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 protoccoli 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

```
  pthread_t tid;
  int clientDesc;
  int *puntClientDesc;
  while (1 == 1)
  {
    if (listen(socketDesc, 10) < 0)
      perror("Impossibile mettersi in ascolto"), exit(-1);
    printf("In ascolto..\n");
```

Listato 2: Configurazione socket del server

```
    write(clientDesc, &risposta, sizeof(char));
    printf("Impossibile registrare utente, riprovare\n");
  }
  else
  {
    risposta = 'y';
    write(clientDesc, &risposta, sizeof(char));
    printf("Utente registrato con successo\n");
  }
  return ret;
}
void quitServer()
{
  printf("Chiusura server in corso..\n");
  close(socketDesc);
  exit(-1);
```

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  Point deployCoords[numberOfPackages];
2  Point packsCoords[numberOfPackages];
3  pthread_mutex_t LogMutex = PTHREAD_MUTEX_INITIALIZER;
4  pthread_mutex_t RegMutex = PTHREAD_MUTEX_INITIALIZER;
5  pthread_mutex_t PlayerMutex = PTHREAD_MUTEX_INITIALIZER;
6  pthread_mutex_t MatrixMutex = PTHREAD_MUTEX_INITIALIZER;
7
8  int main(int argc, char **argv)
9  {
10   if (argc != 2)
11   {
12     printf("Wrong parameters number(Usage: ./server usersFile)\n");
13     exit(-1);
14   }
15   else if (strcmp(argv[1], "Log") == 0)
16   {
17     printf("Cannot use the Log file as a UserList \n");
18     exit(-1);
19   }
20   users = argv[1];
21   struct sockaddr_in mio_indirizzo = configuraIndirizzo();
22   configuraSocket(mio_indirizzo);
23   signal(SIGPIPE, clientCrashHandler);
24   signal(SIGINT, quitServer);
25   signal(SIGHUP, quitServer);
26   startTimer();
27   inizializzaGiocoSenzaPlayer(grigliaDiGiocoConPacchiSenzaOstacoli,
28                               grigliaOstacoliSenzaPacchi, packsCoords);
29   generaPosizioniRaccolta(grigliaDiGiocoConPacchiSenzaOstacoli,
30                           grigliaOstacoliSenzaPacchi, deployCoords);
31   startListening();
32   return 0;
33 }
34 void startListening()
35 {
```

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   printf("Inizio procedura generazione mappa\n");
2   pthread_create(&tidGeneratoreMappa, NULL, threadGenerazioneMappa, NULL);
3 }
4 void startTimer()
5 {
6   printf("Thread timer avviato\n");
7   pthread_create(&tidTimer, NULL, timer, NULL);
8 }
9 int tryLogin(int clientDesc, char name[])
10 {
11   char *userName = (char *)calloc(MAX_BUF, 1);
12   char *password = (char *)calloc(MAX_BUF, 1);
13   int dimName, dimPwd;
14   read(clientDesc, &dimName, sizeof(int));
15   read(clientDesc, &dimPwd, sizeof(int));
16   read(clientDesc, userName, dimName);
17   read(clientDesc, password, dimPwd);
18   int ret = 0;
19   pthread_mutex_lock(&PlayerMutex);
20   if (validateLogin(userName, password, users) &&
21       !isAlreadyLogged(onLineUsers, userName))
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

```
    ret = 1;
    numeroClientLoggati++;
    write(clientDesc, "y", 1);
    strcpy(name, userName);
    Args args = (Args)malloc(sizeof(struct argsToSend));
    args->userName = (char *)calloc(MAX_BUF, 1);
    strcpy(args->userName, name);
    args->flag = 0;
    pthread_t tid;
    pthread_create(&tid, NULL, fileWriter, (void *)args);
    printf("Nuovo client loggato, client loggati : %d\n", numeroClientLoggati);
    onLineUsers = addPlayer(onLineUsers, userName, clientDesc);
    pthread_mutex_unlock(&PlayerMutex);
    printPlayers(onLineUsers);
    printf("\n");
  }
  else
  {
    write(clientDesc, "n", 1);
  }
  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)
  {
```

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   close(clientDescriptor);
2 }
3 int clientDisconnesso(int clientSocket)
4 {
5   char msg[1] = {'u'}; // UP?
6   if (write(clientSocket, msg, sizeof(msg)) < 0)
7     return 1;
8   if (read(clientSocket, msg, sizeof(char)) < 0)
9     return 1;
10  else
11    return 0;
12 }
13 int registraClient(int clientDesc)
14 {
15  char *userName = (char *)calloc(MAX_BUF, 1);
16  char *password = (char *)calloc(MAX_BUF, 1);
17  int dimName, dimPwd;
18  read(clientDesc, &dimName, sizeof(int));
19  read(clientDesc, &dimPwd, sizeof(int));
20  read(clientDesc, userName, dimName);
21  read(clientDesc, password, dimPwd);
22  pthread_mutex_lock(&RegMutex);
23  int ret = appendPlayer(userName, password, users);
24  pthread_mutex_unlock(&RegMutex);
25  char risposta;
26  if (!ret)
27  {
28    risposta = '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

```
1   int flag = 1;
2   // TODO eliminare la lista degli ostacoli dell'utente
3   if (onLineUsers != NULL)
4   {
5     Players prec = onLineUsers;
6     Players top = prec->next;
7     while (top != NULL && flag)
8     {
9       if (write(top->sockDes, msg, sizeof(msg)) < 0)
10      {
11        socketClientCrashato = top->sockDes;
12        printPlayers(onLineUsers);
```

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

```
  strcat(message, "\" logged in at ");
  strcat(message, date);
  strcat(message, "\n");
}

void prepareMessageForConnection(char message[], char ipAddress[], char date[])
{
  strcat(message, ipAddress);
  strcat(message, "\" connected at ");
  strcat(message, date);
  strcat(message, "\n");
}

void putCurrentDateAndTimeInString(char dateAndTime[])
{
  time_t t = time(NULL);
  struct tm *infoTime = localtime(&t);
  strftime(dateAndTime, 64, "%X %x", infoTime);
}

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);
  }
```

Analizzando il codice si può notare l'uso open per aprire in append o, in caso di assenza del file, di creare il file di log ed i vari write per scrivere sul suddetto file; possiamo anche notare come la sezione critica, ovvero la scrittura su uno stesso file da parte di più thread, è gestita tramite un mutex.

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

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

