Università degli Studi di Napoli Federico II

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Corso di Laurea Triennale in Informatica

Classe n. L-31

Progetto di sistemi operativi

Traccia A

Professore:
Finzi Alberto

Candidati:
Turco Mario
Matr. N8600/2503
Longobardi Francesco
Matr. N8600/2468

Anno Accademico
2019/2020

Indice

1 Istruzioni preliminari

1.1 Modalità di compilazione

Il progetto è provvisto di un file makefile il quale è in grado di compilare autonomamente l'intero progetto. Per utilizzare il makefile aprire la cartella del progetto tramite la console di sistema e digitare "make".
In alternativa è possibile compilare manualemente il client ed il server con i seguenti comandi:

```
gcc -o server server.c boardUtility.c parser.c list.c -lpthread
gcc -o client client.c boardUtility.c parser.c list.c -lpthread
```

2 Guida all'uso

2.1 Server

Una volta compilato il progetto è possibile avviare il server digitando da console il seguente comando

```
./server users
```

L'identificativo *users* ri riferisce al nome del file sul quale sarà salvata la lista degli utenti e delle loro credenziali.
È possibile scegliere un nome a piacimento per il file purchè esso sia diverso da *log*.

2.2 Client

Una volta compilato il progetto è possibile avviare il cient digitando da console il seguente comando:

```
./client ip porta
```

Dove *ip* andrà sostituito con l'ip o l'indirizzo URL del server e *porta* andrà sostituito con la porta del server.
Una volta avviato il client comprarià il menu con le scelte 3 possibili: accedi, registrati ed esci.
Una volta effettuata la registrazione dell'utente è possibile effettuare l'accesso al programma al seguito del quale verranno mostrate sia la mappa dei gioco si le istruzioni di gioco.

3 Comunicazione tra client e server

Di seguito verranno illustrate le modalità di comunicazione tra client e server.

3.1 Configurazione del server

Il socket del server viene configurato con famiglia di protocoli PF_NET, con tipo di trasmissione dati SOCK_STREAM e con protocollo TCP. Mostriamo di seguito il codice sorgente:

Listato 1: Configurazione indirizzo del server

```
    }
    char clientAddr[20];
    strcpy(clientAddr, inet_ntoa(address.sin_addr));
    Args args = (Args)malloc(sizeof(struct argsToSend));
    args->userName = (char *)calloc(MAX_BUF, 1);
    strcpy(args->userName, clientAddr);
    args->flag = 2;
    pthread_t tid;
```

Listato 2: Configurazione socket del server

```
  inizializzaGrigliaVuota(grigliaDiGiocoConPacchiSenzaOstacoli);
  generaPosizioniRaccolta(grigliaDiGiocoConPacchiSenzaOstacoli,
                          grigliaOstacoliSenzaPacchi, deployCoords);
  riempiGrigliaConPacchiInPosizioniGenerateCasualmente(
      grigliaDiGiocoConPacchiSenzaOstacoli, packsCoords);
  generaPosizioneOstacoli(grigliaDiGiocoConPacchiSenzaOstacoli,
                          grigliaOstacoliSenzaPacchi);
  printf("Mappa generata\n");
  pthread_exit(NULL);
}
int almenoUnaMossaFatta()
{
  if (numMosse > 0)
    return 1;
  return 0;
}
```

È importante notare anche come il server riesca a gestire in modo concorrente più client tramite l'uso di un thread dedicato ad ogni client. Una volta aver configurato il socket, infatti, il server si mette in ascolto per nuove connessioni in entrata ed ogni volta che viene stabilita una nuova connessione viene avviato un thread per gestire tale connessione. Di seguito il relativo codice:

Listato 3: Procedura di ascolto del server

```
1   struct sockaddr_in mio_indirizzo = configuraIndirizzo();
2   configuraSocket(mio_indirizzo);
3   signal(SIGPIPE, clientCrashHandler);
4   signal(SIGINT, quitServer);
5   signal(SIGHUP, quitServer);
6   startTimer();
7   inizializzaGiocoSenzaPlayer(grigliaDiGiocoConPacchiSenzaOstacoli,
8                               grigliaOstacoliSenzaPacchi, packsCoords);
9   generaPosizioniRaccolta(grigliaDiGiocoConPacchiSenzaOstacoli,
10                          grigliaOstacoliSenzaPacchi, deployCoords);
11  startListening();
12  return 0;
13 }
14 void startListening()
15 {
16   pthread_t tid;
17   int clientDesc;
18   int *puntClientDesc;
19   while (1 == 1)
20   {
21     if (listen(socketDesc, 10) < 0)
22       perror("Impossibile mettersi in ascolto"), exit(-1);
23     printf("In ascolto..\n");
24     if ((clientDesc = accept(socketDesc, NULL, NULL)) < 0)
25     {
26       perror("Impossibile effettuare connessione\n");
27       exit(-1);
28     }
29     printf("Nuovo client connesso\n");
30     struct sockaddr_in address;
31     socklen_t size = sizeof(struct sockaddr_in);
32     if (getpeername(clientDesc, (struct sockaddr *)&address, &size) < 0)
33     {
34       perror("Impossibile ottenere l'indirizzo del client");
35       exit(-1);
```

In particolare al rigo 31 notiamo la creazione di un nuovo thread per gestire la connessione in entrata a cui passiamo il descrittore del client di cui si deve occupare. Dal rigo 16 al rigo 27, estraiamo invece l'indirizzo ip del client per scriverlo sul file di log.

3.2 Configurazione del client

Il cliente invece viene configurato e si connette al server tramite la seguente funzione:

Listato 4: Configurazione e connessione del client

```
1  int connettiAlServer(char **argv) {
2    char *indirizzoServer;
3    uint16_t porta = strtoul(argv[2], NULL, 10);
4    indirizzoServer = ipResolver(argv);
5    struct sockaddr_in mio_indirizzo;
6    mio_indirizzo.sin_family = AF_INET;
7    mio_indirizzo.sin_port = htons(porta);
8    inet_aton(indirizzoServer, &mio_indirizzo.sin_addr);
9    if ((socketDesc = socket(PF_INET, SOCK_STREAM, 0)) < 0)
10     perror("Impossibile creare socket"), exit(-1);
11   else
12     printf("Socket creato\n");
13   if (connect(socketDesc, (struct sockaddr *)&mio_indirizzo,
14               sizeof(mio_indirizzo)) < 0)
15     perror("Impossibile connettersi"), exit(-1);
16   else
17     printf("Connesso a %s\n", indirizzoServer);
18   return socketDesc;
19 }
```

Si noti come al rigo 9 viene configurato il socket ed al rigo 13 viene invece effettuato il tentativo di connessione al server.
Al rigo 3 invece viene convertita la porta inserita in input (argv[2]) dal tipo stringa al tipo della porta (uint16_t ovvero unsigned long integer).
Al rigo 4 notiamo invece la risoluzione dell'url da parte della funzione ipResolver che è riportata di seguito:

Listato 5: Risoluzione url del client

```
1  char *ipResolver(char **argv) {
2    char *ipAddress;
3    struct hostent *hp;
4    hp = gethostbyname(argv[1]);
5    if (!hp) {
6      perror("Impossibile risolvere l'indirizzo ip\n");
7      sleep(1);
8      exit(-1);
9    }
10   printf("Address:\t%s\n", inet_ntoa(*(struct in_addr *)hp->h_addr_list[0]));
11   return inet_ntoa(*(struct in_addr *)hp->h_addr_list[0]);
12 }
```

Al rigo 4, tramite l'url o l'indirizzo ip viene riempita la struttura hostent da cui poi possiamo estrarre l'indirizzo ip presente nel campo h_addr_list che, in effetti, è un array che contiene i vari indirizzi ip associati a quell'host.
Infine, al rigo 11 decidiamo di ritoranre soltanto il primo indirizzo convertito in Internet dot notation.

3.3 Comunicazione tra client e server

La comunicazione tra client e server avviene tramite write e read sul socket.
Il comportamento del server e del client è determinato da particolari messaggi inviati e/o ricevuti che codificano, tramite interi o caratteri, la richiesta da parte del client di usufruire di un determinato servizio e la relativa risposta del server.

3.3.1 Esempio: la prima comunicazione

In particolare, una volta effettuata la connessione, il server attenderà un messaggio dal client per poter avviare una delle tre possibili procedure, ovvero login, registrazione ed uscita (rispettivamente codici: 1,2,3).
Di seguito sono riportate le realtive funzioni di gestione che entrano in esecuzione subito dopo aver stabilito la connessione tra client e server.

Listato 6: Prima comunicazione del server

```
1       !isAlreadyLogged(onLineUsers, userName))
2   {
3     ret = 1;
4     numeroClientLoggati++;
5     write(clientDesc, "y", 1);
6     strcpy(name, userName);
7     Args args = (Args)malloc(sizeof(struct argsToSend));
8     args->userName = (char *)calloc(MAX_BUF, 1);
9     strcpy(args->userName, name);
10    args->flag = 0;
11    pthread_t tid;
12    pthread_create(&tid, NULL, fileWriter, (void *)args);
13    printf("Nuovo client loggato, client loggati : %d\n", numeroClientLoggati);
14    onLineUsers = addPlayer(onLineUsers, userName, clientDesc);
15    pthread_mutex_unlock(&PlayerMutex);
16    printPlayers(onLineUsers);
17    printf("\n");
18  }
19  else
20  {
21    write(clientDesc, "n", 1);
22  }
```

Si noti come il server riceva, al rigo 7, il messaggio codificato da parte del client e metta in esecuzione la funzione corrispondente.

Listato 7: Prima comunicazione del client

```
1  int gestisci() {
2    char choice;
3    while (1) {
4      printMenu();
5      scanf("%c", &choice);
6      fflush(stdin);
7      system("clear");
8      if (choice == '3') {
9        esciDalServer();
10       return (0);
11     } else if (choice == '2') {
12       registrati();
13     } else if (choice == '1') {
14       if (tryLogin())
15         play();
16     } else
17       printf("Input errato, inserire 1,2 o 3\n");
18   }
19 }
```

4 Comunicazione durante la partita

Una volta effettuato il login, il client potrà iniziare a giocare tramite la funzione play (Vedi List. 8 e List. 9) che rappresentano il cuore della comunicazione tra client e server.

4.1 Funzione core del server

La funzione play del server consiste di un ciclo nel quale il server invia al client tre informazioni importanti:

- La griglia di gioco (Rigo 26)
- Il player con le relative informazioni (Righi 28 a 31)
- Un messaggio che notifica al client se è iniziato un nuovo turno oppure no (Righi 40,43,46,53)

Dopodichè il thread del server rimane in attesa di ricevere l'input del client per spostare il giocaore sulla mappa tramite la relativa funzione. (Rigo 33) (Vedi List. 14 Rigo 430 e List. 16 Rigo 296, 331,367, 405) Oltre questo, la funzione play del server si occupa anche di generare la posizione del player appena entra in partita, generare la nuova posizione (Righi 56 a 65) quando viene effettuato il cambio di mappa ed inviare il tempo rimanente o la lista degli utente loggati su richiesta del client.

