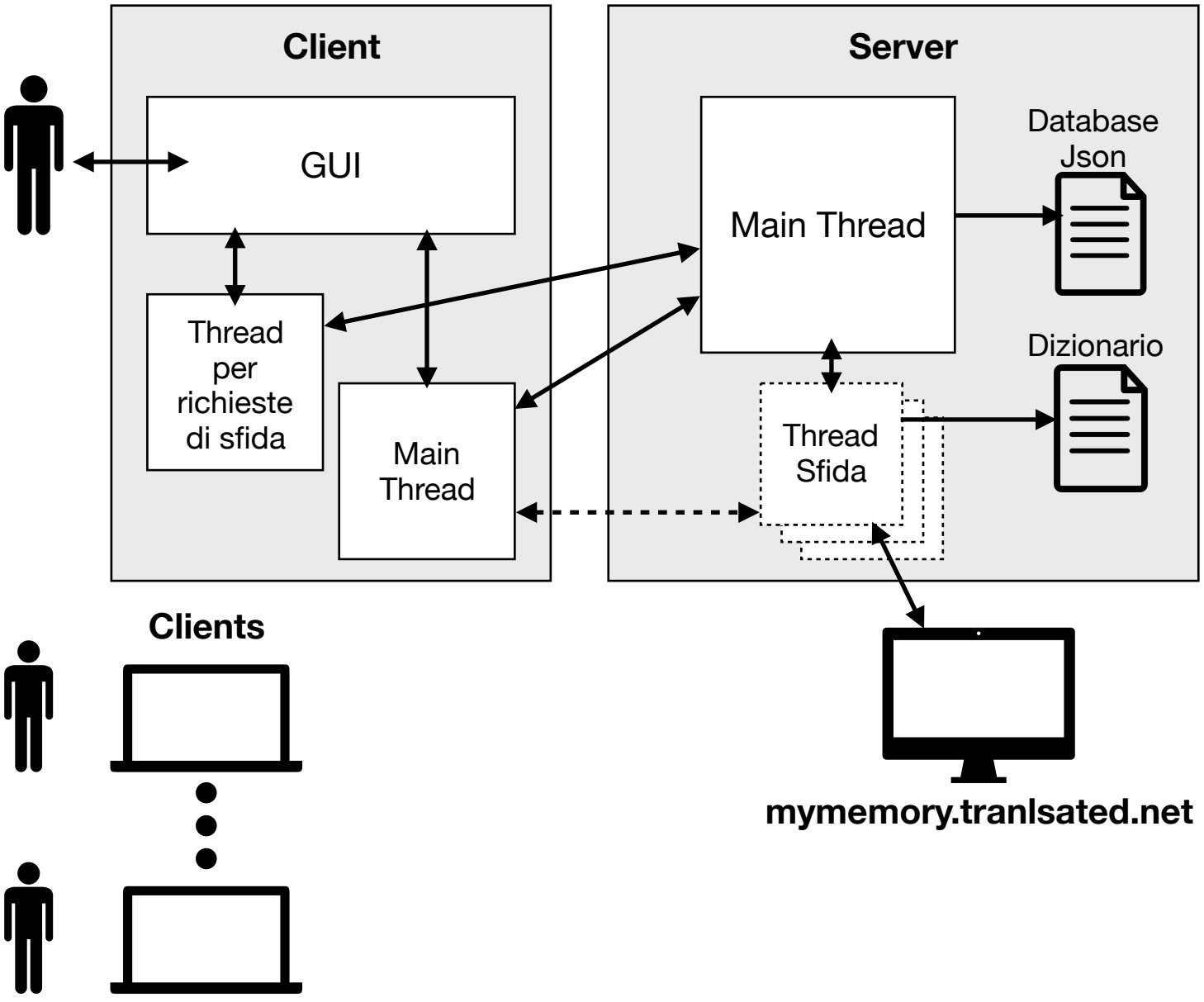
Relazione Progetto Word Quizzle

Simone Baccile
A.A. 2019/20
Laboratorio di Reti, Corso B

1. Descrizione architettura

In questo paragrafo verranno descritte le scelte progettuali di Word Quizzle e la sua architettura generale. Inoltre verrà descritto il protocollo di comunicazione per lo scambio di messaggi tra client e server.

