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

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

Listato 2: Configurazione socket del server

```
    risposta = 'n';
    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);
```

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

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

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. 15 Rigo 430 e List. 17 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. 14 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. 15 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   write(clientDescriptor, &msg, sizeof(msg));
2   close(clientDescriptor);
3 }
4 int clientDisconnesso(int clientSocket)
5 {
6   char msg[1] = {'u'}; // UP?
7   if (write(clientSocket, msg, sizeof(msg)) < 0)
8     return 1;
9   if (read(clientSocket, msg, sizeof(char)) < 0)
10    return 1;
11  else
12    return 0;
13 }
14 int registraClient(int clientDesc)
15 {
16   char *userName = (char *)calloc(MAX_BUF, 1);
17   char *password = (char *)calloc(MAX_BUF, 1);
18   int dimName, dimPwd;
19   read(clientDesc, &dimName, sizeof(int));
20   read(clientDesc, &dimPwd, sizeof(int));
21   read(clientDesc, userName, dimName);
22   read(clientDesc, password, dimPwd);
23   pthread_mutex_lock(&RegMutex);
24   int ret = appendPlayer(userName, password, users);
25   pthread_mutex_unlock(&RegMutex);
26   char risposta;
27   if (!ret)
28   {
```

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

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, username);
  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.

5.3 Modifica della mappa di gioco da parte di più thread

La mappa di gioco è la stessa per tutti i player e c'è il rischio che lo spostamento dei player e/o la raccolta ed il deposito di pacchi possano provocare problemi a causa della race condition che si viene a creare tra i thread. Tutto ciò è stato risolto con una serie di semplici accorgimenti implementativi. Il primo accorgimento, e forse anche il più importante, è la funzione spostaPlayer mostrata qui di seguito.

Listato 13: Funzione spostaPlayer

```
PlayerStats gestisciW(char grigliaDiGioco[ROWS][COLUMNS],
                      char grigliaOstacoli[ROWS][COLUMNS],
                      PlayerStats giocatore, Obstacles *listaOstacoli,
                      Point deployCoords[], Point packsCoords[])
{
  if (giocatore == NULL)
    return NULL;
  int nuovaPosizione[2];
  nuovaPosizione[1] = giocatore->position[1];
  // Aggiorna la posizione vecchia spostando il player avanti di 1
  nuovaPosizione[0] = (giocatore->position[0]) - 1;
  int nuovoScore = giocatore->score;
  int nuovoDeploy[2];
  nuovoDeploy[0] = giocatore->deploy[0];
```

A Codici sorgente

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

A.1 Codice sorgente del client

Listato 14: 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
#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 */
```

```
44    signal(SIGQUIT, clientCrashHandler);
45    signal(SIGTSTP, clientCrashHandler); /* CTRL-Z*/
46    signal(SIGTERM, clientCrashHandler); /* generato da 'kill' */
47    signal(SIGPIPE, serverCrashHandler);
48    char bufferReceive[2];
49    if (argc != 3) {
50      perror("Inserire indirizzo ip/url e porta (./client 127.0.0.1 5200)");
51      exit(-1);
52    }
53    if ((socketDesc = connettiAlServer(argv)) < 0)
54      exit(-1);
55    gestisci(socketDesc);
56    close(socketDesc);
57    exit(0);
58  }
59  void esciDalServer() {
60    int msg = 3;
61    printf("Uscita in corso\n");
62    write(socketDesc, &msg, sizeof(int));
63    close(socketDesc);
64  }
65  int connettiAlServer(char **argv) {
66    char *indirizzoServer;
67    uint16_t porta = strtoul(argv[2], NULL, 10);
68    indirizzoServer = ipResolver(argv);
69    struct sockaddr_in mio_indirizzo;
70    mio_indirizzo.sin_family = AF_INET;
71    mio_indirizzo.sin_port = htons(porta);
72    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];
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 15: Codice sorgente del server

```
1  #include "boardUtility.h"
2  #include "list.h"
3  #include "parser.h"
4  #include <arpa/inet.h>
5  #include <errno.h>
6  #include <fcntl.h>
7  #include <netinet/in.h> //conversioni
8  #include <netinet/ip.h> //struttura
9  #include <pthread.h>
10 #include <signal.h>
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14 #include <sys/socket.h>
15 #include <sys/stat.h>
16 #include <sys/types.h>
17 #include <time.h>
18 #include <unistd.h>
19
20 //struttura di argomenti da mandare al thread che scrive sul file di log
21 struct argsToSend
22 {
23   char *userName;
24   int flag;
25 };
26
27 typedef struct argsToSend *Args;
28 void prepareMessageForLogin(char message[], char username[], char date[]);
29 void sendPlayerList(int clientDesc);
30 PlayerStats gestisciC(char grigliaDiGioco[ROWS][COLUMNS], PlayerStats giocatore,
31                       Point deployCoords[], Point packsCoords[], char name[]);
32 PlayerStats gestisciInput(char grigliaDiGioco[ROWS][COLUMNS],
33                           char grigliaOstacoli[ROWS][COLUMNS], char input,
34                           PlayerStats giocatore, Obstacles *listaOstacoli,
35                           Point deployCoords[], Point packsCoords[],
36                           char name[]);
37 void clonaGriglia(char destinazione[ROWS][COLUMNS], char source[ROWS][COLUMNS]);
38 int almenoUnClientConnesso();
39 void prepareMessageForConnection(char message[], char ipAddress[], char date[]);
```

