UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Relazione del progetto “ricerca con eliminazione”
traccia C
Laboratorio di Sistemi Operativi
Prof. Alberto Finzi
studente: Riccardo Iervolino N860001608

Descrizione Sintetica

Il progetto si articola in un sistema client-server e consente a più utenti di giocare a una ricerca del tesoro nascosto con possibilità di eliminare i propri avversari. E’ stato utilizzato il linguaggio C su piattaforma UNIX. I processi comunicano tramite socket TCP ed è dunque possibile usare client su dispositivi differenti connettendoli alla stessa partita in atto sul server.

Descrizione Dettagliata

Il server mantiene una rappresentazione dell’ambiente in cui oltre agli utenti vengono posizionati ostacoli, oggetti, ed un tesoro.
L’ambiente è rappresentato da una matrice in cui gli utenti si potranno spostare di un passo alla volta nelle quattro direzioni: S, N, E, O, a meno di non essersi scontrati con un ostacolo o un altro utente.
Il server posizionerà nella matrice in modo random gli ostacoli, il tesoro e le armi. Ogni utente, una volta connesso al server, potrà partecipare

alla ricerca, il server comunicherà all’utente il punto di partenza per la ricerca come coordinata (x,y).
Dopo ogni passo l’utente riceverà l’informazione sull’effetto proprio movimento: se lo spostamento porta ad una collisione con un ostacolo oppure con un altro utente, il movimento avrà effetto nullo; se lo spostamento porta nella locazione di un’arma, l’utente potrà raccoglierla. Una volta in possesso di arma, dopo ogni spostamento l’utente potrà vedere ostacoli o altri utenti a distanza di 4 passi nella direzione dello
spostamento, a questo punto l’arma potrà essere usata, una sola volta, per eliminare l’utente visibile più vicino.
La presenza del tesoro verrà notificata al primo utente che raggiungerà la locazione. Quando il tesoro sarà trovato da un utente, il server notificherà agli utenti la fine della sessione e ne genererà una nuova.
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, raccogliere un’arma, usare un’arma, vedere la lista degli utenti in gioco ed eliminati, vedere la posizione di tutti gli ostacoli trovati.
Il server supporta tutte le funzionalità descritte nella sezione relativa al client.
All'avvio del server, è possibile specificare tramite riga di comando la porta TCP sulla quale mettersi in ascolto.
Il server è di tipo concorrente, ovvero e in grado di servire più client simultaneamente.
Ad ogni nuovo client che prova a connettersi il server crea un nuovo processo e connette la sua socket alla socket del client.

Guida d’uso

Per compilare il file server da shell, una volta giunti nella cartella che lo contiene, digitare il comando

gcc -pthread -o LS Lserver.c

Per eseguire il server da shell, una volta giunti nella cartella che lo contiene, digitare il comando

./LS numport

dove numport rappresenta il numero della porta su cui il server deve mettersi in ascolto.

Per compilare il file client da shell, una volta giunti nella cartella che lo contiene, digitare il comando

gcc -o LC Lclient.c

Per eseguire il server da shell, una volta giunti nella cartella che lo contiene, digitare il comando

./LC host numport

dove host rappresenta l'indirizzo che ospita il server (localhost se client e server girano sulla stessa macchina)
e numport rappresenta il numero della porta su cui il server si è messo in ascolto.

How to play

una volta eseguito correttamente il server vi si possono eseguire tutti i client che i desidera.
Una volta eseguito correttamente il client (avviene la connessione a livello TCP) l'utente potrà leggere una breve lista dei comandi eseguibili una volta in gioco, tuttavia è chiamato ad identificarsi, inserendo 1 nel caso in cui sia già registrato o 2 se deve ancora farlo.
l'user inserisce id e password e il server controlla che l'utente esista, che non stia già giocando, e che la password sia corretta nel primo caso; nel secondo invece l'utente inserisce id e password desiderata e il server controlla che l'id sia disponibile e crea un nuovo utente.
Subito dopo questa fase inizia il gioco vero e proprio, dove l'utente avrà a propria disposizione i seguenti comandi:
w per muoversi verso l'alto
a per muoversi a sinistra
s per muoversi verso il basso
d per muoversi verso destra
q per sparare (qualora avesse raccolto un'arma)
e per disconnettersi
c per mostrare la classifica (ovvero tutti i giocatori che hanno preso parte al gioco e il numero di uccisioni da loro totalizzato)

Una volta trovato il tesoro il server disconnette tutti i client, genera una nuova mappa, azzera la classifica e accoglie i nuovi utenti.