1.1 Schema dell'architettura



LEGENDA:

1. **GUI → WQCLIENT**
2. **MAIN THREAD CLIENT → WQCLIENT**
3. **THREAD PER RICHIESTE DI SFIDA → CHALLENGELISTENER**
4. **MAIN THREAD SERVER → MAINWQS, WQSERVER**
5. **THREAD SFIDA → CHALLENGE**
6. **DATABASE JSON → DB.JSON**
7. **DIZIONARIO → DICTIONARY.TXT**

1.2 Protocollo di comunicazione

In questa sezione viene descritto il protocollo di comunicazione usato per lo scambio di messaggi tra client e server. Ad ogni messaggio inviato dal client il server risponde con OK se l'operazione è andata a buon fine oppure KO con un messaggio di descrizione dell'errore se l'operazione è fallita.

- **Login:**

Operazione che permette all'utente di accedere al gioco autenticandosi al server. L'operazione restituisce errore nei seguenti casi: l'utente è già loggato, il nome utente è sbagliato, la password è sbagliata, viene sollevata un'eccezione.
Request: 0 nomeUtente passwordUtente.
Response: OK / KO messaggioErrore.

- **Logout:**

Operazione che permette all'utente di uscire dal gioco. Questa operazione restituisce errore se viene sollevata un'eccezione.
Request: 1 nomeUtente.
Response: OK / KO messaggioErrore.

- **Aggiungere un amico:**

Operazione che permette all'utente di aggiungere un amico alla sua lista di amici. L'operazione restituisce nei seguenti casi: l'utente cerca di aggiungere se stesso tra i suoi amici, il nome utente dell'amico che si vuole aggiungere non esiste, viene sollevata un'eccezione.
Request: 2 nomeUtente nomeUtenteAmico.
Response: OK / KO messaggioErrore.

- **Visualizzare la lista amici:**

Operazione che permette all'utente di visualizzare la sua lista di amici. La lista amici viene restituita come un oggetto json. Se l'utente non ha amici l'oggetto json restituito sarà vuoto. L'operazione restituisce errore se viene sollevata un'eccezione.
Request: 3 nomeUtente .
Response: OK jsonObjectAmici / KO messaggioErrore.

- **Visualizza punteggio:**

Operazione che permette all'utente di visualizzare il suo punteggio attuale. L'operazione restituisce errore se viene sollevata un'eccezione.
Request: 4 nomeUtente.
Response: OK punteggio / KO messaggioErrore.

- **Classifica:**

Operazione che permette all'utente di visualizzare la classifica per punteggio con tutti i suoi amici. La classifica viene inviata come un oggetto json. L'operazione restituisce errore se viene sollevata un'eccezione.
Request: 5 nomeUtente.
Response: OK jsonObjectClassifica / KO messaggioErrore.

- **Sfida un utente:**

Operazione che permette all'utente di sfidare un utente. L'operazione fallisce nei seguenti casi: l'utente sta cercando di sfidare se stesso, l'utente ha già una sfida aperta, il giocatore che si vuole sfidare è occupato oppure offline, scade il tempo massimo di accettazione della sfida, viene lanciata un'eccezione. Se lo sfidante rifiuta la sfida l'utente non riceve messaggio di errore.
Request: 6 nomeUtente nomeUtenteSfidato.
Response: OK Timeout / KO messaggioErrore.

- **Accetta/Rifiuta sfida:**

Operazione che permette all'utente che ha ricevuto una sfida di accettarla o di rifiutarla. L'operazione fallisce se la sfida è scaduta oppure se viene sollevata un'eccezione.
Request: 7 nomeUtente nomeUtenteSfidante 0/1(rifiuta/accetta).
Response: OK / KO messaggioErrore.

- **Traduzione:**

Operazione che permette all'utente di inviare la sua traduzione dopo aver ricevuto una parola dal server. La risposta del server può essere o l'invio di un'altra parola o l'invio del resoconto della sfida o l'invio di un messaggio d'errore. L'operazione fallisce se viene sollevata un'eccezione.
Request: TR parolaTradotta.
Response: OK END risultato(WIN-LOSE_DRAW) punti / TR parolaDaTradurre / KO messaggioErrore.

