Programmation concurrente, réactive et répartie :

Boggle

Trublereau Christopher

Master 1 : STL

Sommaire

I. Description générale

Le but de ce projet est d'implémenter une application clients-serveurs qui permet à des clients (ici, appelés joueurs) de jouer en multi-joueurs au Boggle. L'idée est de développer cette application en protocole TCP textuel, c'est-à-dire, en mode connecté où les clients et le serveur pourront s'échanger des requêtes sous forme de texte à l'aide de sockets.

Chaque commande envoyée par les clients et le serveur correspondra à une requête et sera sous la forme :
MA_COMMANDE_PRINCIPALE/ma_sous_commande1/ma_sous_commande2/**n**
(ma_sous_commande1 et 2 sont facultatives).

Il est également possible d'enrichir notre jeu de Boggle avec des extensions, comme un salon de discussion entre joueurs, ou encore, une interface graphique représentant la grille composée des lettres du tirage.

II. Manuel d'utilisation

Lancez l'exécution de l'application avec le **build.xml** fournit dans l'archive.

Si cela ne marche pas, lancez via **Eclipse** ou avec la commande **java**, en indiquant les 4 arguments suivant :

- Le chemin menant au dictionnaire ;
- La limite de temps à attribuer aux tours de jeu ;
- Le nom de l'hôte hébergeant la partie ;
- Et le pseudo sous lequel vous voulez jouer.

Un message de bienvenue sera affiché pour confirmer votre connexion.
Vous aurez alors accès aux scores de la partie en cours, ainsi qu'à la grille de lettres sur laquelle vous devrez vous baser pour former vos mots.

Après l'affichage de « Mot : », vous pouvez taper le mot que vous voulez. Une fois votre choix confirmé, appuyez sur entrée, et tapez la trajectoire que vous avez utilisé après l'affichage de « Trajectoire : ».

Votre mot et votre trajectoire sera analysé par le serveur et un message de validation (ou d'invalidation) sera affiché. Un mot considéré comme valide, est un mot d'au moins 3 lettres appartenant au dictionnaire et qui est formé avec les lettres de la grille avec la bonne trajectoire.

Après analyse, vous pourrez de nouveau taper un nouveau mot et une nouvelle trajectoire (après les affichages correspondant).

Vous recevrez un message lorsque le temps imparti sera écoulé, ceci marquera la fin du tour et les scores seront affichés. Après un délai de 10 secondes, un nouveau tour commence, une nouvelle grille sera affichée et vous pourrez recommencer à taper des mots et des trajectoires.

Au bout de 10 tours, la partie se termine et le score final est affiché. Le joueur ayant remporté le plus de points, gagne. Alors essayez de former des mots plutôt longs pour maximiser vos points.

Après un délai de 10 secondes, une nouvelle partie commence, ce qui réinitialisera les scores et entamera le premier tour avec une nouvelle grille. Vous n'aurez alors plus qu'à recommencer à former des mots et des trajectoires.

Que le meilleur gagne !

III. Description technique

L'application a été écrite en Java et est découpée en 3 packages :

- **serveur**, contenant :
  - La classe **Chrono** : Pour gérer le décompte avant la fin d'un tour.
  - La classe **Serveur** : Pour lancer les threads pour traiter les requêtes des clients.
  - La classe **Traitement** : Threads de traitement des requêtes client.

- **client**, contenant :
  - La classe **Client** : Pour gérer les joueurs, et envoyer les requêtes au serveur.

- **tests**, contenant :
  - La classe **TestGrille** : Pour tester le tirage aléatoire des 16 dés lors d'un tour de jeu.
  - La classe **TestScan** : Pour tester la saisie par un joueur de 3 mots et de 3 trajectoires consécutifs lors d'un tour.
  - La classe **BoggleMain** : Classe principale permettant de lancer l'application.

L’archive contient les 3 packages décrits précédemment, ainsi que :

- Un fichier **build.xml** : Pour compiler, exécuter, et générer la distribution du programme.

- L’archive **GLAFF-1.2.2.tar.bz2** : Contenant le GLAFF (**glaff-1.2.2.txt**).

