

Mai 2018

Rapport PC2R - Boggle

Amel Arkoub & Ling-Chun SO

Table des matières

Introduction	2
Le sujet	3
Quelque liberté.....	3
Modification apportée au sujet.....	3
Les extensions	3
Description algorithmique du client autonome	3
Le serveur	6
Le processus principal	6
Le thread s'occupant des échanges avec un client donné	6
Le client	7
Fonctionnement	8
Interface graphique et envoi de messages	8
Traitement d'information envoyé par le serveur.....	8
Fiche d'utilisation du serveur et du client	9
Le serveur	9
Le client	9
Conclusion	11

Introduction

Nous souhaitons créer notre propre jeu de lettres multijoueur en ligne, de type Boggle. Nous avons décidé d'écrire le serveur en C et le client en Java. Ces choix se justifient de la manière suivante. Maîtrisant le C et ayant déjà eu l'occasion d'écrire des serveurs dans ce langage, c'est de façon naturelle que nous optons pour ce langage. Quant au client, Java propose la bibliothèque d'interface graphique JavaFX, qui est facile d'utilisation et dont l'esthétisme final est incontestable.

Le sujet

Le protocole de l'énoncé est respecté.

Quelque liberté

Modification apportée au sujet

Nous prîmes l'initiative de considérer uniquement la vérification immédiate des mots envoyés par le client au serveur. Ainsi, il est inutile de lancer le serveur avec l'option – ***immédiat***, le serveur vérifie toujours que le mot envoyé par le client fait bien partie du dictionnaire.

Les extensions

Toutes les extensions obligatoires, à savoir le chat et la vérification immédiate, sont implantées.

Concernant les extensions facultatives, vous pourrez trouver un client graphique, c'est-à-dire le cube et la représentation de la trajectoire, ainsi qu'un client autonome, tricheur absolu et imbattable, qui vient venter son intelligence incontestable sur le chat. De plus, si vous le souhaitez, les grilles peuvent être générées automatiquement.

Description algorithmique du client autonome

Le client autonome calcul toutes les combinaisons possibles du Boggle et envoie les mots correctes au serveur après l'avoir vérifié dans un dictionnaire,

Ce client est très difficile à reconnaître mais peut néanmoins se faire trahir par son ton un peu vantard !

Le client suit cet algorithme :

- 1) Chargement du dictionnaire et de réplique
- 2) Connexion au serveur avec un nom choisi au hasard (peut être enrichi)

- 3) Attente d'un tour
- 4) Calcul toutes les solutions possibles de la grille
- 5) Envoi des solutions au serveur

A partir de l'étape 3, le client autonome devient un peu bavard et envoi quelques messages dans le chat.

Calcul des solutions:

Le calcul de toutes les combinaisons possibles s'effectue par un algorithme récursive qui explore toutes les possibilités et se reposant sur du backtracking.

Il est a noté que l'algorithme qui va être présenté et lancé sur chaque case (point de départ de l'algorithme) de la grille.

L'algorithme procède comme ceci :

Entrée :

- *visit* : une matrice de booléen
- *word* : une chaîne de caractères du mot en construction
- *traj* : une chaîne de caractères de la trajectoire en construction
- *row* : position de la ligne
- *col* : position de la colonne

Si *word* est un mot du dictionnaire et de taille supérieure à 2 alors

On ajoute *word* et *traj* dans les solutions

$visit[row][col] \leftarrow true$ (Cela permet d'éviter de repartir en arrière et de bouclé à l'infini)

On effectue un appel récursif sur les 8 chemins s'ils sont possibles (haut, bas, gauche, droite, haut-gauche, haut-droite ...), c'est-à-dire non visité et ne sortant pas de la matrice.

Dans ce cas-là, on ajoute aussi la nouvelle lettre et trajectoire

$visit[row][col] \leftarrow false$ (cela permet de rendre visitable pour la fonction réursive appelante).

Le serveur

Le processus principal

Une fois le serveur lancé, le processus principal attend sur deux descripteurs de fichier : un correspondant à la socket d'écoute sur le port donné en argument (ou sinon celui par défaut), et l'autre sur un timer.

