Progetto Pong: Documentazione

Benedetta Bonaccorso - 0124002651

16 Novembre, 2024

1 Descrizione del progetto

Il progetto consiste nello sviluppo di una versione semplificata del gioco Pong, implementata utilizzando un'architettura client-server basata su UDP. I giocatori controllano le rispettive racchette attraverso input da tastiera, mentre il server gestisce lo stato globale del gioco e sincronizza le azioni dei client.

1.1 Obiettivi principali

- Implementare un server che gestisce lo stato del gioco e comunica con i client.
- Creare client in grado di inviare input al server e ricevere aggiornamenti sullo stato del gioco.
- Sincronizzare la posizione della pallina e delle racchette tra i client.

2 Descrizione e schema dell'architettura

L'architettura del progetto si basa su un modello client-server:

- Il **server** responsabile della logica del gioco, della gestione della pallina e del punteggio, e invia aggiornamenti ai client.
- I **client** inviano comandi per muovere le racchette e ricevono lo stato aggiornato del gioco.

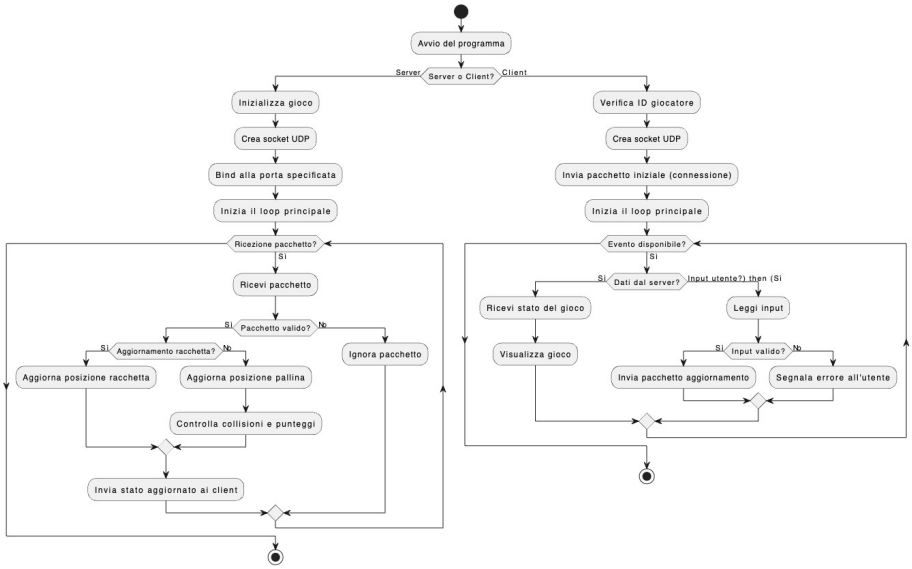

Figura 1: Schema dell'architettura client-server del progetto Pong.

3 Descrizione dello schema

- **Avvio del Server**:
  - Il server viene avviato e crea un socket UDP.
  - Si configura per ascoltare le connessioni sulla porta specificata.
  - Rimane in attesa di pacchetti dai client.
- **Avvio dei Client**:
  - Ogni client viene avviato specificando l'ID del giocatore (`0` o `1`).
  - Il client invia un pacchetto iniziale al server per registrarsi.
  - Si prepara per ricevere lo stato del gioco dal server e gestire l'input dell'utente.
- **Loop Principale del Server**:
  - Riceve pacchetti dai client: Se il pacchetto un aggiornamento della racchetta (tipo `P`), aggiorna la posizione della racchetta corrispondente.
  - Aggiorna lo stato del gioco (posizione della pallina, collisioni, punteggi).
  - Invia lo stato aggiornato a tutti i client connessi.

