Relazione del Progetto - Laboratorio di Reti

Alessandro Puccia
Matricola 547462

Febbraio 2020

Indice

1 Introduzione

Word Quizzle offre principalmente un servizio di sfide tra diversi giocatori che concorrono nella traduzione dall'italiano in inglese di parole scelte dal server in maniera casuale.
Registrandosi a Word Quizzle vengono inoltre forniti ulteriori servizi come la possibilità di gestire una cerchia di amicizie, la visualizzazione delle proprie statistiche di gioco, la propria lista amici e la classifica.

Il server è privo di interfaccia GUI, ma utilizza un'interfaccia CLI per fornire le informazioni essenziali a seguito di eventi come una nuova connessione o la richiesta di un servizio.

Vengono forniti sia un client con interfaccia CLI che un client con interfaccia GUI. In entrambi i client viene data la possibilità di usufruire dei servizi specificati dal contratto e per come sono stati implementati a seguito di un crash con successivo riavvio del server non sarà necessario riavviarli per poterli utilizzare.

2 Descrizione generale

2.1 Server

Il server è stato implementato utilizzando:

- Un thread `Main` che implementa un selettore e che andrà: ad inizializzare le strutture del server all'avvio, ad accettare le nuove richieste di connessione e, se i client hanno inviato una richiesta, provvederà a leggere dai canali pronti in lettura.
- Un `ExecutorService acceptorOperator` formato da un thread che provvederà al parsing della richiesta del client e creerà il task appropriato da fare eseguire.
- Un `ExecutorService workersOperators` formato da due thread che andranno ad eseguire le richieste dei client.
- Un `ExecutorService diskOperator` formato da un thread che andrà a scrivere le informazioni sul disco a seguito di operazioni di registrazione, sfide, aggiunta di un amico.
- Un `ExecutorService answererOperator` formato da un thread che andrà a rispondere ai client al termine delle loro richieste.
- Uno `ScheduledExecutorService timeoutOperator` formato da un thread che andrà a segnalare, attraverso l'utilizzo di variabili atomiche, la scadenza dei `timer T1 o T2` previsti dal contratto.

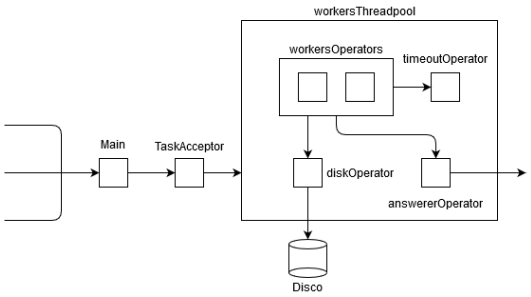

Figure 1: Struttura del server

2.1.1 Avvio del server

All'avvio del server, il thread Main:

- Carica da diversi file le informazioni degli utenti e le parole in italiano da utilizzare durante le sfide.
- Crea un `ExecutorService acceptorOperator`.
- Crea un oggetto di classe `WorkersThreadpool` che andrà ad inizializzare i rimanenti `ExecutorService`. La classe implementa tutti i metodi necessari per la sottomissione dei task.
- Crea il registry `RMI` e l'oggetto remoto che andrà ad essere utilizzato dai client per le operazioni di registrazione.
- Inizializza il selettore e un `ServerSocketChannel` su cui il thread `Main` accetterà le richieste di connessione. Successivamente attraverso l'utilizzo del metodo `select` si metterà in attesa di nuove connessioni o altri eventi riguardanti i canali pronti in lettura/chiusi (a causa di `IOException` o a seguito di un interruzione della connessione da parte del client dopo aver ricevuto la risposta ad un logout).

2.2 Client

2.2.1 Client con GUI

Il client che andrà ad utilizzare l'interfaccia GUI sarà composto da un thread `Main` che implementa un selettore e l'`EDT` di `Swing` che andrà ad effettuare tutte e sole le operazioni relative all'aggiornamento della grafica (attraverso l'utilizzo del metodo `SwingUtilities.invokeLater`) a seguito della ricezione di notifiche da parte del `Main` (ad esempio a seguito di una risposta, di un crash o la ricezione di una nuova sfida).
In questo modo, non si sovraccarica il gestore degli eventi grafici con operazioni che andrebbero a rallentare l'aggiornamento della GUI.