Protocolli di comunicazione client-server

Il client aspetta che l'utente dia un input, lo analizza e in base al tipo di input entra in un blocco di codice dedicato, passa sempre e comunque l'input al server e si mette in ascolto con una o più read sulla socket, in base al blocco in cui si trova (e quindi all'input ricevuto).

Ogni processo del server intanto aspetta costantemente dalla socket l'input dell'utente del client a cui il processo è legato, appena lo riceve lo analizza ed entra in un blocco condizionale dedicato, identicamente al client. I due blocchi in cui si trova l'esecuzione del gioco sono infatti gli analoghi su client e server e comunicano tra loro tramite socket, aspettando ed inviando un esatto numero di caratteri che servano a svolgere correttamente la funzione richiesta dall'utente.

La sincronizzazione tra tutti i client (e quindi anche tra tutti i processi del server) è garantita dalla presenza di un file (MAP.txt) sul quale vengono costantemente aggiornate le variazioni dell'ambiente in base alle regole del gioco.

multithreading del server

Il server esegue una funzione game come start del thread creato con pthread_create, a cui viene passato il parametro newsocket accettato dal main appena un client si connette, infatti ancora prima dell'esecuzione del thread ha messo newsockfd ad accettare i client di sockfd con l'istruzione newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
esegue quindi il sottoprocesso che serve il client e torna ad ascoltare su sockfd (questi rientrerà con l'arrivo di un nuovo client nel while che contiene accept e pthread_create), e il nuovo thread sarà il processo che serve il client appena accettato.

Numero di giocatori

Il server tiene traccia del numero di giocatori, che li rappresenterà sulla mappa così da renderli unici ed evitare fraintendimenti nella lettura della matrice map, nel caso in cui ad esempio un giocatore dovesse spostarsi sulla casella di un

giocatore ucciso prima che questi abbia il tempo di muoversi e quindi di venire a conoscenza della propria eliminazione.

Mutex

la variabile mymutex consente di regolamentare i passaggi critici in cui viene modificata la variabile globale map. Si rivela inutie nella funzione crea_mappa in quanto essa viene eseguita solo dal processo principale, rendendo superflua la regolamentazione dell'accesso alla risorsa globale.

Stampa della classifica

La funzione mostraclassifica ha lo scopo di inviare al client informazioni riguardo i giocatori che si sono connessi alla partita e il numero di uccisioni che hanno totalizzato, per avvisare il client che la lettura del file è terminata viene inviato un carattere speciale '#'

```
void mostraclassifica(int sockfd){
        int fd;
        char c; int n;
        if((fd=open("CLASSIFICA.txt", O_RDONLY , 0))<0)
                error("open");
        while((n=read(fd, &c, sizeof(char)))>0)  {
                if((n=write(sockfd, &c, sizeof(char)))<0)
                        error("write");
        }
        c='#';
        if((n=write(sockfd, &c, sizeof(char)))<0)
                 error("write");
        close(fd);
}
```

Uccisioni

Quando un giocatore viene ucciso riceve questa informazione al primo tentativo di compiere un'azione, infatti il processo che lo serve troverà nella posizione che il client occupata una casella vuota, e restituirà l'istruzione di disconnettersi.
Quando invece un utente spara e colpisce ottiene subito l'informazione della sua scomparsa e la sua classifica viene subito aggiornata con la funzione aggiornaclassifica.

