



Studente

Yuri Bacciarini

5654547

Progetto RC Giugno/Luglio 2017

Relazione

Il codice è composto da un server.py che rappresenta il “gestore” dei client (giocatori) che intendono registrarsi (giocare o consultare la classifica), un client.py che come suggerisce il nome rappresenta il client (giocatore) e una cartella “test” contenente alcuni test per verificare la correttezza del server. Sia server.py che client.py condividono gli script config.py e libs.py.

In questo progetto ho deciso di utilizzare il protocollo di trasporto TCP per realizzare la comunicazione con il server. I messaggi scambiati tra client e server sono una conversione a base 64 di un json. Ogni messaggio dal client al server contiene la chiave “cmd” che rappresenta la tipologia di richiesta e la chiave “data” che contiene le informazioni relative al comando richiesto.

Protocollo

Quando un client vuole tentare una connessione al server invia:

```
{"cmd": "login", "data": {"usr": "user1", "psw": "pass"}}
```

Il Server può rispondere con 3 contenuti diversi:

- In caso l'utente non esista: `{"response": "notknown"}`
- In caso l'utente esista ma che sia già loggato `{"response": "alreadylogged"}`
- In caso l'utente esista e che non sia ancora loggato `{"response": "ok"}`

Quando il client ha ricevuto la risposta di login andato a buon fine può richiedere di mettersi in attesa per giocare tramite il comando:

```
{'cmd': 'play', 'data': {'usr': 'user1'}}
```

Il server confermerà adesso la richiesta di giocare da parte dell'user1 rispondendo con:

```
{"response": "waiting"}
```

Da adesso in poi il client attenderà fino a quando sul server non ci saranno almeno 2 client connessi ossia la condizione per giocare. Appena si registrerà un nuovo client sul server e richiederà di giocare, ad esempio l'user2, entrambi riceveranno dal server il seguente messaggio che racchiude le informazioni della partita iniziata. Il messaggio dei due client differisce semplicemente dal valore della chiave data.first che indica al client se è o meno lui che dovrà iniziare.

```
{'cmd': 'game', 'data': {'last_play': None, 'player': 'X', 'play': {'user2': {'player': 'O', 'done': []}, 'user1': {'player': 'X', 'done': []}}, 'status': 'start', 'first': 1}}
```

Il client con data.first settato ad 1 potrà quindi decidere quale mossa fare tra quelle rimanenti ed accodarla alle sue giocate nell'array data.play.user[?].done. Una volta fatto questo, il client rimuove la chiave data.first, setta il data.status a 'inplay', inserisce correttamente le informazioni dell'ultima giocata nella chiave data.last_play e inoltra il messaggio al server.

Il server adesso verifica che nessuno dei 2 client abbia vinto, in caso negativo e nel caso in cui la partita non sia finita in pareggio inoltra esattamente lo stesso messaggio al giocatore che non è stato l'ultimo ad aver giocato.

Il gioco termina quando c'è una vittoria o un pareggio. In caso di vittoria il server invia al client il messaggio visto precedentemente che racchiude le informazioni delle giocate fatte fino a quel momento con la chiave aggiuntiva data.winner settata all'id del player. In caso di pareggio invece la chiave data.winner è settata a 'none'.

Un ulteriore comando che un client può inviare al server è il comando "table". Questo comando si aspetta di ricevere una risposta con la classifica dei giocatori comprendente l'id del giocatore, il numero di partite effettuate, il numero delle vittorie e la data dell'ultima giocata.

Un'esempio di risposta:

```
{"data": [["user1", 1, 1, "2017-07-15 16:15:41.897996"], ["user5", 0, 0, 0], ["user4", 0, 0, 0], ["user3", 0, 0, 0], ["user2", 1, 0, "2017-07-15 16:15:41.897996"]], "response": "table"}
```

server.py

Il server realizzato sta in attesa di connessioni dai client e quando ne arriva una, essa viene gestita in un thread. Il server contiene principalmente 3 strutture degne di nota:

- il database in sqlite chiamato "data.db" che contiene username, password e statistiche delle giocate.
- il dizionario playersLogged che contiene tutti gli utenti connessi e il riferimento alla socket associata.
- l'array playersReady contenente tutti gli utenti in attesa di giocare.