Quand un client veut se connecter, le processus principal va créer un thread qui s'occupera de la connexion avec ce client.

Avec chaque fois que le timer finit, le processus principal va effectuer une tâche. Voici la séquence de tâches réalisées par le processus :

- 1) Au lancement, le processus envoie le message SESSION/
- 2) Au bout de 10 secondes, le timer envoie un message au processus via le descripteur de fichier pour lui signaler que le temps imparti est écoulé
- 3) Le processus charge une grille et envoie à toutes les connexions établies avec des clients le message TOUR/tirage/
- 4) Au bout de 3 min, le timer envoie un message au processus
- 5) Le processus envoie RFIN/ à tous les clients
- 6) Les étapes 3, 4 et 5 se répètent autant de fois qu'on a de tours par session
- 7) Le processus envoie VAINQUEUR/bilan/
- 8) On retourne à l'étape 1

Le thread s'occupant des échanges avec un client donné

Le thread reçoit les différents messages envoyés par son client, et lui envoie la réponse adéquate.

Le client

Le client a été écrit en Java en s'appuyant sur JAVAFX, nous avons donc codé une interface graphique dont nous allons expliciter chaque fonctionnalité.

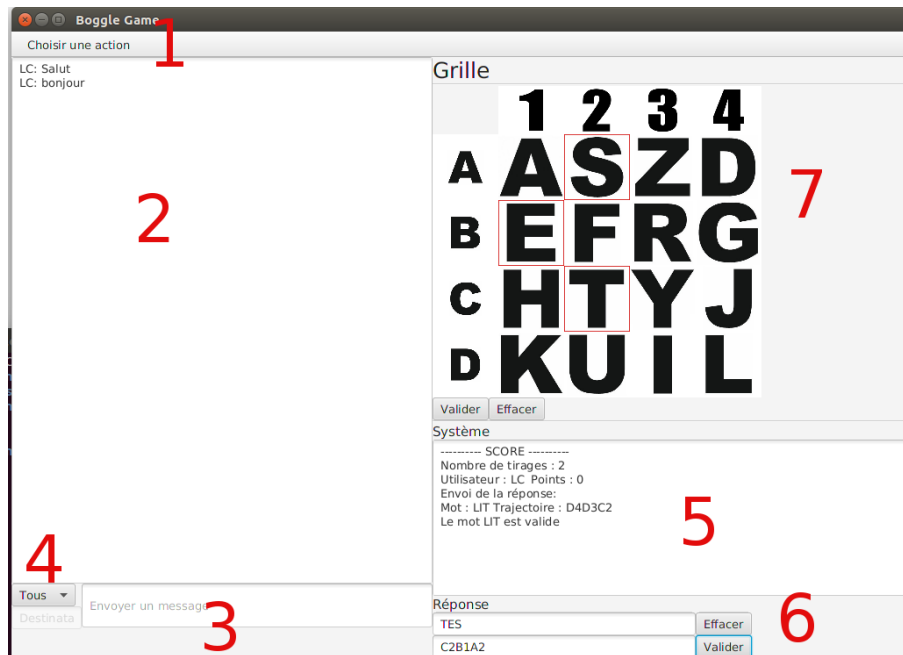


Figure 1 - Capture d'écran du client JAVA

- 1) Ce menu permet de choisir une action à effectuer:
 - Se connecter à un serveur, un popup apparaîtra et demandera les informations du serveur ainsi que le pseudo du joueur
 - Se déconnecter d'un serveur de jeu
 - Quitter l'application
- 2) Cette zone correspond au chat, les messages envoyés et reçus y sont affichés
- 3) Cette zone correspond au contenu que vous souhaitez envoyer au chat
- 4) Ce menu déroulant vous permet de choisir si votre message sera public ou privé, s'il est privé, il y a un champ pour indiquer à qui vous souhaitez l'envoyer.
- 5) Cette zone affiche les informations relatives à la session, c'est-à-dire les connexions de joueurs, le début et fin de tour ainsi que les résultats.
- 6) Permet d'écrire et d'envoyer une réponse (un mot trouvé) au serveur, il faut spécifier le mot et la trajectoire (pas obligatoire cf.7).
- 7) Cette zone contient la grille du Boggle, il est possible de directement tracer la trajectoire, il suffit d'effectuer des clics successifs dans les cases, un encadrement rouge est affiché dans chaque case sélectionnée, dans ce cas, le mot et la trajectoire est construit au fur et à mesure des clics et est visible dans la zone 6.