```
40  void spostaPlayer(char griglia[ROWS][COLUMNS], int vecchiaPosizione[2],
41                    int nuovaPosizione[2], Point deployCoords[],
42                    Point packsCoords[]);
43  int valoreTimerValido();
44  PlayerStats gestisciW(char grigliaDiGioco[ROWS][COLUMNS],
45                        char grigliaOstacoli[ROWS][COLUMNS],
46                        PlayerStats giocatore, Obstacles *listaOstacoli,
47                        Point deployCoords[], Point packsCoords[]);
48  PlayerStats gestisciA(char grigliaDiGioco[ROWS][COLUMNS],
49                        char grigliaOstacoli[ROWS][COLUMNS],
50                        PlayerStats giocatore, Obstacles *listaOstacoli,
51                        Point deployCoords[], Point packsCoords[]);
52  PlayerStats gestisciD(char grigliaDiGioco[ROWS][COLUMNS],
53                        char grigliaOstacoli[ROWS][COLUMNS],
54                        PlayerStats giocatore, Obstacles *listaOstacoli,
55                        Point deployCoords[], Point packsCoords[]);
56  PlayerStats gestisciS(char grigliaDiGioco[ROWS][COLUMNS],
57                        char grigliaOstacoli[ROWS][COLUMNS],
58                        PlayerStats giocatore, Obstacles *listaOstacoli,
59                        Point deployCoords[], Point packsCoords[]);
60  int almenoUnPlayerGenerato();
61  int almenoUnaMossaFatta();
62  void sendTimerValue(int clientDesc);
63  void putCurrentDateAndTimeInString(char dateAndTime[]);
64  void startProceduraGenrazioneMappa();
65  void *threadGenerazioneMappa(void *args);
66  void *fileWriter(void *);
67  int tryLogin(int clientDesc, char name[]);
68  void disconnettiClient(int);
69  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
85  char grigliaDiGiocoConPacchiSenzaOstacoli[ROWS][COLUMNS]; //protetta
86  char grigliaOstacoliSenzaPacchi[ROWS][COLUMNS];           //protetta
87  int numeroClientLoggati = 0;                              //protetto
88  int playerGenerati = 0;                                   //mutex
89  int timerCount = TIME_LIMIT_IN_SECONDS;
90  int turno = 0; //lo cambia solo timer
91  pthread_t tidTimer;
92  pthread_t tidGeneratoreMappa;
93  int socketDesc;
94  Players onLineUsers = NULL; //protetto
95  char *users;
96  int scoreMassimo = 0; //mutex
97  int numMosse = 0;     //mutex
98  Point deployCoords[numberOfPackages];
99  Point packsCoords[numberOfPackages];
100 pthread_mutex_t LogMutex = PTHREAD_MUTEX_INITIALIZER;
101 pthread_mutex_t RegMutex = PTHREAD_MUTEX_INITIALIZER;
102 pthread_mutex_t PlayerMutex = PTHREAD_MUTEX_INITIALIZER;
103 pthread_mutex_t MatrixMutex = PTHREAD_MUTEX_INITIALIZER;
104
105 int main(int argc, char **argv)
106 {
107   if (argc != 2)
108   {
109     printf("Wrong parameters number(Usage: ./server usersFile)\n");
110     exit(-1);
111   }
112   else if (strcmp(argv[1], "Log") == 0)
113   {
```

```
114     printf("Cannot use the Log file as a UserList \n");
115     exit(-1);
116   }
117   users = argv[1];
118   struct sockaddr_in mio_indirizzo = configuraIndirizzo();
119   configuraSocket(mio_indirizzo);
120   signal(SIGPIPE, clientCrashHandler);
121   signal(SIGINT, quitServer);
122   signal(SIGHUP, quitServer);
123   startTimer();
124   inizializzaGiocoSenzaPlayer(grigliaDiGiocoConPacchiSenzaOstacoli,
125                               grigliaOstacoliSenzaPacchi, packsCoords);
126   generaPosizioniRaccolta(grigliaDiGiocoConPacchiSenzaOstacoli,
127                           grigliaOstacoliSenzaPacchi, deployCoords);
128   startListening();
129   return 0;
130 }
131 void startListening()
132 {
133   pthread_t tid;
134   int clientDesc;
135   int *puntClientDesc;
136   while (1 == 1)
137   {
138     if (listen(socketDesc, 10) < 0)
139       perror("Impossibile mettersi in ascolto"), exit(-1);
140     printf("In ascolto..\n");
141     if ((clientDesc = accept(socketDesc, NULL, NULL)) < 0)
142     {
143       perror("Impossibile effettuare connessione\n");
144       exit(-1);
145     }
146     printf("Nuovo client connesso\n");
147     struct sockaddr_in address;
148     socklen_t size = sizeof(struct sockaddr_in);
149     if (getpeername(clientDesc, (struct sockaddr *)&address, &size) < 0)
150     {
151       perror("Impossibile ottenere l'indirizzo del client");
152       exit(-1);
153     }
154     char clientAddr[20];
155     strcpy(clientAddr, inet_ntoa(address.sin_addr));
156     Args args = (Args)malloc(sizeof(struct argsToSend));
157     args->userName = (char *)calloc(MAX_BUF, 1);
158     strcpy(args->userName, clientAddr);
159     args->flag = 2;
160     pthread_t tid;
161     pthread_create(&tid, NULL, fileWriter, (void *)args);
162
163     puntClientDesc = (int *)malloc(sizeof(int));
164     *puntClientDesc = clientDesc;
165     pthread_create(&tid, NULL, gestisci, (void *)puntClientDesc);
166   }
167   close(clientDesc);
168   quitServer();
169 }
170 struct sockaddr_in configuraIndirizzo()
171 {
172   struct sockaddr_in mio_indirizzo;
173   mio_indirizzo.sin_family = AF_INET;
174   mio_indirizzo.sin_port = htons(5200);
175   mio_indirizzo.sin_addr.s_addr = htonl(INADDR_ANY);
176   printf("Indirizzo socket configurato\n");
177   return mio_indirizzo;
178 }
179 void startProceduraGenrazioneMappa()
180 {
181   printf("Inizio procedura generazione mappa\n");
182   pthread_create(&tidGeneratoreMappa, NULL, threadGenerazioneMappa, NULL);
183 }
184 void startTimer()
185 {
186   printf("Thread timer avviato\n");
187   pthread_create(&tidTimer, NULL, timer, NULL);
```

