Universita' degli Studi di Napoli "Parthenope"
Facolta' di Scienze e Tecnologie
Corso di laurea in Informatica (percorso Generale)

Reti di calcolatori

Gioco Quiz

Studente
Davide d'Auria

Matricola
0124002639

Anno Accademico 2023-2024

QuizGame

Davide d'Auria

Dicembre 2024

Contents

1 Introduzione

Il progetto consiste nello sviluppo di un gioco a quiz interattivo che permette a più giocatori di partecipare simultaneamente. Il sistema è composto da un server che gestisce l'inizio della partita e la connessione iniziale tra i partecipanti, e una rete tra i giocatori che consente loro di rispondere alle domande in tempo reale. La comunicazione tra i giocatori avviene tramite una semplice architettura Hybrid Decentralized Peer-to-Peer (P2P), che consente di simulare un gioco interattivo senza la necessità di un server che gestisca ogni singola interazione durante il gioco.

L'obiettivo del gioco è che i partecipanti rispondano correttamente a una serie di domande. Il primo giocatore che fornisce una risposta corretta vince il turno, mentre chi risponde erroneamente viene temporaneamente escluso dalla possibilità di rispondere alla domanda. Il gioco termina quando uno dei giocatori indovina un numero di domande prestabilito.

Il progetto è stato sviluppato utilizzando il linguaggio di programmazione Python, sfruttando la libreria `socket` per gestire le comunicazioni tra il server e i partecipanti. Il gioco è pensato per essere eseguito su una piattaforma Unix-like, che offre gli strumenti necessari per gestire la rete in modo efficiente.

1.1 Obiettivi del progetto

Gli obiettivi principali del progetto sono:

- Creare un gioco a quiz in cui più giocatori possano partecipare simultaneamente su una rete locale.
- Gestire la connessione dei giocatori tramite un server centrale, che si occupa dell'inizio della partita e della connessione iniziale tra i vari peer: il server centrale sceglie un presentatore tra i peers, notifica tutti dell'inizio del gioco e delle informazioni per far connettere tutti i peers tra di loro.
- Implementare un sistema che permetta ai giocatori di "prenotare" la risposta, assicurandosi che solo uno possa rispondere alla volta.
- Fornire ad ogni utente un tempo limitato per la risposta.
- Gestire l'errore di risposta, disabilitando temporaneamente il giocatore che ha sbagliato e dando la possibilità agli altri di rispondere.

2 Architettura del Sistema

L'architettura del sistema è progettata utilizzando un modello **Hybrid Decentralized Peer-to-Peer (P2P)** che combina le caratteristiche di un sistema centralizzato e di uno decentralizzato. Questo approccio è stato scelto per sfruttare i vantaggi di entrambe le architetture: un server centrale per facilitare la connessione e la sincronizzazione iniziale dei partecipanti, e una rete decentralizzata per la gestione del gioco tra i peer una volta che la partita è iniziata.

2.1 Panoramica dell'Architettura

Il sistema è suddiviso in due fasi principali:

- **Fase di Connessione Iniziale (Centralizzata)**: Un server centrale è responsabile della connessione iniziale dei peer, della gestione della sincronizzazione e dell'assegnazione del presentatore. In questa fase, i peer si connettono al server per prepararsi all'inizio del gioco.
- **Fase di Gioco (Decentralizzata)**: Una volta che il gioco inizia, i peer comunicano direttamente tra loro senza l'intervento del server. La comunicazione avviene in modalità Peer-to-Peer (P2P) e ogni partecipante funge sia da client che da server, scambiando domande e risposte.

Il server centrale è utilizzato solo per coordinare la fase iniziale della partita e per avviare il gioco, mentre la parte interattiva, durante la quale i partecipanti rispondono alle domande, avviene in modo decentralizzato.

