Sorbonne Universités

SORBONNE UNIVERSITÉ

CRÉATEURS DE FUTURS DEPUIS 1257

MASTER 1
PROJET PC2R

Jeu de Lettres : ***Boggle***

Réalisé par :

Sofiane GHERSA - 3525755
Naim CHOULLIT - 3602541

Année universitaire : 2017-2018

1. Introduction 03

2. Problématique 03

3. Conception 04

    3.1. Client 04

    3.2. Serveur 09

4. Expérimentation 13

5. Amélioration 13

6. Extensions 13

7. Conclusion 14

1. Introduction

Le monde de l'informatique en général et de la programmation en particulier est très vaste, les informaticien font face à des très grand problèmes a chaque implémentation, parmi ces problèmes est celui de choix d'une architecture et de langage de programmation.

Il existe plusieurs architectures et langages et chaqu'un entre eux il a ses avantages et ces inconvénients.

OCaml est un langage fonctionnel augmenté de traits, permettant la programmation impérative et typé statiquement. Il se prête à la programmation dans un style fonctionnel, impératif ou orienté objet. Pour toutes ces raisons, OCaml entre dans la catégorie des langages multi-paradigme. Son système de type permet un développement logiciel d'une grande fiabilité. De plus en plus d'entreprises, par exemple Facebook et JaneStreet, l'intègrent dans leurs projets.

Java est un langage de programmation orienté objet, la particularité et l'objectif central de Java est que les logiciels écrits dans ce langage doivent être très facilement portables sur plusieurs systèmes d'exploitation tels que Unix, Windows, Mac OS ou GNU/Linux, avec peu ou pas de modifications. Pour cela, divers plateformes et frameworks associés visent à guider, sinon garantir, cette portabilité des applications développées en Java.

2. Problématique

Dans ce travaille on va réaliser un jeu de lettre multijoueurs de type ***Boggle***.

On fournit à tous les participants connectés la même grille (tirage de lettres d'un plateau de 4x4 lettres). les participants envoient des mots construits à partir de ces lettres et à la fin du tour, tous les participants marquent des points en fonction de leurs propositions, ensuite le jeu continue pour tous les participants avec une nouvelle grille.

3. Conception

Nos choix techniques sont :

- L’application est réalisée suivant une architecture client/serveur.
- La partie client est écrite avec langage ***Java.***
- La partie serveur est écrite avec langage ***OCaml version 4.06.0***.

3.1. Client (Customer)

Pour le développement de notre client on a choisi le langage Java qui est un langage Orienté-Objet, multiplateforme ce qui nous permet d'exécuter notre application sur différent plateforme en fonction des clients et qui nous permet de transformer notre Interface en Applet Java si on veut évoluer notre application pour qu’elle s'exécute dans un site web qu’on peut consulter avec n’import quelle navigateur.

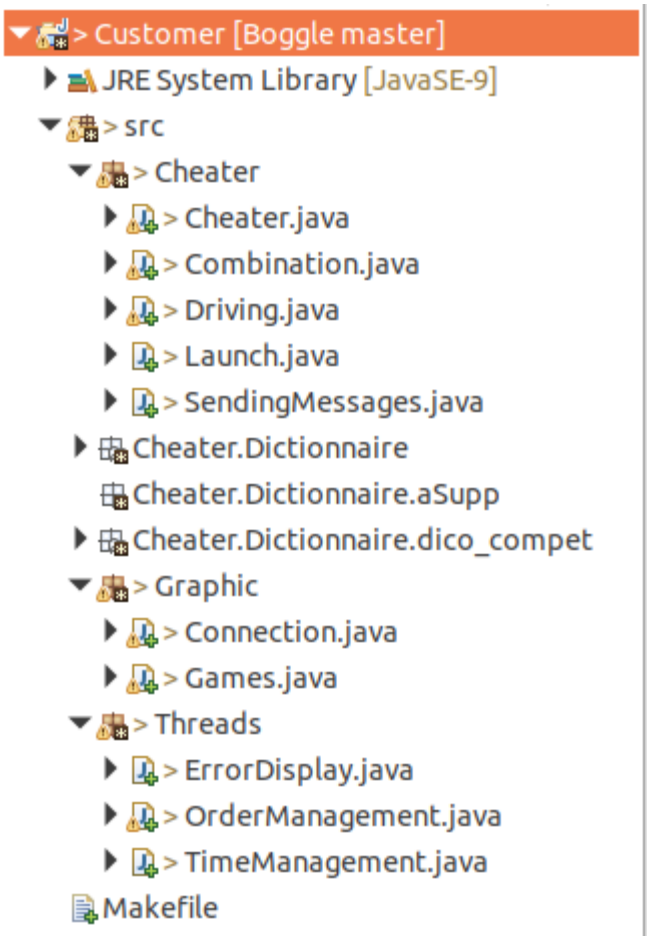
Pour nos besoin on a ajouter quelque commandes au protocole comme :