1.3 Server

Il server è implementato tramite un selettore in modalità non bloccante, al quale è registrato di default un ServerSocketChannel registrato per operazioni di accettazione di nuove connessioni. Ogni nuova richiesta di connessione viene registrata al selettore tramite SocketChannel con operazioni di lettura. Ogni nuovo client connesso può inviare dei messaggi che seguono il protocollo di comunicazione e il server eseguirà l'operazione corrispondente fornendo un messaggio di risposta.
Tutte le operazione descritte nel protocollo vengono eseguite dal main thread del server tranne le operazione di gestione di una sfida. Quando una sfida viene accettata i Channel dei due sfidanti vengono "silenziati" (rimuovendo

l'operazione di lettura dall'interest set del channel) dal selettore del main thread e vengono passati ad un nuovo thread generato per gestire la sfida ed inserito in un Cached Threadpool. Questo thread si occuperà di gestire tutta la sfida in modo tale da non occupare costantemente il server per tutta la durata della sfida. Alla fine della sfida i Channel vengono riattivati per la lettura nel main thread e il thread sfida viene rimosso dal threadpool.
Il thread che gestisce la sfida prima di inviare la prima parola da tradurre ad entrambi i giocatori, seleziona un numero definito di parole dal dizionario leggendo tutte le righe del dizionario. Le parole vengono scelte generando casualmente un numero di riga. Le parole selezionate vengono poi tradotte sfruttando le api di mymemory.translated.net tramite richieste di connessione al servizio. Successivamente il thread può iniziare ad inviare le parole agli sfidanti.
Il server permette la registrazione di nuovi utenti a Word Quizzle tramite RMI esportando in remoto un oggetto che è possibile utilizzare per la registrazione.
Il server mantiene in locale un database con i dati degli utenti tramite un file json. In questo file vengono salvate le informazioni principali degli utenti come username, password, lista amici, punteggio, numero di vittorie, numero di sconfitte e numero di pareggi. Il database non viene aggiornato ad ogni operazione, poiché il server mantiene attiva una struttura contenente i dati degli utenti aggiornati. Tuttavia per avere la persistenza dei dati i salvataggi su database vengono fatti quando un utente esegue l'operazione di logout. In questo caso i dati aggiornati presenti nella struttura vengono salvati sul file json.
Il server adotta anche un meccanismo di salvataggio qualora il server stesso dovesse essere interrotto. Infatti prima della sua chiusura vengono effettuate delle operazione al fine di garantire una corretta chiusura del server stesso. Questo meccanismo permette il salvataggio di tutti i dati presenti nella struttura del server sul database, per tutti gli utenti che sono ancora online, occupati o marcati come da aggiornare.

1.4 Client

Il client implementa una semplice interfaccia grafica che permette all'utente di eseguire le operazioni fornite dalla specifica. In backend invece il client implementa un SocketChannel in modalità bloccante. Il client una volta stabilita la connessione con il server esegue anche un thread che tramite una DatagramSocket si mette in ascolto di eventuali richieste di sfida da parte di altri client. Questo thread verrà interrotto solo quando l'utente eseguirà il logout dal server.
La sfida da parte del client può essere eseguita in maniera differente in base al ruolo che svolge il client nella sfida. Se il client effettua una richiesta di sfida e quest'ultima viene accettata, la sfida continua nel main thread del client. Se invece il client ha ricevuto una richiesta di sfida e tale richiesta

viene accettata la sfida prosegue nel thread secondario, cioè quello che ha ricevuto la richiesta.

2. Schema dei thread e strutture dati

In questo paragrafo vengono descritti i thread che vengono attivati sia dal server che dal client. Inoltre vengono descritte le strutture dati utilizzate all'interno di ogni classe e come queste strutture garantiscono un corretto accesso concorrente alle risorse.

2.1 Thread attivati dal server

Il server attiva un thread timer (ChallengeTimer) quando riceve una richiesta di sfida. Questo thread una volta scaduto un timer invia un timeout all'utente che ha inviato la sfida segnalando che la richiesta di sfida ha superato il limite di tempo per essere accettata.
Un altro thread che può venire attivato dal server è quando un utente accetta la richiesta di sfida inviatagli ad un altro giocatore (Challenge). A questo thread vengono passate le chiavi degli utenti che si stanno sfidando. Le chiavi vengono "silenziate" per il Selector presente nel thread principale del server e vengono registrate ad un nuovo Selector all'interno del thread Challenge. Il thread Challenge a sua volta, dopo aver selezionato un numero di parole dal dizionario e dopo averle tradotte, fa partire un thread timer (TimeExpired) che non fa altro che settare una variabile a true quando scade il tempo massimo per poter rispondere alle domande per i due giocatori. Il thread Challenge si occupa della verifica delle traduzioni, del controllo di eventuali disconnessioni da parte dei giocatori, del controllo sulla scadenza del timer e del calcolo del risultato finale.

