

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
1.1 Modalità di compilazione	1
2 Guida all'uso	1
2.1 Server	1
2.2 Client	1
3 Comunicazione tra client e server	2
3.1 Configurazione del server	2
3.2 Configurazione del client	4
3.3 Comunicazione tra client e server	5
3.3.1 Esempio: la prima comunicazione	5
4 Comunicazione durante la partita	6
4.1 Funzione core del server	6
4.2 Funzione core del client	6
A Codici sorgente	8
A.1 Codice sorgente del client	8
A.2 Codice sorgente del server	12
A.3 Codice sorgente boardUtility	20
A.4 Codice sorgente list	27
A.5 Codice sorgente parser	29

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 manualmente 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* si 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 client 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 comparirà 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 del gioco sia 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 protocolli PF_INET, con tipo di trasmissione dati SOCK_STREAM e con protocollo TCP. Mostriamo di seguito il codice sorgente:

Listato 1: Configurazione indirizzo del server

```
1 struct sockaddr_in configuraIndirizzo() {
2     struct sockaddr_in mio_indirizzo;
3     mio_indirizzo.sin_family = AF_INET;
4     mio_indirizzo.sin_port = htons(5200);
5     mio_indirizzo.sin_addr.s_addr = htonl(INADDR_ANY);
6     printf("Indirizzo socket configurato\n");
7     return mio_indirizzo;
8 }
```

Listato 2: Configurazione socket del server

```
1 void configuraSocket(struct sockaddr_in mio_indirizzo) {
2     if ((socketDesc = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
3         perror("Impossibile creare socket");
4         exit(-1);
5     }
6     if (setsockopt(socketDesc, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) <
7         0)
8         perror("Impossibile impostare il riutilizzo dell'indirizzo ip e della "
9             "porta\n");
10    if ((bind(socketDesc, (struct sockaddr *)&mio_indirizzo,
11        sizeof(mio_indirizzo))) < 0) {
12        perror("Impossibile effettuare bind");
13        exit(-1);
14    }
15 }
```

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 void startListening() {
2     pthread_t tid;
3     int clientDesc;
4     int *puntClientDesc;
5     while (1 == 1) {
6         if (listen(socketDesc, 10) < 0)
7             perror("Impossibile mettersi in ascolto"), exit(-1);
8         printf("In ascolto...\n");
9         if ((clientDesc = accept(socketDesc, NULL, NULL)) < 0) {
10             perror("Impossibile effettuare connessione\n");
11             exit(-1);
12         }
13         printf("Nuovo client connesso\n");
14         puntClientDesc = (int *)malloc(sizeof(int));
15         *puntClientDesc = clientDesc;
16         pthread_create(&tid, NULL, gestisci, (void *)puntClientDesc);
17     }
18     close(clientDesc);
19     quitServer();
20 }
```

In particolare al rigo 19 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.

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 ritornare soltanto il primo indirizzo convertito in ascii.

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 relative funzioni di gestione che entrano in esecuzione subito dopo aver stabilito la connessione tra client e server.

Listato 6: Prima comunicazione del server

```
1 void *gestisci(void *descriptor) {
2     int bufferReceive[2] = {1};
3     int client_sd = *(int *)descriptor;
4     int continua = 1;
5     char name[MAX_BUF];
6     while (continua) {
7         read(client_sd, bufferReceive, sizeof(bufferReceive));
8         if (bufferReceive[0] == 2)
9             registraClient(client_sd);
10        else if (bufferReceive[0] == 1)
11            if (tryLogin(client_sd, name)) {
12                play(client_sd, name);
13                continua = 0;
14            } else if (bufferReceive[0] == 3)
15                disconnettiClient(client_sd);
16        else {
17            printf("Input invalido, uscita...\n");
18            disconnettiClient(client_sd);
19        }
20    }
21    pthread_exit(0);
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 infinito nel quale il server invia al client tre informazioni importanti:

- La griglia di gioco (Rigo 26)
- Il player con le relative informazioni (Rigo 28 a 31)
- Il timer (Rigo 32)
- Un messaggio che notifica al client se è iniziato un nuovo turno oppure no

Dopodichè il thread del server rimane in attesa di ricevere l'input del client per spostare il giocatore sulla mappa tramite la relativa funzione. (Rigo 34)

Oltre questo, la funzione play del server si occupa anche di generare la posizione del player appena entra in partita e generare la nuova posizione del player (Rigo 52 a 66) quando viene effettuato il cambio di mappa (allo scadere del tempo o alla raggiunta del massimo punteggio).

4.2 Funzione core del client

Dall'altro lato, la funzione play del client, è stata mantenuta volutamente semplice. Lo scopo del client è unicamente quello di ricevere i dati forniti dal server, stampare la mappa di gioco e leggere l'input da inviare al server. Unica eccezione è il rigo 30 del client che non richiede la ricezione di ulteriori dati dal server: al rigo 30, infatti si avvia la procedura di disconnessione del client .

Listato 8: Funzione play del server

```

1 void play(int clientDesc, char name[]) {
2     int true = 1;
3     int turnoFinito = 0;
4     int turnoGiocatore = turno;
5     int posizione[2];
6     int destinazione[2] = {-1, -1};
7     PlayerStats giocatore = initStats(destinazione, 0, posizione, 0);
8     Obstacles listaOstacoli = NULL;
9     char inputFromClient;
10    if (timer != 0) {
11        inserisciPlayerNellaGrigliaInPosizioneCasuale(
12            grigliaDiGiocoConPacchiSenzaOstacoli, grigliaOstacoliSenzaPacchi,
13            giocatore->position);
14        playerGenerati++;
15    }
16    while (true) {
17        if (clientDisconnesso(clientDesc)) {
18            freeObstacles(listaOstacoli);
19            disconnettiClient(clientDesc);
20            return;
21        }
22        char grigliaTmp[ROWS][COLUMNS];
23        clonaGriglia(grigliaTmp, grigliaDiGiocoConPacchiSenzaOstacoli);
24        mergeGridAndList(grigliaTmp, listaOstacoli);
25        // invia la griglia
26        write(clientDesc, grigliaTmp, sizeof(grigliaTmp));
27        // invia la struttura del player
28        write(clientDesc, giocatore->deploy, sizeof(giocatore->deploy));
29        write(clientDesc, giocatore->position, sizeof(giocatore->position));
30        write(clientDesc, &giocatore->score, sizeof(giocatore->score));
31        write(clientDesc, &giocatore->hasApack, sizeof(giocatore->hasApack));
32        sendTimerValue(clientDesc);
33        // legge l'input
34        if (read(clientDesc, &inputFromClient, sizeof(char)) > 0)
35            numMosse++;
36        if (inputFromClient == 'e' || inputFromClient == 'E') {
37            freeObstacles(listaOstacoli);
38            listaOstacoli = NULL;
39            disconnettiClient(clientDesc);
40        } else if (inputFromClient == 't' || inputFromClient == 'T') {
41            write(clientDesc, &turnoFinito, sizeof(int));
42            sendTimerValue(clientDesc);
43        } else if (inputFromClient == 'l' || inputFromClient == 'L') {
44            write(clientDesc, &turnoFinito, sizeof(int));
45            sendPlayerList(clientDesc);
46        } else if (turnoGiocatore == turno) {
47            write(clientDesc, &turnoFinito, sizeof(int));
48            giocatore =
49                gestisciInput(grigliaDiGiocoConPacchiSenzaOstacoli,
50                    grigliaOstacoliSenzaPacchi, inputFromClient, giocatore,
51                    &listaOstacoli, deployCoords, packsCoords, name);
52        } else {
53            turnoFinito = 1;
54            write(clientDesc, &turnoFinito, sizeof(int));
55            freeObstacles(listaOstacoli);
56            listaOstacoli = NULL;
57            inserisciPlayerNellaGrigliaInPosizioneCasuale(
58                grigliaDiGiocoConPacchiSenzaOstacoli, grigliaOstacoliSenzaPacchi,
59                giocatore->position);
60            giocatore->score = 0;
61            giocatore->hasApack = 0;
62            giocatore->deploy[0] = -1;
63            giocatore->deploy[1] = -1;
64            turnoGiocatore = turno;
65            turnoFinito = 0;
66            playerGenerati++;
67        }
68    }
69 }

