Progetto Pong: Documentazione

Benedetta Bonaccorso - 0124002651

16 Novembre, 2024

1 Obiettivi principali

Il progetto consiste nello sviluppo di una versione semplificata del gioco Pong, implementata utilizzando un'architettura client-server basata su UDP. Gli obiettivi principali sono:

- 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

2.1 Casi d'uso principali:

- **Avvia il Server :** Il server si avvia e rimane in ascolto delle connessioni client.
- **Connessione al Server :** I client si connettono al server e si registrano come giocatori.
- **Invia Input :** Il client invia il movimento della racchetta al server.
- **Riceve Aggiornamenti Stato Gioco :** Il server invia al client lo stato aggiornato del gioco.
- **Elabora Movimento Pallina:** Il server calcola la nuova posizione della pallina in base alla logica del gioco.
- **Calcola Punteggio :** Il server aggiorna i punteggi se la pallina supera una racchetta.
- **Invia Stato Gioco ai Client :** Il server comunica lo stato aggiornato a tutti i client.
- **Mostra Stato Gioco :** Il client visualizza lo stato aggiornato del gioco sul terminale.

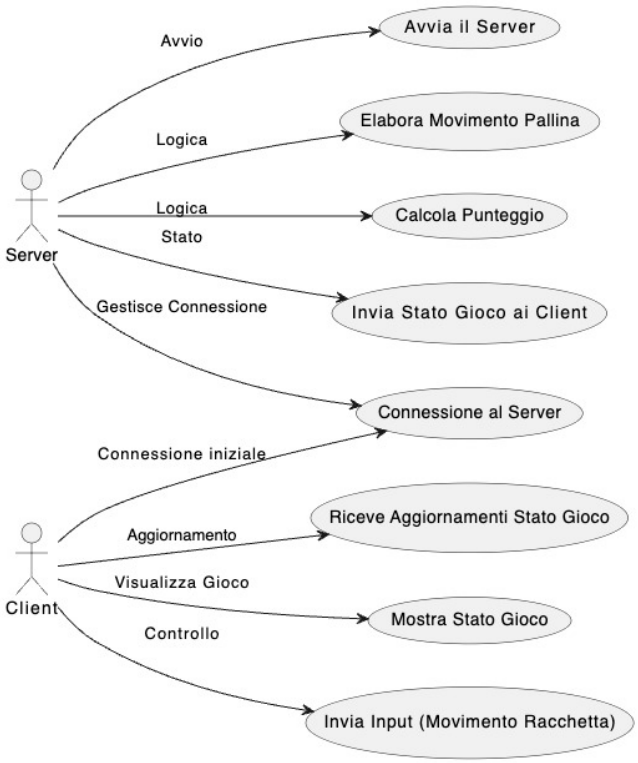

Figura 1: use case diagram

2.1.1 Descrizione delle classi:

1. **GameState:**
   - Contiene gli attributi per lo stato del gioco (posizione pallina, racchette, punteggi).
   - Include metodi per aggiornare la pallina, resettare il gioco e stampare la griglia.
2. **Packet:**
   - una struttura che rappresenta i pacchetti inviati tra client e server.
   - Include il tipo (`type`), la posizione (`x`, `y`).
3. **Server:**
   - Gestisce le connessioni e mantiene lo stato del gioco.
   - Utilizza socket per comunicare con i client.
   - Contiene un istanza di **GameState** per la logica di gioco.
4. **Client:**
   - Rappresenta i giocatori che interagiscono con il server.
   - Invia aggiornamenti al server e visualizza lo stato del gioco ricevuto

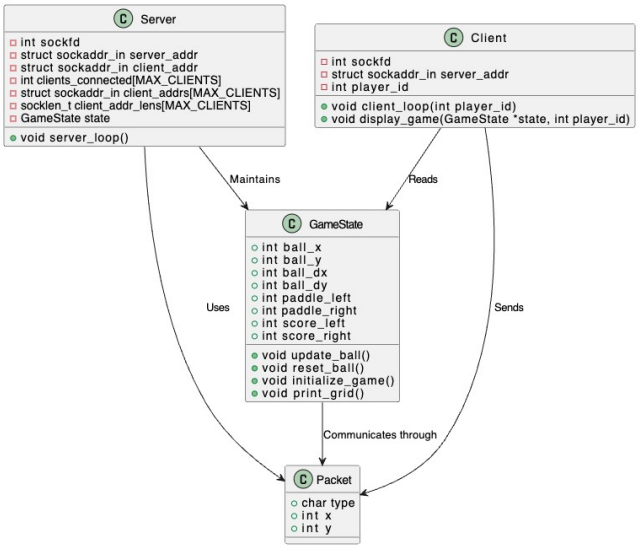

Figura 2: class diagram

2.2 Componenti del Sistema

Le principali componenti del sistema sono:

- **Server centrale**: Il server centrale responsabile della gestione delle connessioni dei giocatori e dello stato globale del gioco. Si occupa di:
  - Assegnare un identificativo ai giocatori (Player 0 e Player 1).
  - Ricevere le posizioni aggiornate delle racchette inviate dai giocatori.
  - Calcolare la nuova posizione della pallina in base alla logica del gioco, tenendo conto di collisioni e punteggi.
  - Distribuire in tempo reale lo stato aggiornato del gioco a entrambi i giocatori.
- **Giocatori (Clients)**: Ogni giocatore e' un client che:
  - Si connette al server per partecipare alla partita.
  - Invia al server le informazioni relative alla posizione della propria racchetta.
  - Riceve lo stato del gioco aggiornato dal server e lo visualizza tramite una rappresentazione grafica a griglia.
- **Pallina virtuale**: La pallina gestita esclusivamente dal server, che:
  - Determina la sua posizione in base alla velocita' e direzione.
  - Gestisce collisioni con le pareti e le racchette.
  - Calcola i punteggi ogni volta che la pallina supera i limiti del campo.
- **Racchette virtuali**: Ogni giocatore controlla una racchetta virtuale:
  - Il movimento della racchetta aggiornato dal client e inviato al server.
  - La posizione delle racchette influisce sulle interazioni con la pallina.

3 Dettagli implementativi dei client/server

3.1 Server centrale

Assegna un identificativo ai giocatori (Player 0 e Player 1): Questa operazione viene effettuata quando il server riceve un pacchetto da un client. Se il client non e' ancora registrato, il server lo identifica e registra il suo indirizzo.

Riceve le posizioni aggiornate delle racchette inviate dai giocatori: Il server riceve pacchetti contenenti le posizioni delle racchette e aggiorna lo stato del gioco.

```c
// Controlla se l'ID del client é valido
if (player_id >= 0 && player_id < MAX_CLIENTS) {
    // Se il client non era ancora connesso, registra il suo indirizzo
    if (!clients_connected[player_id]) {
        clients_connected[player_id] = 1;          // Segna il client come connesso
        client_addrs[player_id] = client_addr;     // Salva l'indirizzo del client
        client_addr_lens[player_id] = addr_len;    // Salva la lunghezza dell'indirizzo
        printf("Client %d collegato\n", player_id);
    }

    // Aggiorna la posizione della racchetta in base al pacchetto ricevuto
    if (packet.type == 'P') { // Se il pacchetto è un aggiornamento della racchetta
        if (player_id == 0)
            state.paddle_left = packet.y;  // Aggiorna la racchetta del player 0
        else
            state.paddle_right = packet.y; // Aggiorna la racchetta del player 1
    }
```

Figura 3: Gestione delle posizioni delle racchette

Calcola la nuova posizione della pallina e gestisce la logica del gioco: La logica del movimento della pallina e delle collisioni implementata nella funzione (`update_ball`). La funzione aggiorna la posizione della pallina in base alla sua direzione (`ball_dx` e `ball_dy`) e controlla se la pallina collide con le pareti superiori o inferiori invertendo la direzione verticale (`ball_dy`). Se la pallina supera i bordi sinistro o destro del campo, incrementa il punteggio dell'avversario e riposiziona la pallina al centro. Infine, verifica le collisioni con le racchette dei giocatori: se la pallina si trova in prossimit della racchetta e nella sua altezza, inverte la direzione orizzontale (`ball_dx`).

```c
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;
    }
}
```

Figura 4: Logica di aggiornamento della pallina

Distribuisce lo stato aggiornato del gioco ai client: Dopo aver aggiornato lo stato del gioco, il server invia il nuovo stato ai client connessi.

```
// Invia lo stato aggiornato a tutti i client connessi
for (int i = 0; i < MAX_CLIENTS; i++) {
    if (clients_connected[i]) { // Verifica se il client è connesso
        sendto(sockfd, &state, sizeof(state), 0,
               (struct sockaddr*)&client_addrs[i], client_addr_lens[i]);
    }
}
```

