Documentazione Applicazione Morra Cinese Distribuita

Daniele Tortoli, matricola: 153371

Luglio 2023

Indice

Elenco delle figure

1 Descrizione dei Requisiti

1.1 Requisiti

All'esame deve essere mostrata l'applicazione in esecuzione e deve essere presentata una relazione che descrive il progetto e deve comprendere:

- Descrizione dei requisiti, ed in particolare delle funzionalità messe a disposizione (es. tramite SRS);
- Descrizione dell'architettura (ad es. tramite diagramma a blocchi o UML);
- Descrizione dei protocolli usati (client-server o peer-to-peer, ad es. tramite diagrammi UML).

1.2 Scopo

Lo scopo di questo progetto è sviluppare un sistema distribuito utilizzando il framework Pyro per implementare un gioco di "Sasso, Carta, Forbice" in un ambiente online. Il sistema consente ai giocatori di registrarsi, partecipare al gioco e ricevere gli esiti delle partite. L'obiettivo principale del progetto è fornire un'interfaccia di gioco intuitiva e sicura all'interno di un ambiente distribuito.

1.3 Finalità del Progetto

La finalità del progetto è dimostrare l'efficacia dell'uso di un sistema distribuito tramite l'implementazione del gioco "Sasso, Carta, Forbice". Utilizzando il framework Pyro, il sistema si basa su una comunicazione client-server per permettere l'interazione tra i giocatori. L'obiettivo generale è fornire una soluzione di gioco scalabile ed efficiente nel contesto distribuito del sistema di gioco. Il progetto mira a soddisfare le esigenze di un'esperienza di gioco distribuita, garantendo allo stesso tempo un accesso sicuro e controllato alle funzionalità del gioco.

2 Descrizione del progetto

Il nostro sistema è un'applicazione di gioco "Sasso, Carta, Forbice" distribuita che funziona secondo un modello client-server. Il server gestisce il gioco, mentre i client possono unirsi, giocare e vedere i risultati del gioco. Inoltre possono richiedere il rematch dopo che la partita è terminata.

2.1 Funzionalità

- **Registrazione dei giocatori**: I client possono registrarsi per partecipare a un gioco.
- **Giocare una partita**: Una volta registrati, i client possono fare una mossa (sasso, carta o forbice).
- **Risultati del gioco**: Dopo che entrambi i giocatori hanno fatto una mossa, il server determina il vincitore e comunica i risultati ai client.
- **Rematch**: Dopo la fine di un gioco, i client possono richiedere un nuovo gioco. Se entrambi i client accettano, il server resetta lo stato del gioco (mantenendo il punteggio) e inizia una nuova partita.

3 Descrizione dell'Architettura

L'architettura del nostro sistema è composta da un server di gioco e più client. Il server gestisce lo stato del gioco, mentre i client interagiscono con il server per partecipare al gioco.

