Rapport de Projet : Jeu de Lettres

PC2R

Paul Charton

Introduction au probleme et journal de progression

Le probleme du devoir consiste a implementer le jeu de lettre *Boggle* avec un architecture client-serveur, avec comme contrainte principal d'utiliser deux languages de programmation de famille différentes pour le client et le serveur.

Dans ce cadre, j'ai choisis le language C pour implementer le serveur et le language scala pour le client, j'ai choisi ces deux languages en particulier car j'ai déjà programmmer des projets plus ou moins important à l'aide de ces languages et me sont donc familier, en plus d'être tout les deux portable. je l'ai ait ensuite réparti comme il me semblait le plus pratique et le plus logique.

J'ai par la suite commencer a travailler sur la connection Client-Serveur en elle même, et donc les sockets, avec des threads coté serveur, a suivi la mise en place d'une interface graphique coté client et enfin la gestion du protocole texte.

Présentation des fontions et structure de données

J'ai trois fonction de thread different pour mon serveur :

- accepteClient : C'est la fonction qui va se charger de l'arriver de nouveau client, dés qu'une connection est accepter, cette fonction va crée une nouvelle instance de dataClient, puis remplir la structure avec toutes les variables communes (grille, timer, condition, mutex, etc. . .), une fois l'instance remplie, elle mise dans un tableau, ce qui va permettre de récuperer toutes les informations propre au client (socket et pseudo principalement) qui serviront notament lors du broadcast de message. La structure du client va ensuite être envoyer a un autre thread dedier aux traitement des clients.
- traiteClient : Cette fonction va s'occuper d'un client exclusivement, ainsi que quasiment toutes les communication entre ce client et le serveur, cette

fonction est charger de demander le pseudo (et le re-demander si le pseudo n'est pas valide), d'envoyer la grille et le timer au bon format, ainsi que de verifier les mots envoyer par le clien et d'ajouter touts les messages reçu dans la file de messages.

- boggle : C'est "le vrai serveur" de cette application. C'est la fonction en charge du tirage aléatoire des grilles, du calcul du timer, de la verification des pseudo ainsi que les messages en tout genres.

J'utilise ensuite plusieurs structure :

- Les structures dataClient et dataServ : Ce sont simplement les structures en charge des informations personnels et commune de chacun, c'est par exemple grâce au champs commun "grille" et a la copie réaliser par la fonction accepteClient que l'on peut transmettre la même grille a tout les clients sans avoir besoin de variables globales.
- Les structures message et fileMessage : Ces deux structures, a l'aide des fonction enfiler et delifer m'ont permit de mettre en place une FIFO pour les messages, en effet dans la fonction traiteClient, on réçoit les messages que souhaite envoyer cette utilisateur aux autres, or la fonction ne posséde pas d'information concernant les sockets des autre clients, ou même leurs pseudo dans le cadre des message privee, traiteClient se charge donc d'ajouter le message reçu a la file des messages a traiter, pour que la fonction boggle puisse ensuite les re-distribuer a qui de droit.

Dans les autres fonctions, il y a : * enleveAccent : Fonction qui permet d'enlever les accents du dictionnaire utiliser pour pouvoir faire correctement les compraraisons. (Ici le dictionnaire de base est el GLAFF).

- rechercheDansDico : Une fonction qui va lire dans le fichier cree a l'aide de la fonction précedente.
- rechercheDansDico : Une fonction qui va lire dans le fichier cree a l'aide de la fonction précedente.
- recherchePseudo : C'est la fonction qui permet de valider un pseudo et d'envoyer un message prive, on parcours la liste des joueurs tout en comparant les pseudo, on renvoie l'index si on trouve et -1 si on ne trouve pas.
- extractPseudo : Fonction qui va jouer avec les indices du tableau tab, pour enlever les caracteres inutile du protocole et ne garder que l'information.
- dejaProposer : Fonction qui permet de savoir si le mot proposer par le joueurs l'a déjà était pas quelqu'un d'autre ou même lui-même.
- ajouterMot : Fonction qui permet d'ajouter le mot a la liste des mots déjà proposer.
- tailleMot : Fonction qui renvoie la taille du mot, utiliser principalement pour les writes.
- valideMot : Fonction qui permet de faire tout les test conforme aux protocoles sur un mot proposer par un joueur.

- messageBroadcast : Fonction qui permet d'envoyer un message a tout les joueurs.
- messagePrive : Fonction qui permet d'envoyer un message a un joueur.

Manuel d'installation

Pour le serveur il suffit d'avoir le programme gcc ainsi que le programme make d'installer, puis se positionner dans le répertoire Serveur, taper les commandes "make" pour compiler et "./exe/serveur" pour lancer le programme.

Pour le client, il faut installer le programme sbt, puis se placer dans le répertoire clientProjet et taper la commande "sbt run" pour lancer l'application.

Guide de l'utilisateur

Le serveur peut se lancer avec les options *s,t,p,d*. *s* pour définir le nombre de session, *t* le nombre de minutes dédier a la réflexion dans un tour, *p* le port de la socket qui va faire la fonction accepteClient et *d* le chemin vers un dictionnaire.

Le client ne prend pas d'option, mais va demander au démarrage votre pseudo, qui servira d'identifiant et qui doit être unique, l'adresse et le port du serveur. La fenetre va ensuite s'ouvrir, il peut il y avoir parfois un petit délai. Pour proposer un mot, il suffit d'appuyer sur les lettres correspondante, le client gere les regles d'adjacent et de ne pas réutiliser les lettres. Pour ensuite envoyer le mot au serveur, il suffit de cliquer dessus. En cliquant sur "Reset" on efface le mot qu'on a commmencer a construire. On peut discuter dans le chat avec le la barre qui se trouve en bas de l'écran, pour un message envoyer a tout les joueurs, simplement taper son message. Pour un message privé, la syntaxe est la suivante : "pseudo:contenu", attention a ne pas mettre d'espace.