Quando avviene una nuova connessione e una richiesta di giocare da parte di un client, il server controlla se nell'array playersReady ci sono almeno 2 giocatori. In caso negativo continua la sua esecuzione altrimenti ne preleva 2 e inizializza la partita notificando entrambi. Durante la partita il server fa da "controllore" per verificare se la partita è in corso,

se è terminata in pareggio oppure qualcuno ha vinto. Per verificare se qualcuno ha vinto controlla che l'array delle giocate di un utente contenga al suo interno un sottoinsieme di un array delle combinazioni vincenti. In caso negativo, se il numero delle giocate totali è 9 la partita termina in pareggio altrimenti è ancora in corso.

client.py

Il client interagisce con il server e simula un giocatore (vedere screen in fondo).

Test

Tutti i test includono il config.py che racchiude le informazioni sul server. Inoltre tutti i test nella funzione setUp racchiudono la creazione della socket e la connessione al server mentre nella funzione tearDown la disconnessione dal server tramite il corretto comando al server.

test_step1.py

Il 1° test contiene 2 parti:

1. Viene effettuata una richiesta di connessione al server con username e password corretti, il server accetta la connessione comunicandolo al client. Si tenta una nuova connessione con le stesse credenziali precedenti, il server rifiuta la connessione perchè l'utente è già attivo.
2. Viene effettuata una richiesta di connessione al server con username e password corretti, il server accetta la connessione comunicandolo al client. Il client richiede al server di giocare, il server comunica al client che è in attesa del suo avversario.

test_step2.py

Il 2° test simula 4 partite tra user1 e user2 nelle quali, con mosse diverse, vince l'user2.

1. Nella prima partita la sequenza delle giocate è:
 - user2 posiziona la X nella cella 1
 - user1 posiziona la O nella cella 5
 - user2 posiziona la X nella cella 9
 - user1 posiziona la O nella cella 7
 - user2 posiziona la X nella cella 3
 - user1 posiziona la O nella cella 4
 - user2 posiziona la X nella cella 2
2. Nella seconda partita la sequenza delle giocate è:
 - user2 posiziona la X nella cella 1
 - user1 posiziona la O nella cella 3
 - user2 posiziona la X nella cella 5
 - user1 posiziona la O nella cella 6
 - user2 posiziona la X nella cella 9
3. Nella terza partita la sequenza delle giocate è:

- user2 posiziona la X nella cella 3
- user1 posiziona la O nella cella 5
- user2 posiziona la X nella cella 9
- user1 posiziona la O nella cella 4
- user2 posiziona la X nella cella 6

4. Nella quarta partita la sequenza è equivalente alla terza semplicemente per avere l'user2 primo in classifica come vedremo nel test_step5.py

test_step3.py

Il 3° test simula 4 partite tra user1 e user2 nelle quali, con mosse diverse, vince l'user1.

1. Nella prima partita la sequenza delle giocate è:

- user2 posiziona la X nella cella 2
- user1 posiziona la O nella cella 1
- user2 posiziona la X nella cella 3
- user1 posiziona la O nella cella 4
- user2 posiziona la X nella cella 5
- user1 posiziona la O nella cella 7

2. Nella seconda partita la sequenza delle giocate è:

- user2 posiziona la X nella cella 9
- user1 posiziona la O nella cella 3
- user2 posiziona la X nella cella 6
- user1 posiziona la O nella cella 5
- user2 posiziona la X nella cella 8
- user1 posiziona la O nella cella 7

3. Nella terza partita la sequenza delle giocate è:

- user2 posiziona la X nella cella 1
- user1 posiziona la O nella cella 5
- user2 posiziona la X nella cella 2
- user1 posiziona la O nella cella 3
- user2 posiziona la X nella cella 4
- user1 posiziona la O nella cella 7

test_step4.py

Il 4° test simula 3 partite tra user1 e user2 nelle quali, con mosse diverse, non vince nessuno.