2.2. Strutture dati del server

Le strutture dati principali utilizzate all'interno del server sono:

- playerMap: una mappa sincronizzata che permette il salvataggio dei dati degli utenti, senza doverli necessariamente salvare ad ogni operazione sul database json. La mappa è sincronizzata sulle operazioni base e operazioni più complesse vengono sincronizzate con l'Object playerLock.
- gameMap: una mappa sincronizzata in cui vengono salvate le richieste di sfida da parte degli utenti. La mappa è sincronizzata sulle operazioni base e poiché vengono effettuate su di essa solo operazioni di get, put o remove non si è presentata la necessità di creare un ulteriore oggetto per sincronizzarla.
- gamePool: è un Cached ThreadPool nel quale vengono inseriti i thread Challenge.

- dict: è il nome del dizionario contenente le parole da tradurre. L'accesso ad esso viene garantito in mutua esclusione con l'utilizzo della variabile di tipo Object dictLock.
- dbName: è il nome del database json utilizzato per mantenere i dati degli utenti. L'accesso concorrente al database è garantito dalla sincronizzazione tramite la variabile Object dbLock.

2.3 Thread attivati dal client

L'unico thread attivato dal client è il thread ChallengeListener. Lo scopo di questo thread è di ricevere tramite una DatagramSocket richieste di sfida UDP. Se la sfida viene accettata essa prosegue all'interno del thread ChallengeListener, se la sfida viene rifiutata invece il thread torna in ascolti di altre richieste di sfida.

3. Descrizione delle classi

In questo paragrafo vengono descritte brevemente le classi implementate e in che modo vengono utilizzate all'interno del progetto.

3.1 MainWQS

Questa classe viene utilizzata per lanciare il server e per definire le impostazioni con le quali lanciarlo.

3.2 WQServer

E' la classe principale che implementa il vero e proprio server. Il metodo setup serve a preparare tutte le strutture dati necessarie al funzionamento del server, compresi Selector, ThreadPool e Registry. Il metodo start invece implementa la fase di ascolto del server, sia per le nuove richieste di connessione, che di lettura di eventuali messaggi in arrivo dai diversi client. Il cuore del server è realizzato dal metodo privato readOp. Questo metodo si occupa di fare il parsing del messaggio e di effettuare l'operazione necessaria a soddisfare la richiesta del client. Infine c'è il metodo privato close che viene utilizzato per chiudere correttamente il server e salvare eventuali dati di client ancora attivi.

3.3 Register

E' l'interfaccia utilizzata per l'implementazione del RMI per la registrazione di nuovi utenti al sistema. L'interfaccia viene utilizzata all'interno della classe RegisterImpl, WQServer e in WQClient.

3.4 RegisterImpl

E' l'implementazione dell'oggetto remoto che verrà utilizzato per effettuare la registrazione da parte di nuovi utenti. Alla sua creazione gli vengono passati

per riferimento la playerMap del server e il nome del database json per effettuare la richiesta e l'inserimento di nuovi utenti. L'unico metodo implementato è il metodo sincronizzato register, che permette appunto la registrazione di nuovi utenti. Questa classe viene utilizzata all'interno di WQServer.

3.5 Player

E' la classe che mantiene le informazioni degli utenti sulla mappa interna al server. Tra le informazioni salvate ci sono l'username, un oggetto PlayerInfo, lo stato attuale dell'utente (online, offline, in gioco, da aggiornare), la sua chiave registrata nel selettore e il numero di porta UDP per contattarlo in caso di sfida. I metodi implementati sono metodi di get e set banali. La classe Player viene utilizzata in WQServer, in RegisterImpl, in Challenge e in ChallengeTimer.

3.6 PlayerInfo

E' la classe che contiene le informazioni degli utenti che verranno effettivamente salvate nel database json. Le informazioni contenute nella classe PlayerInfo sono password, numero di vittorie, di pareggi e di sconfitte, punteggio totale e una lista sincronizzata di amici. Oltre ai metodi tradizionali di get e set, c'è il metodo sincronizzato addFriend che permette l'aggiunta di un nuovo amico alla lista, se non è già presente. Questa classe viene utilizzata all'interno della classe Player e in WQServer.

