ancora deve entrare in
gioco ; -1 Utente Eliminato.
    int i;
    int j;
    int connected;
    struct utenti *next;
};

```

```

//-----
-----

```

//creo gli algoritmi di settaggio delle informazioni


```

//Inserimento in coda di una lista aggiunge alla coda della lista un nuovo utente sem-
plicemente attacca alla fine della lista l'utente tmp
struct utenti *incoda(struct utenti *top, struct utenti *tmp){
    if(top==NULL){
        tmp->next=NULL;
        top=tmp;
        return tmp;
    }
    else {
        tmp=incoda(top->next,tmp);
        top->next=tmp;
        return top;
    }
}

int getij(struct utenti *top, char *user, int opzione){
    // opzione : 0 - Ritorno i , 1 - Ritorno j
    int x;
    if(opzione==0){
        while(top!=NULL){
            if(strcmp(top->username,user)==0){
                x=top->i;
            }
            top=top->next;
        }
    }
    if(opzione==1){
        while(top!=NULL){
            if(strcmp(top->username,user)==0){
                x=top->j;
            }
            top=top->next;
        }
    }
    return x;
}

//prende il colore di appartenenza a una squadra compara la stringa username e user che
indicano l'utente e preleva il colore
int getcolor(struct utenti *top,char *user){
    int x;
    while(top!=NULL){
        if(strcmp(top->username,user)==0){
            x=top->colore;
        }
        top=top->next;
    }
    return x;
}

void setij(struct utenti *top,char *user,int i ,int j){
    while(top!=NULL){
        if(strcmp(top->username,user)==0){
            top->i=i;
            top->j=j;
        }
        top=top->next;
    }
}

//funzione per prendere casualmente un numero in un intervallo
int int_rand(int min, int max){

```

```

    int numero_casuale;
    int differenza;
    differenza=(max-min)+1;
    numero_casuale=rand() % differenza;
    numero_casuale=numero_casuale+min;
    return numero_casuale;
}

//funzione per inserire la bandierina
int inserisciBandiere(struct mappa **a, int riga, int colonna){
    int i,j;
    int result=0;
    if(a[riga][colonna].valore==1 || a[riga][colonna].valore==2) result=1;
    return result;
}

void riempiMatrice(struct mappa **a){
    int muroMin,muroMax,muro,contatore=0;
    int i,j;
    //alcuni elementi sono caratterizzati da un muro
    muroMin=N*M*10/100;
    muroMax=N*M*20/100;
    srand((unsigned int) time(0));
    //Numero di caselle dedicate al muro da inserire nella mia matrice.
    muro=int_rand(muroMin,muroMax);
    //Posiziono ostacoli all'interno della mia matrice in maniera random
    while(contatore < muro){
        i= rand() % N; //Numero Random tra le righe
        j= rand() % M; //Numero Random tra le colonne
        if(controlloPercorso(a,i,j)==0){
            a[i][j].valore=1;
            contatore++;
        }
    }
    //Setto il contatore a 0
    contatore=0;
    //Procedo ad inserire le due bandiere in maniera random.
    while(contatore <2){
        i= rand() % N;
        j= rand() % M;
        if(inserisciBandiere(a,i,j)==0){
            a[i][j].valore=2;
            a[i][j].descrizione=(char*)malloc(sizeof(char));
            if(contatore==0){
                strcpy(a[i][j].descrizione,"Bandiera Blu");
            }
            if(contatore==1){
                strcpy(a[i][j].descrizione,"Bandiera Rossa");
            }
            contatore ++;
        }
    }
}

//Funzione per inserire un nuovo elemento all'interno della lista
struct utenti *nuovo(struct utenti *top,char *user, char *pw){
    struct utenti *tmp=(struct utenti *) malloc(sizeof(struct utenti));
    strcpy(tmp->username,user);
    strcpy(tmp->password,pw);
}

```

```

    tmp->colore=0;
    tmp->connected=0;
    tmp->next=NULL;
    top=incoda(top,tmp);
    return top;
}
//funzione per inserire una nuova mappa e relativi elementi ???
struct utenti *nuovomappa(struct utenti *mappa,char *user, char *pw,int color){
    struct utenti *tmp=(struct utenti *) malloc(sizeof(struct utenti));
    strcpy(tmp->username,user);
    strcpy(tmp->password,pw);
    tmp->colore=color;
    tmp->next=NULL;
    mappa=incoda(mappa,tmp);
    return mappa;
}

struct mappa **nuovaMappa(struct mappa **a,char *user,char *pw,int i, int j){
    int k;
    struct mappa **tmp=(struct mappa **) malloc (sizeof(struct mappa*));
    for(k=0;k<N;k++){
        tmp[k]=(struct mappa*) malloc(sizeof(struct mappa));
    }
    strcpy(tmp[i][j].utente->username,user);
    strcpy(tmp[i][j].utente->password,pw);
    tmp[i][j].utente->next=NULL;
    top=incoda(a[i][j].utente,tmp[i][j].utente);
    return a;
}

struct mappa **inizializzaMatrice(struct mappa **a){
    //Definisco Indice di riga e indice di colonna
    int i,j;
    //Alloco la mia matrice
    a=(struct mappa**)malloc(sizeof(struct mappa*)*N);
    for(i=0;i<N;i++){
        a[i]=(struct mappa*)malloc(sizeof(struct mappa)*M);
        for(j=0;j<M;j++){
            //Setto tutti gli elementi della mia matrice a 0 (Posizione Libera)
            a[i][j].valore=0;
            a[i][j].utente=NULL;
        }
    }
    return a;
}

int inizializzaGiocatore(struct utenti *top,char *user, int i){
    int flag=0;
    while(top!=NULL){
        //Scorro la mia lista fino a trovare l'utente che ha effettuato il login alla
        piattaforma
        if(strcmp(top->username,user)==0){
            if(i==0){
                //Verifico se non ha ancora effettuato l'inserimento sulla mappa
                if ( top->colore == 0 ){
                    flag=1;
                }
            }
            if(i==1 || i==2){
                top->colore=i;
            }
        }
    }
}

```

