Relazione: GameGuessr

Riccardo Aiello - Matricola: 0124002251

Contents

1 Introduzione

Il sistema sviluppato rappresenta un gioco a quiz in rete con un interfaccia realizzata in pygame, implementato con protocollo P2P di tipo **hybrid decentralized**. Il progetto prevede un server centrale che coordina la connessione dei peers e gestisce l'inizio del gioco. Quando il numero minimo di peers è connesso, il server sceglie un presentatore che invia le domande ai partecipanti. Il primo peer a rispondere correttamente vince.

2 traccia richiesta

- Gruppo 1 studente

Scrivere un gioco con protocollo P2P di tipo hybrid decentralized che simula un gioco a quiz. Creare un server centrale a cui tutti i peers si connettono per entrare nella partita Quando tutti i giocatori sono pronti (4 peers) 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. Il presentatore invia una domanda a tutti i partecipanti. I partecipanti possono rispondere tutti contemporaneamente. La prima richiesta servita (corretta) sarà il vincitore.

3 Descrizione dell'Architettura

L'architettura del sistema si basa su un modello client-server ibrido. Il server agisce come punto di coordinamento per avviare il gioco, mentre i peers comunicano con il presentatore durante la fase di gioco.

3.1 Componenti del Sistema

Le principali componenti del sistema sono:

- **Server centrale**: Gestisce la registrazione dei peers, seleziona il presentatore e distribuisce le informazioni necessarie per la connessione peer-to-peer tra i giocatori.
- **Peers**: Si connettono al server centrale per entrare nel gioco e partecipano alla fase di quiz comunicando direttamente con il presentatore.
- **Presentatore**: Un peer selezionato dal server che ha il compito di inviare le domande ai partecipanti.

4 Dettagli Implementativi

4.1 Server

Il server centrale si occupa di:

1. Accettare le connessioni dai peers.
2. Selezionare casualmente il presentatore tra i peers connessi.

3. Inviare a tutti i peers le informazioni necessarie per la connessione al presentatore.

Tutti i passaggi discussi sono contenuti all'interno del seguente metodo.

```python
def start(self):  1 usage  RiccardoHihello
    try:
        self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,  value: 1)
        self.server_socket.bind((self.server, self.port))
    except socket.error as e:
        print(str(e))
        return

    self.server_socket.listen(4)
    print("In attesa di una connessione")

    while len(self.clients) < 4: #accetta fino a 4 giocatori
        connection, addr = self.server_socket.accept()
        client = Client(connection, addr)
        self.clients.append(client)
        print("Connessione accettata da:", format(addr))

    presenter = random.choice(self.clients)
    presenter.conn.send("PRESENTATORE".encode())
    self.clients.remove(presenter)

    for client in self.clients:
        client.conn.send(f"CONNECT:{presenter.addr[0]}:{presenter.addr[1]}".encode()) #invio dei dati per iniziare
                                                                                      #la connessione col presenter

    self.server_socket.close()
```

Figure 1: Server.py

4.2 Peer

4.2.1 presentatore

Il presentatore è responsabile di:

1. Accettare le connessioni dei peers

```python
def listen_for_peers(self):  # funzione per l'ascolto dei peers dal peer host  1 usage  RiccardoHihello
    local_ip, local_port = self.client.getsockname()
    self.client.close()
    self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.client.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,  value: 1)
    self.client.bind((local_ip, int(local_port)))
    self.client.listen(3)

    print("Presentatore in ascolto su:", local_ip, "porta:", local_port)
    threading.Thread(target=self.accept_peers, daemon=True).start()
```

```python
def accept_peers(self):  1 usage  RiccardoHihello
    try:
        while len(self.connected_peers) < 3:
            with self.lock:
                conn, addr = self.client.accept()
                nickname = conn.recv(1024).decode() # nella connessione iniziale il peer
                                                    # invia il suo nickname
                peer = Peer(conn, nickname)
                self.connected_peers.append(peer)
                print(f"Peer connesso: {addr}, Nickname: {nickname}")
            threading.Thread(target=self.handle_peers, args=(peer,), daemon=True).start()
            # creazione del thread per gestire la connessione al singolo peer
    except Exception as e:
        print(f"Errore nel thread di ascolto: {e}")
    finally:
        self.client.close()
```

