Chat-Game
Relazione di progetto

Buizo Manuel
Sabatino Panella

22 agosto 2021

Indice

Capitolo 1

Analisi

Questo progetto è stato concretizzato tramite l'utilizzo del linguaggio di programmazione Python e i socket TCP-IP grazie ai quali è possibile creare connessioni client-server.
Al gioco Chat-Game possono partecipare da 1 a più giocatori, i quali dovranno rispondere alle domande per riuscire a vincere.
Ogni giocatore può accedere alla stanza in modo asincrono prima che l'ultimo giocatore abbia finito, per accedere al gioco (stanza) si dovrà loggare con un proprio nickname.
Una volta entrati, gli sarà assegnato un ruolo e partirà un timer personale.
In quel lasso di tempo il giocatore dovrà rispondere a più domande possibili.
Ma per rispondere alle domande dovrà scegliere prima tra le 3 possibili scelte `(a, b, c)`, una delle quali è una trappola e fa perdere all'istante il giocatore, mentre le altre due celano delle domande. Dando la risposta giusta a una di queste si riceverà un punto, in caso contrario invece verrà tolto.

1.1 Analisi e modello del dominio

Il gioco è composto da molti `Client` che saranno i `Giocatori` e un `MasterServer` cioè `Server` che gestirà questi ultimi.
Avremo quindi nello specifico i seguenti oggetti di gioco:

- **Client**, entità giocatore, il quale dovrà rispondere alle domande;
- **Server**, entità MasterServer che gestirà le domande, il punteggio e le connessioni.

Capitolo 2

Design

2.1 Architettura

2.1.1 MVC

L'architettura Chat-Game è stata impostata seguendo il pattern architetturale MVC, Model, View e Controller, sia per il client che per il server.

Model

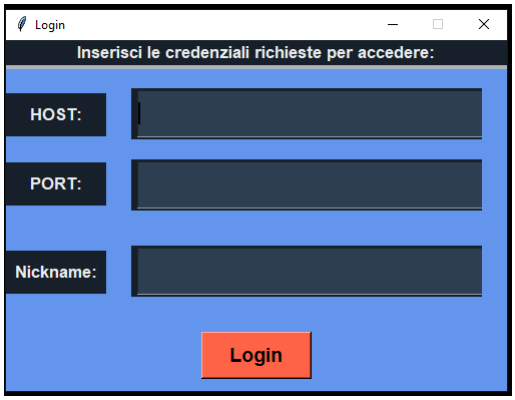
Il model è in sè l'oggetto dell'applicazione. Quest'ultimo è stato implementato in modo da fornire al controller metodi corretti per l'accesso ai dati oltre che a gestire come questi vengano influenzati da parte dell'utente finale.

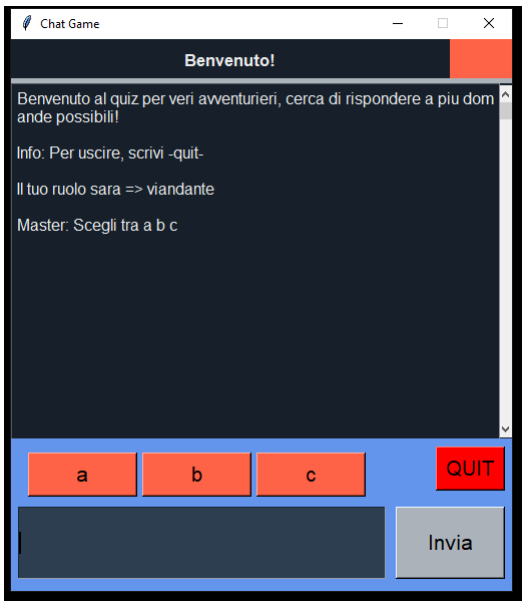
- **Client**: l'oggetto che fornisce i dati per la socket ed il timer del giocatore.
- **Server**, l'oggetto che fornisce i dati per la socket ma anche i dati del gioco, tra cui: punteggio, domande e le connessioni dei client.