```

Listato 9: Funzione play del client

```

1 void play() {
2     PlayerStats giocatore = NULL;
3     int score, deploy[2], position[2], timer;
4     int turnoFinito = 0;
5     int exitFlag = 0, hasApack = 0;
6     while (!exitFlag) {
7         if (serverCaduto())
8             serverCrashHandler();
9         if (read(socketDesc, grigliaDiGioco, sizeof(grigliaDiGioco)) < 1)
10            printf("Impossibile comunicare con il server\n"), exit(-1);
11        if (read(socketDesc, deploy, sizeof(deploy)) < 1)
12            printf("Impossibile comunicare con il server\n"), exit(-1);
13        if (read(socketDesc, position, sizeof(position)) < 1)
14            printf("Impossibile comunicare con il server\n"), exit(-1);
15        if (read(socketDesc, &score, sizeof(score)) < 1)
16            printf("Impossibile comunicare con il server\n"), exit(-1);
17        if (read(socketDesc, &hasApack, sizeof(hasApack)) < 1)
18            printf("Impossibile comunicare con il server\n"), exit(-1);
19        timer = getTimer();
20        giocatore = initStats(deploy, score, position, hasApack);
21        printGrid(grigliaDiGioco, giocatore);
22        char send = getUserInput();
23        write(socketDesc, &send, sizeof(char));
24        read(socketDesc, &turnoFinito, sizeof(turnoFinito));
25        if (turnoFinito) {
26            system("clear");
27            printf("Turno finito\n");
28            sleep(1);
29        } else {
30            if (send == 'e' || send == 'E')
31                printf("Disconnessione in corso...\n"), exit(0);
32            if (send == 't' || send == 'T')
33                printTimer();
34            else if (send == 'l' || send == 'L')
35                printPlayerList();
36        }
37    }
38 }

```

A Codici sorgente

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

A.1 Codice sorgente del client

Listato 10: Codice sorgente del client

```

1 #include "boardUtility.h"
2 #include "list.h"
3 #include "parser.h"
4 #include <arpa/inet.h>
5 #include <fcntl.h>
6 #include <netdb.h>
7 #include <netinet/in.h> //conversioni
8 #include <netinet/in.h>
9 #include <netinet/ip.h> //struttura
10 #include <pthread.h>
11 #include <signal.h>
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <string.h>
15 #include <sys/socket.h>
16 #include <sys/stat.h>
17 #include <sys/types.h>
18 #include <time.h>
19 #include <unistd.h>

```

```

20
21 void printPlayerList();
22 int getTimer();
23 void printTimer();
24 void play();
25 int tryLogin();
26 void printMenu();
27 int connettiAlServer(char **argv);
28 char *ipResolver(char **argv);
29 int registrati();
30 int gestisci();
31 char getUserInput();
32 void clientCrashHandler();
33 void serverCrashHandler();
34 int serverCaduto();
35 void esciDalServer();
36 int isCorrect(char);
37
38 int socketDesc;
39 char grigliaDiGioco[ROWS][COLUMNS];
40
41 int main(int argc, char **argv) {
42     signal(SIGINT, clientCrashHandler); /* CTRL-C */
43     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         timer = getTimer();
130         giocatore = initStats(deploy, score, position, hasApack);
131         printGrid(grigliaDiGioco, giocatore);
132         char send = getUserInput();
133         write(socketDesc, &send, sizeof(char));
134         read(socketDesc, &turnoFinito, sizeof(turnoFinito));
135         if (turnoFinito) {
136             system("clear");
137             printf("Turno finito\n");
138             sleep(1);
139         } else {
140             if (send == 'e' || send == 'E')
141                 printf("Disconnessione in corso...\n"), exit(0);
142             if (send == 't' || send == 'T')
143                 printTimer();
144             else if (send == 'l' || send == 'L')
145                 printPlayerList();
146         }
147     }
148 }
149 void printPlayerList() {
150     system("clear");
151     int lunghezza = 0;
152     char buffer[100];
153     int continua = 1;
154     int number = 1;
155     fprintf(stdout, "Lista dei player: \n");
156     if (!serverCaduto(socketDesc)) {
157         read(socketDesc, &continua, sizeof(continua));
158         while (continua) {
159             read(socketDesc, &lunghezza, sizeof(lunghezza));
160             read(socketDesc, buffer, lunghezza);
161             buffer[lunghezza] = '\0';
162             fprintf(stdout, "%d %s\n", number, buffer);
163             continua--;
164             number++;
165         }
166         sleep(1);
167     }

```

```

168 }
169 void printTimer() {
170     int timer;
171     if (!serverCaduto(socketDesc)) {
172         read(socketDesc, &timer, sizeof(timer));
173         printf("\t\tTempo restante: %d...\n", timer);
174         sleep(1);
175     }
176 }
177 int getTimer() {
178     int timer;
179     if (!serverCaduto(socketDesc))
180         read(socketDesc, &timer, sizeof(timer));
181     return timer;
182 }
183 int tryLogin() {
184     int msg = 1;
185     write(socketDesc, &msg, sizeof(int));
186     system("clear");
187     printf("Inserisci i dati per il Login\n");
188     char username[20];
189     char password[20];
190     printf("Inserisci nome utente(MAX 20 caratteri): ");
191     scanf("%s", username);
192     printf("\nInserisci password(MAX 20 caratteri):");
193     scanf("%s", password);
194     int dimUname = strlen(username), dimPwd = strlen(password);
195     if (write(socketDesc, &dimUname, sizeof(dimUname)) < 0)
196         return 0;
197     if (write(socketDesc, &dimPwd, sizeof(dimPwd)) < 0)
198         return 0;
199     if (write(socketDesc, username, dimUname) < 0)
200         return 0;
201     if (write(socketDesc, password, dimPwd) < 0)
202         return 0;
203     char validate;
204     int ret;
205     read(socketDesc, &validate, 1);
206     if (validate == 'y') {
207         ret = 1;
208         printf("Accesso effettuato\n");
209     } else if (validate == 'n') {
210         printf("Credenziali Errate o Login già effettuato\n");
211         ret = 0;
212     }
213     sleep(1);
214     return ret;
215 }
216 int registrati() {
217     int msg = 2;
218     write(socketDesc, &msg, sizeof(int));
219     char username[20];
220     char password[20];
221     system("clear");
222     printf("Inserisci nome utente(MAX 20 caratteri): ");
223     scanf("%s", username);
224     printf("\nInserisci password(MAX 20 caratteri):");
225     scanf("%s", password);
226     int dimUname = strlen(username), dimPwd = strlen(password);
227     if (write(socketDesc, &dimUname, sizeof(dimUname)) < 0)
228         return 0;
229     if (write(socketDesc, &dimPwd, sizeof(dimPwd)) < 0)
230         return 0;
231     if (write(socketDesc, username, dimUname) < 0)
232         return 0;
233     if (write(socketDesc, password, dimPwd) < 0)
234         return 0;
235     char validate;
236     int ret;
237     read(socketDesc, &validate, sizeof(char));
238     if (validate == 'y') {
239         ret = 1;
240         printf("Registrato con successo\n");
241     }

```

```

242     if (validate == 'n') {
243         ret = 0;
244         printf("Registrazione fallita\n");
245     }
246     sleep(1);
247     return ret;
248 }
249 char *ipResolver(char **argv) {
250     char *ipAddress;
251     struct hostent *hp;
252     hp = gethostbyname(argv[1]);
253     if (!hp) {
254         perror("Impossibile risolvere l'indirizzo ip\n");
255         sleep(1);
256         exit(-1);
257     }
258     printf("Address:\t%s\n", inet_ntoa(*(struct in_addr *)hp->h_addr_list[0]));
259     return inet_ntoa(*(struct in_addr *)hp->h_addr_list[0]);
260 }
261 void clientCrashHandler() {
262     int msg = 3;
263     int rec = 0;
264     printf("\nChiusura client...\n");
265     do {
266         write(socketDesc, &msg, sizeof(int));
267         read(socketDesc, &rec, sizeof(int));
268     } while (rec == 0);
269     close(socketDesc);
270     signal(SIGINT, SIG_IGN);
271     signal(SIGQUIT, SIG_IGN);
272     signal(SIGTERM, SIG_IGN);
273     signal(SIGTSTP, SIG_IGN);
274     exit(0);
275 }
276 void serverCrashHandler() {
277     system("clear");
278     printf("Il server á stato spento o á irraggiungibile\n");
279     close(socketDesc);
280     signal(SIGPIPE, SIG_IGN);
281     premiEnterPerContinuare();
282     exit(0);
283 }
284 char getUserInput() {
285     fflush(stdin);
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 11: Codice sorgente del server