2.2.2 Client con CLI

Il client che andrà ad utilizzare l'interfaccia CLI sarà composto da due thread: il thread `Main` che andrà ad interagire con l'utente attraverso l'utilizzo di uno `Scanner` e una `PrintWriter`, e un thread `ClientSelector` che andrà principalmente ad inviare le richieste al server, leggere le risposte e le

richieste di sfida ricevute e poi comunicare al thread `Main` attraverso un'apposita struttura dati le risposte.
Durante le fasi di sfida è stato utilizzato un `InputStreamReader` che verrà controllato periodicamente attraverso il metodo `ready` per capire se l'utente ha scritto una nuova parola. In questo modo, il `Main` non rischia di bloccarsi in una chiamata del metodo `nextLine` e quindi potrà controllare eventuali messaggi relativi alla terminazione della sfida.

2.2.3 Avvio del client

All'avvio entrambi i tipi di client andranno a predisporre un selettore:

- Solamente alla richiesta da parte dell'utente del login verrà effettuata la connessione al server; nel caso del client CLI verrà creato il thread che interagirà col selettore. In questo modo se il server dovesse andare in crash per poi venir rimesso in esecuzione, non sarà necessario dover riavviare il client.
- Con lo stesso principio, lo stub che implementa la registrazione verrà richiesto solamente alla prima richiesta di registrazione e, a seguito di un eventuale crash del server verrà richiesto.

3 Serializzazione dei dati

I dati degli utenti e le parole in italiano verranno mantenuti su file serializzati attraverso lo standard `JSON`. I file degli utenti saranno distinti in questo modo:

- Una directory per ogni utente che andrà a contenere due file `JSON`: `UserInfo` e `UserFriendlist`.

  Nel primo file verrà mantenuto un oggetto `JSON` che andrà a contenere:

  - `hash`: risultato di una funzione hash applicata alla password specificata dall'utente al momento della registrazione.
  - `player`: un oggetto `JSON` che andrà a contenere al suo interno diversi campi come il nickname, il punteggio, le vittorie, le sconfitte e il rapporto vittorie/sconfitte dell'utente (rateo).

  Nel secondo file viene mantenuto un array `JSON` contenente i nickname degli amici.

  Si è scelto questo approccio perchè risulta più conveniente nel momento in cui il server sarà a regime, in quanto i file saranno più piccoli e la serializzazione comporterà un tempo minore.

- Un file `Words` contenente un array `JSON` di parole italiane.

4 Strutture dati e classi

4.1 Server

Il server nel suo totale andrà ad utilizzare molteplici strutture dati e classi:

- `Player`: mantiene al suo interno le informazioni basilari e relative alle prestazioni del giocatore: nickname, punteggio utente, vittorie, sconfitte, rateo. Verrà utilizzata principalmente dal server, in quanto i client utilizzeranno solamente le informazioni della classe per la deserializzazione attraverso `GSON`.