```
73    if ((socketDesc = socket(PF_INET, SOCK_STREAM, 0)) < 0)
74      perror("Impossibile creare socket"), exit(-1);
75    else
76      printf("Socket creato\n");
77    if (connect(socketDesc, (struct sockaddr *)&mio_indirizzo,
78                sizeof(mio_indirizzo)) < 0)
79      perror("Impossibile connettersi"), exit(-1);
80    else
81      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];
```

```
  char password[20];
  system("clear");
  printf("Inserisci nome utente(MAX 20 caratteri): ");
  scanf("%s", username);
  printf("\nInserisci password(MAX 20 caratteri):");
  scanf("%s", password);
  int dimUname = strlen(username), dimPwd = strlen(password);
  if (write(socketDesc, &dimUname, sizeof(dimUname)) < 0)
    return 0;
  if (write(socketDesc, &dimPwd, sizeof(dimPwd)) < 0)
    return 0;
  if (write(socketDesc, username, dimUname) < 0)
    return 0;
  if (write(socketDesc, password, dimPwd) < 0)
    return 0;
  char validate;
  int ret;
  read(socketDesc, &validate, sizeof(char));
  if (validate == 'y') {
    ret = 1;
    printf("Registrato con successo\n");
  }
  if (validate == 'n') {
    ret = 0;
    printf("Registrazione fallita\n");
  }
  sleep(1);
  return ret;
}
char *ipResolver(char **argv) {
  char *ipAddress;
  struct hostent *hp;
  hp = gethostbyname(argv[1]);
  if (!hp) {
    perror("Impossibile risolvere l'indirizzo ip\n");
    sleep(1);
    exit(-1);
  }
  printf("Address:\t%s\n", inet_ntoa(*(struct in_addr *)hp->h_addr_list[0]));
  return inet_ntoa(*(struct in_addr *)hp->h_addr_list[0]);
}
void clientCrashHandler() {
  int msg = 3;
  int rec = 0;
  printf("\nChiusura client...\n");
  do {
    write(socketDesc, &msg, sizeof(int));
    read(socketDesc, &rec, sizeof(int));
  } while (rec == 0);
  close(socketDesc);
  signal(SIGINT, SIG_IGN);
  signal(SIGQUIT, SIG_IGN);
  signal(SIGTERM, SIG_IGN);
  signal(SIGTSTP, SIG_IGN);
  exit(0);
}
void serverCrashHandler() {
  system("clear");
  printf("Il server á stato spento o á irraggiungibile\n");
  close(socketDesc);
  signal(SIGPIPE, SIG_IGN);
  premiEnterPerContinuare();
  exit(0);
}
char getUserInput() {
  char c;
  c = getchar();
  int daIgnorare;
  while ((daIgnorare = getchar()) != '\n' && daIgnorare != EOF) {
  }
  return c;
}
```

A.2 Codice sorgente del server

Listato 14: Codice sorgente del server

```
#include "boardUtility.h"
#include "list.h"
#include "parser.h"
#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[]);
void spostaPlayer(char griglia[ROWS][COLUMNS], int vecchiaPosizione[2],
                  int nuovaPosizione[2], Point deployCoords[],
                  Point packsCoords[]);
int valoreTimerValido();
PlayerStats gestisciW(char grigliaDiGioco[ROWS][COLUMNS],
                      char grigliaOstacoli[ROWS][COLUMNS],
                      PlayerStats giocatore, Obstacles *listaOstacoli,
                      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[]);
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);
```

```
70  void *timer(void *args);
71  void *gestisci(void *descriptor);
72  void quitServer();
73  void clientCrashHandler(int signalNum);
74  void startTimer();
75  void configuraSocket(struct sockaddr_in mio_indirizzo);
76  struct sockaddr_in configuraIndirizzo();
77  void startListening();
78  int clientDisconnesso(int clientSocket);
79  void play(int clientDesc, char name[]);
80  void prepareMessageForPackDelivery(char message[], char username[], char date[]);
81  int logDelPacco(int flag);
82  int logDelLogin(int flag);
83  int logDellaConnessione(int flag);
84  char grigliaDiGiocoConPacchiSenzaOstacoli[ROWS][COLUMNS];
85  char grigliaOstacoliSenzaPacchi[ROWS][COLUMNS];
86  int numeroClientLoggati = 0;
87  int playerGenerati = 0;
88  int timerCount = TIME_LIMIT_IN_SECONDS;
89  int turno = 0;
90  pthread_t tidTimer;
91  pthread_t tidGeneratoreMappa;
92  int socketDesc;
93  Players onLineUsers = NULL;
94  char *users;
95  int scoreMassimo = 0;
96  int numMosse = 0;
97  Point deployCoords[numberOfPackages];
98  Point packsCoords[numberOfPackages];
99  pthread_mutex_t LogMutex = PTHREAD_MUTEX_INITIALIZER;
100 pthread_mutex_t RegMutex = PTHREAD_MUTEX_INITIALIZER;
101 pthread_mutex_t PlayerMutex = PTHREAD_MUTEX_INITIALIZER;
102 pthread_mutex_t MatrixMutex = PTHREAD_MUTEX_INITIALIZER;
103
104 int main(int argc, char **argv)
105 {
106   if (argc != 2)
107   {
108     printf("Wrong parameters number(Usage: ./server usersFile)\n");
109     exit(-1);
110   }
111   else if (strcmp(argv[1], "Log") == 0)
112   {
113     printf("Cannot use the Log file as a UserList \n");
114     exit(-1);
115   }
116   users = argv[1];
117   struct sockaddr_in mio_indirizzo = configuraIndirizzo();
118   configuraSocket(mio_indirizzo);
119   signal(SIGPIPE, clientCrashHandler);
120   signal(SIGINT, quitServer);
121   signal(SIGHUP, quitServer);
122   startTimer();
123   inizializzaGiocoSenzaPlayer(grigliaDiGiocoConPacchiSenzaOstacoli,
124                               grigliaOstacoliSenzaPacchi, packsCoords);
125   generaPosizioniRaccolta(grigliaDiGiocoConPacchiSenzaOstacoli,
126                           grigliaOstacoliSenzaPacchi, deployCoords);
127   startListening();
128   return 0;
129 }
130 void startListening()
131 {
132   pthread_t tid;
133   int clientDesc;
134   int *puntClientDesc;
135   while (1 == 1)
136   {
137     if (listen(socketDesc, 10) < 0)
138       perror("Impossibile mettersi in ascolto"), exit(-1);
139     printf("In ascolto..\n");
140     if ((clientDesc = accept(socketDesc, NULL, NULL)) < 0)
141     {
142       perror("Impossibile effettuare connessione\n");
143       exit(-1);
```