È anche importante notare il seguente dettaglio implementativo: la griglia di gioco è una matrice globale definita nel file del server che contiene tutti i player, i punti di raccolta ed i pacchi, mentre gli ostacoli sono contenuti in una seconda matrice globale del server. Ogni client però deve vedere soltanto gli ostacoli che ha già scoperto, per questo motivo ad ogni client non viene mandata direttamente la matrice di gioco, bensì, dai righi 22 a 24, inizializziamo una nuova matrice temporanea a cui aggiungiamo gli ostacoli già scoperti dal client (rigo 24) prima di mandarla al client stesso.
In questo modo ci assicuriamo che ogni client visualizzi soltanto gli ostacoli che ha già scoperto.

4.2 Funzione core del client

Dall'altro lato, la funzione play del client, è stata mantenuta il puù semplice possibile. Lo scopo del client è unicamente quello di ricevere i dati forniti dal server, stampare la mappa di gioco e ed inviare un input al server che rappresenta la volontà del giocatore di muoversi, vedere la lista degli utenti, uscire o stampare il timer. Unica eccezzione è il rigo 30 del client che non richiede la ricezione di ulteriori dati dal server: al rigo 23, infatti si avvia la procedura di disconnessione del client (Vedi List. 13 rigo 59).

Listato 8: Funzione play del server

```
  return ret;
}
void *gestisci(void *descriptor)
{
  int bufferReceive[2] = {1};
  int client_sd = *(int *)descriptor;
  int continua = 1;
  char name[MAX_BUF];
  while (continua)
  {
    read(client_sd, bufferReceive, sizeof(bufferReceive));
    if (bufferReceive[0] == 2)
      registraClient(client_sd);
    else if (bufferReceive[0] == 1)
      if (tryLogin(client_sd, name))
      {
        play(client_sd, name);
        continua = 0;
      }
      else if (bufferReceive[0] == 3)
        disconnettiClient(client_sd);
      else
      {
        printf("Input invalido, uscita...\n");
        disconnettiClient(client_sd);
      }
  }
  pthread_exit(0);
}
void play(int clientDesc, char name[])
{
  int true = 1;
  int turnoFinito = 0;
  int turnoGiocatore = turno;
  int posizione[2];
  int destinazione[2] = {-1, -1};
  PlayerStats giocatore = initStats(destinazione, 0, posizione, 0);
  Obstacles listaOstacoli = NULL;
  char inputFromClient;
  if (timer != 0)
  {
    inserisciPlayerNellaGrigliaInPosizioneCasuale(
        grigliaDiGiocoConPacchiSenzaOstacoli, grigliaOstacoliSenzaPacchi,
        giocatore->position);
    playerGenerati++;
  }
  while (true)
  {
    if (clientDisconnesso(clientDesc))
    {
      freeObstacles(listaOstacoli);
      disconnettiClient(clientDesc);
      return;
    }
    char grigliaTmp[ROWS][COLUMNS];
    clonaGriglia(grigliaTmp, grigliaDiGiocoConPacchiSenzaOstacoli);
    mergeGridAndList(grigliaTmp, listaOstacoli);
    // invia la griglia
    write(clientDesc, grigliaTmp, sizeof(grigliaTmp));
    // invia la struttura del player
    write(clientDesc, giocatore->deploy, sizeof(giocatore->deploy));
    write(clientDesc, giocatore->position, sizeof(giocatore->position));
    write(clientDesc, &giocatore->score, sizeof(giocatore->score));
    write(clientDesc, &giocatore->hasApack, sizeof(giocatore->hasApack));
    // legge l'input
    if (read(clientDesc, &inputFromClient, sizeof(char)) > 0)
      numMosse++;
    if (inputFromClient == 'e' || inputFromClient == 'E')
```

Listato 9: Funzione play del client

```
void play() {
  PlayerStats giocatore = NULL;
  int score, deploy[2], position[2], timer;
  int turnoFinito = 0;
  int exitFlag = 0, hasApack = 0;
  while (!exitFlag) {
    if (serverCaduto())
      serverCrashHandler();
    if (read(socketDesc, grigliaDiGioco, sizeof(grigliaDiGioco)) < 1)
      printf("Impossibile comunicare con il server\n"), exit(-1);
    if (read(socketDesc, deploy, sizeof(deploy)) < 1)
      printf("Impossibile comunicare con il server\n"), exit(-1);
    if (read(socketDesc, position, sizeof(position)) < 1)
      printf("Impossibile comunicare con il server\n"), exit(-1);
    if (read(socketDesc, &score, sizeof(score)) < 1)
      printf("Impossibile comunicare con il server\n"), exit(-1);
    if (read(socketDesc, &hasApack, sizeof(hasApack)) < 1)
      printf("Impossibile comunicare con il server\n"), exit(-1);
    giocatore = initStats(deploy, score, position, hasApack);
    printGrid(grigliaDiGioco, giocatore);
    char send = getUserInput();
    if (send == 'e' || send == 'E') {
      esciDalServer();
      exit(0);
    }
    write(socketDesc, &send, sizeof(char));
    read(socketDesc, &turnoFinito, sizeof(turnoFinito));
    if (turnoFinito) {
      system("clear");
      printf("Turno finito\n");
      sleep(1);
    } else {
      if (send == 't' || send == 'T')
        printTimer();
      else if (send == 'l' || send == 'L')
        printPlayerList();
    }
  }
}
```

5 Dettagli implementativi degni di nota

In questa sezione verranno trattati alcuni dettagli implementativi da noi giudicati interessanti in realzione a ciò che è stato studiato durante il corso di sistemi operativi.

5.1 Timer

Lo svolgimento della partite è legato al timer: ogni round durerà un numero finito di secondi od oppure terminarà quando un client raccoglierà il numero massimo di pacchi.

Subito dopo aver configurato il socket, il server inizia la procedura di avvio del timer (Vedi List. 14 rigo 89 e 144) che farà partire un thread il quale si occuperà di decrementare e resettare correttamente il timer (definito come variabile globale del server).

Listato 10: Funzione di gestione del timer

```
1   read(clientDesc, password, dimPwd);
2   pthread_mutex_lock(&RegMutex);
3   int ret = appendPlayer(userName, password, users);
4   pthread_mutex_unlock(&RegMutex);
5   char risposta;
6   if (!ret)
7   {
8     risposta = 'n';
9     write(clientDesc, &risposta, sizeof(char));
10    printf("Impossibile registrare utente, riprovare\n");
11  }
12  else
13  {
14    risposta = 'y';
15    write(clientDesc, &risposta, sizeof(char));
16    printf("Utente registrato con successo\n");
17  }
18  return ret;
19 }
20 void quitServer()
21 {
22   printf("Chiusura server in corso..\n");
23   close(socketDesc);
24   exit(-1);
25 }
26 void *threadGenerazioneMappa(void *args)
27 {
28   fprintf(stdout, "Rigenerazione mappa\n");
```

Analizzando il codice della funzione di modifica del timer si può notare un dettaglio abbastanza interessante: il thread che esegue la funzione del timer è legato ad un altro thread, ovvero quello della generazione di una nuova mappa. Oltre ad un thread per gestire ogni client abbiamo quindi anche un altro thread che va a gestire il tempo , il quale attraverso un altro thread riesce a controllare la generazione della mappa e degli utenti allo scadere del tempo. Si noti anche come, tramite il pthread_join, il timer attenda la terminazione del secondo thread prima di resettare il timer e ricominciare il contro alla rovescia. [1]

[1] Altro dettaglio meno importante, ma comunque degno di nota è il fatto che il timer non inizia il conto alla rovescia se non c'è almeno un giocatore loggato, se questo non è stato posizionato sulla mappa e se questo non ha effettuato la prima mossa. Al rigo 2 c'è anche da giustificare la variabile "cambiato" che non è nient'altro che un flag, il quale impedisce al server di stampare in stdout il valore del timer nel caso in cui esso sia stato appena resettato e non sia ancora iniziato il conto alla rovescia. Ciò evita che, prima che inizi il conto alla rovescia, il server continui a stampare il valore massimo del timer

Per completezza si riporta anche la funzionione iniziale del thread di generazione mappa

Listato 11: Generazione nuova mappa e posizione players

```
void disconnettiClient(int clientDescriptor)
{
  if (numeroClientLoggati > 0)
    numeroClientLoggati--;
  pthread_mutex_lock(&PlayerMutex);
  onLineUsers = removePlayer(onLineUsers, clientDescriptor);
  pthread_mutex_unlock(&PlayerMutex);
  printPlayers(onLineUsers);
  int msg = 1;
  printf("Client disconnesso (client attualmente loggati: %d)\n",
         numeroClientLoggati);
  write(clientDescriptor, &msg, sizeof(msg));
```

5.2 Gestione del file di Log

Una delle funzionalità del server è quella di creare un file di log con varie informazioni durante la sua esecuzione. Riteniamo l'implmentazione di questa funzione piuttosto interessante poichè, oltre ad essere una funzione gestita tramite un thread, fa uso sia di molte chiamate di sistema studiate durante il corso ed utilizza anche il mutex per risolvere eventuali race condition. Riportiamo di seguito il codice:

Listato 12: Funzione di log

```
void *fileWriter(void *args)
{
  int fDes = open("Log", O_RDWR | O_CREAT | O_APPEND, S_IWUSR | S_IRUSR);
  if (fDes < 0)
  {
    perror("Error while opening log file");
    exit(-1);
  }
  Args info = (Args)args;
  char dateAndTime[64];
  putCurrentDateAndTimeInString(dateAndTime);
  if (logDelPacco(info->flag))
  {
    char message[MAX_BUF] = "";
    prepareMessageForPackDelivery(message, info->userName, dateAndTime);
    pthread_mutex_lock(&LogMutex);
    write(fDes, message, strlen(message));
    pthread_mutex_unlock(&LogMutex);
  }
  else if (logDelLogin(info->flag))
  {
    char message[MAX_BUF] = "\"";
    prepareMessageForLogin(message, info->userName, dateAndTime);
    pthread_mutex_lock(&LogMutex);
    write(fDes, message, strlen(message));
    pthread_mutex_unlock(&LogMutex);
  }
  else if (logDellaConnessione(info->flag))
  {
    char message[MAX_BUF] = "\"";
    prepareMessageForConnection(message, info->userName, dateAndTime);
    pthread_mutex_lock(&LogMutex);
    write(fDes, message, strlen(message));
    pthread_mutex_unlock(&LogMutex);
  }
  close(fDes);
  free(info);
  pthread_exit(NULL);
}
```

A Codici sorgente

Di seguito sono riportati tutti i codici sorgenti integrali del progetto.

