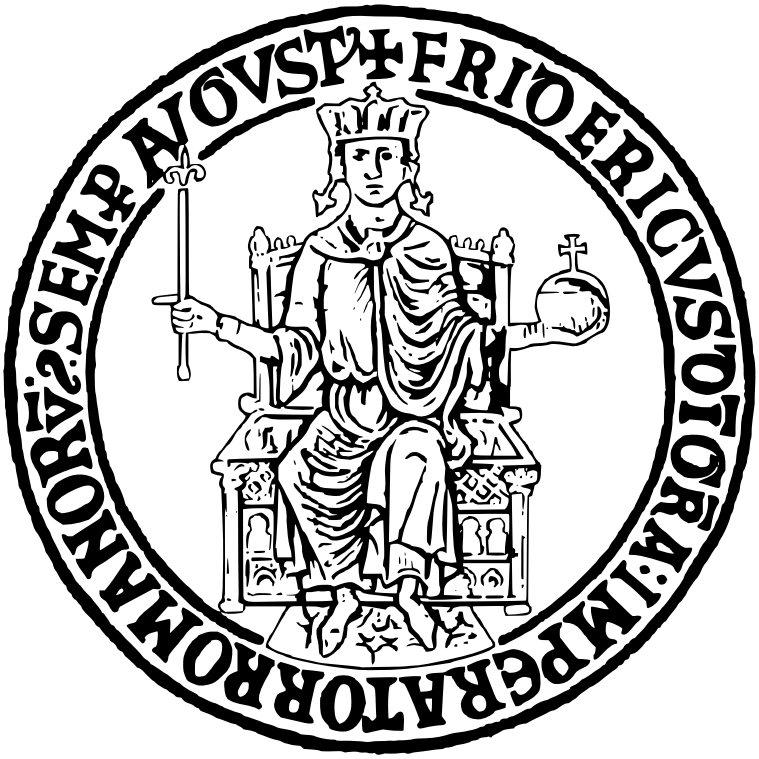
Università degli Studi di Napoli Federico II



Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di Laurea Triennale in Informatica

MarioStrikersCMD

Autori
Salvatore Franzese N86003142
Claudio Riccio N86003291

Docente
Prof. Alessandra Rossi

Anno Accademico 2023-2024

Indice

Capitolo 1

Panoramica del Software

1.1 Server

Il Server gestisce un numero di giocatori definito all'interno dell'header file `utils/constants.h` dalla costante `MAX_PLAYERS`. Una volta avviato, esso attenderà la connessione di tutti i giocatori. Oltre al numero di giocatori, è possibile personalizzare:

- La durata massima di un infortunio (`MAX_INJURY_TIME`);
- La durata della partita (`MAX_MATCH_DURATION`);
- La durata di una espulsione (`EXPEL_DURATION`);

Una volta connessi i giocatori, il Server chiederà a ognuno di essi di scegliere un nickname e un ruolo. Dopodiché, inizierà la partita e il Server mostrerà una telecronaca in real-time. Alla fine del match, la telecronaca ed altre statistiche verranno salvate in un file di log chiamato `log_match.txt` che verrà esportato dal container Docker se il programma è stato eseguito mediante `./run_app.sh`.

1.2 Client

Una volta avviato il Client, il giocatore potrà scegliere un nickname e un ruolo. Il ruolo determinerà le statistiche del giocatore. Ogni azione possibile (dribbling, tiro, infortunio, conquista del pallone) è determinata da un tiro di dado simulato usando la funzione `rand()%6`. Al risultato del tiro di dado viene poi sommato un valore determinato dal ruolo del giocatore:

Ruolo	Tiro	Dribbling	Velocità
Attaccante	+2	+1	+0
Centrocampista	+1	+1	+1
Difensore	+0	+0	+3

Tabella 1.1: Bonus

Le statistiche "tiro" e "dribbling" vengono sommate rispettivamente quando vengono eseguite le azioni di tiro e dribbling. "Velocità" influisce sulla conquista del pallone. L'infortunio avviene in caso di fallimento critico, ossia quando il tiro di dado restituisce uno 0. In caso di infortunio, viene sorteggiato un giocatore avversario casuale, che viene espulso per un intervallo di tempo. In caso esso fosse già espulso, non accadrà nulla.