```

1
2 #include "boardUtility.h"
3 #include "list.h"
4 #include "parser.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 struct argsToSend {
21     char *userName;
22     int flag;
23 };
24 typedef struct argsToSend *Args;
25 void sendPlayerList(int clientDesc);
26 PlayerStats gestisciC(char grigliaDiGioco[ROWS][COLUMNS], PlayerStats giocatore,
27     Point deployCoords[], Point packsCoords[], char name[]);
28 PlayerStats gestisciInput(char grigliaDiGioco[ROWS][COLUMNS],
29     char grigliaOstacoli[ROWS][COLUMNS], char input,
30     PlayerStats giocatore, Obstacles *listaOstacoli,
31     Point deployCoords[], Point packsCoords[],
32     char name[]);
33 void clonaGriglia(char destinazione[ROWS][COLUMNS], char source[ROWS][COLUMNS]);
34 int almenoUnClientConnesso();
35 int valoreTimerValido();
36 int almenoUnPlayerGenerato();
37 int almenoUnaMossaFatta();
38 void sendTimerValue(int clientDesc);
39 void startProceduraGenrazioneMappa();
40 void *threadGenerazioneMappa(void *args);
41 void *fileWriter(void *);
42 int tryLogin(int clientDesc, char name[]);
43 void disconnettiClient(int);
44 int registraClient(int);
45 void *timer(void *args);
46 void *gestisci(void *descriptor);
47 void quitServer();
48 void clientCrashHandler(int signalNum);
49 void startTimer();
50 void configuraSocket(struct sockaddr_in mio_indirizzo);
51 struct sockaddr_in configuraIndirizzo();
52 void startListening();
53 int clientDisconnesso(int clientSocket);
54 void play(int clientDesc, char name[]);
55
56 char grigliaDiGiocoConPacchiSenzaOstacoli[ROWS][COLUMNS];
57 char grigliaOstacoliSenzaPacchi[ROWS][COLUMNS];
58 int numeroClientLoggati = 0;
59 int playerGenerati = 0;
60 int timerCount = TIME_LIMIT_IN_SECONDS;
61 int turno = 0;
62 pthread_t tidTimer;
63 pthread_t tidGeneratoreMappa;
64 int socketDesc;
65 Players onLineUsers = NULL;
66 char *users;
67 int scoreMassimo = 0;
68 int numMosse = 0;
69 Point deployCoords[numberOfPackages];
70 Point packsCoords[numberOfPackages];
71 pthread_mutex_t LogMutex = PTHREAD_MUTEX_INITIALIZER;
72 pthread_mutex_t RegMutex = PTHREAD_MUTEX_INITIALIZER;
73 pthread_mutex_t PlayerMutex = PTHREAD_MUTEX_INITIALIZER;
74
75 int main(int argc, char **argv) {
76     if (argc != 2) {
77         printf("Wrong parameters number(Usage: ./server usersFile)\n");
78         exit(-1);
79     } else if (strcmp(argv[1], "Log") == 0) {
80         printf("Cannot use the Log file as a UserList \n");
81         exit(-1);
82     }
83     users = argv[1];
84     struct sockaddr_in mio_indirizzo = configuraIndirizzo();
85     configuraSocket(mio_indirizzo);
86     signal(SIGPIPE, clientCrashHandler);
87     signal(SIGINT, quitServer);
88     signal(SIGHUP, quitServer);
89     startTimer();

```



```

90     inizializzaGiocoSenzaPlayer(grigliaDiGiocoConPacchiSenzaOstacoli,
91                                 grigliaOstacoliSenzaPacchi, packsCoords);
92     generaPosizioniRaccolta(grigliaDiGiocoConPacchiSenzaOstacoli,
93                             grigliaOstacoliSenzaPacchi, deployCoords);
94     startListening();
95     return 0;
96 }
97 void startListening() {
98     pthread_t tid;
99     int clientDesc;
100    int *puntClientDesc;
101    while (1 == 1) {
102        if (listen(socketDesc, 10) < 0)
103            perror("Impossibile mettersi in ascolto"), exit(-1);
104        printf("In ascolto..\n");
105        if ((clientDesc = accept(socketDesc, NULL, NULL)) < 0) {
106            perror("Impossibile effettuare connessione\n");
107            exit(-1);
108        }
109        printf("Nuovo client connesso\n");
110        puntClientDesc = (int *)malloc(sizeof(int));
111        *puntClientDesc = clientDesc;
112        pthread_create(&tid, NULL, gestisci, (void *)puntClientDesc);
113    }
114    close(clientDesc);
115    quitServer();
116 }
117 struct sockaddr_in configuraIndirizzo() {
118     struct sockaddr_in mio_indirizzo;
119     mio_indirizzo.sin_family = AF_INET;
120     mio_indirizzo.sin_port = htons(5200);
121     mio_indirizzo.sin_addr.s_addr = htonl(INADDR_ANY);
122     printf("Indirizzo socket configurato\n");
123     return mio_indirizzo;
124 }
125 void startProceduraGenerazioneMappa() {
126     printf("Inizio procedura generazione mappa\n");
127     pthread_create(&tidGeneratoreMappa, NULL, threadGenerazioneMappa, NULL);
128 }
129 void startTimer() {
130     printf("Thread timer avviato\n");
131     pthread_create(&tidTimer, NULL, timer, NULL);
132 }
133 int tryLogin(int clientDesc, char name[]) {
134     char *userName = (char *)calloc(MAX_BUF, 1);
135     char *password = (char *)calloc(MAX_BUF, 1);
136     int dimName, dimPwd;
137     read(clientDesc, &dimName, sizeof(int));
138     read(clientDesc, &dimPwd, sizeof(int));
139     read(clientDesc, userName, dimName);
140     read(clientDesc, password, dimPwd);
141     int ret = 0;
142     if (validateLogin(userName, password, users) &&
143         !isAlreadyLogged(onLineUsers, userName)) {
144         ret = 1;
145         numeroClientLoggati++;
146         write(clientDesc, "y", 1);
147         strcpy(name, userName);
148         Args args = (Args)malloc(sizeof(struct argsToSend));
149         args->userName = (char *)calloc(MAX_BUF, 1);
150         strcpy(args->userName, name);
151         args->flag = 0;
152         pthread_t tid;
153         pthread_create(&tid, NULL, fileWriter, (void *)args);
154         printf("Nuovo client loggato, client loggati : %d\n", numeroClientLoggati);
155         pthread_mutex_lock(&PlayerMutex);
156         onLineUsers = addPlayer(onLineUsers, userName, clientDesc);
157         pthread_mutex_unlock(&PlayerMutex);
158         printPlayers(onLineUsers);
159         printf("\n");
160     } else {
161         write(clientDesc, "n", 1);
162     }
163     return ret;

```

```

164 }
165 void *gestisci(void *descriptor) {
166     int bufferReceive[2] = {1};
167     int client_sd = *(int *)descriptor;
168     int continua = 1;
169     char name[MAX_BUF];
170     while (continua) {
171         read(client_sd, bufferReceive, sizeof(bufferReceive));
172         if (bufferReceive[0] == 2)
173             registraClient(client_sd);
174         else if (bufferReceive[0] == 1)
175             if (tryLogin(client_sd, name)) {
176                 play(client_sd, name);
177                 continua = 0;
178             } else if (bufferReceive[0] == 3)
179                 disconnettiClient(client_sd);
180             else {
181                 printf("Input invalido, uscita...\n");
182                 disconnettiClient(client_sd);
183             }
184     }
185     pthread_exit(0);
186 }
187 void play(int clientDesc, char name[]) {
188     int true = 1;
189     int turnoFinito = 0;
190     int turnoGiocatore = turno;
191     int posizione[2];
192     int destinazione[2] = {-1, -1};
193     PlayerStats giocatore = initStats(destinazione, 0, posizione, 0);
194     Obstacles listaOstacoli = NULL;
195     char inputFromClient;
196     if (timer != 0) {
197         inserisciPlayerNellaGrigliaInPosizioneCasuale(
198             grigliaDiGiocoConPacchiSenzaOstacoli, grigliaOstacoliSenzaPacchi,
199             giocatore->position);
200         playerGenerati++;
201     }
202     while (true) {
203         if (clientDisconnesso(clientDesc)) {
204             freeObstacles(listaOstacoli);
205             disconnettiClient(clientDesc);
206             return;
207         }
208         char grigliaTmp[ROWS][COLUMNS];
209         clonaGriglia(grigliaTmp, grigliaDiGiocoConPacchiSenzaOstacoli);
210         mergeGridAndList(grigliaTmp, listaOstacoli);
211         // invia la griglia
212         write(clientDesc, grigliaTmp, sizeof(grigliaTmp));
213         // invia la struttura del player
214         write(clientDesc, giocatore->deploy, sizeof(giocatore->deploy));
215         write(clientDesc, giocatore->position, sizeof(giocatore->position));
216         write(clientDesc, &giocatore->score, sizeof(giocatore->score));
217         write(clientDesc, &giocatore->hasApack, sizeof(giocatore->hasApack));
218         sendTimerValue(clientDesc);
219         // legge l'input
220         if (read(clientDesc, &inputFromClient, sizeof(char)) > 0)
221             numMosse++;
222         if (inputFromClient == 'e' || inputFromClient == 'E') {
223             freeObstacles(listaOstacoli);
224             listaOstacoli = NULL;
225             disconnettiClient(clientDesc);
226         } else if (inputFromClient == 't' || inputFromClient == 'T') {
227             write(clientDesc, &turnoFinito, sizeof(int));
228             sendTimerValue(clientDesc);
229         } else if (inputFromClient == 'l' || inputFromClient == 'L') {
230             write(clientDesc, &turnoFinito, sizeof(int));
231             sendPlayerList(clientDesc);
232         } else if (turnoGiocatore == turno) {
233             write(clientDesc, &turnoFinito, sizeof(int));
234             giocatore =
235                 gestisciInput(grigliaDiGiocoConPacchiSenzaOstacoli,
236                             grigliaOstacoliSenzaPacchi, inputFromClient, giocatore,
237                             &listaOstacoli, deployCoords, packsCoords, name);

```

