Relazione Progetto Laboratorio Sistemi Operativi: BattleBash

Maurizio Minieri – Alessio Spina,
Traccia A,
A.A. 2016/2017.

Descrizione del Problema:

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

Il server manterrà una rappresentazione dell'ambiente in cui verranno posizionati i soldati, gli ostacoli e altri elementi del gioco. 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. Il server posizionerà nella matrice di elementi del gioco (rappresentati opportunamente, ad esempio con numeri: 0 libero, 1 ostacolo, 2 mine, 3 armi) in posizione random. Ogni esercito dovrà difendere la propria bandiera e prendere quella del nemico. Le due bandiere dei due eserciti saranno disposte a caso dal server nel campo di gioco; per l'esercito 1 nella parte
sinistra del campo, per l'esercito 2 in quella destra. Ogni utente, una volta connesso al server, potrà partecipare al gioco con un soldato, indicando l'esercito di appartenenza e una posizione (x,y) sulla mappa.

Al momento della creazione il soldato riceverà 5000 punti da spendere durante il gioco. Per fare un passo l'utente spenderà 1 punto e riceverà l'informazione sull'effetto proprio movimento.

Nel caso in cui lo spostamento porti:

a) ad una collisione con un ostacolo l'effetto sarà nullo;

b) nella locazione di un altro utente nemico, il soldato con meno punti ne perderà 200;

c) nella locazione di una mina, il soldato perderà 600 punti; d) nella locazione di un'arma, il soldato prenderà 300 punti;

e) nella locazione della bandiera nemica l'effetto sarà la vittoria per l'esercito di appartenenza. Un utente senza più punti sarà eliminato dal gioco.

Quando un utente avrà conquistato la bandiera nemica o allo scadere del tempo di gioco, il server notificherà a tutti gli utenti del gioco il risultato finale, la fine della sessione e ne genererà una nuova. Nel caso di tempo scaduto il server notificherà il pareggio.
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, disconnettersi, vedere la lista degli utenti collegati e la squadra di appartenenza, vedere la posizione di tutti i soldati della propria squadra indicando i punti rimanenti;

vedere la posizione degli oggetti (ostacoli, armi e mine) incontrati da tutti i soldati del proprio esercito dall'inizio del gioco.

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 è 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 eliminazioni o del ritrovamento delle bandiere.

Client

Strategie di Risoluzione e Strutture Dati Utilizzate:

Questo paragrafo specifica le strategie di risoluzione e le strutture dati che sono state utilizzate, principalmente dalla applicazione Client, al fine di risolvere il problema dato.

In primo luogo, quando l'applicazione verrà avviata, permetterà all'utente, inserendo le credenziali (username e password), di effettuare il **login**. L'account può essere già precedentemente registrato oppure è possibile crearne uno nuovo. Successivamente si dovrà decidere la **squadra** di appartenenza e dare in input le **coordinate** di "*spawn*" per poter subentrare fisicamente nella mappa. Quest'ultima viene gestita mediante una struttura dati **matriciale di caratteri,** composta da 80 righe e 30 colonne.

Inizialmente il Client, genererà la mappa *"vergine",* ossia composta solamente dalle mura di limite, in modo da impedire la fuoriuscita del player dal campo di battaglia, e le basi difensive di entrambe le squadre. Successivamente riceverà dal Server tutte le informazioni relative agli oggetti e i player presenti sulla mappa, in modo da aggiornarla completamente.

Una volta entrati in partita, l'utente potrà effettuare degli spostamenti nelle direzioni N , S , E , O mediante i tasti "*a,s,d,w*" oppure potrà accedere al menù di gioco con il tasto "*m*". Tale fornirà le informazioni specificate dalla traccia.

Ad ogni spostamento effettuato dall'utente, la mappa di gioco si aggiornerà con tutte le posizioni degli altri players e oggetti correnti. L'applicazione gestisce l'elenco di tutti i players presenti in gioco con le corrispettive informazioni, mediante la struttura dati **Struct Players** P[104].

Il client rimarrà collegato al Server fin quando non saranno terminati i **5 minuti di tempo** disponibili per completare la partita, oppure quando una delle due squadre non avrà catturato la **bandiera**.

Le informazioni riguardanti all'indirizzo IP e la porta verranno gestite dalle strutture dati **struct sockaddr** e **struct hostent.**

Funzioni Principali utilizzate dal Client:

Di seguito verranno elencate le funzioni principali che vengono utilizzate dal Client:

- **int** loginClient(**int** sd ,**char** username[]);
    - loginClient effettua le operazioni di Logging trasmettendo l'username e password, scelti dall'utente, al Server. Quest'ultimo ne verificherà l'autenticità. Se il logging è andato a buon fine allora la funzione restituirà 1, altrimenti 0.

- **void** StampaObjects(**char** username[],**char** nome_p,**char** nome_o,**int** X,**int** Y);
    - StampaObjects stampa gli oggetti che sono stati presi da un determinato player.

- **void** receiveObjectDestroyed(**int** sd, **char** team);
    - receiveObjectDestroyed riceve dal Server un elenco di oggetti presi da un determinato team.

- **void** addPlayer(**int** X,**int** Y,**char** nome,**char*** username,**int** team,**int** punteggio,**int** indice,**int** i);
    - AddPlayer aggiunge alla struttura **struct Players P** un utente appena collegato, tenendo conto delle sue coordinate geografiche, del nome in gioco, dell'username, del team di appartenenza e del punteggio.

- **void** stampaPlayers(**char**** M, **int** n_players, **int** io);

    - stampaPlayers stampa l’elenco dei players appartenenti della stessa squadra del client che ha richiamato questa funzione.

- **void** stampaCompagni(**char**** M,char team, **int** n_players, **int** io);
    - stampaCompagni è una funzione utilizzata nel Menu di spostamento, ed effettua l’elenco dei compagni presenti in squadra.

- **int** requestToSpawn(**int** * x, **int** * y, **int** team,**int** sd, char **Mappa);
    - Questa funzione comunica al server le coordinate di spawn richiesta dall’utente. Si assicura che il player non nasca sui bordi della mappa, simboleggiati dal carattere ‘#’. Se il server da conferma, allora la richiesta è andata a buon fine, quindi la funzione restituirà 1, altrimenti 0.

- **void** RequestMovementClient(**int** sd, **char** spostamento , **char** username[]);
    - Invia al server, lo spostamento e l’username che il player vuole effettuare.

- **void** receivePlayers(**int** sd , **char** **Mappa, **int** *n_players);
    - Riceve dal server l’elenco dei players attualmente loggati e aggiorna sulla Mappa la loro posizione.

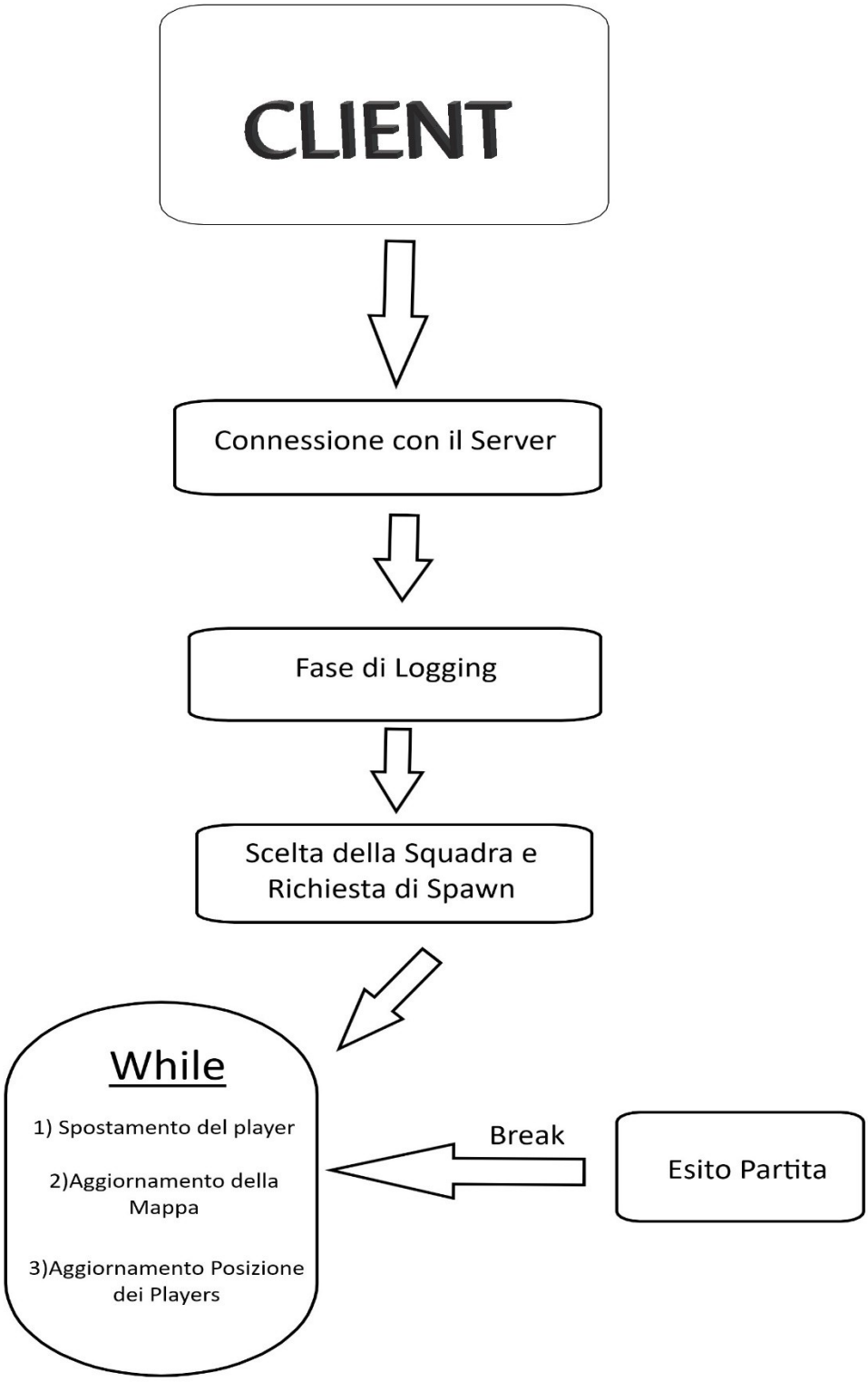
- **char**** allocaMappa();
    - Alloca dinamicamente la matrice adibita a contenere il campo di gioco.

- **char**** GeneraMappa();
    - Inserisce all’interno della mappa i bordi di gioco e le basi delle corrispettive squadre. Entrambe le strutture vengono rappresentate con il carattere “#”.

- **int** ReceiveData(**char** **Matrice, **int** *TOT_OBJECTS,**int** sd);
    - Riceve l’elenco degli oggetti dal Server e aggiorna il loro stato (visibile o non visibile) sulla Mappa.

- **int** receive_fine_game(**int** sd);
    - Riceve dal server l’esito della partita:
        - Hanno vinto i rossi,
        - Hanno vinto i verdi,
        - Pareggio.

Rappresentazione grafica del funzionamento del Client.


CLIENT
Connessione con il Server
Fase di Logging
Scelta della Squadra e Richiesta di Spawn
While
1) Spostamento del player
2)Aggiornamento della Mappa
3)Aggiornamento Posizione dei Players
Break
Esito Partita

Guida d'uso:

Per compilare ed eseguire l'applicazione, bisogna lanciare da terminale i seguenti comandi:

1) gcc client.c Funzioni_Client/funzioni_client.c -o client -lpthread
2) ./client Argomento1 Argomento2

Gli argomenti del comando ./client, corrispondo a indirizzo IP del Server e la porta su cui è in ascolto.

Esempio d'uso : "./client 93.257.12.155 8080"

Server

Strategie di Risoluzione e Strutture Dati Utilizzate:

Il server, come richiesto dalla traccia, effettua connessioni tramite socket TCP ed è di tipo concorrente, in grado quindi di stabilire più collegamenti contemporaneamente.

Le strutture principali che vengono adoperate dall'applicazione sono:

```
struct TPlayer
{
  char username[20];
  char nome;
  int X;
  int Y;
  char team;
  int punteggio;
  int indice_player;
};

typedef struct TPlayer Player;
```

```
struct Objects
{
 char username[20];
 char nome_p;
 char nome_o;
 char team;
 int X;
 int Y;
};
 struct Objects O[230];
```

Struct TPlayer è la struttura che memorizza i dati dei players presenti in gioco, tenendo conto dell'username, il nome identificativo sul campo di battaglia, le coordinate X,Y, il team di appartenenza, il punteggio e l'indice del player, ovvero un intero che indentifica ciascun giocatore all'interno della struttura.

Struct Objects è la struttura che memorizza i dati relativi agli oggetti dinamici (mine, armi e bandiere), presenti sulla mappa, che vengono calpestati/presi dai players. Vengono salvati l'username, il nome e il team del player che è entrato in contatto con l'oggetto, le coordinate e il nome oggetto; ovvero la sua tipologia (mina, arma , ostacolo o bandiera).