- **Loop Principale dei Client**:
  - Attende eventi da:
    - **Socket del server**: aggiorna e visualizza lo stato del gioco ricevuto.
    - **Input dell'utente**: lutente inserisce la posizione della propria racchetta.
  - Invia i movimenti della racchetta al server.
  - Esce dal loop se l'utente digita "exit".
- **Logica del Gioco** (gestita dal server):
  - La pallina si muove e cambia direzione quando colpisce:
    - I bordi superiori/inferiori.
    - Una delle racchette.
  - Incrementa il punteggio di un giocatore se la pallina supera il bordo opposto.
  - Reset della pallina al centro in caso di punto.
- **Fine del Gioco**:
  - Un client pu terminare il gioco digitando "exit".
  - Il server termina solo se interrotto manualmente.

4 Codice del progetto

Di seguito sono riportati i file principali del progetto.

4.1 Codice del server

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <time.h>
#include "game.h"

#define PORT 12345          // Porta su cui il server
   ascolta le connessioni
```

```c
#define BALL_SPEED_MS 500  // Velocit  di aggiornamento della pallina in millisecondi
#define MAX_CLIENTS 2      // Numero massimo di client che possono connettersi

typedef struct {
    char type; // Tipo di pacchetto: 'B' = aggiornamento pallina, 'P' = aggiornamento racchetta
    int x, y;  // Posizione della pallina o della racchetta
} Packet;

// Stato di connessione dei client: 0 = non connesso, 1 = connesso
int clients_connected[MAX_CLIENTS] = {0};

// Array per memorizzare gli indirizzi dei client connessi
struct sockaddr_in client_addrs[MAX_CLIENTS];

// Array per memorizzare la lunghezza degli indirizzi dei client connessi
socklen_t client_addr_lens[MAX_CLIENTS];

void server_loop() {
    int sockfd;                          // Socket del server
    struct sockaddr_in server_addr, client_addr; // Indirizzi del server e dei client
    socklen_t addr_len = sizeof(client_addr);    // Lunghezza dell'indirizzo del client
    GameState state;                     // Stato del gioco
    Packet packet;                       // Pacchetto ricevuto dal client

    // Inizializza il gioco con le posizioni iniziali di pallina e racchette
    initialize_game(&state);

    // Seed per la generazione casuale della direzione della pallina
    srand(time(NULL));
```

```
39
40      // Creazione del socket UDP
41      sockfd = socket(AF_INET, SOCK_DGRAM, 0);
42      if (sockfd < 0) {
43          perror("Errore nella creazione del socket");
44          exit(EXIT_FAILURE);
45      }
46
47      // Configurazione dell'indirizzo del server
48      server_addr.sin_family = AF_INET;            //
           Protocollo IPv4
49      server_addr.sin_addr.s_addr = INADDR_ANY;    //
           Accetta connessioni da qualsiasi indirizzo
50      server_addr.sin_port = htons(PORT);          //
           Porta specificata
51
52      // Bind del socket all'indirizzo e porta specificati
53      if (bind(sockfd, (struct sockaddr*)&server_addr,
           sizeof(server_addr)) < 0) {
54          perror("Errore nel bind del socket");
55          exit(EXIT_FAILURE);
56      }
57
58      printf("Server in ascolto sulla porta %d\n", PORT);
59
60      // Loop principale del server
61      while (1) {
62          // Ricezione di un pacchetto dal client
63          ssize_t bytes_received = recvfrom(sockfd, &
               packet, sizeof(packet), 0,
64                                            (struct
                                                 sockaddr*)&
                                                 client_addr
                                                 , &addr_len
                                                 );
65          if (bytes_received < 0) {
66              perror("Errore nella ricezione del pacchetto
                   ");
67              continue; // Continua al prossimo ciclo in
                   caso di errore
68          }
69
```