```
188 }
189 int tryLogin(int clientDesc, char name[])
190 {
191   char *userName = (char *)calloc(MAX_BUF, 1);
192   char *password = (char *)calloc(MAX_BUF, 1);
193   int dimName, dimPwd;
194   read(clientDesc, &dimName, sizeof(int));
195   read(clientDesc, &dimPwd, sizeof(int));
196   read(clientDesc, userName, dimName);
197   read(clientDesc, password, dimPwd);
198   int ret = 0;
199   pthread_mutex_lock(&PlayerMutex);
200   if (validateLogin(userName, password, users) &&
201       !isAlreadyLogged(onLineUsers, userName))
202   {
203     ret = 1;
204     numeroClientLoggati++;
205     write(clientDesc, "y", 1);
206     strcpy(name, userName);
207     Args args = (Args)malloc(sizeof(struct argsToSend));
208     args->userName = (char *)calloc(MAX_BUF, 1);
209     strcpy(args->userName, name);
210     args->flag = 0;
211     pthread_t tid;
212     pthread_create(&tid, NULL, fileWriter, (void *)args);
213     printf("Nuovo client loggato, client loggati : %d\n", numeroClientLoggati);
214     onLineUsers = addPlayer(onLineUsers, userName, clientDesc);
215     pthread_mutex_unlock(&PlayerMutex);
216     printPlayers(onLineUsers);
217     printf("\n");
218   }
219   else
220   {
221     write(clientDesc, "n", 1);
222   }
223   return ret;
224 }
225 void *gestisci(void *descriptor)
226 {
227   int bufferReceive[2] = {1};
228   int client_sd = *(int *)descriptor;
229   int continua = 1;
230   char name[MAX_BUF];
231   while (continua)
232   {
233     read(client_sd, bufferReceive, sizeof(bufferReceive));
234     if (bufferReceive[0] == 2)
235       registraClient(client_sd);
236     else if (bufferReceive[0] == 1)
237       if (tryLogin(client_sd, name))
238       {
239         play(client_sd, name);
240         continua = 0;
241       }
242       else if (bufferReceive[0] == 3)
243         disconnettiClient(client_sd);
244       else
245       {
246         printf("Input invalido, uscita...\n");
247         disconnettiClient(client_sd);
248       }
249   }
250   pthread_exit(0);
251 }
252 void play(int clientDesc, char name[])
253 {
254   int true = 1;
255   int turnoFinito = 0;
256   int turnoGiocatore = turno;
257   int posizione[2];
258   int destinazione[2] = {-1, -1};
259   PlayerStats giocatore = initStats(destinazione, 0, posizione, 0);
260   Obstacles listaOstacoli = NULL;
261   char inputFromClient;
```