Questa funzione viene eseguita quando il processo del server dedicato riceve dal client l'istruzione q, ovvero di sparare, e nella direzione del suo ultimo spostamento il client aveva incrociato un nemico nelle quattro caselle che l'arma consente di visualizzare

```
if(map[X][Y]=='U') {
                                        write(newsockfd, "M", sizeof(char));
                                        aggiornaclassifica(name);
                                        stampamappa(map, X, Y, 0);
}
```

infatti 'U' si trova nelle coordinate X Y, ricavate tramite un controllo dell'ultima direzione percorsa (ricevuta anch'essa dal client che ne conserva memoria se armato).
Il client riceve l'informazione M, che rappresenta l'uccisione, la classifica viene aggiornata (viene aggiunto 1 al numero di uccisioni dell'utente con id == name, infatti name è lo stesso con cui l'utente si era inizialmente connesso e la variabile resta immutata trovandosi nel medesimo processo). Infine viene aggiornata anche la mappa, eliminando l'utente in posizione (X,Y) e sostituendovi uno spazio vuoto.

Spostamenti

il client ed il server comunicano per singoli caratteri e quando il client è chiamato a fornire un'istruzione da parte dell'utente il carattere corrisponde al carattere immesso da tastiera. Se dovesse essere una direzione il programma decide quale esso sia con la seguente riga
`if(ch=='w') tx=tx-1; else if (ch=='s') tx=tx+1; else if (ch=='a') ty=ty-1; else if (ch=='d') ty=ty+1;`
e si sposta sulla propria mappa in base alla risposta del server, che controlla la direzione con gli stessi if e successivamente verifica il contenuto della casella raggiunta.

Ciclo client-server

E' possibile identificare un ciclo delle comunicazioni tra il client ed il processo del server che lo serve.
Tale ciclo inizia con l'acquisizione dell'input da parte dell'utente e l'attesa di tale input da parte del processo server, e continua con l'analisi di questo carattere da parte del server e l'attesa da parte del client. Il server in questa fase

esegue l’istruzione ricevuta e in base al risultato restituisce un carattere speciale, quindi torna all’inizio del proprio ciclo. Il client invece per finire analizza l’ultimo input dal server, e in base anche all’input iniziale dell’utente giunge ad una conclusione che determinerà l’uscita dal ciclo o il suo continuare e un conseguente output su schermo.

Sorgente

SERVER

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <pthread.h>
int player=0;
char map[10][21];
pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;
void *game(void *);
void aggiornaclassifica(char name[50]);
void cancellagiocatori(char (*map)[21]);
void print(char (*map)[21]);
int tesoro();
void error(const char *msg) {
    perror(msg);
    exit(1);
}
int Search_in_File(char *fname, char *str);
void crea_mappa (char (* map)[21] );
void mostraclassifica(int sockfd);
void classifica();
int main(int argc, char *argv[]) {

    int sockfd, newsockfd, portno;
    socklen_t clilen;

    pthread_t tid;
```

```
    struct sockaddr_in serv_addr, cli_addr;

    if (argc < 2) {
            fprintf(stderr, "ERROR, no port provided\n");
            exit(1);
    }
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
            error("ERROR opening socket");
    bzero((char *) &serv_addr, sizeof(serv_addr));
    portno = atoi(argv[1]);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);
    if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
           error("ERROR on binding");
    listen(sockfd, 1);
    clilen = sizeof(cli_addr);


    while(1){
              remove("CLASSIFICA.txt");
              srand(time(NULL));
              crea_mappa(map);
              while (tesoro()==1) {

                    newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
                    if (newsockfd < 0)
                            error("ERROR on accept");
                    if( pthread_create( &tid , NULL ,  game , (void*)(intptr_t) newsockfd) < 0)
                    {
                      perror("could not create thread");
                    }
                    listen(sockfd, 1);

              }
    }

  return 0;
}

void *game(void *sockfd){
          int conn, find, t, i, X, Y, ind, count, esc;    int n; char myid;
          char ch;char buf[10];
          int start, x,y, armed, sel;
          char buffer[256], job[20], name[50];
          int newsockfd=(int)(intptr_t)sockfd;
          esc=0;
          conn=0;
          int T=1;FILE * fp;
          while( conn == 0) {
```

