Laboratorio di Sistemi Operativi

Indice

1 **Introduzione** 3

1.1 Obiettivi del progetto 3

2 **Architettura del sistema** 4

3 **Struttura del codice** 5

3.1 Server (server.c) 5

3.2 Client (client.c) 5

3.3 Moduli di gioco(tris_game.c e tris_game.h) 5

4 **Gestione delle partite** 6

4.1 Avvio dei Servizi: 6

4.2 Rivincita e rotazione: 7

5 **Funzioni di Utilità e Gestione Comandi:** 7

5.1 Server: 7

5.2 Client: 8

6 **Docker e Docker Compose:** 9

6.1 Vantaggi: 9

6.2 Servizi(docker-compose-yml): 9

7 **Conclusioni:** 10

1 Introduzione

Questo documento fornisce un'implementazione del classico gioco del **Tris (Tic-Tac-Toe)** giocabile in modalità multiplayer su rete. Il progetto è composto da un **Server** centrale che gestisce le partite e le interazioni tra i giocatori, e da un **Client** che permette agli utenti di connettersi al server, creare o unirsi a partite e giocare. L'intera applicazione è containerizzata utilizzando **Docker** e **Docker Compose** per facilitare il setup e il deployment.

1.1 Obiettivi del progetto

Creare un'applicazione **Client-Server** concorrente per giocare a **Tris**.

- Gestire più partite simultaneamente tra coppie di client.
- Offrire un sistema di rivincita automatico.
- Automatizzare il deployment tramite Docker e Docker Compose

2 Architettura del sistema

Il sistema è basato su un'architettura client-server modulare, composta dai seguenti componenti principali:

• **Server (server.c):** Il cuore dell'applicazione. È un'applicazione C che rimane in ascolto su una porta specifica, accettando connessioni da più client. Gestisce lo stato delle partite, le richieste dei giocatori e la comunicazione tra di essi. Utilizza select() per gestire efficientemente I/O su più socket.

• **Client (client.c):** L'interfaccia utente. È un'applicazione C che si connette al server e permette all'utente di inviare comandi (es. creare partite, fare mosse) e ricevere aggiornamenti sullo stato del gioco.

• **Logica di Gioco (tris_game.c, tris_game.h):** Una libreria separata che incapsula la logica del gioco del Tris, inclusa l'inizializzazione del tabellone, la gestione delle mosse e la determinazione del vincitore o del pareggio. Questo modulo è utilizzato dal server.

• **Docker Compose (docker-compose.yml):** Un file di configurazione che orchestra il deployment del server e dei client come servizi Docker separati, definendo la loro build, dipendenze, network e mappature di porte.
L'interazione avviene tramite messaggi testuali scambiati su socket TCP. I client inviano comandi al server, e il server risponde con messaggi di stato, tabellone di gioco aggiornato o notifiche.

3 Struttura del codice

3.1 Server (server.c)

Avvia un socket TCP su una porta fissa (8080).

• Ogni client è rappresentato da una struttura Client che contiene stato, simbolo e riferimento alla partita.
• Le partite sono rappresentate da strutture Game.
• Comandi implementati: create, join, accept, reject, move, leave, rematch, list, quit.

3.2 Client (client.c)

- Si connette al server leggendo host e porta da variabili d'ambiente (SERVER_HOST, SERVER_PORT).
- Inoltra comandi digitati dall'utente al server e stampa le risposte.

3.3 Moduli di gioco(tris_game.c e tris_game.h)

- **init_game:** inizializza una griglia vuota.
- **make_move:** applica una mossa, se valida.
- **check_winner:** verifica la condizione di vittoria, sconfitta o pareggio.
- **print_board:** converte la griglia in testo leggibile.

4 Gestione delle partite

Ogni partita è identificata da un ID univoco e può trovarsi in uno di questi stati:

- **Attesa:** l'host ha creato la partita ma non c'è ancora un avversario.
- **In corso:** due giocatori stanno giocando.
- **Terminata:** la partita si è conclusa per vittoria, sconfitta o pareggio.

4.1 Avvio dei Servizi:

Ecco un esempio di una sessione di gioco tipica:

1. **Avvio dei Servizi:** L'utente avvia **docker-compose build**. Il server si avvia e il primo client si connette automaticamente.

2. **Client 1 (Giocatore A):** Digita **create**. Il server crea una nuova partita, assegna l'ID al Giocatore A e gli comunica che è il Giocatore X, in attesa di un avversario.

3. **Client 2 (Giocatore B):** L'utente avvia un nuovo container client (**docker-compose run --rm client**). Giocatore B digita **list** per vedere le partite disponibili e trova l'ID della partita creata da Giocatore A. Successivamente, digita **join <game_id>**. Il server notifica Giocatore A che Giocatore B vuole unirsi.