```
262   if (timer != 0)
263   {
264     inserisciPlayerNellaGrigliaInPosizioneCasuale(
265         grigliaDiGiocoConPacchiSenzaOstacoli, grigliaOstacoliSenzaPacchi,
266         giocatore->position);
267     playerGenerati++;
268   }
269   while (true)
270   {
271     if (clientDisconnesso(clientDesc))
272     {
273       freeObstacles(listaOstacoli);
274       disconnettiClient(clientDesc);
275       return;
276     }
277     char grigliaTmp[ROWS][COLUMNS];
278     clonaGriglia(grigliaTmp, grigliaDiGiocoConPacchiSenzaOstacoli);
279     mergeGridAndList(grigliaTmp, listaOstacoli);
280     // invia la griglia
281     write(clientDesc, grigliaTmp, sizeof(grigliaTmp));
282     // invia la struttura del player
283     write(clientDesc, giocatore->deploy, sizeof(giocatore->deploy));
284     write(clientDesc, giocatore->position, sizeof(giocatore->position));
285     write(clientDesc, &giocatore->score, sizeof(giocatore->score));
286     write(clientDesc, &giocatore->hasApack, sizeof(giocatore->hasApack));
287     // legge l'input
288     if (read(clientDesc, &inputFromClient, sizeof(char)) > 0)
289       numMosse++;
290     if (inputFromClient == 'e' || inputFromClient == 'E')
291     {
292       freeObstacles(listaOstacoli);
293       listaOstacoli = NULL;
294       disconnettiClient(clientDesc);
295     }
296     else if (inputFromClient == 't' || inputFromClient == 'T')
297     {
298       write(clientDesc, &turnoFinito, sizeof(int));
299       sendTimerValue(clientDesc);
300     }
301     else if (inputFromClient == 'l' || inputFromClient == 'L')
302     {
303       write(clientDesc, &turnoFinito, sizeof(int));
304       sendPlayerList(clientDesc);
305     }
306     else if (turnoGiocatore == turno)
307     {
308       write(clientDesc, &turnoFinito, sizeof(int));
309       giocatore =
310           gestisciInput(grigliaDiGiocoConPacchiSenzaOstacoli,
311                         grigliaOstacoliSenzaPacchi, inputFromClient, giocatore,
312                         &listaOstacoli, deployCoords, packsCoords, name);
313     }
314     else
315     {
316       turnoFinito = 1;
317       write(clientDesc, &turnoFinito, sizeof(int));
318       freeObstacles(listaOstacoli);
319       listaOstacoli = NULL;
320       inserisciPlayerNellaGrigliaInPosizioneCasuale(
321           grigliaDiGiocoConPacchiSenzaOstacoli, grigliaOstacoliSenzaPacchi,
322           giocatore->position);
323       giocatore->score = 0;
324       giocatore->hasApack = 0;
325       giocatore->deploy[0] = -1;
326       giocatore->deploy[1] = -1;
327       turnoGiocatore = turno;
328       turnoFinito = 0;
329       playerGenerati++;
330     }
331   }
332 }
333 void sendTimerValue(int clientDesc)
334 {
335   if (!clientDisconnesso(clientDesc))
```

```
336     write(clientDesc, &timerCount, sizeof(timerCount));
337 }
338 void clonaGriglia(char destinazione[ROWS][COLUMNS],
339                   char source[ROWS][COLUMNS])
340 {
341   int i = 0, j = 0;
342   for (i = 0; i < ROWS; i++)
343   {
344     for (j = 0; j < COLUMNS; j++)
345     {
346       destinazione[i][j] = source[i][j];
347     }
348   }
349 }
350 void clientCrashHandler(int signalNum)
351 {
352   char msg[0];
353   int socketClientCrashato;
354   int flag = 1;
355   // TODO eliminare la lista degli ostacoli dell'utente
356   if (onLineUsers != NULL)
357   {
358     Players prec = onLineUsers;
359     Players top = prec->next;
360     while (top != NULL && flag)
361     {
362       if (write(top->sockDes, msg, sizeof(msg)) < 0)
363       {
364         socketClientCrashato = top->sockDes;
365         printPlayers(onLineUsers);
366         disconnettiClient(socketClientCrashato);
367         flag = 0;
368       }
369       top = top->next;
370     }
371   }
372   signal(SIGPIPE, SIG_IGN);
373 }
374 void disconnettiClient(int clientDescriptor)
375 {
376   if (numeroClientLoggati > 0)
377     numeroClientLoggati--;
378   pthread_mutex_lock(&PlayerMutex);
379   onLineUsers = removePlayer(onLineUsers, clientDescriptor);
380   pthread_mutex_unlock(&PlayerMutex);
381   printPlayers(onLineUsers);
382   int msg = 1;
383   printf("Client disconnesso (client attualmente loggati: %d)\n",
384          numeroClientLoggati);
385   write(clientDescriptor, &msg, sizeof(msg));
386   close(clientDescriptor);
387 }
388 int clientDisconnesso(int clientSocket)
389 {
390   char msg[1] = {'u'}; // UP?
391   if (write(clientSocket, msg, sizeof(msg)) < 0)
392     return 1;
393   if (read(clientSocket, msg, sizeof(char)) < 0)
394     return 1;
395   else
396     return 0;
397 }
398 int registraClient(int clientDesc)
399 {
400   char *userName = (char *)calloc(MAX_BUF, 1);
401   char *password = (char *)calloc(MAX_BUF, 1);
402   int dimName, dimPwd;
403   read(clientDesc, &dimName, sizeof(int));
404   read(clientDesc, &dimPwd, sizeof(int));
405   read(clientDesc, userName, dimName);
406   read(clientDesc, password, dimPwd);
407   pthread_mutex_lock(&RegMutex);
408   int ret = appendPlayer(userName, password, users);
409   pthread_mutex_unlock(&RegMutex);
```

