Boogle Master Project

MASTER STL – PC2R

2017-2018

Rapport

William Sergeant

Vincent Havinh

Index

Client

Interface

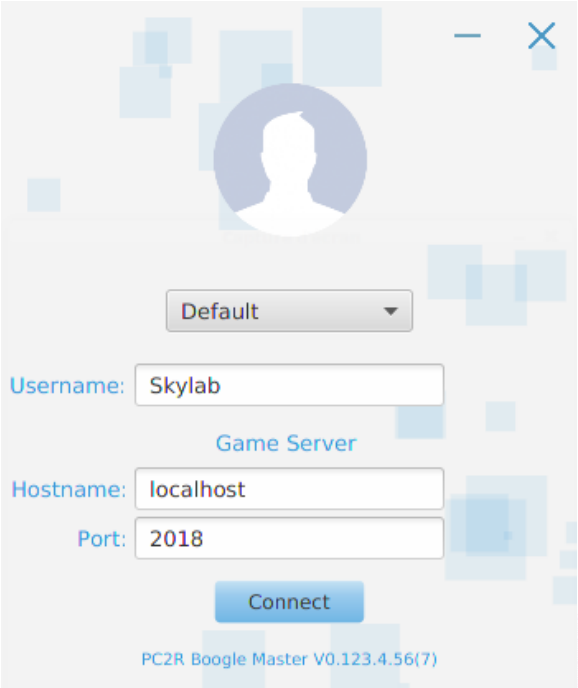
L'interface est réalisée en **Java**. Plus précisément on utilise la librairie **JavaFX** ainsi que des ressources externes (images, sons, layouts) pour les vues avec **FXML** qui permet l'implémentation de scènes et de feuilles **CSS** afin de définir des styles pour les groupes d'éléments.

Java et **JavaFX** furent choisi car nous disposons d'un socle de connaissances quant à leur utilisation ce qui nous a permis d'être plus efficaces et innovants lors du développement du client. L'introduction de l'utilisation de documents **FXML** et **CSS** en adéquation avec le code fonctionnel est cependant une nouveauté pour nous. Cela nous a permis de mieux utiliser les composants graphiques de **JavaFX** ainsi que de faciliter les interactions entres les vues et les contrôleurs tout en élargissant notre cercle de compétences. Nous utilisons donc un framework simili MVC, le Modèle et la Vue étant sensiblement entrelacés.