2.2 Descrizione Dettagliata dell'Architettura

1. **Server Centrale**:
    - Gestisce la connessione dei peer e la sincronizzazione iniziale.
    - Seleziona un presentatore tra i partecipanti.
    - Avvia il gioco e fornisce le informazioni necessarie per la connessione tra i peer durante la fase di gioco.
2. **Rete Peer-to-Peer (P2P)**:
    - Dopo l'inizio del gioco, i peer comunicano direttamente tra loro, inviando e ricevendo domande e risposte.

- Ogni partecipante agisce come client e server, senza dipendere dal server centrale.
- È implementato un sistema di "prenotazione della risposta", che impedisce a più di un giocatore di rispondere simultaneamente. Se un giocatore sbaglia la risposta, il sistema disabilita temporaneamente quel partecipante e consente agli altri di rispondere.

2.3 Schema dell'Architettura

Lo schema seguente illustra l'architettura del sistema, mostrando il flusso di informazioni tra il server centrale e i peer, nonché la transizione dalla fase centralizzata alla fase decentralizzata. La comunicazione tra i partecipanti avviene in modalità Peer-to-Peer durante il gioco, ma la sincronizzazione iniziale è gestita dal server centrale.

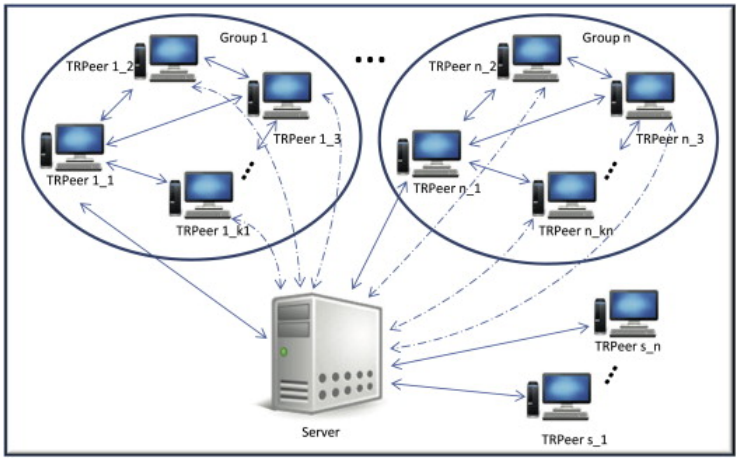

Figure 1: Schema dell'architettura Hybrid Decentralized P2P

Nello schema sopra, possiamo vedere che la fase iniziale (connessione e sincronizzazione) è gestita dal server centrale, mentre la fase di gioco avviene tramite una rete decentralizzata P2P, con i peer che interagiscono direttamente tra di loro. Ogni peer è sia client che server durante il gioco.

2.4 Vantaggi dell'Architettura Hybrid P2P

L'adozione di un'architettura Hybrid Decentralized P2P offre vari vantaggi:

- **Facilità di gestione iniziale**: La fase di connessione e sincronizzazione tra i partecipanti è semplificata grazie al server centrale.
- **Efficienza nella fase di gioco**: Una volta che il gioco è avviato, i peer comunicano direttamente, riducendo il carico sul server e migliorando l'efficienza.
- **Resilienza**: La rete decentralizzata migliora la resilienza del sistema, poiché ogni peer può gestire autonomamente le proprie connessioni.

2.5 Svantaggi

Nonostante i vantaggi, l'architettura presenta anche alcune limitazioni, come:

- **Punto di fallimento singolo**: Il server centrale rappresenta un potenziale punto di fallimento; se il server dovesse guastarsi, l'intera rete potrebbe subire un'interruzione.
- **Scalabilità limitata**: Aumentando il numero di peer, il carico sul server centrale potrebbe diventare difficile da gestire, riducendo la capacità di scalare efficacemente.