2. Inviare le informazioni ai peers (domande, conferme di risposta)

```python
def send_to_peers(self, data):  # invio domanda ai peers  2 usages  RiccardoHihello
    for peer in self.connected_peers:
        try:
            peer.conn.send(data.encode())
        except Exception as e:
            print(f"Errore invio dati al peer: {e}")
```

3. Gestire i messaggi dei peers, azione implementata usando una lock per gestire il possibile invio di più risposte contemporanee

```python
def handle_peers(self, peer): #funzione per la gestione dei messaggi inviati dai peers al presenter
    while True:
        try:
            data = peer.conn.recv(1024).decode()
            if not data:
                break
            elif data.startswith("risposta:") and self.current_question:
                with self.lock:
                    if self.correct_answer and data.strip() == self.correct_answer:
                        self.current_question = None
                        peer.add_points()
                        self.send_to_peers("risposta corretta ricevuta")
                    elif self.correct_answer and data.strip() != self.correct_answer:
                        peer.conn.send("risposta sbagliata".encode())
                    else:
                        print("Errore nell'invio della verifica di risposta")
            elif data.startswith("punteggi?"):
                peer.conn.send(self.get_statistics().encode())
            else:
                print(f"Dati non riconosciuti: {data}")
        except Exception as e:
            print(f"Errore gestione della connessione al peer: {e}")
            break

    print(f"connessione persa")
    self.connected_peers.remove(peer)
    peer.conn.close()
```

4.2.2 giocatore

Il giocatore è responsabile di:

1. Connettersi al server.

```
def server_connection(self):  # funzione per la connessione al server  1 usage  RiccardoHihello
    try:
        self.client.connect(self.server_address)
        threading.Thread(target=self.handle_server, args=(), daemon=True).start()
        return True
    except Exception as e:
        print(f"Errore di connessione al server: {e}")
        return False
```

   In base al primo messaggio che il server invierà il peer potrà essere scelto come presentatore, e quindi iniziare ad ascoltare per la connessione dei giocatori mentre nel caso dei giocatori sarà ricevuta una stringa contenente indirizzo e porta del presentatore a cui connettersi

```
def handle_server(self):  # funzione per gestire le risposte del server
    response = self.client.recv(1024).decode()
    if response == "PRESENTATORE":
        self.is_presenter = True
        self.listen_for_peers()
    elif response.startswith("CONNECT:"):
        ip, port = response.strip("CONNECT:").split(":")
        self.presenter_addr = (ip, int(port))
    else:
        print(f"Errore nella ricezione del messaggio")
```

Figure 2: gestione dei messaggi del server

2. Connettersi al presentatore e attendere l'inizio del gioco. Il peer appena iniziata la connessione invia il nickname scelto al presentatore.

```
def connect_to_presenter(self):  # funzione per connettersi al peer host  1 usage  RiccardoHihello
    try:
        conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        conn.connect(self.presenter_addr)
        conn.send(self.nickname.encode())
        peer = Peer(conn, self.nickname)  # creazione di un oggetto Peer per il presentatore
        self.connected_peers.append(peer)
        threading.Thread(target=self.handle_presenter, args=(peer,), daemon=True).start()
        return True
    except Exception as e:
        print(f"Error nella connessione al presentatore: {e}")
        return False
```

Figure 3: connessione al presentatore

3. Inviare la risposta al presentatore una volta ricevuta la domanda.

```python
def send_answer(self, data):  # invio della risposta del peer al presentatore
    if not self.is_presenter:
        presenter_peer = self.connected_peers[0]
        try:
            presenter_peer.conn.send(data.encode())
        except Exception as e:
            print(e)
```