```
410   char risposta;
411   if (!ret)
412   {
413     risposta = 'n';
414     write(clientDesc, &risposta, sizeof(char));
415     printf("Impossibile registrare utente, riprovare\n");
416   }
417   else
418   {
419     risposta = 'y';
420     write(clientDesc, &risposta, sizeof(char));
421     printf("Utente registrato con successo\n");
422   }
423   return ret;
424 }
425 void quitServer()
426 {
427   printf("Chiusura server in corso..\n");
428   close(socketDesc);
429   exit(-1);
430 }
431 void *threadGenerazioneMappa(void *args)
432 {
433   fprintf(stdout, "Rigenerazione mappa\n");
434   inizializzaGrigliaVuota(grigliaDiGiocoConPacchiSenzaOstacoli);
435   generaPosizioniRaccolta(grigliaDiGiocoConPacchiSenzaOstacoli,
436                           grigliaOstacoliSenzaPacchi, deployCoords);
437   riempiGrigliaConPacchiInPosizioniGenerateCasualmente(
438       grigliaDiGiocoConPacchiSenzaOstacoli, packsCoords);
439   generaPosizioneOstacoli(grigliaDiGiocoConPacchiSenzaOstacoli,
440                           grigliaOstacoliSenzaPacchi);
441   printf("Mappa generata\n");
442   pthread_exit(NULL);
443 }
444 int almenoUnaMossaFatta()
445 {
446   if (numMosse > 0)
447     return 1;
448   return 0;
449 }
450 int almenoUnClientConnesso()
451 {
452   if (numeroClientLoggati > 0)
453     return 1;
454   return 0;
455 }
456 int valoreTimerValido()
457 {
458   if (timerCount > 0 && timerCount <= TIME_LIMIT_IN_SECONDS)
459     return 1;
460   return 0;
461 }
462 int almenoUnPlayerGenerato()
463 {
464   if (playerGenerati > 0)
465     return 1;
466   return 0;
467 }
468 void *timer(void *args)
469 {
470   int cambiato = 1;
471   while (1)
472   {
473     if (almenoUnClientConnesso() && valoreTimerValido() &&
474         almenoUnPlayerGenerato() && almenoUnaMossaFatta())
475     {
476       cambiato = 1;
477       sleep(1);
478       timerCount--;
479       fprintf(stdout, "Time left: %d\n", timerCount);
480     }
481     else if (numeroClientLoggati == 0)
482     {
483       timerCount = TIME_LIMIT_IN_SECONDS;
```

```
484       if (cambiato)
485       {
486         fprintf(stdout, "Time left: %d\n", timerCount);
487         cambiato = 0;
488       }
489     }
490     if (timerCount == 0 || scoreMassimo == packageLimitNumber)
491     {
492       playerGenerati = 0;
493       numMosse = 0;
494       printf("Reset timer e generazione nuova mappa..\n");
495       startProceduraGenrazioneMappa();
496       pthread_join(tidGeneratoreMappa, NULL);
497       turno++;
498       timerCount = TIME_LIMIT_IN_SECONDS;
499     }
500   }
501 }
502
503 void configuraSocket(struct sockaddr_in mio_indirizzo)
504 {
505   if ((socketDesc = socket(PF_INET, SOCK_STREAM, 0)) < 0)
506   {
507     perror("Impossibile creare socket");
508     exit(-1);
509   }
510   if (setsockopt(socketDesc, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) <
511       0)
512     perror("Impossibile impostare il riutilizzo dell'indirizzo ip e della "
513            "porta\n");
514   if ((bind(socketDesc, (struct sockaddr *)&mio_indirizzo,
515             sizeof(mio_indirizzo))) < 0)
516   {
517     perror("Impossibile effettuare bind");
518     exit(-1);
519   }
520 }
521
522 PlayerStats gestisciInput(char grigliaDiGioco[ROWS][COLUMNS],
523                           char grigliaOstacoli[ROWS][COLUMNS], char input,
524                           PlayerStats giocatore, Obstacles *listaOstacoli,
525                           Point deployCoords[], Point packsCoords[],
526                           char name[])
527 {
528   if (giocatore == NULL)
529   {
530     return NULL;
531   }
532   if (input == 'w' || input == 'W')
533   {
534     giocatore = gestisciW(grigliaDiGioco, grigliaOstacoli, giocatore,
535                           listaOstacoli, deployCoords, packsCoords);
536   }
537   else if (input == 's' || input == 'S')
538   {
539     giocatore = gestisciS(grigliaDiGioco, grigliaOstacoli, giocatore,
540                           listaOstacoli, deployCoords, packsCoords);
541   }
542   else if (input == 'a' || input == 'A')
543   {
544     giocatore = gestisciA(grigliaDiGioco, grigliaOstacoli, giocatore,
545                           listaOstacoli, deployCoords, packsCoords);
546   }
547   else if (input == 'd' || input == 'D')
548   {
549     giocatore = gestisciD(grigliaDiGioco, grigliaOstacoli, giocatore,
550                           listaOstacoli, deployCoords, packsCoords);
551   }
552   else if (input == 'p' || input == 'P')
553   {
554     giocatore = gestisciP(grigliaDiGioco, giocatore, deployCoords, packsCoords);
555   }
556   else if (input == 'c' || input == 'C')
557   {
```