- ***TIME/temps/*** : Pour indiquer le temps qui reste pour la fin de tour courant.

- ***USERS/user1*user2*user3*.....*userN/*** : Qui indique les joueurs connectés déjà avant la connexion de joueur qui reçoit cette commande.

- ***ERREUR/userExist/*** : Indique au joueur que le login qu’il choisit ce n’est pas possible de l’utiliser parce que il a un autre joueur qu’il utilise déjà.

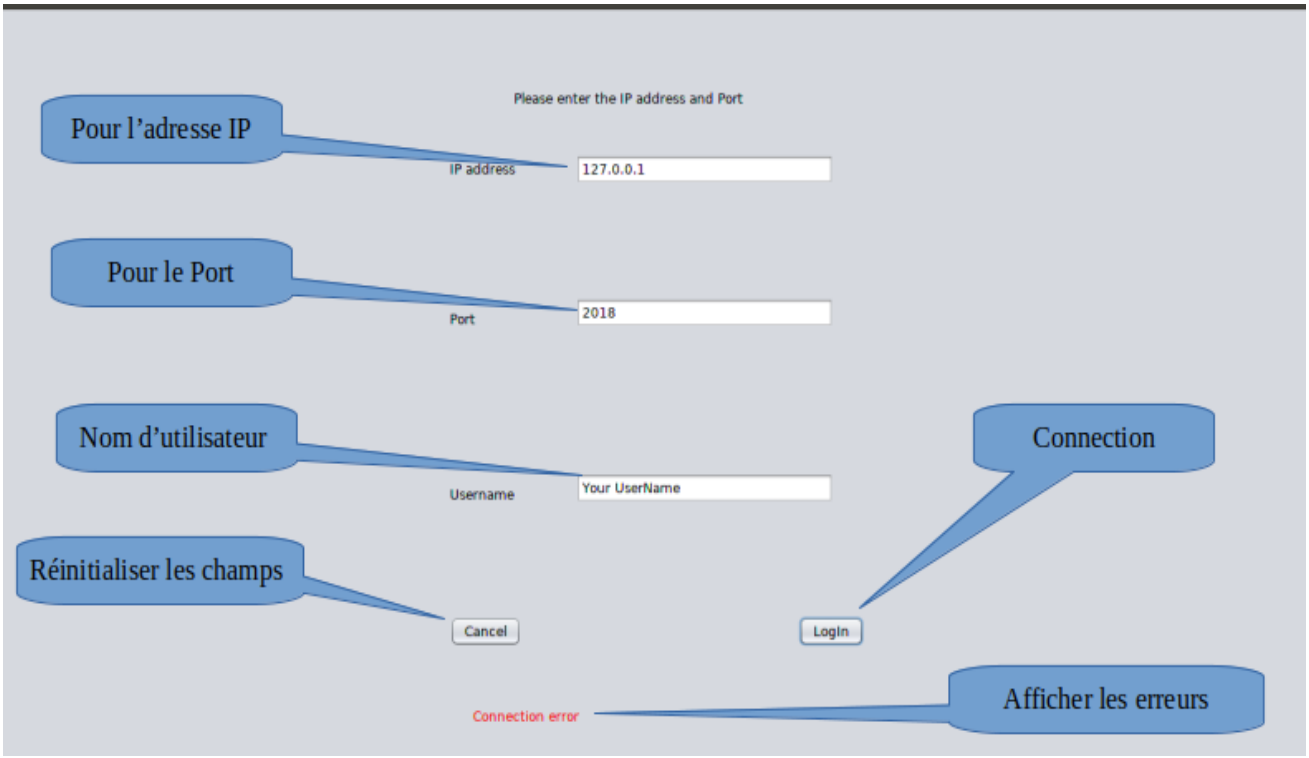
3.1.1 hiérarchie des packages et classes