```
sel=0;
bzero(buffer, 256);
bzero(name, 50);
n = read(newsockfd, buffer, 255);
if (n < 0)
	error("ERROR reading from socket");

strncpy(job, buffer, 5);

fp = fopen("labirinto.txt", "ab+");
if(fp == NULL)
	error("open");


if(job[0]=='k'){
	strcpy(name, buffer+5);
	find=Search_in_File("labirinto.txt", name);
	if(find==1){
		for ( i=5; i<230; i++) if(buffer[i]==' ') t=i;
		bzero(name, 50);
		strncpy(name, buffer+5, t-5);//printf("\n\nt=%d\n\n%s",t, name);
		find=Search_in_File("CLASSIFICA.txt", name);
		if(find!=1){
			conn=1;
		}
		else sel=1;
	}
}
else if(job[0]=='u'){
	for ( i=5; i<230; i++) if(buffer[i]==' ') t=i;
	strncpy(name, buffer+5, t-5);//printf("\n\nt=%d\n\n%s",t, name);
	find=Search_in_File("labirinto.txt", name);

	if(find==0){
		conn=1;
		strcpy(name, buffer+5);
		n = fwrite(name, strlen(name), sizeof(char), fp);
		fclose(fp);
		if (n < 0)
			error("ERROR writing to file");
	}
}else;
if(conn==1){
	n = write(newsockfd, "y", 1);
	if (n < 0)
		error("ERROR writing to socket");
}else if(sel==1){
	n = write(newsockfd, "j", 1);
	if (n < 0)
		error("ERROR writing to socket");
 }
 else{
```

```
		n = write(newsockfd, "n", 1);
		if (n < 0)
			error("ERROR writing to socket");
	   }
	}
	for ( i=5; i<230; i++) if(buffer[i]==' ') t=i;
	bzero(name, 50);
	strncpy(name, buffer+5, t-5);
	strcat(name, " 0\n");
	classifica(name);
	bzero(name, 50);
	strncpy(name, buffer+5, t-5);
	start=0;
	while(start==0) { x=rand()%10; y=rand()%20;
		if((x==0||y==0||x==9||y==19)&&map[x][y]==' '&&(x!=0||y!=0)) {
			sprintf(buf, "%d", x);
			pthread_mutex_lock(&mymutex);
			map[x][y]=player+'0'; start=1;
			myid=player+'0';
			player++;
			pthread_mutex_unlock(&mymutex);
			n=write(newsockfd, buf, sizeof(buf));
			sprintf(buf, "%d", y);

			n=write(newsockfd, buf, sizeof(buf));
		}

	}

	armed=0;

	while(esc==0){

		ch=' ';
		read(newsockfd, &ch, 1);
		if(map[x][y]!=myid&&map[x][y]!='L')	{ write(newsockfd, "K", 1); esc=1;}
		else if(map[x][y]=='L') {write(newsockfd, "L", 1); esc=2;}
		else if(ch=='c') {
			mostraclassifica(newsockfd);
			if(map[x][y]==' ')	{ write(newsockfd, "K", 1); esc=1;}
			else if(map[x][y]=='L') {write(newsockfd, "L", 1); esc=2;}
			else write(newsockfd, "!", 1);
		}
		else if(ch=='e') esc=1;
		else if(ch=='q') {
			armed=0;
			read(newsockfd, &ch, 1);
			X=x; Y=y;
			count=0;
			do{
				if(ch=='w') X=X-1; else if (ch=='s') X=X+1; else if (ch=='a') Y=Y-1;
else if (ch=='d') Y=Y+1;
```

```
                        count++;
                }while(count<4&&(map[X][Y]<'0'||map[X][Y]>'9'));
                if(map[X][Y]>='0'&&map[X][Y]<='9') {
                        pthread_mutex_lock(&mymutex);
                        map[X][Y]=' ';
                        pthread_mutex_unlock(&mymutex);
                        write(newsockfd, "M", 1);
                        aggiornaclassifica(name);
                }else write(newsockfd, "!", 1);
                if(map[x][y]==' ')      { write(newsockfd, "K", 1); esc=1;}
                else if(map[x][y]=='L') {write(newsockfd, "L", 1); esc=2;}
                else write(newsockfd, "!", 1);
        }
        else{
                X=x; Y=y;
                count=0;
                ind=1;
                if(armed==1) ind=4;
                while(count<ind){
                        if(ch=='w') X=X-1; else if (ch=='s') X=X+1; else if (ch=='a') Y=Y-1;
else if (ch=='d') Y=Y+1;
                        if (X>=0&&X<10&&Y>=0&&Y<20) {
                                write(newsockfd, &map[X][Y], 1);
                                if((map[X][Y]==' '||map[X][Y]=='A')&&count==0) {