```
70          // Determina l'ID del client che ha inviato il
               pacchetto
71          int player_id = packet.x;
72
73          // Controlla se l'ID del client    valido
74          if (player_id >= 0 && player_id < MAX_CLIENTS) {
75              // Se il client non era ancora connesso,
                   registra il suo indirizzo
76              if (!clients_connected[player_id]) {
77                  clients_connected[player_id] = 1;
                                // Segna il client come
                       connesso
78                  client_addrs[player_id] = client_addr;
                          // Salva l'indirizzo del client
79                  client_addr_lens[player_id] = addr_len;
                         // Salva la lunghezza dell'
                       indirizzo
80                  printf("Client %d collegato\n",
                       player_id);
81              }
82
83              // Aggiorna la posizione della racchetta in
                   base al pacchetto ricevuto
84              if (packet.type == 'P') { // Se il pacchetto
                       un aggiornamento della racchetta
85                  if (player_id == 0)
86                      state.paddle_left = packet.y;  //
                           Aggiorna la racchetta del player
                           0
87                  else
88                      state.paddle_right = packet.y; //
                           Aggiorna la racchetta del player
                           1
89              }
90
91              // Aggiorna la posizione della pallina in
                   base alla logica del gioco
92              update_ball(&state);
93
94              // Stampa lo stato del gioco sul server (
                   solo a scopo di debug)
95              print_grid(&state);
96
```

```
97              // Invia lo stato aggiornato a tutti i
                   client connessi
98              for (int i = 0; i < MAX_CLIENTS; i++) {
99                  if (clients_connected[i]) { // Verifica
                       se il client    connesso
100                     sendto(sockfd, &state, sizeof(state)
                           , 0,
101                             (struct sockaddr*)&
                                client_addrs[i],
                                client_addr_lens[i]);
102                 }
103             }
104         }
105     }
106
107     // Chiude il socket al termine del loop (non verr
        mai raggiunto in questo caso)
108     close(sockfd);
109 }
110
111 int main() {
112     server_loop(); // Avvia il loop principale del
        server
113     return 0;      // Termina il programma (non verr
        mai raggiunto)
114 }
```

Listing 1: Codice principale del server

4.2 Codice del client

```
1  //
2  //Created by Benedetta Bonaccorso on 20/10/24.
3  //
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <unistd.h>
8  #include <arpa/inet.h>
9  #include "game.h"
10
11 #define PORT 12345 // Porta del server
12 #define SERVER_IP "127.0.0.1" // Indirizzo IP del server
```

```
13
14 typedef struct {
15     char type; // 'B' = ball update, 'P' = paddle update
16     int x, y;  /* x: Identifica il giocatore (0 o 1) in
       caso di aggiornamento racchetta.
17                   y: La posizione della racchetta o
                        della pallina.*/
18 } Packet;
19 // griglia
20 void display_game(GameState *state, int player_id) {
21     system("clear");
22     printf("Score: Player 1 - %d | Player 2 - %d\n",
       state->score_left, state->score_right);
23
24     for (int y = 0; y < GRID_SIZE; y++) {
25         for (int x = 0; x < GRID_SIZE; x++) {
26             if (x == state->ball_x && y == state->ball_y
                )
27                 printf("O "); // Mostra la pallina
28             else if (x == 0 && y >= state->paddle_left
                && y < state->paddle_left + PADDLE_SIZE)
29                 printf("| "); //Mostra la racchetta del
                    giocatore 0
30             else if (x == GRID_SIZE - 1 && y >= state->
                paddle_right && y < state->paddle_right +
                 PADDLE_SIZE)
31                 printf("| ");  // Mostra la racchetta
                    del giocatore 1
32             else
33                 printf(". ");  // Celle vuote
34         }
35         printf("\n");
36     }
37 }
38
39 void client_loop(int player_id) {
40     int sockfd;
41     struct sockaddr_in server_addr; //struct per l'
       indirizzo del server
42     Packet packet;  // Pacchetto da inviare
43     GameState state; // Stato attuale del gioco
44
45     // Configurazione socket
```