4. **Client 1 (Giocatore A):** Riceve la notifica e digita **accept**. Il server imposta lo stato della partita su "in corso", assegna 'O' a Giocatore B e invia il tabellone iniziale ad entrambi. È il turno di Giocatore A (X).

5. **Turni di Gioco:**

- **Giocatore A (X)** digita **move 0 0**. Il server valida la mossa, aggiorna il tabellone e lo invia a entrambi i giocatori. Il turno passa a Giocatore B (O).

- **Giocatore B (O)** digita **move 1 1**. Il server valida, aggiorna e invia il tabellone. Il turno passa a Giocatore A.

- Questo processo continua fino a quando un giocatore vince, perde la partita o ha un pareggio.

6. **Fine Partita:**
   Quando **check_winner** rileva una vittoria, sconfitta o un pareggio, il server comunica il risultato ai giocatori. I giocatori possono quindi digitare **rematch** per richiedere una nuova partita o **leave** per uscire.
   Riceve la notifica e digita **accept**. Il server imposta lo stato della partita su "in corso", assegna 'O' a Giocatore B e invia il tabellone iniziale ad entrambi. È il turno di Giocatore A (X).

7. **Disconnessione:**
   Un giocatore può digitare **quit** per disconnettersi dal server.

4.2 Rivincita e rotazione:

- Dopo una vittoria: il vincitore resta e attende un nuovo sfidante diventando l'host.
- Dopo un pareggio: entrambi possono digitare rematch.
- Se entrambi lo fanno, la partita viene resettata con i turni invertiti.

5 Funzioni di Utilità e Gestione Comandi:

5.1 Server:

Il server implementa diverse funzioni per gestire la logica del gioco e la comunicazione:

- **send_to_client**(): Invia un messaggio testuale a un client specifico.
- **initialize_client**(): inizializza una nuova struttura Client e invia un messaggio di benvenuto con l'elenco dei comandi disponibili.
- **cleanup_game**(): Resetta una struttura Game, rendendo lo slot disponibile per nuove partite.
- **remove_client_from_game**(): Rimuove un client da una partita, gestendo le conseguenze(es. notifica all'altro giocatore, pulizia delle partita se vuota).
- **remove_client**(): Gestisce la disconnessione completa di un client, rimuovendolo da qualsiasi partita e dal set master_fds.
- **find_game_by_id**(), **find_game_by_player_fd**(), **find_client_by_fd**(): Funzioni di ricerca per trovare partite o client.
- **print_game_list**(): Invia al client la lista delle partite attive.
- **send_game_state_to_players**(): Invia lo stato attuale del tabellone e le informazioni sul turno ai giocatori di una partita.

5.2 Client:

I comandi dei client sono parsati e gestiti da funzioni dedicate:

- **handle_create_command**(): Permette a un client di creare una nuova partita. Se ha successo, il client diventa il "proprietario" della partita (Giocatore X) e la partita entra nello stato GAME_WAITING_FOR_PLAYER

- **handle_join_command**(): Permette a un client di unirsi a una partita esistente specificando l'ID. Il client che si unisce diventa il Giocatore O e il proprietario della partita viene notificato.

- **handle_accept_command**(): Il proprietario di una partita accetta la richiesta di unione di un altro giocatore, avviando la partita (GAME_IN_PROGRESS).

- **handle_reject_command**(): Il proprietario di una partita rifiuta la richiesta di unione di un altro giocatore.

- **handle_leave_command**(): Un giocatore lascia la partita corrente.

- **handle_move_command**(): Un giocatore tenta di effettuare una mossa specificando riga e colonna. Il server valida la mossa e aggiorna lo stato del gioco.

- **handle_rematch_command**(): Permette ai giocatori di richiedere una rivincita dopo una partita terminata.

- **handle_client_data**(): Funzione principale che riceve i dati dai client e delega la gestione ai comandi

6 Docker e Docker Compose:

6.1 Vantaggi:

- Isolamento delle dipendenze.
- Portabilità e riproducibilità dell'ambiente.
- Esecuzione facilitata su macOS, Linux, Windows.

6.2 Servizi(docker-compose-yml):

- **Server:** esegue server.c in background.
- **Client:** eseguibile interattivo (può essere lanciato più volte per simulare più utenti).

7 Conclusioni:

Il progetto ha pienamente raggiunto gli obiettivi prefissati, realizzando un ambiente stabile e scalabile per il gioco del Tris in rete. L'architettura modulare, unita all'utilizzo di Docker, garantisce una solida base per future estensioni, come l'integrazione di un'interfaccia grafica o il supporto ad altri giochi.