1. Serveur

1.1. Serveur :

La classe Serveur est un thread contenant :

- Un constructeur **Serveur(int limiteChrono, String dictionnaire)** :

Initialise le nom du dictionnaire et la durée limite que doit atteindre le chronomètre pour marquer la fin d’un tour (limiteChrono doit être compris entre 3min et 5min, comme spécifié dans l’énoncé du projet, sinon erreur).

Après initialisation de ces variables d’instance, le serveur crée un ServerSocket paramétré sur le port 2018, puis lance sa méthode run().

- public void run() :

Se met sur écoute pour réceptionner un socket client afin de communiquer avec ce dernier (ecoute.accept()). Une fois le socket initialisé, une instance de la classe Traitement est créée pour gérer les requêtes du client.

1.2. Traitement :

Le classe Traitement est un thread initialisé par la classe Serveur pour déléguer la gestion des requêtes des clients. Elle contient :

- Un constructeur **Traitement(Socket client, int limiteChrono, String dico)** :

Initialise le lecteur de dictionnaire avec le nom du dictionnaire donné en paramètre (BufferedReader lecteurDico). Ainsi que la limite du chronomètre (donné en paramètre et fournit par la classe Serveur, comme le nom du dictionnaire).

Ensuite, les canaux d'entrée et de sortie du socket client sont synchronisés avec ceux du serveur :
inchannel = **new** BufferedReader(
new InputStreamReader(client.getInputStream()));
outchannel = **new** DataOutputStream(client.getOutputStream());

Pour finir, la méthode run() est lancée.

- public void run() :

Gère toutes les requêtes envoyées par le client.

Tant que le client ne s'est pas déconnecté (while(!deco)), une String récupère la commande de ce dernier (en lisant sur le canal d'entrée du serveur) et la découpe en sous-commande à l'aide d'un split (avec pour séparateur '/').

Puis, si le nombre de tour de jeu est égal à 0, le serveur démarre une nouvelle session en réinitialisant les scores à 0 et en envoyant la commande « SESSION/**\n** » aux joueurs (en écrivant sur le canal de sortie).
De plus, la commande « TOUR/**\n** » sera lue par le serveur.

Si le délai imparti est atteint, on arrête le chronomètre (currentchrono.stop()), puis on envoie les commandes « RFIN/**\n** » et « BILANMOTS/ » avec les scores et les mots proposés par les joueurs.
On attend ensuite 10sec avant de passer à la suite, afin de laisser le temps aux joueurs de voir les résultats du tour.

Si nous n'avons pas atteint la limite de tour (LIMITETOUR initialisée à 10 par défaut), alors la commande lue par le serveur sera « TOUR/**\n** ».
Sinon, nous envoyons le score final aux joueurs avec la commande «VAINQUEUR/» pour marquer la fin de la session. Puis, le nombre de tour est réinitialisé à 0 afin de redémarrer une nouvelle session.

Après tous les cas énumérés précédemment, le serveur lit la commande que le client lui envoie, en utilisant un switch (switch(sousCommande[0])), dont les case correspondront à l'une des commande principales citées ci-dessous :

. « CONNEXION » : Affichage de la connexion d'un nouveau joueur à tout le monde (affichage sur la sortie standard et envoie de la commande «CONNECTE/»). Puis, envoie de la commande «BIENVENUE/» pour donner les scores et le tirage courant si le joueur rejoint une session en cours.
Erreur si le nombre de sous-commandes est incorrect.

. «SORT» : Affichage de la déconnexion d'un joueur à tout le monde (affichage sur la sortie standard et envoie de la commande « DECONNEXION/ »).
Puis, sortie de la boucle pour fermer le socket client et ainsi ne plus recevoir ses requêtes (deco = true).
Erreur si le nombre de sous-commandes est incorrect.

. « TOUR » : Incrémentation de 1 du nombre de tour, puis, génération de la grille aléatoire à l'aide de la méthode privée genereGrille().
Ensuite, démarrage du chronomètre (currentchrono.start()), et envoie de la commande «TOUR/» aux joueurs pour qu'ils puissent commencer à saisir des mots et des trajectoires pour le tour actuel.