```
46      sockfd = socket(AF_INET, SOCK_DGRAM, 0); //socket
           udp
47      if (sockfd < 0) {
48          perror("Socket creation failed");
49          exit(EXIT_FAILURE);
50      }
51
52      server_addr.sin_family = AF_INET;  // IPv4
53      server_addr.sin_addr.s_addr = inet_addr(SERVER_IP);
           // Indirizzo del server
54      server_addr.sin_port = htons(PORT); // Porta del
           server
55
56      packet.type = 'P';  // Tipo di pacchetto:
           aggiornamento racchetta
57      packet.x = player_id;  // ID del giocatore
58      packet.y = 0; // Posizione iniziale della racchetta
59      sendto(sockfd, &packet, sizeof(packet), 0, (struct
           sockaddr*)&server_addr, sizeof(server_addr));
60      // Invia il pacchetto contenente il tipo di
           aggiornamento ('P' per paddle) e la posizione
61      // al server tramite il socket. Specifica l'
           indirizzo e la porta del server.
62      printf("Client %d connesso al server\n", player_id);
63
64      fd_set read_fds; // Set per monitorare input da
           socket e tastiera
65      int max_fd = sockfd > STDIN_FILENO ? sockfd :
           STDIN_FILENO;
66      // Determina il valore pi  alto tra il file
           descriptor del socket (sockfd)
67      // e lo standard input (STDIN_FILENO). Questo
           necessario per il funzionamento
68      // di select(), che richiede il file descriptor pi
            alto da monitorare.
69
70      while (1) {
71          FD_ZERO(&read_fds);
72          FD_SET(sockfd, &read_fds);    // Aggiungi socket
               per ricezione dati
73          FD_SET(STDIN_FILENO, &read_fds); // Aggiungi
              input da tastiera
74
```

```
75          // Aspetta eventi su uno dei file descriptor
76          if (select(max_fd + 1, &read_fds, NULL, NULL,
                NULL) < 0) {
77              perror("Errore in select()");
78              break;
79          }
80          // Controlla se ci sono dati dal server
81          if (FD_ISSET(sockfd, &read_fds)) {
82              ssize_t recv_len = recvfrom(sockfd, &state,
                    sizeof(state), 0, NULL, NULL);
83              if (recv_len > 0) {
84                  display_game(&state,player_id); //
                        Visualizza lo stato ricevuto
85              } else {
86                  perror("Errore nella ricezione dal
                        server");
87                  break;
88              }
89          }
90
91          // Controlla input utente
92          if (FD_ISSET(STDIN_FILENO, &read_fds)) {
93              char buffer[16]; // Buffer per l'input
94              if (fgets(buffer, sizeof(buffer), stdin)) {
95              // Controlla se l'utente vuole uscire
96              if (strncmp(buffer, "exit", 4) == 0) {
97                      printf("Uscita dal gioco.\n");
98                      break; // Esce dal loop principale
99                  }
100
101
102
103                 packet.y = atoi(buffer); // Converte l'
                        input in intero
104
105                 // Validazione input
106                 if (packet.y < 0 || packet.y > GRID_SIZE
                        - PADDLE_SIZE) {
107                     printf("Input non valido. Riprova.\n
                            ");
108                     continue;
109                 }
110
```

```
111                     packet.type = 'P';
112                     packet.x = player_id;
113
114                     // Invia aggiornamento al server
115                     sendto(sockfd, &packet, sizeof(packet),
                            0, (struct sockaddr*)&server_addr,
                            sizeof(server_addr));
116                 } else {
117                     perror("Errore nella lettura dell'input"
                            );
118                 }
119             }
120         }
121
122         close(sockfd);
123     }
124
125     int main(int argc, char *argv[]) {
126         // Verifica che venga passato l'ID del giocatore (
                player_id).
127         if (argc < 2) {
128             // Stampa un messaggio di utilizzo per informare
                    l'utente del formato corretto.
129             printf("Usage: %s <player_id>\n", argv[0]);
130             return 1;
131         }
132
133         int player_id = atoi(argv[1]);
134         if (player_id != 0 && player_id != 1) {//bisogna
                inserire player 0 o player 1
135             printf("Errore: player_id deve essere 0 o 1.\n")
                    ;
136             return 1;
137         }
138
139         client_loop(player_id);
140         return 0;
141     }
```

Listing 2: Codice principale del client

4.3 Codice della logica di gioco (`game.c`)