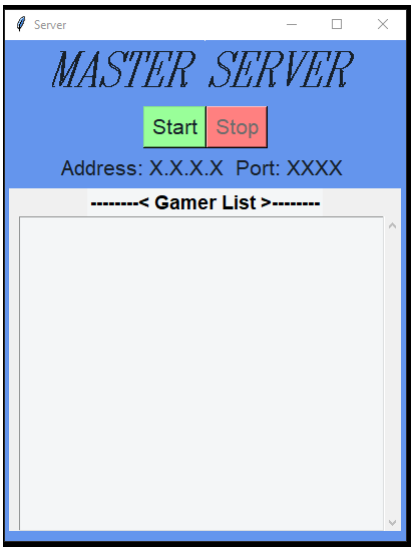
View

La view è composta dalle GUI del gioco, le quali rimangono indipendenti dal model e dal controller. E' la componente grafica del gioco.

- **Client**: fornisce due finestre per interagire, una di login per instaurare la connessione, ed una di chat per interagire col gioco (Server).
- **Server**, composto da un'unica finestra dove si potrà aprire e chiudere la socket del server; più in basso si potra visualizzare il punteggio di ogni giocatore, oltre al relativo ruolo e nickname.


Login
Inserisci le credenziali richieste per accedere:
HOST:
PORT:
Nickname:
Login

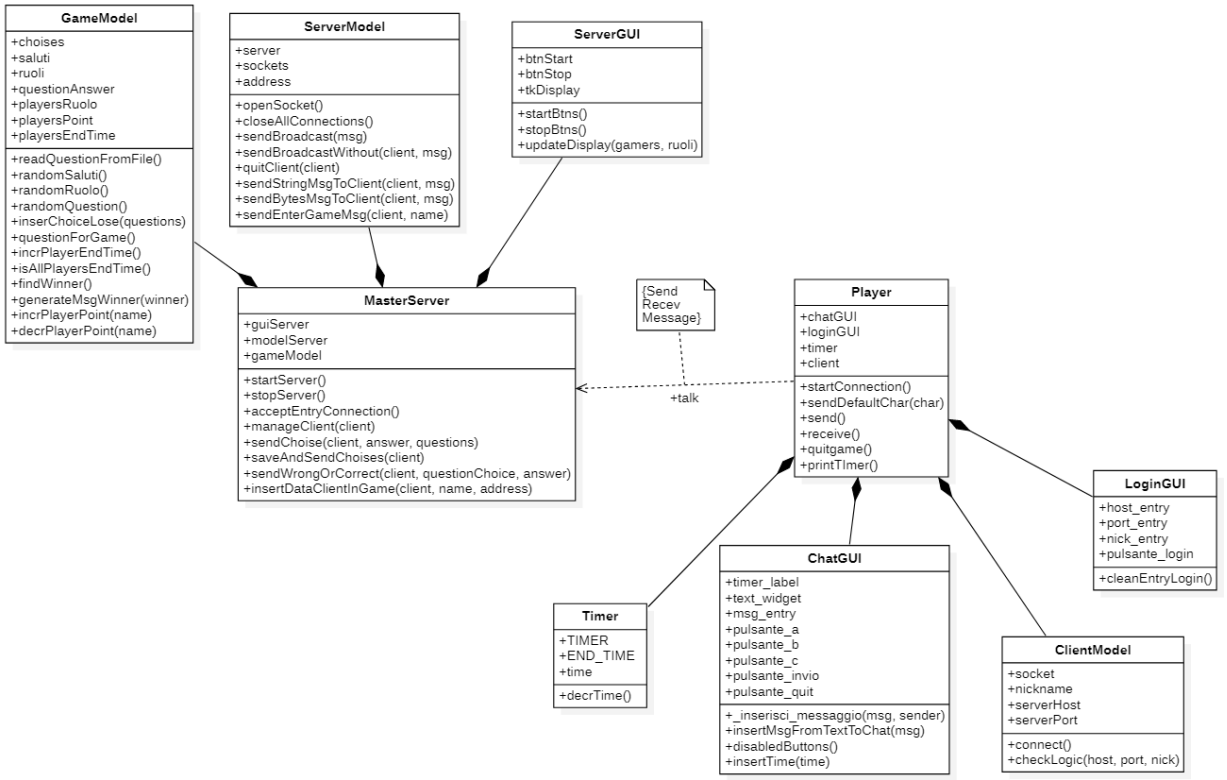

Chat Game
Benvenuto!
Benvenuto al quiz per veri avventurieri, cerca di rispondere a piu dom ande possibili!
Info: Per uscire, scrivi -quit-
Il tuo ruolo sara => viandante
Master: Scegli tra a b c
a
b
c
QUIT
Invia