1. Nella prima partita la sequenza delle giocate è:

- user2 posiziona la X nella cella 1
- user1 posiziona la O nella cella 5
- user2 posiziona la X nella cella 7
- user1 posiziona la O nella cella 4

- user2 posiziona la X nella cella 6
- user1 posiziona la O nella cella 2
- user2 posiziona la X nella cella 8
- user1 posiziona la O nella cella 9
- user2 posiziona la X nella cella 3

2. Nella seconda partita la sequenza delle giocate è:

- user2 posiziona la X nella cella 1
- user1 posiziona la O nella cella 5
- user2 posiziona la X nella cella 2
- user1 posiziona la O nella cella 3
- user2 posiziona la X nella cella 7
- user1 posiziona la O nella cella 4
- user2 posiziona la X nella cella 6
- user1 posiziona la O nella cella 9
- user2 posiziona la X nella cella 8

3. Nella terza partita la sequenza delle giocate è:

- user2 posiziona la X nella cella 3
- user1 posiziona la O nella cella 5
- user2 posiziona la X nella cella 9
- user1 posiziona la O nella cella 6
- user2 posiziona la X nella cella 4
- user1 posiziona la O nella cella 7
- user2 posiziona la X nella cella 1
- user1 posiziona la O nella cella 2
- user2 posiziona la X nella cella 8

test_step5.py

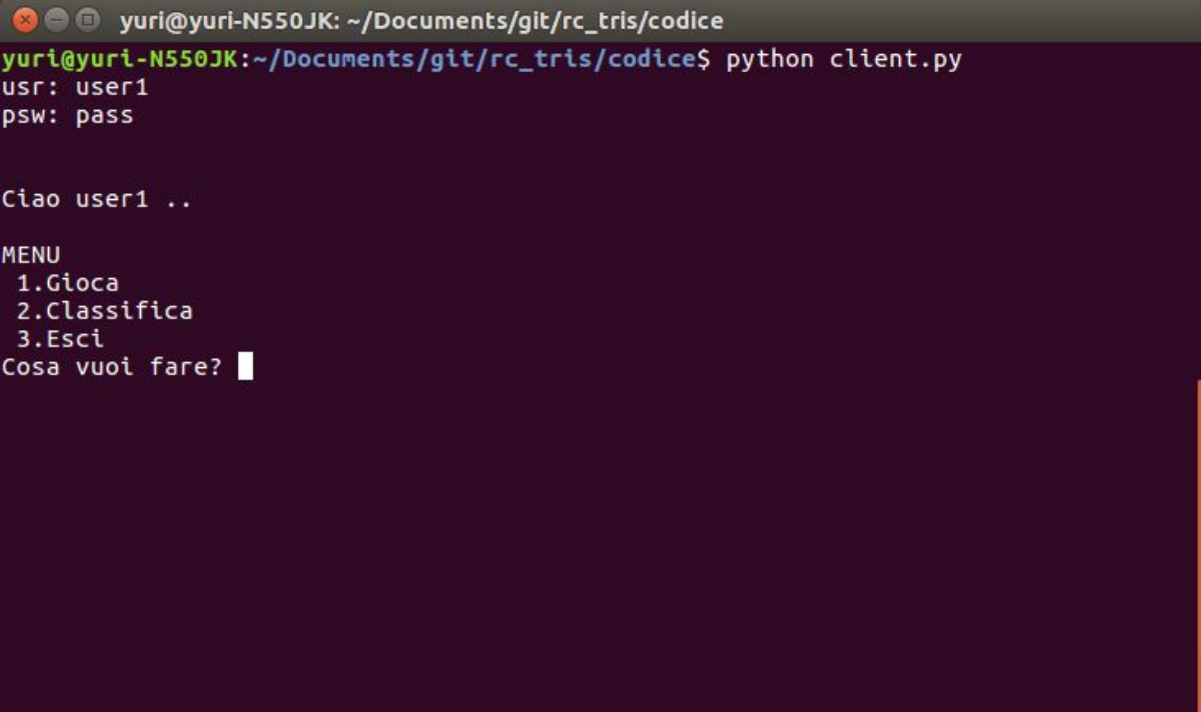
Il 5° test, dopo che il server ha immagazzinato le giocate e le vittorie dei test precedenti, richiede la classifica aspettandosi la seguente:

usr	games	won	lastGame
user2	10	4	2017-07-15 15:26:28.923037
user1	10	3	2017-07-15 15:26:28.923037
user5	0	0	0
user4	0	0	0
user3	0	0	0