```
144       }
145       printf("Nuovo client connesso\n");
146       struct sockaddr_in address;
147       socklen_t size = sizeof(struct sockaddr_in);
148       if (getpeername(clientDesc, (struct sockaddr *)&address, &size) < 0)
149       {
150         perror("Impossibile ottenere l'indirizzo del client");
151         exit(-1);
152       }
153       char clientAddr[20];
154       strcpy(clientAddr, inet_ntoa(address.sin_addr));
155       Args args = (Args)malloc(sizeof(struct argsToSend));
156       args->userName = (char *)calloc(MAX_BUF, 1);
157       strcpy(args->userName, clientAddr);
158       args->flag = 2;
159       pthread_t tid;
160       pthread_create(&tid, NULL, fileWriter, (void *)args);
161
162       puntClientDesc = (int *)malloc(sizeof(int));
163       *puntClientDesc = clientDesc;
164       pthread_create(&tid, NULL, gestisci, (void *)puntClientDesc);
165     }
166     close(clientDesc);
167     quitServer();
168   }
169   struct sockaddr_in configuraIndirizzo()
170   {
171     struct sockaddr_in mio_indirizzo;
172     mio_indirizzo.sin_family = AF_INET;
173     mio_indirizzo.sin_port = htons(5200);
174     mio_indirizzo.sin_addr.s_addr = htonl(INADDR_ANY);
175     printf("Indirizzo socket configurato\n");
176     return mio_indirizzo;
177   }
178   void startProceduraGenrazioneMappa()
179   {
180     printf("Inizio procedura generazione mappa\n");
181     pthread_create(&tidGeneratoreMappa, NULL, threadGenerazioneMappa, NULL);
182   }
183   void startTimer()
184   {
185     printf("Thread timer avviato\n");
186     pthread_create(&tidTimer, NULL, timer, NULL);
187   }
188   int tryLogin(int clientDesc, char name[])
189   {
190     char *userName = (char *)calloc(MAX_BUF, 1);
191     char *password = (char *)calloc(MAX_BUF, 1);
192     int dimName, dimPwd;
193     read(clientDesc, &dimName, sizeof(int));
194     read(clientDesc, &dimPwd, sizeof(int));
195     read(clientDesc, userName, dimName);
196     read(clientDesc, password, dimPwd);
197     int ret = 0;
198     pthread_mutex_lock(&PlayerMutex);
199     if (validateLogin(userName, password, users) &&
200         !isAlreadyLogged(onLineUsers, userName))
201     {
202       ret = 1;
203       numeroClientLoggati++;
204       write(clientDesc, "y", 1);
205       strcpy(name, userName);
206       Args args = (Args)malloc(sizeof(struct argsToSend));
207       args->userName = (char *)calloc(MAX_BUF, 1);
208       strcpy(args->userName, name);
209       args->flag = 0;
210       pthread_t tid;
211       pthread_create(&tid, NULL, fileWriter, (void *)args);
212       printf("Nuovo client loggato, client loggati : %d\n", numeroClientLoggati);
213       onLineUsers = addPlayer(onLineUsers, userName, clientDesc);
214       pthread_mutex_unlock(&PlayerMutex);
215       printPlayers(onLineUsers);
216       printf("\n");
217     }
```

```
218   else
219   {
220     write(clientDesc, "n", 1);
221   }
222   return ret;
223 }
224 void *gestisci(void *descriptor)
225 {
226   int bufferReceive[2] = {1};
227   int client_sd = *(int *)descriptor;
228   int continua = 1;
229   char name[MAX_BUF];
230   while (continua)
231   {
232     read(client_sd, bufferReceive, sizeof(bufferReceive));
233     if (bufferReceive[0] == 2)
234       registraClient(client_sd);
235     else if (bufferReceive[0] == 1)
236       if (tryLogin(client_sd, name))
237       {
238         play(client_sd, name);
239         continua = 0;
240       }
241       else if (bufferReceive[0] == 3)
242         disconnettiClient(client_sd);
243       else
244       {
245         printf("Input invalido, uscita...\n");
246         disconnettiClient(client_sd);
247       }
248   }
249   pthread_exit(0);
250 }
251 void play(int clientDesc, char name[])
252 {
253   int true = 1;
254   int turnoFinito = 0;
255   int turnoGiocatore = turno;
256   int posizione[2];
257   int destinazione[2] = {-1, -1};
258   PlayerStats giocatore = initStats(destinazione, 0, posizione, 0);
259   Obstacles listaOstacoli = NULL;
260   char inputFromClient;
261   if (timer != 0)
262   {
263     inserisciPlayerNellaGrigliaInPosizioneCasuale(
264         grigliaDiGiocoConPacchiSenzaOstacoli, grigliaOstacoliSenzaPacchi,
265         giocatore->position);
266     playerGenerati++;
267   }
268   while (true)
269   {
270     if (clientDisconnesso(clientDesc))
271     {
272       freeObstacles(listaOstacoli);
273       disconnettiClient(clientDesc);
274       return;
275     }
276     char grigliaTmp[ROWS][COLUMNS];
277     clonaGriglia(grigliaTmp, grigliaDiGiocoConPacchiSenzaOstacoli);
278     mergeGridAndList(grigliaTmp, listaOstacoli);
279     // invia la griglia
280     write(clientDesc, grigliaTmp, sizeof(grigliaTmp));
281     // invia la struttura del player
282     write(clientDesc, giocatore->deploy, sizeof(giocatore->deploy));
283     write(clientDesc, giocatore->position, sizeof(giocatore->position));
284     write(clientDesc, &giocatore->score, sizeof(giocatore->score));
285     write(clientDesc, &giocatore->hasApack, sizeof(giocatore->hasApack));
286     // legge l'input
287     if (read(clientDesc, &inputFromClient, sizeof(char)) > 0)
288       numMosse++;
289     if (inputFromClient == 'e' || inputFromClient == 'E')
290     {
291       freeObstacles(listaOstacoli);
```