```

        top->vita=3000;
        flag=1;
    }

}

//Scorro la mia lista per trovare il mio username
top=top->next;
}
return flag;
}

struct utenti *inserisciColore(struct utenti *top, int color, char *user){
    while(top!=NULL){
        if(strcmp(top->username,user)==0){
            top->colore=color;
        }
        top=top->next;
    }
    return top;
}

int inserisciSoldato(struct mappa **a,char *user,char *passw,int i , int j, int color,int k){
    // 1 - Inserimento Non riuscito in quanto vi è un ostacolo
    // 2 - Inserimento Effettuato con successo.
    // 3 - Inserimento Effettuato con successo ed eliminato i punti vita a gli utenti
    di un altro colore.
    int flag=0;
    char colore[6];
    //Controllo che i valori passati siano compresi nel range delle mie variabili.
    if(i<N && j < M){
        //Controllo se la partita non è terminata
        if(a[i][j].valore==2){
            color=getcolor(top,user);
            if(color==1 && strcmp(a[i][j].descrizione,"Bandiera Rossa")==0 || color==2
&& strcmp(a[i][j].descrizione,"Bandiera Blu")==0){

                vittoria=1;
                flag=8;
                if(color==1){ strcpy(colore,"Blu"); scriviFile(1);}
                else{   strcpy(colore,"Rosso"); scriviFile(2);}
                strcpy(userVittoria,user);
                strcpy(vittoriaMessage,"La partita e stata vinta da: ");
                strcat(vittoriaMessage,user);
                strcat(vittoriaMessage,"\nLa squadra ");
                strcat(vittoriaMessage,colore);
                strcat(vittoriaMessage," ha vinto la partita.");
                strcat(vittoriaMessage,"\0");

            }
            else{
                a[i][j].utente=nuovomappa(a[i][j].utente,user,passw,color);
                setij(top,user,i,j);
                flag=2;
            }

        }

        //Controllo se vi è un muro

        else if(a[i][j].valore==1){
            if(k==0){

```

```

        a[i][j].utente=nuovomappa(a[i][j].utente,user,passw,color);
        setij(top,user,i,j);
        flag=2;
    }
    else{
        a[i][j].x=1;
        flag=1;
    }
}

//Altrimenti Procedo con l'inserimento.
else if(a[i][j].utente==NULL){
    //Inserisci una funzione per inserire all'interno anche il colore del mio
soldato
    a[i][j].utente=nuovomappa(a[i][j].utente,user,passw,color);
    setij(top,user,i,j);
    flag=2;
}
//Nel caso in cui vi è anche un altro soldato sulla mappa, controllo se sono
alleati o nemici
//Se sono alleati l'effetto sarà nullo, altrimenti il nemico perderà 500 punti
vita.
else if( a[i][j].utente!=NULL){
    //Scorro la lista degli utenti presenti in questa posizione
    while(a[i][j].utente!=NULL){
        //Verifico se ci sono utenti diversi da quello principale
        //Verifico il colore
        if(a[i][j].utente->colore!=color){
            //Procedo ad eliminare la vita
            eliminaVita(a[i][j].utente,color);
            flag=3;
        }

        a[i][j].utente=a[i][j].utente->next;
    }
    if(flag!=3) flag=2;

    a[i][j].utente=nuovomappa(a[i][j].utente,user,passw,color);
    setij(top,user,i,j);
}
}else flag=0;

return flag;
}

```

```

//-----
//-----
//algoritmi per la ricerca cancellazione e altro
//Ricerca di un utente all'interno della lista semplice scansione di una lista con confronto che riceve in ingresso la lista e un user ritorna uno se c'è corrispondenza
int ricerca(struct utenti *top,char *user){
    int i=0;
    struct utenti *tmp;
    tmp=top;
    while(tmp!=NULL){
        if(strcmp(tmp->username,user)==0){
            i=1;
        }
        tmp=tmp->next;
    }
}

```

```

    }
    return i;
}

// struttura per eliminare il file di log dopo la chiusura del gioco
void eliminatoFile(char *user){
    FILE *log;
    time_t rawtime;
    time (&rawtime);
    struct tm* leggibile;
    leggibile = localtime(&rawtime);
    log=fopen("log.txt", "a");
    fprintf(log, "%d/%s/%d %d:%d:%d L'utente %s è stata eliminato\n", leggibile-
>tm_mday, mesi[leggibile->tm_mon], leggibile->tm_year + 1900, leggibile-
>tm_hour, leggibile->tm_min, leggibile->tm_sec, user);
    fclose(log);
}

int controlloPercorso( struct mappa **a, int riga, int colonna){
    int i, j; //scorrono la matrice servono per i for
    int c1=0, c2=0;
    int result=0;
    for(j=0; j < M ; j++){
        if(a[riga][j].valore==1) c2++;
    }
    for (i=0; i < N ; i++){
        if(a[i][colonna].valore==1) c1++;
    }
    if(c1 >= N-1 || c2 >= M-1) result=1;
    //Verifico che non inserisca una cella vuota all'interno della mia matrice seguito
da muri
    for(i=0; i<N; i++){
        for(j=0; j<M; j++){
            if(i+2 < N && i-1 > -1 && j-1 > -1 && j+1 < M) {
                if(a[i+2][j].valore == 1 && a[i-1][j-1].valore==1 &&
a[i+1][j+1].valore==1){
                    result=0;
                }
            }
            if(i-2 > -1 && i+1 < N && j-1 > -1 && j+1 < M){
                if(a[i-2][j].valore==1 && a[i-1][j-1].valore==1 &&
a[i+1][j+1].valore==1){
                    result=0;
                }
            }
            if(j+2 < M && j-1 > -1 && i-1 > -1 && i+1 < N){
                if(a[i][j+2].valore==1 && a[i-1][j-1].valore==1 &&
a[i+1][j+1].valore==1){
                    result=0;
                }
            }
            if(j-2 > -1 && j+1 < N && i-1 > -1 && i+1 < M){
                if(a[i][j-2].valore==1 && a[i-1][j-1].valore==1 &&
a[i+1][j+1].valore==1){
                    result=0;
                }
            }
            if(i==0){
                if(a[i+1][j].valore==1 && a[i][j+1].valore==1){
                    result=0;
                }
            }
        }
    }
}

```