Capture d'écran de la hiérarchie des classes

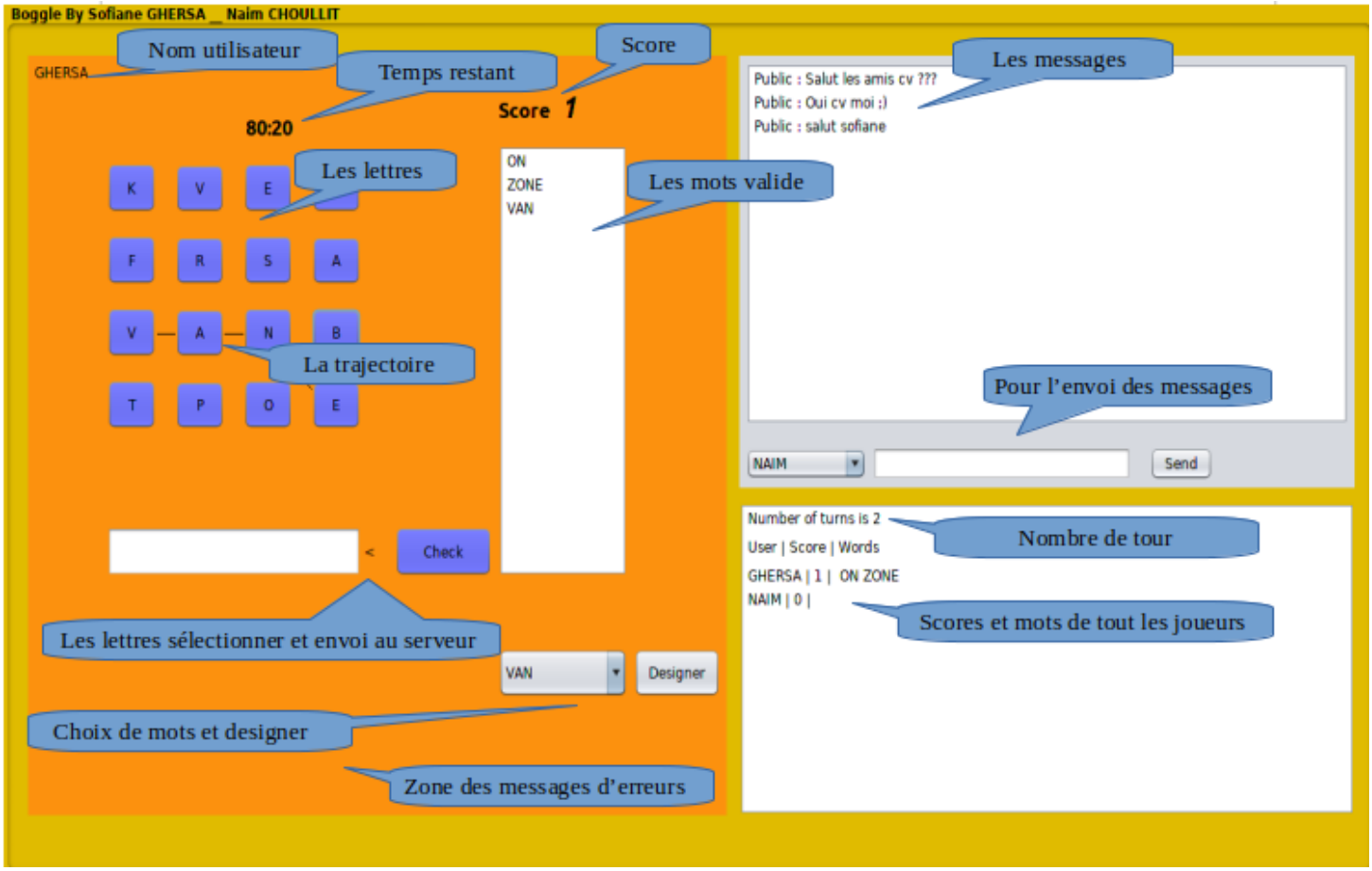
- **Package *Graphic*** *:* Il contient l'ensembles des classes qui extend de ***JFrame*** est qui represents les différentes interfaces graphiques, dans ce package on trouve :

    - ***Connection :*** Le main de coté client elle permet au utilisateurs d'entrer l'adresse IP, le Port de serveur et leurs nom d'utilisateur.