```

238     } else {
239         turnoFinito = 1;
240         write(clientDesc, &turnoFinito, sizeof(int));
241         freeObstacles(listaOstacoli);
242         listaOstacoli = NULL;
243         inserisciPlayerNellaGrigliaInPosizioneCasuale(
244             grigliaDiGiocoConPacchiSenzaOstacoli, grigliaOstacoliSenzaPacchi,
245             giocatore->position);
246         giocatore->score = 0;
247         giocatore->hasApack = 0;
248         giocatore->deploy[0] = -1;
249         giocatore->deploy[1] = -1;
250         turnoGiocatore = turno;
251         turnoFinito = 0;
252         playerGenerati++;
253     }
254 }
255 }
256 void sendTimerValue(int clientDesc) {
257     if (!clientDisconnesso(clientDesc))
258         write(clientDesc, &timerCount, sizeof(timerCount));
259 }
260 void clonaGriglia(char destinazione[ROWS][COLUMNS],
261                  char source[ROWS][COLUMNS]) {
262     int i = 0, j = 0;
263     for (i = 0; i < ROWS; i++) {
264         for (j = 0; j < COLUMNS; j++) {
265             destinazione[i][j] = source[i][j];
266         }
267     }
268 }
269 void clientCrashHandler(int signalNum) {
270     char msg[0];
271     int socketClientCrashato;
272     int flag = 1;
273     // TODO eliminare la lista degli ostacoli dell'utente
274     if (onLineUsers != NULL) {
275         Players prec = onLineUsers;
276         Players top = prec->next;
277         while (top != NULL && flag) {
278             if (write(top->sockDes, msg, sizeof(msg)) < 0) {
279                 socketClientCrashato = top->sockDes;
280                 printPlayers(onLineUsers);
281                 disconnettiClient(socketClientCrashato);
282                 flag = 0;
283             }
284             top = top->next;
285         }
286     }
287     signal(SIGPIPE, SIG_IGN);
288 }
289 void disconnettiClient(int clientDescriptor) {
290     if (numeroClientLoggati > 0)
291         numeroClientLoggati--;
292     pthread_mutex_lock(&PlayerMutex);
293     onLineUsers = removePlayer(onLineUsers, clientDescriptor);
294     pthread_mutex_unlock(&PlayerMutex);
295     printPlayers(onLineUsers);
296     int msg = 1;
297     printf("Client disconnesso (client attualmente loggati: %d)\n",
298           numeroClientLoggati);
299     write(clientDescriptor, &msg, sizeof(msg));
300     close(clientDescriptor);
301 }
302 int clientDisconnesso(int clientSocket) {
303     char msg[1] = {'u'}; // UP?
304     if (write(clientSocket, msg, sizeof(msg)) < 0)
305         return 1;
306     if (read(clientSocket, msg, sizeof(char)) < 0)
307         return 1;
308     else
309         return 0;
310 }
311 int registraClient(int clientDesc) {

```

```

312 char *userName = (char *)calloc(MAX_BUF, 1);
313 char *password = (char *)calloc(MAX_BUF, 1);
314 int dimName, dimPwd;
315 read(clientDesc, &dimName, sizeof(int));
316 read(clientDesc, &dimPwd, sizeof(int));
317 read(clientDesc, userName, dimName);
318 read(clientDesc, password, dimPwd);
319 pthread_mutex_lock(&RegMutex);
320 int ret = appendPlayer(userName, password, users);
321 pthread_mutex_unlock(&RegMutex);
322 char risposta;
323 if (!ret) {
324     risposta = 'n';
325     write(clientDesc, &risposta, sizeof(char));
326     printf("Impossibile registrare utente, riprovare\n");
327 } else {
328     risposta = 'y';
329     write(clientDesc, &risposta, sizeof(char));
330     printf("Utente registrato con successo\n");
331 }
332 return ret;
333 }
334 void quitServer() {
335     printf("Chiusura server in corso..\n");
336     close(socketDesc);
337     exit(-1);
338 }
339 void *threadGenerazioneMappa(void *args) {
340     fprintf(stdout, "Rigenerazione mappa\n");
341     inizializzaGrigliaVuota(grigliaDiGiocoConPacchiSenzaOstacoli);
342     generaPosizioniRaccolta(grigliaDiGiocoConPacchiSenzaOstacoli,
343                             grigliaOstacoliSenzaPacchi, deployCoords);
344     riempiGrigliaConPacchiInPosizioniGenerateCasualmente(
345         grigliaDiGiocoConPacchiSenzaOstacoli, packsCoords);
346     generaPosizioneOstacoli(grigliaDiGiocoConPacchiSenzaOstacoli,
347                             grigliaOstacoliSenzaPacchi);
348     timerCount = TIME_LIMIT_IN_SECONDS;
349     printf("Mappa generata\n");
350     pthread_exit(NULL);
351 }
352 int almenoUnaMossaFatta() {
353     if (numMosse > 0)
354         return 1;
355     return 0;
356 }
357 int almenoUnClientConnesso() {
358     if (numeroClientLoggati > 0)
359         return 1;
360     return 0;
361 }
362 int valoreTimerValido() {
363     if (timerCount > 0 && timerCount <= TIME_LIMIT_IN_SECONDS)
364         return 1;
365     return 0;
366 }
367 int almenoUnPlayerGenerato() {
368     if (playerGenerati > 0)
369         return 1;
370     return 0;
371 }
372 void *timer(void *args) {
373     int cambiato = 1;
374     while (1) {
375         if (almenoUnClientConnesso() && valoreTimerValido() &&
376             almenoUnPlayerGenerato() && almenoUnaMossaFatta()) {
377             cambiato = 1;
378             sleep(1);
379             timerCount--;
380             fprintf(stdout, "Time left: %d\n", timerCount);
381         } else if (numeroClientLoggati == 0) {
382             timerCount = TIME_LIMIT_IN_SECONDS;
383             if (cambiato) {
384                 fprintf(stdout, "Time left: %d\n", timerCount);
385                 cambiato = 0;

```