- `Clique`: mantiene al suo interno l'hashing della password dell'utente, un oggetto `Player` contenente le sue informazioni e una `ConcurrentHashMap<String, Player> friends` che conterrà le informazioni degli amici.
- `UsersGraph`: mantiene al suo interno una `ConcurrentHashMap<String, Clique> users` che andrà a memorizzare le "cricche" di ogni utente.
- `Challenge`: mantiene al suo interno lo stato di una partita così composto:
  - Un `AtomicInteger` che indica se una partita è stata accettata (stato 1), declinata (stato 2) o ancora non c'è stata risposta da parte dello sfidato o non è ancora scattato il `timeout T1` (stato 0).
  - Un `AtomicInteger` che indica in una partita quanti giocatori hanno terminato le parole a disposizione, se è scaduto il `timer T2` o se uno o entrambi gli utenti si sono disconnessi. Questa variabile permetterà di capire se sarà possibile inviare l'esito della sfida.
  - Un `Array` di stringhe contenenti le parole italiane scelte per la partita, esso verrà caricato solamente nella fase di setup della sfida e poi utilizzato successivamente.
  - Due oggetti `PlayerScore`: classe interna che incapsula lo stato di un utente all'interno della partita: a quale parola è arrivato, il punteggio attuale, il numero di parole corrette, il numero di parole non corrette e la relativa `SelectionKey` a cui inviare l'esito della sfida oppure una nuova parola.
- Un'`ArrayList<String> italianWords` che andrà a contenere le parole italiane da utilizzare durante le sfide, come già detto prima verrà caricata all'avvio del server, e quindi potrà essere utilizzata in maniera sicura da parte dei thread componenti del `workersOperators` in quanto acceduta in sola lettura.
- Un'`HashMap<SelectionKey, Future<String>> connectedUsers` che andrà a contenere i mapping tra connessioni e nickname degli utenti loggati.
  La decisione di utilizzare i `Future` in questo contesto è stata presa per evitare situazioni di inconsistenza nello stato dell'utente; ad esempio in cui il client effettui un operazione di login e crashi immediatamente. In questo modo prima di effettuare l'operazione di cleanup (che tra le altre cose rimuoverà l'utente dalla struttura dati contenente gli utenti online) verrà comunque completata l'operazione di login e solo successivamente eseguito il cleanup.
  Inoltre ottenendo il mapping tra `SelectionKey` e nickname dell'utente sarà possibile risalire alle informazioni dell'utente disconnesso.
- Una `ConcurrentHashMap<String, String> translatedWords` che andrà a contenere i mapping tra le parole in italiano e le relative traduzioni in inglese.
  Questa struttura dati verrà caricata dinamicamente e le traduzioni persisteranno per tutto il tempo di vita del server, in questo modo si andrà a richiedere la traduzione al servizio esterno solamente se essa non è presente nella struttura dati.
- Una `ConcurrentHashMap<String, Future<Challenge>> currentGames` che andrà a contenere i mapping tra nickname dell'utente e gli oggetti che incapsulano lo stato di una partita.
  Anche in questo caso vengono utilizzati i Future per evitare situazioni di inconsistenza relativi però allo stato GAMING dell'utente.
- Una `ConcurrentHashMap<String, InetSocketAddress>` che mappa i nickname degli utenti con i rispettivi indirizzi.

Questi indirizzi saranno utilizzati quando dovrà essere inoltrata una sfida via UDP ad un altro client.
La struttura dati sarà utilizzata anche per capire se un utente risulta online o meno.

4.2 Client con interfaccia GUI

Il client con interfaccia GUI utilizza quattro strutture dati così definite:

- Un `ArrayBlockingQueue<String> request` da una posizione che conterrà la richiesta da inviare al server con il protocollo `TCP`.
- Un `ArrayBlockingQueue<String> registrationRequest` da una posizione che conterrà i parametri per la registrazione via `RMI`.
- Un `ArrayBlockingQueue<String> challengeUpdateRequest` da una posizione che conterrà la richiesta di eliminazione dalla struttura dati `challengersRequests` a seguito di un rifiuto da parte dell'utente che ha ricevuto la sfida.
- Un `HashSet<String, Long> challengersRequests` che conterrà i mapping tra il nickname di un utente che ha inviato la sfida e un'approssimazione in millisecondi dell'istante di invio del pacchetto UDP contenente il messaggio.
  Questa struttura sarà poi utilizzata periodicamente dal thread `Main` per rimuovere le richieste di sfida che non sono più valide a seguito del timeout T1 e quindi aggiornare l'interfaccia grafica.