```
292       listaOstacoli = NULL;
293       disconnettiClient(clientDesc);
294     }
295     else if (inputFromClient == 't' || inputFromClient == 'T')
296     {
297       write(clientDesc, &turnoFinito, sizeof(int));
298       sendTimerValue(clientDesc);
299     }
300     else if (inputFromClient == 'l' || inputFromClient == 'L')
301     {
302       write(clientDesc, &turnoFinito, sizeof(int));
303       sendPlayerList(clientDesc);
304     }
305     else if (turnoGiocatore == turno)
306     {
307       write(clientDesc, &turnoFinito, sizeof(int));
308       giocatore =
309           gestisciInput(grigliaDiGiocoConPacchiSenzaOstacoli,
310                         grigliaOstacoliSenzaPacchi, inputFromClient, giocatore,
311                         &listaOstacoli, deployCoords, packsCoords, name);
312     }
313     else
314     {
315       turnoFinito = 1;
316       write(clientDesc, &turnoFinito, sizeof(int));
317       freeObstacles(listaOstacoli);
318       listaOstacoli = NULL;
319       inserisciPlayerNellaGrigliaInPosizioneCasuale(
320           grigliaDiGiocoConPacchiSenzaOstacoli, grigliaOstacoliSenzaPacchi,
321           giocatore->position);
322       giocatore->score = 0;
323       giocatore->hasApack = 0;
324       giocatore->deploy[0] = -1;
325       giocatore->deploy[1] = -1;
326       turnoGiocatore = turno;
327       turnoFinito = 0;
328       playerGenerati++;
329     }
330   }
331 }
332 void sendTimerValue(int clientDesc)
333 {
334   if (!clientDisconnesso(clientDesc))
335     write(clientDesc, &timerCount, sizeof(timerCount));
336 }
337 void clonaGriglia(char destinazione[ROWS][COLUMNS],
338                   char source[ROWS][COLUMNS])
339 {
340   int i = 0, j = 0;
341   for (i = 0; i < ROWS; i++)
342   {
343     for (j = 0; j < COLUMNS; j++)
344     {
345       destinazione[i][j] = source[i][j];
346     }
347   }
348 }
349 void clientCrashHandler(int signalNum)
350 {
351   char msg[0];
352   int socketClientCrashato;
353   int flag = 1;
354   // TODO eliminare la lista degli ostacoli dell'utente
355   if (onLineUsers != NULL)
356   {
357     Players prec = onLineUsers;
358     Players top = prec->next;
359     while (top != NULL && flag)
360     {
361       if (write(top->sockDes, msg, sizeof(msg)) < 0)
362       {
363         socketClientCrashato = top->sockDes;
364         printPlayers(onLineUsers);
365         disconnettiClient(socketClientCrashato);
```

```
366         flag = 0;
367       }
368       top = top->next;
369     }
370   }
371   signal(SIGPIPE, SIG_IGN);
372 }
373 void disconnettiClient(int clientDescriptor)
374 {
375   if (numeroClientLoggati > 0)
376     numeroClientLoggati--;
377   pthread_mutex_lock(&PlayerMutex);
378   onLineUsers = removePlayer(onLineUsers, clientDescriptor);
379   pthread_mutex_unlock(&PlayerMutex);
380   printPlayers(onLineUsers);
381   int msg = 1;
382   printf("Client disconnesso (client attualmente loggati: %d)\n",
383          numeroClientLoggati);
384   write(clientDescriptor, &msg, sizeof(msg));
385   close(clientDescriptor);
386 }
387 int clientDisconnesso(int clientSocket)
388 {
389   char msg[1] = {'u'}; // UP?
390   if (write(clientSocket, msg, sizeof(msg)) < 0)
391     return 1;
392   if (read(clientSocket, msg, sizeof(char)) < 0)
393     return 1;
394   else
395     return 0;
396 }
397 int registraClient(int clientDesc)
398 {
399   char *userName = (char *)calloc(MAX_BUF, 1);
400   char *password = (char *)calloc(MAX_BUF, 1);
401   int dimName, dimPwd;
402   read(clientDesc, &dimName, sizeof(int));
403   read(clientDesc, &dimPwd, sizeof(int));
404   read(clientDesc, userName, dimName);
405   read(clientDesc, password, dimPwd);
406   pthread_mutex_lock(&RegMutex);
407   int ret = appendPlayer(userName, password, users);
408   pthread_mutex_unlock(&RegMutex);
409   char risposta;
410   if (!ret)
411   {
412     risposta = 'n';
413     write(clientDesc, &risposta, sizeof(char));
414     printf("Impossibile registrare utente, riprovare\n");
415   }
416   else
417   {
418     risposta = 'y';
419     write(clientDesc, &risposta, sizeof(char));
420     printf("Utente registrato con successo\n");
421   }
422   return ret;
423 }
424 void quitServer()
425 {
426   printf("Chiusura server in corso..\n");
427   close(socketDesc);
428   exit(-1);
429 }
430 void *threadGenerazioneMappa(void *args)
431 {
432   fprintf(stdout, "Rigenerazione mappa\n");
433   inizializzaGrigliaVuota(grigliaDiGiocoConPacchiSenzaOstacoli);
434   generaPosizioniRaccolta(grigliaDiGiocoConPacchiSenzaOstacoli,
435                           grigliaOstacoliSenzaPacchi, deployCoords);
436   riempiGrigliaConPacchiInPosizioniGenerateCasualmente(
437       grigliaDiGiocoConPacchiSenzaOstacoli, packsCoords);
438   generaPosizioneOstacoli(grigliaDiGiocoConPacchiSenzaOstacoli,
439                           grigliaOstacoliSenzaPacchi);
```

```
440   printf("Mappa generata\n");
441   pthread_exit(NULL);
442 }
443 int almenoUnaMossaFatta()
444 {
445   if (numMosse > 0)
446     return 1;
447   return 0;
448 }
449 int almenoUnClientConnesso()
450 {
451   if (numeroClientLoggati > 0)
452     return 1;
453   return 0;
454 }
455 int valoreTimerValido()
456 {
457   if (timerCount > 0 && timerCount <= TIME_LIMIT_IN_SECONDS)
458     return 1;
459   return 0;
460 }
461 int almenoUnPlayerGenerato()
462 {
463   if (playerGenerati > 0)
464     return 1;
465   return 0;
466 }
467 void *timer(void *args)
468 {
469   int cambiato = 1;
470   while (1)
471   {
472     if (almenoUnClientConnesso() && valoreTimerValido() &&
473         almenoUnPlayerGenerato() && almenoUnaMossaFatta())
474     {
475       cambiato = 1;
476       sleep(1);
477       timerCount--;
478       fprintf(stdout, "Time left: %d\n", timerCount);
479     }
480     else if (numeroClientLoggati == 0)
481     {
482       timerCount = TIME_LIMIT_IN_SECONDS;
483       if (cambiato)
484       {
485         fprintf(stdout, "Time left: %d\n", timerCount);
486         cambiato = 0;
487       }
488     }
489     if (timerCount == 0 || scoreMassimo == packageLimitNumber)
490     {
491       playerGenerati = 0;
492       numMosse = 0;
493       printf("Reset timer e generazione nuova mappa..\n");
494       startProceduraGenrazioneMappa();
495       pthread_join(tidGeneratoreMappa, NULL);
496       turno++;
497       timerCount = TIME_LIMIT_IN_SECONDS;
498     }
499   }
500 }
501
502 void configuraSocket(struct sockaddr_in mio_indirizzo)
503 {
504   if ((socketDesc = socket(PF_INET, SOCK_STREAM, 0)) < 0)
505   {
506     perror("Impossibile creare socket");
507     exit(-1);
508   }
509   if (setsockopt(socketDesc, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) <
510       0)
511     perror("Impossibile impostare il riutilizzo dell'indirizzo ip e della "
512            "porta\n");
513   if ((bind(socketDesc, (struct sockaddr *)&mio_indirizzo,
```