Inoltre il server utilizza una **Matrice** in modo da memorizzare tutti gli oggetti presenti sulla mappa. Essa è formata da 4 colonne che rappresentano concettualmente:

1) Tipologia di Oggetto (arma, mina, bandiera)
2) Coordinata X
3) Coordinata Y
4) Presenza sulla Mappa

Esempio di una generica matrice:

Tipologia di oggetto	**Coordinata X**	**Coordinata Y**	**Presenza sulla Mappa**
Arma (A)	3	22	V
Mina (M)	44	31	F
Bandiera (F)	12	11	V
Ostacolo(O)	49	1	V

```
//creazione del socket
if((sd = socket(PF_INET,SOCK_STREAM,0)) < 0)
{
  perror("Errore Creazione Socket");
  exit(-1);
}

// rende il socket riusabile
setsockopt(sd,SOL_SOCKET,SO_REUSEADDR,&val,sizeof(val));
```

Quando l'applicazione viene eseguita, effettuerà la creazione della **socket** (mostrata dall'immagine precedente) e subito dopo procederà con la configurazione relativa all'indirizzo IP, scegliendo il dominio **AF_INET** e la **porta** data in input.

```
//=========================CONFIG. INDIRIZZO IP=========================
memset((void *)&serv_ind, 0, sizeof(serv_ind)); // pulisce ind
serv_ind.sin_family = AF_INET; // ind di tipo INET
serv_ind.sin_port = htons(porta); // scelgo porta non priv.
serv_ind.sin_addr.s_addr = htonl(INADDR_ANY); // connessioni da ogni ip
//======================================================================
```

Successivamente lancia la funzione **bind** in modo da assegnare un indirizzo al descrittore della socket. Ed infine si metterà in ascolto di eventuali connessioni tramite la funzione l**isten.**

```c
//configurazione bind
 if (bind(sd, (struct sockaddr *)&serv_ind, sizeof(serv_ind)) < 0)
 {
   perror("Errore nella bind");
   exit(-1);
 }

//accetta fino a 20 connessioni, resta in ascolto sulla socket
 if (listen(sd, BACKLOG) < 0 )
 {
   perror("Errore nella listen");
   exit(-1);
 }
```

Il Server darà inizio alla partita quando capterà la prima connessione di un Client con la conseguente fase di **logging**.

Tramite il thread "**timing_thread**" si terrà conto dello scorrimento del tempo, in modo da segnalare la fine della partita, nel caso in cui siano passati 5 minuti dall'inizio del match.

Per ogni Client connesso, il server creerà un 2 thread, il primo (**connection_handler**) si occuperà della fase di logging, in cui il player dovrà registrarsi o utilizzare un account già memorizzato dal server, decidere le coordinate di "spawn" e il team di appartenenza.

Il secondo thread (**recv_movement_client**) avrà la mansione di gestire gli spostamenti del player sulla mappa, e gestire i vari trasferimenti di dati tra Client-Server.

Funzioni Principali utilizzate dal Server:

Di seguito verranno elencate le funzioni più importanti che caratterizzano il Server:

- **int** verify_client_registration(**char** *username, **char*** password);

    - Verifica se l'username e la password, inseriti dall'utente sono validi, confrontandoli con gli accounts già precedentemente registrati nel file Accounts.txt.

- **int** do_registration(**char** *username, **char*** password);
    - Effettua la registrazione di un nuovo account, assicurandosi che l'username inserito dall'utente non sia già presente all'interno del file Accounts.txt

- **int** login_state(**int** sd, **char** username[], **Player** P[], **int** n_players);
    - Questa funzione gestisce la fase di logging del player

- **int** loginServer(**int**,**char** username[], **char** password[]);
    - Riceve dal Client, username e password che l'utente ha trasmesso in input.

- **void** riempiNomi(**char** nomi_rossi[],**char** nomi_verdi[]);
    - Riempie i corrispettivi array di caratteri con la sequenza letterale alfabetica prima in maiuscolo (A-Z) e successivamente in maiuscolo (a-z).

- **char**** GeneraMatrice(**int*** n_o,**int*** n_a,**int*** n_m,**int*** TOT);
    - Genera la Matrice che memorizza tutti oggetti dinamici presenti sulla mappa. Riceve in input il numero di ostacoli, armi,mine e il loro totale.

- void write_on_file**(int** fp **, int** type, **char** *username,**char** nome, **char** team,**int** punteggio,**int** count_partite);
    - Scrive su differenti file tutte le informazioni di gioco: eliminazioni, squadra vincitrice e i player presenti in partita

- **char** giveNameClient(**int** sd, **char** nomi[]);
    - Assegna ad ogni player che effettua con successo il login, un nome che lo indentificherà all'interno del campo di Battaglia.

- **void** generateObstacles(**char** **Mappa,**int** n_ostacoli,**int** *indice);
    - Genera un numero randomico di ostacoli

- **void** generateWeapons(**char** **Mappa, **int** n_armi, **int*** indice);
    - Genera un numero randomico di armi

- **void** generateMines(**char** **Mappa, **int** n_mine, **int** *indice);
    - Genera un numero randomico di Mine

- **void** generateFlags(**char** **Mappa, **int** tot);
  - Genera due Bandiere.

- **int** check_username(**char** username[] , **Player** P[] , **int** n_players);
  - Verifica se un l'username inserito da un player corrisponde a quello di un altro giocatore già loggato.

- **void** addObject(**int*** n_oggetti,**char** nome_p,**char** team,**char** username[],**int** X,**int** Y,**char** nome_o);
  - Quando un player prende/calpesta una arma/mina, viene richiamata questa funzione che permette di salvare nella struttura Objects: il nome, il team, l'username, le coordinate del player e la tipologia di oggetto.

- **int** sendObjectDestroyed(**int** sd, **int** n_oggetti);
  - Trasferisce al Client, la struttura Objects.

- **int** spawnPlayer(**int** C_X , **int** C_Y, **char** nome,**char**** Mappa,**int** TOT,**Player** P[]);
  - Controlla se lo spawn del player è regolare, in caso positivo aggiorna le sue coordinate nella struttura Player.

- **void** addPlayer(**Player** P[],**int** X,**int** Y,**char** nome,**int** team,**char** username[], **int*** n_players, **int** sd);
  - Quando la fase di logging va a buon fine, il player appena accettato viene memorizzato nella struttura Player.

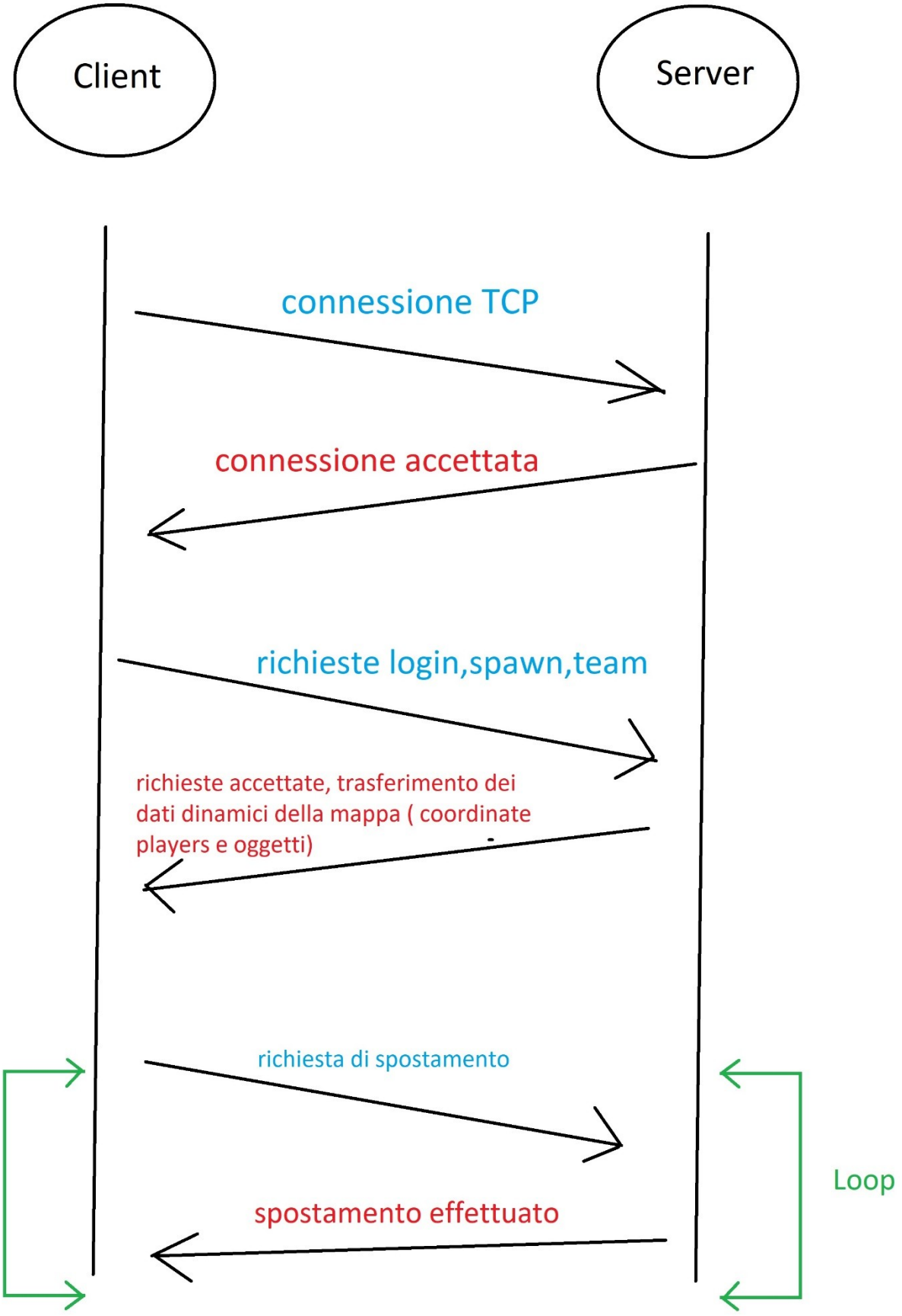
- **int** SendData(**char** **Matrice, **int** TOT_OBJECTS,**int** sd);
  - Si occupa di trasferire il contenuto (informazioni su armi,mine, ostacoli e bandiere) che vi è all'interno della Matrice al client

- **int** sendPlayers(**Player** P[], **int** n_players, **int** sd);
  - Trasferisce il contenuto della struttura Player al Client.

- **int** readRequestMovementClient(**int** sd,**char** **Mappa, **Player** P[],**int** n_players, **int** TOT_OBJECTS, **int** *n_oggetti, **int*** fine_game,**int** fp);
  - Legge la richiesta di spostamento del player.

- **int** check_movement_client(char username[], **int** sd, **char** ** Mappa, Player P[], **int** n_players, **char** spostamento,**int** TOT_OBJECTS,**int** *n_oggetti,int *fine_game,**int** fp);
  - Controlla che lo spostamento del player sia lecito.

Rappresentazione grafica del funzionamento del Server.


Client
Server
connessione TCP
connessione accettata
richieste login,spawn,team
richieste accettate, trasferimento dei dati dinamici della mappa (coordinate players e oggetti)
richiesta di spostamento
spostamento effettuato
Loop

Guida D'uso:

Per compilare ed eseguire l’applicazione, bisogna eseguire i seguenti comandi:

1) gcc server.c Funzioni_Server/funzioni_server.c -o server -lpthread
2) ./server Argomento1

L’argomento del comando ./server corrisponde al numero della porta in cui il server deve mettersi in ascolto.