4.3 Client con interfaccia CLI

Il client con interfaccia CLI utilizza tre strutture dati così definite:

- Un `ArrayBlockingQueue<String> request` da una posizione che conterrà la richiesta da inviare al server con il protocollo `TCP`.
- Un `ArrayBlockingQueue<String> responses` da tre posizioni (in modo da poter contenere il massimo numero di risposte che il server può inviare in un dato momento, ad esempio una parola, l'esito della sfida, e un messaggio che notifichi una disconnessione del server) che conterrà la risposta del server.
- Una `ConcurrentHashMap<String, Long> challenges` che conterrà i mapping tra nickname dell'utente che ha inviato una sfida e l'approssimazione in millisecondi dell'istante di tempo di invio della sfida.
  In questo caso è stata utilizzata una struttura thread-safe poichè la struttura dati verrà utilizzata in lettura dal thread `Main` e in scrittura dal thread `ClientSelector`. Inoltre a causa delle limitazioni dell'interfaccia CLI e per non rendere l'interfaccia caotica non è stato implementato l'aggiornamento delle sfide durante la loro visualizzazione, come invece accade per l'interfaccia GUI.

5 Task implementati dal server

5.1 Task relativi alle operazioni dei client

- `TaskAcceptor`: come già detto prima questo task effettua il parsing della richiesta ricevuta dal client per creare il task appropriato.

- `LoginTask`: questo task effettuerà tutte le operazioni relative alla fase di login, in particolare controllerà che l'utente sia registrato, la correttezza della password e che non abbia già effettuato il login. Successivamente verrà spedito all'`answererOperator` il task che implementa l'invio di risposta. Come detto prima, il task è di tipo `Future`, in particolare verrà restituita una stringa contenente il nickname del client se l'operazione è andata a buon fine per evitare le situazioni di inconsistenza discusse prima.

- `LogoutTask`: questo task andrà a rimuovere l'utente dalla struttura dati `onlineUsers`. Successivamente verrà spedito all'`answererOperator` il task che implementa l'invio di risposta.

- `AddFriendTask`: questo task effettuerà tutte le operazioni relative alla richiesta di aggiunta di un amico, in particolare controllerà che sia registrato, e che non sia già presente tra gli amici. Successivamente verranno inviati due task al `diskOperator` che andrà a serializzare nei rispettivi file `JSON` la lista amici aggiornata. Dopodichè verrà spedito all'`answererOperator` il task che implementa l'invio di risposta.

- `ShowFriendsTask`, `ShowUserscoreTask`, `ShowRanksTask`: tutti questi task creeranno dei nuovi oggetti contenenti la copia attuale della lista amici, del punteggio utente e della classifica. Nel caso del task di visualizzazione classifica essa verrà inviata non ordinata, sarà il client a provvedere al suo ordinamento dopo aver effettuato la deserializzazione. Successivamente verrà spedito all'`answererOperator` il task che implementa l'invio di risposta.

- `StartChallengeTask`: questo task implementa le prime operazioni fondamentali della sfida, in particolare controlla che l'utente sfidato sia un amico dell'utente che ha originato la sfida, che sia online e che non sia già impegnato in un'altra partita. Se questi controlli vanno a buon fine, provvederà a creare l'oggetto `Challenge` che andrà ad incapsulare le informazioni della sfida e invierà all'utente sfidato la richiesta attraverso il protocollo `UDP`.
  Dopodichè inizializzerà un task che sarà eseguito dal `timeoutOperator` allo scadere del `timer T1`, questo task andrà a chiudere il `DatagramSocket` precedentemente aperto e controllerà se la sfida è stata accettata o, in caso contrario rimuoverà l'oggetto dalla struttura `currentGames` e invierà all'`answererOperator` il task che implementa l'invio di risposta.

- `SetupChallengeTask`: nel caso in cui verrà inviata al server una risposta affermativa da parte dell'utente sfidato, entro e non oltre il `timer T1`, verrà creato questo task che andrà a scegliere le parole italiane da inviare agli utenti e, se le relative traduzioni non sono ancora presenti nella struttura `translatedWords`, ad interagire col servizio esterno MyMemory per ottenerle. Successivamente verrà spedita ai rispettivi client la prima parola e verrà inizializzato il task da sottomettere al `timeoutOperator` per il controllo allo scadere del `timeout T2`.
  Allo scadere del `timeout T2`, verrà controllato se la sfida è già stata terminata a seguito della terminazione delle parole da parte dei due client oppure a causa di una disconnessione da parte di uno o entrambi i giocatori. In caso contrario, invierà per l'esecuzione un task che si occuperà della terminazione della sfida.