```

    }
    else if(i==N-1){
        if(a[i-1][j].valore==1 && a[i][j+1].valore==1){
            result=0;
        }
    }
    else if(j==0){
        if(a[i][j-1].valore==1 && a[i+1][j].valore==1){
            result=0;
        }
    }
    else if(j==M-1){
        if(a[i-1][j].valore==0 && a[i][j-1].valore==1){
            result=0;
        }
    }
}

}

return result;
}

```

//Funzione per il login, controlla se corrispondono username e password all'interno della lista

```

int login(struct utenti *top,char *user,char*pw){
    int i=0;
    while(top!=NULL){
        if(strcmp(top->username , user) == 0 && strcmp(top->password , pw) == 0){
            top->connected=1;
            i=1;
        }
        top=top->next;
    }
    return i;
}

```

```

void deallocaMatrice(struct mappa **a){
    int i,j;
    for(i=0;i<N;i++){
        free(a[i]);
    }
    free(a);
}

```

```

void vita(struct utenti *top, char *user,int i){
    while(top!=NULL){
        if(strcmp(top->username,user)==0){
            if(i==0){
                top->vita=top->vita-500;
                if(top->vita <1) eliminatoFile(user);
            }
        }
    }
}

```

```

        if(i==1){
            top->vita=top->vita -1;
        }
    }
    top=top->next;
}

//Funzione Per Eliminare Vita agli utenti del gioco.
void eliminaVita(struct utenti *mappa,int color){
    char *user;
    //Scorro la lista degli utenti presenti sulla posizione sulla mappa
    while(mappa!=NULL){
        //Se il colore degli utenti presenti in quella posizione è diverso da quello
        dell'utente che arriva in quella posizione procedo
        //a decrementare la vita
        if(mappa->colore!=color){
            user=(char * )malloc(sizeof(char));
            strcpy(user,mappa->username);
            //Alloco spazio per la variabile che mi conterrà le informazioni dell
            utente presente sulla mappa
            //Scorro la mia lista degli utenti per trovare l'utente trovato e decremen-
            targli la vita
            vita(top,user,0);
        }

        mappa=mappa->next;
    }
}

```

```

struct utenti *elimina(struct utenti *top,char *user){
    struct utenti *tmp;
    if(top!=NULL){
        if(strcmp(top->username,user)==0){
            tmp=top->next;
            free (top);
            top=tmp;
        }
        else top->next=elimina(top->next,user);
    }
    return top;
}

```

```

int verificaVita(struct utenti *top, char *user){
    int flag=0;
    while(top!=NULL){
        if(strcmp(top->username,user)==0){
            if(top->vita > 0) flag=1;
        }
        top=top->next;
    }
    return flag;
}

```

```

int spostamento(struct mappa **a, char *user, char *passw, int c){
    char prova[2];

```



```

int flag,color,k,i,j;
i=getij(top,user,0);
j=getij(top,user,1);
color=getcolor(top,user);
if(vittoria==1){
    flag=9;
    c=9;
}
else if(verificaVita(top,user)==0){
    c=10;
    flag=10;
}
switch(c){
    case 1:
        if(i-1 <= -1){
            flag=4;
            break;
        }
        flag=inserisciSoldato(a,user,passw,i-1,j,color,1);
        vita(top,user,1);
        a[i][j].utente=elimina(a[i][j].utente,user);
        break;
    case 2:
        if(i+1 > N-1 ){
            flag=5;
            break;
        }
        flag=inserisciSoldato(a,user,passw,i+1,j,color,1);
        vita(top,user,1);
        a[i][j].utente=elimina(a[i][j].utente,user);
        break;
    case 3:
        if(j+1 > M-1){
            flag=6;
            break;
        }
        flag=inserisciSoldato(a,user,passw,i,j+1,color,1);
        vita(top,user,1);
        a[i][j].utente=elimina(a[i][j].utente,user);
        break;
    case 4:
        if(j-1 <= -1){
            flag=7;
            break;
        }
        flag=inserisciSoldato(a,user,passw,i,j-1,color,1);
        vita(top,user,1);
        a[i][j].utente=elimina(a[i][j].utente,user);
        break;
    case 9: //Vittoria da parte di un utente.
        break;
    case 10: //Utente non ha abbastanza vita per spostarsi.
        break ;
    default:
        //Qualcosa è andato storto.
        break;
}
return flag;

```

```

}

```

```
//funzione per vedere se c'è almeno un giocatore per squadra
int start(struct utenti *top){
```

```
    int k=0;
```

```
    int j=0;
```

```
    while(top!=NULL){
```

```
        if(top->connected==1){
```

```
            if(top->colore==1){
```

```
                k++;
```

```
            }
```

```
            else if(top->colore==2){
```

```
                j++;
```

```
            }
```

```
        }
```

```
        top=top->next;
```

```
    }
```

```
    if (k>0&& j>0){
```

```
        return 1;
```

```
    }
```

```
    return 0;
```

```
}
```

```
void disconnect(struct utenti *top,char *user){
```

```
    while(top!=NULL){
```

```
        if(strcmp(top->username,user)==0){
```

```
            top->connected=0;
```

```
        }
```

```
        top=top->next;
```

```
    }
```

```
}
```

```
void disconnectall(struct utenti *top){
```

```
    while(top!=NULL){
```

```
        top->colore=0;
```

```
        top->i=0;
```

```
        top->j=0;
```

```
        top->vita=0;
```

```
        top->connected=0;
```

```
        top=top->next;
```

```
    }
```

```
}
```

```
void richiesta(int *socket_desc)
```

```
{
```

```
    //Informazioni per gestire la richiesta.
```

```
    int sock = *(int*)socket_desc;
```

```
    int i,j,k,i1,j1,eliminazione;
```

```
    int read_size,flag=1,opzione,muro;
```

```
    char a,*message,*message2 ,matrix[Z],numero[1], cli-
```

```
ent_message[1024],client2_message[1024], *username, *password,*user,*pw,prova[2];
```

