Programmazione di reti - Relazione di progetto

Costruzione di un sistema di gioco
tramite una chat TCP-based (Traccia 3)

Giovanni Antonioni 0000922658
Luca Rubboli 0000923420

Indice

1 Descrizione del progetto

1.1 Analisi dei requisiti

Il gruppo si pone come obiettivo l'implementazione di un software in linguaggio Python che permetta di sviluppare un ChatGame.
Un ChatGame comprende due entità principali, un server (Master) e uno o più client (Players), i quali rappresentano i giocatori partecipanti ad una partita.
L'applicazione in questione deve permettere al server di definire un ruolo per ogni client e, una volta raggiunto il numero esatto di partecipanti per poter comporre una stanza di gioco, avviare la partita.
La partita ha una durata temporale limitata, e nel corso di essa il server propone in prima battuta al giocatore 3 opzioni, 2 di queste portano ad una domanda, mentre la terza, detta opzione trabocchetto, comporta l'esclusione dalla stanza di gioco e quindi l'eliminazione dalla partita corrente.
Ogni domanda prevede una sola risposta giusta, che porta all'incremento di un'unità del punteggio del giocatore, mentre una risposta negativa decurta il punteggio di quest'ultimo sempre di un'unità.
Quando la partita termina, se all'interno della stanza di gioco sono ancora presenti dei client, il server ne decreterà il vincitore, e rimuoverà poi tutti i giocatori per poter incominciare una nuova sessione.

1.2 Descrizione concettuale delle entità del gioco

- Server: Il server è l'entità centrale del gioco. Instaura il collegamento con i client, avvia e termina la partita e ne controlla l'andamento memorizzando i punteggi dei giocatori.
- Client: Il client rappresenta l'entità giocatore. Interagisce con il server tramite uno scambio di messaggi.

1.3 Design dell'applicazione

Il modello architetturale dell'applicazione è basato su una struttura client - server in cui viene stabilita una connessione TCP in grado di mettere in comunicazione tali entità.
Il server ammette la gestione parallela di più client attraverso il multithreading, alla connessione di un nuovo giocatore (tramite client), infatti, viene avviato un nuovo thread in grado di gestire le varie fasi di gioco per quel singolo client.
Per quanto riguarda invece la durata limitata della partita la gestione temporale è affidata ad un thread timer separato, che ne scandisce il momento di inizio e di fine.