Server
MASTER SERVER
Start
Stop
Address: X.X.X.X Port: XXXX
--------< Gamer List >--------

Controller

Il controller è quel componente dell'architettura che ha il compito di controllare e gestire l'ordine e la consequenzialità degli eventi.

- **Client**: In questo caso sara il modulo Player.py a gestire i dati e la parte grafica.
- **Server**: Il MasterServer.py gestirà il gioco e la connessione con il Player.



2.2 Design dettagliato

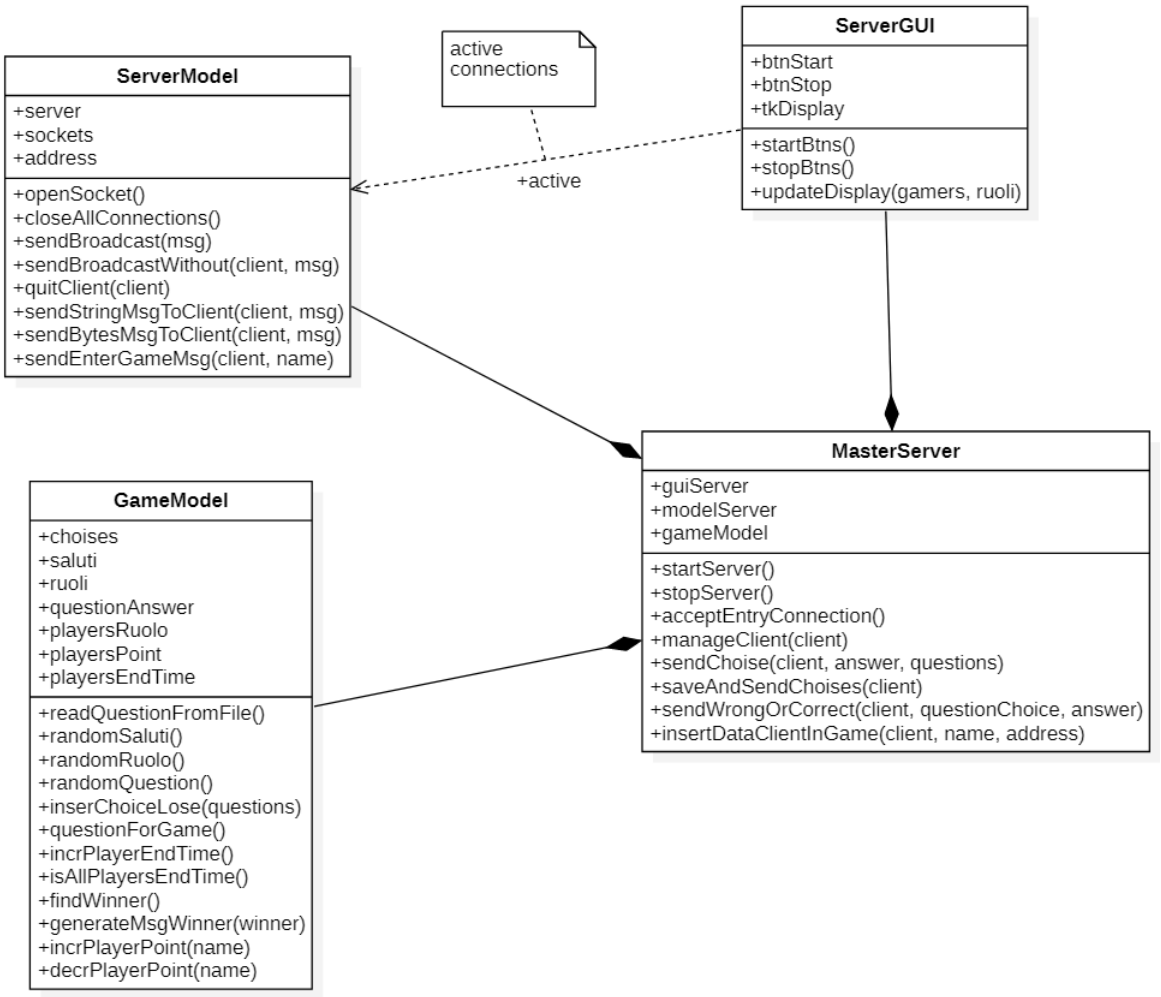
2.2.1 Server-MasterServer

MasterServer

Il server tiene traccia di ogni connessione che instaura con il client. Ha inoltre la capacità di mandare loro i messaggi e riceverli a sua volta.

Funzionalità:

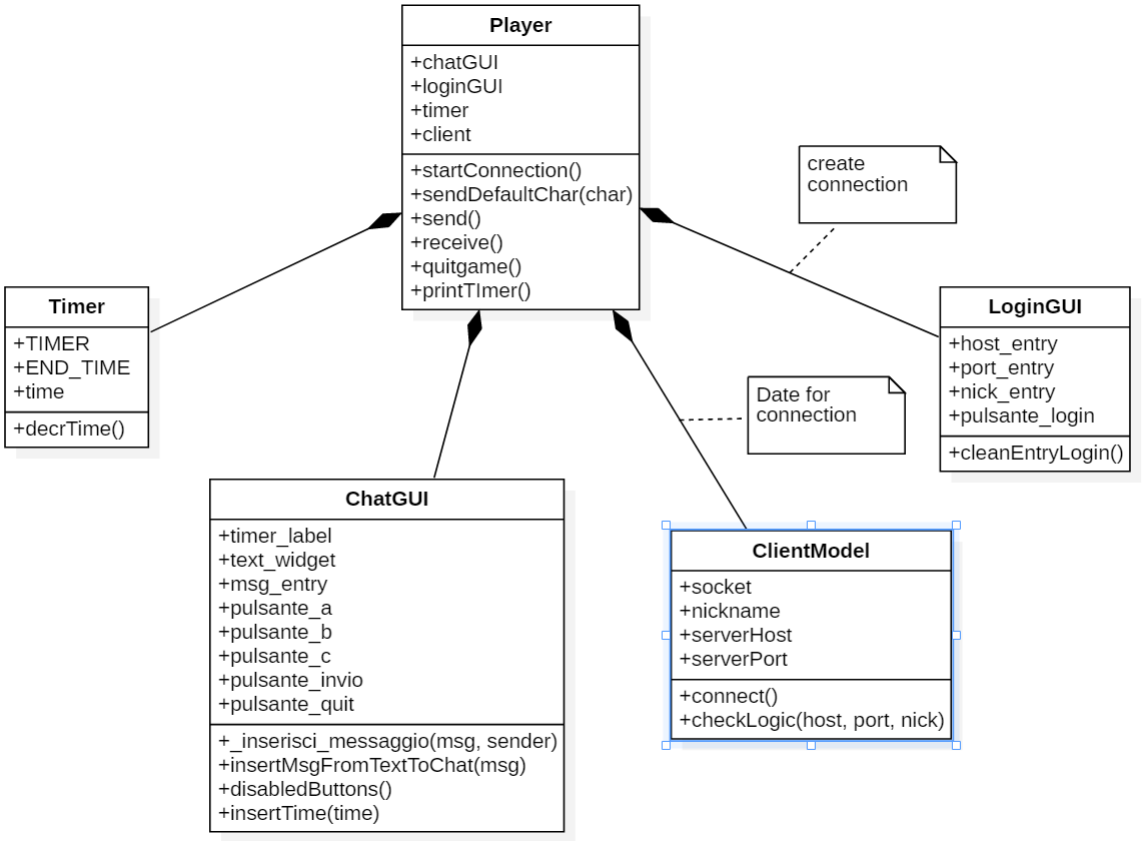
- **Connessioni**: Il server si serve del modulo socket per generare connessioni. Tramite quest'ultimo rimane in ascolto dei giocatori che si vogliono unire al gioco.
- **Creazione delle domande**: Per la creazione delle domande abbiamo usato un file di tipo: ".txt", scritto in modo schematico per i tre tipi di messaggi: saluto, ruolo e domanda. Di seguito degli esempi:
  saluto -> ciao ;
  ruolo -> elfo ;
  domanda -> quanto fa 2 + 2 -> 4.
- **Scelta della domanda**: Vengono prese randomicamente tre domande dal file ".txt" per creare una lista. Successivamente viene inserita in una delle tre possibli opzioni la trappola (sempre in modo casuale). Una volta che il client avrà scelto tra: "a, b, c", si invierà la domanda associata alla sua scelta (o la trappola, se sarà sfortunato).
- **Scelta della trappola**: In caso di trappola, il server invierà un messaggio di sconfitta, chiudendo la connessione del giocatore associato.
- **Aggionamento punteggio**: Ogni volta che un giocatore risponderà alle domande, verrà quindi aggionato il display del server con i relativi punteggi.
- **Fine partita**: Il server, ogni qualvolta gli arrivi il messaggio `keytime`, incrementerà un contatore e confronterà il numero di connessioni instaurate; se risultano uguali, allora calcolerà il punteggio massimo e manderà ad ogni giocatore il messaggio del giocatore vincitore. In caso contrario aspetterà gli altri giocatori.