```
    FILE *fp;
```

```

message2=(char *)malloc(sizeof(char));
message=(char*)malloc(sizeof(char));
username=(char*)malloc(sizeof(char));

while(flag){

    if(vittoria==1){
        win(vittoriaMessage,sock);
        deallocaMatrice(matrice);
        disconnectall(top);
        matrice=inizializzaMatrice(matrice);
        riempiMatrice(matrice);
        stampaMatrice(matrice);

        vittoria=0;
    }

    if(read_size = recv(sock , client_message ,sizeof(client_message), 0)){
        opzione=atoi(client_message);
        switch (opzione){
            case 1:
                write(sock,"Inserisci Username\0",sizeof("Inserisci Username\0"));
                strncpy(client_message," ",sizeof(client_message));
                if(read_size=recv(sock,client_message,sizeof(client_message), 0)){
                    username=(char *)malloc(sizeof(client_message));
                    strcpy(username,client_message);
                    write(sock,"Inserisci Password\0",sizeof("Inserisci Pass-
word\0"));
                    strncpy(client_message," ",sizeof(client_message));

                    if(read_size=recv(sock,client_message,sizeof(client_message),0)){
                        password=(char*)malloc(sizeof(client_message));
                        strcpy(password,client_message);
                    }
                }

                if(login(top,username,password)==1){
                    write(sock,"Login Effettuato\0",sizeof("Login Effettuato\0"));
                }
                else{
                    write(sock,"Login Fallito\0",sizeof("Login Fallito\0"));
                }
                break;

            case 2:
                write(sock,"Inserisci Username:\0",sizeof("Inserisci
Username:\0"));
                strncpy(client_message," ",sizeof(client_message));
                if(read_size=recv(sock,client_message,sizeof(client_message), 0)){
                    username=(char *)malloc(sizeof(client_message));
                    strcpy(username,client_message);
                    if(ricerca(top,username)) {
                        write(sock,"Username gia presente\0", sizeof("Username gia
presente\0"));
                        break;
                    }
                }

```

```

    }
    write(sock, "Inserisci Password\0", sizeof("Inserisci Password\0"));
    strncpy(client_message, " ", sizeof(client_message));

    if(read_size=recv(sock, client_message, sizeof(client_message), 0)){
        password=(char*)malloc(sizeof(client_message));
        strcpy(password, client_message);
    }
    fp=fopen("user.txt", "a");
    if(fp==NULL){
        puts("Problema con apertura file");
        break;
    }
    fprintf(fp, "%s:%s\n", username, password);
    fclose(fp);

    write(sock, "Registrazione effettuata con succe-
so\0", sizeof("Registrazione effettuata con successo\0"));

    //Inserisco I miei utenti all'interno di una linked list.
    top=nuovo(top, username, password);

    break;

case 3:
    write(sock, " ", sizeof(" "));
    strcpy(client_message, " ");
    strcpy(client2_message, " ");
    if(read_size=recv(sock, client2_message, sizeof(client2_message),
0)){

        username=(char*)malloc(sizeof(client_message));
        strcpy(username, client2_message);
        if(inizializzaGiocatore(top, username, 0)==1){
            write(sock, "1", sizeof("1"));
            strcpy(client_message, " ");

if(read_size=recv(sock, client_message, sizeof(client_message), 0)){
            //Devo effettuare l'inserimento del colore delle ar-
mate.

            opzione=atoi(client_message);

            if(inizializzaGiocatore(top, username, opzione)==1){
                //Ho impostato la squadra per l'username
                while (start(top)!=1){

                    sleep(2);

                }

                stampamatrix(top, matrix, username, i1, j1, 1);
                write(sock, matrix, strlen(matrix));

if(read_size=recv(sock, client_message, sizeof(client_message), 0)){

                    write(sock, "Inserisci Riga dove posizionare il
giocatore\0", sizeof("Inserisci Riga dove posizionare il giocatore\0"));
                    strcpy(client_message, " \0");

```

```

giocatore
        if(read_size=recv(sock,client2_message,2,0)){
            //Ricevuto la posizione dove inserire il
            i=atoi(client2_message);

        }
        write(sock,"Inserisci Colonna dove posizionare
il giocatore\0",sizeof("Inserisci Colonna dove posizionare il giocatore\0"));

        if(read_size=recv(sock,client2_message,2,0)){
            j=atoi(client2_message);
        }

if(inserisciSoldato(matrice,username,password,i,j,opzione,0)>1)
    {
        //Ha inserito Correttamente il soldato.
    }

    }
    }
    }
    }
    else write(sock,"0",sizeof("0"));
}

break;
case 4://Spostamento

        write(sock,"Dove ti vuoi spostare?\n1 - Nord\n2 - Sud\n3 -
Est\n4 - Ovest\0" , sizeof("Dove ti vuoi spostare?\n1 - Nord\n2 - Sud\n3 - Est\n4 -
Ovest\0"));

        strcpy(client_message, " ");
        //Ricevo informazioni dal client riguardo lo spostamento.

if(read_size=recv(sock,client_message,sizeof(client_message),0)){
    opzione=atoi(client_message);
    write(sock," ", sizeof (" "));
    strncpy(client_message," ",sizeof(client_message));

if(read_size=recv(sock,client_message,sizeof(client_message),0)){
    username=(char *)malloc(sizeof(client_message));
    strcpy(username,client_message);
    write(sock," ",sizeof(" "));
    strcpy(client_message," ");

if(read_size=recv(sock,client_message,sizeof(client_message),0)){
    password=(char *) malloc(sizeof(client_message));
    strcpy(password,client_message);
    strcpy(client_message," ");
    //Richiamo una funzione che mi effettua lo sposta-
mento e mi decrementa di uno la vita del mio personaggio.
    //Poichè siamo in presenza di un server concor-
rente, lo spostamento è stato effettuato tenendo conto di alcune considerazioni:

```

//In ogni posizione , la mappa avrà solamente l'ultimo giocatore che è arrivato in quel punto, in modo tale, da evitare di poter eliminare il giocatore avversario con mosse di tipo nord - sud , est-ovest e viceversa.

//Nel caso in cui quindi si verifichi una collisione, il giocatore avversario perderà 500 punti vita e verrà temporaneamente eliminato dalla mappa, mantenendo le informazioni riguardanti la sua precedente posizione.