. «TROUVE» : On vérifie que le mot et la trajectoire trouvé par le joueur est correct à l'aide de la méthode privée motCorrect(String mot, String trajectoire).
Si c'est le cas, on concatène le mot parmi la liste des mots proposés et valides du joueur (motsproposes+=sousCommande[1]), on incrémente les points selon le barème défini par la méthode privée bareme(String mot). Puis, on envoie «MVALIDE/ » au joueur pour lui confirmer la validation de son mot.
Sinon, on envoie «MINVALIDE/ » au joueur pour lui confirmer l'invalidation de son mot.

. « ENVOI » (Extension du Chat entre joueurs) : Affichage du message à tous les joueurs sur la sortie standard, puis confirmation de la réception dudit message en envoyant «RECEPTION/» au joueur.
Erreur si le nombre de sous-commandes est incorrect.

. default : Affichage sur le flux d'erreur si aucune des commandes principales citées ci-dessus ne sont lues par le serveur.

- private String genereGrille() :

Pour générer le tirage aléatoire à chaque tour.

On initialise un tableau de 17 String dont les cases 1 à 16 contiennent chacune les lettres d'un des 16 dés de la version internationale.
Puis, pour chaque case du tableau, on initialise notre grille avec un caractère choisit au hasard à l'aide d'un Math.random().
Pour avoir 4 lettres par ligne, afin de représenter la grille telle qu'elle est dans le jeu officiel, nous testons si le reste de la division de l'indice courant par 4 est égale à 0 (d'où le parcours à partir de l'indice 1, sinon on aurait une lettre en trop sur la 1ère ligne).

- private boolean motCorrect(String mot, String trajectoire) :

Pour vérifier si le mot et la trajectoire donné par le joueur est correct.

Tout d'abord, on vérifie si le mot contient moins de 3 lettres. Si c'est le cas, le mot est incorrect (return false).

Sinon, pour chaque ligne du dictionnaire (line), on parcourt mot à mot (wordsinline) afin de savoir si l'un des mots correspond à celui qu'on cherche. Si c'est le cas, le mot est correct car présent dans le dictionnaire (return true).

Sinon, tous les mots de toutes les lignes ont été parcourus, donc le mot n'est pas présent dans le dictionnaire (return false).

- private int bareme(String mot) :

Pour calculer les points à chaque mot trouvé.

On se base sur le barème donné dans l'énoncé :

- Si le mot fait 3 ou 4 lettres, on gagne 1 point (return 1).
- Si le mot fait 5 lettres, on gagne 2 point (return 2).
- Si le mot fait 6 lettres, on gagne 3 point (return 3).
- Si le mot fait 7 lettres, on gagne 5 point (return 5).
- Sinon, le mot fait plus de 8 lettres, donc on gagne 11 points (return 11).

2. Client

La classe Client contient :

- Un constructeur **Client(String nameHost, String pseudo)** :

Initialise le pseudo du joueur, lui permettant de s'identifier pour jouer.
Puis, initialise un socket contenant le nom de l'hôte et le numéro de port 2018 pour se connecter au serveur.

Ensuite, les canaux d'entrée et de sortie du socket serveur sont synchronisés avec ceux du client :

```
inchannel = new BufferedReader(
new InputStreamReader (serveur.getInputStream()));
outchannel = new DataOutputStream(serveur.getOutputStream());
```

Pour finir, envoie de la commande « CONNEXION/ » avec le pseudo du joueur pour rejoindre la session de jeu.

- public void traitementClient() :

Gère toutes les requêtes envoyées par le serveur.

Tant que le client ne s'est pas déconnecté (while(!deco)), une String récupère la commande envoyée par le serveur (en lisant sur le canal d'entrée du client) et la découpe en sous-commande à l'aide d'un split (avec pour séparateur '/').

Ensuite, le client lit la commande que le serveur lui envoie, en utilisant un switch (switch(sousCommande[0])), dont les case correspondront à l'une des commande principales citées ci-dessous :

. « BIENVENUE » : Message de bienvenue au joueur, avec réception du tirage et des scores de la session en cours (affichage sur la sortie standard).
Erreur si le nombre de sous-commandes est incorrect.

. « CONNECTE » : Confirmation par le serveur de la connexion du joueur (affichage sur la sortie standard).
Erreur si le nombre de sous-commandes est incorrect.