2.6 Conclusioni

L'architettura Hybrid Decentralized P2P si rivela ideale per sistemi di piccole e medie dimensioni, dove il numero di partecipanti non è troppo elevato e la gestione del server centrale è ben controllata. Essa offre un buon equilibrio tra centralizzazione e decentralizzazione, rendendo il sistema facile da gestire durante la fase iniziale, pur garantendo una comunicazione efficiente tra i peer durante il gioco.

3 Dettagli Implementativi del Server

Il server del gioco a quiz è implementato utilizzando Python e sfrutta il modulo `socket` per la comunicazione tra client (peer) e server. La logica di gestione dei peer e del flusso di gioco è implementata utilizzando thread separati per ogni connessione. Di seguito sono descritti i dettagli implementativi relativi al server.

3.1 Struttura della Classe `QuizServer`

La classe `QuizServer` gestisce il server di gioco e contiene le funzioni necessarie per l'avvio e la gestione del gioco, nonché la gestione delle connessioni tra peer. Il server ascolta sulla porta specificata per le connessioni in ingresso e accetta le connessioni dai peer che desiderano partecipare al gioco.

- **Attributi della classe:**
  - `self.server`: Un oggetto socket che permette al server di ascoltare le connessioni in ingresso dai peer.
  - `self.peers`: Una lista che tiene traccia delle connessioni attive dei peer registrati, composta da tuple contenenti la connessione e l'indirizzo di ciascun peer.
  - `self.players`: Il numero massimo di peer che possono partecipare al gioco.
  - `self.winning_score`: Il punteggio necessario per vincere la partita.
  - `self.lock`: Un oggetto `Lock` di `threading` utilizzato per gestire l'accesso concorrente alla lista dei peer.
- **Metodi principali:**
  - `__init__`: Il costruttore della classe, che inizializza il server, imposta il numero massimo di giocatori e il punteggio necessario per vincere.
  - `handle_client`: Questo metodo gestisce la connessione di ciascun peer. All'inizio, il server riceve un messaggio di registrazione da parte del peer e lo aggiunge alla lista dei peer registrati. Se il numero di peer registrati è sufficiente, il server avvia il gioco.
  - `start_game`: Questo metodo seleziona un presentatore casuale tra i peer registrati e invia un messaggio a tutti i peer per notificare l'inizio del gioco, fornendo anche informazioni sul presentatore e sul punteggio necessario per vincere.
  - `run`: Questo metodo avvia il server, mettendolo in ascolto su una porta specificata per le connessioni dei peer. Quando un peer si connette, il server avvia un nuovo thread per gestire la connessione.

3.2 Flusso di Esecuzione del Server

Il flusso di esecuzione del server può essere descritto nei seguenti passaggi:

1. Il server avvia la sua esecuzione chiamando il metodo `run`, che mette il server in ascolto su una porta predefinita (ad esempio, 12345).
2. Ogni volta che un peer si connette, il server avvia un nuovo thread che esegue il metodo `handle_client`.
3. Nel metodo `handle_client`, il server riceve un messaggio di registrazione dal peer. Questo messaggio contiene il numero di porta su cui il peer sta ascoltando, permettendo al server di conoscere l'indirizzo effettivo del peer.
4. Se il numero di peer registrati raggiunge il numero massimo di giocatori, il server avvia il gioco chiamando il metodo `start_game`.
5. Nel metodo `start_game`, un presentatore viene selezionato casualmente dalla lista dei peer e viene inviato un messaggio a tutti i peer per notificare l'inizio del gioco e le informazioni necessarie per la partecipazione (come il presentatore e il punteggio necessario per vincere).

3.3 Gestione della Concorrenza

Il server gestisce la concorrenza utilizzando il modulo `threading` di Python. Ogni volta che un nuovo peer si connette, viene avviato un nuovo thread per gestire la connessione in modo che il server possa continuare a ricevere altre connessioni senza bloccare l'esecuzione. Questo approccio permette al server di gestire più connessioni simultaneamente.