3.1 Diagramma UML dell'architettura

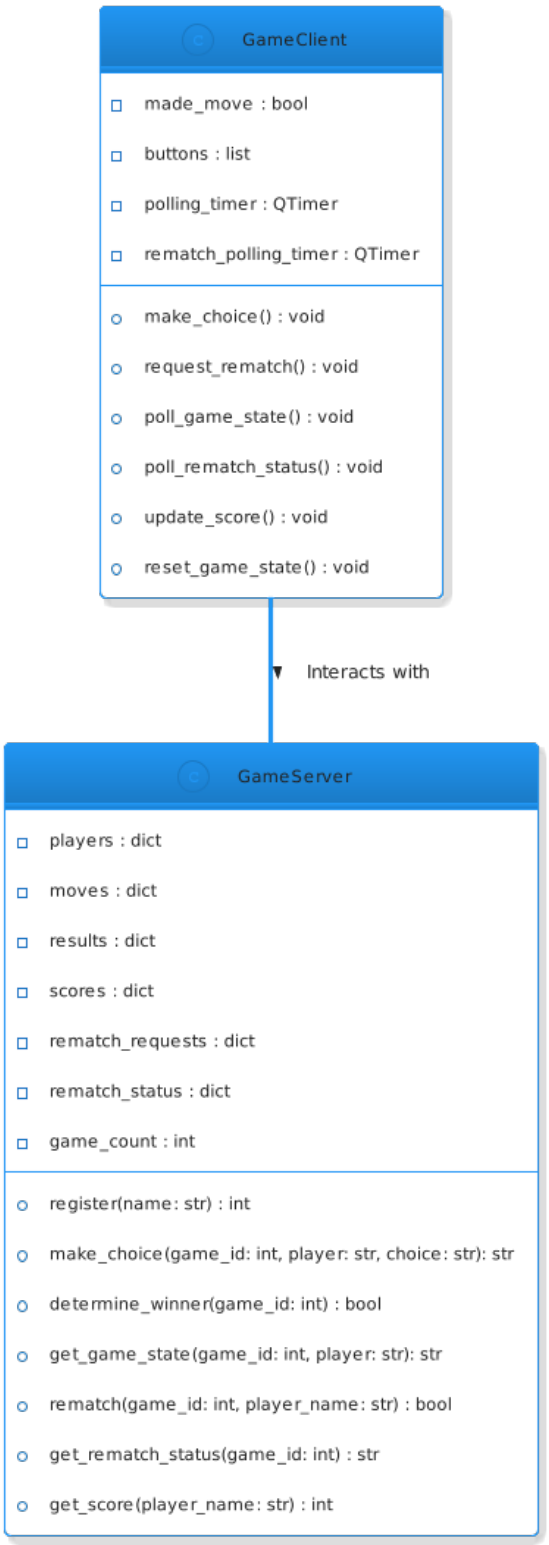


Figura 1: Diagramma UML dell'architettura

4 Descrizione dei Protocolli Usati

Il nostro sistema utilizza un protocollo client-server. I client inviano richieste al server e il server risponde a queste richieste. Ad esempio, un client può inviare una richiesta di registrazione al server, o inviare una mossa da giocare.
Tutte le interazioni tra il client e il server avvengono tramite richieste e risposte, utilizzando il polling per controllare lo stato del gioco. Questo rende il nostro sistema adatto per un ambiente distribuito, poiché non c'è bisogno di mantenere una connessione persistente tra client e server.
Le interazioni specifiche tra client e server sono descritte nel seguente diagramma di sequenza:

Figura 2: Diagramma di sequenza

4.1 Requisiti funzionali

Il sistema si basa su una struttura client-server organizzata come segue.

4.1.1 Programma client

È uno script Python denominato "client.py" che:

- Inizializza i due proxy Pyro: GameServer e GameState;
- Gestisce la registrazione del giocatore;
- Fornisce un menu per le interazioni con il GameServer, consentendo al giocatore di selezionare le azioni desiderate (ad es. scelta di sasso, carta o forbici).

Il client utilizza gli oggetti remoti contattando i proxy Pyro per interagire con essi. Renderà gli oggetti remoti trasparenti per il giocatore.
Il client all'avvio dovrà effettuare alcune operazioni di setup:

- Individua l'URI dell'oggetto necessario utilizzando il Pyro Name Server (nel nostro caso "PYRO:MorraCinese.game");
- Crea un oggetto speciale "Proxy" che effettua le chiamate all'oggetto remoto. Una volta ottenuto il "Proxy", il client potrà utilizzare tali oggetti remoti come se fossero oggetti locali:

```
game_server = Pyro5.api.Proxy("PYRO:MorraCinese.game@localhost:55894")
```

4.1.2 Programma server

È uno script Python denominato "server.py" che:

- Inizializza e gestisce gli oggetti remoti, come le partite e i giocatori;
- Gestisce le richieste del client, come la registrazione dei giocatori, la gestione delle mosse e la determinazione del vincitore;
- Espone i metodi di Pyro per l'interazione con il client.

5 README: Descrizione funzionalità, guida all'installazione e all'esecuzione

5.1 Morra Cinese con Pyro5

Questo progetto implementa una semplice versione di Morra Cinese (Rock, Paper, Scissors) utilizzando Pyro5 per la comunicazione tra client e server. Il progetto è costituito da due file principali: `server.py` e `client.py`.

5.2 Funzionalità del Codice

5.2.1 Server.py

Il server gestisce l'intero stato del gioco. Le funzionalità principali includono:

- Gestione delle partite, dei giocatori, delle mosse, dei risultati e delle richieste di rematch.
- Mantenimento dello storico dei punteggi dei giocatori.