A.1 Codice sorgente del client

Listato 13: Codice sorgente del client

```
#include "boardUtility.h"
#include "list.h"
#include "parser.h"
#include <arpa/inet.h>
#include <fcntl.h>
#include <netdb.h>
#include <netinet/in.h> //conversioni
```

```c
#include <netinet/in.h>
#include <netinet/ip.h> //struttura
#include <pthread.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>

void printPlayerList();
int getTimer();
void printTimer();
void play();
int tryLogin();
void printMenu();
int connettiAlServer(char **argv);
char *ipResolver(char **argv);
int registrati();
int gestisci();
char getUserInput();
void clientCrashHandler();
void serverCrashHandler();
int serverCaduto();
void esciDalServer();
int isCorrect(char);

int socketDesc;
char grigliaDiGioco[ROWS][COLUMNS];

int main(int argc, char **argv) {
  signal(SIGINT, clientCrashHandler); /* CTRL-C */
  signal(SIGHUP, clientCrashHandler); /* Chiusura della console */
  signal(SIGQUIT, clientCrashHandler);
  signal(SIGTSTP, clientCrashHandler); /* CTRL-Z*/
  signal(SIGTERM, clientCrashHandler); /* generato da 'kill' */
  signal(SIGPIPE, serverCrashHandler);
  char bufferReceive[2];
  if (argc != 3) {
    perror("Inserire indirizzo ip/url e porta (./client 127.0.0.1 5200)");
    exit(-1);
  }
  if ((socketDesc = connettiAlServer(argv)) < 0)
    exit(-1);
  gestisci(socketDesc);
  close(socketDesc);
  exit(0);
}
void esciDalServer() {
  int msg = 3;
  printf("Uscita in corso\n");
  write(socketDesc, &msg, sizeof(int));
  close(socketDesc);
}
int connettiAlServer(char **argv) {
  char *indirizzoServer;
  uint16_t porta = strtoul(argv[2], NULL, 10);
  indirizzoServer = ipResolver(argv);
  struct sockaddr_in mio_indirizzo;
  mio_indirizzo.sin_family = AF_INET;
  mio_indirizzo.sin_port = htons(porta);
  inet_aton(indirizzoServer, &mio_indirizzo.sin_addr);
  if ((socketDesc = socket(PF_INET, SOCK_STREAM, 0)) < 0)
    perror("Impossibile creare socket"), exit(-1);
  else
    printf("Socket creato\n");
  if (connect(socketDesc, (struct sockaddr *)&mio_indirizzo,
              sizeof(mio_indirizzo)) < 0)
    perror("Impossibile connettersi"), exit(-1);
  else
    printf("Connesso a %s\n", indirizzoServer);
```

```
82    return socketDesc;
83  }
84  int gestisci() {
85    char choice;
86    while (1) {
87      printMenu();
88      scanf("%c", &choice);
89      fflush(stdin);
90      system("clear");
91      if (choice == '3') {
92        esciDalServer();
93        return (0);
94      } else if (choice == '2') {
95        registrati();
96      } else if (choice == '1') {
97        if (tryLogin())
98          play();
99      } else
100       printf("Input errato, inserire 1,2 o 3\n");
101   }
102 }
103 int serverCaduto() {
104   char msg = 'y';
105   if (read(socketDesc, &msg, sizeof(char)) == 0)
106     return 1;
107   else
108     write(socketDesc, &msg, sizeof(msg));
109   return 0;
110 }
111 void play() {
112   PlayerStats giocatore = NULL;
113   int score, deploy[2], position[2], timer;
114   int turnoFinito = 0;
115   int exitFlag = 0, hasApack = 0;
116   while (!exitFlag) {
117     if (serverCaduto())
118       serverCrashHandler();
119     if (read(socketDesc, grigliaDiGioco, sizeof(grigliaDiGioco)) < 1)
120       printf("Impossibile comunicare con il server\n"), exit(-1);
121     if (read(socketDesc, deploy, sizeof(deploy)) < 1)
122       printf("Impossibile comunicare con il server\n"), exit(-1);
123     if (read(socketDesc, position, sizeof(position)) < 1)
124       printf("Impossibile comunicare con il server\n"), exit(-1);
125     if (read(socketDesc, &score, sizeof(score)) < 1)
126       printf("Impossibile comunicare con il server\n"), exit(-1);
127     if (read(socketDesc, &hasApack, sizeof(hasApack)) < 1)
128       printf("Impossibile comunicare con il server\n"), exit(-1);
129     giocatore = initStats(deploy, score, position, hasApack);
130     printGrid(grigliaDiGioco, giocatore);
131     char send = getUserInput();
132     if (send == 'e' || send == 'E') {
133       esciDalServer();
134       exit(0);
135     }
136     write(socketDesc, &send, sizeof(char));
137     read(socketDesc, &turnoFinito, sizeof(turnoFinito));
138     if (turnoFinito) {
139       system("clear");
140       printf("Turno finito\n");
141       sleep(1);
142     } else {
143       if (send == 't' || send == 'T')
144         printTimer();
145       else if (send == 'l' || send == 'L')
146         printPlayerList();
147     }
148   }
149 }
150 void printPlayerList() {
151   system("clear");
152   int lunghezza = 0;
153   char buffer[100];
154   int continua = 1;
155   int number = 1;
```

```
156   fprintf(stdout, "Lista dei player: \n");
157   if (!serverCaduto(socketDesc)) {
158     read(socketDesc, &continua, sizeof(continua));
159     while (continua) {
160       read(socketDesc, &lunghezza, sizeof(lunghezza));
161       read(socketDesc, buffer, lunghezza);
162       buffer[lunghezza] = '\0';
163       fprintf(stdout, "%d) %s\n", number, buffer);
164       continua--;
165       number++;
166     }
167     sleep(1);
168   }
169 }
170 void printTimer() {
171   int timer;
172   if (!serverCaduto(socketDesc)) {
173     read(socketDesc, &timer, sizeof(timer));
174     printf("\t\tTempo restante: %d...\n", timer);
175     sleep(1);
176   }
177 }
178 int getTimer() {
179   int timer;
180   if (!serverCaduto(socketDesc))
181     read(socketDesc, &timer, sizeof(timer));
182   return timer;
183 }
184 int tryLogin() {
185   int msg = 1;
186   write(socketDesc, &msg, sizeof(int));
187   system("clear");
188   printf("Inserisci i dati per il Login\n");
189   char username[20];
190   char password[20];
191   printf("Inserisci nome utente(MAX 20 caratteri): ");
192   scanf("%s", username);
193   printf("\nInserisci password(MAX 20 caratteri):");
194   scanf("%s", password);
195   int dimUname = strlen(username), dimPwd = strlen(password);
196   if (write(socketDesc, &dimUname, sizeof(dimUname)) < 0)
197     return 0;
198   if (write(socketDesc, &dimPwd, sizeof(dimPwd)) < 0)
199     return 0;
200   if (write(socketDesc, username, dimUname) < 0)
201     return 0;
202   if (write(socketDesc, password, dimPwd) < 0)
203     return 0;
204   char validate;
205   int ret;
206   read(socketDesc, &validate, 1);
207   if (validate == 'y') {
208     ret = 1;
209     printf("Accesso effettuato\n");
210   } else if (validate == 'n') {
211     printf("Credenziali Errate o Login giá effettuato\n");
212     ret = 0;
213   }
214   sleep(1);
215   return ret;
216 }
217 int registrati() {
218   int msg = 2;
219   write(socketDesc, &msg, sizeof(int));
220   char username[20];
221   char password[20];
222   system("clear");
223   printf("Inserisci nome utente(MAX 20 caratteri): ");
224   scanf("%s", username);
225   printf("\nInserisci password(MAX 20 caratteri):");
226   scanf("%s", password);
227   int dimUname = strlen(username), dimPwd = strlen(password);
228   if (write(socketDesc, &dimUname, sizeof(dimUname)) < 0)
229     return 0;
```

```
230   if (write(socketDesc, &dimPwd, sizeof(dimPwd)) < 0)
231     return 0;
232   if (write(socketDesc, username, dimUname) < 0)
233     return 0;
234   if (write(socketDesc, password, dimPwd) < 0)
235     return 0;
236   char validate;
237   int ret;
238   read(socketDesc, &validate, sizeof(char));
239   if (validate == 'y') {
240     ret = 1;
241     printf("Registrato con successo\n");
242   }
243   if (validate == 'n') {
244     ret = 0;
245     printf("Registrazione fallita\n");
246   }
247   sleep(1);
248   return ret;
249 }
250 char *ipResolver(char **argv) {
251   char *ipAddress;
252   struct hostent *hp;
253   hp = gethostbyname(argv[1]);
254   if (!hp) {
255     perror("Impossibile risolvere l'indirizzo ip\n");
256     sleep(1);
257     exit(-1);
258   }
259   printf("Address:\t%s\n", inet_ntoa(*(struct in_addr *)hp->h_addr_list[0]));
260   return inet_ntoa(*(struct in_addr *)hp->h_addr_list[0]);
261 }
262 void clientCrashHandler() {
263   int msg = 3;
264   int rec = 0;
265   printf("\nChiusura client...\n");
266   do {
267     write(socketDesc, &msg, sizeof(int));
268     read(socketDesc, &rec, sizeof(int));
269   } while (rec == 0);
270   close(socketDesc);
271   signal(SIGINT, SIG_IGN);
272   signal(SIGQUIT, SIG_IGN);
273   signal(SIGTERM, SIG_IGN);
274   signal(SIGTSTP, SIG_IGN);
275   exit(0);
276 }
277 void serverCrashHandler() {
278   system("clear");
279   printf("Il server á stato spento o á irraggiungibile\n");
280   close(socketDesc);
281   signal(SIGPIPE, SIG_IGN);
282   premiEnterPerContinuare();
283   exit(0);
284 }
285 char getUserInput() {
286   char c;
287   c = getchar();
288   int daIgnorare;
289   while ((daIgnorare = getchar()) != '\n' && daIgnorare != EOF) {
290   }
291   return c;
292 }
```