```
558       giocatore =
559           gestisciC(grigliaDiGioco, giocatore, deployCoords, packsCoords, name);
560     }
561
562     // aggiorna la posizione dell'utente
563     return giocatore;
564   }
565
566   PlayerStats gestisciC(char grigliaDiGioco[ROWS][COLUMNS], PlayerStats giocatore,
567                         Point deployCoords[], Point packsCoords[], char name[])
568   {
569     pthread_t tid;
570     if (giocatore->hasApack == 0)
571     {
572       return giocatore;
573     }
574     else
575     {
576       if (isOnCorrectDeployPoint(giocatore, deployCoords))
577       {
578         Args args = (Args)malloc(sizeof(struct argsToSend));
579         args->userName = (char *)calloc(MAX_BUF, 1);
580         strcpy(args->userName, name);
581         args->flag = 1;
582         pthread_create(&tid, NULL, fileWriter, (void *)args);
583         giocatore->score += 10;
584         if (giocatore->score > scoreMassimo)
585           scoreMassimo = giocatore->score;
586         giocatore->deploy[0] = -1;
587         giocatore->deploy[1] = -1;
588         giocatore->hasApack = 0;
589       }
590       else
591       {
592         if (!isOnAPack(giocatore, packsCoords) &&
593             !isOnADeployPoint(giocatore, deployCoords))
594         {
595           int index = getHiddenPack(packsCoords);
596           if (index >= 0)
597           {
598             packsCoords[index]->x = giocatore->position[0];
599             packsCoords[index]->y = giocatore->position[1];
600             giocatore->hasApack = 0;
601             giocatore->deploy[0] = -1;
602             giocatore->deploy[1] = -1;
603           }
604         }
605         else
606           return giocatore;
607       }
608     }
609     return giocatore;
610   }
611
612   void sendPlayerList(int clientDesc)
613   {
614     int lunghezza = 0;
615     char name[100];
616     Players tmp = onLineUsers;
617     int numeroClientLoggati = dimensioneLista(tmp);
618     printf("%d ", numeroClientLoggati);
619     if (!clientDisconnesso(clientDesc))
620     {
621       write(clientDesc, &numeroClientLoggati, sizeof(numeroClientLoggati));
622       while (numeroClientLoggati > 0 && tmp != NULL)
623       {
624         strcpy(name, tmp->name);
625         lunghezza = strlen(tmp->name);
626         write(clientDesc, &lunghezza, sizeof(lunghezza));
627         write(clientDesc, name, lunghezza);
628         tmp = tmp->next;
629         numeroClientLoggati--;
630       }
631     }
```

```
632 }
633
634 void prepareMessageForPackDelivery(char message[], char username[], char date[])
635 {
636   strcat(message, "Pack delivered by \"");
637   strcat(message, username);
638   strcat(message, "\" at ");
639   strcat(message, date);
640   strcat(message, "\n");
641 }
642
643 void prepareMessageForLogin(char message[], char username[], char date[])
644 {
645   strcat(message, username);
646   strcat(message, "\" logged in at ");
647   strcat(message, date);
648   strcat(message, "\n");
649 }
650
651 void prepareMessageForConnection(char message[], char ipAddress[], char date[])
652 {
653   strcat(message, ipAddress);
654   strcat(message, "\" connected at ");
655   strcat(message, date);
656   strcat(message, "\n");
657 }
658
659 void putCurrentDateAndTimeInString(char dateAndTime[])
660 {
661   time_t t = time(NULL);
662   struct tm *infoTime = localtime(&t);
663   strftime(dateAndTime, 64, "%X %x", infoTime);
664 }
665
666 void *fileWriter(void *args)
667 {
668   int fDes = open("Log", O_RDWR | O_CREAT | O_APPEND, S_IWUSR | S_IRUSR);
669   if (fDes < 0)
670   {
671     perror("Error while opening log file");
672     exit(-1);
673   }
674   Args info = (Args)args;
675   char dateAndTime[64];
676   putCurrentDateAndTimeInString(dateAndTime);
677   if (logDelPacco(info->flag))
678   {
679     char message[MAX_BUF] = "";
680     prepareMessageForPackDelivery(message, info->userName, dateAndTime);
681     pthread_mutex_lock(&LogMutex);
682     write(fDes, message, strlen(message));
683     pthread_mutex_unlock(&LogMutex);
684   }
685   else if (logDelLogin(info->flag))
686   {
687     char message[MAX_BUF] = "\"";
688     prepareMessageForLogin(message, info->userName, dateAndTime);
689     pthread_mutex_lock(&LogMutex);
690     write(fDes, message, strlen(message));
691     pthread_mutex_unlock(&LogMutex);
692   }
693   else if (logDellaConnessione(info->flag))
694   {
695     char message[MAX_BUF] = "\"";
696     prepareMessageForConnection(message, info->userName, dateAndTime);
697     pthread_mutex_lock(&LogMutex);
698     write(fDes, message, strlen(message));
699     pthread_mutex_unlock(&LogMutex);
700   }
701   close(fDes);
702   free(info);
703   pthread_exit(NULL);
704 }
705
```