Capitolo 2

Dettagli implementativi

La comunicazione tra Client e Server avviene tramite `socket` di rete:

```
int sock_fd = socket(AF_INET, SOCK_STREAM, 0);
```

Ad ogni giocatore è associato un `thread`, che cerca di accedere alla risorsa `pallone`, gestita da un semaforo `fastmutex`:

```
pthread_mutex_t ball = PTHREAD_MUTEX_INITIALIZER;
```

Di seguito riportiamo la struct rappresentante il giocatore e il metodo eseguito da ogni thread durante la partita.

```
typedef struct player
{
    int fd; // File descriptor della socket associata al giocatore
    pthread_t tid; // Thread id del thread associato al giocatore
    char name[20]; // Nickname
    player_role_enum player_role; // Ruolo
    stats_t stats; // Statistiche del giocatore
    team_enum team; // Squadra
    int is_injured; // Determina se un giocatore è attualemente infortunato
    int injury_time; // Determina quanto durerà l'infortunio del giocatore
    int is_expelled; // Determina se un giocatore è attualemente espulso
} player_t;

typedef enum player_role
{
    STRIKER,
    CENTER_MIDFIELDER,
    DEFENDER
} player_role_enum;

typedef enum team
{
    TEAM_A,
    TEAM_B
} team_enum;

typedef struct stats
{
    int shooting;
    int dribbling;
    int speed;
} stats_t;
```

```
void *kick_off(void *args)
{
    kick_off_data_t *tmp = (kick_off_data_t*) args;
    player_t current_player = tmp->player;
    player_t *opponents = tmp->opponents;
    srand(time(NULL));
    int MIN_WAIT = 3;

    while (1)
    {
        sleep(rand()%5 + MIN_WAIT - current_player.stats.speed);

        if (current_player.is_expelled == 1)
        {
            sleep(EXPEL_DURATION);
            current_player.is_expelled = 0;
        }

        if (pthread_mutex_trylock(&ball) == 0)
        {
            ball_possession(&current_player);

            if (current_player.is_injured)
            {
                expel_player(opponents);
            }

            pthread_mutex_unlock(&ball);

            if (current_player.is_injured)
            {
                bench_player(&current_player);
            }
        }

        sleep(1);
    }
}
```

Capitolo 3

Build & Esecuzione del software

3.1 Docker

Il Server può essere lanciato in esecuzione mediante Docker. In particolare, nella root directory del progetto è presente `compose.yml`, che è possibile lanciare in esecuzione eseguendo lo script `./run_app.sh`:

```
#!/bin/bash

docker compose up
docker cp MarioStrikersCMD_server:/tmp/log_match.txt log_match.txt
```

3.2 Linux

Pre-requisti:

- Compilatore gcc installato.
  - Ubuntu: `sudo apt install build-essential`.
  - Arch: `sudo pacman -S base-devel`.
  - Fedora: `sudo dnf groupinstall "Development Tools" "Development Libraries`.
- Docker e/o Docker compose installato.

Per facilitare la compilazione del programma sono stati scritti degli script BASH. Basterà posizionarsi nella cartella del progetto col comando `cd` e successivamente eseguire:

- `./build.sh` per compilare sia Client sia Server.

```
#!/bin/bash

gcc -g -o main_server \
    server/main.c server/connection_handler.c server/match.c server/logger.c

gcc -g -o main_client \
```

```
    client/main.c

chmod +x ./main_server
chmod +x ./main_client
```

- `./build_server.sh` per compilare solo il Server.

```
#!/bin/bash

gcc -g -o main_server \
    server/main.c server/connection_handler.c server/match.c server/logger.c

chmod +x ./main_server
```

- `./build_client.sh` per compilare solo il Client.

```
#!/bin/bash

gcc -g -o main_client \
    client/main.c

chmod +x ./main_client
```

Il Client e il Server possono essere eseguiti rispettivamente con `./main_client` e `./main_server`; in alternativa, il Server può essere eseguito con `./run_app.sh`, che eseguirà un container Docker e alla fine dell'esecuzione esporterà in automatico dal container il log (`log_match.txt`) della partita.

3.3 Windows

Pre-requisiti

- Aver installato la WSL - Windows Subsystem for Linux.
  - Comando per installarla tramite PowerShell: `wsl --install`
- Aver installato una qualsiasi distribuzione di Linux (Ubuntu, Arch, Fedora) sulla WSL.
- Compilatore gcc installato.
  - Ubuntu: `sudo apt install build-essential`
  - Arch: `sudo pacman -S base-devel`
  - Fedora: `sudo dnf groupinstall "Development Tools" "Development Libraries"`
- Docker e/o Docker compose installato nella WSL.

Il Client e il Server possono essere eseguiti rispettivamente con `./main_client` e `./main_server` da WSL; in alternativa, sempre all'interno della WSL, il Server può essere eseguito con lo script `./run_app.sh`, che eseguirà un container Docker e alla fine dell'esecuzione esporterà in automatico il log (`log_match.txt`) della partita.

3.4 Indirizzo IP e Porta

La configurazione di default permette di eseguire Server e Client in locale. Se si vuole eseguire il Server in remoto, va modificata la costante `Server_IP_ADDRESS` in `utils/constants.h`.
Di default, viene usata la porta `12122`; e nel caso si desideri sostituirla, va modificato il file `compose.yml` e la costante `PORT` in `utils/constants.h`.

Capitolo 4

Contributori

Salvatore Franzese
N86003142
salvator.franzese@studenti.unina.it
salvatorefranzese99@gmail.com

Claudio Riccio
N86003291
clau.riccio@studenti.unina.it
riccioclaudio2000@gmail.com