```

386     }
387 }
388 if (timerCount == 0 || scoreMassimo == packageLimitNumber) {
389     playerGenerati = 0;
390     numMosse = 0;
391     printf("Reset timer e generazione nuova mappa..\n");
392     startProceduraGenrazioneMappa();
393     pthread_join(tidGeneratoreMappa, NULL);
394     turno++;
395     timerCount = TIME_LIMIT_IN_SECONDS;
396 }
397 }
398 }
399
400 void configuraSocket(struct sockaddr_in mio_indirizzo) {
401     if ((socketDesc = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
402         perror("Impossibile creare socket");
403         exit(-1);
404     }
405     if (setsockopt(socketDesc, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) <
406         0)
407         perror("Impossibile impostare il riutilizzo dell'indirizzo ip e della "
408             "porta\n");
409     if ((bind(socketDesc, (struct sockaddr *)&mio_indirizzo,
410         sizeof(mio_indirizzo))) < 0) {
411         perror("Impossibile effettuare bind");
412         exit(-1);
413     }
414 }
415
416 PlayerStats gestisciInput(char grigliaDiGioco[ROWS][COLUMNS],
417     char grigliaOstacoli[ROWS][COLUMNS], char input,
418     PlayerStats giocatore, Obstacles *listaOstacoli,
419     Point deployCoords[], Point packsCoords[],
420     char name[]) {
421     if (giocatore == NULL) {
422         return NULL;
423     }
424     if (input == 'w' || input == 'W') {
425         giocatore = gestisciW(grigliaDiGioco, grigliaOstacoli, giocatore,
426             listaOstacoli, deployCoords, packsCoords);
427     } else if (input == 's' || input == 'S') {
428         giocatore = gestisciS(grigliaDiGioco, grigliaOstacoli, giocatore,
429             listaOstacoli, deployCoords, packsCoords);
430     } else if (input == 'a' || input == 'A') {
431         giocatore = gestisciA(grigliaDiGioco, grigliaOstacoli, giocatore,
432             listaOstacoli, deployCoords, packsCoords);
433     } else if (input == 'd' || input == 'D') {
434         giocatore = gestisciD(grigliaDiGioco, grigliaOstacoli, giocatore,
435             listaOstacoli, deployCoords, packsCoords);
436     } else if (input == 'p' || input == 'P') {
437         giocatore = gestisciP(grigliaDiGioco, giocatore, deployCoords, packsCoords);
438     } else if (input == 'c' || input == 'C') {
439         giocatore =
440             gestisciC(grigliaDiGioco, giocatore, deployCoords, packsCoords, name);
441     }
442
443     // aggiorna la posizione dell'utente
444     return giocatore;
445 }
446
447 PlayerStats gestisciC(char grigliaDiGioco[ROWS][COLUMNS], PlayerStats giocatore,
448     Point deployCoords[], Point packsCoords[], char name[]) {
449     pthread_t tid;
450     if (giocatore->hasApack == 0) {
451         return giocatore;
452     } else {
453         if (isOnCorrectDeployPoint(giocatore, deployCoords)) {
454             Args args = (Args)malloc(sizeof(struct argsToSend));
455             args->userName = (char *)calloc(MAX_BUF, 1);
456             strcpy(args->userName, name);
457             args->flag = 1;
458             pthread_create(&tid, NULL, fileWriter, (void *)args);
459             giocatore->score += 10;

```

```

460     if (giocatore->score > scoreMassimo)
461         scoreMassimo = giocatore->score;
462     giocatore->deploy[0] = -1;
463     giocatore->deploy[1] = -1;
464     giocatore->hasApack = 0;
465 } else {
466     if (!isOnAPack(giocatore, packsCoords) &&
467         !isOnADeployPoint(giocatore, deployCoords)) {
468         int index = getHiddenPack(packsCoords);
469         if (index >= 0) {
470             packsCoords[index]->x = giocatore->position[0];
471             packsCoords[index]->y = giocatore->position[1];
472             giocatore->hasApack = 0;
473         }
474     } else
475         return giocatore;
476 }
477 }
478 return giocatore;
479 }
480
481 void sendPlayerList(int clientDesc) {
482     int lunghezza = 0;
483     char name[100];
484     Players tmp = onLineUsers;
485     int numeroClientLoggati = dimensioneLista(tmp);
486     printf("%d ", numeroClientLoggati);
487     if (!clientDisconnesso(clientDesc)) {
488         write(clientDesc, &numeroClientLoggati, sizeof(numeroClientLoggati));
489         while (numeroClientLoggati > 0 && tmp != NULL) {
490             strcpy(name, tmp->name);
491             lunghezza = strlen(tmp->name);
492             write(clientDesc, &lunghezza, sizeof(lunghezza));
493             write(clientDesc, name, lunghezza);
494             tmp = tmp->next;
495             numeroClientLoggati--;
496         }
497     }
498 }
499
500 void *fileWriter(void *args) {
501     int fDes = open("Log", O_RDWR | O_CREAT | O_APPEND, S_IWUSR | S_IRUSR);
502     if (fDes < 0) {
503         perror("Error while opening log file");
504         exit(-1);
505     }
506     Args info = (Args)args;
507     time_t t = time(NULL);
508     struct tm *infoTime = localtime(&t);
509     char toPrint[64];
510     strftime(toPrint, sizeof(toPrint), "%X %x", infoTime);
511     if (info->flag == 1) {
512         char message[MAX_BUF] = "Pack delivered by \";
513         strcat(message, info->userName);
514         char at[] = "\" at ";
515         strcat(message, at);
516         strcat(message, toPrint);
517         strcat(message, "\n");
518         pthread_mutex_lock(&LogMutex);
519         write(fDes, message, strlen(message));
520         pthread_mutex_unlock(&LogMutex);
521     } else if (info->flag == 0) {
522         char message[MAX_BUF] = "\"";
523         strcat(message, info->userName);
524         strcat(message, "\" connected at ");
525         strcat(message, toPrint);
526         strcat(message, "\n");
527         pthread_mutex_lock(&LogMutex);
528         write(fDes, message, strlen(message));
529         pthread_mutex_unlock(&LogMutex);
530     }
531     close(fDes);
532     pthread_exit(NULL);
533 }

```

A.3 Codice sorgente boardUtility

Listato 12: Codice header utility del gioco 1

```
1  #include "list.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5  #include <unistd.h>
6  #define ROWS 10
7  #define COLUMNS 30
8  #define numberOfObstacles 35
9  #define numberOfPackages 15
10 #define TIME_LIMIT_IN_SECONDS 30
11 #define packageLimitNumber 4
12 #define MATRIX_DIMENSION sizeof(char) * ROWS * COLUMNS
13 #define RED_COLOR "\x1b[31m"
14 #define GREEN_COLOR "\x1b[32m"
15 #define RESET_COLOR "\x1b[0m"
16
17 struct Coord {
18     int x;
19     int y;
20 };
21 typedef struct Coord *Point;
22 void rimuoviPaccoDaArray(int posizione[2], Point packsCoords[]);
23 void printMenu();
24 int getHiddenPack(Point packsCoords[]);
25 void stampaIstruzioni(int i);
26 int isOnADeployPoint(PlayerStats giocatore, Point deployCoords[]);
27 int isOnCorrectDeployPoint(PlayerStats giocatore, Point deployCoords[]);
28 PlayerStats gestisciP(char grigliaDiGioco[ROWS][COLUMNS], PlayerStats giocatore,
29                      Point deployCoords[], Point packsCoords[]);
30 PlayerStats gestisciA(char grigliaDiGioco[ROWS][COLUMNS],
31                      char grigliaOstacoli[ROWS][COLUMNS],
32                      PlayerStats giocatore, Obstacles *listaOstacoli,
33                      Point deployCoords[], Point packsCoords[]);
34 PlayerStats gestisciD(char grigliaDiGioco[ROWS][COLUMNS],
35                      char grigliaOstacoli[ROWS][COLUMNS],
36                      PlayerStats giocatore, Obstacles *listaOstacoli,
37                      Point deployCoords[], Point packsCoords[]);
38 PlayerStats gestisciS(char grigliaDiGioco[ROWS][COLUMNS],
39                      char grigliaOstacoli[ROWS][COLUMNS],
40                      PlayerStats giocatore, Obstacles *listaOstacoli,
41                      Point deployCoords[], Point packsCoords[]);
42 void inizializzaGiocoSenzaPlayer(char grigliaDiGioco[ROWS][COLUMNS],
43                                 char grigliaConOstacoli[ROWS][COLUMNS],
44                                 Point packsCoords[]);
45 void inserisciPlayerNellaGrigliaInPosizioneCasuale(
46     char grigliaDiGioco[ROWS][COLUMNS], char grigliaOstacoli[ROWS][COLUMNS],
47     int posizione[2]);
48 void inizializzaGrigliaVuota(char grigliaDiGioco[ROWS][COLUMNS]);
49 void generaPosizioneOstacoli(char grigliaDiGioco[ROWS][COLUMNS],
50                              char grigliaOstacoli[ROWS][COLUMNS]);
51 void riempiGrigliaConPacchiInPosizioniGenerateCasualmente(
52     char grigliaDiGioco[ROWS][COLUMNS], Point packsCoords[]);
53 void printGrid(char grigliaDaStampare[ROWS][COLUMNS], PlayerStats stats);
54 void start(char grigliaDiGioco[ROWS][COLUMNS],
55            char grigliaOstacoli[ROWS][COLUMNS]);
56 void riempiGrigliaConGliOstacoli(char grigliaDiGioco[ROWS][COLUMNS],
57                                  char grigliaOstacoli[ROWS][COLUMNS]);
58 void generaPosizioniRaccolta(char grigliaDiGioco[ROWS][COLUMNS],
59                              char grigliaOstacoli[ROWS][COLUMNS],
60                              Point coord[]);
61 PlayerStats gestisciW(char grigliaDiGioco[ROWS][COLUMNS],
62                      char grigliaOstacoli[ROWS][COLUMNS],
63                      PlayerStats giocatore, Obstacles *listaOstacoli,
64                      Point deployCoords[], Point packsCoords[]);
65 void mergeGridAndList(char grid[ROWS][COLUMNS], Obstacles top);
66 void scegliPosizioneRaccolta(Point coord[], int deploy[]);
67 int colpitoOstacolo(char grigliaOstacoli[ROWS][COLUMNS], int posizione[2]);
68 int colpitoPacco(Point packsCoords[], int posizione[2]);
69 int colpitoPlayer(char grigliaDiGioco[ROWS][COLUMNS], int posizione[2]);
```