```
flag=spostamento(matrice,username,password,opzione);
switch(flag){
    case 1:
        write(sock,"Non e' stato potuto completare
lo spostamento in quanto e' presente un ostacolo\0",sizeof("Non e' stato potuto com-
pletare lo spostamento in quanto e' presente un ostacolo\0"));
        break;
    case 2:
        write(sock,"Spostamento effettuato con suc-
cesso\0",sizeof("Spostamento effettuato con successo\0"));
        break;
    case 3:
        write(sock,"Lo Spostamento ha portato ad
una collisione con altri giocatori appartenenti alla squadra nemica che hanno perso 500
punti vita\0",sizeof("Lo Spostamento ha portato ad una collisione con altri giocatori
appartenenti alla squadra nemica che hanno perso 500 punti vita\0"));
        break;
    case 4:
        write(sock,"Spostamento fallito!\nNon e'
possibile andare ancora a Nord \0",sizeof("Spostamento fallito!\nNon è possibile andare
ancora a Nord \0"));
        break;
    case 5:
        write(sock,"Spostamento fallito!\nNon e'
possibile andare ancora a Sud \0",sizeof("Spostamento fallito!\nNon è possibile andare
ancora a Sud \0"));
        break;
    case 6:
        write(sock,"Spostamento fallito!\nNon e'
possibile andare ancora a Est \0",sizeof("Spostamento fallito!\nNon e' possibile andare
ancora a Est \0"));
        break;
    case 7:
        write(sock,"Spostamento fallito!\nNon e'
possibile andare ancora a Ovest \0",sizeof("Spostamento fallito!\nNon è possibile an-
dare ancora a Ovest \0"));
        break;
    case 8:
        write(sock,"1\0",sizeof("1\0"));
        break;
    case 9:

write(sock,vittoriaMessage,strlen(vittoriaMessage));
        break;
    case 10:
        write(sock,"Sei stato eliminato,non puoi
spostarti\0",sizeof("Sei stato eliminato,non puoi spostarti\0"));
        default:
            break;
    }
}
}
```

```

        break;
        case 5:
            write(sock,"Disconnessione in
corso...\0",sizeof("Disconnessione in corso...\0"));
        if(read_size=recv(sock,client_message,sizeof(client_message),0)){
            strcpy(username,client_message);
            disconnect(top,username);
        }
        break;
        case 6:
            message2=NULL;
            message2=utentiConnessi(top,message2);
            //Stampa degli utenti collegati.
            write(sock,message2,strlen(message2));

            break;
        case 7://Stampa degli ostacoli
            write(sock," ",sizeof(" "));

        if(read_size=recv(sock,client_message,sizeof(client_message),0)){
            strcpy(username,client_message);
            mes-
sage=stampaostacoli(matrice,message);
            write(sock,message,strlen(message));
        }

        break;
        case 8:
            write(sock," ",sizeof(" "));

        if(read_size=recv(sock,client_message,sizeof(client_message),0)){
            strcpy(username,client_message);
            mes-
sage2=alleati(top,username,message2);
            write(sock,message2,strlen(message2));
        }

        break;
        case 9:
            //Mi ricavo le informazioni su dove si
trova l'utente e gli stampo *
            i1=getij(top,username,0);
            j1=getij(top,username,1);
            write(sock," ", sizeof (" "));
            strncpy(client_message,"
",sizeof(client_message));

        if(read_size=recv(sock,client_message,sizeof(client_message),0)){
            strcpy(username,client_message);

            stampama-
trix(top,matrix,username,i1,j1,0);
            write(sock,matrix,strlen(matrix));
        }
        break;
        default:
            write(sock,"Comando non ricono-
sciuto\0",sizeof("Comando non riconosciuto\0"));
            break;

```

```

    }
}

    if(read_size == 0)
    {

    }
    else if(read_size == -1)
    {
        perror("ricezione fallita");
    }
}

//Libero il puntatore.
free(socket_desc);

}
//funzione per scrivere i file di log
void scriviFile(int i){
    char colore[6];
    FILE *log;
    time_t rawtime;
    time(&rawtime);
    struct tm* leggibile;
    leggibile = localtime(&rawtime);
    if(i==1) strcpy(colore,"Blu");
    else if(i==2) strcpy(colore,"Rosso");
    log=fopen("log.txt","a");
    fprintf(log,"%d/%s/%d %d:%d:%d La Bandiera %s è stata trovata.\n",leggibile-
>tm_mday,mesi[leggibile->tm_mon],leggibile->tm_year + 1900,leggibile-
>tm_hour,leggibile->tm_min,leggibile->tm_sec,colore);
    fprintf(log,"%d/%s/%d %d:%d:%d Partita Terminata\n",leggibile-
>tm_mday,mesi[leggibile->tm_mon],leggibile->tm_year + 1900,leggibile-
>tm_hour,leggibile->tm_min,leggibile->tm_sec);
    fprintf(log,"%d/%s/%d %d:%d:%d Nuova Partita Generata\n",leggibile-
>tm_mday,mesi[leggibile->tm_mon],leggibile->tm_year + 1900,leggibile-
>tm_hour,leggibile->tm_min,leggibile->tm_sec);
    fclose(log);
}
//-----
//funzioni di stampa

void stampaMatrice(struct mappa **a){
    FILE *fp;
    fp=fopen("mappa.txt","w");
    int i,j;
    for(i=0;i<N;i++){
        for(j=0;j<M;j++){
            fprintf(fp,"%d ",a[i][j].valore);
        }
        fprintf(fp,"\n");
    }
    fclose(fp);
}