. « DECONNEXION » : Envoie de la commande «SORT/» suivi du pseudo du joueur pour que ce dernier se déconnecte de la session de jeu.
Confirmation par le serveur de la déconnexion du joueur (affichage sur la sortie standard).
Puis, sortie de la boucle pour fermer le socket serveur et ainsi ne plus recevoir ses requêtes (deco = true).
Erreur si le nombre de sous-commandes est incorrect.

. « SESSION » : Message du serveur pour signaler aux joueurs qu'une nouvelle session de jeu commence (affichage sur la sortie standard).

. « VAINQUEUR » : Message du serveur pour signaler aux joueurs que la session de jeu est terminée (affichage sur la sortie standard, avec les scores finaux).
Erreur si le nombre de sous-commandes est incorrect.

. « TOUR » : Début d'un nouveau tour et affichage du tirage courant aux joueurs (affichage sur la sortie standard).
Le joueur peut alors saisir des mots avec leur trajectoire respective à l'aide de la méthode privée formuleMots().
Erreur si le nombre de sous-commandes est incorrect.

. « MVALIDE » : Confirmation par le serveur de la validation du mot saisi par le joueur.
Erreur si le nombre de sous-commandes est incorrect.

. « MINVALIDE » : Confirmation par le serveur de l'invalidation du mot saisi par le joueur.
Erreur si le nombre de sous-commandes est incorrect.

. « BILANMOTS » : Message du serveur prévenant les joueurs de la fin du tour actuel (affichage sur la sortie standard, avec les mots saisis et valides des joueurs, ainsi que le score de ces derniers).
Erreur si le nombre de sous-commandes est incorrect.

. « RECEPTION » (Extension du Chat entre joueurs) : Accusé de réception du message de Chat envoyé par le joueur.
Erreur si le nombre de sous-commandes est incorrect.

. default : Affichage sur le flux d'erreur si aucune des commandes principales citées ci-dessus ne sont lues par le client.

- private void formuleMots() :

Pour gérer la saisie des mots et des trajectoires du joueur pendant un tour de jeu.

Tant que le délai imparti du tour n'est pas écoulé
(!sousCommande[0].equals(«RFIN»)), une invite de saisie d'un mot et d'une trajectoire est affichée au joueur.
Il peut alors saisir sur l'entrée standard son mot, puis la trajectoire qu'il a utilisée pour le former.
Pour ce faire, on utilise une instance de la classe Scanner pour le mot, et une autre instance de cette même classe pour la trajectoire (**new** Scanner(System.***in***);).

Puis, on récupère les String tapées sur l'entrée standard à l'aide de la méthode nextLine() de Scanner ; pour finalement, envoyer la commande « TROUVE/ » au serveur afin de lui dire qu'un mot a été trouvé (ce qui va provoquer la vérification de ce dernier par le biais de la méthode motCorrect(String mot, String trajectoire) décrite dans la sous-partie **1.2.**).

3. Tests

Les tests sont découpés en 3 classes :

- public class TestGrille :

Main permettant de tester la génération aléatoire de grilles de Boggle.

Son implémentation est la même que celle de la méthode privée genereGrille() décrite dans la sous-partie **1.2.**

La seule différence est qu'on affiche la grille sur la sortie standard.

- public class TestScan :

Main permettant de tester la saisie de 3 mots et de 3 trajectoires consécutifs.

L'implémentation est la même que celle de la méthode privée formuleMots() décrite dans la partie **2.**

La différence est que nous bouclons 3 fois, afin d'effectuer 3 saisies. De plus, l'affichage des mots et des trajectoires se fait sur la sortie standard.

- public class BoggleMain :

Main permettant de tester l'ensemble de l'application.

Une instance de la classe Serveur est créée avec les 2 premiers arguments saisis sur la ligne de commande, respectivement : le chemin menant au dictionnaire, et le délai imparti de chaque tour.

Puis, une instance de la classe Client est créée avec les 2 derniers arguments saisis sur la ligne de commande, respectivement : le nom de l'hôte pour indiquer vers qui on se connecte, et le pseudo sous lequel on veut jouer.

Étant donné que la classe Client n'est pas un thread, nous devons appeler la méthode traitementClient() manuellement (d'où le cli.traitementClient()).

IV. Remarques

En raison d'un problème d'implémentation du client en OCaml, les 2 parties ont été faites en Java.

Les commentaires présents dans le code sont là pour combler ce qui a pu être omis dans ce rapport.