Esempio ./server 8080

Codice Sorgente: Server

server.c

```
#include <errno.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <pthread.h> // Threads
#include <sys/socket.h> // Socket
#include <sys/un.h> // Connection
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <netdb.h>
#include <signal.h>
#include "Funzioni_Server/funzioni_server.h"
#define BACKLOG 20
#define DIM 104
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
```

```
Player P[104];
char nomi_verdi[52];
char nomi_rossi[52];
pthread_t thread_id, thread_id2;
int n_players=0;
int n_oggetti=0;
int fine_game=0;
int tempo_scaduto=0;
int count_partite=1;
char **Matrice;
int TOT=0;
int fp[3];



//gcc server.c Funzioni_Server/funzioni_server.c -o server -lpthread &&
./server
void *connection_handler(void *);
void *recv_movement_client(void *);
void *timing_thread();

int main (int argc, char **argv)
{
  srand(time(NULL));
  struct sockaddr_in serv_ind; // server indirizzo
  struct sockaddr_in client;  // client indirizzo
  socklen_t lung;
  int porta;
  int sd,val=1,accsd,check;
  char indcli[128];
  openFile(fp);
  signal(SIGPIPE, SIG_IGN);


  int n_ostacoli=0;
  int n_armi=0;
  int n_mine=0;

  if( (argc) < 2 )
  {
    fprintf(stdout, "\nInserisci la porta in cui il Server si metterÃ  in
ascolto!\n");
    exit(-1);
  }
  porta=atoi(argv[1]);

 //creazione del socket
  if((sd = socket(PF_INET,SOCK_STREAM,0)) < 0)
  {
    perror("Errore Creazione Socket");
    exit(-1);
  }

  // rende il socket riusabile
  setsockopt(sd,SOL_SOCKET,SO_REUSEADDR,&val,sizeof(val));


  //=======================CONFIG. INDIRIZZO IP=========================
  memset((void *)&serv_ind, 0, sizeof(serv_ind)); // pulisce ind
  serv_ind.sin_family = AF_INET; // ind di tipo INET
```

```
  serv_ind.sin_port = htons(porta); // scelgo porta non priv.
  serv_ind.sin_addr.s_addr = htonl(INADDR_ANY); // connessioni da ogni ip
//=========================================================================


 //configurazione bind
  if (bind(sd, (struct sockaddr *)&serv_ind, sizeof(serv_ind)) < 0)
  {
    perror("Errore nella bind");
    exit(-1);
  }

//accetta fino a 20 connessioni, resta in ascolto sulla socket
  if (listen(sd, BACKLOG) < 0 )
  {
    perror("Errore nella listen");
    exit(-1);
  }

  char hello[50];

puts("\n\t\t\t\t\t\t\tSTART SERVER!");

char accsd_c;

do{


    fprintf(stdout, "\nINIZIO PARTITA!\n");
    fine_game=0;
    riempiNomi(nomi_rossi,nomi_verdi);
    Matrice=GeneraMatrice(&n_ostacoli,&n_armi,&n_mine,&TOT);

    write_on_file(fp[0],0,"X",'X','X',0,count_partite);
    write_on_file(fp[1],0,"X",'X','X',0,count_partite);
    write_on_file(fp[2],0,"X",'X','X',0,count_partite);
    count_partite++;

  if( pthread_create( &thread_id2 , NULL ,  timing_thread , NULL) < 0)
  {
    perror("could not create thread");
    return 1;
  }

  if(fine_game==0)
  {
   while(accsd = accept(sd, (struct sockaddr *)&client, &lung))
   {
      if(fine_game>0)
       break;

      sprintf(hello,"Connection accepted from Client_socket: %d",accsd);
      puts(hello);

      accsd_c=accsd + '0';


      if( (write(accsd,&accsd_c,1)) < 0 )
      {
        perror("Error write accsd");
```

```
        close(sd);
      }

      if( pthread_create( &thread_id , NULL ,  connection_handler , (void*)
&accsd) < 0)
       {
           perror("could not create thread");
           return 1;
       }

  }
}


}while(1);


  return 0;
}



/
*================================THREAD====================================*
/
void *connection_handler(void *socket_desc)
{

  int sd = *(int*)socket_desc;
  int check, check_login_state;
  char team,nome,username[20];
  void *status;


  do{
      puts("lancio login state");
      check_login_state=login_state(sd,username,P,n_players);

      if(check_login_state == 0)
      {
         char msg[30];
         sprintf(msg,"Login Rifiutato da Client_socket = [%d]",sd);
         puts(msg);
      }
      else
      {

        char msg[30];
        sprintf(msg,"Login Accettato da USER=[%s] => Client_socket =
[%d]",username,sd);
        puts(msg);
        int X,Y;

        team=leggiTeam(sd);

        if(team == '<')
        {
          fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
          close(sd);
          pthread_exit(status);
```

```
}
else if(team == 'V')
{
  nome=giveNameClient(sd,nomi_verdi);

    if(nome == '<')
    {
      fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
      close(sd);
      pthread_exit(status);
    }
}
else
{
  nome=giveNameClient(sd,nomi_rossi);

  if(nome == '<')
  {
    fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
    close(sd);
    pthread_exit(status);
  }

}

do {
     check=leggiRequestSpawn(&X,&Y,sd,nome,Matrice,TOT,P);

     if(check == 2)
     {
       perror("Error Write response");
       fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
       close(sd);
       pthread_exit(status);
     }

     char response;
     if(check == 1) response = 'V';
     else response = 'F';


     if( (write(sd,&response,1)) < 0 )
    {
      perror("Error Write response");
      fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
      //freePlayer(P,sd,&n_players);
      close(sd);
      pthread_exit(status);
     }


   } while(check == 0);

  addPlayer(P,X,Y,nome,team,username,&n_players,sd);
  write_on_file(fp[0],1,username,nome,team,0,0);
  printf("\n\nSTAMPA PLAYERS:\n");
  stampaPlayers(P,n_players);

}
```

```
 } while(check_login_state == 0);
//  pthread_mutex_unlock(&m);

if( pthread_create( &thread_id2 , NULL ,  recv_movement_client , (void*)
&sd) < 0)
 {
     perror("could not create thread2");

}

pthread_join(thread_id2,NULL);

pthread_exit(status);
}




void *recv_movement_client(void *socket_desc)
{
  int sd = *(int*)socket_desc;
  int check;

  //pthread_mutex_lock(&m);

      if(sendPlayers(P,n_players,sd))
      {
        fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
        freePlayer(P,sd,&n_players,fp[1],nomi_verdi,nomi_rossi);
        close(sd);
      }

      if(SendData(Matrice,TOT,sd))
      {
        fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
        freePlayer(P,sd,&n_players,fp[1],nomi_verdi,nomi_rossi);
        close(sd);
      }


    while(1)
    {
        //fprintf(stdout, "\nN_PLAYERS=%d\n",n_players );

if(readRequestMovementClient(sd,Matrice,P,n_players,TOT,&n_oggetti,&fine_gam
e,fp[2]) == 2)
      {
          fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
          freePlayer(P,sd,&n_players,fp[1],nomi_verdi,nomi_rossi);
          close(sd);
          break;
      }

      if(sendPlayers(P,n_players,sd))
      {
        fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
        freePlayer(P,sd,&n_players,fp[1],nomi_verdi,nomi_rossi);
        close(sd);
        break;
```

```
      }

      if(SendData(Matrice,TOT,sd))
      {
        fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
        freePlayer(P,sd,&n_players,fp[1],nomi_verdi,nomi_rossi);
        close(sd);
        break;
      }

      if(read_Request_sendObjectDestroyed(sd,n_oggetti))
      {
        fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
        freePlayer(P,sd,&n_players,fp[1],nomi_verdi,nomi_rossi);
        close(sd);
        break;
      }


if(send_fine_game(sd,fine_game,P,&n_players,fp[1],nomi_verdi,nomi_rossi)>0)
       {
        fprintf(stdout, "\nN_PLAYERS=%d\n", n_players);
        break;
       }

  }

}


void *timing_thread()
{
  int seconds=300;
  void *status;

  tempo_scaduto=0;

  while(seconds>0)
  {
    if(fine_game>0)
    {
      fprintf(stdout, "\nFINE-GAME= %d", fine_game);
      break;
    }
    seconds--;
    sleep(1);
  }

  fprintf(stdout,"\nFINE TEMPO\n");

  //fprintf(stdout, "\nFINE-GAME2= %d", fine_game);
  if(fine_game!=0)
  {
   fine_game=3;
   tempo_scaduto=1;
  }

  pthread_exit(status);
}
```