- `WordTask`: questo task andrà, nel caso in cui la partita non sia già terminata, a controllare la correttezza della parola inviata dall'utente ed aggiornerà le statistiche della partita per quel giocatore di conseguenza. Se il giocatore che ha inviato la traduzione e l'avversario hanno terminato le proprie parole a disposizione la partita verrà terminata, in caso contrario verrà inviata al giocatore la prossima parola se presente attraverso un task da sottoporre all'`answererOperator`.

- `EndChallengeTask`: questo task si occuperà della fase di terminazione della sfida, aggiornando le statistiche globali dei giocatori partecipanti. Una sfida può terminare con una vittoria per un utente e di conseguenza una sconfitta per l'altro oppure con un pareggio che non andrà ad influenzare il numero di vittorie e sconfitte di entrambi i giocatori. Successivamente verrà spedito all'`answererOperator` il task che implementa l'invio della risposta ad entrambi i giocatori. Dopodichè verranno inviati due task al `diskOperator` che andrà a serializzare nei rispettivi file `JSON` le informazioni utente aggiornate.

- Task eseguito a seguito di disconnessione del client: implementato come `lambda function` si occuperà di effettuare il cleanup.

5.2 Registrazione

La registrazione, per contratto, avviene tramite `RMI`. L'oggetto remoto che la implementa controllerà che l'utente non sia già registrato al servizio dopodichè creerà gli oggetti necessari per incapsulare le sue informazioni.
Successivamente invierà per l'esecuzione un task al `diskOperator` che implementerà le operazioni di serializzazione delle informazioni del nuovo utente e restituirà al chiamante una `String` contenente l'esito.

5.3 Task relativi alle operazioni di I/O sul disco

`WriteFriendilistTask`, `WriteUserInfoTask`: questi task andranno ad effettuare la serializzazione delle informazioni del client e successivamente andranno ad effettuare la scrittura sui rispettivi file utilizzando le funzionalità offerte dalla libreria `GSON`.

5.4 Task relativi alle operazioni di I/O su `TCP`

- `SendSimpleResponseTask`: questo task invierà l'esito dell'operazione richiesta dal client. Essendo gli esiti di lunghezza minore del `ByteBuffer` predisposto al momento della connessione, verranno inviati con una sola invocazione del metodo `write` sul `SocketChannel`.

- `SendFriendlistTask`, `SendUserscoreTask`, `SendRanksTask`: questi task invieranno rispettivamente la lista amici, le statistiche dell'utente, e la classifica. Queste informazioni potrebbero eccedere la dimensione del `ByteBuffer` predisposto al momento della connessione e quindi dovranno potenzialmente essere scritte attraverso multiple chiamate del metodo `write` sul `SocketChannel` associato al client.

6 Protocollo di comunicazione

6.1 `RequestMessages`

Questo `Enum` implementa le possibili richieste che i client possono effettuare, in particolare:

- `REGISTER nickname password`: utilizzato dal client con interfaccia GUI, per notificare il thread `Main` sulla richiesta di registrazione da sottoporre via `RMI`.

- `LOGIN nickname password`: utilizzato dal client per l'invio delle informazioni di login (nickname e password).

- `LOGOUT nickname`: utilizzato dal client per l'invio del nickname per l'operazione di logout.
- `LOGOUT_DISCONNECT`: utilizzato dal thread `Main` del server per notificare l' `acceptorOperator` la disconnessione di un client.
- `ADD_FRIEND userNickname friendNickname`: utilizzato dal client per richiedere l'aggiunta di un amico, insieme al nickname dell'utente che invia la richiesta e il nickname dell'utente da aggiungere.
- `SHOW_FRIEND userNickname`: utilizzato dal client per richiedere l'invio della lista amici.
- `SHOW_RANKS userNickname`: utilizzato dal client per richiedere l'invio della classifica.
- `SHOW_USERSCORE userNickname`: utilizzato dal client per richiedere l'invio delle proprie statistiche.
- `CHALLENGE_FROM challengerNickname challengedNickname`: utilizzato dal client per iniziare una nuova sfida.
- `CHALLENGE_FROM challengerNickname time`: utilizzato dal server per l'inoltro della sfida via `UDP`.
- `CHALLENGE_ACCEPTED challengerNickname challengedNickname`: utilizzato dal client che ha ricevuto una sfida per accettarla.
- `CHALLENGE_REFUSED challengerNickname`: utilizzato dal client che ha ricevuto una sfida per rifiutarla.
- `TRANSLATION userNickname word translation`: utilizzato dal client per inviare una traduzione.