Il metodo `self.lock` è utilizzato per proteggere l'accesso alla lista `self.peers`, evitando che due thread possano modificare contemporaneamente la stessa risorsa condivisa. Questo è particolarmente importante quando si aggiungono nuovi peer alla lista durante la registrazione.

3.4 Comunicazione tra il Server e i Peer

Il server comunica con i peer utilizzando il formato `JSON` per inviare messaggi strutturati. Alcuni dei principali messaggi inviati dal server includono:

- **Messaggio di Registrazione**: Quando un peer si registra, invia un messaggio di tipo `"REGISTER"` che contiene il numero di porta su cui il

peer sta ascoltando. Il server risponde con un messaggio `"REGISTERED"` per confermare la registrazione.

- **Messaggio di Inizio Gioco**: Quando il gioco inizia, il server invia un messaggio di tipo `"START"` a tutti i peer, indicando chi è il presentatore e quali sono i peer rimanenti. Il messaggio contiene anche il punteggio necessario per vincere.

3.5 Considerazioni Finali

Il server è progettato per gestire un numero limitato di peer (fino a un massimo di 3, configurabile all'avvio) e avvia il gioco non appena il numero di peer registrati è sufficiente. Nonostante il sistema sia semplice, la gestione delle connessioni concorrenti tramite thread e lock consente una comunicazione fluida tra i peer. Inoltre, l'uso di JSON per i messaggi consente una strutturazione chiara e facilmente estendibile per future funzionalità.

4 Dettagli Implementativi del Peer

Il client (o peer) rappresenta un giocatore o un presentatore all'interno del gioco a quiz. Ogni peer si connette al server centrale per registrarsi e avviare il gioco. A seconda del ruolo assegnato dal server (presentatore o partecipante), il peer esegue una serie di azioni specifiche. Di seguito vengono descritti i metodi principali della classe `QuizPeer`:

- **Metodi principali:**
  - `__init__`: Il costruttore della classe, che inizializza le informazioni necessarie per connettersi al server centrale, come l'indirizzo e la porta del server, la lista dei peer e il ruolo (presentatore o partecipante).
  - `connect_to_server`: Questo metodo si occupa di stabilire la connessione con il server centrale, inviare il messaggio di registrazione, e attendere la risposta dal server. Se la registrazione è andata a buon fine, il peer si prepara a ricevere la notifica dell'inizio del gioco.
  - `listen_for_game`: Una volta registrato, questo metodo resta in attesa di un messaggio dal server che indica l'inizio della partita. Quando il messaggio di inizio partita viene ricevuto, il peer acquisisce il ruolo

di "PRESENTATORE" o "PLAYER" a seconda delle informazioni ricevute dal server e avvia la relativa funzione.

- `start_presenter`: Se il peer è stato designato come presentatore, questo metodo gestisce l'invio delle domande a tutti i partecipanti. Il presentatore invia una domanda a ciascun peer e raccoglie le risposte. La domanda è definita all'interno del codice come una stringa fissa, ma in un'applicazione reale potrebbe essere dinamica.
- `start_player`: Se il peer è un partecipante, questo metodo attende una domanda dal presentatore. Una volta ricevuta, il partecipante può inserire la propria risposta e inviarla al presentatore.

Il peer si connette al server utilizzando una connessione TCP, gestendo la comunicazione tramite socket. La gestione della connessione e l'invio/ricezione dei messaggi avviene in modo sincrono per garantire che ogni fase del gioco sia completata prima di procedere. In particolare:

- Il peer si registra al server inviando un messaggio di tipo `REGISTER` e riceve una risposta di conferma.
- Quando la partita inizia, il peer riceve il messaggio di avvio con le informazioni sul presentatore e gli altri partecipanti.
- Se il peer è il presentatore, invia una domanda a ciascun partecipante.
- Se il peer è un partecipante, riceve una domanda dal presentatore e invia una risposta.