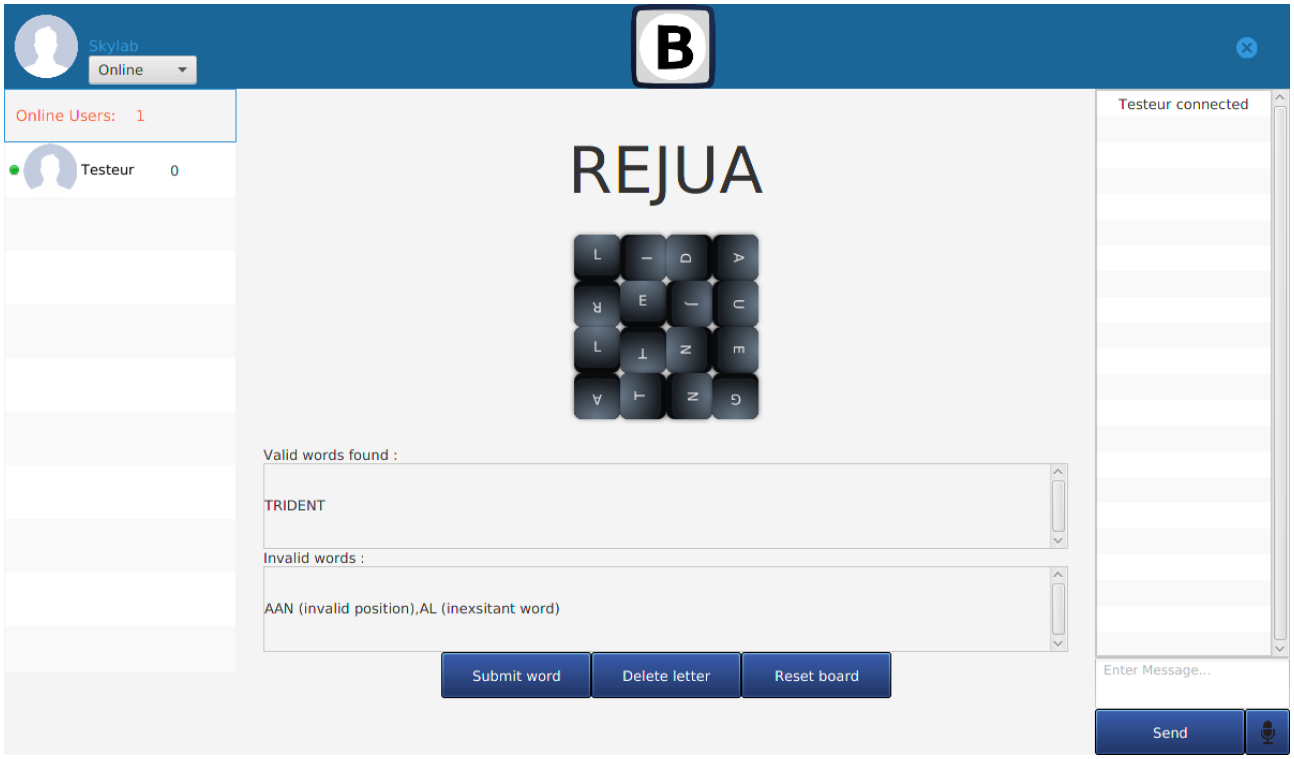
Vues

Deux vues principales ont été créées lors de la réalisation de ce projet :

- Une vue pour l'écran d'identification des joueurs, permettant de renseigner l'adresse et le port de connexion au serveur de jeu, ainsi que le pseudonyme du client lors de sa session de jeu.

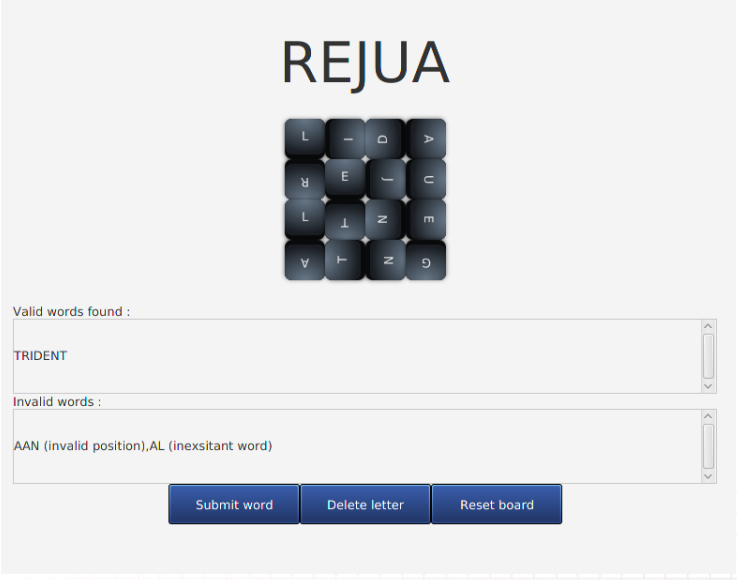


- Une vue contenant la liste des joueurs présent ainsi que leurs scores, un panneau latéral comprenant un espace de chat entre les joueurs et enfin une partie centrale représentant le plateau de jeu.