Capture d'écran de l'interface graphique de : Connection

- ***Games :*** Si l'utilisateur arrive à réussir la connexion avec le serveur la fenêtre *Connection* va disparaître et une nouvelle fenêtre s'affiche cette dernière est l'interface de *Games* qui contient l'ensembles des lettres, les scores des autres joueurs et les messages échangés.



Capture d'écran de l'interface graphique de : *Games*

- **Package *Threads*** *:* Il contient l'ensembles des classes qui extend de ***Thread*** et qui gères les différents partis de l'interface, on a choisi d'utiliser les threads pour implémenter le principe de programmation répartie qui permet l'exécution de plusieurs fonctions en même temps, dans ce package on trouve :

  - ***OrderManagement*** : C'est un thread qui est lancée directement au première connexion avec le serveur, il attends en boucle infini des commandes qui arrivent de serveur et il les traités ou il fait appel au fonctions de la class mère qui est l'interface ***Games***.

- ***ErrorDisplay*** : C’est un thread qui s’occupe d’afficher un message d’erreur sur l’interface **Connection** et ***Games*** et ces messages proviennent de serveur qui indique par exemple le serveur n’est pas accessible, un mot qui n’existe pas dans le dictionnaire, nouveaux joueur qui viens de connecter, la fin de tour et la fin de session.

- ***TimeManagement*** : C’est un thread qui s’occupe d’afficher le temps qui reste pour la fin de tour.

- ***Cheater :*** Il contient l’ensembles des classes qui va s'occuper de jouer le rôle d’un joueur tricheur et elle font semblant d'être un vrai client par l’envoi des messages aléatoire, dans ce package on trouve :

  - ***Launch :*** C’est une class mère qui contient la class Main.

  - ***Driving :*** C’est un thread qui tourne en boucle est qui attend des commandes arrivants de serveur on a besoin de traite que ces deux commandes : *TOUR/tirage/* et *CONNEXION/user/* .

  - ***Cheater :*** C’est une class qui initialise les variable et la connexion avec le serveur, elle arrête le thread ***Combination*** si c’est la fin de tour avant la fin de faire tout les combinaison possible et elle lance une nouvelle instance de ce thread avec une nouvelle grille reçu.

  - ***Combination :*** C’est un thread qui s’occupe de construire tous les combinaison possible a partir de la grille reçu, il tester l'existence de mot dans son dictionnaire avant de l'envoyer au serveur, on a pas traiter la trajectoire donc notre tricheur il envoie des mots qui ont une trajectoire faux c’est notre but, si on a un serveur qui détecte le comportement des robots ca sera encore plus difficile qu’il détecte notre tricheur à côté des messages envoyer aléatoirement.

  combinaison c’est la fonction qui s’occupe principalement de faire des combinaisons possible à partir de grille “ ensemble des lettres “ et nbLet le nombre de lettre de mot a formé à partir de cette grille, on appelle cette fonction avec la grille reçu et avec cette grille une deuxième fois mais on inverse les lettres “ abc devient cba “, on appelle cette fonction avec nbLet allant de 2 à 15.

```java
public void combinaisons(Vector<String> ListLettres,Vector<String> ListTirag, Vector<String> Grille,Vector<String> OrderTirage, int nbLet) {
    if(nbLet==0) {
        AddNewMotTrouve(ListLettres, ListTirag);
    }else {

        Vector<String> GrilleRest;
        Vector<String> GrilleRestTirag;
        Vector<String> ListLettresPlus;
        Vector<String> ListTiragPlus;

        for (int i = 0; i < Grille.size(); i++) {

            GrilleRest = new Vector<>();
            for (int j = i; j < Grille.size(); j++) {

                GrilleRest.add(Grille.get(j));
            }

            GrilleRestTirag = new Vector<>();
            for (int j = i; j < OrderTirage.size(); j++) {

                GrilleRestTirag.add(OrderTirage.get(j));
            }

            ListLettresPlus = new Vector<>();
            ListLettresPlus = (Vector<String>) ListLettres.clone();  ListLettresPlus.addElement(Grille.get(i));

            ListTiragPlus = new Vector<>();
            ListTiragPlus = (Vector<String>) ListTirag.clone();  ListTiragPlus.addElement(OrderTirage.get(i));

            combinaisons(ListLettresPlus, ListTiragPlus, GrilleRest, GrilleRestTirag, nbLet-1);

        }
    }
}
```

