Enrica Contrini MATR. 941723
Livia Del Gaudio MATR. 916481
Ines Fraccalvieri MATR. 916251

Prof. G.Pau e A. Piroddi
4 giugno 2021

Relazione "Guess the Answer"

Sviluppo di un'applicazione client-server

La consegna della traccia da noi scelta prevedeva lo sviluppo di un'applicazione client-server che emulasse un Multiplayer Game testuale, di cui sono state fornite le varie specifiche. Il nostro gruppo, basandosi sull'architettura del rock-paper-scissors game fornita a lezione, ha sviluppato un'applicazione che sfrutta il concetto di socket e il protocollo TCP per lo scambio di informazioni tra i client, in cui il server si pone come intermediario.

Per realizzare ciò, abbiamo innanzitutto definito la coppia indirizzo-porta (previa importazione del modulo python "socket") di client e server, per poi stabilire la famiglia dei socket(AF_INET) e il tipo (nel nostro caso TCP):

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM), nel server (riga 61)

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM), nel client (riga 278, all'interno della funzione dedicata alla connessione al server: connect_to_server). Successivamente abbiamo associato l'oggetto creato (sia nel server che nel client) all'indirizzo e alla porta desiderati, precedentemente definiti, mediante:

server.bind((HOST_ADDR, HOST_PORT)) nel server (riga 65) e, analogamente,

client.connect((HOST_ADDR, HOST_PORT)) nel client (riga 279).

Per quanto riguarda il server, esso viene poi messo in ascolto per possibili connessioni da parte dei client e si dà avvio a un thread che svolgerà la funzione accept_clients (righe 66-67):

server.listen(5)

threading._start_new_thread(accept_clients, (server, " "))

In accept_clients il server accetta le connessioni e appende i clients alla lista dedicata, per poi far partire un thread che si occuperà della ricezione di messaggi da un dato client e dell'invio

agli altri connessi, secondo quanto definito nella funzione send_receive_client_msg: il server dà inizialmente il benvenuto a entrambi i giocatori, poi invia a ciascuno il nome dell'avversario e infine si mette in attesa. L'invio dei messaggi avviene mediante:

client[i].send("string".encode( )), dove i rappresenta la posizione del client nella lista. Quando il server riceve un messaggio da parte di un client, lo assegna a una variabile (data) per poterlo poi inoltrare all'altro client (secondo il meccanismo di send descritto prima):

data = client_connection.recv(4096)

Nel nostro caso, per come abbiamo costruito la logica del gioco, il server riceverà unicamente il punteggio del client al termine del gioco che sarà poi inviato all'avversario, così da permettere a ciascuno client di determinare se il proprio giocatore ha vinto, perso o pareggiato. Qualora il server ricevesse un messaggio contenente la stringa "{quit}", uscirebbe dal loop di ascolto e provvederebbe a chiudere la connessione col client e ad eliminarlo dalla lista (caso che si verifica se l'utente sceglie l'opzione trabocchetto):

```
if data == bytes("{quit}", "utf8"): break                    (riga 120)
…
del clients[idx]                                             (riga 155)
client_connection.close( ) .
```

Dal punto di vista del client invece, una volta stabilita la connessione col server, fa partire un thread che eseguirà un loop infinito nella funzione receive_message_from_server, dove si gestiscono con una sequenza di if-elif-else i diversi messaggi in arrivo, distinguendo mediante il prefisso ("welcome" per il messaggio di benvenuto e "opponent_name" per il nome dell'avversario):

```
from_server = sck.recv(4096),   dove sck corrisponder alla socket del client, (riga 309)
…
if from_server.startswith("welcome".encode( )):
  …              (encode rappresenta la codifica necessaria ai messaggi per "viaggiare"
                                                                    in TCP)
elif from_server.startswith("opponent_name$".encode( )):
  …              (da qui si avvia il gioco facendo partire un thread per il countdown)
else:
  opponent_score = from_server.decode("utf8")
```

In quest'ultimo caso è infatti proprio necessario estrarre il messaggio, decodificandolo, poiché contiene il punteggio dell'avversario, necessario per decretare il vincitore. Ciascun client invia il proprio score quando il thread che esegue un loop di tempo prestabilito esce dal ciclo (riga 270):

client.send(bytes("%i" %your_score, "utf8")), dove il risultato numerico viene codificato in bytes per poter essere inviato al server.

La connessione al server viene chiusa , così come la finestra grafica principale, se l'utente seleziona il trabocchetto (nel qual caso viene impostato un flag "trap" a True), uscendo così dal loop infinito per la ricezione dei messaggi dal server (riga 304):

```
client.send(bytes("{quit}", "utf8"))
sck.close()
window_main.quit()
break .
```

Per quanto riguarda la logica del gioco, all'avvio del client viene stabilito il suo ruolo (attraverso l'estrazione random di un numero che costituisce la chiave del ruolo, definiti nel dizionario defruolo), sulla base del quale verranno selezionate le domande da porgli (le domande, con le relative risposte, sono definite in dizionari appositi presenti in un modulo esterno chiamato "Modulo_Domande"). Infatti, quando l'utente sceglie una delle opzioni (1, 2 o 3), viene determinata la domanda da mostrargli a partire da un numero random che stabilisce la posizione della prima domanda (posFirstQuestion), estratta casualmente e inserita nella variabile answerA (il numero estratto viene utilizzato sia per risalire alla domanda, sia alla risposta). La seconda domanda è semplicemente definita come answerB = answerA + 1, e l'opzione rimasta prevederà invece il trabocchetto, per cui l'utente ha automaticamente perso e sarà disconnesso dal server. Si prevedono una serie di casi diversi mostrati con una sequenza di if-elif-else per stabilire la domanda corretta da mostrare combinando il ruolo, la prima domanda estratta e la sua posizione. Una volta mostrata la domanda, l'utente dovrà inserire la propria risposta e verificarne la correttezza premendo sul pulsante "Check", che "cattura" la stringa inserita e la confronta con la risposta corretta nella funzione dedicata "CheckAnswer". Il responso viene mostrato automaticamente e viene di conseguenza aggiornato il punteggio. Allo scadere del timer il client invia il proprio punteggio al server, riceve quello dell'avversario (secondo il meccanismo illustrato in precedenza) e stabilisce se l'utente ha vinto o meno, mostrandolo sull'applicazione.

Infine, l'aspetto grafico viene gestito mediante il modulo di python "Tkinter", opportunamente importato sia nel client che nel server.

Un file "README" contenente uno user manual è fornito appresso alla relazione, per rendere più chiaro a chi non avesse esperienza in materia come avviare il codice e il funzionamento del game realizzato.