```
514             sizeof(mio_indirizzo))) < 0)
515   {
516     perror("Impossibile effettuare bind");
517     exit(-1);
518   }
519 }
520
521 PlayerStats gestisciInput(char grigliaDiGioco[ROWS][COLUMNS],
522                           char grigliaOstacoli[ROWS][COLUMNS], char input,
523                           PlayerStats giocatore, Obstacles *listaOstacoli,
524                           Point deployCoords[], Point packsCoords[],
525                           char name[])
526 {
527   if (giocatore == NULL)
528   {
529     return NULL;
530   }
531   if (input == 'w' || input == 'W')
532   {
533     giocatore = gestisciW(grigliaDiGioco, grigliaOstacoli, giocatore,
534                           listaOstacoli, deployCoords, packsCoords);
535   }
536   else if (input == 's' || input == 'S')
537   {
538     giocatore = gestisciS(grigliaDiGioco, grigliaOstacoli, giocatore,
539                           listaOstacoli, deployCoords, packsCoords);
540   }
541   else if (input == 'a' || input == 'A')
542   {
543     giocatore = gestisciA(grigliaDiGioco, grigliaOstacoli, giocatore,
544                           listaOstacoli, deployCoords, packsCoords);
545   }
546   else if (input == 'd' || input == 'D')
547   {
548     giocatore = gestisciD(grigliaDiGioco, grigliaOstacoli, giocatore,
549                           listaOstacoli, deployCoords, packsCoords);
550   }
551   else if (input == 'p' || input == 'P')
552   {
553     giocatore = gestisciP(grigliaDiGioco, giocatore, deployCoords, packsCoords);
554   }
555   else if (input == 'c' || input == 'C')
556   {
557     giocatore =
558         gestisciC(grigliaDiGioco, giocatore, deployCoords, packsCoords, name);
559   }
560
561   // aggiorna la posizione dell'utente
562   return giocatore;
563 }
564
565 PlayerStats gestisciC(char grigliaDiGioco[ROWS][COLUMNS], PlayerStats giocatore,
566                       Point deployCoords[], Point packsCoords[], char name[])
567 {
568   pthread_t tid;
569   if (giocatore->hasApack == 0)
570   {
571     return giocatore;
572   }
573   else
574   {
575     if (isOnCorrectDeployPoint(giocatore, deployCoords))
576     {
577       Args args = (Args)malloc(sizeof(struct argsToSend));
578       args->userName = (char *)calloc(MAX_BUF, 1);
579       strcpy(args->userName, name);
580       args->flag = 1;
581       pthread_create(&tid, NULL, fileWriter, (void *)args);
582       giocatore->score += 10;
583       if (giocatore->score > scoreMassimo)
584         scoreMassimo = giocatore->score;
585       giocatore->deploy[0] = -1;
586       giocatore->deploy[1] = -1;
587       giocatore->hasApack = 0;
```

```
588       }
589       else
590       {
591         if (!isOnAPack(giocatore, packsCoords) &&
592             !isOnADeployPoint(giocatore, deployCoords))
593         {
594           int index = getHiddenPack(packsCoords);
595           if (index >= 0)
596           {
597             packsCoords[index]->x = giocatore->position[0];
598             packsCoords[index]->y = giocatore->position[1];
599             giocatore->hasApack = 0;
600             giocatore->deploy[0] = -1;
601             giocatore->deploy[1] = -1;
602           }
603         }
604         else
605           return giocatore;
606       }
607     }
608     return giocatore;
609   }
610
611   void sendPlayerList(int clientDesc)
612   {
613     int lunghezza = 0;
614     char name[100];
615     Players tmp = onLineUsers;
616     int numeroClientLoggati = dimensioneLista(tmp);
617     printf("%d ", numeroClientLoggati);
618     if (!clientDisconnesso(clientDesc))
619     {
620       write(clientDesc, &numeroClientLoggati, sizeof(numeroClientLoggati));
621       while (numeroClientLoggati > 0 && tmp != NULL)
622       {
623         strcpy(name, tmp->name);
624         lunghezza = strlen(tmp->name);
625         write(clientDesc, &lunghezza, sizeof(lunghezza));
626         write(clientDesc, name, lunghezza);
627         tmp = tmp->next;
628         numeroClientLoggati--;
629       }
630     }
631   }
632
633   void prepareMessageForPackDelivery(char message[], char username[], char date[])
634   {
635     strcat(message, "Pack delivered by \"");
636     strcat(message, username);
637     strcat(message, "\" at ");
638     strcat(message, date);
639     strcat(message, "\n");
640   }
641
642   void prepareMessageForLogin(char message[], char username[], char date[])
643   {
644     strcat(message, username);
645     strcat(message, "\" logged in at ");
646     strcat(message, date);
647     strcat(message, "\n");
648   }
649
650   void prepareMessageForConnection(char message[], char ipAddress[], char date[])
651   {
652     strcat(message, ipAddress);
653     strcat(message, "\" connected at ");
654     strcat(message, date);
655     strcat(message, "\n");
656   }
657
658   void putCurrentDateAndTimeInString(char dateAndTime[])
659   {
660     time_t t = time(NULL);
661     struct tm *infoTime = localtime(&t);
```