```
706 void spostaPlayer(char griglia[ROWS][COLUMNS], int vecchiaPosizione[2],
707                   int nuovaPosizione[2], Point deployCoords[],
708                   Point packsCoords[])
709 {
710
711   pthread_mutex_lock(&MatrixMutex);
712   griglia[nuovaPosizione[0]][nuovaPosizione[1]] = 'P';
713   if (eraUnPuntoDepo(vecchiaPosizione, deployCoords))
714     griglia[vecchiaPosizione[0]][vecchiaPosizione[1]] = '_';
715   else if (eraUnPacco(vecchiaPosizione, packsCoords))
716     griglia[vecchiaPosizione[0]][vecchiaPosizione[1]] = '$';
717   else
718     griglia[vecchiaPosizione[0]][vecchiaPosizione[1]] = '-';
719   pthread_mutex_unlock(&MatrixMutex);
720 }
721
722 PlayerStats gestisciW(char grigliaDiGioco[ROWS][COLUMNS],
723                       char grigliaOstacoli[ROWS][COLUMNS],
724                       PlayerStats giocatore, Obstacles *listaOstacoli,
725                       Point deployCoords[], Point packsCoords[])
726 {
727   if (giocatore == NULL)
728     return NULL;
729   int nuovaPosizione[2];
730   nuovaPosizione[1] = giocatore->position[1];
731   // Aggiorna la posizione vecchia spostando il player avanti di 1
732   nuovaPosizione[0] = (giocatore->position[0]) - 1;
733   int nuovoScore = giocatore->score;
734   int nuovoDeploy[2];
735   nuovoDeploy[0] = giocatore->deploy[0];
736   nuovoDeploy[1] = giocatore->deploy[1];
737   if (nuovaPosizione[0] >= 0 && nuovaPosizione[0] < ROWS)
738   {
739     if (casellaVuotaOValida(grigliaDiGioco, grigliaOstacoli, nuovaPosizione))
740     {
741       spostaPlayer(grigliaDiGioco, giocatore->position, nuovaPosizione,
742                    deployCoords, packsCoords);
743     }
744     else if (colpitoOstacolo(grigliaOstacoli, nuovaPosizione))
745     {
746       *listaOstacoli =
747           addObstacle(*listaOstacoli, nuovaPosizione[0], nuovaPosizione[1]);
748       nuovaPosizione[0] = giocatore->position[0];
749       nuovaPosizione[1] = giocatore->position[1];
750     }
751     else if (colpitoPlayer(grigliaDiGioco, nuovaPosizione))
752     {
753       nuovaPosizione[0] = giocatore->position[0];
754       nuovaPosizione[1] = giocatore->position[1];
755     }
756     giocatore->deploy[0] = nuovoDeploy[0];
757     giocatore->deploy[1] = nuovoDeploy[1];
758     giocatore->score = nuovoScore;
759     giocatore->position[0] = nuovaPosizione[0];
760     giocatore->position[1] = nuovaPosizione[1];
761   }
762   return giocatore;
763 }
764
765 PlayerStats gestisciD(char grigliaDiGioco[ROWS][COLUMNS],
766                       char grigliaOstacoli[ROWS][COLUMNS],
767                       PlayerStats giocatore, Obstacles *listaOstacoli,
768                       Point deployCoords[], Point packsCoords[])
769 {
770   if (giocatore == NULL)
771   {
772     return NULL;
773   }
774   int nuovaPosizione[2];
775   nuovaPosizione[1] = giocatore->position[1] + 1;
776   nuovaPosizione[0] = giocatore->position[0];
777   int nuovoScore = giocatore->score;
778   int nuovoDeploy[2];
779   nuovoDeploy[0] = giocatore->deploy[0];
```

```
780   nuovoDeploy[1] = giocatore->deploy[1];
781   if (nuovaPosizione[1] >= 0 && nuovaPosizione[1] < COLUMNS)
782   {
783     if (casellaVuotaOValida(grigliaDiGioco, grigliaOstacoli, nuovaPosizione))
784     {
785       spostaPlayer(grigliaDiGioco, giocatore->position, nuovaPosizione,
786                    deployCoords, packsCoords);
787     }
788     else if (colpitoOstacolo(grigliaOstacoli, nuovaPosizione))
789     {
790       printf("Ostacolo\n");
791       *listaOstacoli =
792           addObstacle(*listaOstacoli, nuovaPosizione[0], nuovaPosizione[1]);
793       nuovaPosizione[0] = giocatore->position[0];
794       nuovaPosizione[1] = giocatore->position[1];
795     }
796     else if (colpitoPlayer(grigliaDiGioco, nuovaPosizione))
797     {
798       nuovaPosizione[0] = giocatore->position[0];
799       nuovaPosizione[1] = giocatore->position[1];
800     }
801     giocatore->deploy[0] = nuovoDeploy[0];
802     giocatore->deploy[1] = nuovoDeploy[1];
803     giocatore->score = nuovoScore;
804     giocatore->position[0] = nuovaPosizione[0];
805     giocatore->position[1] = nuovaPosizione[1];
806   }
807   return giocatore;
808 }
809 PlayerStats gestisciA(char grigliaDiGioco[ROWS][COLUMNS],
810                       char grigliaOstacoli[ROWS][COLUMNS],
811                       PlayerStats giocatore, Obstacles *listaOstacoli,
812                       Point deployCoords[], Point packsCoords[])
813 {
814   if (giocatore == NULL)
815     return NULL;
816   int nuovaPosizione[2];
817   nuovaPosizione[0] = giocatore->position[0];
818   // Aggiorna la posizione vecchia spostando il player avanti di 1
819   nuovaPosizione[1] = (giocatore->position[1]) - 1;
820   int nuovoScore = giocatore->score;
821   int nuovoDeploy[2];
822   nuovoDeploy[0] = giocatore->deploy[0];
823   nuovoDeploy[1] = giocatore->deploy[1];
824   if (nuovaPosizione[1] >= 0 && nuovaPosizione[1] < COLUMNS)
825   {
826     if (casellaVuotaOValida(grigliaDiGioco, grigliaOstacoli, nuovaPosizione))
827     {
828       printf("Casella vuota \n");
829       spostaPlayer(grigliaDiGioco, giocatore->position, nuovaPosizione,
830                    deployCoords, packsCoords);
831     }
832     else if (colpitoOstacolo(grigliaOstacoli, nuovaPosizione))
833     {
834       printf("Ostacolo\n");
835       *listaOstacoli =
836           addObstacle(*listaOstacoli, nuovaPosizione[0], nuovaPosizione[1]);
837       nuovaPosizione[0] = giocatore->position[0];
838       nuovaPosizione[1] = giocatore->position[1];
839     }
840     else if (colpitoPlayer(grigliaDiGioco, nuovaPosizione))
841     {
842       printf("colpito player\n");
843       nuovaPosizione[0] = giocatore->position[0];
844       nuovaPosizione[1] = giocatore->position[1];
845     }
846     giocatore->deploy[0] = nuovoDeploy[0];
847     giocatore->deploy[1] = nuovoDeploy[1];
848     giocatore->score = nuovoScore;
849     giocatore->position[0] = nuovaPosizione[0];
850     giocatore->position[1] = nuovaPosizione[1];
851   }
852   return giocatore;
853 }
```