```

70 int casellaVuota(char grigliaDiGioco[ROWS][COLUMNS],
71                 char grigliaOstacoli[ROWS][COLUMNS], int posizione[2]);
72 void spostaPlayer(char griglia[ROWS][COLUMNS], int vecchiaPosizione[2],
73                 int nuovaPosizione[2], Point deployCoords[],
74                 Point packsCoords[]);
75 int arrivatoADestinazione(int posizione[2], int destinazione[2]);
76 int eraUnPuntoDepo(int vecchiaPosizione[2], Point depo[]);
77 int eraUnPacco(int vecchiaPosizione[2], Point packsCoords[]);
78 int isOnAPack(PlayerStats giocatore, Point packsCoords[]);

```

Listato 13: Codice sorgente utility del gioco 1

```

1  #include "boardUtility.h"
2  #include "list.h"
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6  #include <unistd.h>
7  void printMenu() {
8      system("clear");
9      printf("\t Cosa vuoi fare?\n");
10     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         rimuoviPaccoFromArray(giocatore->position, packsCoords);
97     }
98     giocatore->deploy[0] = nuovoDeploy[0];
99     giocatore->deploy[1] = nuovoDeploy[1];
100     return giocatore;
101 }
102 void printGrid(char grigliaDaStampare[ROWS][COLUMNS], PlayerStats stats) {
103     system("clear");
104     printf("\n\n");
105     int i = 0, j = 0;
106     for (i = 0; i < ROWS; i++) {
107         printf("\t");
108         for (j = 0; j < COLUMNS; j++) {
109             if (stats != NULL) {
110                 if ((i == stats->deploy[0] && j == stats->deploy[1]) ||
111                     (i == stats->position[0] && j == stats->position[1]))
112                     if (grigliaDaStampare[i][j] == 'P' && stats->hasApack == 1)
113                         printf(GREEN_COLOR "%c" RESET_COLOR, grigliaDaStampare[i][j]);
114                     else
115                         printf(RED_COLOR "%c" RESET_COLOR, grigliaDaStampare[i][j]);
116                     else
117                         printf("%c", grigliaDaStampare[i][j]);
118             } else
119                 printf("%c", grigliaDaStampare[i][j]);
120         }
121         stampaIstruzioni(i);
122         if (i == 8)
123             printf(GREEN_COLOR "\t\t Punteggio: %d" RESET_COLOR, stats->score);
124         printf("\n");
125     }
126 }
127 void stampaIstruzioni(int i) {
128     if (i == 0)
129         printf("\t\t ISTRUZIONI ");
130     if (i == 1)
131         printf("\t\t Inviare 't' per il timer.");
132     if (i == 2)
133         printf("\t\t Inviare 'e' per uscire");
134     if (i == 3)
135         printf("\t\t Inviare 'p' per raccogliere un pacco");

```

```

136     if (i == 4)
137         printf("\t Inviare 'c' per consegnare il pacco");
138     if (i == 5)
139         printf("\t Inviare 'w'/'s' per andare sopra/sotto");
140     if (i == 6)
141         printf("\t Inviare 'a'/'d' per andare a dx/sx");
142     if (i == 7)
143         printf("\t Inviare 'l' per la lista degli utenti ");
144 }
145 // aggiunge alla griglia gli ostacoli visti fino ad ora dal client
146 void mergeGridAndList(char grid[ROWS][COLUMNS], Obstacles top) {
147     while (top) {
148         grid[top->x][top->y] = 'O';
149         top = top->next;
150     }
151 }
152 /* Genera la posizione degli ostacoli */
153 void generaPosizioneOstacoli(char grigliaDiGioco[ROWS][COLUMNS],
154                             char grigliaOstacoli[ROWS][COLUMNS]) {
155     int x, y, i;
156     inizializzaGrigliaVuota(grigliaOstacoli);
157     srand(time(0));
158     for (i = 0; i < numberOfObstacles; i++) {
159         x = rand() % COLUMNS;
160         y = rand() % ROWS;
161         if (grigliaDiGioco[y][x] == '-')
162             grigliaOstacoli[y][x] = 'O';
163         else
164             i--;
165     }
166 }
167 void rimuoviPaccoDaArray(int posizione[2], Point packsCoords[]) {
168     int i = 0, found = 0;
169     while (i < numberOfPackages && !found) {
170         if ((packsCoords[i]->x == posizione[0] &&
171             packsCoords[i]->y == posizione[1]) {
172             packsCoords[i]->x = -1;
173             packsCoords[i]->y = -1;
174             found = 1;
175         }
176         i++;
177     }
178 }
179 // sceglie una posizione di raccolta tra quelle disponibili
180 void scegliPosizioneRaccolta(Point coord[], int deploy[]) {
181     int index = 0;
182     srand(time(NULL));
183     index = rand() % numberOfPackages;
184     deploy[0] = coord[index]->x;
185     deploy[1] = coord[index]->y;
186 }
187 /*genera posizione di raccolta di un pacco*/
188 void generaPosizioniRaccolta(char grigliaDiGioco[ROWS][COLUMNS],
189                             char grigliaOstacoli[ROWS][COLUMNS],
190                             Point coord[]) {
191     int x, y;
192     srand(time(0));
193     int i = 0;
194     for (i = 0; i < numberOfPackages; i++) {
195         coord[i] = (Point)malloc(sizeof(struct Coord));
196     }
197     i = 0;
198     for (i = 0; i < numberOfPackages; i++) {
199         x = rand() % COLUMNS;
200         y = rand() % ROWS;
201         if (grigliaDiGioco[y][x] == '-' && grigliaOstacoli[y][x] == '-') {
202             coord[i]->x = y;
203             coord[i]->y = x;
204             grigliaDiGioco[y][x] = '_';
205             grigliaOstacoli[y][x] = '_';
206         } else
207             i--;
208     }
209 }

```