                                        if(map[X][Y]=='A') armed=1;
                                        pthread_mutex_lock(&mymutex);
                                        map[X][Y]=map[x][y];
                                        map[x][y]=' ';
                                        pthread_mutex_unlock(&mymutex);
                                        x=X;
                                        y=Y;

                                }
                                else if(map[X][Y]=='T'&&count==0) {
                                        esc=2;
                                        cancellagiocatori(map);
                                }
                        }
                        else write(newsockfd, "!", 1);
                        count++;
                }
        }
   }
   close(newsockfd);
}
void crea_mappa (char (*map) [21]) {
        srand(time(NULL));
        int i, j;
        int T = 0;
        for(i=0; i<21; i++)
                for (j=0; j<10; j++)
```

```
                        map[j][i]=' ';
        for(i=0; i<21; i++) {
                for (j=0; j<10; j++) {

                        if(i==20) map[j][i]='\n'; else {
                                if (i>0 && j>0 && i<19 && j<9) {
                                        if(((map[j-1][i-1]!='O'&&map[j-1][i]!='O')&&(map[j][i-1]!
='O'))||rand()%20==0){
                                                if(i>4&&j>2&&T==0&&rand()%15==0) { map[j]
[i]='T'; T=1;}
                                                else map[j][i]='O';
                                        }
                                        else if (rand()%5==0) map[j][i]='A';
                                        else map[j][i]=' ';
                                }else map[j][i]=' ';
                        }
        }
   }
        if(T==0) map[(rand()%9)+1][(rand()%19)+1]='T';

}
int Search_in_File(char *fname, char *str) {
        FILE *fp;
        size_t len = 0;
        char *temp;
        ssize_t read;

        if((fp = fopen(fname, "r")) == NULL) {
                return(0);
        }

        while ((read = getline(&temp, &len, fp)) != -1) {

                        if(strstr(temp, str)){

                                        return(1);


                }
        }




        if(fp) {
                fclose(fp);
        }

        return(0);
}
```

```
void cancellagiocatori(char (*map)[21]){
	for (int i=0; i<10; i++){
		for (int j=0; j<21; j++){
			if(map[i][j]>='0'&&map[i][j]<='9') {
				pthread_mutex_lock(&mymutex);
				map[i][j]='L';
				pthread_mutex_unlock(&mymutex);

			}
			else if(map[i][j]='T') {
				pthread_mutex_lock(&mymutex);
				map[i][j]=' ';
				pthread_mutex_unlock(&mymutex);
			}
		}
	}
}


void print(char (*map)[21]){
	for (int i=0; i<10; i++)
		for (int j=0; j<21; j++)
			write(STDOUT_FILENO, &map[i][j], 1);
}

int tesoro(){
	int ret;
	ret=0;

	for (int i=0; i<10; i++){
		for (int j=0; j<21; j++){
			if(map[i][j]=='T') ret=1;
		}
	}
	return ret;
}

void classifica(char name[50]){
	FILE * fd;

	fd = fopen("CLASSIFICA.txt", "ab+");
	fprintf(fd, "%s", name);
	fclose(fd);
}

void mostraclassifica(int sockfd){
	int fd;
	char c; int n;
	fd=open("CLASSIFICA.txt", O_RDONLY , 0);
	while((n=read(fd, &c, 1))>0)  { write(sockfd, &c, 1);}
```

```
	c='#';write(sockfd, &c, 1);
	close(fd);
}

void aggiornaclassifica(char name[50]){
	FILE *fp;
	int i,j, tmp, no,z;
	size_t len = 0;
	char temp[50][50];
	char * a;
	int count=0;
	int fd, readx;
	char x;
	fp = fopen("CLASSIFICA.txt", "r");
	i=0;
	while (getline(&a, &len, fp) != -1) {
			j=0;
			if(strstr(a, name)){

				tmp=a[strlen(a)-2]-'0';
				tmp++;
				a[strlen(a)-2]=tmp+'0';
			}
			strcpy(temp[i], a);
			i++;
	}