A.2 Codice sorgente del server

Listato 14: Codice sorgente del server

```
1 #include "boardUtility.h"
2 #include "list.h"
3 #include "parser.h"
```

```c
#include <arpa/inet.h>
#include <errno.h>
#include <fcntl.h>
#include <netinet/in.h> //conversioni
#include <netinet/ip.h> //struttura
#include <pthread.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>

//struttura di argomenti da mandare al thread che scrive sul file di log
struct argsToSend
{
  char *userName;
  int flag;
};

typedef struct argsToSend *Args;
void prepareMessageForLogin(char message[], char username[], char date[]);
void sendPlayerList(int clientDesc);
PlayerStats gestisciC(char grigliaDiGioco[ROWS][COLUMNS], PlayerStats giocatore,
                      Point deployCoords[], Point packsCoords[], char name[]);
PlayerStats gestisciInput(char grigliaDiGioco[ROWS][COLUMNS],
                          char grigliaOstacoli[ROWS][COLUMNS], char input,
                          PlayerStats giocatore, Obstacles *listaOstacoli,
                          Point deployCoords[], Point packsCoords[],
                          char name[]);
void clonaGriglia(char destinazione[ROWS][COLUMNS], char source[ROWS][COLUMNS]);
int almenoUnClientConnesso();
void prepareMessageForConnection(char message[], char ipAddress[], char date[]);
int valoreTimerValido();
int almenoUnPlayerGenerato();
int almenoUnaMossaFatta();
void sendTimerValue(int clientDesc);
void putCurrentDateAndTimeInString(char dateAndTime[]);
void startProceduraGenrazioneMappa();
void *threadGenerazioneMappa(void *args);
void *fileWriter(void *);
int tryLogin(int clientDesc, char name[]);
void disconnettiClient(int);
int registraClient(int);
void *timer(void *args);
void *gestisci(void *descriptor);
void quitServer();
void clientCrashHandler(int signalNum);
void startTimer();
void configuraSocket(struct sockaddr_in mio_indirizzo);
struct sockaddr_in configuraIndirizzo();
void startListening();
int clientDisconnesso(int clientSocket);
void play(int clientDesc, char name[]);
void prepareMessageForPackDelivery(char message[], char username[], char date[]);
int logDelPacco(int flag);
int logDelLogin(int flag);
int logDellaConnessione(int flag);
char grigliaDiGiocoConPacchiSenzaOstacoli[ROWS][COLUMNS];
char grigliaOstacoliSenzaPacchi[ROWS][COLUMNS];
int numeroClientLoggati = 0;
int playerGenerati = 0;
int timerCount = TIME_LIMIT_IN_SECONDS;
int turno = 0;
pthread_t tidTimer;
pthread_t tidGeneratoreMappa;
int socketDesc;
Players onLineUsers = NULL;
char *users;
int scoreMassimo = 0;
int numMosse = 0;
```

```
78  Point deployCoords[numberOfPackages];
79  Point packsCoords[numberOfPackages];
80  pthread_mutex_t LogMutex = PTHREAD_MUTEX_INITIALIZER;
81  pthread_mutex_t RegMutex = PTHREAD_MUTEX_INITIALIZER;
82  pthread_mutex_t PlayerMutex = PTHREAD_MUTEX_INITIALIZER;
83
84  int main(int argc, char **argv)
85  {
86    if (argc != 2)
87    {
88      printf("Wrong parameters number(Usage: ./server usersFile)\n");
89      exit(-1);
90    }
91    else if (strcmp(argv[1], "Log") == 0)
92    {
93      printf("Cannot use the Log file as a UserList \n");
94      exit(-1);
95    }
96    users = argv[1];
97    struct sockaddr_in mio_indirizzo = configuraIndirizzo();
98    configuraSocket(mio_indirizzo);
99    signal(SIGPIPE, clientCrashHandler);
100   signal(SIGINT, quitServer);
101   signal(SIGHUP, quitServer);
102   startTimer();
103   inizializzaGiocoSenzaPlayer(grigliaDiGiocoConPacchiSenzaOstacoli,
104                               grigliaOstacoliSenzaPacchi, packsCoords);
105   generaPosizioniRaccolta(grigliaDiGiocoConPacchiSenzaOstacoli,
106                           grigliaOstacoliSenzaPacchi, deployCoords);
107   startListening();
108   return 0;
109 }
110 void startListening()
111 {
112   pthread_t tid;
113   int clientDesc;
114   int *puntClientDesc;
115   while (1 == 1)
116   {
117     if (listen(socketDesc, 10) < 0)
118       perror("Impossibile mettersi in ascolto"), exit(-1);
119     printf("In ascolto..\n");
120     if ((clientDesc = accept(socketDesc, NULL, NULL)) < 0)
121     {
122       perror("Impossibile effettuare connessione\n");
123       exit(-1);
124     }
125     printf("Nuovo client connesso\n");
126     struct sockaddr_in address;
127     socklen_t size = sizeof(struct sockaddr_in);
128     if (getpeername(clientDesc, (struct sockaddr *)&address, &size) < 0)
129     {
130       perror("Impossibile ottenere l'indirizzo del client");
131       exit(-1);
132     }
133     char clientAddr[20];
134     strcpy(clientAddr, inet_ntoa(address.sin_addr));
135     Args args = (Args)malloc(sizeof(struct argsToSend));
136     args->userName = (char *)calloc(MAX_BUF, 1);
137     strcpy(args->userName, clientAddr);
138     args->flag = 2;
139     pthread_t tid;
140     pthread_create(&tid, NULL, fileWriter, (void *)args);
141
142     puntClientDesc = (int *)malloc(sizeof(int));
143     *puntClientDesc = clientDesc;
144     pthread_create(&tid, NULL, gestisci, (void *)puntClientDesc);
145   }
146   close(clientDesc);
147   quitServer();
148 }
149 struct sockaddr_in configuraIndirizzo()
150 {
151   struct sockaddr_in mio_indirizzo;
```

```
152   mio_indirizzo.sin_family = AF_INET;
153   mio_indirizzo.sin_port = htons(5200);
154   mio_indirizzo.sin_addr.s_addr = htonl(INADDR_ANY);
155   printf("Indirizzo socket configurato\n");
156   return mio_indirizzo;
157 }
158 void startProceduraGenrazioneMappa()
159 {
160   printf("Inizio procedura generazione mappa\n");
161   pthread_create(&tidGeneratoreMappa, NULL, threadGenerazioneMappa, NULL);
162 }
163 void startTimer()
164 {
165   printf("Thread timer avviato\n");
166   pthread_create(&tidTimer, NULL, timer, NULL);
167 }
168 int tryLogin(int clientDesc, char name[])
169 {
170   char *userName = (char *)calloc(MAX_BUF, 1);
171   char *password = (char *)calloc(MAX_BUF, 1);
172   int dimName, dimPwd;
173   read(clientDesc, &dimName, sizeof(int));
174   read(clientDesc, &dimPwd, sizeof(int));
175   read(clientDesc, userName, dimName);
176   read(clientDesc, password, dimPwd);
177   int ret = 0;
178   pthread_mutex_lock(&PlayerMutex);
179   if (validateLogin(userName, password, users) &&
180       !isAlreadyLogged(onLineUsers, userName))
181   {
182     ret = 1;
183     numeroClientLoggati++;
184     write(clientDesc, "y", 1);
185     strcpy(name, userName);
186     Args args = (Args)malloc(sizeof(struct argsToSend));
187     args->userName = (char *)calloc(MAX_BUF, 1);
188     strcpy(args->userName, name);
189     args->flag = 0;
190     pthread_t tid;
191     pthread_create(&tid, NULL, fileWriter, (void *)args);
192     printf("Nuovo client loggato, client loggati : %d\n", numeroClientLoggati);
193     onLineUsers = addPlayer(onLineUsers, userName, clientDesc);
194     pthread_mutex_unlock(&PlayerMutex);
195     printPlayers(onLineUsers);
196     printf("\n");
197   }
198   else
199   {
200     write(clientDesc, "n", 1);
201   }
202   return ret;
203 }
204 void *gestisci(void *descriptor)
205 {
206   int bufferReceive[2] = {1};
207   int client_sd = *(int *)descriptor;
208   int continua = 1;
209   char name[MAX_BUF];
210   while (continua)
211   {
212     read(client_sd, bufferReceive, sizeof(bufferReceive));
213     if (bufferReceive[0] == 2)
214       registraClient(client_sd);
215     else if (bufferReceive[0] == 1)
216       if (tryLogin(client_sd, name))
217       {
218         play(client_sd, name);
219         continua = 0;
220       }
221       else if (bufferReceive[0] == 3)
222         disconnettiClient(client_sd);
223       else
224       {
225         printf("Input invalido, uscita...\n");
```

```
226         disconnettiClient(client_sd);
227       }
228   }
229   pthread_exit(0);
230 }
231 void play(int clientDesc, char name[])
232 {
233   int true = 1;
234   int turnoFinito = 0;
235   int turnoGiocatore = turno;
236   int posizione[2];
237   int destinazione[2] = {-1, -1};
238   PlayerStats giocatore = initStats(destinazione, 0, posizione, 0);
239   Obstacles listaOstacoli = NULL;
240   char inputFromClient;
241   if (timer != 0)
242   {
243     inserisciPlayerNellaGrigliaInPosizioneCasuale(
244         grigliaDiGiocoConPacchiSenzaOstacoli, grigliaOstacoliSenzaPacchi,
245         giocatore->position);
246     playerGenerati++;
247   }
248   while (true)
249   {
250     if (clientDisconnesso(clientDesc))
251     {
252       freeObstacles(listaOstacoli);
253       disconnettiClient(clientDesc);
254       return;
255     }
256     char grigliaTmp[ROWS][COLUMNS];
257     clonaGriglia(grigliaTmp, grigliaDiGiocoConPacchiSenzaOstacoli);
258     mergeGridAndList(grigliaTmp, listaOstacoli);
259     // invia la griglia
260     write(clientDesc, grigliaTmp, sizeof(grigliaTmp));
261     // invia la struttura del player
262     write(clientDesc, giocatore->deploy, sizeof(giocatore->deploy));
263     write(clientDesc, giocatore->position, sizeof(giocatore->position));
264     write(clientDesc, &giocatore->score, sizeof(giocatore->score));
265     write(clientDesc, &giocatore->hasApack, sizeof(giocatore->hasApack));
266     // legge l'input
267     if (read(clientDesc, &inputFromClient, sizeof(char)) > 0)
268       numMosse++;
269     if (inputFromClient == 'e' || inputFromClient == 'E')
270     {
271       freeObstacles(listaOstacoli);
272       listaOstacoli = NULL;
273       disconnettiClient(clientDesc);
274     }
275     else if (inputFromClient == 't' || inputFromClient == 'T')
276     {
277       write(clientDesc, &turnoFinito, sizeof(int));
278       sendTimerValue(clientDesc);
279     }
280     else if (inputFromClient == 'l' || inputFromClient == 'L')
281     {
282       write(clientDesc, &turnoFinito, sizeof(int));
283       sendPlayerList(clientDesc);
284     }
285     else if (turnoGiocatore == turno)
286     {
287       write(clientDesc, &turnoFinito, sizeof(int));
288       giocatore =
289           gestisciInput(grigliaDiGiocoConPacchiSenzaOstacoli,
290                         grigliaOstacoliSenzaPacchi, inputFromClient, giocatore,
291                         &listaOstacoli, deployCoords, packsCoords, name);
292     }
293     else
294     {
295       turnoFinito = 1;
296       write(clientDesc, &turnoFinito, sizeof(int));
297       freeObstacles(listaOstacoli);
298       listaOstacoli = NULL;
299       inserisciPlayerNellaGrigliaInPosizioneCasuale(
```

```
300             grigliaDiGiocoConPacchiSenzaOstacoli, grigliaOstacoliSenzaPacchi,
301             giocatore->position);
302         giocatore->score = 0;
303         giocatore->hasApack = 0;
304         giocatore->deploy[0] = -1;
305         giocatore->deploy[1] = -1;
306         turnoGiocatore = turno;
307         turnoFinito = 0;
308         playerGenerati++;
309       }
310     }
311   }
312   void sendTimerValue(int clientDesc)
313   {
314     if (!clientDisconnesso(clientDesc))
315       write(clientDesc, &timerCount, sizeof(timerCount));
316   }
317   void clonaGriglia(char destinazione[ROWS][COLUMNS],
318                     char source[ROWS][COLUMNS])
319   {
320     int i = 0, j = 0;
321     for (i = 0; i < ROWS; i++)
322     {
323       for (j = 0; j < COLUMNS; j++)
324       {
325         destinazione[i][j] = source[i][j];
326       }
327     }
328   }
329   void clientCrashHandler(int signalNum)
330   {
331     char msg[0];
332     int socketClientCrashato;
333     int flag = 1;
334     // TODO eliminare la lista degli ostacoli dell'utente