Fonctionnement

Cette application peut être divisée en deux parties, l'interface graphique et l'envoi de message, mais aussi la récupération et traitement d'information des messages envoyés par le serveur.

Interface graphique et envoi de messages

Le fichier `BoggleWindow.java` contient toutes les instanciations des éléments graphiques de l'application du client Boggle.

Lorsque la demande de connexion est envoyée, le client récupère le flux qui sert à envoyer vers le serveur et toute action de l'application, c'est-à-dire validation d'envoi d'un mot, validation d'un message écrit dans ce flux et donc envoi l'information au serveur.

Traitement d'information envoyé par le serveur

Le fichier `GameRunner.java` contient le code permettant de récupérer les informations envoyées par le serveur.

Ainsi, celui-ci revient simplement à une boucle qui attend une entrée sur le flux entrant et après parsing et analyse de l'information effectue l'action adéquate tel que charger une nouvelle grille, récupérer un message, afficher le résultat.

Il est à noter que si l'action doit modifier l'application JavaFX, il faut englober l'action dans la méthode `Platform.runLater()` en raison du fait que le Thread `GameRunner` n'est pas un thread JavaFX.

Fiche d'utilisation du serveur et du client

Le serveur

Un makefile est disponible dans Boggle_PC2R/Boggle_Serveur.

Pour lancer le serveur avec les options de vous voulez, veuillez entrer les commandes suivantes dans une console, à partir de Boggle_PC2R/Boggle_Serveur :

- make
- ./bin/serveur [-port] *num_port* [-tours] *nb_tours* [-grilles] *grille1 ... grilleN*

On note que chaque option est facultative, et les valeurs par défaut sont :

port=2006, tours=3, et les grilles sont générées automatiquement

Ainsi vous pouvez lancer le serveur en ne faisant que :

- ./bin/serveur

Ou bien une composition de votre choix, comme par exemple :

- ./bin/serveur -tours 10

Si vous optez pour l'option -grilles *grille1 ... grilleN*, sachez que ces grilles seront proposées en boucles.

Le client

Un fichier build.xml est disponible pour générer une distribution avec ant dans Boggle_PC2R/Boggle_Client.

Pour générer une distribution, veuillez entrer les commandes suivantes dans la console, à partir de Boggle_PC2R/Boggle_Client :

- ant clean

- ant jar

L'exécutable jar se trouvera dans le dossier Boggle_PC2R/Boggle_Client/jar.

Pour lancer le client, veuillez à partir de Boggle_PC2R/Boggle_Client écrire dans la console:

- java -jar jar/Boggle_Client.jar [-serveur] *ip* [-port] *numport*

Vous pouvez soit lancer avec aucun argument comme ceci :

- java -jar jar/Boggle_Client.jar

Ou alors spécifier chaque champ par exemple:

- java -jar jar/Boggle_Client.jar -serveur 127.0.0.1 -port 2018

Le client autonome

Un fichier build.xml est disponible pour générer une distribution avec ant dans Boggle_PC2R/BoggleCheaterClient.

Pour générer une distribution, veuillez entrer les commandes suivantes dans la console, à partir de Boggle_PC2R/BoggleCheaterClient :

- ant clean
- ant jar

L'exécutable jar se trouvera dans le dossier Boggle_PC2R/BoggleCheaterClient/jar.

Pour lancer le client, veuillez à partir de Boggle_PC2R/Boggle_Client écrire dans la console:

- java -jar jar/Boggle_Cheater_Client.jar -serveur *ip* -port *numport*

Par exemple :

- java -jar jar/Boggle_Cheater_Client.jar -serveur 127.0.0.1 -port 2018

Conclusion