6.2 `ResponseMessages`

Queto `Enum` implementa le possibili risposte che il server può inviare al client, in particolare:

- `USER_ALREADY_REGISTERED` codice 0: per indicare che l'utente è già registrato a Word Quizzle.
- `USER_NOT_EXISTS` codice 1: per indicare che il nickname specificato non esiste, utilizzato come risposta ad un'operazione di login.
- `WRONG_PASSWORD` codice 2: per indicare che la password specificata dal client è errata.
- `USER_ALREADY_LOGGED` codice 3: per indicare che il client risulta già aver effettuato un'operazione di login senza aver effettuato un logout.
- `USER_NOT_EXISTS_ADD` codice 4: per indicare che l'utente specificato non esiste, utilizzato come risposta ad un'operazione di aggiunta amico.
- `FRIEND_ALREADY_ADDED` codice 5: per indicare che l'utente specificato risulta essere già nella lista amici.
- `FRIEND_NOT_EXISTS` codice 6: per indicare che l'utente specificato non è presente nella lista amici.
- `CHALLENGE_REFUSED`, codice 7: per indicare ad un utente che aveva originato una richiesta di sfida che essa è stata rifiutata.

- `FRIEND_NOT_ONLINE` codice 8: per indicare che l'utente a cui si voleva inviare la sfida non risulta online.
- `FRIEND_IN_GAME` codice 9: per indicare che l'utente a cui si voleva inviare la sfida risulta già essere in gioco (NB: per in gioco si intende che si è già nella fase di setup della sfida (`SetupChallengeTask`)).
- `CHALLENGE_TIMEOUTED` codice 10: per indicare all'utente che voleva accettare la sfida la scadenza del `timeout T1`.
- `USER_CRASHED` codice 11: questa risposta viene spedita ad un utente che ha provato ad accettare una sfida inviata da un client che si è disconnesso.
- `USER_REGISTERED` codice 50: risposta inviata a seguito di una registrazione con successo.
- `USER_LOGGED` codice 51: risposta inviata a seguito di un login con successo
- `USER_LOGOUTED` codice 52: risposta inviato a seguito di un logout con successo.
- `FRIEND_ADDED` codice 53: risposta inviata a seguito dell'aggiunta di un amico avvenuta con successo.
- `CHALLENGE_ACCEPTED` codice 54: risposta inviata all'utente che ha originariamente inviato la sfida per notificarlo della sua accettazione.
- `FRIENDLIST` codice 55: codice di risposta a cui segue la lista amici in formato JSON:
- `RANKS` codice 56: codice di risposta a cui segue la classifica in formato JSON:
- `USERSCORE` codice 57: codice di risposta a cui seguono le statistiche del giocatore in formato JSON.
- `CHALLENGE_REQUEST_ACCEPTED` codice 58: risposta inviata all'utente che ha ricevuto la sfida e ha deciso di accettarla.
- `WORD` codice 59: utilizzata dal server per inviare una parola in italiano durante la partita.
- `RESULT` codice 60: utilizzata dal server per inviare il risultato della partita a seguito della terminazione. Saranno forniti, in ordine: l'esito della partita (che può essere `WIN`, `LOSS` o `TIE`), il punteggio della partita, il numero di parole esatte, il numero di parole non esatte, il numero di parole a cui non è stata data una risposta e il punteggio dell avversario.