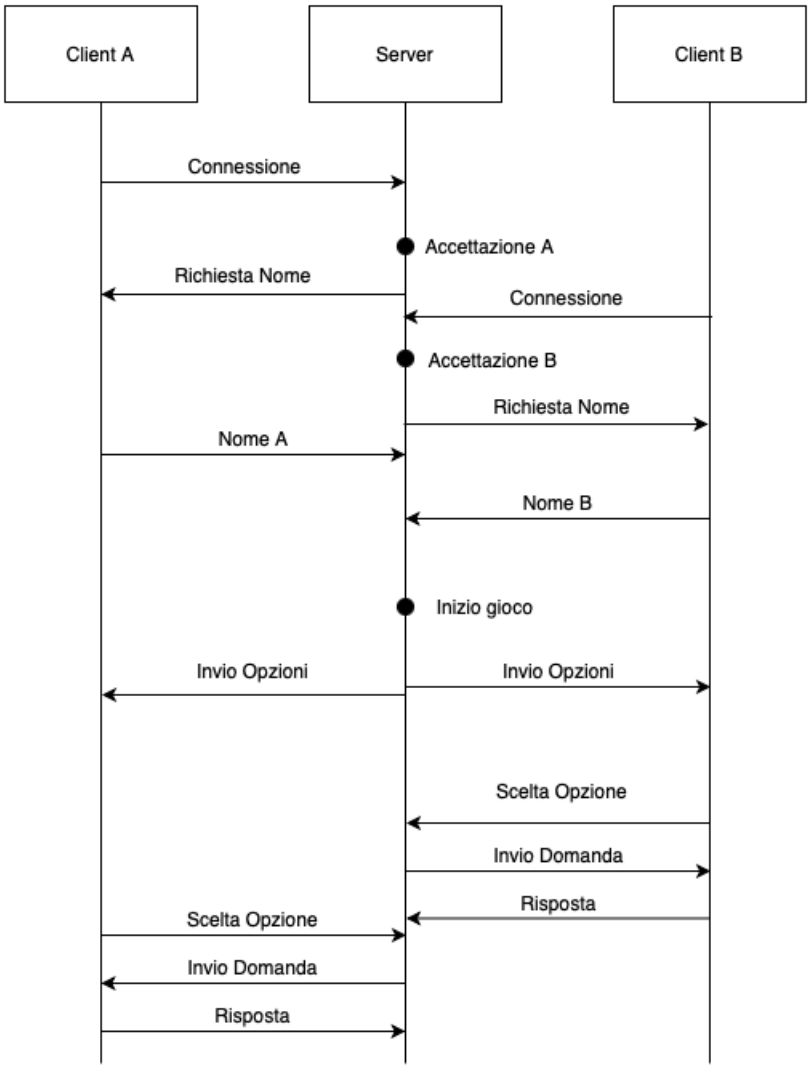

Figura 1: Flow dell'applicazione

1.4 Fasi di gioco

- Inizializzazione: In questa fase viene avviato il server, successivamente il pulsante Start consente al server di mettersi in ascolto di eventuali richieste di connessione da parte di uno o più client.

- Connessione: In questa fase vengono avviati uno o più client, che mandano una richiesta di connessione TCP al server mediante socket, all'avvenuta connessione il server richiede il nome con cui il client decide di iniziare il gioco, necessario per la sua identificazione all'interno della partita in corso (a client diversi devono corrispondere nomi differenti).

- Sala di attesa: In questa fase i client autenticati tramite nome saranno messi in attesa, in questo stato il client non potrà comunicare in alcun modo con il server.
  L'attesa perdurerà finchè non verrà raggiunto l'esatto numero di partecipanti per avviare la partita.

- Gioco: Dopo aver raggiunto il numero di partecipanti per avviare la partita, il server avviserà tutti i partecipanti di prepararsi, dopo un breve lasso di tempo infatti incomincerà il gioco e i client verranno riportati al loro stato originario, permettendo nuovamente la loro comunicazione con il server.
  Durante tutta la durata della partita, il server richiede ad ogni giocatore di effettuare delle scelte, ognuna di esse si compone di 3 proposte, 2 di queste celano una domanda che permetterà al client di incrementare di un'unità il suo punteggio in caso di risposta giusta o di decrementarlo della stessa quantità in caso di risposta errata.
  La proposta rimanente, detta opzione trabocchetto, comporterà l'espulsione del giocatore dalla partita in corso, escludendolo anche dalla classifica finale.

- Vincitore: Terminata la durata temporale di una partita, i giocatori rimanenti verranno riportati ad uno stato di attesa, il server decreterà poi il vincitore e, dopo averlo comunicato ai giocatori ancora in partita e fatto passare un breve intervallo di tempo per permetterne la visione, cesserà le comunicazioni con i client che verranno chiusi.
  Il server è pronto a poter gestire una nuova partita.

2 Architettura dell'applicazione

2.1 Presentazione degli UML

Per modellare l'applicazione abbiamo strutturato la logica del dominio in classi separate.

2.2 Client - Server

In questa prima figura possiamo osservare la parte centrale dell'applicazione, ovvero le entità Client e Server descritte in precedenza.