```

210  /*Inserisci dei pacchi nella griglia di gioco nella posizione casuale */
211  void riempiGrigliaConPacchiInPosizioniGenerateCasualmente(
212      char grigliaDiGioco[ROWS][COLUMNS], Point packsCoords[]) {
213      int x, y, i = 0;
214      for (i = 0; i < numberOfPackages; i++) {
215          packsCoords[i] = (Point)malloc(sizeof(struct Coord));
216      }
217      srand(time(0));
218      for (i = 0; i < numberOfPackages; i++) {
219          x = rand() % COLUMNS;
220          y = rand() % ROWS;
221          if (grigliaDiGioco[y][x] == '-') {
222              grigliaDiGioco[y][x] = '$';
223              packsCoords[i]->x = y;
224              packsCoords[i]->y = x;
225          } else
226              i--;
227      }
228  }
229  /*Inserisci gli ostacoli nella griglia di gioco */
230  void riempiGrigliaConGliOstacoli(char grigliaDiGioco[ROWS][COLUMNS],
231                                  char grigliaOstacoli[ROWS][COLUMNS]) {
232      int i, j = 0;
233      for (i = 0; i < ROWS; i++) {
234          for (j = 0; j < COLUMNS; j++) {
235              if (grigliaOstacoli[i][j] == 'O')
236                  grigliaDiGioco[i][j] = 'O';
237          }
238      }
239  }
240  void inserisciPlayerNellaGrigliaInPosizioneCasuale(
241      char grigliaDiGioco[ROWS][COLUMNS], char grigliaOstacoli[ROWS][COLUMNS],
242      int posizione[2]) {
243      int x, y;
244      srand(time(0));
245      printf("Inserisco player\n");
246      do {
247          x = rand() % COLUMNS;
248          y = rand() % ROWS;
249      } while (grigliaDiGioco[y][x] != '-' && grigliaOstacoli[y][x] != '-');
250      grigliaDiGioco[y][x] = 'P';
251      posizione[0] = y;
252      posizione[1] = x;
253  }
254  void inizializzaGiocoSenzaPlayer(char grigliaDiGioco[ROWS][COLUMNS],
255                                  char grigliaConOstacoli[ROWS][COLUMNS],
256                                  Point packsCoords[]) {
257      inizializzaGrigliaVuota(grigliaDiGioco);
258      riempiGrigliaConPacchiInPosizioniGenerateCasualmente(grigliaDiGioco,
259                                                            packsCoords);
260      generaPosizioneOstacoli(grigliaDiGioco, grigliaConOstacoli);
261      return;
262  }
263  void spostaPlayer(char griglia[ROWS][COLUMNS], int vecchiaPosizione[2],
264                  int nuovaPosizione[2], Point deployCoords[],
265                  Point packsCoords[]) {
266      griglia[nuovaPosizione[0]][nuovaPosizione[1]] = 'P';
267      if (eraUnPuntoDepo(vecchiaPosizione, deployCoords))
268          griglia[vecchiaPosizione[0]][vecchiaPosizione[1]] = '-';
269      else if (eraUnPacco(vecchiaPosizione, packsCoords))
270          griglia[vecchiaPosizione[0]][vecchiaPosizione[1]] = '$';
271      else
272          griglia[vecchiaPosizione[0]][vecchiaPosizione[1]] = '-';
273  }
274  int eraUnPuntoDepo(int vecchiaPosizione[2], Point depo[]) {
275      int i = 0, ret = 0;
276      while (ret == 0 && i < numberOfPackages) {
277          if ((depo[i]->y == vecchiaPosizione[1] &&
278              depo[i]->x == vecchiaPosizione[0]) {
279              ret = 1;
280          }
281          i++;
282      }
283      return ret;

```

```

284 }
285 int eraUnPacco(int vecchiaPosizione[2], Point packsCoords[]) {
286     int i = 0, ret = 0;
287     while (ret == 0 && i < numberOfPackages) {
288         if ((packsCoords[i])>y == vecchiaPosizione[1] &&
289             (packsCoords[i])>x == vecchiaPosizione[0]) {
290             ret = 1;
291         }
292         i++;
293     }
294     return ret;
295 }
296 PlayerStats gestisciW(char grigliaDiGioco[ROWS][COLUMNS],
297                       char grigliaOstacoli[ROWS][COLUMNS],
298                       PlayerStats giocatore, Obstacles *listaOstacoli,
299                       Point deployCoords[], Point packsCoords[]) {
300     if (giocatore == NULL)
301         return NULL;
302     int nuovaPosizione[2];
303     nuovaPosizione[1] = giocatore->position[1];
304     // Aggiorna la posizione vecchia spostando il player avanti di 1
305     nuovaPosizione[0] = (giocatore->position[0]) - 1;
306     int nuovoScore = giocatore->score;
307     int nuovoDeploy[2];
308     nuovoDeploy[0] = giocatore->deploy[0];
309     nuovoDeploy[1] = giocatore->deploy[1];
310     if (nuovaPosizione[0] >= 0 && nuovaPosizione[0] < ROWS) {
311         if (casellaVuotaOValida(grigliaDiGioco, grigliaOstacoli, nuovaPosizione)) {
312             spostaPlayer(grigliaDiGioco, giocatore->position, nuovaPosizione,
313                         deployCoords, packsCoords);
314         } else if (colpitoOstacolo(grigliaOstacoli, nuovaPosizione)) {
315             *listaOstacoli =
316                 addObstacle(*listaOstacoli, nuovaPosizione[0], nuovaPosizione[1]);
317             nuovaPosizione[0] = giocatore->position[0];
318             nuovaPosizione[1] = giocatore->position[1];
319         } else if (colpitoPlayer(grigliaDiGioco, nuovaPosizione)) {
320             nuovaPosizione[0] = giocatore->position[0];
321             nuovaPosizione[1] = giocatore->position[1];
322         }
323         giocatore->deploy[0] = nuovoDeploy[0];
324         giocatore->deploy[1] = nuovoDeploy[1];
325         giocatore->score = nuovoScore;
326         giocatore->position[0] = nuovaPosizione[0];
327         giocatore->position[1] = nuovaPosizione[1];
328     }
329     return giocatore;
330 }
331 PlayerStats gestisciD(char grigliaDiGioco[ROWS][COLUMNS],
332                       char grigliaOstacoli[ROWS][COLUMNS],
333                       PlayerStats giocatore, Obstacles *listaOstacoli,
334                       Point deployCoords[], Point packsCoords[]) {
335     if (giocatore == NULL) {
336         return NULL;
337     }
338     int nuovaPosizione[2];
339     nuovaPosizione[1] = giocatore->position[1] + 1;
340     nuovaPosizione[0] = giocatore->position[0];
341     int nuovoScore = giocatore->score;
342     int nuovoDeploy[2];
343     nuovoDeploy[0] = giocatore->deploy[0];
344     nuovoDeploy[1] = giocatore->deploy[1];
345     if (nuovaPosizione[1] >= 0 && nuovaPosizione[1] < COLUMNS) {
346         if (casellaVuotaOValida(grigliaDiGioco, grigliaOstacoli, nuovaPosizione)) {
347             spostaPlayer(grigliaDiGioco, giocatore->position, nuovaPosizione,
348                         deployCoords, packsCoords);
349         } else if (colpitoOstacolo(grigliaOstacoli, nuovaPosizione)) {
350             printf("Ostacolo\n");
351             *listaOstacoli =
352                 addObstacle(*listaOstacoli, nuovaPosizione[0], nuovaPosizione[1]);
353             nuovaPosizione[0] = giocatore->position[0];
354             nuovaPosizione[1] = giocatore->position[1];
355         } else if (colpitoPlayer(grigliaDiGioco, nuovaPosizione)) {
356             nuovaPosizione[0] = giocatore->position[0];
357             nuovaPosizione[1] = giocatore->position[1];

```

```

358     }
359     giocatore->deploy[0] = nuovoDeploy[0];
360     giocatore->deploy[1] = nuovoDeploy[1];
361     giocatore->score = nuovoScore;
362     giocatore->position[0] = nuovaPosizione[0];
363     giocatore->position[1] = nuovaPosizione[1];
364 }
365 return giocatore;
366 }
367 PlayerStats gestisciA(char grigliaDiGioco[ROWS][COLUMNS],
368                     char grigliaOstacoli[ROWS][COLUMNS],
369                     PlayerStats giocatore, Obstacles *listaOstacoli,
370                     Point deployCoords[], Point packsCoords[]) {
371     if (giocatore == NULL)
372         return NULL;
373     int nuovaPosizione[2];
374     nuovaPosizione[0] = giocatore->position[0];
375     // Aggiorna la posizione vecchia spostando il player avanti di 1
376     nuovaPosizione[1] = (giocatore->position[1]) - 1;
377     int nuovoScore = giocatore->score;
378     int nuovoDeploy[2];
379     nuovoDeploy[0] = giocatore->deploy[0];
380     nuovoDeploy[1] = giocatore->deploy[1];
381     if (nuovaPosizione[1] >= 0 && nuovaPosizione[1] < COLUMNS) {
382         if (casellaVuotaOValida(grigliaDiGioco, grigliaOstacoli, nuovaPosizione)) {
383             printf("Casella vuota \n");
384             spostaPlayer(grigliaDiGioco, giocatore->position, nuovaPosizione,
385                         deployCoords, packsCoords);
386         } else if (colpitoOstacolo(grigliaOstacoli, nuovaPosizione)) {
387             printf("Ostacolo\n");
388             *listaOstacoli =
389                 addObstacle(*listaOstacoli, nuovaPosizione[0], nuovaPosizione[1]);
390             nuovaPosizione[0] = giocatore->position[0];
391             nuovaPosizione[1] = giocatore->position[1];
392         } else if (colpitoPlayer(grigliaDiGioco, nuovaPosizione)) {
393             printf("colpito player\n");
394             nuovaPosizione[0] = giocatore->position[0];
395             nuovaPosizione[1] = giocatore->position[1];
396         }
397         giocatore->deploy[0] = nuovoDeploy[0];
398         giocatore->deploy[1] = nuovoDeploy[1];
399         giocatore->score = nuovoScore;
400         giocatore->position[0] = nuovaPosizione[0];
401         giocatore->position[1] = nuovaPosizione[1];
402     }
403     return giocatore;
404 }
405 PlayerStats gestisciS(char grigliaDiGioco[ROWS][COLUMNS],
406                     char grigliaOstacoli[ROWS][COLUMNS],
407                     PlayerStats giocatore, Obstacles *listaOstacoli,
408                     Point deployCoords[], Point packsCoords[]) {
409     if (giocatore == NULL) {
410         return NULL;
411     }
412     // crea le nuove statistiche
413     int nuovaPosizione[2];
414     nuovaPosizione[1] = giocatore->position[1];
415     nuovaPosizione[0] = (giocatore->position[0]) + 1;
416     int nuovoScore = giocatore->score;
417     int nuovoDeploy[2];
418     nuovoDeploy[0] = giocatore->deploy[0];
419     nuovoDeploy[1] = giocatore->deploy[1];
420     // controlla che le nuove statistiche siano corrette
421     if (nuovaPosizione[0] >= 0 && nuovaPosizione[0] < ROWS) {
422         if (casellaVuotaOValida(grigliaDiGioco, grigliaOstacoli, nuovaPosizione)) {
423             spostaPlayer(grigliaDiGioco, giocatore->position, nuovaPosizione,
424                         deployCoords, packsCoords);
425         } else if (colpitoOstacolo(grigliaOstacoli, nuovaPosizione)) {
426             printf("Ostacolo\n");
427             *listaOstacoli =
428                 addObstacle(*listaOstacoli, nuovaPosizione[0], nuovaPosizione[1]);
429             nuovaPosizione[0] = giocatore->position[0];
430             nuovaPosizione[1] = giocatore->position[1];
431         } else if (colpitoPlayer(grigliaDiGioco, nuovaPosizione)) {

```

