Scelte implementative

- Per la comunicazione tra client server viene utilizzato java.io. Il client comprende un primo Scanner per leggere l'input dell'utente da tastiera, un secondo Scanner (che legge da InputStream) chiamato "in" per leggere le risposte del server e un PrintWriter (creato tramite OutputStream) chiamato "out" per inviare i comandi al server. Il server contiene uno scanner "in" che legge i comandi del client tramite InputStream e un PrintWriter "out" per inviare le risposte al client.

- Per quanto riguarda la gestione dei suggerimenti il server restituisce una stringa con '+' quando la lettera e' presente nella soluzione e in posizione corretta, '?' quando la lettera e' presente ma in posizione sbagliata e 'X' quando la lettera non e' presente nella soluzione. Il server, nel caso in cui la lettera sia presente nel vocabolario(words.txt), prima effettua un conteggio delle occorrenze delle lettere nella soluzione(le salva tramite la Hashmap "answerMap"), poi, da questo conteggio diminuisce le occorrenze delle lettere verdi (lettere presenti e in posizione corretta) e infine costruisce la stringa del suggerimento come spiegato precedentemente inserendo '?' in base al numero di occorrenze rimaste della relativa lettera (decrementandole di 1 per ogni '?' inserito)

- La gestione dei comandi avviene tramite uno switch statement nel server.

- l'inizializzazione dei parametri nel client viene effettuata leggendo tramite BufferedReader un file testuale , facendo la split di ogni riga usando il carattere '=' per poi inserire ogni valore dopo l'uguale in un array di stringhe e successivamente inizializzare le variabili con questi valori. Le variabili inizializzate nel client sono serverPort (la porta del server) groupIp (l'ip del gruppo di multicast) udpPort (la porta su cui viene effettuato il bind del MulticastSocket). Nel server l' inizializzazione avviene nello stesso modo; le variabili inizializzate sono wordTime (il tempo che passa tra il cambio di una parola e l'altra e quindi il reset dello stato del giocatore (numero di tentativi e variabili di stato)), jsonFilePath(il path del file json utilizzato per memorizzare i dati dei giocatori), wordsFilePath(path del file contenente le parole scelte dal server), serverPort (porta del server) udpAddress (indirizzo ip del gruppo di multicast) ,udpPort (la porta a cui inviare i messaggi di multicast).

- La gestione dei messaggi udp avviene nel seguente modo: quando il client ha effettuato il login ,e quindi ha ricevuto come risposta dal server "Player logged in" viene creato un thread dal client che chiama il metodo receiveUDPMessage(), metodo che consiste nella creazione

un MulticastSocket , nell'unione al gruppo di multicast e nell'ascolto continuo di messaggi tramite un while(true), costruendo un DatagramPacket per la ricezione dei messaggi e aggiungendo ogni messaggio ricevuto dal MulticastSocket all'ArrayList chiamato udpNotifications; quando l'utente inviera' il comando showMeSharing e quindi ricevera' come risposta dal server "Sharing other people last games" verranno stampate tutte le stringhe (notifiche) all'interno di udpNotifications (quindi le notifiche ricevute fino a quel momento) tramite output stream. Dalla parte del server ogni volta che un client invia il comando "share" il server va a prendere dalla ConcurrentHashMap "usersStats" il risultato e il numero di tentativi dell'ultimo gioco relativi al giocatore corrente (quello che ha fatto il login) e li invia come stringa unica tramite il metodo sendUDPMessage, metodo che crea un DatagramSocket da cui invia un DatagramPacket contenente la stringa. La creazione del thread dal client viene fatta per evitare il blocco dello scambio di messaggi che verrebbe causato dal metodo receive (blocca fino a che non viene ricevuto un datagramma)

- Le parole vengono lette dal server dopo l'inizializzazione dei parametri di configurazione; tramite scanner vengono lette le righe del file words.txt, salvate in un ArrayList, e, successivamente viene scelta la parola tramite il metodo Random().nextInt. Il tempo che intercorre tra 2 parole viene gestito controllando la differenza tra il tempo corrente (currentTime) e il tempo iniziale (startTime), entrambi misurati tramite System.currentTimeMillis().