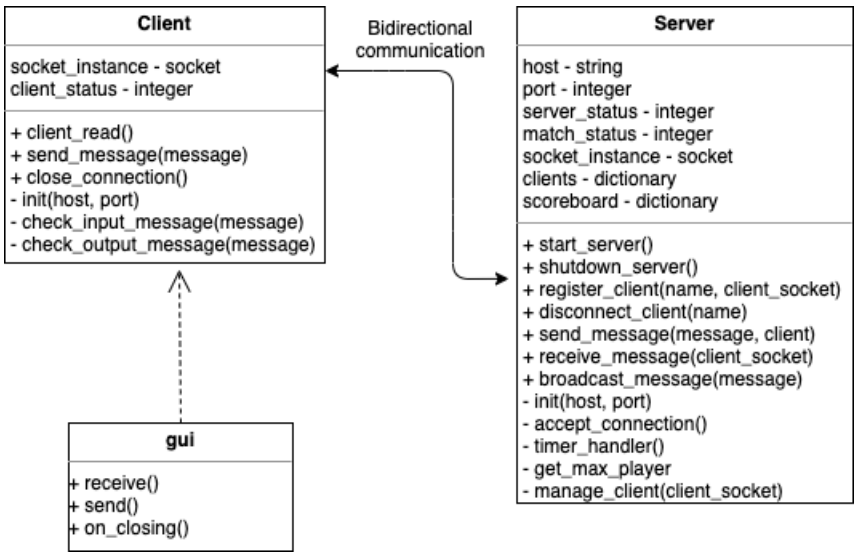

Figura 2: Client, Server

Per poter visualizzare i messaggi scambiati abbiamo sviluppato una classe gui la quale adopera un'istanza di Client per la visualizzazione dei messaggi scambiati.

Alla creazione, un client, istanzia una nuova socket la quale tenta di connettersi al server.

2.3 Server

Come già accennato il Server è l'unità centrale del gioco che coordina le varie fasi.

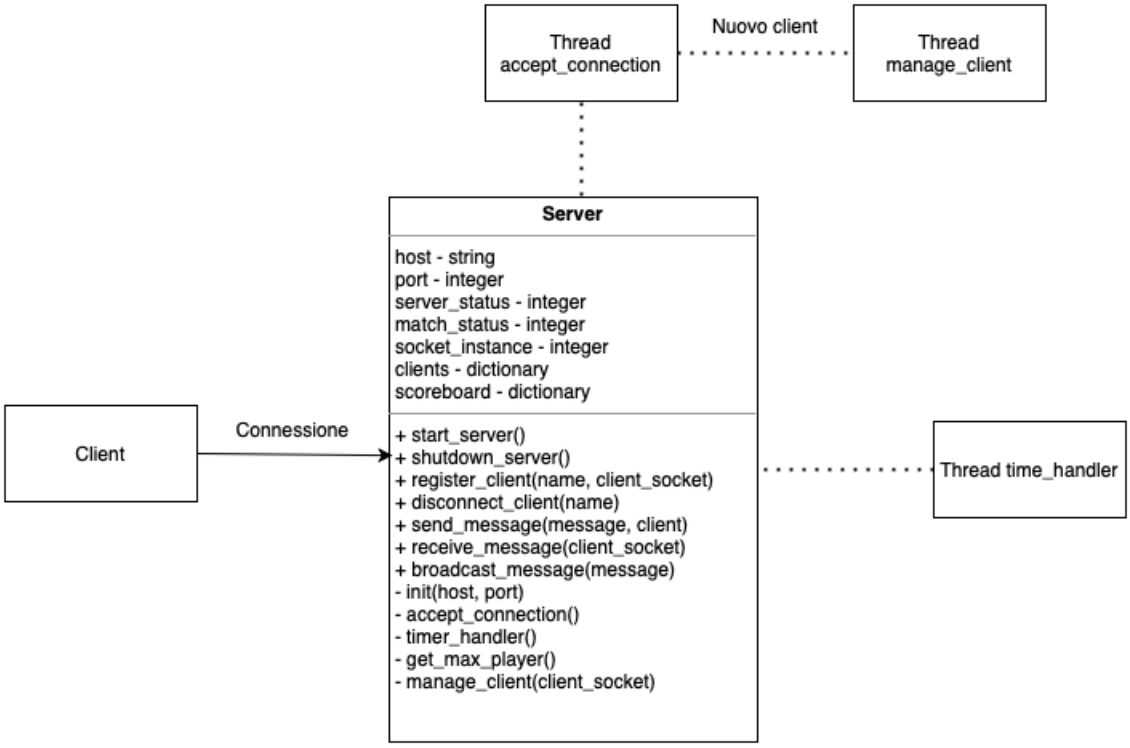

Figura 3: Server

2.3.1 Avvio

Tramite il comando `start_server()` il server viene avviato e si susseguono le seguenti fasi