335     if (onLineUsers != NULL)
336     {
337       Players prec = onLineUsers;
338       Players top = prec->next;
339       while (top != NULL && flag)
340       {
341         if (write(top->sockDes, msg, sizeof(msg)) < 0)
342         {
343           socketClientCrashato = top->sockDes;
344           printPlayers(onLineUsers);
345           disconnettiClient(socketClientCrashato);
346           flag = 0;
347         }
348         top = top->next;
349       }
350     }
351     signal(SIGPIPE, SIG_IGN);
352   }
353   void disconnettiClient(int clientDescriptor)
354   {
355     if (numeroClientLoggati > 0)
356       numeroClientLoggati--;
357     pthread_mutex_lock(&PlayerMutex);
358     onLineUsers = removePlayer(onLineUsers, clientDescriptor);
359     pthread_mutex_unlock(&PlayerMutex);
360     printPlayers(onLineUsers);
361     int msg = 1;
362     printf("Client disconnesso (client attualmente loggati: %d)\n",
363            numeroClientLoggati);
364     write(clientDescriptor, &msg, sizeof(msg));
365     close(clientDescriptor);
366   }
367   int clientDisconnesso(int clientSocket)
368   {
369     char msg[1] = {'u'}; // UP?
370     if (write(clientSocket, msg, sizeof(msg)) < 0)
371       return 1;
372     if (read(clientSocket, msg, sizeof(char)) < 0)
373       return 1;
```

```
374   else
375     return 0;
376 }
377 int registraClient(int clientDesc)
378 {
379   char *userName = (char *)calloc(MAX_BUF, 1);
380   char *password = (char *)calloc(MAX_BUF, 1);
381   int dimName, dimPwd;
382   read(clientDesc, &dimName, sizeof(int));
383   read(clientDesc, &dimPwd, sizeof(int));
384   read(clientDesc, userName, dimName);
385   read(clientDesc, password, dimPwd);
386   pthread_mutex_lock(&RegMutex);
387   int ret = appendPlayer(userName, password, users);
388   pthread_mutex_unlock(&RegMutex);
389   char risposta;
390   if (!ret)
391   {
392     risposta = 'n';
393     write(clientDesc, &risposta, sizeof(char));
394     printf("Impossibile registrare utente, riprovare\n");
395   }
396   else
397   {
398     risposta = 'y';
399     write(clientDesc, &risposta, sizeof(char));
400     printf("Utente registrato con successo\n");
401   }
402   return ret;
403 }
404 void quitServer()
405 {
406   printf("Chiusura server in corso..\n");
407   close(socketDesc);
408   exit(-1);
409 }
410 void *threadGenerazioneMappa(void *args)
411 {
412   fprintf(stdout, "Rigenerazione mappa\n");
413   inizializzaGrigliaVuota(grigliaDiGiocoConPacchiSenzaOstacoli);
414   generaPosizioniRaccolta(grigliaDiGiocoConPacchiSenzaOstacoli,
415                           grigliaOstacoliSenzaPacchi, deployCoords);
416   riempiGrigliaConPacchiInPosizioniGenerateCasualmente(
417       grigliaDiGiocoConPacchiSenzaOstacoli, packsCoords);
418   generaPosizioneOstacoli(grigliaDiGiocoConPacchiSenzaOstacoli,
419                           grigliaOstacoliSenzaPacchi);
420   printf("Mappa generata\n");
421   pthread_exit(NULL);
422 }
423 int almenoUnaMossaFatta()
424 {
425   if (numMosse > 0)
426     return 1;
427   return 0;
428 }
429 int almenoUnClientConnesso()
430 {
431   if (numeroClientLoggati > 0)
432     return 1;
433   return 0;
434 }
435 int valoreTimerValido()
436 {
437   if (timerCount > 0 && timerCount <= TIME_LIMIT_IN_SECONDS)
438     return 1;
439   return 0;
440 }
441 int almenoUnPlayerGenerato()
442 {
443   if (playerGenerati > 0)
444     return 1;
445   return 0;
446 }
447 void *timer(void *args)
```

```
448 {
449   int cambiato = 1;
450   while (1)
451   {
452     if (almenoUnClientConnesso() && valoreTimerValido() &&
453         almenoUnPlayerGenerato() && almenoUnaMossaFatta())
454     {
455       cambiato = 1;
456       sleep(1);
457       timerCount--;
458       fprintf(stdout, "Time left: %d\n", timerCount);
459     }
460     else if (numeroClientLoggati == 0)
461     {
462       timerCount = TIME_LIMIT_IN_SECONDS;
463       if (cambiato)
464       {
465         fprintf(stdout, "Time left: %d\n", timerCount);
466         cambiato = 0;
467       }
468     }
469     if (timerCount == 0 || scoreMassimo == packageLimitNumber)
470     {
471       playerGenerati = 0;
472       numMosse = 0;
473       printf("Reset timer e generazione nuova mappa..\n");
474       startProceduraGenrazioneMappa();
475       pthread_join(tidGeneratoreMappa, NULL);
476       turno++;
477       timerCount = TIME_LIMIT_IN_SECONDS;
478     }
479   }
480 }
481
482 void configuraSocket(struct sockaddr_in mio_indirizzo)
483 {
484   if ((socketDesc = socket(PF_INET, SOCK_STREAM, 0)) < 0)
485   {
486     perror("Impossibile creare socket");
487     exit(-1);
488   }
489   if (setsockopt(socketDesc, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) <
490       0)
491     perror("Impossibile impostare il riutilizzo dell'indirizzo ip e della "
492            "porta\n");
493   if ((bind(socketDesc, (struct sockaddr *)&mio_indirizzo,
494             sizeof(mio_indirizzo))) < 0)
495   {
496     perror("Impossibile effettuare bind");
497     exit(-1);
498   }
499 }
500
501 PlayerStats gestisciInput(char grigliaDiGioco[ROWS][COLUMNS],
502                           char grigliaOstacoli[ROWS][COLUMNS], char input,
503                           PlayerStats giocatore, Obstacles *listaOstacoli,
504                           Point deployCoords[], Point packsCoords[],
505                           char name[])
506 {
507   if (giocatore == NULL)
508   {
509     return NULL;
510   }
511   if (input == 'w' || input == 'W')
512   {
513     giocatore = gestisciW(grigliaDiGioco, grigliaOstacoli, giocatore,
514                           listaOstacoli, deployCoords, packsCoords);
515   }
516   else if (input == 's' || input == 'S')
517   {
518     giocatore = gestisciS(grigliaDiGioco, grigliaOstacoli, giocatore,
519                           listaOstacoli, deployCoords, packsCoords);
520   }
521   else if (input == 'a' || input == 'A')
```

```
522     {
523       giocatore = gestisciA(grigliaDiGioco, grigliaOstacoli, giocatore,
524                             listaOstacoli, deployCoords, packsCoords);
525     }
526     else if (input == 'd' || input == 'D')
527     {
528       giocatore = gestisciD(grigliaDiGioco, grigliaOstacoli, giocatore,
529                             listaOstacoli, deployCoords, packsCoords);
530     }
531     else if (input == 'p' || input == 'P')
532     {
533       giocatore = gestisciP(grigliaDiGioco, giocatore, deployCoords, packsCoords);
534     }
535     else if (input == 'c' || input == 'C')
536     {
537       giocatore =
538           gestisciC(grigliaDiGioco, giocatore, deployCoords, packsCoords, name);
539     }
540
541     // aggiorna la posizione dell'utente
542     return giocatore;
543   }
544
545   PlayerStats gestisciC(char grigliaDiGioco[ROWS][COLUMNS], PlayerStats giocatore,
546                         Point deployCoords[], Point packsCoords[], char name[])
547   {
548     pthread_t tid;
549     if (giocatore->hasApack == 0)
550     {
551       return giocatore;
552     }
553     else
554     {
555       if (isOnCorrectDeployPoint(giocatore, deployCoords))
556       {
557         Args args = (Args)malloc(sizeof(struct argsToSend));
558         args->userName = (char *)calloc(MAX_BUF, 1);
559         strcpy(args->userName, name);
560         args->flag = 1;
561         pthread_create(&tid, NULL, fileWriter, (void *)args);
562         giocatore->score += 10;
563         if (giocatore->score > scoreMassimo)
564           scoreMassimo = giocatore->score;
565         giocatore->deploy[0] = -1;
566         giocatore->deploy[1] = -1;
567         giocatore->hasApack = 0;
568       }
569       else
570       {
571         if (!isOnAPack(giocatore, packsCoords) &&
572             !isOnADeployPoint(giocatore, deployCoords))
573         {
574           int index = getHiddenPack(packsCoords);
575           if (index >= 0)
576           {
577             packsCoords[index]->x = giocatore->position[0];
578             packsCoords[index]->y = giocatore->position[1];
579             giocatore->hasApack = 0;
580             giocatore->deploy[0] = -1;
581             giocatore->deploy[1] = -1;
582           }
583         }
584         else
585           return giocatore;
586       }
587     }
588     return giocatore;
589   }
590
591   void sendPlayerList(int clientDesc)
592   {
593     int lunghezza = 0;
594     char name[100];
595     Players tmp = onLineUsers;
```

```
596   int numeroClientLoggati = dimensioneLista(tmp);
597   printf("%d ", numeroClientLoggati);
598   if (!clientDisconnesso(clientDesc))
599   {
600     write(clientDesc, &numeroClientLoggati, sizeof(numeroClientLoggati));
601     while (numeroClientLoggati > 0 && tmp != NULL)
602     {
603       strcpy(name, tmp->name);
604       lunghezza = strlen(tmp->name);
605       write(clientDesc, &lunghezza, sizeof(lunghezza));
606       write(clientDesc, name, lunghezza);
607       tmp = tmp->next;
608       numeroClientLoggati--;
609     }
610   }
611 }
612
613 void prepareMessageForPackDelivery(char message[], char username[], char date[])
614 {
615   strcat(message, "Pack delivered by \"");
616   strcat(message, username);
617   strcat(message, "\" at ");
618   strcat(message, date);
619   strcat(message, "\n");
620 }
621
622 void prepareMessageForLogin(char message[], char username[], char date[])
623 {
624   strcat(message, username);
625   strcat(message, "\" logged in at ");
626   strcat(message, date);
627   strcat(message, "\n");
628 }
629
630 void prepareMessageForConnection(char message[], char ipAddress[], char date[])
631 {
632   strcat(message, ipAddress);
633   strcat(message, "\" connected at ");
634   strcat(message, date);
635   strcat(message, "\n");
636 }
637
638 void putCurrentDateAndTimeInString(char dateAndTime[])
639 {
640   time_t t = time(NULL);
641   struct tm *infoTime = localtime(&t);
642   strftime(dateAndTime, 64, "%X %x", infoTime);
643 }
644
645 void *fileWriter(void *args)
646 {
647   int fDes = open("Log", O_RDWR | O_CREAT | O_APPEND, S_IWUSR | S_IRUSR);
648   if (fDes < 0)
649   {
650     perror("Error while opening log file");
651     exit(-1);
652   }
653   Args info = (Args)args;
654   char dateAndTime[64];
655   putCurrentDateAndTimeInString(dateAndTime);
656   if (logDelPacco(info->flag))
657   {
658     char message[MAX_BUF] = "";
659     prepareMessageForPackDelivery(message, info->userName, dateAndTime);
660     pthread_mutex_lock(&LogMutex);
661     write(fDes, message, strlen(message));
662     pthread_mutex_unlock(&LogMutex);
663   }
664   else if (logDelLogin(info->flag))
665   {
666     char message[MAX_BUF] = "\"";
667     prepareMessageForLogin(message, info->userName, dateAndTime);
668     pthread_mutex_lock(&LogMutex);
669     write(fDes, message, strlen(message));
```

```
    pthread_mutex_unlock(&LogMutex);
  }
  else if (logDellaConnessione(info->flag))
  {
    char message[MAX_BUF] = "\"";
    prepareMessageForConnection(message, info->userName, dateAndTime);
    pthread_mutex_lock(&LogMutex);
    write(fDes, message, strlen(message));
    pthread_mutex_unlock(&LogMutex);
  }
  close(fDes);
  free(info);
  pthread_exit(NULL);
}

int logDelPacco(int flag)
{
  if (flag == 1)
    return 1;
  return 0:
}
int logDelLogin(int flag)
{
  if (flag == 0)
    return 1;
  return 0:
}
int logDellaConnessione(int flag)
{
  if (flag == 2)
    return 1;
  return 0:
}
```