3.7 GameInfo

E' la classe che implementa le informazioni relative ad una richiesta di sfida. Le informazioni salvate sono la chiave dello sfidante nel selettore e il Timer avviato per la scadenza della richiesta. I metodi implementati sono la getSocket che restituisce il canale associato alla chiave e il cancelTimer che permette la cancellazione del timer. Questa classe viene utilizzata all'interno di WQServer e di ChallengeTimer.

3.8 Challenge

La classe Challenge estende la classe Thread ed è il thread che viene lanciato lato server per gestire la sfida tra due utenti. Alla sua creazione gli vengono passate le chiavi dei giocatori, le strutture Player dei giocatori, il nome del dizionario e l'oggetto per garantire la mutua esclusione sull'accesso al dizionario. Il metodo run implementa l'esecuzione della sfida tramite un selettore non bloccante in cui vengono registrate le chiavi dei due sfidanti. Dopo aver selezionato le parole della sfida dal dizionario e averle tradotte tramite richieste al sito mymemory.translated.net, inizia ad inviare le parole da tradurre ai due sfidanti. Prima di inviare la prima parola il thread Challenge attiva anche il thread TimeExpired che aggiornerà una variabile booleana che segnerà la fine del tempo necessario per tradurre le parole. Alla fine della sfida viene invocato il metodo checkWin per inviare i risultati

della sfida ai due utenti. La classe Challenge viene utilizzata all'interno di WQServer.

3.9 ChallengeTimer
E' la classe utilizzata per segnalare il timeout della richiesta di sfida da parte di un utente. Estende la classe TimerTask ed invia il timeout all'utente che aveva proposto la sfida. Inoltre rimuove dalla gameMap il GameInfo relativo alla sfida. La classe ChallengeTimer viene utilizzata all'interno di WQServer.

3.10 TimeExpired
La classe TimeExpired estende la TimerTask e viene utilizzata per segnalare la fine del tempo disponibile all'interno di una sfida per completare tutte le traduzioni. Questa classe viene utilizzata all'interno della classe Challenge.

3.11 WQClient
E' la classe che implementa il client che fa da tramite tra l'utente, tramite GUI, e il server. La GUI è realizzata in Swing tramite un JFrame principale e diversi JPanel distribuiti tramite un CardLayout. Il cambio tra i vari JPanel avviene tramite bottoni. Le stesse richieste al server vengono fatte tramite bottoni presenti in un menù principale, mostrato solo dopo aver effettuato l'accesso. Il backend del client che comunica con il server è realizzato tramite una SocketChannel in modalità bloccante. Ad ogni richiesta inviata al server corrisponde sempre una risposta ricevuta al client e mostrata all'utente tramite JOptionPane.

3.12 ChallengeListener
ChallengeListener estende la classe Thread e viene lanciato dal WQClient quando viene stabilita la connessione con il server. ChallengeListener si mette in ascolto di pacchetti UDP di richiesta di sfida tramite un DatagramSocket. Quando vengono ricevuti i pacchetti viene mostrato all'utente un JOptionPanel in cui poter accettare o rifiutare la proposta di sfida. Se la sfida viene accettata prosegue in ChallengeListener, altrimenti viene mandata una risposta al server con il rifiuto. Questo thread viene interrotto quando il client si disconnette dal server. Questa classe viene utilizzata nella classe WQClient.

4. Istruzioni di compilazione e esecuzione

In questa sezione viene descritto come compilare ed eseguire il codice del progetto.

4.1 Compilazione

Una volta scompattato il progetto eseguire all'interno della cartella principale i seguenti comandi per compilare tutti i file necessari:
javac -cp .:./api/:./api/gson-2.8.6.jar MainWQS.java
javac -cp .:./api/:./api/gson-2.8.6.jar WQClient.java

4.2 Esecuzione

Per eseguire il progetto bisogna prima avviare il server con il seguente comando:
java -cp .:./api/:./api/gson-2.8.6.jar MainWQS &
Successivamente è possibile avviare uno o più client con il seguente comando:
java -cp .:./api/:./api/gson-2.8.6.jar WQClient [&]

4.3 Informazione generali

E' possibile personalizzare alcune caratteristiche del server modificando le seguenti variabili:

- TIMEOUT in MainWQS: tempo di accettazione massimo per accettazione sfida.
- WORDS in Challenge: numero di parole totali da tradurre per la sfida.
- TIMEOUT in Challenge: tempo massimo per completare la sfida.

Inoltre se si vuole resettare il database di partenza "db.json" fornito, è possibile cancellare e riavviare il server che creerà automaticamente un nuovo database vuoto.