- Viene instaurata un nuova socket TCP, incaricata di riceve nuove comunicazioni dall'esterno.
- Viene creato un nuovo thread incaricato di accettare i nuovi client che si connettono. Questo esegue in parallelo il metodo `accept_connection()`.

2.3.2 Gestione client

Per ogni client il server crea in parallelo un nuovo thread che esegue il metodo `manage_client()`. Questo permette di controllare un singolo giocatore permettendone:

- La registrazione del nome e l'assegnazione di un ruolo.
- L'immissione nella sala d'attesa.
- Gestione delle opzioni e delle domande/risposte con annessa modifica del punteggio.

2.3.3 Timer

Al raggiungimento dell'esatto numero di giocatori, il server avvia un'ulteriore thread che esegue il metodo `timer_handler()` e dà avvio alla partita mantenendo un timer.
Allo scadere di questo viene fatto visualizzare il giocatore avente punteggio più alto, i client vengono scollegati ed il server viene spento.

2.3.4 Shutdown

Il Server prevede anche una funzione di shutdown la quale scollega i giocatori attualmente registrati chiudendo la connessione socket.

2.3.5 Server Status:

- `SERVER_RUNNING_STATUS`: Il Server è attualmente in esecuzione. I client possono instaurare una connessione con esso.
- `SERVER_STOPPED_STATUS`: Il Server è spento, nessuna connessione in ingresso è accettata.

2.3.6 Server match status:

- `SERVER_MATCH_STARTED`: Nel Server è stata avviata una partita, non sono ammessi ulteriori client i quali, al momento della registrazione, verranno fatti disconnettere.
- `SERVER_MATCH_PAUSED`: Nel Server non è attiva alcuna partita. I client possono connettersi liberamente.

2.4 Client

Il client rappresenta l'entità giocatore.

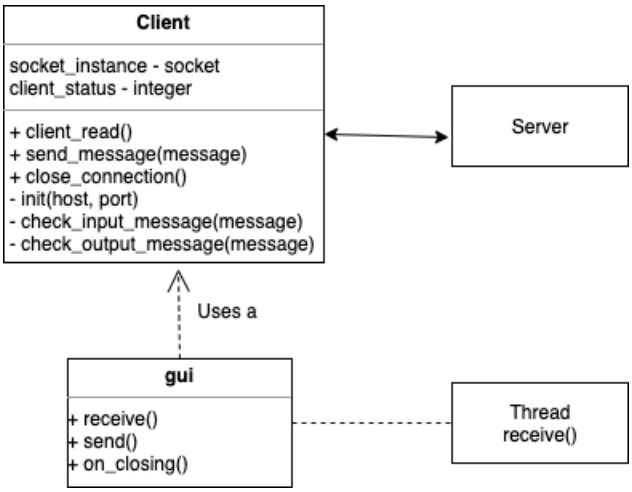

Figura 4: Client

2.4.1 Avvio

L'avvio del client avviene mediante l'esecuzione del file `gui.py`

- Viene caricata un'interfaccia grafica con il supporto del pacchetto `tkinter` messo a disposizione da Python.
- Viene predisposta una istanza della classe client, la quale si occupa di configurare una nuova socket TCP e di instaurare la connessione con il server.
- Viene avviato un thread di ricezione sulla socket che notifica e successivamente stampa a video sull'interfaccia grafica i messaggi ricevuti.