Capture d'écran de la méthode combinaison

- ***SendingMessages :*** C’est un thread qui envoie chaque 4 seconds un messages à tous les joueurs et il a une partie d'aléatoire, il choisi aléatoirement un message parmi ceux qui sont enregistrés à l’initialisation.

3.2. Serveur

Le serveur est intégralement codé en ocaml
Il est séparé en 5 modules principaux :

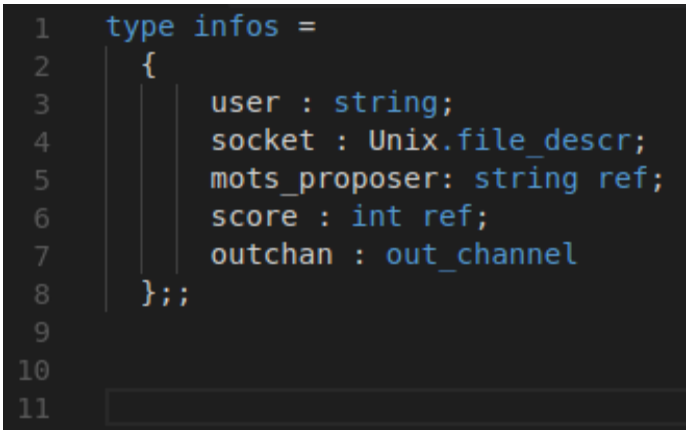
- ***Server_manager*** **:** qui a comme principale rôle d'accepter les connections des clients en créant un thread pour chacun.
- ***Connection_manager*** **:** pour traiter les différentes requêtes d’un client spécifique.
- ***Tour*** **:** pour gérer l'exécution des tours et la génération des tirages.

- ***Global_functions*** : qui contient différentes fonctions partagé entre les modules.

- ***Main*** **:** pour lancer le serveur.

- ***Global_functions*** **:**

Le serveur nécessite certaine ressource relative aux client ou à la sessions de jeu actuel pour fonctionner. Il utilise le type infos qui permet de stocker toute les informations relative au joueurs qui est connecté et les différentes fonctions pour lire dans le dictionnaire ou la conversion de certains type de données.



Capture d'écran de type infos

- ***user :*** Pseudo du joueur.
- ***socket / outchan :*** La socket et les flux d'entrée et de sortie utilisé afin d'envoyer des requêtes aux clients.
- ***mots_proposer :*** l'ensemble des mots valides proposer par le client.
- ***score :*** Le score de ce joueur dans la session actuel.

- ***Server_manager*** **:**

Le module serveur à pour rôle de gérer la connexion des clients et de les synchroniser avec les tours de jeu en cour. Il crée un thread chaque nouvelle connection d'un client.

- ***Connexion_manager* :**

Le module principale de la parti serveur, il s'occupe des différents requêtes des clients.

A chaque nouvelle connection d'un client, on stock ces informations et on l'ajoute dans une liste ***clients*** géré par le serveur qui est protégé par un verrou (**mutex**) afin d'éviter qu'elle ne devienne corrompu par un accès simultanés de plusieurs Thread.

Envoi des requêtes aux clients :

Le serveur dispose d'un certain nombre de fonctions lui permettant de générer automatiquement les string des requêtes à envoyer aux clients. La majoritées de ces fonctions sont juste des concaténations de string mais quelques une d'entre elles font appelle au module Tour afin de vérifier les mots proposés par le client, ou l'ajouter dans la liste des mots valides proposés par ce dernier, cette liste est **motsValides** qui protéger avec un **verrou**. avant d'envoyer une réponse

Nous envoyons aussi des requêtes d'erreur aux clients lorsque leurs requêtes ne respectent pas le bon format.