ServerModel
+server
+sockets
+address
+openSocket()
+closeAllConnections()
+sendBroadcast(msg)
+sendBroadcastWithout(client, msg)
+quitClient(client)
+sendStringMsgToClient(client, msg)
+sendBytesMsgToClient(client, msg)
+sendEnterGameMsg(client, name)
active connections
+active
ServerGUI
+btnStart
+btnStop
+tkDisplay
+startBtns()
+stopBtns()
+updateDisplay(gamers, ruoli)
MasterServer
+guiServer
+modelServer
+gameModel
+startServer()
+stopServer()
+acceptEntryConnection()
+manageClient(client)
+sendChoise(client, answer, questions)
+saveAndSendChoises(client)
+sendWrongOrCorrect(client, questionChoice, answer)
+insertDataClientInGame(client, name, address)
GameModel
+choises
+saluti
+ruoli
+questionAnswer
+playersRuolo
+playersPoint
+playersEndTime
+readQuestionFromFile()
+randomSaluti()
+randomRuolo()
+randomQuestion()
+inserChoiceLose(questions)
+questionForGame()
+incrPlayerEndTime()
+isAllPlayersEndTime()
+findWinner()
+generateMsgWinner(winner)
+incrPlayerPoint(name)
+decrPlayerPoint(name)

2.2.2 Client-Player

Player

- **Login**: Tramite l'interfaccia grafica, il giocatore inserirà l'host e la porta del server per accedere a quest'utimo, assieme ad un nickname che il server utilizzerà per memorizzare il punteggio.
- **Bottoni di scelta**: Il client interagirà con il server attraverso la finestra di chat, rispondendo alle domande, con la possibilità di avvalersi dei bottoni per la scelta rapida tra le possibili opzioni, che potranno sostituire l'inserimento via chat dell'opzione desiderata (a, b, c).
- **Timer**: Il timer è gestito da un thread che decrementa il tempo da un numero di secondi massimo specificato, nel corso di una partita. Il timer si attiva una volta entrati nella stanza; allo scadere di quest'ultimo verranno bloccati i bottoni e il client invierà il messaggio `keytimer` al server, il quale incrementerà il conteggio dei giocatori che hanno finito il tempo.



2.2.3 Thread

Il progetto è stato creato usando più di un thread, per avere le seguenti funzionalità: connessioni, timer ed attesa delle nuove connessioni. In questi thread, il server è il thread principale che gestisce le connessioni.