A.3 Codice sorgente boardUtility

Listato 15: Codice header utility del gioco 1

```
#include "list.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#define ROWS 10
#define COLUMNS 30
#define numberOfObstacles 35
#define numberOfPackages 15
#define TIME_LIMIT_IN_SECONDS 30
#define packageLimitNumber 4
#define MATRIX_DIMENSION sizeof(char) * ROWS *COLUMNS
#define RED_COLOR "\x1b[31m"
#define GREEN_COLOR "\x1b[32m"
#define RESET_COLOR "\x1b[0m"

struct Coord {
  int x;
  int y;
};
typedef struct Coord *Point;
void rimuoviPaccoDaArray(int posizione[2], Point packsCoords[]);
void printMenu();
int getHiddenPack(Point packsCoords[]);
void stampaIstruzioni(int i);
int isOnADeployPoint(PlayerStats giocatore, Point deployCoords[]);
int isOnCorrectDeployPoint(PlayerStats giocatore, Point deployCoords[]);
PlayerStats gestisciP(char grigliaDiGioco[ROWS][COLUMNS], PlayerStats giocatore,
                      Point deployCoords[], Point packsCoords[]);
PlayerStats gestisciA(char grigliaDiGioco[ROWS][COLUMNS],
                      char grigliaOstacoli[ROWS][COLUMNS],
                      PlayerStats giocatore, Obstacles *listaOstacoli,
                      Point deployCoords[], Point packsCoords[]);
```

```
PlayerStats gestisciD(char grigliaDiGioco[ROWS][COLUMNS],
                      char grigliaOstacoli[ROWS][COLUMNS],
                      PlayerStats giocatore, Obstacles *listaOstacoli,
                      Point deployCoords[], Point packsCoords[]);
PlayerStats gestisciS(char grigliaDiGioco[ROWS][COLUMNS],
                      char grigliaOstacoli[ROWS][COLUMNS],
                      PlayerStats giocatore, Obstacles *listaOstacoli,
                      Point deployCoords[], Point packsCoords[]);
void inizializzaGiocoSenzaPlayer(char grigliaDiGioco[ROWS][COLUMNS],
                                 char grigliaConOstacoli[ROWS][COLUMNS],
                                 Point packsCoords[]);
void inserisciPlayerNellaGrigliaInPosizioneCasuale(
    char grigliaDiGioco[ROWS][COLUMNS], char grigliaOstacoli[ROWS][COLUMNS],
    int posizione[2]);
void inizializzaGrigliaVuota(char grigliaDiGioco[ROWS][COLUMNS]);
void generaPosizioneOstacoli(char grigliaDiGioco[ROWS][COLUMNS],
                             char grigliaOstacoli[ROWS][COLUMNS]);
void riempiGrigliaConPacchiInPosizioniGenerateCasualmente(
    char grigliaDiGioco[ROWS][COLUMNS], Point packsCoords[]);
void printGrid(char grigliaDaStampare[ROWS][COLUMNS], PlayerStats stats);
void start(char grigliaDiGioco[ROWS][COLUMNS],
           char grigliaOstacoli[ROWS][COLUMNS]);
void riempiGrigliaConGliOstacoli(char grigliaDiGioco[ROWS][COLUMNS],
                                 char grigliaOstacoli[ROWS][COLUMNS]);
void generaPosizioniRaccolta(char grigliaDiGioco[ROWS][COLUMNS],
                             char grigliaOstacoli[ROWS][COLUMNS],
                             Point coord[]);
PlayerStats gestisciW(char grigliaDiGioco[ROWS][COLUMNS],
                      char grigliaOstacoli[ROWS][COLUMNS],
                      PlayerStats giocatore, Obstacles *listaOstacoli,
                      Point deployCoords[], Point packsCoords[]);
void mergeGridAndList(char grid[ROWS][COLUMNS], Obstacles top);
void scegliPosizioneRaccolta(Point coord[], int deploy[]);
int colpitoOstacolo(char grigliaOstacoli[ROWS][COLUMNS], int posizione[2]);
int colpitoPacco(Point packsCoords[], int posizione[2]);
int colpitoPlayer(char grigliaDiGioco[ROWS][COLUMNS], int posizione[2]);
int casellaVuota(char grigliaDiGioco[ROWS][COLUMNS],
                 char grigliaOstacoli[ROWS][COLUMNS], int posizione[2]);
void spostaPlayer(char griglia[ROWS][COLUMNS], int vecchiaPosizione[2],
                  int nuovaPosizione[2], Point deployCoords[],
                  Point packsCoords[]);
int arrivatoADestinazione(int posizione[2], int destinazione[2]);
int eraUnPuntoDepo(int vecchiaPosizione[2], Point depo[]);
int eraUnPacco(int vecchiaPosizione[2], Point packsCoords[]);
int isOnAPack(PlayerStats giocatore, Point packsCoords[]);
```