Funzioni_server.c

```
#include <errno.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <pthread.h> // Threads
#include <sys/socket.h> // Socket
#include <sys/un.h> // Connection
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "funzioni_server.h"

#define SIZE_RIGHE 30
#define SIZE_COLONNE 80

#define RED     "\x1b[31m"
#define GREEN   "\x1b[32m"
#define COLOR_RESET   "\x1b[0m"

char ARMA = 124;                 // '|'
char MINA = '*';
char OSTACOLO = 94;             //^
char FLAG = 36;                 //$


int loginServer(int accsd,char username[], char password[])
{
  char tmpchar;
  int nbytes;


    if( (read(accsd,&tmpchar,1)) < 0 )
    {
      perror("Error reading registration on DB");
      exit(-1);
    }
      printf("\nLetto [%c]", tmpchar);

    if( (nbytes=read(accsd,username,20)) < 0 )
    {
      perror("Error Reading from Client: username");
      exit(-1);
    }
    username[nbytes]='\0';
    printf("\nLetto [%s]", username);


    if( (nbytes=read(accsd,password,20)) < 0 )
    {
      perror("Error reading from Client:password");
      exit(-1);
    }
    password[nbytes]='\0';
```

```
    printf("\nLetto [%s]", password);



    printf("\nRichiesta di login da [%s] [%s]
[%c]",username,password,tmpchar);

    if(tmpchar == 'y')
      return 1;

    else
      return 0;

}



int verify_client_registration(char username[], char password[])
{
   FILE *fp;
   char user_temp[20], pass_temp[20];

   if((fp=fopen("Accounts.txt","r"))<0)
   {
     perror("Errore apertura file : Accounts.txt");
   }

   puts("\nverify_client_registration:");

   while(!feof(fp))
   {
      fscanf(fp,"%s %s\n",user_temp,pass_temp);
      fprintf(stdout,"\nUser:[%s] Pass:[%s]", user_temp,pass_temp);

        if((strcmp(user_temp,username) == 0) &&
(strcmp(pass_temp,password)==0))
          return 1;
   }

   fclose(fp);
   return 0;

}

int do_registration(char username[], char password[])
{
    FILE *fp;
    int fd;
    char user_temp[20], pass_temp[20],text[41];


    if((fp=fopen("Accounts.txt","r"))<0)
    {
      perror("Errore apertura file : Accounts.txt");
    }

    while(!feof(fp))
    {
       fscanf(fp,"%s %s\n",user_temp,pass_temp);
```

```
       printf("\nuser:%s pass:%s", user_temp,pass_temp);

         if(strcmp(user_temp,username) == 0)
           return 0;
    }

    fclose(fp);

    if((fd=open("Accounts.txt", O_WRONLY | O_APPEND ,S_IRUSR | S_IWUSR ))<0)
    {
      perror("Errore apertura file : Accounts.txt");
    }


    strcpy(text,"\n");
    strcat(text,username);
    strcat(text," ");
    strcat(text,password);

    fprintf(stdout, "\ntext: %s", text);

    lseek(fd,0,SEEK_END);
    write(fd,text,strlen(text));

    close(fd);

    return 1;

}


int login_state(int sd, char username[], Player P[], int n_players)
{
  char password[20];

  if(loginServer(sd,username,password))
  {
      if(verify_client_registration(username,password))
      {

        if(check_username(username,P,n_players))
        {
          write(sd,"Y",1);
          return 1;
        }
        else
        {
          write(sd ,"X",1);
          return 0;
        }

      }
      else
      {
       printf("\nLogin Failed: Utente o Password errati!\n");
        write(sd,"F",1);
        return 0;
      }
```

```
   }
   else
   {
    if(do_registration(username,password))
    {
      printf("\nRegistrazione Effettuata!\n");
      write(sd,"Y",1);
      return 1;
    }

    else
    {
      printf("\nRegistrazione Fallita!\n");
      write(sd,"F",1);
      return 0;
    }
 }




}

int check_username(char username[] , Player P[] , int n_players)
{
      int i=0;

    //  fprintf(stdout, "\nSto in check_username con %s",username);


      for(i=0;i<n_players;i++)
      {
          if(strcmp(P[i].username,username)==0)
            return 0;
      }

      return 1;
}


char leggiTeam(int sd)
{
  char team;

    if( (read(sd,&team,1)) < 0 )
    {
      perror("Error Read Team from Client");
      return '<';
    }


return team;
}


char cercaNome(char nomi[])
{
  int i;
  char nome;
  for(i=0;i<52;i++)
```

```
  {
     if(nomi[i] != '@')
     {
        nome=nomi[i];
        nomi[i]='@';
        return nome;
      }
  }

  return '@'; // array vuoto!
}



void riempiNomi(char nomi_rossi[],char nomi_verdi[])
{
  int i=0;
  char lettera_start1='A';
  char lettera_start2='a';

  for(i=0;i<26;i++)
  {
    nomi_rossi[i]=lettera_start1;
    nomi_verdi[i]=lettera_start1;
    nomi_rossi[i+26]=lettera_start2;
    nomi_verdi[i+26]=lettera_start2;
    lettera_start1++;
    lettera_start2++;
  }

}



char giveNameClient(int sd, char nomi[])
{
  char nome;

    nome=cercaNome(nomi);

    if( (write(sd,&nome,1)) < 0 )
    {
      perror("Error giveNameClient()");
      return '<';
    }


  return nome;
}

int leggiRequestSpawn(int *C_X , int * C_Y, int sd,char nome,char** Mappa,
int TOT_OBJECTS,Player P[])
{

  int x=0,y=0;
  int x_temp,y_temp;

      if( (recv(sd, &x_temp, 4, 0)) < 0 )
      {
        perror("Error Read Coordinata_X");
```

```
        return 2;
      }

      if( ((recv(sd, &y_temp, 4, 0)) < 0 ))
      {
        perror("Error Read Coordinata_Y");
        return 2;
      }

   x = ntohl(x_temp);
   y = ntohl(y_temp);

   *C_X=x;
   *C_Y=y;

   int check=spawnPlayer(*C_X,*C_Y,nome,Mappa,TOT_OBJECTS,P);

   return check;

}

void setTime(char *time_string)
{
  time_t current_time;

  current_time = time(NULL);
  if (current_time == ((time_t)-1))
   {
       (void) fprintf(stderr, "Failure to obtain the current time.\n");
       exit(EXIT_FAILURE);
   }

   /* Convert to local time format. */
   time_string = ctime(&current_time);

   if (time_string == NULL)
   {
       (void) fprintf(stderr, "Failure to convert the current time.\n");
       exit(EXIT_FAILURE);
   }

   /* Print to stdout. ctime() has already added a terminating newline
character. */
    (void) printf("Current time is %s", time_string);

}

void removeFiles()
{
  remove("LoginClients.txt");
  remove("EliminationClients.txt");
  remove("Flag.txt");
}

void openFile(int fp[])
{

   removeFiles();
```

```
   if( (fp[0]=open("LoginClients.txt",O_WRONLY | O_CREAT | O_APPEND, S_IRUSR
| S_IWUSR)) < 0 )
   {
     perror("Error create or open file LoginClients.txt");
     //exit(-1);
   }

   if( (fp[1]=open("EliminationClients.txt",O_WRONLY | O_CREAT | O_APPEND,
S_IRUSR | S_IWUSR)) < 0 )
   {
     perror("Error create or open file EliminationClients.txt");
  //   exit(-1);
   }

   if( (fp[2]=open("Flag.txt",O_WRONLY | O_CREAT | O_APPEND, S_IRUSR |
S_IWUSR)) < 0 )
   {
     perror("Error create or open file Objects.txt");
  //   exit(-1);
   }



}

void write_on_file(int fp , int type, char *username,char nome, char
team,int punteggio,int count_partite)
{
    char message[200];
    time_t timer;
    char buffer[26];
    struct tm* tm_info;

    time(&timer);
    tm_info = localtime(&timer);

    strftime(buffer, 26, "%Y-%m-%d %H:%M:%S", tm_info);
    puts(buffer);

    if(type==0)
      sprintf(message,"[PARTITA: %d]\n",count_partite);

    if(type==1) // logging
    sprintf(message, "Username: [%s] - Nome: [%c] - Team: [%c] - Time:
[%s]\n",username,nome,team,buffer);

    else if(type == 2) //eliminazioni per DISCONNESSIONE
      sprintf(message, "Username: [%s] - Nome: [%c] - Team: [%c] - Time:
[%s] - Punteggio: [%d]
(Disconnesso)\n",username,nome,team,buffer,punteggio);

    else if(type == 3) // eliminazioni per MORTE
      sprintf(message, "Username: [%s] - Nome: [%c] - Team: [%c] - Time:
[%s] - Punteggio: [%d] (Morto)\n",username,nome,team,buffer,punteggio);

    else if(type == 4) // flag
      sprintf(message, "Bandiera presa da:\nUsername: [%s] - Nome: [%c] -
Team: [%c] - Time: [%s]\n",username,nome,team,buffer);

    if( (write(fp,message,strlen(message))) < 0 )
```

```
    {
      perror("Error Write on LoginClients.txt");
      exit(-1);
    }

}

void addPlayer(Player P[],int X,int Y,char nome,int team,char username[],
int *n_players, int sd)
{
         strcpy(P[*n_players].username,username);
         P[*n_players].nome=nome;
         P[*n_players].team=team;
         P[*n_players].X=X;
         P[*n_players].Y=Y;
         P[*n_players].punteggio=5000;
         P[*n_players].indice_player=sd;
         (*n_players)++;

}

void liberaNome(char nome,char nomi[])
{

  if(nome>='A' && nome <= 'Z')
    nomi[nome-65]=nome;
  else
    nomi[nome-71]=nome;

}

void freePlayer(Player P[],int player_eliminato, int *n_players,int fp, char
nomi_verdi[],char nomi_rossi[])
{
    int i,indice_temp,i_temp;

    for(i=0; i<*n_players;i++)
      if(P[i].indice_player == player_eliminato)
      {
          indice_temp=P[i].indice_player;
          i_temp=i;
      }

      if(P[i_temp].team == 'V')
        liberaNome(P[i_temp].nome, nomi_verdi);
      else
        liberaNome(P[i_temp].nome, nomi_rossi);

      if(P[i_temp].punteggio>0)
      {
        write_on_file(fp,2,P[i_temp].username,
P[i_temp].nome,P[i_temp].team,P[i_temp].punteggio,0);
        fprintf(stdout, "\n* Player => P[%d] | Username [%s] : Eliminato per
DISCONNESSIONE *\n",i_temp,P[i_temp].username);
      }
      else
      {
        write_on_file(fp,3,P[i_temp].username,
P[i_temp].nome,P[i_temp].team,P[i_temp].punteggio,0);
```

```
        fprintf(stdout, "\n* Player => P[%d] | Username [%s] : Eliminato per
MORTE *\n",i_temp,P[i_temp].username);
      }

  //  fprintf(stdout, "\nindice_temp=%d && indice_i =
%d",indice_temp,i_temp);

    (*n_players)--;

    for(i=i_temp+1;i<*n_players+2;i++)
    {
          P[i_temp].nome=P[i].nome;
          P[i_temp].team=P[i].team;
          P[i_temp].X=P[i].X;
          P[i_temp].Y=P[i].Y;
          P[i_temp].punteggio=P[i].punteggio;
          strcpy(P[i_temp].username,P[i].username);

          if(i!=*n_players+1)
             P[i_temp].indice_player=P[i].indice_player;


            i_temp=i;
    }
}

void stampaPlayers(Player P[] , int n_players)
{
      int i;

      for(i=0;i<n_players;i++)
      {
        printf("\n[P%d]: USERNAME=[%s] | ",i+1,P[i].username);
        printf(COLOR_RESET "NOME=[%c] | ",P[i].nome);
        printf(COLOR_RESET "TEAM=[%c] | ",P[i].team);
        printf(COLOR_RESET "PUNTEGGIO= [%d] | ",P[i].punteggio);
        printf(COLOR_RESET "X=[%d] | Y=[%d] | ",P[i].X, P[i].Y);
        printf(COLOR_RESET "INDICE=[%d] |",P[i].indice_player);
     }
       printf(COLOR_RESET "\n");

}




int spawnPlayer(int C_X , int C_Y, char nome,char** Mappa,int TOT,Player
P[])
{
    int i;

     char x_char=C_X+'0';
      char y_char=C_Y+'0';



      for(i=0;i<TOT;i++)
      {
              if(Mappa[i][0]==OSTACOLO)
```

```
            {
                if(Mappa[i][1]==x_char &&Mappa[i][2]==y_char)
                    return 0;
            }
      }

  return 1;

}

char** allocaMatrice(int n)
{
    int i;
    char **M=malloc(n*sizeof(char*));

    for(i=0;i<n;i++)
      M[i]=malloc(4*sizeof(char));

//      fprintf(stdout, "\nMatrice Allocata");

    return M;
}



char** GeneraMatrice(int* n_o,int* n_a,int* n_m,int* TOT)
{
     *n_o=(rand()%31)+30;     //da 30 a 60 ostacoli
     *n_a=(rand()%31)+30;     //da 30 a 60 armi
     *n_m=(rand()%51)+50;      //da 50 a 100 mine
     *TOT=(*n_o)+(*n_a)+(*n_m)+2;
     int indice_matrice=0;
  //   fprintf(stdout, "\nPrima di Alloca");
     char** M=allocaMatrice(*TOT);


      generateObstacles(M,*n_o,&indice_matrice);
      generateWeapons(M,*n_a,&indice_matrice);
      generateMines(M,*n_m,&indice_matrice);
      generateFlags(M,*TOT);

  //    fprintf(stdout, "\nDopo i generate");
      return M;

}


void generateObstacles(char **Mappa,int n_ostacoli,int *indice)
{
  int i,C_X,C_Y;
  char C_X_c,C_Y_c;

          for(i=(*indice);i<n_ostacoli;i++)
          {

                C_X=(rand()%27)+1;
                C_Y=(rand()%78)+1;

                if(C_X==11&&(C_Y==8||C_Y==9||C_Y==10))      //sposto
eventuali ostacoli spawnati in cima alla barriera di sinistra, in su
```

```
                    C_X=C_X-1;

                    //sposto eventuali ostacoli spawnati a destra alla
barriera di sinistra,a destra
                if(C_Y==10&&(C_X==12||C_X==13||C_X==14||C_X==15||C_X==16||
C_X==17||C_X==18))
                    C_Y=C_Y+1;

                if(C_X==19&&(C_Y==8||C_Y==9||C_Y==10))    //sposto
eventuali ostacoli spawnati in basso alla barriera di sinistra,in basso
                    C_X=C_X+1;

                    //sposto eventuali ostacoli spawnati nella colonna
della bandiera di sinistra due posti a destra
                if(C_Y==9&&(C_X==12||C_X==13||C_X==14||C_X==15||C_X==16||
C_X==17||C_X==18))
                   C_Y=C_Y+2;



                if(C_X==11&&(C_Y==69||C_Y==70||C_Y==71))      //sposto
eventuali ostacoli spawnati in cima alla barriera di destra, in su
                    C_X=C_X-1;

                    //sposto eventuali ostacoli spawnati a destra alla
barriera di destra,a sinistra
                if(C_Y==69&&(C_X==12||C_X==13||C_X==14||C_X==15||C_X==16||
C_X==17||C_X==18))
                    C_Y=C_Y-1;

                if(C_X==19&&(C_Y==69||C_Y==70||C_Y==71))    //sposto
eventuali ostacoli spawnati in basso alla barriera di destra,in basso
                    C_X=C_X+1;

                    //sposto eventuali ostacoli spawnati nella colonna
della bandiera di destra due posti a sinistra
                 if(C_Y==70&&(C_X==12||C_X==13||C_X==14||C_X==15||C_X==16||
C_X==17||C_X==18))
                     C_Y=C_Y-2;



            C_X_c='0'+C_X;
            C_Y_c='0'+C_Y;

          //  printf("\n [X] C_X=%d -> %c | C_Y=%d -> %c",
C_X,C_X_c,C_Y,C_Y_c);

            Mappa[i][0]=OSTACOLO;
            Mappa[i][1]=C_X_c;
            Mappa[i][2]=C_Y_c;
            Mappa[i][3]='V';
        }


                   *indice=i;
}

void generateWeapons(char **Mappa,int n_weapons,int* indice)
{
```

```
  //printf("indice2=%d\n",*indice);
  //  printf("n_weapins=%d\n",n_weapons);
  int i,C_X,C_Y;
  char C_X_c,C_Y_c;

  for(i=(*indice);i<n_weapons+(*indice);i++)
        {
            C_X=(rand()%27)+1;
            C_Y=(rand()%78)+1;


             if(C_X==11&&(C_Y==8||C_Y==9||C_Y==10))      //sposto eventuali
armi spawnati in cima alla barriera di sinistra, in su
                    C_X=C_X-1;

                    //sposto eventuali armi spawnati a destra alla barriera
di sinistra,a destra
                if(C_Y==10&&(C_X==12||C_X==13||C_X==14||C_X==15||C_X==16||
C_X==17||C_X==18))
                    C_Y=C_Y+1;

                if(C_X==19&&(C_Y==8||C_Y==9||C_Y==10))    //sposto eventuali
armi spawnati in basso alla barriera di sinistra,in basso
                    C_X=C_X+1;



                if(C_X==11&&(C_Y==69||C_Y==70||C_Y==71))      //sposto