2.4.2 Ingresso in partita

Quando la connessione client - server viene instaurata il server richiede al client di inserire il nome di gioco, esso dovrà essere differente dai nomi già registrati nella stanza attuale. In caso di scelta di un nome già presente all'interno della stanza ne verrà richiesto un altro.
A questo punto se la stanza non ha raggiunto il numero esatto di partecipanti il nuovo giocatore viene inserito, altrimenti il server comunica che la partita è già stata avviata e lo scollega, invitandolo a riprovare più tardi.

2.4.3 Invio messaggi

L'utente può inviare messaggi scrivendoli nell'apposito TextBox dell'interfaccia grafica, successivamente premendo il pulsante Send viene inviato il testo mediante la socket gestita dal client.

2.4.4 Stati del client

Il client dispone di 2 stati

- `CLIENT_PAUSED_STATUS`, i messaggi che il client invierà non verranno recapitati al server.
  È lo stato che caratterizza il client quando è in attesa di incominciare la partita e quando, a partita conclusa, attende e successivamente visualizza la tabella dei risultati.
  Quando il server invia il comando di pausa al client, il client tradurrà questo messaggio e lo stamperà a video sull'interfaccia grafica per spiegare cosa sta succedendo.

- `CLIENT_RUNNING_STATUS`, il client può comunicare con il server.
  Quando il server invia il comando di run al client, il client tradurrà questo messaggio e lo stamperà a video sull'interfaccia grafica per spiegare cosa sta succedendo.

2.4.5 Esclusione dalla partita

Il giocatore può abbandonare in qualsiasi momento la partita, sia chiudendo l'interfaccia grafica, sia digitando il comando speciale `!quit`.
L'esclusione della partita può verificarsi anche quando il client sceglie l'opzione trabocchetto tra le 3 proposte dal server. In questo caso il server comunicherà la scelta dell'opzione trabocchetto e dopo un breve lasso di tempo il giocatore verrà espulso dalla partita e cancellato dalla tabella dei risultati.

2.5 Comandi

Per facilitare la comunicazione client - server sono stati introdotti 3 comandi speciali

- `QUIT_MESSAGE`: "!quit" è un comando bidirezionale, sia il server che il client possono in qualunque momento cessare la connessione, e quindi chiudere la sessione in corso.
  Sortisce questo effetto sia se il server lo invia al client, sia se il client lo invia al server.

- `CLIENT_PAUSED_MESSAGE`: "!pause" è un comando monodirezionale, setta a `CLIENT_PAUSED_STATUS` lo status del client, sortisce questo effetto solo se il server invia questo messaggio al client.

- `CLIENT_RUNNING_MESSAGE`: "!run" è un comando monodirezionale, setta a `CLIENT_RUNNING_STATUS` lo status del client, sortisce questo effetto solo se il server invia questo messaggio al client.

2.6 TCP

La connessione client - server avviene mediante protocollo di trasporto TCP.
Dopo una fase di handshaking tra le due entità, questa tipologia di connessione permette una comunicazione full-duplex, ovvero scambi anche simultanei di messaggi tra le due parti.
Essa garantisce sia un trasferimento affidabile di dati, non è necessario quindi ricontrollare l'integrità dei messaggi scambiati, sia il servizio di controllo della congestione, strozzando i processi d'invio qualora il traffico di rete sia eccessivo.
Al termine della sessione di gioco però le connessioni client - server vanno esplicitamente chiuse.