Metodi Principali

- `register(name)`: Registra un nuovo giocatore per una partita.
- `make_choice(game_id, player, choice)`: Consente a un giocatore di fare una mossa.
- `determine_winner(game_id)`: Determina il vincitore di una partita.
- `get_game_state(game_id, player)`: Ritorna lo stato attuale della partita per un determinato giocatore.

- `rematch(game_id, player_name)`: Gestisce le richieste di rematch.
- `get_rematch_status(game_id)`: Ritorna lo stato attuale della richiesta di rematch.
- `get_score(player_name)`: Ritorna il punteggio attuale di un giocatore.

5.2.2 Client.py

Il client crea un'interfaccia utente per il gioco, permettendo ai giocatori di interagire con il server.

Funzionalità Principali

- Permette ai giocatori di fare una mossa e mostra il risultato della partita.
- Permette ai giocatori di richiedere un rematch alla fine di una partita.
- Mostra lo stato attuale della partita e il punteggio dei giocatori.

5.3 Installazione e Utilizzo

1. Assicurati di avere installato Python e Pyro5 sul tuo computer.
2. Clona o scarica questo repository.
3. Esegui `server.py` per avviare il server di gioco.
4. Esegui `client.py` per ogni giocatore che vuole unirsi al gioco.
5. Segui le istruzioni visualizzate sull'interfaccia del client per giocare.

5.4 Dipendenze

- Python 3.8+
- Pyro5
- PyQt5

Nota: Assicurati di installare tutte le dipendenze prima di eseguire il codice. Puoi installare le dipendenze con il seguente comando:

```
pip install pyro5 pyqt5
```

5.5 Struttura del Codice

5.5.1 Server.py

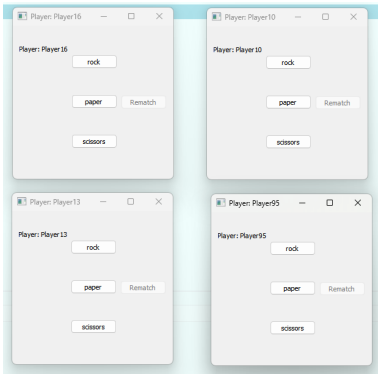
```
import Pyro5.api

@Pyro5.api.expose
class GameServer(object):
    def __init__(self):
        # Initialize game state

    def register(self, name):
        # Register a new player for a game

    def make_choice(self, game_id, player, choice):
        # Allow a player to make a move
```

```
    def determine_winner(self, game_id):
        # Determine the winner of a game

    def get_game_state(self, game_id, player):
        # Return the current state of the game for a certain player

    def rematch(self, game_id, player_name):
        # Handle rematch requests

    def get_rematch_status(self, game_id):
        # Return the current state of the rematch request

    def get_score(self, player_name):
        # Return the current score of a player
```

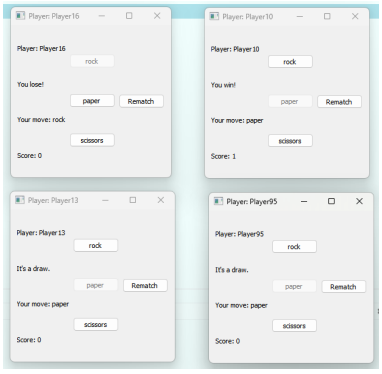
5.5.2 Client.py

```
import Pyro5.api

class GameClient(object):
    def __init__(self):
        # Initialize client state

    def make_choice(self):
        # Let the player make a move

    def request_rematch(self):
        # Let the player request a rematch

    def get_game_state(self):
        # Show the current state of the game

    def get_score(self):
        # Show the current score of the player
```

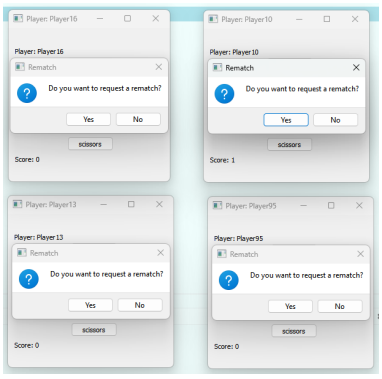
6 Immagini applicazione



(a) Client: Registrati



(b) Client: Risultati



(c) Client: Rematch

Figura 3: Tre immagini