Figure 4: invio delle risposte dal peer

Di seguito è riportato il metodo che gestisce i messaggi da parte del presentatore per il peer:

```python
def handle_presenter(self, peer): #funzione per la gestione dei messaggi inviati dal presenter
    # funzione per l'organizzazione della comunicazione tra peers
    while True:
        try:
            data = peer.conn.recv(1024).decode()
            if not data:
                break
            elif data == "risposta corretta ricevuta":
                self.current_question = None
                self.correct_answer = None
                self.send_answer("punteggi?")
            elif data == "risposta sbagliata":
                self.correct_answer = False
            elif data.startswith("domanda"):
                self.current_question = data.strip("domanda")
            elif data.startswith("Punteggi"):
                self.points_stats = data.strip("punteggi:")
        except Exception as e:
            print(f"Errore gestione della connessione al presenter: {e}")
            break
    print(f"connessione persa")
    self.connected_peers.remove(peer)
    peer.conn.close()
```

Figure 5: Connection.py

Tutti i metodi discussi finora sono inseriti all'interno della classe **Connection.py**, creata con lo scopo di diminuire le responsabilità dalla classe Client.

La classe **Client** si occupa di disegnare l'interfaccia di gioco e di gestire l'input dell'utente.

5 Diagrammi UML

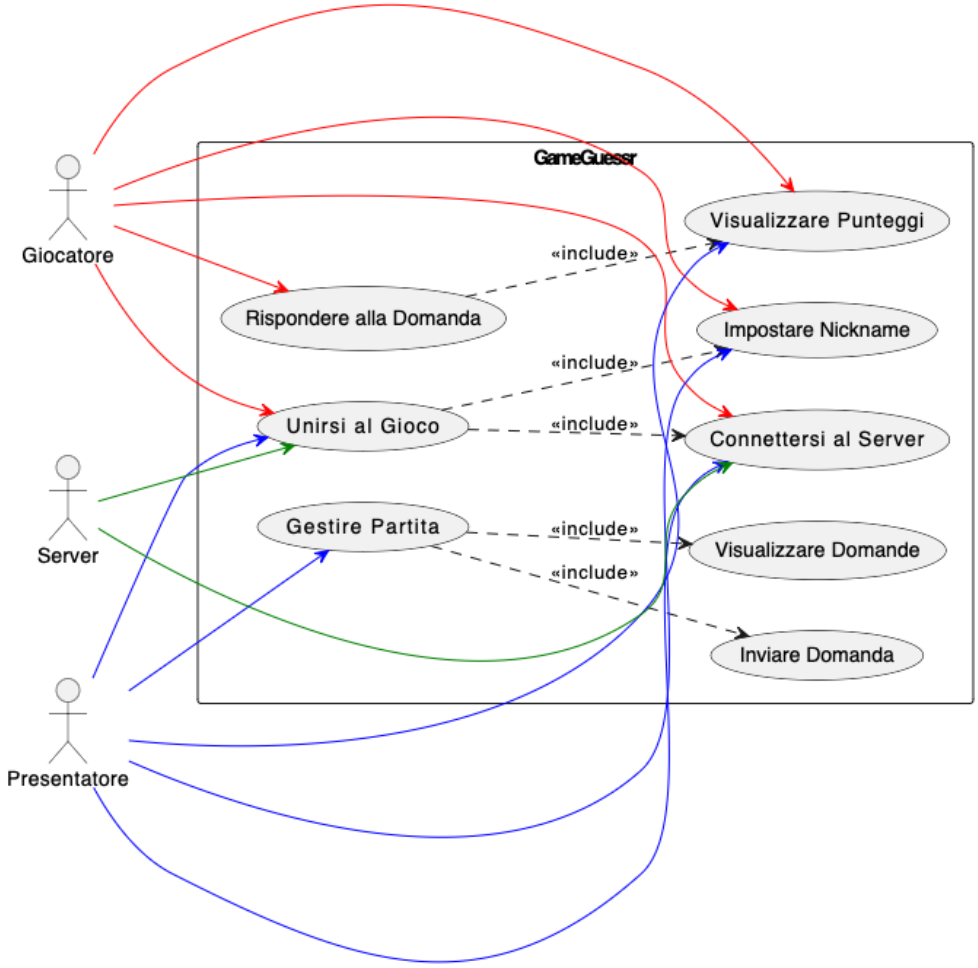

Figure 6: Diagramma dei casi d'uso

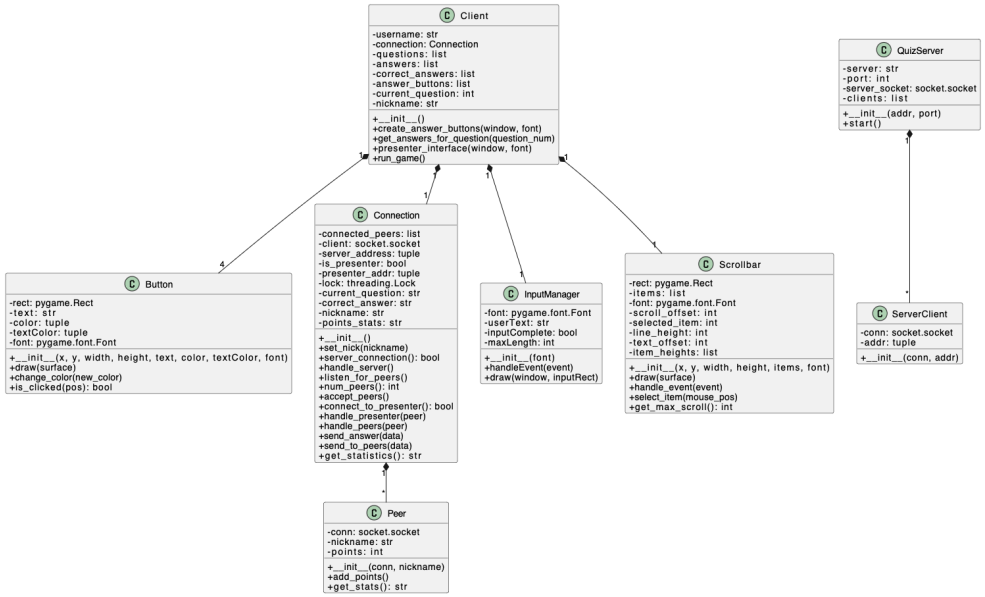

Figure 7: Diagramma delle classi

Istruzioni per l'installazione e l'avvio

Requisiti: **Python 3.x**

1. **Dipendenze**

   ```
   pip install pygame
   ```

2. **Repository**

   ```
   git clone https://github.com/Riccardohihello/GameGuessr
   ```

3. **Naviga nella directory del progetto**

   ```
   cd GameGuessr
   ```

4. **Avvia il server**

   ```
   python Server.py
   ```

5. **Avvio dei client**

   ```
   python Client.py
   ```

   (fino a un massimo di 4 incluso il presentatore)

6. **Avvio tramite script (opzionale)**

   ```
   bash launchScript.sh
   ```

   Ho incluso anche uno script bash all'interno del progetto per lanciare in sequenza tutte le istanze.

5.1 Istruzioni di Gioco

- Ogni peer si connette al server per registrarsi.
- Una volta raggiunto il numero di peers (4), il server seleziona un presentatore.
- Il presentatore può inviare una domanda ai partecipanti (selezionando la domanda con il mouse e poi premendo 'invia domanda').
- I peers rispondono (premendo la lettera desiderata) e la prima risposta corretta riceve punti.