L'architettura a Peer-to-Peer (P2P) è implementata in modo tale che ogni peer è responsabile della connessione e comunicazione diretta con gli altri peer. In questa configurazione, il server centrale agisce come punto di coordinamento iniziale per la registrazione dei peer, ma non gestisce la comunicazione durante la partita.

5 Classe `QuizPeerGUI`

La classe `QuizPeerGUI` gestisce l'interfaccia grafica del gioco tramite la libreria `tkinter`. Essa fornisce una GUI interattiva per il presentatore e il giocatore, aggiornando dinamicamente gli elementi in base agli eventi del gioco e alle azioni degli utenti.

5.1 Attributi principali:

- `peer`: istanza della classe `QuizPeer`, gestisce la connessione con il server e la logica di gioco.
- `server_connected`: stato di connessione con il server.
- `root`: la finestra principale dell'interfaccia grafica.
- `role_label`, `status_label`: etichette per visualizzare il ruolo e lo stato dell'utente.
- `start_button`: pulsante per avviare il peer e la connessione al server.

5.2 Metodi principali:

- `init`: Costruttore che inizializza la GUI e prepara l'interfaccia. Configura la finestra principale e gli stili, creando i frame per il presentatore e il giocatore.
- `start_peer`: Avvia il peer e tenta di connettersi al server. Avvia i thread per la connessione e la gestione del gioco.
- `connect_to_server_with_status`: Gestisce la connessione al server e aggiorna lo stato nella GUI. Mostra un messaggio di errore se la connessione fallisce.
- `update_status`: Aggiorna lo stato nella GUI (ad esempio, "Connesso", "In attesa").
- `show_presenter_gui`: Mostra l'interfaccia grafica del presentatore, nascondendo quella del giocatore.
- `show_player_gui`: Mostra l'interfaccia grafica del giocatore, nascondendo quella del presentatore.
- `send_question`: Invia la domanda e la risposta corretta ai giocatori quando il presentatore preme il pulsante "Invia Domanda".
- `submit_answer`: Gestisce l'invio della risposta del giocatore alla domanda corrente.
- `handle_buzz`: Gestisce la prenotazione del giocatore, notificando al server che il giocatore ha prenotato il turno.

5.3 Funzionamento del programma:

Quando l'utente avvia il programma, viene creata una finestra con un'interfaccia grafica che si adatta dinamicamente in base al ruolo del peer (presentatore o giocatore). Il programma gestisce la connessione al server e la sincronizzazione tra i vari peer, consentendo ai giocatori di rispondere alle domande e di vedere il punteggio aggiornato in tempo reale. La GUI è progettata per essere reattiva e per gestire gli eventi come prenotazioni e timeout durante il gioco.

article graphicx

6 Diagramma UML

Nel diagramma UML seguente, vengono rappresentate le classi principali del sistema Quiz Game: `QuizPeer`, `QuizPeerGUI`, e `QuizServer`. Le classi sono collegate tra loro attraverso relazioni di associazione, indicando le interazioni tra di esse.