```
662   strftime(dateAndTime, 64, "%X %x", infoTime);
663 }
664
665 void *fileWriter(void *args)
666 {
667   int fDes = open("Log", O_RDWR | O_CREAT | O_APPEND, S_IWUSR | S_IRUSR);
668   if (fDes < 0)
669   {
670     perror("Error while opening log file");
671     exit(-1);
672   }
673   Args info = (Args)args;
674   char dateAndTime[64];
675   putCurrentDateAndTimeInString(dateAndTime);
676   if (logDelPacco(info->flag))
677   {
678     char message[MAX_BUF] = "";
679     prepareMessageForPackDelivery(message, info->userName, dateAndTime);
680     pthread_mutex_lock(&LogMutex);
681     write(fDes, message, strlen(message));
682     pthread_mutex_unlock(&LogMutex);
683   }
684   else if (logDelLogin(info->flag))
685   {
686     char message[MAX_BUF] = "\"";
687     prepareMessageForLogin(message, info->userName, dateAndTime);
688     pthread_mutex_lock(&LogMutex);
689     write(fDes, message, strlen(message));
690     pthread_mutex_unlock(&LogMutex);
691   }
692   else if (logDellaConnessione(info->flag))
693   {
694     char message[MAX_BUF] = "\"";
695     prepareMessageForConnection(message, info->userName, dateAndTime);
696     pthread_mutex_lock(&LogMutex);
697     write(fDes, message, strlen(message));
698     pthread_mutex_unlock(&LogMutex);
699   }
700   close(fDes);
701   free(info);
702   pthread_exit(NULL);
703 }
704
705 void spostaPlayer(char griglia[ROWS][COLUMNS], int vecchiaPosizione[2],
706                   int nuovaPosizione[2], Point deployCoords[],
707                   Point packsCoords[]) {
708
709   pthread_mutex_lock(&MatrixMutex);
710   griglia[nuovaPosizione[0]][nuovaPosizione[1]] = 'P';
711   if (eraUnPuntoDepo(vecchiaPosizione, deployCoords))
712     griglia[vecchiaPosizione[0]][vecchiaPosizione[1]] = '_';
713   else if (eraUnPacco(vecchiaPosizione, packsCoords))
714     griglia[vecchiaPosizione[0]][vecchiaPosizione[1]] = '$';
715   else
716     griglia[vecchiaPosizione[0]][vecchiaPosizione[1]] = '-';
717   pthread_mutex_unlock(&MatrixMutex);
718 }
719
720 PlayerStats gestisciW(char grigliaDiGioco[ROWS][COLUMNS],
721                       char grigliaOstacoli[ROWS][COLUMNS],
722                       PlayerStats giocatore, Obstacles *listaOstacoli,
723                       Point deployCoords[], Point packsCoords[]) {
724   if (giocatore == NULL)
725     return NULL;
726   int nuovaPosizione[2];
727   nuovaPosizione[1] = giocatore->position[1];
728   // Aggiorna la posizione vecchia spostando il player avanti di 1
729   nuovaPosizione[0] = (giocatore->position[0]) - 1;
730   int nuovoScore = giocatore->score;
731   int nuovoDeploy[2];
732   nuovoDeploy[0] = giocatore->deploy[0];
733   nuovoDeploy[1] = giocatore->deploy[1];
734   if (nuovaPosizione[0] >= 0 && nuovaPosizione[0] < ROWS) {
735     if (casellaVuotaOValida(grigliaDiGioco, grigliaOstacoli, nuovaPosizione)) {
```

```
736       spostaPlayer(grigliaDiGioco, giocatore->position, nuovaPosizione,
737                    deployCoords, packsCoords);
738     } else if (colpitoOstacolo(grigliaOstacoli, nuovaPosizione)) {
739       *listaOstacoli =
740           addObstacle(*listaOstacoli, nuovaPosizione[0], nuovaPosizione[1]);
741       nuovaPosizione[0] = giocatore->position[0];
742       nuovaPosizione[1] = giocatore->position[1];
743     } else if (colpitoPlayer(grigliaDiGioco, nuovaPosizione)) {
744       nuovaPosizione[0] = giocatore->position[0];
745       nuovaPosizione[1] = giocatore->position[1];
746     }
747     giocatore->deploy[0] = nuovoDeploy[0];
748     giocatore->deploy[1] = nuovoDeploy[1];
749     giocatore->score = nuovoScore;
750     giocatore->position[0] = nuovaPosizione[0];
751     giocatore->position[1] = nuovaPosizione[1];
752   }
753   return giocatore;
754 }
755
756 PlayerStats gestisciD(char grigliaDiGioco[ROWS][COLUMNS],
757                       char grigliaOstacoli[ROWS][COLUMNS],
758                       PlayerStats giocatore, Obstacles *listaOstacoli,
759                       Point deployCoords[], Point packsCoords[]) {
760   if (giocatore == NULL) {
761     return NULL;
762   }
763   int nuovaPosizione[2];
764   nuovaPosizione[1] = giocatore->position[1] + 1;
765   nuovaPosizione[0] = giocatore->position[0];
766   int nuovoScore = giocatore->score;
767   int nuovoDeploy[2];
768   nuovoDeploy[0] = giocatore->deploy[0];
769   nuovoDeploy[1] = giocatore->deploy[1];
770   if (nuovaPosizione[1] >= 0 && nuovaPosizione[1] < COLUMNS) {
771     if (casellaVuotaOValida(grigliaDiGioco, grigliaOstacoli, nuovaPosizione)) {
772       spostaPlayer(grigliaDiGioco, giocatore->position, nuovaPosizione,
773                    deployCoords, packsCoords);
774     } else if (colpitoOstacolo(grigliaOstacoli, nuovaPosizione)) {
775       printf("Ostacolo\n");
776       *listaOstacoli =
777           addObstacle(*listaOstacoli, nuovaPosizione[0], nuovaPosizione[1]);
778       nuovaPosizione[0] = giocatore->position[0];
779       nuovaPosizione[1] = giocatore->position[1];
780     } else if (colpitoPlayer(grigliaDiGioco, nuovaPosizione)) {
781       nuovaPosizione[0] = giocatore->position[0];
782       nuovaPosizione[1] = giocatore->position[1];
783     }
784     giocatore->deploy[0] = nuovoDeploy[0];
785     giocatore->deploy[1] = nuovoDeploy[1];
786     giocatore->score = nuovoScore;
787     giocatore->position[0] = nuovaPosizione[0];
788     giocatore->position[1] = nuovaPosizione[1];
789   }
790   return giocatore;
791 }
792 PlayerStats gestisciA(char grigliaDiGioco[ROWS][COLUMNS],
793                       char grigliaOstacoli[ROWS][COLUMNS],
794                       PlayerStats giocatore, Obstacles *listaOstacoli,
795                       Point deployCoords[], Point packsCoords[]) {
796   if (giocatore == NULL)
797     return NULL;
798   int nuovaPosizione[2];
799   nuovaPosizione[0] = giocatore->position[0];
800   // Aggiorna la posizione vecchia spostando il player avanti di 1
801   nuovaPosizione[1] = (giocatore->position[1]) - 1;
802   int nuovoScore = giocatore->score;
803   int nuovoDeploy[2];
804   nuovoDeploy[0] = giocatore->deploy[0];
805   nuovoDeploy[1] = giocatore->deploy[1];
806   if (nuovaPosizione[1] >= 0 && nuovaPosizione[1] < COLUMNS) {
807     if (casellaVuotaOValida(grigliaDiGioco, grigliaOstacoli, nuovaPosizione)) {
808       printf("Casella vuota \n");
809       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)) {
      printf("colpito player\n");
      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;
}
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 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[]);
int casellaVuotaOValida(char grigliaDiGioco[ROWS][COLUMNS],
                        char grigliaOstacoli[ROWS][COLUMNS], int posizione[2]);
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[]);
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[]);
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]);
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[]) {
```