Traitement des requêtes client :

Le serveur dispose d'une fonction de traitement pour chacune des requêtes qui sont envoyer par un client.

- ***treat_request* message :** Elle fait appelle a une fonction qui correspond aux type de requête de client.

- ***connect client* :** Si le client est déjà connecté, on lui envoie un message d'erreur, sinon on récupère ces information et l'ajouter dans la liste des clients connectés, puis on lui envoie le tirage de tour courant avec les scores qui correspond à chaque joueur connecté et le temps restant pour la fin de tour courant.
  Dans le cas ou c'est le premier client connecté, le serveur lance une nouvelle session grâce à un signal envoyé par ce client (Condition.signal)

- ***start_session* :** Pour indiquer au premier client connecté le commencement d'une nouvelle session

- ***signal_connexion client*** **:** Signale la connection d'un joueur aux autres joueurs.
- ***deconnect user*** **:** après avoir reçu une requête de déconnection d'un jour, on ferme sa socket et on signal sa déconnection aux autres joueurs.

- ***trouve mot trajectoire*** **:** c'est la méthode principale de jeu, elle se compose de plusieurs tests :

    - vérification de la trajectoire
    - vérifier que le mot proposé existe dans la langue française
    - vérifier que le mot proposé n'a jamais été déja proposer

  dans le cas ou ces tests sont vérifiés, on ajoute ce mot dans les liste des mots de ce jours, dans la liste des mots valides et on mit à jour son score selon la longueur de mot.

  dans l'autre cas, on envoie un message d'erreur qui dépend de test qui n'a pas été vérifier

- ***chat*** **:**

    - ***broadcast_message msg*** **:** envoie d'un message à tous les joueurs connectés.

    - ***send_message_to msg user*** : envoie d'un message à un joueur spécifique.

Module Tour :

la séquence de tour se lance à chaque premier joueur connecté, à partir de deuxième tour l'envoie des tirage se fait automatiquement pour tous les joueur connecté, un tour différent de premier tour se lance à chaque terminaison de tour qui est lui précède.

- **start_tour num :** pour le lancement d'un nouveau tour, il génère un nouveau tirage puis il crée un thread qui traitera la terminaison de ce dernier.

- **expiration clients :** a chaque fin d'un tour, le bilan de tour sera envoyé à tous les joueurs, il sauvegarde les données de derniers dans le journal, puis il lance un nouveau tour avec un temps de terminaison.

4. Expérimentation

Pour tester notre travaille vous pouvez récupérer la source de projet sur GITHUB le liens est : https://github.com/GHERSASofiane/Boggle .

En premier suivez les étapes décrit dans la section 3.2. Serveur pour lancer le serveur et la section 3.1 Client pour lancer aussi l'interface graphique correspond au client.

5. Amélioration :

Vu qu'on a utilisé une architecture 2-tiers, dans le cas d'une panne de serveur, les différentes données des sessions précédentes seront perdu, l'amélioration qu'on peut proposer est de changer notre architecture a 3-tiers en sauvegardant nos données d'une base de données par exemple.

6. Extensions :

On a pu implémenté l'intégralité de extensions obligatoires demandées dans ce projet.

- **Extensions facultative :**
  - **Journal**
  - **Chat**
  - **Client autonome**
  - **Client graphique**
  - **Persistance :** à l'exception de pouvoir rejoindre précédemment quitter

7. Conclusion

Dans le cadre de notre PC2R nous avons pu développer jeu de lettre multijoueurs qui permet à plusieurs joueurs de participer.

Nous avons pu mettre en corrélation les aspects théoriques et pratiques, sans éclipser les difficultés que nous avons rencontrées dans ce projet, c'est-à-dire les échanges entre un serveur et plusieurs clients, la gestion des threads et la programmation reparti.

Au terme de ce projet, notre sentiment est très positif ; nous avons effectivement acquis de nombreuses connaissances profondes de programmation concurrent et reparti. Nous avons acquis des compétences nouvelles également la programmation fonctionnelle d'une part en OCaml et un langage multi-paradigme typé dynamiquement d'autre part, nous avons pu développer l'esprit d'équipe et la gestion de projets grâce à GitHub, le service web d'hébergement et de gestion de développement de logiciels.