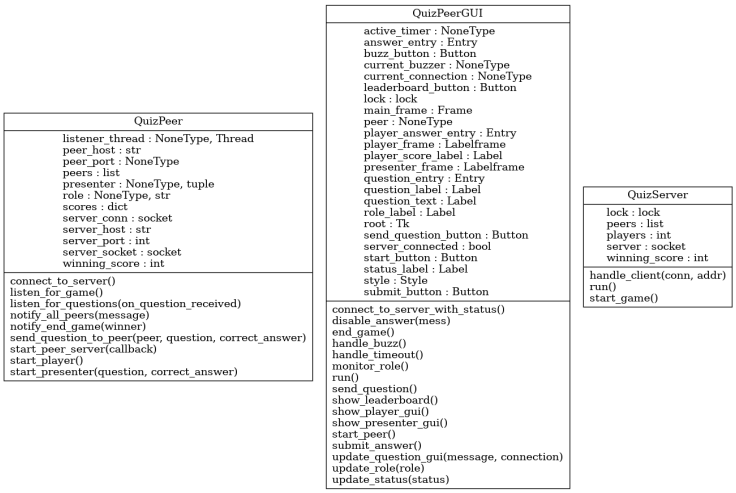

Figure 2: Diagramma UML del sistema Quiz Game

6.1 Descrizione delle Classi e delle Relazioni

- **Classe** `QuizPeer`:
  - Questa classe rappresenta i giocatori (peer) del gioco. Gestisce la connessione al server e la logica di invio e ricezione dei dati relativi

al gioco.

  - Ha una relazione di *associazione* con il `QuizServer`, poiché il peer si connette al server per comunicare.

- **Classe** `QuizPeerGUI`:

  - La classe `QuizPeerGUI` si occupa dell'interfaccia grafica del gioco, creando la GUI per l'interazione dell'utente. Usa il modulo `tkinter` per gestire gli eventi e aggiornare la vista.
  - Essa ha una relazione di *associazione* con la classe `QuizPeer`, poiché la GUI dipende dalle informazioni e interazioni fornite dal peer (giocatore).

- **Classe** `QuizServer`:

  - Il `QuizServer` è il cuore del sistema. Gestisce la logica del gioco, invia domande ai giocatori e raccoglie le risposte. Inoltre, è responsabile del punteggio e della gestione delle sessioni di gioco.
  - Ha una relazione di *associazione* con la classe `QuizPeer`, poiché il server invia e riceve dati dai peer.

7 Guida all'uso del Codice

Questa sezione descrive come avviare e utilizzare il codice sviluppato per il gioco del quiz, con particolare riferimento al server e alla connessione dei client.

7.1 Avvio del Server

Per avviare il server, eseguire il file Python contenente la classe `QuizServer`. Al momento dell'esecuzione, il server richiederà:

- Il numero di giocatori desiderato.
- Il punteggio necessario per vincere la partita.

Il server inizierà ad accettare connessioni dai client fino a raggiungere il numero di giocatori configurato, quindi avvierà il gioco.

7.2 Connessione dei Client

Ogni client deve eseguire un file Python che gli consente di connettersi al server. I client si registreranno inviando un messaggio contenente il proprio indirizzo IP e la porta su cui desiderano ricevere i dati. Una volta raggiunto il numero di client necessario, il server avvierà il gioco e selezionerà un presentatore.

7.3 Completamento del Gioco

Il gioco procede con le domande e le risposte fino a quando un giocatore non raggiunge il punteggio definito. A quel punto, il gioco si conclude e il vincitore viene annunciato.

7.4 Esecuzione del Codice

Per eseguire il codice, seguire i seguenti passaggi:

1. Avviare il server con il comando:

   ```
   python server.py
   ```

2. Avviare i client, ognuno in un terminale separato, con il comando:

   ```
   python quiz_game_gui.py
   ```

7.5 Errori Comuni

In caso di errori durante l'esecuzione del gioco, i seguenti problemi potrebbero verificarsi:

- **Limite di giocatori raggiunto**: Se il numero massimo di client è già connesso, il server rifiuterà ulteriori connessioni.
- **Porta non valida**: Se un client invia una porta non valida, il server restituirà un errore.
- **Connessione rifiutata**: Se il server non è in esecuzione o non è raggiungibile, la connessione dei client fallirà.

8 Conclusioni

In questo lavoro, ho sviluppato un gioco del quiz che consente di gestire domande e risposte in modo semplice ed efficace. Il progetto ha funzionato bene per quanto riguarda la gestione del punteggio e l'interfaccia utente. In futuro, sarebbe interessante arricchire l'esperienza con feedback visivi e sonori per rendere il gioco ancora più coinvolgente. Complessivamente, il progetto offre una solida base per eventuali sviluppi futuri.