Screen

I seguenti screen sono un esempio di esecuzione, il server non è negli screen ma è in esecuzione:

user1 loggato

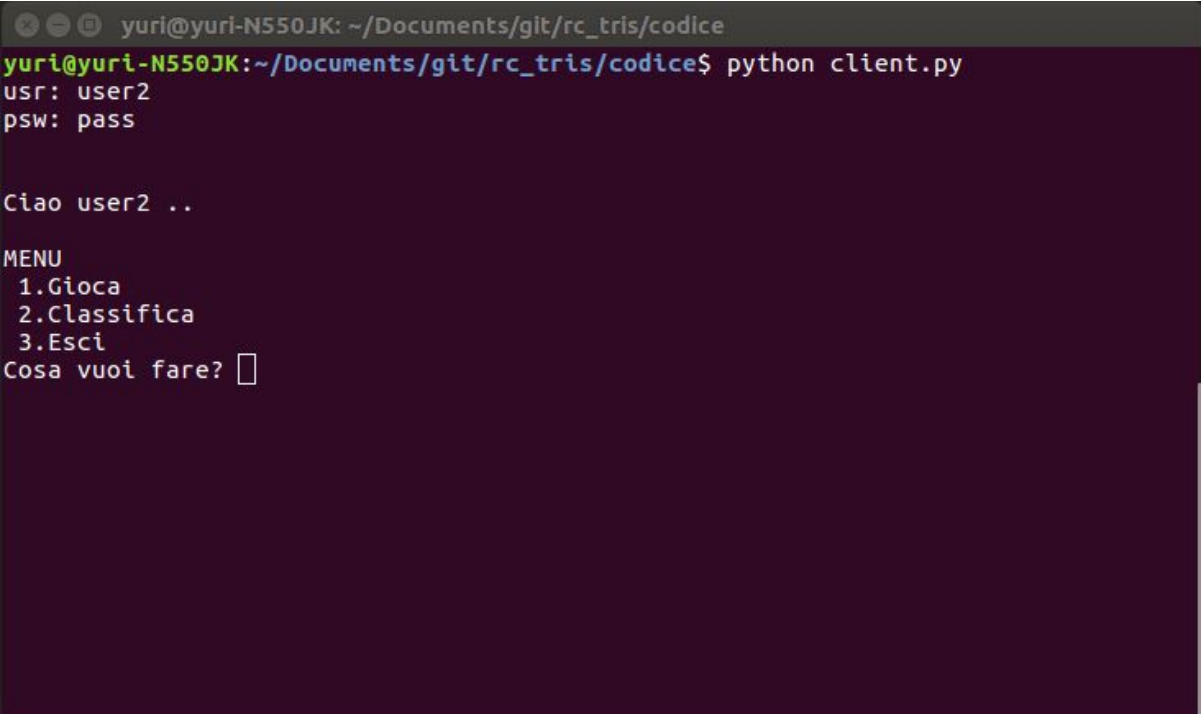
A terminal window with a dark purple background and a grey title bar. The title bar contains window control icons and the text 'yuri@yuri-N550JK: ~/Documents/git/rc_tris/codice'. The terminal shows the execution of 'python client.py', followed by login credentials 'usr: user1' and 'psw: pass'. The program then displays a greeting 'Ciao user1 ..' and a menu with three options: '1.Gioca', '2.Classifica', and '3.Esci'. It ends with the prompt 'Cosa vuoi fare?' followed by a cursor.

```
yuri@yuri-N550JK: ~/Documents/git/rc_tris/codice
yuri@yuri-N550JK:~/Documents/git/rc_tris/codice$ python client.py
usr: user1
psw: pass

Ciao user1 ..

MENU
 1.Gioca
 2.Classifica
 3.Esci
Cosa vuoi fare? █
```

user2 loggato

A terminal window with a dark purple background and a grey title bar. The title bar contains window control icons and the text 'yuri@yuri-N550JK: ~/Documents/git/rc_tris/codice'. The terminal shows the execution of 'python client.py', followed by login credentials 'usr: user2' and 'psw: pass'. The program then displays a greeting 'Ciao user2 ..' and a menu with three options: '1.Gioca', '2.Classifica', and '3.Esci'. It ends with the prompt 'Cosa vuoi fare?' followed by a cursor.

```
yuri@yuri-N550JK: ~/Documents/git/rc_tris/codice
yuri@yuri-N550JK:~/Documents/git/rc_tris/codice$ python client.py
usr: user2
psw: pass

Ciao user2 ..

MENU
 1.Gioca
 2.Classifica
 3.Esci
Cosa vuoi fare? █
```