Client-Player

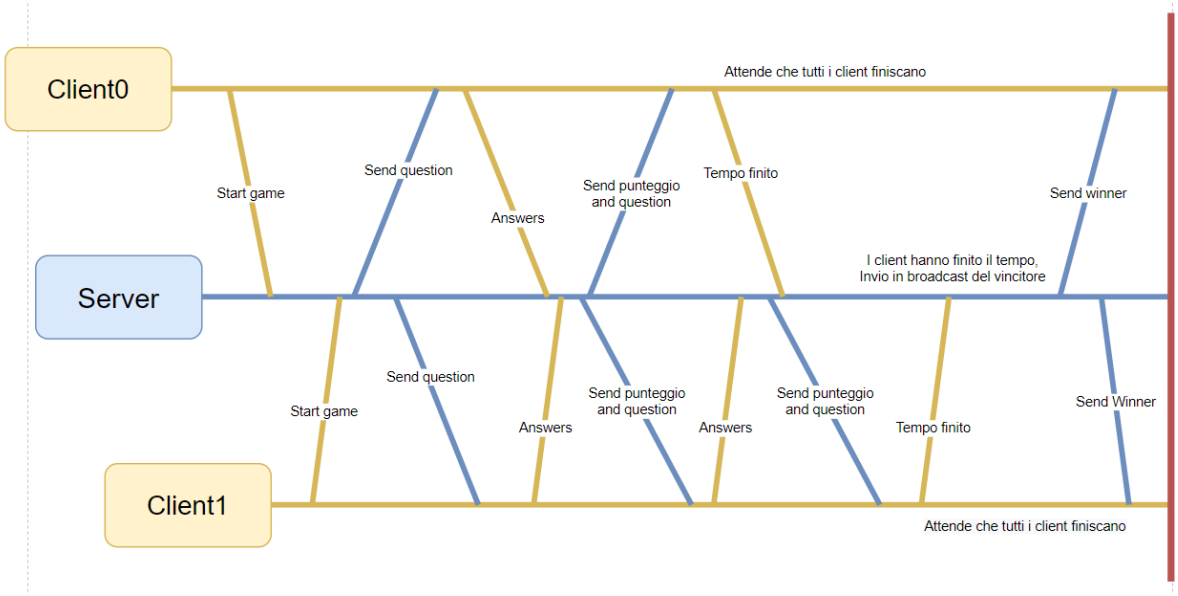
Per ogni client creato vengono automaticamente creati 4 thread:

- **Ricezione dei messaggi**: Questo thread si occupa di rimanere in ascolto dei messaggi in arrivo dal server, in modo da non bloccare il thread della finestra chat.
- **Timer**: In questo specifico thread, il suo utilizzo permette di trasmettere l'informazione del tempo scaduto del proprio client.
- **Finestra login**: Thread che gestisce le funzionalità del login, quindi la trasmissione di credenziali da client a server.
- **Finestra chat**: Thread che gestisce le funzionalità della finestra di chat, quindi la trasmissione di scelte, la trasmissione di domande e l'invio di risposte inserite dal client, inviate nella chat box.

Server

All'attivazione del server viene inizializzato un thread per instaurare le connessioni in entrata. Per ogni connessione vi è un thread che gestisce la ricezione dei messaggi, in modo da non bloccare il thread della finestra del server.

- **Finestra**: Thread che gestisce le funzionalità della finestra del MasterServer, quindi la stampa di giocatori connessi, i loro ruoli e relativi punteggio.
- **Ricezione delle connessioni**: Thread con il compito di instaurare le connessioni in entrata.
- **Gestione delle connessioni**: In questo specifico thread vengono gestite le funzionalità del gioco stesso, cui la ricezione dei messaggi da parte del client, e l'invio delle corrispettive risposte o domande, il tutto visualizzato tramite la chat del client.


Client0
Server
Client1
Attende che tutti i client finiscano
Start game
Send question
Answers
Send punteggio
and question
Tempo finito
Send winner
I client hanno finito il tempo,
Invio in broadcast del vincitore
Start game
Send question
Answers
Send punteggio
and question
Answers
Send punteggio
and question
Tempo finito
Send Winner
Attende che tutti i client finiscano