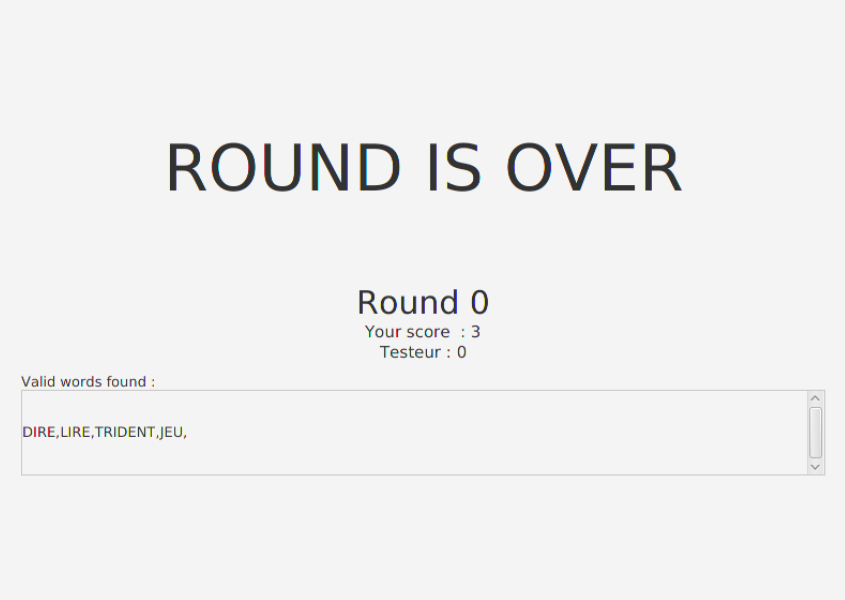


Cette partie centrale peut elle-même afficher plus vues :

Le plateau de jeu avec la matrice de dés, la liste des mots validés et invalidés ainsi que le mot en cours de confection par le joueur. De plus l'utilisateur dispose de trois boutons afin de pouvoir effacer la dernière lettre sélectionnée, de réinitialiser le mot complet et enfin de soumettre sa proposition au serveur.

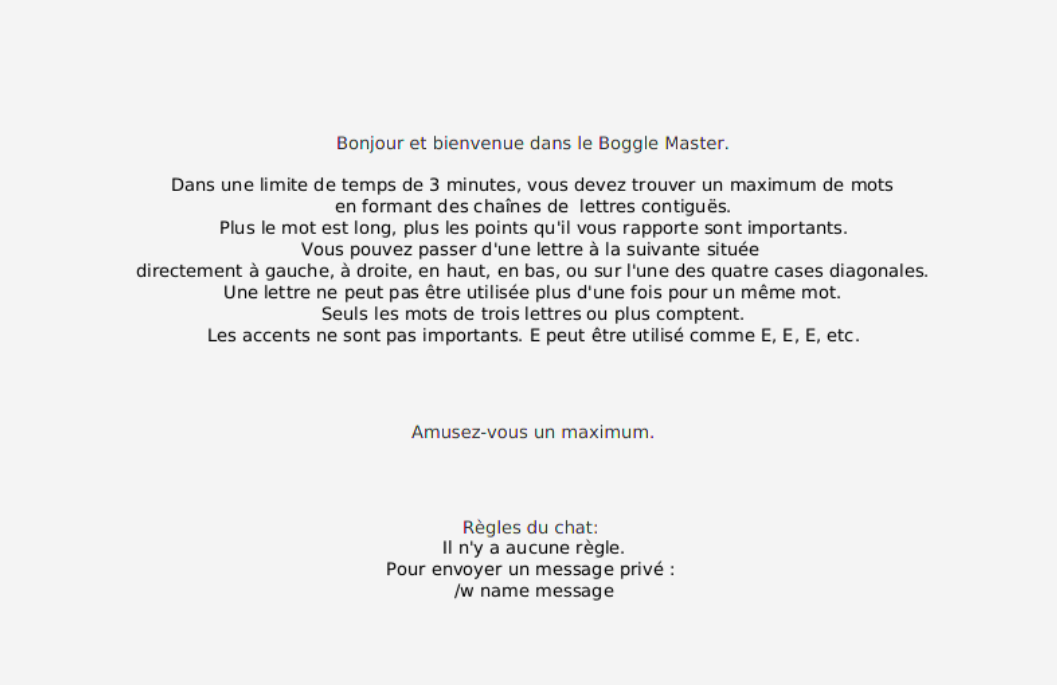


Le panneau de résultat du tour écoulé avec le nombre de points accumulés par chaque joueur avec la liste des mots validés par le serveur pour chaque client.