Figura 5: Distribuzione degli aggiornamenti

3.2 Giocatori (Clients)

1. Si connettono al server per partecipare alla partita: Quando un client si avvia, invia un pacchetto iniziale al server con il proprio ID.

```
packet.type = 'P';  // Tipo di pacchetto: aggiornamento racchetta
packet.x = player_id;  // ID del giocatore
packet.y = 0; // Posizione iniziale della racchetta
sendto(sockfd, &packet, sizeof(packet), 0, (struct sockaddr*)&server_addr, sizeof(server_addr));
// Invia il pacchetto contenente il tipo di aggiornamento ('P' per paddle) e la posizione
// al server tramite il socket. Specifica l'indirizzo e la porta del server.
printf("Client %d connesso al server\n", player_id);
```

Figura 6: Connessione del client

Invia al server le informazioni relative alla posizione della racchetta: Il client legge l'input utente per spostare la racchetta e invia il pacchetto aggiornato al server. Questa funzionalita' gestita nel **client loop**. Quando il client riceve input dall'utente tramite tastiera, converte l'input nella posizione verticale desiderata della racchetta (`packet.y`). Se la posizione e' valida, prepara un pacchetto di tipo `'P'` che include l'ID del giocatore (`packet.x`) e la nuova posizione. Questo pacchetto viene inviato al server tramite la funzione `sendto()`, aggiornando cosi lo stato della racchetta sul server in tempo reale.

```
    // Controlla input utente
    if (FD_ISSET(STDIN_FILENO, &read_fds)) {
        char buffer[16]; // Buffer per l'input
        if (fgets(buffer, sizeof(buffer), stdin)) {
        // Controlla se l'utente vuole uscire
        if (strncmp(buffer, "exit", 4) == 0) {
                printf("Uscita dal gioco.\n");
                break; // Esce dal loop principale
            }
              packet.y = atoi(buffer); // Converte l'input in intero

            // Validazione input
            if (packet.y < 0 || packet.y > GRID_SIZE - PADDLE_SIZE) {
                printf("Input non valido. Riprova.\n");
                continue;
            }

            packet.type = 'P';
            packet.x = player_id;

            // Invia aggiornamento al server
            sendto(sockfd, &packet, sizeof(packet), 0, (struct sockaddr*)&server_addr, sizeof(server_addr));
        } else {
            perror("Errore nella lettura dell'input");
        }
    }
}
```

Figura 7: Gestione input utente

Ricezione dei dati dal server: Il client utilizza la funzione `recvfrom` per ricevere un pacchetto contenente lo stato aggiornato del gioco (`GameState`) inviato dal server. Questo pacchetto include la posizione della pallina, delle racchette e il punteggio.

Controllo della validita' dei dati ricevuti: Dopo aver ricevuto il pacchetto, viene verificato che la lunghezza dei dati ricevuti sia corretta, per assicurarsi che non ci siano errori di trasmissione. I dati ricevuti vengono passati alla funzione `display_game`, che disegna una griglia di gioco sul terminale.

```
// Controlla se ci sono dati dal server
if (FD_ISSET(sockfd, &read_fds)) {
    ssize_t recv_len = recvfrom(sockfd, &state, sizeof(state), 0, NULL, NULL);
    if (recv_len > 0) {
        display_game(&state,player_id); // Visualizza lo stato ricevuto
    } else {
        perror("Errore nella ricezione dal server");
        break;
    }
}
```

Figura 8: Ricezione degli aggiornamenti

4 Manuale utente

4.1 Compilazione

Per compilare il progetto, utilizzare i seguenti comandi:

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

5 Esecuzione del programma

Per avviare il gioco:

- Server:

  ```
  ./server
  ```

- Client:

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

  dove `<player_id>` puo' essere 0 o 1.

6 Controlli del gioco

- Inserire un numero per spostare la racchetta su o giu.
- Digitare `exit` per chiudere il client.

7 Conclusioni

Il progetto implementa con successo un semplice gioco Pong utilizzando la comunicazione UDP. Le funzionalita' principali includono:

- Sincronizzazione in tempo reale tra client e server.
- Gestione degli stati di gioco.
- Sistema di punteggio funzionante.