```c
//
// Created by Benedetta Bonaccorso on 20/10/24.
//
#include "game.h"
#include <stdio.h>
#include <stdlib.h>
void update_ball(GameState *state) {
    state->ball_x += state->ball_dx;
    state->ball_y += state->ball_dy;

    // Collisione con le pareti verticali
    if (state->ball_y <= 0 || state->ball_y >= GRID_SIZE
        - 1) {
        state->ball_dy *= -1;
    }

    // Controllo punti
    if (state->ball_x < 0) {
        state->score_right++;
        reset_ball(state);
    } else if (state->ball_x >= GRID_SIZE) {
        state->score_left++;
        reset_ball(state);
    }

    // Controllo collisione con le pedine
    if (state->ball_x == 1 && state->ball_y >= state->
       paddle_left &&
        state->ball_y < state->paddle_left + PADDLE_SIZE
           ) {
        state->ball_dx *= -1;
    }
    if (state->ball_x == GRID_SIZE - 2 && state->ball_y
       >= state->paddle_right &&
        state->ball_y < state->paddle_right +
           PADDLE_SIZE) {
        state->ball_dx *= -1;
    }
}

void reset_ball(GameState *state) {
    state->ball_x = GRID_SIZE / 2;
```

```
38      state->ball_y = GRID_SIZE / 2;
39      state->ball_dx = (rand() % 2 == 0) ? -1 : 1;
40      state->ball_dy = (rand() % 2 == 0) ? -1 : 1;
41  }
42
43  void initialize_game(GameState *state) {
44      state->ball_x = GRID_SIZE / 2;
45      state->ball_y = GRID_SIZE / 2;
46      state->ball_dx = -1;
47      state->ball_dy = 1;
48      state->paddle_left = GRID_SIZE / 2 - PADDLE_SIZE /
            2;
49      state->paddle_right = GRID_SIZE / 2 - PADDLE_SIZE /
            2;
50      state->score_left = 0;
51      state->score_right = 0;
52  }
53
54  void print_grid(GameState* state) {
55      char grid[10][10];
56
57      // Inizializza la griglia vuota
58      for (int i = 0; i < 10; i++) {
59          for (int j = 0; j < 10; j++) {
60              grid[i][j] = ' ';
61          }
62      }
63
64      // Posiziona la pallina
65      grid[state->ball_y][state->ball_x] = 'O';
66
67      // Posiziona le racchette
68      for (int i = -1; i <= 1; i++) { // Racchetta
            sinistra
69          if (state->paddle_left + i >= 0 && state->
                paddle_left + i < 10) {
70              grid[state->paddle_left + i][0] = '|';
71          }
72      }
73
74      for (int i = -1; i <= 1; i++) { // Racchetta destra
75          if (state->paddle_right + i >= 0 && state->
                paddle_right + i < 10) {
```

```
76              grid[state->paddle_right + i][9] = '|';
77          }
78      }
79
80      // Stampa la griglia
81      printf("\n");
82      for (int i = 0; i < 10; i++) {
83          for (int j = 0; j < 10; j++) {
84              printf("%c ", grid[i][j]);
85          }
86          printf("\n");
87      }
88      printf("\n");
89  }
```

Listing 3: Funzioni per la gestione del gioco

5 Manuale utente

5.1 Compilazione del codice

Per compilare il progetto, utilizza il comando `gcc`:

```
gcc -o server server.c game.c
gcc -o client client.c game.c
```

5.2 Esecuzione del programma

- Avvia il server con il comando:

  ```
  ./server
  ```

- Avvia i client con il comando:

  ```
  ./client <player_id>
  ```

  Dove `<player_id>` pu essere 0 o 1.

5.3 Controlli del gioco

- Inserisci un numero per spostare la racchetta su o gi.
- Digita `exit` per chiudere il client.

6 Conclusioni

Il progetto implementa con successo un semplice gioco Pong utilizzando la comunicazione UDP. Le funzionalit principali includono la sincronizzazione in tempo reale tra client e server e una gestione robusta degli stati di gioco.