7 Interfaccia GUI

L'interfaccia GUI è stata implementata attraverso l'utilizzo dell'interfaccia `Swing`. E' stato esteso il `JFrame` della libreria e diversi `JPanel` che saranno visualizzabili dall'utente:

- `MainFrame`: il `JFrame` principale dell'applicazione utilizza come layout un `BorderLayout` e al suo interno un altro panello con layout `CardLayout` per poter cambiare tra i diversi `JPanel` a seguito di un'azione dell'utente.
  Il costruttore del frame andrà ad allocare tutti gli altri `JPanel` ed aggiungerli al panello con `CardLayout`.

- `LoginPanel`: pannello che implementa due `JTextField` per poter inserire i parametri per l'operazione di login e una `JLable` cliccabile con cui sarà possibile passare al panello di registrazione.
- `MainMenuPanel`: pannello che implementa un menù di bottoni che permettono di richiedere i servizi di Word Quizzle.
- `RegistrationPanel`: pannello che implementa due `JTextField` per poter inserire i parametri per l'operazione di registrazione.
- `AddFriendPanel`: pannello che implementa un `JTextField` per poter inserire il nickname dell'amico da aggiungere.
- `ShowUserscorePanel`: pannello che implementa la visualizzazione del punteggio utente, vittorie, sconfitte e rateo dell'utente.
- `ShowFriendlistPanel`: pannello che implementa un `JScrollablePane` in cui all'interno viene posta una `JTable` che andrà a contenere i nickname degli amici.
- `ShowRanksPanel`: pannello che implementa un `JScrollablePane` in cui all'interno viene posta una `JTable` che andrà a contenere la classifica che sarà ordinata in questo modo: punteggio utente > vittorie > sconfitte.
  L'ordinamento sarà effettuato dal thread `Main` attraverso l'utilizzo di un opportuno `Comparator`.
- `ShowChallengesPanel`: pannello che implementa un `JComboBox` che conterrà i nickname degli utenti che hanno inviato una sfida. Se non sono presenti richieste di sfida non sarà possibile andare in questo pannello (il bottone contenuto nel `MainMenuPanel` sarà disattivato). Inoltre la `JComboBox` verrà aggiornata dinamicamente dall'`EDT` di `Swing` a seguito di un'accettazione, rifiuto, notifiche di nuove partite o per partite non più valide.

Il `MainFrame` e tutti i panelli implementano al loro interno dei metodi che possono essere utilizzati dal thread `Main` per la notifica di eventi come la ricezione di una nuova sfida, una risposta e così via. Tutti i pannelli implementano bottoni che permettono di poter cambiare i panel visualizzati e una `gif` di caricamento che sarà visualizzata quando verrà inviata una richiesta al server (disattivando tutti gli altri bottoni del pannello). E' stato inoltre aggiunta una classe `WQGUIUtilities` che implementa dei metodi statici per la creazione di alcuni componenti come `JButton`, `JLabel`, `JTextField`, `JPasswordField` e `JTable` che saranno utilizzati da tutti i pannelli.

8 Librerie utilizzate

Sono state utilizzate solamente due librerie esterne:

- `GSON`: questa libreria implementa le operazioni di serializzazione e deserializzazione di stringhe in formato `JSON`. E' stata utilizzata questa libreria in quanto risulta molto facile la deserializzazione e la serializzazione con i metodi `fromJson` e `toJson`, fornendo solamente la stringa da serializzare/deserializzare, il `FileInputStream` costruito sul file `JSON` e la definizione della classe dell'oggetto.
- `jBCrypt`: questa libreria implementa in modo essenziale l'hashing di una stringa con soltanto due metodi: `BCrypt.hashpw` per calcolare l'hash a partire da una password con l'aggiunta di un `salt` e `BCrypt.checkpw` per verificare che una password candidata sia corretta rispetto l'hashing fornito.

9 Istruzioni per l'utilizzo

Nella directory `out` sono presenti tre directory `server_jar`, `client_gui_jar` e `client_cli_jar`, in cui all'interno sono presenti i rispettivi `jar` e i file necessari per l'esecuzione.
Per eseguire il tutto basta eseguire il comando `java -jar xxxx.jar`.