//Stampa della lista lato server, per debug

```



```

void stampa(struct utenti *top){
    int i=0;
    while(top!=NULL){
        puts(top->username);
        puts(top->password);
        top=top->next;
    }
}

char *utentiConnessi(struct utenti *top,char *users){
    users=NULL;
    users=(char * )malloc(2000*sizeof(char));
    int k;
    char numero[5];
    strcpy(users,"Utenti Connessi Alla Piattaforma:\n");
    while(top!=NULL){
        if(top->connected==1){
            strcat(users,"Utente: ");
            strcat(users,top->username);
            strcat(users," Squadra di appartenenza: ");
            if(top->colore==1){
                strcat(users,"BLU");
            }
            else if(top->colore==2){
                strcat(users,"ROSSO");
            }
            strcat(users," Punti Vita: ");
            k=sprintf(numero,5,"%d",top->vita);
            strcat(users,numero);
            strcat(users,"\n");
        }
        top=top->next;
    }

    return users;
}

char *alleati(struct utenti *top,char *user,char *stringa){
    stringa=realloc(stringa,2000*sizeof(char));
    int k;
    char numero[5];
    int color;
    color=getcolor(top,user);
    strcpy(stringa,"Soldati Alleati:\n");

    while(top!=NULL){
        if(strcmp(top->username,user)!=0){
            if(top->colore==color){
                strcat(stringa,"Utente: ");
                strcat(stringa,top->username);
                strcat(stringa," Posizione : (");
                sprintf(numero,3,"%d",top->i);
                strcat(stringa,numero);
                strcat(stringa,",");
                sprintf(numero,3,"%d",top->j);
                strcat(stringa,numero);
                strcat(stringa,") ");
                strcat(stringa," Punti Vita: ");
                sprintf(numero,5,"%d",top->vita);
                strcat(stringa,numero);
                strcat(stringa,"\n");
            }
        }
    }
}

```

```

        top=top->next;
    }
    return stringa;
}

void stampamatrix(struct utenti *top,char *matrix,char *username,int i1,int j1,int k){
    int i,j;
    char numero[5];
    if(k==1){
        i1=-2;
        j1=-2;
    }
    strcpy(matrix,"Mappa Della Partita: \n");
    for(i=-1;i<N;i++){
        for(j=-1;j<M;j++){
            if(i== -1 && j== -1){
                strcat(matrix,"x ");
            }
            else if(i== -1){
                sprintf(numero,3,"%d",j);
                strcat(matrix,numero);
                strcat(matrix," ");
            }

            else if(j== -1) {
                sprintf(numero,3,"%d",i);
                strcat(matrix,numero);
                strcat(matrix," ");
            }
            else if(i==i1 && j==j1) strcat(matrix,"o ");

            else strcat(matrix,"- ");

        }
        strcat(matrix,"\n");
    }

    strcat(matrix,"\0");
}

```

```

char *stampaostacoli(struct mappa **a,char *matrix){

    matrix=realloc(matrix,2000*sizeof(char));
    strcpy(matrix,"Stampo la mappa degli ostacoli incontrati:\n-x: Rappresenta
posizione in cui e' stato incontrato un ostacolo\n");
    int i,j,k;
    char numero[5];
    for(i=-1;i<N;i++){
        for(j=-1;j<M;j++){
            if(i== -1 && j== -1){
                strcat(matrix,"x ");
            }
            else if(i== -1){
                k=sprintf(numero,3,"%d",j);
                strcat(matrix,numero);
                strcat(matrix," ");
            }

        }
    }
}

```

```

        else if(j== -1) {
            k=snprintf(numero,3,"%d",i);
            strcat(matrix,numero);
            strcat(matrix," ");
        }
        else if(a[i][j].x==1) strcat(matrix,"x ");
        else strcat(matrix,"- ");

    }
    strcat(matrix,"\n");
}
strcat(matrix,"\0");
return matrix;
}

//-----
//main

int main (int argc, char *argv[]){
    FILE *log;
    int socket_desc,client_sock,c,read_size,t;
    struct sockaddr_in server,client[20];
    char str[20];
    int t_free[16];
    int n;
    pthread_t tid;
    pthread_t t_id[16];
    pthread_t daeid[16];
    pthread_attr_t attr;
    struct utenti *top=NULL;
    time_t rawtime;
    time (&rawtime);
    struct tm* leggibile;
    FILE *fp;
    //All'apertura del server creo il mio file contenente gli utenti che si regis-
    treranno alla mia applicazione.
    fp=fopen("user.txt","w");
    if(fp==NULL){
        perror("errore file");
        return 1;
    }

    matrice=inizializzaMatrice(matrice);

    riempiMatrice(matrice);

    stampaMatrice(matrice);

    //Controllo input
    if(argc < 2){
        write(STDERR_FILENO,"Attenzione, sintassi errata.\n Esegui il programma
        selezionando la porta\n",sizeof("Attenzione, sintassi errata.\n Esegui il programma
        selezionando la porta\n"));
        exit(1);
    }
    //Creo il mio socket
    socket_desc=socket(AF_INET,SOCK_STREAM,0);
    if(socket_desc == -1){
        write(STDERR_FILENO,"Impossibile creare socket\n",sizeof("Impossibile creare
        socket\n"));
    }

```

```

        return 1;
    }
    puts("Socket Creata");

    //Creo la struttura del mio server
    server.sin_family=AF_INET;
    server.sin_addr.s_addr=INADDR_ANY;
    server.sin_port=htons(atoi(argv[1]));

    //Effettuo la Bind
    if(bind(socket_desc,(struct sockaddr *)&server,sizeof(server))<0){
        perror("Bind fallita");
        return 1;
    }
    puts("In attesa di connessioni...");
    //Effettuo listen
    listen(socket_desc,16);
    leggibile = localtime(&rawtime);
    log=fopen("log.txt","a");
    fprintf(log,"%d/%s/%d %d:%d:%d Nuova Partita Generata.\n",leggibile-
>tm_mday,mesi[leggibile->tm_mon],leggibile->tm_year + 1900,leggibile-
>tm_hour,leggibile->tm_min,leggibile->tm_sec);
    fclose(log);
    while(1){
        leggibile = localtime(&rawtime);
        n=sizeof(client);

        if((client_sock=accept(socket_desc,(struct sockaddr *)&client[visits],&n))< 0){
            perror("accept failed\n");
            exit(1);
        }

        sockd[visits]=client_sock;
        visits++;

        log=fopen("log.txt","a");

        inet_ntop(AF_INET,&client[visits].sin_addr,str,INET_ADDRSTRLEN);
        if(visits%2==0){
            fprintf(log,"%d/%s/%d %d:%d:%d Nuova connessione da client %2s
\n",leggibile->tm_mday,mesi[leggibile->tm_mon],leggibile->tm_year + 1900,leggibile-
>tm_hour,leggibile->tm_min,leggibile->tm_sec,inet_ntoa(client[visits-1].sin_addr));
        }
        fclose(log);

        if(pthread_create(&tid,NULL,(void *)richiesta,&client_sock)){
            perror("Impossibile avviare un thread");
            return 0;
        }
        pthread_detach(tid);

        if(client_sock<0){
            perror("Accettazione fallita");
            return 1;
        }
    }
    return 0;
}

```

4.1 Client.c