Strutture dati

- Per il salvataggio dei dati di ogni giocatore(sia username, password che statistiche) viene utilizzata una ConcurrentHashMap chiamata "usersStats". La ConcurrentHashMap viene caricata da file json all'avvio del server chiamando il metodo loadRegisteredPlayersFromJsonFile(); il metodo istanzia un oggetto Gson, un JsonReader per leggere da file json e un Type utilizzato dal metodo gson.fromJson() per deserializzare il Json letto dal parse tree in un oggetto del tipo Type utilizzando il meccanismo delle Reflection. Il Type in questo caso e' ConcurrentHashMap<String, UserStats>, ovvero il tipo della mappa "usersStats" che associa allo username di ogni giocatore un oggetto di tipo UserStats. L'oggetto UserStats contiene le seguenti informazioni relative al giocatore identificato dallo username: password, gamesPlayed, gamesWon, winCounter,

lastWinStreakLength, maxWinStreakLength, lastGameResult, attemptsPerWin; l'oggetto inoltre offre le seguenti funzionalita': addGameWon() per aggiungere una vittoria, addAttemptsOfCurrentWin() per aggiungere il numero di tentativi dell'ultima vittoria, setWinStreakLength() per aggiornare la lunghezza della win streak massima, setLastGameResult() per impostare il risultato dell'ultima partita come Victory/Loss, getLastGameResult(), getLastGameAttempts() e printStats() per stampare le statistiche del giocatore in un' unica riga. La struttura usersStats, oltre che all'avvio del server viene utilizzata nei seguenti casi: al momento della vittoria per aumentare i contatori delle partite giocate e vinte, per impostare il numero di tentativi della vittoria corrente e impostare il risultato della partita a "Victory", al momento della sconfitta per aumentare il contatore delle partite giocate, aggiornare il valore della win streak e impostare il risultato della partita a "Loss", al momento della registrazione per controllare se il giocatore e' gia presente nella struttura dati o altrimenti registrarlo, al momento del login per controllare che username e password siano corretti, nel comando sendMeStatistics per mostrare le statistiche del giocatore tramite il metodo printStats() e nel comando share per ottenere il risultato e il numero di tentativi dell'ultima partita prima di mandarli come messaggio broadcast. Subito dopo gli eventi di vittoria/sconfitta e registrazione viene chiamato il metodo updateJSONFile() che converte la struttura dati usersStats in formato json e la salva nel apposito file tramite FileWriter.write().

Schema thread attivati

- Il server consiste in un thread principale che chiama il metodo main e quindi gestisce lo scambio di messaggi tra client e server utilizzando una FixedThreadPool per generare un thread per ogni richiesta di connessione da parte dei client. Il client e' composto da un thread che chiama il metodo main e quindi si occupa ricevere l'input dall'utente,inviare comandi al server e ricevere le risposte dal server, e un thread che, subito dopo il log in del giocatore si mette in ascolto dei messaggi di broadcast.

Istruzioni su come compilare ed eseguire il progetto

Componenti progetto

- 3 file java (WordleClientMain.java, WordleServer.java , UserStats.java)
- 1 file jar (gson-2.10.jar)
- 2 file jar eseguibili (client.jar e server.jar)
- 2 file txt (serverconfig e client config usati come file configurazione)
- 1 file txt di parole (words.txt)

Compilazione

- cd “pathCartella” (per spostarsi nella cartella con file sorgente e libreria/e)
- javac -cp *.jar *.java (compila tutti i file java con tutte le librerie)

Esecuzione file jar

- cd “pathCartella” (per spostarsi nella cartella contenente i jar files eseguibili e i file txt di configurazione)
- java -jar server.jar (avvia il server)
- java -jar client.jar (chiaramente in una seconda finestra cmd, avvia il client)

Guida comandi

- Una volta avviato il client l’utente puo’ inviare username e password (per registrazione/riconoscimento)

- Dopo la risposta del server l’utente reinserisce username password per effettuare il login

- L’utente adesso puo’ inviare un comando al server tra playWORDLE, logout, sendWord, sendMeStatistics, share, showMeSharing

- Il comando sendWord e’ valido solo se il giocatore sta giocando (cioe’ dopo l’invio del comando playWORDLE e prima che il gioco sia terminato)

- Una volta inviato il comando playWORDLE l’utente puo’ scrivere il comando sendWord per inviare una parola

- Quando viene estratta una nuova parola il giocatore riceve una notifica (“The word has changed”) che indica l’ annullamento dell’ ultimo comando e il reset dello stato del giocatore. Se l’utente vuole giocare con la nuova parola deve quindi riscrivere playWORDLE.

- Il comando sendMeStatistics e’ valido solo quando il giocatore non sta giocando(quindi prima di inviare il comando playWORDLE o dopo la fine del gioco)

- Il comando share e’ valido solo dopo che l’utente ha terminato il gioco(dopo aver vinto o perso)

- Il comando showMeSharing puo’ essere effettuato in qualsiasi momento a parte subito dopo il comando sendWord(showMeSharing verrebbe preso come tentativo)

- Il logout puo’ essere effettuato in qualsiasi momento a parte subito dopo il comando sendWord(logout verrebbe preso come tentativo). Dopo il logout l’utente viene disconnesso.