```
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 16: Codice header utility del gioco 1

```
#include "list.h"
```

```c
#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 17: Codice sorgente utility del gioco 1

```c
#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");
```

```
11    printf("\t2 Registrati\n");
12    printf("\t3 Esci\n");
13  }
14  int colpitoOstacolo(char grigliaOstacoli[ROWS][COLUMNS], int posizione[2]) {
15    if (grigliaOstacoli[posizione[0]][posizione[1]] == 'O')
16      return 1;
17    return 0;
18  }
19  int colpitoPacco(Point packsCoords[], int posizione[2]) {
20    int i = 0;
21    for (i = 0; i < numberOfPackages; i++) {
22      if (packsCoords[i]->x == posizione[0] && packsCoords[i]->y == posizione[1])
23        return 1;
24    }
25    return 0;
26  }
27  int casellaVuotaOValida(char grigliaDiGioco[ROWS][COLUMNS],
28                          char grigliaOstacoli[ROWS][COLUMNS], int posizione[2]) {
29    if (grigliaDiGioco[posizione[0]][posizione[1]] == '-' ||
30        grigliaDiGioco[posizione[0]][posizione[1]] == '_' ||
31        grigliaDiGioco[posizione[0]][posizione[1]] == '$')
32      if (grigliaOstacoli[posizione[0]][posizione[1]] == '-' ||
33          grigliaOstacoli[posizione[0]][posizione[1]] == '_' ||
34          grigliaOstacoli[posizione[0]][posizione[1]] == '$')
35        return 1;
36    return 0;
37  }
38  int colpitoPlayer(char grigliaDiGioco[ROWS][COLUMNS], int posizione[2]) {
39    if (grigliaDiGioco[posizione[0]][posizione[1]] == 'P')
40      return 1;
41    return 0;
42  }
43  int isOnCorrectDeployPoint(PlayerStats giocatore, Point deployCoords[]) {
44    int i = 0;
45    for (i = 0; i < numberOfPackages; i++) {
46      if (giocatore->deploy[0] == deployCoords[i]->x &&
47          giocatore->deploy[1] == deployCoords[i]->y) {
48        if (deployCoords[i]->x == giocatore->position[0] &&
49            deployCoords[i]->y == giocatore->position[1])
50          return 1;
51      }
52    }
53    return 0;
54  }
55  int getHiddenPack(Point packsCoords[]) {
56    int i = 0;
57    for (i = 0; i < numberOfPackages; i++) {
58      if (packsCoords[i]->x == -1 && packsCoords[i]->y == -1)
59        return i;
60    }
61    return -1;
62  }
63  int isOnAPack(PlayerStats giocatore, Point packsCoords[]) {
64    int i = 0;
65    for (i = 0; i < numberOfPackages; i++) {
66      if (giocatore->position[0] == packsCoords[i]->x &&
67          giocatore->position[1] == packsCoords[i]->y)
68        return 1;
69    }
70    return 0;
71  }
72  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;
221     y = rand() % ROWS;
222     if (grigliaDiGioco[y][x] == '-') {
223       grigliaDiGioco[y][x] = '$';
224       packsCoords[i]->x = y;
225       packsCoords[i]->y = x;
226     } else
227       i--;
228   }
229 }
230 /*Inserisci gli ostacoli nella griglia di gioco */
231 void riempiGrigliaConGliOstacoli(char grigliaDiGioco[ROWS][COLUMNS],
232                                  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 18: 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 19: Codice sorgente utility del gioco 2

```
#include "list.h"
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#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     }
69     L->next = removePlayer(L->next, sockDes);
70   }
71   return L;
72 }
73 void freePlayers(Players L) {
74   if (L != NULL) {
75     freePlayers(L->next);
76     free(L);
77   }
78 }
79 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 20: 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 21: 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);
  write(fileDes, name, strlen(name));
  write(fileDes, " ", 1);
  write(fileDes, pwd, strlen(pwd));
  write(fileDes, "\n", 1);
  close(fileDes);
  return 1;
}
int isRegistered(char *name, char *file) {
  char command[MAX_BUF] = "cat  ";
  strcat(command, file);
  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