5.2 Interfaccia

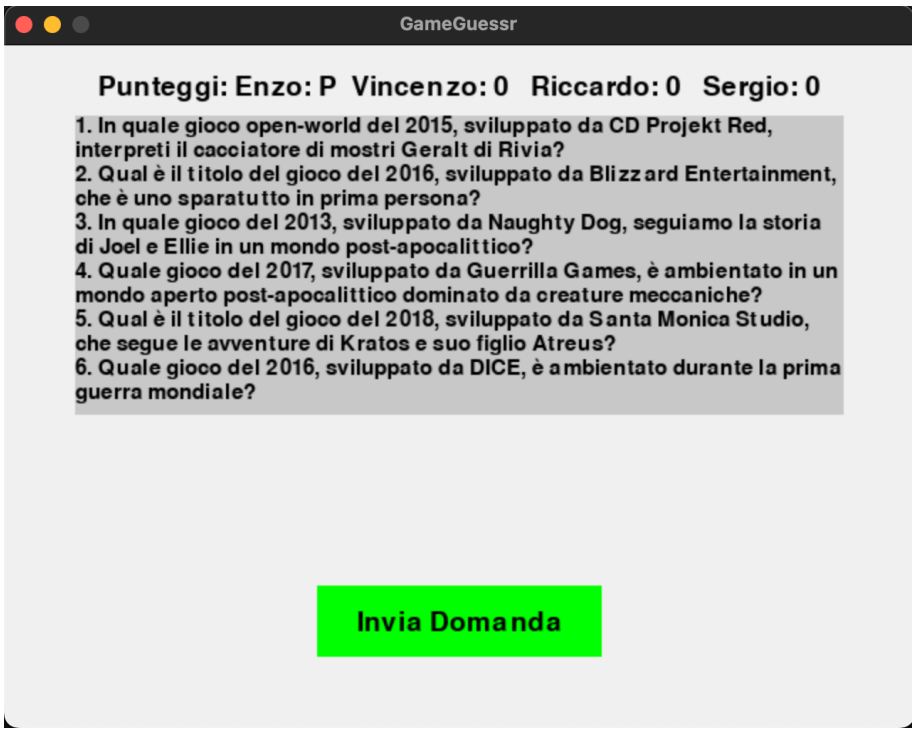

Figure 8: Interfaccia del presentatore

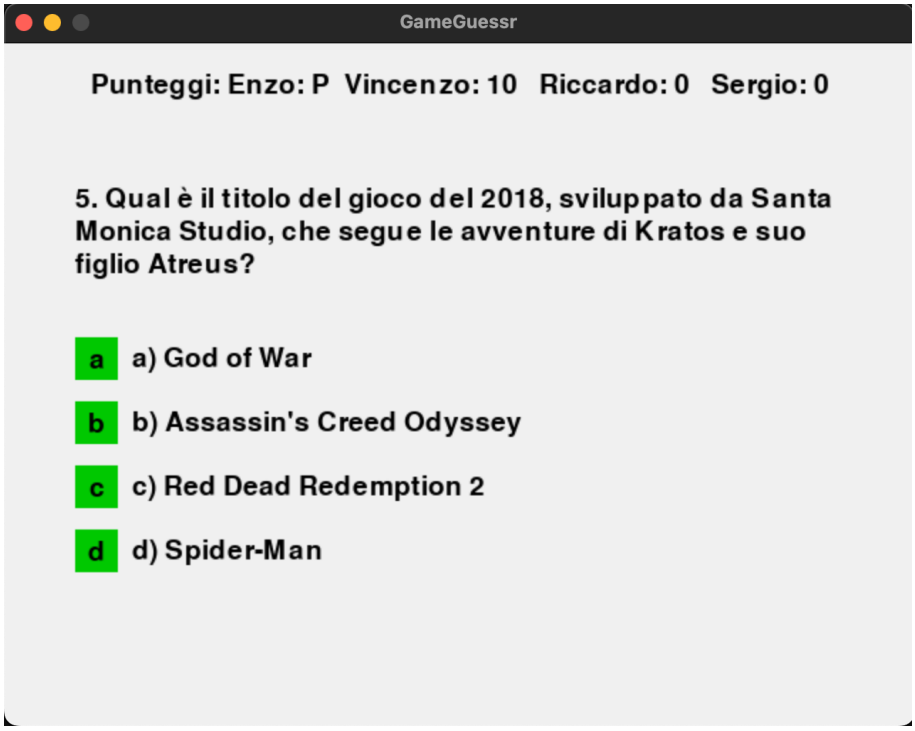

Figure 9: Interfaccia del peer