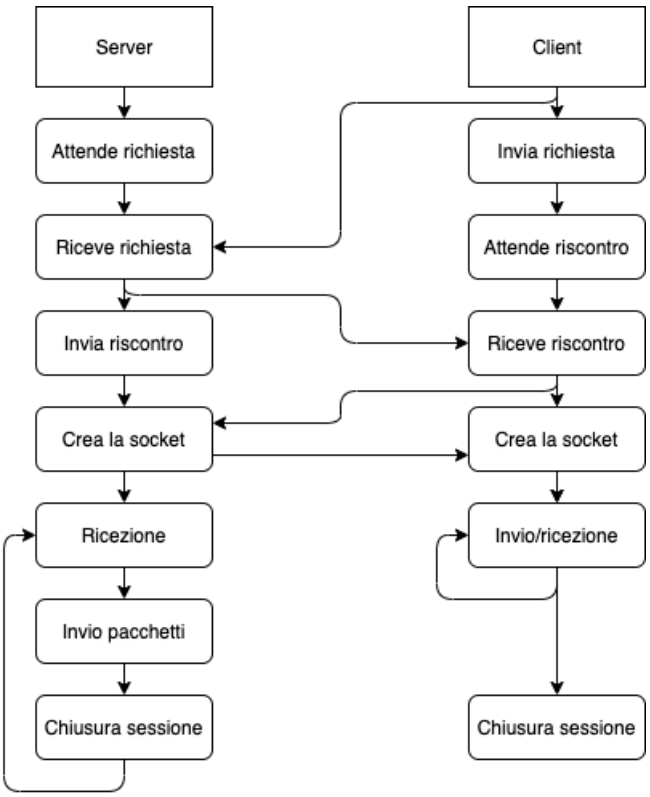

Figura 5: Schema connessione TCP

3 Sviluppo

3.1 Strumentazione adoperata

- GIT: per poter lavorare in gruppo abbiamo optato per l'utilizzo di un repository git in modo da poter condividere modifiche evitando eventuali problemi di sovrascrittura di file.
- IDE: Come IDE di sviluppo abbiamo adoperato PyCharm di IntelliJ il quale ci ha permesso di effettuare una fase di debugging in maniera più agile.

3.2 Metodo di sviluppo

Prima di procedere con la stesura del codice dell'applicativo abbiamo progettato degli schemi uml di supporto in modo da suddividere correttamente le varie entità presenti e il carico di lavoro.

3.3 Possibili miglioramenti

Pur avendo raggiunto la quasi totalità degli obiettivi prefissati, possiamo individuare un paio di punti per cui l'applicazione può essere ulteriormente migliorata:

- Miglior gestione delle eccezioni di varia natura.
- Miglioramento della funzionalità di shutdown che deve poter chiudere anche i client che non hanno effettuato alcuna registrazione nel server (ovvero client che non hanno inviato il nome).
- Miglioramento delle GUI.

4 Configurazione

È possibile modificare alcune configurazioni dell'applicazione attraverso il file `utils/app_variables.py`.

- `HOST`: L'indirizzo del server host con cui innestare i collegamenti.
- `PORT`: Il numero di porta per il collegamento.
- `BUFFER_SIZE`: La dimensione del buffer per la ricezione dei messaggi durante la connessione.
- `MATCH_TIMER`: Durata della partita dal momento in cui il server ne comunica l'inizio.
- `PARTICIPANTS`: L'esatto numero dei giocatori che devono entrare nella sessione per far si che la partita venga avviata.

È possibile inoltre modificare le domande che il server pone al client e anche le risposte che il server si aspetta per determinare il punteggio positivo attraverso il file `utils/questions.py`, oltre che i ruoli che il server distribuisce ai client mediante il file `utils/roles.py`.

Si sconsiglia l'eventuale modifica delle altre variabili presenti nel file.

5 Guida all'avvio

Per poter avviare correttamente l'applicazione occorre

- Posizionarsi con la console nella directory root del progetto.
- Avviare il server tramite il comando `python -m src.server.server`.
- All'apertura della GUI premere il pulsante di avvio ("Start").
- Aprire due client attraverso il comando `python -m src.client.gui` lanciato sempre all'interno della cartella root del progetto in due differenti terminali.

Nota : Lanciare e attivare sempre il server prima di lanciare i clients, assicurarsi che non vi siano applicazioni che utilizzino una connessione 53000 (o quella eventualemnte configurata nel file di configurazione `utils/app_variables.py`).