```
73    int i = 0;
74    for (i = 0; i < numberOfPackages; i++) {
75      if (giocatore->position[0] == deployCoords[i]->x &&
76          giocatore->position[1] == deployCoords[i]->y)
77        return 1;
78    }
79    return 0;
80  }
81  void inizializzaGrigliaVuota(char griglia[ROWS][COLUMNS]) {
82    int i = 0, j = 0;
83    for (i = 0; i < ROWS; i++) {
84      for (j = 0; j < COLUMNS; j++) {
85        griglia[i][j] = '-';
86      }
87    }
88  }
89  PlayerStats gestisciP(char grigliaDiGioco[ROWS][COLUMNS], PlayerStats giocatore,
90                        Point deployCoords[], Point packsCoords[]) {
91    int nuovoDeploy[2];
92    if (colpitoPacco(packsCoords, giocatore->position) &&
93        giocatore->hasApack == 0) {
94      scegliPosizioneRaccolta(deployCoords, nuovoDeploy);
95      giocatore->hasApack = 1;
96      rimuoviPaccoDaArray(giocatore->position, packsCoords);
97    }
98    giocatore->deploy[0] = nuovoDeploy[0];
99    giocatore->deploy[1] = nuovoDeploy[1];
100   return giocatore;
101 }
102
103 void printGrid(char grigliaDaStampare[ROWS][COLUMNS], PlayerStats stats) {
104   system("clear");
105   printf("\n\n");
106   int i = 0, j = 0;
107   for (i = 0; i < ROWS; i++) {
108     printf("\t");
109     for (j = 0; j < COLUMNS; j++) {
110       if (stats != NULL) {
111         if ((i == stats->deploy[0] && j == stats->deploy[1]) ||
112             (i == stats->position[0] && j == stats->position[1]))
113           if (grigliaDaStampare[i][j] == 'P' && stats->hasApack == 1)
114             printf(GREEN_COLOR "%c" RESET_COLOR, grigliaDaStampare[i][j]);
115           else
116             printf(RED_COLOR "%c" RESET_COLOR, grigliaDaStampare[i][j]);
117         else
118           printf("%c", grigliaDaStampare[i][j]);
119       } else
120         printf("%c", grigliaDaStampare[i][j]);
121     }
122     stampaIstruzioni(i);
123     if (i == 8)
124       printf(GREEN_COLOR "\t\t Punteggio: %d" RESET_COLOR, stats->score);
125     printf("\n");
126   }
127 }
128 void stampaIstruzioni(int i) {
129   if (i == 0)
130     printf("\t \t ISTRUZIONI ");
131   if (i == 1)
132     printf("\t Inviare 't' per il timer.");
133   if (i == 2)
134     printf("\t Inviare 'e' per uscire");
135   if (i == 3)
136     printf("\t Inviare 'p' per raccogliere un pacco");
137   if (i == 4)
138     printf("\t Inviare 'c' per consegnare il pacco");
139   if (i == 5)
140     printf("\t Inviare 'w'/'s' per andare sopra/sotto");
141   if (i == 6)
142     printf("\t Inviare 'a'/'d' per andare a dx/sx");
143   if (i == 7)
144     printf("\t Inviare 'l' per la lista degli utenti ");
145 }
146 // aggiunge alla griglia gli ostacoli visti fino ad ora dal client
```

```
147 void mergeGridAndList(char grid[ROWS][COLUMNS], Obstacles top) {
148   while (top) {
149     grid[top->x][top->y] = 'O';
150     top = top->next;
151   }
152 }
153 /* Genera la posizione degli ostacoli */
154 void generaPosizioneOstacoli(char grigliaDiGioco[ROWS][COLUMNS],
155                              char grigliaOstacoli[ROWS][COLUMNS]) {
156   int x, y, i;
157   inizializzaGrigliaVuota(grigliaOstacoli);
158   srand(time(0));
159   for (i = 0; i < numberOfObstacles; i++) {
160     x = rand() % COLUMNS;
161     y = rand() % ROWS;
162     if (grigliaDiGioco[y][x] == '-')
163       grigliaOstacoli[y][x] = 'O';
164     else
165       i--;
166   }
167 }
168 void rimuoviPaccoDaArray(int posizione[2], Point packsCoords[]) {
169   int i = 0, found = 0;
170   while (i < numberOfPackages && !found) {
171     if ((packsCoords[i])->x == posizione[0] &&
172         (packsCoords[i])->y == posizione[1]) {
173       (packsCoords[i])->x = -1;
174       (packsCoords[i])->y = -1;
175       found = 1;
176     }
177     i++;
178   }
179 }
180 // sceglie una posizione di raccolta tra quelle disponibili
181 void scegliPosizioneRaccolta(Point coord[], int deploy[]) {
182   int index = 0;
183   srand(time(NULL));
184   index = rand() % numberOfPackages;
185   deploy[0] = coord[index]->x;
186   deploy[1] = coord[index]->y;
187 }
188 /*genera posizione di raccolta di un pacco*/
189 void generaPosizioniRaccolta(char grigliaDiGioco[ROWS][COLUMNS],
190                              char grigliaOstacoli[ROWS][COLUMNS],
191                              Point coord[]) {
192   int x, y;
193   srand(time(0));
194   int i = 0;
195   for (i = 0; i < numberOfPackages; i++) {
196     coord[i] = (Point)malloc(sizeof(struct Coord));
197   }
198   i = 0;
199   for (i = 0; i < numberOfPackages; i++) {
200     x = rand() % COLUMNS;
201     y = rand() % ROWS;
202     if (grigliaDiGioco[y][x] == '-' && grigliaOstacoli[y][x] == '-') {
203       coord[i]->x = y;
204       coord[i]->y = x;
205       grigliaDiGioco[y][x] = '_';
206       grigliaOstacoli[y][x] = '_';
207     } else
208       i--;
209   }
210 }
211 /*Inserisci dei pacchi nella griglia di gioco nella posizione casuale */
212 void riempiGrigliaConPacchiInPosizioniGenerateCasualmente(
213     char grigliaDiGioco[ROWS][COLUMNS], Point packsCoords[]) {
214   int x, y, i = 0;
215   for (i = 0; i < numberOfPackages; i++) {
216     packsCoords[i] = (Point)malloc(sizeof(struct Coord));
217   }
218   srand(time(0));
219   for (i = 0; i < numberOfPackages; i++) {
220     x = rand() % COLUMNS;
```