```
#include<stdio.h> //printf
#include <stdlib.h>
#include<string.h> //strlen
#include<sys/socket.h> //socket
#include<arpa/inet.h> //inet_addr
#include <termios.h>
#include <unistd.h>
#include <pthread.h>
int flag;

void premenu(){
    puts("Per poter giocare devi registrarti e successivamente loggare ");
    puts("il gioco consiste ad un simil ruba bandiera i player si muoveranno allo scuro
");
    puts("Che Operazione vuoi eseguire?\n1-Accedere\n2-Registrarti\n");
}
void loggedmenu(){
    puts(" Benvenuto nel gioco\n Regole\n a- ad una collisione con un ostaco-
lolâ€™effetto sarÃ nullo;");
    puts(" b-nella locazione di un altro utente nemico, lâ€™utente nemico perderÃ
500punti ");
    puts(" c-nella locazione della bandiera nemica lâ€™effetto sarÃ la vittoria\n");
    puts("Che Operazione vuoi eseguire?\n1-Spostamento\n2-Disconnettiti\n3-Visualizza
Utenti Collegati\n4-Visualizza Ostacoli Incontrati\n5-Posizioni soldati della propria
squadra con vita rimanente\n6-Stampa Mappa");
}

void connettiti(int sock){
    int scelta,indice=0,n,i;
    char message[1000] , server_reply[2000], serv-
er_reply2[2000],server_reply3[2000],server_reply4[2000],server_reply5[2000],server_repl
y6[2000];
    char comando[2],username[24],password[24],posizione[5];
    char *pw,ch;
    ssize_t nchr=0;
    pthread_t figlio;

    do{

        do{
            system("clear");
            premenu();
            while(1){
                scanf("%s",comando);
                scelta=atoi(comando);
                if(scelta==1 || scelta==2) break;
                puts("Inserisci un operazione valida.\n");
            }
            switch (scelta){

                case 1:
                    strcpy(server_reply, " ");
                    if( send(sock , comando , strlen(comando) , 0) < 0){
                        puts("Send failed");
                        break;
                    }
                }
            }
        } while(1);
    } while(1);
}
```

```

    }

    strcpy(server_reply, " ");
    if( recv(sock , server_reply , 2000 , 0) < 0){
        puts("Errore ricezione messaggio");
        break;
    }
    puts(server_reply);
    scanf("%s",username);
    if(send(sock,username,strlen(username),0) < 0){
        puts("Errore comunicazione server");
        break;
    }
    if(recv(sock,server_reply,2000,0) < 0){
        puts("Errore ricezione messaggio");
        break;
    }
    puts(server_reply);
    pw=getpass("");

    strcpy(password,pw);
    if(send(sock,pw,strlen(pw), 0) < 0 ){
        puts("Errore comunicazione server");
        break;
    }
    strcpy(server_reply, " ");
    if(recv(sock,server_reply,2000,0) < 0){
        puts("Errore ricezione messaggio");
        break;
    }
}

if(strcmp(server_reply,"Login Fallito")==0){

    flag=1;
}
else flag=10;

puts(server_reply);
sleep(1);
system("clear");

//DovrÃ² effettuare l'accesso al server
break;
case 2:
do{

    if( send(sock , "2" , sizeof("2") , 0) < 0){
        puts("Send failed");
        break;
    }
    strcpy(server_reply, " ");
    if( recv(sock , server_reply , sizeof(server_reply) , 0) < 0){
        puts("Errore ricezione messaggio");
        break;
    }
    puts(server_reply);
    scanf("%s",username);
    if(send(sock,username,strlen(username),0) < 0){
        puts("Errore comunicazione server");
        break;
    }
}

```

```

        if(recv(sock,server_reply,sizeof(server_reply),0) < 0){
            puts("Errore ricezione messaggio");
            break;
        }
        if(strcmp(server_reply,"Username gia presente")==0){
            puts("Username gia presente, ripeti selezione\n");
            flag=2;
        }
        else flag=3;
    }while(flag!=3);
    puts(server_reply);
    pw=getpass("");
    if(send(sock,pw,strlen(pw), 0) < 0 ){
        puts("Errore comunicazione server");
        break;
    }

    if(recv(sock,server_reply2,sizeof(server_reply2),0) < 0){
        puts("Errore ricezione messaggio");
        break;
    }
    puts(server_reply2);
    sleep(2);

    system("clear");
    break;

    default:
        //Qualcosa non ha funzionato
        break;
    }
}while(flag!=10);

//Comunicazione con il Server

if( send(sock , "3" , sizeof("3") , 0) < 0){
    puts("Send failed");
}
if( recv(sock , server_reply ,sizeof(server_reply) , 0) < 0){
    puts("Errore ricezione messaggio");
}
if(send(sock,username,strlen(username),0) < 0){
    puts("Errore comunicazione server");
}

strcpy(server_reply," ");
if(recv(sock,server_reply2,sizeof(server_reply2),0) < 0 ){
    puts("Errore ricezione messaggio");
}

if(strcmp(server_reply2,"1")==0){
    //Deve effettuare il primo inserimento.

    do{
        puts("\n\nInserisci l'esercito di appartenenza.\n1 - BLU\n2 - ROS-
SO\n");
        scanf("%s",comando);
        if(strcmp(comando,"1")==0){

```