eventuali ostacoli spawnati in cima alla barriera di destra, in su
                    C_X=C_X-1;

                    //sposto eventuali ostacoli spawnati a destra alla
barriera di destra,a sinistra
                if(C_Y==69&&(C_X==12||C_X==13||C_X==14||C_X==15||C_X==16||
C_X==17||C_X==18))
                    C_Y=C_Y-1;

                if(C_X==19&&(C_Y==69||C_Y==70||C_Y==71))    //sposto
eventuali ostacoli spawnati in basso alla barriera di destra,in basso
                    C_X=C_X+1;

            C_X_c='0'+C_X;
            C_Y_c='0'+C_Y;

          //   printf("\n [>] C_X=%d -> %c | C_Y=%d -> %c",
C_X,C_X_c,C_Y,C_Y_c);

            Mappa[i][0]=ARMA;
            Mappa[i][1]=C_X_c;
            Mappa[i][2]=C_Y_c;
            Mappa[i][3]='V';
        }

      *indice=i;
}



void generateMines(char **Mappa,int n_mines,int *indice)
{
```

```
//  printf("indice3=%d\n",*indice);
//printf("n_mine=%d\n",n_mines);
int i,C_X,C_Y;
char C_X_c,C_Y_c;

for(i=(*indice);i<n_mines+(*indice);i++)
      {
          C_X=(rand()%27)+1;
          C_Y=(rand()%78)+1;

           if(C_X==11&&(C_Y==8||C_Y==9||C_Y==10))      //sposto eventuali
mine spawnati in cima alla barriera di sinistra, in su
                  C_X=C_X-1;

                  //sposto eventuali mine spawnati a destra alla barriera di
sinistra,a destra
             if(C_Y==10&&(C_X==12||C_X==13||C_X==14||C_X==15||C_X==16||
C_X==17||C_X==18))
                  C_Y=C_Y+1;

             if(C_X==19&&(C_Y==8||C_Y==9||C_Y==10))    //sposto eventuali
mine spawnati in basso alla barriera di sinistra,in basso
                  C_X=C_X+1;


             if(C_X==11&&(C_Y==69||C_Y==70||C_Y==71))      //sposto
eventuali ostacoli spawnati in cima alla barriera di destra, in su
                  C_X=C_X-1;

                  //sposto eventuali ostacoli spawnati a destra alla
barriera di destra,a sinistra
             if(C_Y==69&&(C_X==12||C_X==13||C_X==14||C_X==15||C_X==16||
C_X==17||C_X==18))
                  C_Y=C_Y-1;

             if(C_X==19&&(C_Y==69||C_Y==70||C_Y==71))    //sposto eventuali
ostacoli spawnati in basso alla barriera di destra,in basso
                  C_X=C_X+1;

          C_X_c='0'+C_X;
          C_Y_c='0'+C_Y;
      //    printf("\n [*] C_X=%d -> %c | C_Y=%d -> %c",
C_X,C_X_c,C_Y,C_Y_c);
          Mappa[i][0]=MINA;
          Mappa[i][1]=C_X_c;
          Mappa[i][2]=C_Y_c;
          Mappa[i][3]='V';
      }

   *indice=i;

}

void generateFlags(char **Mappa,int Tot)
{

  int C_X,C_Y;
   char C_X_c,C_Y_c;
```

```
    C_X=(rand()%7)+12;
    C_Y=9;                  //da 5 a 9

     C_X_c='0'+C_X;
     C_Y_c='0'+C_Y;
    // printf("\n [F] C_X=%d -> %c | C_Y=%d -> %c", C_X,C_X_c,C_Y,C_Y_c);
    Mappa[Tot-2][0]=FLAG;
    Mappa[Tot-2][1]=C_X_c;
    Mappa[Tot-2][2]=C_Y_c;
    Mappa[Tot-2][3]='V';


    C_X=(rand()%7)+12;
    C_Y=70;                 //da 70 a 74

  //  printf("\n [F] C_X=%d -> %c | C_Y=%d -> %c", C_X,C_X_c,C_Y,C_Y_c);
    C_X_c='0'+C_X;
    C_Y_c='0'+C_Y;

    Mappa[Tot-1][0]=FLAG;
    Mappa[Tot-1][1]=C_X_c;
    Mappa[Tot-1][2]=C_Y_c;
    Mappa[Tot-1][3]='V';

}

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


int SendData(char **Matrice, int TOT_OBJECTS,int sd)
{
    int TOT_OBJECTS_conv=0;

    TOT_OBJECTS_conv=htonl(TOT_OBJECTS);

    fprintf(stdout, "\nINVIO %d", TOT_OBJECTS);

    if( (send(sd, (const char*)&TOT_OBJECTS_conv, 4, 0)) < 0 )
    {
      perror("Error Write TOT_OBJECTS_conv");
      return 1;
    }

    int i,j;

    for(i=0;i<TOT_OBJECTS;i++)
      for(j=0;j<4;j++)
          if( (write(sd, &Matrice[i][j], 1)) < 0 )
          {
            perror("Error Write Carattere Matrice");
            return 1;
          }

    return 0;

}
```

```
int readRequestMovementClient(int sd,char **Mappa, Player P[],int n_players,
int TOT_OBJECTS, int *n_oggetti,int *fine_game,int fp)
{
    char spostamento,username[20];
    int nbytes, check = 0;

    if( (nbytes=read(sd, username, 20)) < 0)
    {
      perror("\nError username readRequestMovementClient()");
      return 2;
    }

    username[nbytes]='\0';

    if( (read(sd,&spostamento,1)) < 0 )
    {
      perror("\nError spostamento readRequestMovementClient()");
      return 2;
    }

    if(spostamento != 'm')
    {
    fprintf(stdout, "\nLetta Richiesta di spostamento da [%s] in
[%c]",username, spostamento);


check=check_movement_client(username,sd,Mappa,P,n_players,spostamento,TOT_OB
JECTS,n_oggetti,fine_game,fp);
    stampaPlayers(P,n_players);
   }

  return check;


}

int find_player_by_username(Player P[], char username[], int n_players, int
*X , int *Y , int *indice)
{
  int i;

    for(i=0; i<n_players; i++)
    {
           if(strcmp(P[i].username,username)==0)
           {
              *X=P[i].X;
              *Y=P[i].Y;
              *indice=i;
              return 1;
           }
    }

    return 0;
}

void find_and_destroy_object(char **Mappa, int X, int Y, int TOT_OBJECTS,
char OBJECT,int *X_O)
{
```

```
  int i;

//fprintf(stdout, "\nFIND AND DESTROY (%d)(%d)(%c)",X,Y,OBJECT);
    for(i=0;i<TOT_OBJECTS;i++)
    {
      //fprintf(stdout, "\n[%c][%d][%d][%c]", Mappa[i][0], Mappa[i][1] -
'0', Mappa[i][2] - '0',Mappa[i][3]);
      if(decode_char(Mappa[i][1]) == X && decode_char(Mappa[i][2]) == Y &&
Mappa[i][0] == OBJECT)
        *X_O=i;

    }

}

int check_plaver_vs_player(Player P[], int n_players,int indice,char
spostamento)
{
  int i;
    for(i=0;i<n_players;i++)
    {
      if(spostamento == 'a')
      {
       if(P[indice].X == P[i].X && P[indice].Y - 1  == P[i].Y)
        if(P[indice].team != P[i].team)
          if(P[indice].punteggio >= P[i].punteggio)
          {
            P[i].punteggio=P[i].punteggio - 200;

            if(P[i].team == 'V')
            {
             P[i].X=15;
             P[i].Y=8;
            }
            else
            {
              P[i].X=15;
              P[i].Y=71;
            }

            return 1;
          }
          else
          {
            P[indice].punteggio=P[indice].punteggio - 200;

            if(P[indice].team == 'V')
            {
             P[indice].X=15;
             P[indice].Y=8;
            }
            else
            {
              P[indice].X=15;
              P[indice].Y=71;
            }

            return 1;
          }
        }
```

```
else if (spostamento == 'd')
{
  if(P[indice].X == P[i].X && P[indice].Y + 1  == P[i].Y)
   if(P[indice].team != P[i].team)
     if(P[indice].punteggio >= P[i].punteggio)
     {
       P[i].punteggio=P[i].punteggio - 200;

       if(P[i].team == 'V')
       {
        P[i].X=15;
        P[i].Y=8;
       }
       else
       {
         P[i].X=15;
         P[i].Y=71;
       }

       return 1;
     }
     else
     {
       P[indice].punteggio=P[indice].punteggio - 200;

       if(P[indice].team == 'V')
       {
        P[indice].X=15;
        P[indice].Y=8;
       }
       else
       {
         P[indice].X=15;
         P[indice].Y=71;
       }

       return 1;
     }
}
else if (spostamento == 'w')
{
  if(P[indice].X - 1 == P[i].X && P[indice].Y  == P[i].Y)
   if(P[indice].team != P[i].team)
     if(P[indice].punteggio >= P[i].punteggio)
     {
       P[i].punteggio=P[i].punteggio - 200;

       if(P[i].team == 'V')
       {
        P[i].X=15;
        P[i].Y=8;
       }
       else
       {
         P[i].X=15;
         P[i].Y=71;
       }

       return 1;
     }
```

```
        else
        {
          P[indice].punteggio=P[indice].punteggio - 200;

          if(P[indice].team == 'V')
          {
           P[indice].X=15;
           P[indice].Y=8;
          }
          else
          {
            P[indice].X=15;
            P[indice].Y=71;
          }

          return 1;
        }
    }
    else
    {
      if(P[indice].X + 1 == P[i].X && P[indice].Y  == P[i].Y)
       if(P[indice].team != P[i].team)
         if(P[indice].punteggio >= P[i].punteggio)
         {
           P[i].punteggio=P[i].punteggio - 200;

           if(P[i].team == 'V')
           {
            P[i].X=15;
            P[i].Y=8;
           }
           else
           {
             P[i].X=15;
             P[i].Y=71;
           }

           return 1;
         }
         else
         {
           P[indice].punteggio=P[indice].punteggio - 200;

           if(P[indice].team == 'V')
           {
            P[indice].X=15;
            P[indice].Y=8;
           }
           else
           {
             P[indice].X=15;
             P[indice].Y=71;
           }

           return 1;
         }
    }


}
```

```
   return 0;
}


int check_movement_client(char username[], int sd, char ** Mappa, Player
P[], int n_players, char spostamento , int TOT_OBJECTS, int *n_oggetti, int
*fine_game, int fp)
{
  int X,Y,indice,X_flag,Y_flag,X_O;

      //trovo le cordinate del player mediante l'username
      if(!find_player_by_username(P,username,n_players,&X,&Y,&indice))
          return 0;

      fprintf(stdout, "\nTrovate Cordinate di %s = in P[%d][ %d , %d ]",
username, indice, X,Y);

      char object;



     if(P[indice].team == 'V')
      {
          X_flag=decode_char(Mappa[TOT_OBJECTS-2][1]);
          Y_flag=decode_char(Mappa[TOT_OBJECTS-2][2]);
     }
      else
      {
          X_flag=decode_char(Mappa[TOT_OBJECTS-1][1]);
          Y_flag=decode_char(Mappa[TOT_OBJECTS-1][2]);
       }

      // printf("\nstampa team=%c - FLAG (%d,
%d)\n",P[indice].team,X_flag,Y_flag);

  switch (spostamento) {

                case 'a':

                    if((P[indice].Y-1)==10 &&(P[indice].X==11 ||
P[indice].X==12|| P[indice].X==13|| P[indice].X==14|| P[indice].X==15 ||
P[indice].X==16|| P[indice].X==17|| P[indice].X==18|| P[indice].X==19) )
                      {
                          fprintf(stdout, "\nSPOSTAMENTO NON VALIDO_1\n");
                      }
                      else if(((P[indice].Y-1)==71 &&P[indice].X==19)||
((P[indice].Y-1)==71 &&P[indice].X==11)||((P[indice].Y-
1)==69&&(P[indice].X==12 ||P[indice].X==13 || P[indice].X==14 ||
P[indice].X==15 || P[indice].X==16 ||P[indice].X==17 ||P[indice].X==18)))
                      {
                          fprintf(stdout, "\nSPOSTAMENTO NON VALIDO_2\n");
                      }
                      else if ((P[indice].Y - 1 == Y_flag) && (P[indice].X
== X_flag)) {
                          fprintf(stdout, "\nSPOSTAMENTO NON VALIDO SULLA
FLAG\n");
                      }
                      else
if(check_plaver_vs_player(P,n_players,indice,'a'))
                      {
```

```
                        puts("Collisione con Player");
                        P[indice].Y--;
                        P[indice].punteggio--;
                      }
                      else
                      {
                          if(P[indice].Y-1>0 )
                          {
                              object=searchData(Mappa,X,Y-1,TOT_OBJECTS);
                              fprintf(stdout, "\nobject [%c]\n", object);

                                if(object != 'N')
                                {
                                  fprintf(stdout, "\nsono entrato in X\n");
                                  if(object == MINA)
                                  {
                                    P[indice].Y -= 1;

find_and_destroy_object(Mappa,P[indice].X,P[indice].Y,TOT_OBJECTS,MINA,&X_O)
;

                                    if(Mappa[X_O][3] != '+')
                                    {
                                      Mappa[X_O][3]='+';
                                      P[indice].punteggio -= 600;

addObject(n_oggetti,P[indice].nome,P[indice].team,P[indice].username,P[indic
e].X,P[indice].Y,MINA);
                                    }
                                    else
                                      P[indice].punteggio--;

                                    return 1;
                                  }
                                  else if (object == ARMA)
                                  {
                                    P[indice].Y -= 1;



find_and_destroy_object(Mappa,P[indice].X,P[indice].Y,TOT_OBJECTS,ARMA,&X_O)
;

                                    if(Mappa[X_O][3] != '+')
                                    {
                                    Mappa[X_O][3]='+';
                                    P[indice].punteggio += 300;

addObject(n_oggetti,P[indice].nome,P[indice].team,P[indice].username,P[indic
e].X,P[indice].Y,ARMA);
                                    }
                                    else
                                      P[indice].punteggio--;

                                     return 1;
                                  }
                                  else if (object == FLAG)
                                  {
                                  if(P[indice].team == 'V')
```

```
                                    *fine_game=1;
                                  else
                                     *fine_game=2;

                                     P[indice].Y -= 1;

addObject(n_oggetti,P[indice].nome,P[indice].team,P[indice].username,P[indic
e].X,P[indice].Y,FLAG);

write_on_file(fp,4,P[indice].username,P[indice].nome,P[indice].team,P[indice
].punteggio,0);
                                     return 0;
                                   }
                                }
                                else if (object == 'N')
                                {
                                  fprintf(stdout, "\nsono entrato in N\n");
                                  P[indice].Y = P[indice].Y - 1;
                                  P[indice].punteggio--;
                                  return 1;
                                }
                            }
                         }

                      break;

            case 'd':

            if(((P[indice].Y+1)==8 &&P[indice].X==19)||((P[indice].Y+1)==8
&&P[indice].X==11)||((P[indice].Y+1)==10&&(P[indice].X==12 ||P[indice].X==13
|| P[indice].X==14 || P[indice].X==15 || P[indice].X==16 ||P[indice].X==17
||P[indice].X==18)))
                {
                       fprintf(stdout, "\nSPOSTAMENTO NON VALIDO_1\n");
                }
                 else if((P[indice].Y+1)==69 &&(P[indice].X==11 ||
P[indice].X==12|| P[indice].X==13|| P[indice].X==14|| P[indice].X==15 ||
P[indice].X==16|| P[indice].X==17|| P[indice].X==18 || P[indice].X==19 ) )
              {
                        fprintf(stdout, "\nSPOSTAMENTO NON VALIDO_2\n");
              }
              else if ((P[indice].Y + 1 == Y_flag) && (P[indice].X ==
X_flag)) {
                  fprintf(stdout, "\nSPOSTAMENTO NON VALIDO SULLA FLAG\n");
              }
              else if(check_plaver_vs_player(P,n_players,indice,'d'))