	fclose(fp);
	remove("CLASSIFICA.txt");
	fp=fopen("CLASSIFICA.txt", "w");
	while(i>=0){
		fprintf(fp, "%s", temp[i]);



		i--;
	}
	fclose(fp);
}
```

CLIENT

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <termios.h>

char getch();

void stampakill(int sockfd);
void stampamappa( char (*map)[21], int a);
void error(const char *msg)
{
    perror(msg);
    exit(0);
}

void connectusr(char buffer[256], char str[230], int sockfd);
void aggiornamappa(char buffer[256], int disp);
int main(int argc, char *argv[])
{
    int sockfd, portno, n, disp, end=0; int X, Y, tx, ty; int armed=0; int count, ind, aa, bb; int ack=0;
int shootx=-1, shooty=-1;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char str [230];
    char inp;
    char x,y;
    char buf[10];
    char ch, cc, sh;
    char buffer[256];
    char map[10][21];
    for(int i=0; i<21; i++) {
          for (int j=0; j<10; j++)

                              if(i==20) map[j][i]='\n'; else map[j][i]='?';

    }

    if (argc < 3) {
       fprintf(stderr,"usage %s hostname port\n", argv[0]);
       exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr,"ERROR, no such host\n");
        exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
```

```
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
         (char *)&serv_addr.sin_addr.s_addr,
         server->h_length);
    serv_addr.sin_port = htons(portno);
    //printf("h_addr: %s\n", inet_ntoa(serv_addr.sin_addr));
    if (connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
        error("ERROR connecting");
    write(STDOUT_FILENO, "how to play:\n\n\tw up\na left\t\td right\n\ts down\n\nq shoot (if
armed)\ne exit\nc show #kill\n\n", sizeof(char)*86);
    connectusr(buffer, str, sockfd);
    write(STDOUT_FILENO, "user connected.\n", sizeof(char)*17);
    sleep(1);
    system("clear");


    n = read(sockfd, buf, sizeof(buf));
    X=atoi(buf);
    n = read(sockfd, buf, sizeof(buf));
    Y=atoi(buf);
    map[X][Y]='U';
    armed=0;
    while(end == 0){
          stampamappa(map, ack);
          ack=0;
          ch=' ';
          while(ch!='w'&&ch!='a'&&ch!='s'&&ch!='d'&&ch!='q'&&ch!='e'&&ch!='c')ch=getch();
          tx=X; ty=Y;
          if(ch=='w') tx=tx-1; else if (ch=='s') tx=tx+1; else if (ch=='a') ty=ty-1; else if (ch=='d')
ty=ty+1;
          if(ch!='q'&&ch!='e'&&ch!='c') {
                sh=ch;
                n = write(sockfd, &ch, sizeof(char));
                count=0;
                aa=-1; bb=-1;
                ind=1;
                if(armed==1) ind=4;
                shootx=-1; shooty=-1;
                while (count<ind)
                {

                      n = read(sockfd, &cc, sizeof(char));

                      if (n < 0)
                            error("ERROR reading from socket");
                      if (cc=='K') end=4;
                      else if(cc=='L') end=2;
                      else if (count==0){


                            if (cc=='T') end=1;
                            else if (cc==' '||cc=='A') {
```

```
                                map[X][Y]=' ';
                                aa=tx; bb=ty;

                                if(cc=='A') {
                                        armed=1;
                                        ack=1;
                                }

                        }
                        else if (cc=='O') map[tx][ty]='O';
                        else if (cc>='0'&&cc<='9') {
                                map[tx][ty]='N';
                                shootx=tx; shooty=ty;
                        }
                        else if(cc!='!') map[tx][ty]=cc;


                }
                else{
                                if(cc>='0'&&cc<='9') {
                                        map[tx][ty]='N';
                                        if(shootx==-1){
                                                shootx=tx; shooty=ty;
                                        }
                                }
                                else if(cc=='A') map[tx][ty]=' ';
                                else if(cc!='!') map[tx][ty]=cc;

                }
                if(tx<X) tx--;else if (tx>X) tx++;else if(ty<Y) ty--; else if(ty>Y) ty++;
                count++;
                if(aa!=-1) {X=aa; Y=bb;}