```
    y = rand() % ROWS;
    if (grigliaDiGioco[y][x] == '-') {
      grigliaDiGioco[y][x] = '$';
      packsCoords[i]->x = y;
      packsCoords[i]->y = x;
    } else
      i--;
  }
}
/*Inserisci gli ostacoli nella griglia di gioco */
void riempiGrigliaConGliOstacoli(char grigliaDiGioco[ROWS][COLUMNS],
                                 char grigliaOstacoli[ROWS][COLUMNS]) {
  int i, j = 0;
  for (i = 0; i < ROWS; i++) {
    for (j = 0; j < COLUMNS; j++) {
      if (grigliaOstacoli[i][j] == 'O')
        grigliaDiGioco[i][j] = 'O';
    }
  }
}
void inserisciPlayerNellaGrigliaInPosizioneCasuale(
    char grigliaDiGioco[ROWS][COLUMNS], char grigliaOstacoli[ROWS][COLUMNS],
    int posizione[2]) {
  int x, y;
  srand(time(0));
  printf("Inserisco player\n");
  do {
    x = rand() % COLUMNS;
    y = rand() % ROWS;
  } while (grigliaDiGioco[y][x] != '-' && grigliaOstacoli[y][x] != '-');
  grigliaDiGioco[y][x] = 'P';
  posizione[0] = y;
  posizione[1] = x;
}
void inizializzaGiocoSenzaPlayer(char grigliaDiGioco[ROWS][COLUMNS],
                                 char grigliaConOstacoli[ROWS][COLUMNS],
                                 Point packsCoords[]) {
  inizializzaGrigliaVuota(grigliaDiGioco);
  riempiGrigliaConPacchiInPosizioniGenerateCasualmente(grigliaDiGioco,
                                                       packsCoords);
  generaPosizioneOstacoli(grigliaDiGioco, grigliaConOstacoli);
  return;
}

int eraUnPuntoDepo(int vecchiaPosizione[2], Point depo[]) {
  int i = 0, ret = 0;
  while (ret == 0 && i < numberOfPackages) {
    if ((depo[i])->y == vecchiaPosizione[1] &&
        (depo[i])->x == vecchiaPosizione[0]) {
      ret = 1;
    }
    i++;
  }
  return ret;
}
int eraUnPacco(int vecchiaPosizione[2], Point packsCoords[]) {
  int i = 0, ret = 0;
  while (ret == 0 && i < numberOfPackages) {
    if ((packsCoords[i])->y == vecchiaPosizione[1] &&
        (packsCoords[i])->x == vecchiaPosizione[0]) {
      ret = 1;
    }
    i++;
  }
  return ret;
}

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
```

```
70 Obstacles initObstacleNode(int x, int y);
71 #endif
```

Listato 18: Codice sorgente utility del gioco 2

```
1  #include "list.h"
2  #include <pthread.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6
7  Players initPlayerNode(char *name, int sockDes) {
8    Players L = (Players)malloc(sizeof(struct TList));
9    L->name = (char *)malloc(MAX_BUF);
10   strcpy(L->name, name);
11   L->sockDes = sockDes;
12   L->next = NULL;
13   return L;
14 }
15 PlayerStats initStats(int deploy[], int score, int position[], int flag) {
16   PlayerStats L = (PlayerStats)malloc(sizeof(struct Data));
17   L->deploy[0] = deploy[0];
18   L->deploy[1] = deploy[1];
19   L->score = score;
20   L->hasApack = flag;
21   L->position[0] = position[0];
22   L->position[1] = position[1];
23   return L;
24 }
25 Obstacles initObstacleNode(int x, int y) {
26   Obstacles L = (Obstacles)malloc(sizeof(struct TList2));
27   L->x = x;
28   L->y = y;
29   L->next = NULL;
30   return L;
31 }
32 Obstacles addObstacle(Obstacles L, int x, int y) {
33   Obstacles tmp = initObstacleNode(x, y);
34   if (L != NULL)
35     tmp->next = L;
36   return tmp;
37 }
38 int dimensioneLista(Players L) {
39   int size = 0;
40   Players tmp = L;
41   while (tmp != NULL) {
42     size++;
43     tmp = tmp->next;
44   }
45   return size;
46 }
47 int isAlreadyLogged(Players L, char *name) {
48   int ret = 0;
49   if (L != NULL) {
50     if (strcmp(L->name, name) == 0)
51       return 1;
52     ret = isAlreadyLogged(L->next, name);
53   }
54   return ret;
55 }
56 Players addPlayer(Players L, char *name, int sockDes) {
57   Players tmp = initPlayerNode(name, sockDes);
58   if (L != NULL)
59     tmp->next = L;
60   return tmp;
61 }
62 Players removePlayer(Players L, int sockDes) {
63   if (L != NULL) {
64     if (L->sockDes == sockDes) {
65       Players tmp = L->next;
66       free(L);
67       return tmp;
68     }
```

```
    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);
    printPlayers(L->next);
  }
  printf("\n");
}
void printObstacles(Obstacles L) {
  if (L != NULL) {
    printf("X:%d Y:%d ->", L->x, L->y);
    printObstacles(L->next);
  }
}
```

A.5 Codice sorgente parser

Listato 19: Codice header utility del gioco 3

```
int appendPlayer(char *name, char *pwd, char *file);
int isRegistered(char *name, char *file);
int openFileRDWRAPP(char *file);
int validateLogin(char *name, char *pwd, char *file);
int openFileRDON(char *file);
void premiEnterPerContinuare();
```

Listato 20: Codice sorgente utility del gioco 3

```
#include "parser.h"
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#define MAX_BUF 200
int openFileRDWRAPP(char *file) {
  int fileDes = open(file, O_RDWR | O_CREAT | O_APPEND, S_IRUSR | S_IWUSR);
  if (fileDes < 0)
    perror("Errore apertura file\n"), exit(-1);
  return fileDes;
}
int openFileRDON(char *file) {
  int fileDes = open(file, O_RDONLY);
  if (fileDes < 0)
    perror("Errore apertura file\n"), exit(-1);
  return fileDes;
}
int appendPlayer(char *name, char *pwd, char *file) {
  if (isRegistered(name, file))
    return 0;
  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