              {
                puts("Collisione con Player");
                P[indice].Y++;
                P[indice].punteggio--;
              }
              else
              {
                       if(P[indice].Y+1<79)
                       {
                           object=searchData(Mappa,X,Y+1,TOT_OBJECTS);
                           fprintf(stdout, "\nobject [%c]\n", object);

                         if(object != 'N')
```

```
                        {
                          fprintf(stdout, "\nsono entrato in X\n");
                          if(object == MINA)
                          {
                            P[indice].Y += 1;


find_and_destroy_object(Mappa,P[indice].X,P[indice].Y,TOT_OBJECTS,MINA,&X_O)
;

                            if(Mappa[X_O][3] != '+')
                            {
                              Mappa[X_O][3]='+';
                              P[indice].punteggio -= 600;

addObject(n_oggetti,P[indice].nome,P[indice].team,P[indice].username,P[indic
e].X,P[indice].Y,MINA);
                            }
                            else
                              P[indice].punteggio--;

                            return 1;
                          }
                          else if (object == ARMA)
                          {
                            P[indice].Y += 1;


find_and_destroy_object(Mappa,P[indice].X,P[indice].Y,TOT_OBJECTS,ARMA,&X_O)
;

                            if(Mappa[X_O][3] != '+')
                            {
                            Mappa[X_O][3]='+';
                            P[indice].punteggio += 300;

addObject(n_oggetti,P[indice].nome,P[indice].team,P[indice].username,P[indic
e].X,P[indice].Y,ARMA);
                            }
                            else
                              P[indice].punteggio--;

                             return 1;
                          }
                          else if (object == FLAG)
                          {
                            if(P[indice].team == 'V')
                              *fine_game=1;
                            else
                               *fine_game=2;

                              P[indice].Y += 1;

addObject(n_oggetti,P[indice].nome,P[indice].team,P[indice].username,P[indic
e].X,P[indice].Y,FLAG);

write_on_file(fp,4,P[indice].username,P[indice].nome,P[indice].team,P[indice
].punteggio,0);
                             return 0;
```

```
                        }
                      }
                      else if (object == 'N')
                      {
                        fprintf(stdout, "\nsono entrato in N\n");
                        P[indice].Y = P[indice].Y + 1;
                        P[indice].punteggio--;
                        return 1;
                      }
                     }
                }

                 break;



                   case 'w':

            //     if(  (P[indice].X==20&&(P[indice].Y==8 ||P[indice].Y==9||
P[indice].Y==10) ) || (P[indice].X==12&&(P[indice].Y==8 ||P[indice].Y==9) )
)
               if(  (P[indice].X==20&&(P[indice].Y==8 ||P[indice].Y==9||
P[indice].Y==10 ||P[indice].Y==69 ||P[indice].Y==70||P[indice].Y==71 ) ) ||
(P[indice].X==12&&(P[indice].Y==8 ||P[indice].Y==9||P[indice].Y==70 ||
P[indice].Y==71) )   )
                  {
                      printf("no\n");
                  }
                  else if ((P[indice].Y == Y_flag) && (P[indice].X - 1 ==
X_flag)) {
                      fprintf(stdout, "\nSPOSTAMENTO NON VALIDO SULLA
FLAG\n");
                  }
                  else if(check_plaver_vs_player(P,n_players,indice,'w'))
                  {
                    puts("Collisione con Player");
                    P[indice].X--;
                    P[indice].punteggio--;
                  }
                  else{

                      if(P[indice].X-1>0)
                      {
                      object=searchData(Mappa,X-1,Y,TOT_OBJECTS);
                      fprintf(stdout, "\nobject [%c]\n", object);

                    if(object != 'N')
                    {

                      if(object == MINA)
                      {
                        P[indice].X -= 1;


find_and_destroy_object(Mappa,P[indice].X,P[indice].Y,TOT_OBJECTS,MINA,&X_O)
;


                        if(Mappa[X_O][3] != '+')
                        {
```

```
                        Mappa[X_O][3]='+';
                        P[indice].punteggio -= 600;

addObject(n_oggetti,P[indice].nome,P[indice].team,P[indice].username,P[indic
e].X,P[indice].Y,MINA);
                      }
                      else
                        P[indice].punteggio--;

                      return 1;
                    }
                    else if (object == ARMA)
                    {
                      P[indice].X -= 1;



find_and_destroy_object(Mappa,P[indice].X,P[indice].Y,TOT_OBJECTS,ARMA,&X_O)
;

                      if(Mappa[X_O][3] != '+')
                      {
                      Mappa[X_O][3]='+';
                      P[indice].punteggio += 300;

addObject(n_oggetti,P[indice].nome,P[indice].team,P[indice].username,P[indic
e].X,P[indice].Y,ARMA);
                      }
                      else
                        P[indice].punteggio--;

                       return 1;
                    }
                    else if (object == FLAG)
                    {
                      if(P[indice].team == 'V')
                        *fine_game=1;
                      else
                         *fine_game=2;

                       P[indice].X -= 1;

write_on_file(fp,4,P[indice].username,P[indice].nome,P[indice].team,P[indice
].punteggio,0);

addObject(n_oggetti,P[indice].nome,P[indice].team,P[indice].username,P[indic
e].X,P[indice].Y,FLAG);
                       return 0;
                    }
                  }
                  else if (object == 'N')
                  {
                    fprintf(stdout, "\nsono entrato in N\n");
                    P[indice].X = P[indice].X - 1;
                    P[indice].punteggio--;
                    return 1;
                  }
              }
          }
```

```
                    break;

                    case 's':
                  //  if(  (P[indice].X==10&&(P[indice].Y==8 ||
P[indice].Y==9 ||P[indice].Y==10 )) || (P[indice].X==18&&(P[indice].Y==8 ||
P[indice].Y==9 )))
                  if(  (P[indice].X==10&&(P[indice].Y==8 ||P[indice].Y==9 ||
P[indice].Y==10|| P[indice].Y==69 ||P[indice].Y==70 ||P[indice].Y==71)) ||
(P[indice].X==18&&(P[indice].Y==8 ||P[indice].Y==9 ||P[indice].Y==70 ||
P[indice].Y==71)))
                    {
                        printf("\n");
                    }
                    else if ((P[indice].Y == Y_flag) && (P[indice].X + 1 ==
X_flag)) {
                        fprintf(stdout, "\nSPOSTAMENTO NON VALIDO SULLA
FLAG\n");
                    }
                    else if(check_plaver_vs_player(P,n_players,indice,'s'))
                    {
                      puts("Collisione con Player");
                      P[indice].X++;
                      P[indice].punteggio--;
                    }
                    else
                    {
                    if(P[indice].X+1<29)
                    {
                        object=searchData(Mappa,X+1,Y,TOT_OBJECTS);
                        fprintf(stdout, "\nobject [%c]\n", object);

                      if(object != 'N')
                      {
                        fprintf(stdout, "\nsono entrato in X\n");

                        if(object == MINA)
                        {
                          P[indice].X += 1;


find_and_destroy_object(Mappa,P[indice].X,P[indice].Y,TOT_OBJECTS,MINA,&X_O)
;

                            if(Mappa[X_O][3] != '+')
                            {
                              Mappa[X_O][3]='+';
                              P[indice].punteggio -= 600;

//write_on_file(fp,3,P[indice].username,P[indice].nome,P[indice].team,P[indi
ce].punteggio,0);

addObject(n_oggetti,P[indice].nome,P[indice].team,P[indice].username,P[indic
e].X,P[indice].Y,MINA);
                            }
                            else
                              P[indice].punteggio--;
```

```
                            return 1;
                          }
                          else if (object == ARMA)
                          {
                            P[indice].X += 1;


find_and_destroy_object(Mappa,P[indice].X,P[indice].Y,TOT_OBJECTS,ARMA,&X_O)
;

                            if(Mappa[X_O][3] != '+')
                            {
                            Mappa[X_O][3]='+';
                            P[indice].punteggio += 300;

addObject(n_oggetti,P[indice].nome,P[indice].team,P[indice].username,P[indic
e].X,P[indice].Y,ARMA);
                            }
                            else
                              P[indice].punteggio--;

                             return 1;
                          }
                          else if (object == FLAG)
                          {
                            if(P[indice].team == 'V')
                              *fine_game=1;
                            else
                               *fine_game=2;

                             P[indice].X += 1;

write_on_file(fp,4,P[indice].username,P[indice].nome,P[indice].team,P[indice
].punteggio,0);

addObject(n_oggetti,P[indice].nome,P[indice].team,P[indice].username,P[indic
e].X,P[indice].Y,FLAG);
                             return 0;
                          }
                        }
                        else if (object == 'N')
                        {
                          fprintf(stdout, "\nsono entrato in N\n");
                          P[indice].X = P[indice].X + 1;
                          P[indice].punteggio--;
                          return 1;
                      }
                  }
              }


                    break;
          }

}

char searchData(char **M, int X , int Y, int TOT)
{
    int i;
```

```
      for(i=0;i<TOT;i++)
          if((decode_char(M[i][1]) == X)  && (decode_char(M[i][2]) == Y))
          {
            fprintf(stdout, "\nCollisione con [%c]", M[i][0] );
            return M[i][0];
          }

    return 'N';
}

void addObject(int* n_oggetti,char nome_p,char team,char username[],int
X,int Y,char nome_o)
{
      strcpy(O[*n_oggetti].username,username);
      O[*n_oggetti].nome_p=nome_p;
      O[*n_oggetti].nome_o=nome_o;
      O[*n_oggetti].team=team;
      O[*n_oggetti].X=X;
      O[*n_oggetti].Y=Y;

      (*n_oggetti)++;

}

int decode_char(char c)
{
    int n=c-'0';

      return n;
}

int read_Request_sendObjectDestroyed(int sd,int n_oggetti)
{
  char ch;
  int check=0;

  if( (read(sd,&ch,1)) < 0 )
  {
    perror("Error read read_Request_sendObjectDestroyed");
    return 1;
  }

  if(ch == 'Y')
    check=sendObjectDestroyed(sd,n_oggetti);

  return check;
}

int sendObjectDestroyed(int sd, int n_oggetti)
{
    char team  , temp[4] , n_letture_c;
    int i, n_letture=0;


    if( (read(sd,&team,1)) < 0 )
    {
      perror("Error read team sendObjectDestryed()");
      return 1;
    }
```

```
    for(i=0;i<n_oggetti;i++)
      if(team==O[i].team)
          n_letture++;

    n_letture_c=n_letture +'0';

    if( (write(sd,&n_letture_c,1)) < 0 )
    {
      perror("Error send n_letture in sendObjectDestryed");
      return 1;
    }


    for(i=0;i<n_oggetti;i++)
    {
      if(team == O[i].team)
      {

          if( ((write(sd,O[i].username, 20)) < 0 ))
          {
            perror("Error write username in sendObjectDestroyed");
            return 1;

          }

          temp[0]=O[i].nome_o;
          temp[1]=O[i].X + '0';
          temp[2]=O[i].Y + '0';
          temp[3]=O[i].nome_p;

          if( ((write(sd,temp,4)) < 0 ))
          {
            perror("Error write temp in sendObjectDestroyed");
            return 1;
          }

      }
    }

    return 0;
}

int sendPlayers(Player P[], int n_players, int sd)
{
    int i;
    char  X_c, Y_c , n_players_c,sd_temp,punteggio_string[10];
    char temp[5];

    n_players_c=n_players + '0';

  //  fprintf(stdout, "\nSto inviando [%d] ==> [%c]", n_players ,
n_players_c );

    if( (write(sd, &n_players_c, 1)) < 0 )
    {
      perror("Error Write n_players in sendPlayers");
      return 1;
    }
```

```
    for(i=0;i<n_players;i++)
    {

         X_c= P[i].X + '0';
         Y_c= P[i].Y + '0';
         sd_temp=P[i].indice_player + '0';

            temp[0]=P[i].nome;
            temp[1]=P[i].team;
            temp[2]=X_c;
            temp[3]=Y_c;
            temp[4]=sd_temp;

      fprintf(stdout, "\nInvio nome=[%c] | squadra=[%c] | X=[%c] Y=[%c] |
INDICE = [%c]\n", temp[0] , temp[1] , temp[2] , temp[3], temp[4]);


           if( (write(sd,temp,5)) < 0 )
           {
             perror("Error Write data in sendPlayers()");
             return 1;
           }

    //       puts("STO PROVANDO A MANDARE L'USERNAME");
          if( (write(sd,P[i].username,20)) < 0 )
           {
             perror("Error Write username in sendPlayers");
             return 1;
           }
      //     puts("HO MANDATO L'USERNAME");

          sprintf(punteggio_string,"%d",P[i].punteggio);

          if( (write(sd,punteggio_string,10)) < 0 )
          {
            perror("Error Write punteggio_string in sendPlayers");
            return 1;
          }


       }

    return 0;

}

int send_fine_game(int sd,int fine_game,Player P[],int* n_players,int fp,
char nomi_verdi[], char nomi_rossi[])
{
  if(fine_game==1)
  {
    fprintf(stdout, "\n\n\nFINE PARTITA!");
    if( (write(sd,"F",1)) < 0 )
    {
      fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
      perror("Error send fine_game Server.c");
    }

    freePlayer(P,sd,n_players,fp,nomi_verdi,nomi_rossi);
```

```
    close(sd);
    return 1;
  }
  else if(fine_game==2)
  {
    if( (write(sd,"B",1)) < 0 )
    {
      fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
      perror("Error send fine_game Server.c");
    }

    freePlayer(P,sd,n_players,fp,nomi_verdi,nomi_rossi);
    close(sd);
    return 1;
  }
  if(fine_game==3)
  {
    if( (write(sd,"P",1)) < 0 )
    {
      fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
      perror("Error send fine_game Server.c");
    }

    freePlayer(P,sd,n_players,fp,nomi_verdi,nomi_rossi);
    close(sd);
    return 1;
  }
  else
  {
    if( ((write(sd,"Q",1)) < 0 ))
    {
      perror("Error send fine_game Server.c");
      fprintf(stdout, "\nClient_Socket[%d] Disconnect\n", sd);
      freePlayer(P,sd,n_players,fp,nomi_verdi,nomi_rossi);
      close(sd);

    }
    return 0;
  }
}
```