La vue de fin de session, avec les résultats finaux pour chaque joueur et le titre de vainqueur.

La vue de début de session, dans l’attente de commencement du premier tour.



Note : il a été décidé de ne pas implémenter d’horloge ni de chronomètre pour le tour courant afin de laisser les joueurs libres d’esprit et s’épargner le stress de voir les secondes diminuer.

Liaisons

Les interactions entre l’interface et son contrôleur se font majoritairement pas le biais de boutons, lesquels se sont vu attribuer une fonction spécifique lors de leur appel.

En outre au lieu de devoir créer, instancier, personnaliser le bouton et par la suite lui affecter un écouteur d’événement définissant le comportement à avoir, en cas de clique du bouton par exemple, celui-ci est simplement défini par une ligne dans le fichier FXML correspondant :

<Button fx:id="**buttonSendWord**" onAction="**#sendWordAction**"
text="Submit word" id="**rich-blue**"/>

Le champ ***fx:id*** est l'identité du button dans le fichier FXML, celui-ci peut être utiliser par le contrôleur java associé. L'instruction dans le code du contrôleur **@FXML Button buttonSendWord** permet de récupérer l'instance de ce bouton (après le chargement du fichier FXML par le contrôleur, moment où les instanciations des composants sont effectuées). Ainsi celui-ci sera directement utilisable et référera au même composant dans l'interface et dans le code.

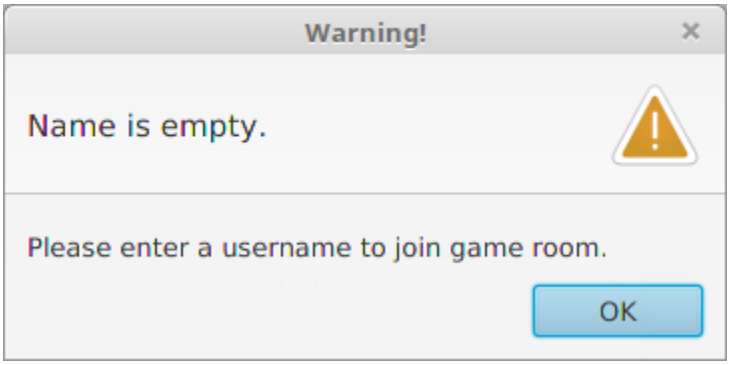
L'autre champ ***onAction*** défini la fonction associée en cas d'activation du contrôleur. Ici lors du clique sur le bouton la fonction **sendWordAction** sera éxécutée. Ces fonctions fonctionnant tels les event Listeners et Handlers, il est possible de passer un argument de type **Event**. Autrement il est possible de paramétrer un champ user-data dans le composants FXML et de récupérer cette valeur lors de l'appel. Par exemple lors de l'appel de la fonction **addLetter** (appelée lors du clique sur une lettre de la matrice de jeu) le champ user-data contient la position du bouton (par exemple **user-data= « A1 »**). En récupérant également le champs de texte du bouton (**« L »**) on ajoute au mot en cours **la lettre L et la trajectoire A1**.

Enfin le champs ***id*** est une référence interne au composant, elle servira notamment à définir les références aux fichiers **CSS** afin d'appliquer les styles appropriés aux composants de manière automatique.

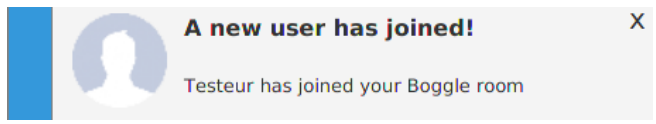
Fonctionnalités

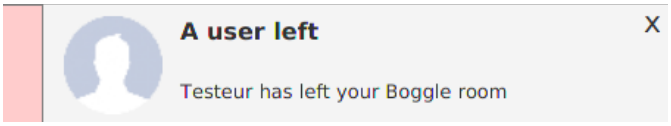
Afin de mettre en place un confort d'utilisation pour le joueur, a été mis en place divers améliorations :

La plupart des exceptions sont gérer avec l'appel d'une fenêtre modale explicitant les raisons de l'erreur au joueur : Erreur de connexion au serveur, connexion perdue ou autre action de jeu prohibée.