Listato 16: Codice sorgente utility del gioco 1

```
#include "boardUtility.h"
#include "list.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
void printMenu() {
  system("clear");
  printf("\t Cosa vuoi fare?\n");
  printf("\t1 Gioca\n");
  printf("\t2 Registrati\n");
  printf("\t3 Esci\n");
}
int colpitoOstacolo(char grigliaOstacoli[ROWS][COLUMNS], int posizione[2]) {
  if (grigliaOstacoli[posizione[0]][posizione[1]] == 'O')
    return 1;
  return 0;
}
int colpitoPacco(Point packsCoords[], int posizione[2]) {
  int i = 0;
  for (i = 0; i < numberOfPackages; i++) {
    if (packsCoords[i]->x == posizione[0] && packsCoords[i]->y == posizione[1])
      return 1;
  }
  return 0;
```

```
}
int casellaVuotaOValida(char grigliaDiGioco[ROWS][COLUMNS],
                        char grigliaOstacoli[ROWS][COLUMNS], int posizione[2]) {
  if (grigliaDiGioco[posizione[0]][posizione[1]] == '-' ||
      grigliaDiGioco[posizione[0]][posizione[1]] == '_' ||
      grigliaDiGioco[posizione[0]][posizione[1]] == '$')
    if (grigliaOstacoli[posizione[0]][posizione[1]] == '-' ||
        grigliaOstacoli[posizione[0]][posizione[1]] == '_' ||
        grigliaOstacoli[posizione[0]][posizione[1]] == '$')
      return 1;
  return 0;
}
int colpitoPlayer(char grigliaDiGioco[ROWS][COLUMNS], int posizione[2]) {
  if (grigliaDiGioco[posizione[0]][posizione[1]] == 'P')
    return 1;
  return 0;
}
int isOnCorrectDeployPoint(PlayerStats giocatore, Point deployCoords[]) {
  int i = 0;
  for (i = 0; i < numberOfPackages; i++) {
    if (giocatore->deploy[0] == deployCoords[i]->x &&
        giocatore->deploy[1] == deployCoords[i]->y) {
      if (deployCoords[i]->x == giocatore->position[0] &&
          deployCoords[i]->y == giocatore->position[1])
        return 1;
    }
  }
  return 0;
}
int getHiddenPack(Point packsCoords[]) {
  int i = 0;
  for (i = 0; i < numberOfPackages; i++) {
    if (packsCoords[i]->x == -1 && packsCoords[i]->y == -1)
      return i;
  }
  return -1;
}
int isOnAPack(PlayerStats giocatore, Point packsCoords[]) {
  int i = 0;
  for (i = 0; i < numberOfPackages; i++) {
    if (giocatore->position[0] == packsCoords[i]->x &&
        giocatore->position[1] == packsCoords[i]->y)
      return 1;
  }
  return 0;
}
int isOnADeployPoint(PlayerStats giocatore, Point deployCoords[]) {
  int i = 0;
  for (i = 0; i < numberOfPackages; i++) {
    if (giocatore->position[0] == deployCoords[i]->x &&
        giocatore->position[1] == deployCoords[i]->y)
      return 1;
  }
  return 0;
}
void inizializzaGrigliaVuota(char griglia[ROWS][COLUMNS]) {
  int i = 0, j = 0;
  for (i = 0; i < ROWS; i++) {
    for (j = 0; j < COLUMNS; j++) {
      griglia[i][j] = '-';
    }
  }
}
PlayerStats gestisciP(char grigliaDiGioco[ROWS][COLUMNS], PlayerStats giocatore,
                      Point deployCoords[], Point packsCoords[]) {
  int nuovoDeploy[2];
  if (colpitoPacco(packsCoords, giocatore->position) &&
      giocatore->hasApack == 0) {
    scegliPosizioneRaccolta(deployCoords, nuovoDeploy);
    giocatore->hasApack = 1;
    rimuoviPaccoDaArray(giocatore->position, packsCoords);
  }
  giocatore->deploy[0] = nuovoDeploy[0];
  giocatore->deploy[1] = nuovoDeploy[1];
```

```
100   return giocatore;
101 }
102 void printGrid(char grigliaDaStampare[ROWS][COLUMNS], PlayerStats stats) {
103   system("clear");
104   printf("\n\n");
105   int i = 0, j = 0;
106   for (i = 0; i < ROWS; i++) {
107     printf("\t");
108     for (j = 0; j < COLUMNS; j++) {
109       if (stats != NULL) {
110         if ((i == stats->deploy[0] && j == stats->deploy[1]) ||
111             (i == stats->position[0] && j == stats->position[1]))
112           if (grigliaDaStampare[i][j] == 'P' && stats->hasApack == 1)
113             printf(GREEN_COLOR "%c" RESET_COLOR, grigliaDaStampare[i][j]);
114           else
115             printf(RED_COLOR "%c" RESET_COLOR, grigliaDaStampare[i][j]);
116         else
117           printf("%c", grigliaDaStampare[i][j]);
118       } else
119         printf("%c", grigliaDaStampare[i][j]);
120     }
121     stampaIstruzioni(i);
122     if (i == 8)
123       printf(GREEN_COLOR "\t\t Punteggio: %d" RESET_COLOR, stats->score);
124     printf("\n");
125   }
126 }
127 void stampaIstruzioni(int i) {
128   if (i == 0)
129     printf("\t \t ISTRUZIONI ");
130   if (i == 1)
131     printf("\t Inviare 't' per il timer.");
132   if (i == 2)
133     printf("\t Inviare 'e' per uscire");
134   if (i == 3)
135     printf("\t Inviare 'p' per raccogliere un pacco");
136   if (i == 4)
137     printf("\t Inviare 'c' per consegnare il pacco");
138   if (i == 5)
139     printf("\t Inviare 'w'/'s' per andare sopra/sotto");
140   if (i == 6)
141     printf("\t Inviare 'a'/'d' per andare a dx/sx");
142   if (i == 7)
143     printf("\t Inviare 'l' per la lista degli utenti ");
144 }
145 // aggiunge alla griglia gli ostacoli visti fino ad ora dal client
146 void mergeGridAndList(char grid[ROWS][COLUMNS], Obstacles top) {
147   while (top) {
148     grid[top->x][top->y] = 'O';
149     top = top->next;
150   }
151 }
152 /* Genera la posizione degli ostacoli */
153 void generaPosizioneOstacoli(char grigliaDiGioco[ROWS][COLUMNS],
154                              char grigliaOstacoli[ROWS][COLUMNS]) {
155   int x, y, i;
156   inizializzaGrigliaVuota(grigliaOstacoli);
157   srand(time(0));
158   for (i = 0; i < numberOfObstacles; i++) {
159     x = rand() % COLUMNS;
160     y = rand() % ROWS;
161     if (grigliaDiGioco[y][x] == '-')
162       grigliaOstacoli[y][x] = 'O';
163     else
164       i--;
165   }
166 }
167 void rimuoviPaccoDaArray(int posizione[2], Point packsCoords[]) {
168   int i = 0, found = 0;
169   while (i < numberOfPackages && !found) {
170     if ((packsCoords[i])->x == posizione[0] &&
171         (packsCoords[i])->y == posizione[1]) {
172       (packsCoords[i])->x = -1;
173       (packsCoords[i])->y = -1;
```

```
174       found = 1;
175     }
176     i++;
177   }
178 }
179 // sceglie una posizione di raccolta tra quelle disponibili
180 void scegliPosizioneRaccolta(Point coord[], int deploy[]) {
181   int index = 0;
182   srand(time(NULL));
183   index = rand() % numberOfPackages;
184   deploy[0] = coord[index]->x;
185   deploy[1] = coord[index]->y;
186 }
187 /*genera posizione di raccolta di un pacco*/
188 void generaPosizioniRaccolta(char grigliaDiGioco[ROWS][COLUMNS],
189                              char grigliaOstacoli[ROWS][COLUMNS],
190                              Point coord[]) {
191   int x, y;
192   srand(time(0));
193   int i = 0;
194   for (i = 0; i < numberOfPackages; i++) {
195     coord[i] = (Point)malloc(sizeof(struct Coord));
196   }
197   i = 0;
198   for (i = 0; i < numberOfPackages; i++) {
199     x = rand() % COLUMNS;
200     y = rand() % ROWS;
201     if (grigliaDiGioco[y][x] == '-' && grigliaOstacoli[y][x] == '-') {
202       coord[i]->x = y;
203       coord[i]->y = x;
204       grigliaDiGioco[y][x] = '_';
205       grigliaOstacoli[y][x] = '_';
206     } else
207       i--;
208   }
209 }
210 /*Inserisci dei pacchi nella griglia di gioco nella posizione casuale */
211 void riempiGrigliaConPacchiInPosizioniGenerateCasualmente(
212     char grigliaDiGioco[ROWS][COLUMNS], Point packsCoords[]) {
213   int x, y, i = 0;
214   for (i = 0; i < numberOfPackages; i++) {
215     packsCoords[i] = (Point)malloc(sizeof(struct Coord));
216   }
217   srand(time(0));
218   for (i = 0; i < numberOfPackages; i++) {
219     x = rand() % COLUMNS;
220     y = rand() % ROWS;
221     if (grigliaDiGioco[y][x] == '-') {
222       grigliaDiGioco[y][x] = '$';
223       packsCoords[i]->x = y;
224       packsCoords[i]->y = x;
225     } else
226       i--;
227   }
228 }
229 /*Inserisci gli ostacoli nella griglia di gioco */
230 void riempiGrigliaConGliOstacoli(char grigliaDiGioco[ROWS][COLUMNS],
231                                  char grigliaOstacoli[ROWS][COLUMNS]) {
232   int i, j = 0;
233   for (i = 0; i < ROWS; i++) {
234     for (j = 0; j < COLUMNS; j++) {
235       if (grigliaOstacoli[i][j] == 'O')
236         grigliaDiGioco[i][j] = 'O';
237     }
238   }
239 }
240 void inserisciPlayerNellaGrigliaInPosizioneCasuale(
241     char grigliaDiGioco[ROWS][COLUMNS], char grigliaOstacoli[ROWS][COLUMNS],
242     int posizione[2]) {
243   int x, y;
244   srand(time(0));
245   printf("Inserisco player\n");
246   do {
247     x = rand() % COLUMNS;
```

```
248     y = rand() % ROWS;
249   } while (grigliaDiGioco[y][x] != '-' && grigliaOstacoli[y][x] != '-');
250   grigliaDiGioco[y][x] = 'P';
251   posizione[0] = y;
252   posizione[1] = x;
253 }
254 void inizializzaGiocoSenzaPlayer(char grigliaDiGioco[ROWS][COLUMNS],
255                                  char grigliaConOstacoli[ROWS][COLUMNS],
256                                  Point packsCoords[]) {
257   inizializzaGrigliaVuota(grigliaDiGioco);
258   riempiGrigliaConPacchiInPosizioniGenerateCasualmente(grigliaDiGioco,
259                                                        packsCoords);
260   generaPosizioneOstacoli(grigliaDiGioco, grigliaConOstacoli);
261   return;
262 }
263 void spostaPlayer(char griglia[ROWS][COLUMNS], int vecchiaPosizione[2],
264                   int nuovaPosizione[2], Point deployCoords[],
265                   Point packsCoords[]) {
266   griglia[nuovaPosizione[0]][nuovaPosizione[1]] = 'P';
267   if (eraUnPuntoDepo(vecchiaPosizione, deployCoords))
268     griglia[vecchiaPosizione[0]][vecchiaPosizione[1]] = '_';
269   else if (eraUnPacco(vecchiaPosizione, packsCoords))
270     griglia[vecchiaPosizione[0]][vecchiaPosizione[1]] = '$';
271   else
272     griglia[vecchiaPosizione[0]][vecchiaPosizione[1]] = '-';
273 }
274 int eraUnPuntoDepo(int vecchiaPosizione[2], Point depo[]) {
275   int i = 0, ret = 0;
276   while (ret == 0 && i < numberOfPackages) {
277     if ((depo[i])->y == vecchiaPosizione[1] &&
278         (depo[i])->x == vecchiaPosizione[0]) {
279       ret = 1;
280     }
281     i++;
282   }
283   return ret;
284 }
285 int eraUnPacco(int vecchiaPosizione[2], Point packsCoords[]) {
286   int i = 0, ret = 0;
287   while (ret == 0 && i < numberOfPackages) {
288     if ((packsCoords[i])->y == vecchiaPosizione[1] &&
289         (packsCoords[i])->x == vecchiaPosizione[0]) {
290       ret = 1;
291     }
292     i++;
293   }
294   return ret;
295 }
296 PlayerStats gestisciW(char grigliaDiGioco[ROWS][COLUMNS],
297                       char grigliaOstacoli[ROWS][COLUMNS],
298                       PlayerStats giocatore, Obstacles *listaOstacoli,
299                       Point deployCoords[], Point packsCoords[]) {
300   if (giocatore == NULL)
301     return NULL;
302   int nuovaPosizione[2];
303   nuovaPosizione[1] = giocatore->position[1];
304   // Aggiorna la posizione vecchia spostando il player avanti di 1
305   nuovaPosizione[0] = (giocatore->position[0]) - 1;
306   int nuovoScore = giocatore->score;
307   int nuovoDeploy[2];
308   nuovoDeploy[0] = giocatore->deploy[0];
309   nuovoDeploy[1] = giocatore->deploy[1];
310   if (nuovaPosizione[0] >= 0 && nuovaPosizione[0] < ROWS) {
311     if (casellaVuotaOValida(grigliaDiGioco, grigliaOstacoli, nuovaPosizione)) {
312       spostaPlayer(grigliaDiGioco, giocatore->position, nuovaPosizione,
313                    deployCoords, packsCoords);
314     } else if (colpitoOstacolo(grigliaOstacoli, nuovaPosizione)) {
315       *listaOstacoli =
316           addObstacle(*listaOstacoli, nuovaPosizione[0], nuovaPosizione[1]);
317       nuovaPosizione[0] = giocatore->position[0];
318       nuovaPosizione[1] = giocatore->position[1];
319     } else if (colpitoPlayer(grigliaDiGioco, nuovaPosizione)) {
320       nuovaPosizione[0] = giocatore->position[0];
321       nuovaPosizione[1] = giocatore->position[1];
```

```
322       }
323       giocatore->deploy[0] = nuovoDeploy[0];
324       giocatore->deploy[1] = nuovoDeploy[1];
325       giocatore->score = nuovoScore;
326       giocatore->position[0] = nuovaPosizione[0];
327       giocatore->position[1] = nuovaPosizione[1];
328     }
329     return giocatore;
330   }
331   PlayerStats gestisciD(char grigliaDiGioco[ROWS][COLUMNS],
332                         char grigliaOstacoli[ROWS][COLUMNS],
333                         PlayerStats giocatore, Obstacles *listaOstacoli,
334                         Point deployCoords[], Point packsCoords[]) {
335     if (giocatore == NULL) {
336       return NULL;
337     }
338     int nuovaPosizione[2];
339     nuovaPosizione[1] = giocatore->position[1] + 1;
340     nuovaPosizione[0] = giocatore->position[0];
341     int nuovoScore = giocatore->score;
342     int nuovoDeploy[2];
343     nuovoDeploy[0] = giocatore->deploy[0];
344     nuovoDeploy[1] = giocatore->deploy[1];
345     if (nuovaPosizione[1] >= 0 && nuovaPosizione[1] < COLUMNS) {
346       if (casellaVuotaOValida(grigliaDiGioco, grigliaOstacoli, nuovaPosizione)) {
347         spostaPlayer(grigliaDiGioco, giocatore->position, nuovaPosizione,
348                      deployCoords, packsCoords);
349       } else if (colpitoOstacolo(grigliaOstacoli, nuovaPosizione)) {
350         printf("Ostacolo\n");
351         *listaOstacoli =
352             addObstacle(*listaOstacoli, nuovaPosizione[0], nuovaPosizione[1]);
353         nuovaPosizione[0] = giocatore->position[0];
354         nuovaPosizione[1] = giocatore->position[1];
355       } else if (colpitoPlayer(grigliaDiGioco, nuovaPosizione)) {
356         nuovaPosizione[0] = giocatore->position[0];
357         nuovaPosizione[1] = giocatore->position[1];
358       }
359       giocatore->deploy[0] = nuovoDeploy[0];
360       giocatore->deploy[1] = nuovoDeploy[1];
361       giocatore->score = nuovoScore;
362       giocatore->position[0] = nuovaPosizione[0];
363       giocatore->position[1] = nuovaPosizione[1];
364     }
365     return giocatore;
366   }
367   PlayerStats gestisciA(char grigliaDiGioco[ROWS][COLUMNS],
368                         char grigliaOstacoli[ROWS][COLUMNS],
369                         PlayerStats giocatore, Obstacles *listaOstacoli,
370                         Point deployCoords[], Point packsCoords[]) {
371     if (giocatore == NULL)
372       return NULL;
373     int nuovaPosizione[2];
374     nuovaPosizione[0] = giocatore->position[0];
375     // Aggiorna la posizione vecchia spostando il player avanti di 1
376     nuovaPosizione[1] = (giocatore->position[1]) - 1;
377     int nuovoScore = giocatore->score;
378     int nuovoDeploy[2];
379     nuovoDeploy[0] = giocatore->deploy[0];
380     nuovoDeploy[1] = giocatore->deploy[1];
381     if (nuovaPosizione[1] >= 0 && nuovaPosizione[1] < COLUMNS) {
382       if (casellaVuotaOValida(grigliaDiGioco, grigliaOstacoli, nuovaPosizione)) {
383         printf("Casella vuota \n");
384         spostaPlayer(grigliaDiGioco, giocatore->position, nuovaPosizione,
385                      deployCoords, packsCoords);
386       } else if (colpitoOstacolo(grigliaOstacoli, nuovaPosizione)) {
387         printf("Ostacolo\n");
388         *listaOstacoli =
389             addObstacle(*listaOstacoli, nuovaPosizione[0], nuovaPosizione[1]);
390         nuovaPosizione[0] = giocatore->position[0];
391         nuovaPosizione[1] = giocatore->position[1];
392       } else if (colpitoPlayer(grigliaDiGioco, nuovaPosizione)) {
393         printf("colpito player\n");
394         nuovaPosizione[0] = giocatore->position[0];
395         nuovaPosizione[1] = giocatore->position[1];
```

```
    }
    giocatore->deploy[0] = nuovoDeploy[0];
    giocatore->deploy[1] = nuovoDeploy[1];
    giocatore->score = nuovoScore;
    giocatore->position[0] = nuovaPosizione[0];
    giocatore->position[1] = nuovaPosizione[1];
  }
  return giocatore;
}
PlayerStats gestisciS(char grigliaDiGioco[ROWS][COLUMNS],
                      char grigliaOstacoli[ROWS][COLUMNS],
                      PlayerStats giocatore, Obstacles *listaOstacoli,
                      Point deployCoords[], Point packsCoords[]) {
  if (giocatore == NULL) {
    return NULL;
  }
  // crea le nuove statistiche
  int nuovaPosizione[2];
  nuovaPosizione[1] = giocatore->position[1];
  nuovaPosizione[0] = (giocatore->position[0]) + 1;
  int nuovoScore = giocatore->score;
  int nuovoDeploy[2];
  nuovoDeploy[0] = giocatore->deploy[0];
  nuovoDeploy[1] = giocatore->deploy[1];
  // controlla che le nuove statistiche siano corrette
  if (nuovaPosizione[0] >= 0 && nuovaPosizione[0] < ROWS) {
    if (casellaVuotaOValida(grigliaDiGioco, grigliaOstacoli, nuovaPosizione)) {
      spostaPlayer(grigliaDiGioco, giocatore->position, nuovaPosizione,
                   deployCoords, packsCoords);
    } else if (colpitoOstacolo(grigliaOstacoli, nuovaPosizione)) {
      printf("Ostacolo\n");
      *listaOstacoli =
          addObstacle(*listaOstacoli, nuovaPosizione[0], nuovaPosizione[1]);
      nuovaPosizione[0] = giocatore->position[0];
      nuovaPosizione[1] = giocatore->position[1];
    } else if (colpitoPlayer(grigliaDiGioco, nuovaPosizione)) {
      nuovaPosizione[0] = giocatore->position[0];
      nuovaPosizione[1] = giocatore->position[1];
    }
    giocatore->deploy[0] = nuovoDeploy[0];
    giocatore->deploy[1] = nuovoDeploy[1];
    giocatore->score = nuovoScore;
    giocatore->position[0] = nuovaPosizione[0];
    giocatore->position[1] = nuovaPosizione[1];
  }
  return giocatore;
}
int arrivatoADestinazione(int posizione[2], int destinazione[2]) {
  if (posizione[0] == destinazione[0] && posizione[1] == destinazione[1])
    return 1;
  return 0;
}
```