Codice Sorgente: Client

Client.c

```
#include <errno.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/types.h>
```

```
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <pthread.h> // Threads
#include <sys/socket.h> // Socket
#include <sys/un.h> // Connection
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <signal.h>
#include <assert.h>
#include "Funzioni_Client/funzioni_client.h"
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
//pthread_cond_t v;

#define BACKLOG 20
#define RIGHE 30
#define COLONNE 80

int n_players=0;
char **Mappa;
char username[20];
char spostamento;
int indice;
int TOT;



//gcc client.c Funzioni_Client/funzioni_client.c -o client  -lpthread &&
./client
void INThandler(int);


int main (int argc, char **argv)
{
  int sd; // id del socket
  int richiesta;
  struct sockaddr_in serv_ind;
  struct hostent *ipserv;
  int porta;
  pthread_t thread_id1,thread_id2;
  int Coordinata_X,Coordinata_Y;
  Mappa=GeneraMappa();
  char temp_sd,team_char;
  int ok,io=0;


  if( ( argc ) < 3 )
  {
    fprintf(stdout, "\nArgomento non valido, inserisci l'indizzo IP del
Server e la Porta\n Esempio: ./server 95.13.14.156 8080\n");
    exit(-1);
  }

  porta=atoi(argv[2]);
  fprintf(stdout, "\nConnessione ==> [%s]:[%d]\n", argv[1],porta);

  //==================CONF. INDIZZO IP==============================
  memset((void *)&serv_ind, 0, sizeof(serv_ind)); // pulizia ind
  serv_ind.sin_family = AF_INET; // ind di tipo INET
```

```
serv_ind.sin_port = htons(porta); // porta a cui collegarsi
inet_aton(argv[1],&serv_ind.sin_addr);
//=====================================================================

if ( (sd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
   perror("Errore in creazione socket");
   exit(-1);
}

if (connect(sd, (struct sockaddr *) &serv_ind, sizeof(serv_ind)) == -1) {
perror("Errore nella connessione");
exit(-1);
}


if( (read(sd,&temp_sd,1)) < 0 )
{
  perror("Error read indice");
  exit(-1);
}

indice=temp_sd - '0';
Logo();
loginClient(sd,username);
int team=teamSelection(sd);

if(team==0) team_char='V';
else team_char='R';

char nome=requestName(sd);

 do
 {
  ok=requestToSpawn(&Coordinata_X,&Coordinata_Y,team,sd,Mappa);
  fprintf(stdout, "Ho restituito ok=%d", ok );
  if(ok==0)
    fprintf(stdout,"\nPosizione Errata!\n");
 }
 while(ok==0);

 riepilogoDati(nome,team,Coordinata_X,Coordinata_Y);

 receivePlayers(sd,Mappa,&n_players);
 ReceiveData(Mappa,&TOT,sd);
 io=find_me(n_players,indice);
 stampaMappa(Mappa,n_players,io);

while(1)
{

    //receiveObjectDestroyed(sd,team_char);
    richiesta=Menu(Mappa,&n_players,io,sd,username);
    receivePlayers(sd,Mappa,&n_players);
    ReceiveData(Mappa,&TOT,sd);
    io=find_me(n_players,indice);

    if(check_my_punteggio(sd,io))
      break;
```

```
      system("clear");
      stampaMappa(Mappa,n_players,io);
      request_receive_objects_destroyed(sd,richiesta,team_char);

      if(richiesta>=0)
      {
        io=find_me(n_players,indice);
        stampaMappa(Mappa,n_players,io);
      }

      if(receive_fine_game(sd))
      {
        stampaMappa(Mappa,n_players,io);
        break;
      }

  }


 return 0;
}
```

Funzioni_client.c

```
#include <errno.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <pthread.h> // Threads
#include <sys/socket.h> // Socket
#include <sys/un.h> // Connection
#include <stdio.h>
#include <stdlib.h>
#include "funzioni_client.h"

char ARMA = 124;                 // '|'
char MINA = '*';
char OSTACOLO = 94;             //^
char FLAG = 36;                 //$


#define RED     "\x1b[31m"
#define GREEN   "\x1b[32m"
#define COLOR_RESET   "\x1b[0m"


#define SIZE_RIGHE 30
#define SIZE_COLONNE 80
```

```
void Logo()
{
      char* logo[8];
      system("@cls||clear");

      logo[0]="\t\t\t\t\t\t\t\t\t\t\t BENVENUTO IN";
      logo[1]=GREEN"\t\t\t\t\t
" RED "
      logo[2]=GREEN"\t\t\t\t\t
" RED "
      logo[3]=GREEN"\t\t\t\t\t
" RED "
      logo[4]=GREEN"\t\t\t\t\t
" RED "
      logo[5]=GREEN"\t\t\t\t\t
" RED "
      logo[6]=GREEN"\t\t\t\t\t
" RED "";


      printf("%s",logo[0]);
      printf("\n\n");
      printf("%s",logo[1]);
      printf("\n");
       printf("%s",logo[2]);
       printf("\n");
      printf("%s",logo[3]);
       printf("\n");
       printf("%s",logo[4]);
        printf("\n");
      printf("%s",logo[5]);
       printf("\n");
       printf("%s",logo[6]);
        printf("\n");
      printf(COLOR_RESET"");
      printf("\n\n");

}


int loginClient(int sd,char username[])
{
       char password[20];
       char tmpchar,go[20];
       int n;
       char risposta_Server;

  do {
    //  clearBuffer();
      printf("\nIl tuo account è gia registrato? (y/n): ");

          do
         {
            scanf(" %c",&tmpchar);
         }while(  (tmpchar!='y'&&tmpchar!='n') && printf("tasto non valido,
Riscrivi : ")  );



       if( (write(sd,&tmpchar,1)) < 0 )
```

```
      {
        perror("Error Write Require Registration on DB");
        exit(-1);
      }


    do
    {
      printf("\n[USERNAME(max 9 caratteri)]: ");
      clearBuffer();
      fgets(username,20,stdin);
      username[strlen(username)-1]='\0';

    } while((strlen(username) > 9  && fprintf(stdout, "Lunghezza non
valida!")));


      if( (write(sd,username,strlen(username))) < 0 )
      {
        perror("Error Write username to Server");
        exit(-1);
      }

    do
    {
      printf("\n[PASSWORD(max 9 caratteri)]: ");
      fgets(password,20,stdin);
      password[strlen(password)-1]='\0';
    }while((strlen(password)>9  &&  fprintf(stdout, "Lunghezza non
valida!")));

    fprintf(stdout, "Inseriti Username=[%s] Password=[%s]", username,
password);


      if( (write(sd,password, strlen(password))) < 0 )
      {
        perror("Error Write Password to Server");
        exit(-1);
      }


        if( (read(sd,&risposta_Server,1)) < 0 )
        {
          perror("Error read risposta_Server");
        }

       fprintf(stdout, "\nrisposta_Server=[%c]", risposta_Server );


      if(risposta_Server == 'F')
        printf("\nLogin Errato\n");


      else if(risposta_Server == 'X')
        printf("\nUTENTE GIA' LOGGATO CON QUESTO USERNAME\n");

      else
        printf("\nLogin EFFETTUATO!\n");
```

```
    //  clearBuffer();
    } while(risposta_Server != 'Y');



}



int Menu(char** M, int *n_players,int io, int sd, char username[])
//da aggiungere al progetto
{
    int richiesta=-1;
    char tasto; //   SPOSTAMENTO!!!!!!!


    printf("[a-w-d-s] per muoverti, m per aprire il menu\ninput : ");
     do
     {
        while((tasto=getchar()!='\n')&& (tasto!=EOF));
        scanf("%c",&tasto);

     }while(  (tasto!='a'&&tasto!='w'&&tasto!='d'&&tasto!='s'&&tasto!='m')
&& printf("tasto non valido, Riscrivi : ")  );

       RequestMovementClient(sd, tasto, username);

                    // system("@cls||clear");
                if(tasto == 'm')
                {

printf("x-----------------------------------------------------------------x");
                    printf("\n| 0) Per disconnetterti\t\t\t\t\t     |\n|
1) Per vedere la lista degli utenti collegati\t\t     |\n| 2) Per vedere i
tuoi compagni\t\t\t\t     |\n| ");
                    printf("3) Per vedere la posizione degli oggetti
incontrati \t     |\n");

printf("x-----------------------------------------------------------------x");
                    printf("\n  SCELTA = ");
                    scanf("%d",&richiesta);

                     switch(richiesta)
                     {
                         case 0:
                              close(sd);
                              exit(1);
                         break;

                         case 1:

                             stampaPlayers(M,*n_players,io);
                             //richiesta=-1;
                             Pulisci();
                         break;

                         case 2:
```

```
stampaCompagni(M,P[io].team,*n_players,io);
                                //    richiesta=-1;
                                     Pulisci();
                                break;

                                case 3:
                                     //richiesta=-1;
                                    // Pulisci();
                                break;

                                default:
                                    printf("\nScelta non valida");
                                //    richiesta=-1;
                                    Pulisci();
                                break;
                            }
                        }


return richiesta;


}

int check_my_punteggio(int sd , int io)
{
    if(P[io].punteggio <= 0)
    {
      fprintf(stdout, "\n\nSEI STATO ELIMINATO!\n");
      close(sd);
      return 1;
    }
return 0;
}

void request_receive_objects_destroyed(int sd,int richiesta,char team_char)
{
   if(richiesta == 3)
   {

    if( (write(sd, "Y", 1)) < 0 )
    {
      perror("Error write request_receive_objects_destroyed ");
      //exit(-1);
    }

    receiveObjectDestroyed(sd,team_char);
    Pulisci();
  }
    else
     if( (write(sd, "N", 1)) < 0 )
     {
       perror("Error write request_receive_objects_destroyed");
       //exit(-1);
     }
}

void addPlayer(int X,int Y,char nome,char* username,int team,int
punteggio,int indice,int i)      //da aggiungere al progetto
```

```
{
         P[i].X=X;
         P[i].Y=Y;
         P[i].nome=nome;
         strcpy(P[i].username,username);
         P[i].team=team;
         P[i].punteggio=punteggio;
        // n_players++;
         P[i].indice_player=indice;
}


void StampaPunteggio(int io)       //da aggiungere al progetto
{
    printf("PUNTEGGIO : [%d]\n",P[io].punteggio);
}

int find_me(int n_players, int indice)
{
  int i;

   for(i=0;i<n_players;i++)
      if(P[i].indice_player == indice)
        return i;

//return 0;
}


void stampaPlayers(char** M, int n_players, int io)                    //da
aggiungere al progetto
{
      int i;
        system("@cls||clear");
          stampaMappa(M,n_players,io);
            printf("\n\tUTENTI COLLEGATI\n\n");

        for(i=0;i<n_players;i++)
        {
            if(i==io)
            {
                if(P[i].team=='V')
                {
                     printf("P%d* : username=" GREEN "%s
",i+1,P[i].username);
                     printf(COLOR_RESET "team=" GREEN "%c   ",P[i].team);
                }
                else
                {
                      printf("P%d* : username=" RED "%s
",i+1,P[i].username);
                      printf(COLOR_RESET "team=" RED "%c   ",P[i].team);
                }
            }
            else
            {
                if(P[i].team=='V')
                {
                     printf("P%d  : username=" GREEN "%s
",i+1,P[i].username);
```

```
                    printf(COLOR_RESET "team=" GREEN "%c   ",P[i].team);
                }
                else
                {
                     printf("P%d  : username=" RED "%s
",i+1,P[i].username);
                     printf(COLOR_RESET "team=" RED "%c   ",P[i].team);
                }

            }
              printf(COLOR_RESET "\n");
        }
        printf(COLOR_RESET "\n");
}

void stampaCompagni(char** M,char team,int n_players, int io)         //da
aggiungere al progetto
{
    int i;
     system("@cls||clear");
       for(i=0;i<n_players;i++)
       {
           if(P[i].team==team)
               M[P[i].X][P[i].Y]=P[i].nome;
       }

       stampaMappa(M,n_players, io);

       for(i=0;i<n_players;i++)
       {
           if(P[i].team==team)
               M[P[i].X][P[i].Y]=' ';
       }


       printf("\n\tCOMPAGNI DI SQUADRA\n\n");

     for(i=0;i<n_players;i++)
     {
            if(P[i].team==team)
            {
                if(team=='V')
                {
                   printf("nome=" GREEN "%c   ",P[i].nome);
                    printf(COLOR_RESET "Posizione(%d,",P[i].X);
                   printf("%d)   ",P[i].Y);
                   printf("punteggio=%d",P[i].punteggio);
                }
                else
                {
                   printf("nome=" RED "%c   ",P[i].nome);
                   printf(COLOR_RESET "Posizione(%d,",P[i].X);
                   printf("%d)   ",P[i].Y);
                   printf("punteggio=%d",P[i].punteggio);
                }

                printf("\n");
            }
     }
```

```
}

void Pulisci()                  //da aggiungere al progetto
{
   printf("\n\nPremi Enter per continuare... ");
     char prev=0;

        while(1)
        {
            char c = getchar();

            if(c == '\n' && prev == c)
            {

                system("@cls||clear");
                break;
            }

            prev = c;
        }
}


int teamSelection(int sd)
{
  char squadra;
  int team;

  printf("\nScegli il team di appartenenza (0=verde,1=rosso)\n");
       while( scanf("%d",&team)!=1||team<0||team>1)
       {
                            printf("Team non valido : ");
                            while (getchar() != '\n');
       }

       if(team == 0) squadra = 'V' ;
       else squadra = 'R';

       if( (write(sd,&squadra, 1)) < 0 )
       {
         perror("Error Write Team to Server");
         exit(-1);
       }

  return team;
}

char requestName(int sd)
{
  char nome;

    if( (read(sd,&nome,1)) < 0 )
    {
      perror("Error request Name");
      exit(-1);
    }

    //fprintf(stdout, "letto %c", nome);
```

```
  return nome;
}


int requestToSpawn(int * x, int * y, int team,int sd, char **Mappa)
{

do{

  printf("\nScrivi la tua posizione (riga,colonna)\nriga(1-28): ");

     while( scanf("%d",x)!=1||*x<1||*x>28)
     {
            printf("Coordinata riga non valida, Reinserisci : ");
            while (getchar() != '\n');
     }

     if(team==0)
     {

          printf("colonna(1-39) : ");
          while( scanf("%d",y)!=1||*y<1||*y>39)
          {
                    printf("Coordinata colonna non valida, Reinserisci : ");
                    while (getchar() != '\n');
          }



     }
     else
     {

          printf("colonna(40-78) : ");
          while( scanf("%d",y)!=1||*y<1||*y<40||*y>78)
          {
                    printf("Coordinata colonna non valida, Reinserisci : ");
                    while (getchar() != '\n');
          }
     }

   } while(Mappa[*x][*y]=='#' && fprintf(stdout, "\nSpawn Non Valido su una
Barriera"));

    int conv_x=htonl(*x);
    int conv_y=htonl(*y);
    char response;




     if( (send(sd, (const char*)&conv_x, 4, 0)) < 0 )
     {
       perror("Error Write Coordinata_X");
       exit(-1);
     }

     if( (send(sd, (const char*)&conv_y, 4, 0)) < 0 )
     {
       perror("Error Write Coordinata_Y");
```

```
       exit(-1);
     }


     if( (read(sd,&response,1)) < 0 )
     {
       perror("Error Read Response from server");
       exit(-1);
     }


     if(response == 'V') return 1;
     else return 0;

}

void riepilogoDati(char nome, int team, int X, int Y)
{
  system("clear");
  Logo();
  fprintf(stdout,             "\n+--------------------Riepilogo
Dati-------------------+");
    if(team==0)
      fprintf(stdout,"\n| NOME=["GREEN"%c"COLOR_RESET"] |
TEAM=["GREEN"VERDI"COLOR_RESET"] | Cordinate di Spawn=[%d,%d] |",nome,X,Y);
    else
    fprintf(stdout,"\n| NOME=["RED"%c"COLOR_RESET"] |
TEAM=["RED"ROSSI"COLOR_RESET"] | Cordinate di Spawn=[%d,%d] |",nome,X,Y);
  fprintf(stdout, COLOR_RESET
"\n+-----------------------------------------------------+");

  getchar();
  fprintf(stdout, "\nPremi un tasto per continuare...");
  while( getchar() != '\n' );

}

char** allocaMappa()
{
    int i;
    char **M=malloc(SIZE_RIGHE*sizeof(char *));

    for(i=0;i<SIZE_RIGHE;i++)
      M[i]=malloc(SIZE_COLONNE*sizeof(char));

    return M;
}

char** GeneraMappa()
{
    int i,j;
    char **M=allocaMappa();

    for(i=0;i<SIZE_RIGHE;i++)
    {
      for(j=0;j<SIZE_COLONNE;j++)
      {
        if(i == 0) //riga in alto
          M[i][j]='#';
```

```
        else if(j==0 && i!=0) // riga a sx
          M[i][j]='#';

        else if(j==(SIZE_COLONNE-1) && i != 0) // riga a dx
          M[i][j]='#';

        else if(i==(SIZE_RIGHE-1)) // riga sotto
         M[i][j]='#';

        else if ((i==11 || i==12 || i==13 || i == 14 || i==15 || i == 16 ||
i == 17 || i == 18 || i == 19) && j == 10)
          M[i][j]='#';

        else if ((i==11 && j == 9) || (i == 19 && j == 9) || (i == 11 && j
== 8 ) || (i == 19 && j == 8 ))
          M[i][j]='#';

        else if (((i==11 || i==12 || i==13 || i == 14 || i==15 || i == 16 ||
i == 17 || i == 18 || i == 19) && j == 69))
          M[i][j]='#';

        else if ((i==11 && j == 70) || (i == 19 && j == 70) || (i == 11 && j
== 71 ) || (i == 19 && j == 71 ))
          M[i][j]='#';
        else if(M[i][j]!=ARMA && M[i][j]!=MINA && M[i][j]!=OSTACOLO)
           M[i][j]=' ';

      }
    }


     //generateFlags(M);                  //Tutte le cose dovranno essere
nascoste finchè sarà un giocatore ad andarci sopra e a scoprirle


    return M;
}


void stampaMappa(char **M, int n_players, int io)
{
  int i,j,c=0,r=0,k;
  char team;

   for(i=0;i<n_players;i++)
     M[P[i].X][P[i].Y]=P[i].nome;



printf("\n ");
    StampaPunteggio(io);                                          //da
aggiungere al progetto

printf("\n ");


       for(i=0;i<80;i++)
       {
              printf("%d",c);
              if(c==9)
```

```
                    c=-1;
               c++;
      }


         printf("\n");
 for(i=0;i<SIZE_RIGHE;i++)
 {
      printf(COLOR_RESET "%d",r);
      if(r==9)
         r=-1;
      r++;


     for(j=0;j<SIZE_COLONNE;j++)
     {
          if(j<40 && (M[i][j]=='#'  || M[i][j]==FLAG ))
          {
               printf(GREEN   "%c",M[i][j]);
          }
          else if(j>=40 && (M[i][j]=='#'  || M[i][j]==FLAG))
          {
                printf(RED    "%c",M[i][j]);
          }
          else if( (M[i][j]>='A'&&M[i][j]<='Z')  || (M[i][j]>='a'&&M[i]
[j]<='z'))
          {
               if(P!=NULL)
               {
                          for(k=0;k<n_players;k++)
                          {

                              if(M[i][j]==P[k].nome && i==P[k].X &&
j==P[k].Y)
                              {
                                   team=P[k].team;
                                   break;
                              }
                          }
               }

              if(team=='V')
                  printf(GREEN   "%c",M[i][j]);
              else
                 printf(RED   "%c",M[i][j]);
          }
          else
             printf(COLOR_RESET "%c",M[i][j]);
     }
   printf("\n");
 }

  printf(COLOR_RESET "\n");
}




void clearBuffer(){
```