                map[X][Y]='U';

        }
}
else if(ch=='q'&&armed==1)          {
        n = write(sockfd, &ch, sizeof(char));
        n = write(sockfd, &sh, sizeof(char));
        armed=0;
        if(shootx!=-1&&map[shootx][shooty]=='N') map[shootx][shooty]=' ';
        n = read(sockfd, &cc, sizeof(char));
        if(cc=='M') ack=2;
        //funzione per eliminare un nemico dalla mappa nelle 4 caselle puntate dal giocatore
        read(sockfd, &cc, sizeof(char));
        if(cc=='L') end=2;
        else if(cc=='K') end=4;
}
else if(ch=='e'){
        n = write(sockfd, &ch, sizeof(char));
        end=3;
```

```
        }
        else if(ch=='c'){
                n = write(sockfd, &ch, sizeof(char));
                stampakill(sockfd);
                read(sockfd, &cc, sizeof(char));
                if(cc=='L') end=2;
                else if(cc=='K') end=4;
        }
    }
    if(end==1) printf("congratulazioni! hai vinto!\n");
    else if(end==2) printf("game over! il tesoro è stato trovato!\n");
    else if(end==3) printf("disconnessione riuscita!\n");
    else if(end==4) printf("sei stato ucciso!\n");
    else printf("game over! sei stato ucciso!\n");
    close(sockfd);
    return 0;
}


void connectusr(char buffer[256], char str[230], int sockfd){
        int conn=0, ch=0, n=0, space;
        char c;
        while (conn==0){
                printf("Enter 1 to connect as a known user\nEnter 2 to create a new user\n");
                scanf("%d", &ch);
                printf("Please enter your username and password: ");
                bzero(buffer,256);
                bzero(str,256);
                scanf("%c", &c);
                fgets(str,230,stdin);
                space=0;
                for(int i=0; i<230; i++){
                        if(str[i]==' ') space++;
                }
                if (space!=1) ch=0;
                if(ch==1){
                   strcpy(buffer, "known");

                   strcat(buffer, str);
                   n = write(sockfd,buffer,strlen(buffer));
                   if (n < 0)
                         error("ERROR writing to socket");
                   bzero(buffer,256);
                   n = read(sockfd,buffer,1);
                   if (n < 0)
                         error("ERROR reading from socket");
                   if(buffer[0]=='y')  conn=1;
                   else if (buffer[0]=='n')printf("not found\n");
                   else printf("already playing\n");
                }
                else if(ch == 2){
```

```
                    strcpy(buffer, "unkno");

                    strcat(buffer, str);
                    n = write(sockfd,buffer,strlen(buffer));
                    if (n < 0)
                        error("ERROR writing to socket");
                    bzero(buffer,256);
                    n = read(sockfd,buffer, 1);
                    if (n < 0)
                        error("ERROR reading from socket");
                    if(buffer[0]=='y') {printf ("user created and connected\n"); conn=1;}
                    else if (buffer[0]=='n')printf("username already taken\n");
                }
                else printf("wrong input\n");
        }
}




void stampamappa( char (*map)[21], int a){
        system("clear");
        for (int i=0; i<10; i++){
                for (int j=0; j<21; j++){
                        write(STDOUT_FILENO, &map[i][j], sizeof(char));

                }
        }
        if(a==1)        write(STDOUT_FILENO, "\narma acquisita\n" , sizeof(char)*18);
        if(a==2)        write(STDOUT_FILENO, "\nnemico eliminato\n" , sizeof(char)*20);
}


char getch()
{
         int ch;
         struct termios oldt;
         struct termios newt;
         tcgetattr(STDIN_FILENO, &oldt);
         newt = oldt;
         newt.c_lflag &= ~(ICANON | ECHO);
         tcsetattr(STDIN_FILENO, TCSANOW, &newt);
         ch = getchar();
         tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
         return ch;
}

void stampakill(int sockfd){
        system("clear");
        char c;
        read(sockfd, &c, sizeof(char));
```

```
        while(c!='#'){
                write(STDOUT_FILENO, &c, sizeof(char));
                read(sockfd, &c, sizeof(char));

        }  sleep(1);
}
```