user2 posiziona la X nella cella 1

```
yuri@yuri-N550JK: ~/Documents/git/rc_tris/codice
Ciao user2 ..

MENU
 1.Gioca
 2.Classifica
 3.Esci
Cosa vuoi fare? 1
.. aspettando un avversario ..

Trovato avversario
Sei il giocatore X
Il primo turno è il tuo!

 1 | 2 | 3
  |  | 
---|---
 4 | 5 | 6
  |  | 
---|---
 7 | 8 | 9
  |  | 
Mossa? 
```

user1 posiziona la O nella cella 2

```
yuri@yuri-N550JK: ~/Documents/git/rc_tris/codice
MENU
 1.Gioca
 2.Classifica
 3.Esci
Cosa vuoi fare? 1
.. aspettando un avversario ..

Trovato avversario
Sei il giocatore O
.. Aspettando la prima mossa dell'avversario ..

 X | 2 | 3
  |  | 
---|---
 4 | 5 | 6
  |  | 
---|---
 7 | 8 | 9
  |  | 
Mossa? 2

```

user2 posiziona la X nella cella 5

```
yuri@yuri-N550JK: ~/Documents/git/rc_tris/codice  


|       |   |   |
|-------|---|---|
| 1     | 2 | 3 |
| ----- |   |   |
| 4     | 5 | 6 |
| ----- |   |   |
| 7     | 8 | 9 |

  
Mossa? 1  


|       |   |   |
|-------|---|---|
| X     | 0 | 3 |
| ----- |   |   |
| 4     | 5 | 6 |
| ----- |   |   |
| 7     | 8 | 9 |

  
Mossa? 
```

user1 posiziona la O nella cella 6

```
yuri@yuri-N550JK: ~/Documents/git/rc_tris/codice  


|       |   |   |
|-------|---|---|
| X     | 2 | 3 |
| ----- |   |   |
| 4     | 5 | 6 |
| ----- |   |   |
| 7     | 8 | 9 |

  
Mossa? 2  


|       |   |   |
|-------|---|---|
| X     | 0 | 3 |
| ----- |   |   |
| 4     | X | 6 |
| ----- |   |   |
| 7     | 8 | 9 |

  
Mossa? 6  

```


user2 posiziona la X nella cella 9

```
yuri@yuri-N550JK: ~/Documents/git/rc_tris/codice

X | 0 | 3
---
4 | 5 | 6
---
7 | 8 | 9
Mossa? 5
X | 0 | 3
---
4 | X | 0
---
7 | 8 | 9
Mossa? 
```

user2 vincitore

```
yuri@yuri-N550JK: ~/Documents/git/rc_tris/codice

4 | X | 0
---
7 | 8 | 9
Mossa? 9
X | 0 | 3
---
4 | X | 0
---
7 | 8 | X
Complimenti! Sei il vincitore!
MENU
1.Gioca
2.Classifica
3.Esci
Cosa vuoi fare? 
```

user1 perdente

```
yuri@yuri-N550JK: ~/Documents/glt/rc_tris/codice
4 | X | 6
|   |   |
-----
7 | 8 | 9
|   |   |
Mossa? 6
X | 0 | 3
|   |   |
-----
4 | X | 0
|   |   |
-----
7 | 8 | X
|   |   |
Mi dispiace, non hai vinto!
MENU
1.Gioca
2.Classifica
3.Esci
Cosa vuoi fare? 
```

user2 richiede la classifica

```
yuri@yuri-N550JK: ~/Documents/glt/rc_tris/codice
Complimenti! Sei il vincitore!
MENU
1.Gioca
2.Classifica
3.Esci
Cosa vuoi fare? 2
+-----+
| usr | games | won | lastGame |
+-----+
| user2 | 1 | 1 | 2017-07-15 16:47:44.540079 |
+-----+
| user5 | 0 | 0 | 0 |
+-----+
| user4 | 0 | 0 | 0 |
+-----+
| user3 | 0 | 0 | 0 |
+-----+
| user1 | 1 | 0 | 2017-07-15 16:47:44.540079 |
+-----+
MENU
1.Gioca
2.Classifica
3.Esci
Cosa vuoi fare? 
```