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

int ReceiveData(char **Mappa, int *TOT_OBJECTS,int sd)
{
  int TOT_OBJECTS_temp=0;

      if( (recv(sd, &TOT_OBJECTS_temp, 4, 0)) < 0 )
      {
        perror("Error Read Coordinata_X");
        exit(-1);
      }

      *TOT_OBJECTS = ntohl(TOT_OBJECTS_temp);

      //fprintf(stdout, "\nHo ricevuto %d", *TOT_OBJECTS);

      int i,j,k;
      char temp[4];

      for(i=0;i<*TOT_OBJECTS;i++)
      {
        for(j=0;j<4;j++)
            if( (read(sd, &temp[j], 1)) < 0 )
            {
              perror("Error Read carattere Matrice");
              exit(-1);
            }

            char X_c=temp[1];
            int X_i=X_c-'0';
            char Y_c=temp[2];
            int Y_i=Y_c-'0';

          //  printf("\n %d %d", X_i , Y_i);

            if(temp[3] == 'V')
             Mappa[X_i][Y_i]=temp[0];
            else
              Mappa[X_i][Y_i]=' ';
      }

}

void RequestMovementClient(int sd, char spostamento , char username[])
{
    //fprintf(stdout , "\nInvio Richiesta di Spostamento [%s %c] al
Server\n", username , spostamento);

      if( (write(sd,username,strlen(username))) < 0 )
      {
        perror("Error Write Username - RequestMovementClient() ");
        //exit(-1);
      }

      if( (write(sd,&spostamento,1) < 0 ))
      {
        perror("Error Write spostamento - RequestMovementClient() ");
      //  exit(-1);
```

```
      }

}

void receiveObjectDestroyed(int sd, char team)
{
    int i,n,X,Y, n_letture, nbytes;
    char username[20],temp[4], n_letture_c;

    if( (write(sd,&team,1)) < 0 )
    {
      perror("Error write team in receiveObjectDestroyed");
      //exit(-1);
    }

    if( (read(sd,&n_letture_c,1)) < 0 )
    {
      perror("Error receive n_letture receiveObjectDestroyed");
    //  exit(-1);
    }

    n_letture=n_letture_c - '0';

    for(i=0; i<n_letture; i++) // DA AGGIUSTARE! DEVO SAPERE IL NUMERO DI
ITERAZIONI DA FARE
    {
        if( (nbytes=read(sd,username,20)) < 0 )
        {
          perror("Error receive username receiveObjectDestroyed");
        //  exit(-1);
        }
          username[nbytes]='0';

        if( (read(sd,temp,4)) < 0 )
        {
          perror("Error temp read receiveObjectDestroyed");
          //exit(-1);
        }

        X=temp[1] - '0';
        Y=temp[2] - '0';

        StampaObjects(username,temp[3],temp[0],X,Y);

    }

}

void StampaObjects(char username[],char nome_p,char nome_o,int X,int Y)
{
        printf("USERNAME=%s NOME=%c : ",username,nome_p);
        printf("[%c] -> (%d,%d)\n",nome_o,X,Y);
}

void receivePlayers(int sd , char **Mappa, int *n_players)
{
        int punteggio, X , Y , i,indice, old_X , old_Y;
        char username[20],punteggio_string[10];
        char n_players_c, punteggio_c , X_c , Y_c , nome , team;
        char temp[5];
```

```
        if( (read(sd, &n_players_c, 1)) < 0 )
        {
          perror("Error Read n_players in receivePlayers");
          exit(-1);
        }

        *n_players=n_players_c - '0';

         fprintf(stdout, "\nRicevuto n_players = %d\n", *n_players);

         for(i=0;i<*n_players; i++)
         {

           if( (read(sd,temp,5)) < 0 )
           {
             perror("Error Read team in receivePlayers");
             exit(-1);
           }

        //  puts("ATTENDO L'USERNAME");
           if( (read(sd,username,20)) < 0 )
           {
             perror("Error Read username in receivePlayers");
             exit(-1);
           }
          // puts("USERNAME RICEVUTO");

          if( (read(sd,punteggio_string,10)) < 0 )
          {
            perror("Error Read punteggio_string in receivePlayers");
            exit(-1);
          }


            X=temp[2] - '0';
            Y=temp[3]  - '0';
            indice=temp[4] - '0';
            punteggio=atoi(punteggio_string);

          //  puts("Prima addPlayer");
        fprintf(stdout, "\nRicevuto username=[%s] nome=[%c] | squadra=[%c] |
Punteggio=[%d] | X=[%d] Y=[%d] | indice =[%d]",username, temp[0] ,
temp[1] ,punteggio, X , Y, indice);

        old_X=P[i].X;
        old_Y=P[i].Y;
        Mappa[old_X][old_Y]=' ';

        fprintf(stdout, "\nP[%d] old_X=%d , old_Y=%d",i,old_X,old_Y);

        addPlayer(X,Y,temp[0],username,temp[1],punteggio,indice,i);

        }
      //   puts("Esco dal For");

}

int receive_fine_game(int sd)
{
```

```
    char temp=' ';

    if( (read(sd,&temp,1)) < 0 )
    {
      //perror("Error read receive_fine_game");
      fprintf(stdout, "PAREGGIO");
    }

    fprintf(stdout, "\nRicevuto fine_temp=%c",temp);

    if(temp == 'F')
    {
      fprintf(stdout, "\n\nHanno vinto i VERDI!\nFINE PARTITA.\n");
      return 1;
    }
    else if (temp == 'B')
    {
      fprintf(stdout, "\n\nHanno vinto i ROSSI!\nFINE PARTITA.\n");
      return 1;
    }
    else if (temp == 'P')
    {
      fprintf(stdout, "\n\nPareggio\nFINE PARTITA.\n");
      return 1;
    }

  return 0;
}
```