Une notification visuelle et sonore en bas à droite de l'écran afin de prévenir de la connexion d'un nouveau joueur ou du départ de celui-ci. Nous aurions également pu ajouter une notification pour le début d'une session / tour de jeu.

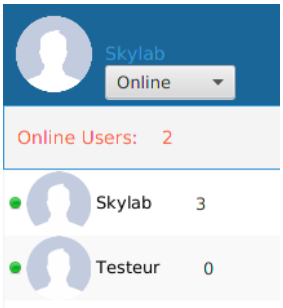




D'autres améliorations visuelles sont présentes mais non fonctionnelles, l'implémentation n'ayant pas pu être portée à son terme par manque de temps de développement :

- L'affichage d'une petite bulle de couleur verte / orange / rouge à côté du pseudonyme de chaque joueur définissant son statut : En Ligne / Occupé / Absent .

- L’utilisation d’images de profil personnalisées pour chaque joueur, lui permettant une identité visuelle unique dans la liste des joueurs et le chat.



Contrôleurs

Il existe un contrôleur associé à chaque vue du programme. Ainsi la vue **LoginView**.fxml met en place les composants, **LoginStyle**.css permet d’appliquer les styles et **LoginController**.java les interactions entre le code de traitement des données et l’interface.

De même pour **ChatView**.fxml, **ChatStyle**.css et **ChatController**.java.

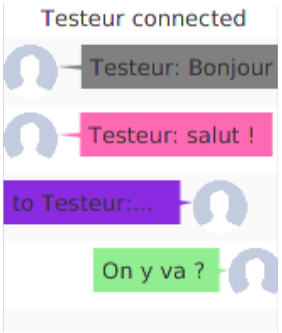
Le **ChatController** contient les fonctions de gestion de l’affichage et des informations du jeu.

Celui-ci gère les comportements internes tels que la sélection des lettres, l’ajout de message au chat et le basculement entre les écrans de session / plateau de jeu / résultats du tour.

De plus celui-ci communique avec le serveur de jeu en envoyant des messages défini selon le protocole.

Chat

Une fenêtre de chat est présente dans un panneau latéral à gauche. Celle-ci utilise une classe tiers (Bubble) qui fourni un affichage plus agréable pour les messages. Les messages publics envoyés par le joueur s'affichent en vert (en violet pour les privés), les messages public reçu sont en blanc (en rose pour les privés). De plus les messages du serveur sont en gris.



L'envoi de message privés se fait selon le système du « ***whispering*** » :

afin de contacter directement un joueur, John, pour lui dire '*bonjour*' on tapera la ligne suivante dans le chat :

```
/w John bonjour
```

Uniquement le destinataire pourra lire le message.

Une implémentation supplémentaire mais non achevée est l'envoi de messages vocaux, ceux-ci seraient enregistrés par le client, convertis en bytecode puis envoyés au serveur sous forme de chaîne. Ils seront recomposés par le client cible et le message vocal sera restituer dans sa forme originale.

Ceux-ci sont malheureusement non testables pour le moment.

Threads

Ce contrôleur communique efficacement avec l'interface afin de mettre à jour celle-ci avec les informations correspondantes. Afin de passer la main au thread JavaFX pour changer les données de l'écran on utilisera la méthode **Platform.runlater(()→ {runnable*}) ;***

Cela nous permettra d'effectuer des modifications mineur de l'affichage tel que le changement des scores lorsque le thread sera disponible.

Pour des traitement plus complexes tels que l'ajout d'un nouveau message à l'interface, nous utiliseront un système différents : les **Tasks.** Celles-ci implémentent l'interface Worker, héritage de Swing, permettant de faire des calculs plus lourds en parallèle et de prévenir l'interface du changement alors disponible, évitant ainsi le freeze de l'application.

1 Serveur

1.1 Titre 2

1.1.1 Titre 3