A.4 Codice sorgente list

Listato 17: Codice header utility del gioco 2

```
#ifndef DEF_LIST_H
#define DEF_LIST_H
#define MAX_BUF 200
#include <pthread.h>
// players
struct TList {
  char *name;
  struct TList *next;
  int sockDes;
} TList;

struct Data {
  int deploy[2];
  int score;
```

```
  int position[2];
  int hasApack;
} Data;

// Obstacles
struct TList2 {
  int x;
  int y;
  struct TList2 *next;
} TList2;

typedef struct Data *PlayerStats;
typedef struct TList *Players;
typedef struct TList2 *Obstacles;

// calcola e restituisce il numero di player commessi dalla lista L
int dimensioneLista(Players L);

// inizializza un giocatore
Players initPlayerNode(char *name, int sockDes);

// Crea un nodo di Stats da mandare a un client
PlayerStats initStats(int deploy[], int score, int position[], int flag);

// Inizializza un nuovo nodo
Players initNodeList(char *name, int sockDes);

// Aggiunge un nodo in testa alla lista
// La funzione ritorna sempre la testa della lista
Players addPlayer(Players L, char *name, int sockDes);

// Rimuove solo un occorrenza di un nodo con il socket Descriptor
//  specificato dalla lista
// La funzione ritorna sempre la testa della lista
Players removePlayer(Players L, int sockDes);

// Dealloca la lista interamente
void freePlayers(Players L);

// Stampa la lista
void printPlayers(Players L);

// Controlla se un utente á giá loggato
int isAlreadyLogged(Players L, char *name);

// Dealloca la lista degli ostacoli
void freeObstacles(Obstacles L);

// Stampa la lista degli ostacoli
void printObstacles(Obstacles L);

// Aggiunge un ostacolo in testa
Obstacles addObstacle(Obstacles L, int x, int y);

// Inizializza un nuovo nodo ostacolo
Obstacles initObstacleNode(int x, int y);
#endif
```

Listato 18: Codice sorgente utility del gioco 2

```
#include "list.h"
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

Players initPlayerNode(char *name, int sockDes) {
  Players L = (Players)malloc(sizeof(struct TList));
  L->name = (char *)malloc(MAX_BUF);
  strcpy(L->name, name);
  L->sockDes = sockDes;
  L->next = NULL;
  return L;
```

```
}
PlayerStats initStats(int deploy[], int score, int position[], int flag) {
  PlayerStats L = (PlayerStats)malloc(sizeof(struct Data));
  L->deploy[0] = deploy[0];
  L->deploy[1] = deploy[1];
  L->score = score;
  L->hasApack = flag;
  L->position[0] = position[0];
  L->position[1] = position[1];
  return L;
}
Obstacles initObstacleNode(int x, int y) {
  Obstacles L = (Obstacles)malloc(sizeof(struct TList2));
  L->x = x;
  L->y = y;
  L->next = NULL;
  return L;
}
Obstacles addObstacle(Obstacles L, int x, int y) {
  Obstacles tmp = initObstacleNode(x, y);
  if (L != NULL)
    tmp->next = L;
  return tmp;
}
int dimensioneLista(Players L) {
  int size = 0;
  Players tmp = L;
  while (tmp != NULL) {
    size++;
    tmp = tmp->next;
  }
  return size;
}
int isAlreadyLogged(Players L, char *name) {
  int ret = 0;
  if (L != NULL) {
    if (strcmp(L->name, name) == 0)
      return 1;
    ret = isAlreadyLogged(L->next, name);
  }
  return ret;
}
Players addPlayer(Players L, char *name, int sockDes) {
  Players tmp = initPlayerNode(name, sockDes);
  if (L != NULL)
    tmp->next = L;
  return tmp;
}
Players removePlayer(Players L, int sockDes) {
  if (L != NULL) {
    if (L->sockDes == sockDes) {
      Players tmp = L->next;
      free(L);
      return tmp;
    }
    L->next = removePlayer(L->next, sockDes);
  }
  return L;
}
void freePlayers(Players L) {
  if (L != NULL) {
    freePlayers(L->next);
    free(L);
  }
}
void freeObstacles(Obstacles L) {
  if (L != NULL) {
    freeObstacles(L->next);
    free(L);
  }
}
void printPlayers(Players L) {
  if (L != NULL) {
    printf("%s ->", L->name);
```

```
88      printPlayers(L->next);
89    }
90    printf("\n");
91  }
92  void printObstacles(Obstacles L) {
93    if (L != NULL) {
94      printf("X:%d Y:%d ->", L->x, L->y);
95      printObstacles(L->next);
96    }
97  }
```

A.5 Codice sorgente parser

Listato 19: Codice header utility del gioco 3

```
1  int appendPlayer(char *name, char *pwd, char *file);
2  int isRegistered(char *name, char *file);
3  int openFileRDWRAPP(char *file);
4  int validateLogin(char *name, char *pwd, char *file);
5  int openFileRDON(char *file);
6  void premiEnterPerContinuare();
```

Listato 20: Codice sorgente utility del gioco 3

```
1  #include "parser.h"
2  #include <errno.h>
3  #include <fcntl.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <sys/stat.h>
8  #include <sys/types.h>
9  #include <unistd.h>
10 #define MAX_BUF 200
11 int openFileRDWRAPP(char *file) {
12   int fileDes = open(file, O_RDWR | O_CREAT | O_APPEND, S_IRUSR | S_IWUSR);
13   if (fileDes < 0)
14     perror("Errore apertura file\n"), exit(-1);
15   return fileDes;
16 }
17 int openFileRDON(char *file) {
18   int fileDes = open(file, O_RDONLY);
19   if (fileDes < 0)
20     perror("Errore apertura file\n"), exit(-1);
21   return fileDes;
22 }
23 int appendPlayer(char *name, char *pwd, char *file) {
24   if (isRegistered(name, file))
25     return 0;
26   int fileDes = openFileRDWRAPP(file);
27   write(fileDes, name, strlen(name));
28   write(fileDes, " ", 1);
29   write(fileDes, pwd, strlen(pwd));
30   write(fileDes, "\n", 1);
31   close(fileDes);
32   return 1;
33 }
34 int isRegistered(char *name, char *file) {
35   char command[MAX_BUF] = "cat  ";
36   strcat(command, file);
37   char toApp[] = " |cut -d\" \" -f1|grep \"^";
38   strcat(command, toApp);
39   strcat(command, name);
40   char toApp2[] = "$\">tmp";
41   strcat(command, toApp2);
42   int ret = 0;
43   system(command);
44   int fileDes = openFileRDON("tmp");
45   struct stat info;
```

```
46    fstat(fileDes, &info);
47    if ((int)info.st_size > 0)
48      ret = 1;
49    close(fileDes);
50    system("rm tmp");
51    return ret;
52  }
53  int validateLogin(char *name, char *pwd, char *file) {
54    if (!isRegistered(name, file))
55      return 0;
56    char command[MAX_BUF] = "cat  ";
57    strcat(command, file);
58    char toApp[] = " |grep \"^";
59    strcat(command, toApp);
60    strcat(command, name);
61    strcat(command, " ");
62    strcat(command, pwd);
63    char toApp2[] = "$\">tmp";
64    strcat(command, toApp2);
65    int ret = 0;
66    system(command);
67    int fileDes = openFileRDON("tmp");
68    struct stat info;
69    fstat(fileDes, &info);
70    if ((int)info.st_size > 0)
71      ret = 1;
72    close(fileDes);
73    system("rm tmp");
74    return ret;
75  }
76  void premiEnterPerContinuare() {
77    fflush(stdin);
78    printf("Premi Invio per continuare\n");
79    char c = getchar();
80  }
```

Listati