```

        if(send(sock,"1",sizeof("1"),0) < 0){
            puts("Send Failed");
            break;
        }
        flag=1;
    }
    else if(strcmp(comando,"2")==0){
        if(send(sock,"2",sizeof("2"),0) < 0){
            puts("Send Failed");
            break;
        }
        flag=1;
    }
    else puts("\nInserisci un valore valido.\n");
    //voglio vedere se ci sono almeno un giocatore per squadra
    puts("Attendere altri giocatori ... \n");

    if(flag==1){
        //Mi stampo la matrice che ricevo dal server.
        strcpy(server_reply, " ");

        if(recv(sock,server_reply,sizeof(server_reply),0) < 0){
            puts("Errore ricezione messaggio");
            break;
        }
        puts("\n");
        puts(server_reply);
        strcpy(server_reply, " ");
        if(send(sock,"1",sizeof("1"),0) < 0){
            puts("Send Failed");
            break;
        }
    }
    if(recv(sock,server_reply,sizeof(server_reply),0) < 0){
        puts("Errore ricezione messaggio");
        break;
    }
    puts("L'inserimento di una riga o colonna fuori dalla dimensione
della \nmatrice comporta il posizionamento 0,0\n");
    puts("\n");
    puts(server_reply);
    scanf("%s",posizione);
    if(send(sock,posizione,strlen(posizione), 0) <0){
        puts("Send Failed");
        break;
    }

    if(recv(sock,server_reply3,sizeof(server_reply3),0) < 0) {
        puts("Errore ricezione messaggio");
        break;
    }

    puts(server_reply3);
    scanf("%s",posizione);
    if(send(sock,posizione,strlen(posizione),0) <0){
        puts("Sendi Failed");
        break;
    }
    flag=2;
}
}while(flag!=2);

```



```

        break;
    }
    //Mando al server la password
    if(send(sock,password,strlen(password),0) < 0){
        puts("Send Failed");
        break;
    }
    if(recv(sock,server_reply,sizeof(server_reply),0) < 0){
        puts("Errore ricezione messaggio");
        break;
    }
    if(strcmp(server_reply,"1")==0){
        puts("hai vinto");
        scelta=2;
        while(getchar()!='\n');
        ch=getchar();

        flag=5;
    }
    else {
        puts(server_reply);
        puts("Clicca invio per continuare");
        //Libero il buffer
        while(getchar()!='\n');
        ch=getchar();
    }
    break;
case 2:

    strcpy(server_reply2," ");
    if(recv(sock,server_reply2,sizeof(server_reply2),0) < 0) {
        puts("Errore ricezione messaggio");
        break;
    }
    puts(server_reply2);
    if(send(sock,username,strlen(username),0) < 0) {
        puts("Send Failed");
        break;
    }

    flag=5;

    break;

case 3:

    strcpy(server_reply3," ");
    if(recv(sock,server_reply3,sizeof(server_reply3),0)<0){
        puts("Errore ricezione messaggio");
        break;
    }
    puts(server_reply3);
    puts("Clicca invio per continuare");
    //Libero il buffer
    while(getchar()!='\n');
    ch=getchar();
    break;

```

case 4:

```
    if(recv(sock,server_reply4,sizeof(server_reply4),0) < 0){
        puts("Errore ricezione messaggio");
        break;
    }
    if(send(sock,username,strlen(username),0) < 0){
        puts("Send Failed");
        break;
    }

    if(recv(sock,server_reply4,sizeof(server_reply4),0) < 0){
        puts("Errore ricezione messaggio");
        break;
    }
    puts(server_reply4);
    puts("Clicca invio per continuare");
    //Libero il buffer
    while(getchar()!='\n');
    ch=getchar();

    break;
```

case 5:

```
    if(recv(sock,server_reply5,10,0) < 0){
        puts("Errore ricezione messaggio");
        break;
    }
    if(send(sock,username,strlen(username),0) < 0){
        puts("Send Failed");
        break;
    }
    if(recv(sock,server_reply5,sizeof(server_reply5),0) < 0){
        puts("Errore ricezione messaggio");
        break;
    }
    puts(server_reply5);
    puts("Clicca invio per continuare");
    //Libero il buffer
    while(getchar()!='\n');
    ch=getchar();

    break;
```

case 6:

```
    if(recv(sock,server_reply2,sizeof(server_reply2),0) < 0) {
        puts("Errore ricezione messaggio");
        break;
    }
    //Mando al server l'username
    if(send(sock,username,strlen(username),0) < 0) {
        puts("Send Failed");
        break;
    }
    strcpy(server_reply2, " ");
    //Ricevo la mia matrice
    if(recv(sock,server_reply2,sizeof(server_reply2),0) < 0){
        puts("Errore ricezione messaggio");
        break;
    }
}
```

```

        puts(server_reply2);
        puts("Clicca invio per continuare");
        //Libero il buffer
        while(getchar()!='\n');
        ch=getchar();

        break;

    default:
        strcpy(server_reply2, " ");
        if(recv(sock,server_reply3,sizeof(server_reply3),0) < 0){
            puts("Errore ricezione messaggio");
            break;
        }
        puts(server_reply3);
        break;
    }

}

    puts("Torno indietro");
}while(1);

}

```

```

int main(int argc , char *argv[])
{
    int sock,sock2;
    struct sockaddr_in server,server2;
    pthread_t figlio;
    char mynumb[3];
    int id;

    memset((void*)&server,0,sizeof(server)); //Inizializzazione
    //Creo il socket
    sock = socket(AF_INET , SOCK_STREAM , 0);
    if (sock == -1)
    {
        perror("socket");
        return 1;
    }
    sock2 = socket(PF_INET , SOCK_STREAM , 0);
    if (sock2 == -1)
    {
        perror("socket");
        return 1;
    }
    if(argv[2]!=NULL){
        server.sin_addr.s_addr= inet_addr(argv[2]);
    }
    else server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_family = AF_INET;

```

```
server.sin_port = htons( atoi(argv[1]));

if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
{
    perror("connessione fallita");
    return 1;
}

connettiti(sock);

return 0;
}
```