```

432     nuovaPosizione[0] = giocatore->position[0];
433     nuovaPosizione[1] = giocatore->position[1];
434 }
435 giocatore->deploy[0] = nuovoDeploy[0];
436 giocatore->deploy[1] = nuovoDeploy[1];
437 giocatore->score = nuovoScore;
438 giocatore->position[0] = nuovaPosizione[0];
439 giocatore->position[1] = nuovaPosizione[1];
440 }
441 return giocatore;
442 }
443 int arrivatoADestinazione(int posizione[2], int destinazione[2]) {
444     if (posizione[0] == destinazione[0] && posizione[1] == destinazione[1])
445         return 1;
446     return 0;
447 }

```

A.4 Codice sorgente list

Listato 14: Codice header utility del gioco 2

```

1  #ifndef DEF_LIST_H
2  #define DEF_LIST_H
3  #define MAX_BUF 200
4  #include <pthread.h>
5  // players
6  struct TList {
7      char *name;
8      struct TList *next;
9      int sockDes;
10 } TList;
11
12 struct Data {
13     int deploy[2];
14     int score;
15     int position[2];
16     int hasApack;
17 } Data;
18
19 // Obstacles
20 struct TList2 {
21     int x;
22     int y;
23     struct TList2 *next;
24 } TList2;
25
26 typedef struct Data *PlayerStats;
27 typedef struct TList *Players;
28 typedef struct TList2 *Obstacles;
29
30 // calcola e restituisce il numero di player commessi dalla lista L
31 int dimensioneLista(Players L);
32
33 // inizializza un giocatore
34 Players initPlayerNode(char *name, int sockDes);
35
36 // Crea un nodo di Stats da mandare a un client
37 PlayerStats initStats(int deploy[], int score, int position[], int flag);
38
39 // Inizializza un nuovo nodo
40 Players initNodeList(char *name, int sockDes);
41
42 // Aggiunge un nodo in testa alla lista
43 // La funzione ritorna sempre la testa della lista
44 Players addPlayer(Players L, char *name, int sockDes);
45
46 // Rimuove solo un occorrenza di un nodo con il socket Descriptor
47 // specificato dalla lista
48 // La funzione ritorna sempre la testa della lista
49 Players removePlayer(Players L, int sockDes);
50

```

```

51 // Dealloca la lista interamente
52 void freePlayers(Players L);
53
54 // Stampa la lista
55 void printPlayers(Players L);
56
57 // Controlla se un utente á già loggato
58 int isAlreadyLogged(Players L, char *name);
59
60 // Dealloca la lista degli ostacoli
61 void freeObstacles(Obstacles L);
62
63 // Stampa la lista degli ostacoli
64 void printObstacles(Obstacles L);
65
66 // Aggiunge un ostacolo in testa
67 Obstacles addObstacle(Obstacles L, int x, int y);
68
69 // Inizializza un nuovo nodo ostacolo
70 Obstacles initObstacleNode(int x, int y);
71 #endif

```

Listato 15: 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         }
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) {
80     if (L != NULL) {
81         freeObstacles(L->next);
82         free(L);
83     }
84 }
85 void printPlayers(Players L) {
86     if (L != NULL) {
87         printf("%s ->", L->name);
88         printPlayers(L->next);
89     }
90     printf("\n");
91 }
92 void printObstacles(Obstacles L) {
93     if (L != NULL) {
94         printf("X:%d Y:%d ->", L->x, L->y);
95         printObstacles(L->next);
96     }
97 }

```

A.5 Codice sorgente parser

Listato 16: Codice header utility del gioco 3

```

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

```

Listato 17: Codice sorgente utility del gioco 3

```

1 #include "parser.h"
2 #include <errno.h>
3 #include <fcntl.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <sys/stat.h>

```



```

8 #include <sys/types.h>
9 #include <unistd.h>
10 #define MAX_BUF 200
11 int openFileRDWRAPP(char *file) {
12     int fileDes = open(file, O_RDWR | O_CREAT | O_APPEND, S_IRUSR | S_IWUSR);
13     if (fileDes < 0)
14         perror("Errore apertura file\n"), exit(-1);
15     return fileDes;
16 }
17 int openFileRDON(char *file) {
18     int fileDes = open(file, O_RDONLY);
19     if (fileDes < 0)
20         perror("Errore apertura file\n"), exit(-1);
21     return fileDes;
22 }
23 int appendPlayer(char *name, char *pwd, char *file) {
24     if (isRegistered(name, file))
25         return 0;
26     int fileDes = openFileRDWRAPP(file);
27     write(fileDes, name, strlen(name));
28     write(fileDes, " ", 1);
29     write(fileDes, pwd, strlen(pwd));
30     write(fileDes, "\n", 1);
31     close(fileDes);
32     return 1;
33 }
34 int isRegistered(char *name, char *file) {
35     char command[MAX_BUF] = "cat ";
36     strcat(command, file);
37     char toApp[] = " |cut -d\" \" -f1|grep \"^\"";
38     strcat(command, toApp);
39     strcat(command, name);
40     char toApp2[] = "$\">tmp";
41     strcat(command, toApp2);
42     int ret = 0;
43     system(command);
44     int fileDes = openFileRDON("tmp");
45     struct stat info;
46     fstat(fileDes, &info);
47     if ((int)info.st_size > 0)
48         ret = 1;
49     close(fileDes);
50     system("rm tmp");
51     return ret;
52 }
53 int validateLogin(char *name, char *pwd, char *file) {
54     if (!isRegistered(name, file))
55         return 0;
56     char command[MAX_BUF] = "cat ";
57     strcat(command, file);
58     char toApp[] = " |grep \"^\"";
59     strcat(command, toApp);
60     strcat(command, name);
61     strcat(command, " ");
62     strcat(command, pwd);
63     char toApp2[] = "$\">tmp";
64     strcat(command, toApp2);
65     int ret = 0;
66     system(command);
67     int fileDes = openFileRDON("tmp");
68     struct stat info;
69     fstat(fileDes, &info);
70     if ((int)info.st_size > 0)
71         ret = 1;
72     close(fileDes);
73     system("rm tmp");
74     return ret;
75 }
76 void premiEnterPerContinuare() {
77     fflush(stdin);
78     printf("Premi Invio per continuare\n");
79     char c = getchar();
80 }

```

Listati

1	Configurazione indirizzo del server	2
2	Configurazione socket del server	2
3	Procedura di ascolto del server	2
4	Configurazione e connessione del client	4
5	Risoluzione url del client	4
6	Prima comunicazione del server	5
7	Prima comunicazione del client	5
8	Funzione play del server	7
9	Funzione play del client	8
10	Codice sorgente del client	8
11	Codice sorgente del server	12
12	Codice header utility del gioco 1	20
13	Codice sorgente utility del gioco 1	21
14	Codice header utility del gioco 2	27
15	Codice sorgente utility del gioco 2	28
16	Codice header utility del gioco 3	29
17	Codice sorgente utility del gioco 3	29