Simon Fauconnier
Steve Henriquet

Scrabble en java

Table des matières

Scrabble en java

Cahier des charges :

Descriptif du projet :

Le produit fini sera un jeu de Scrabble multijoueur (2 joueurs maximum).

Règle :

Le jeu se déroule en tour par tour et pour désigner le premier joueur, chaque joueur tire une lettre, celui possédant la plus proche du « A » tire en premier les 7 lettres et commence. Le but est de former des mots avec ses lettres pour marquer un certain nombre de points (cf. plus loin).

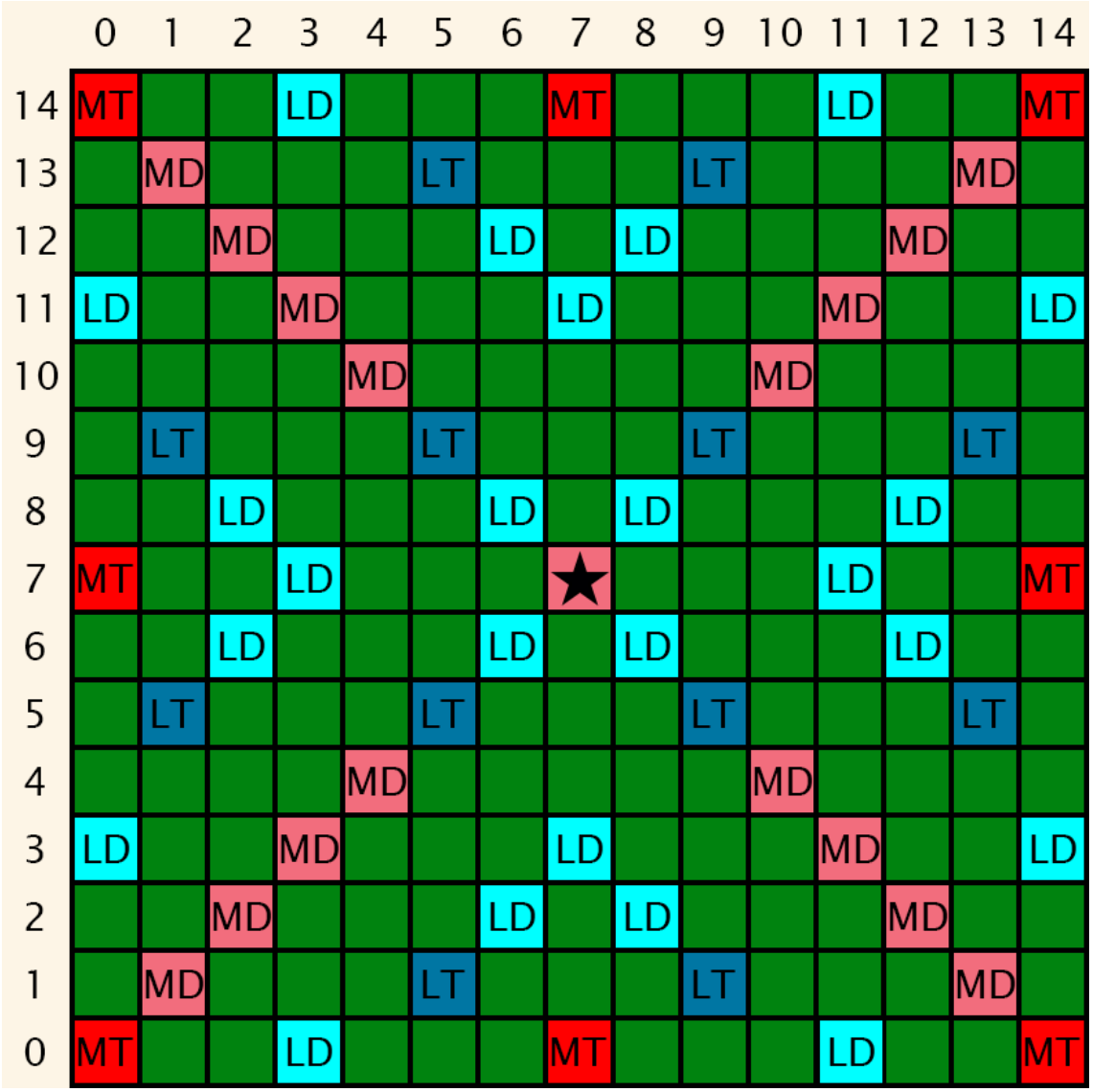
Les points sont calculés en fonction de la valeur de chaque lettre et de possible bonus dispersé sur le plateau. Le jeu se termine lorsqu'un joueur n'a plus de lettre ou lorsqu'il lui est impossible de former un mot.

Implémentation :

Le jeu sera composé de :

- Un plateau de jeu (15x15) avec certaines cases bonus dispersées sur ce dernier suivant le pattern suivant :

  Le premier mot doit être posé sur l'étoile au centre du plateau.



- Les joueurs commenceront avec 7 lettres aléatoirement choisies dans la sélection suivante :
  - *0 point* : **Joker** × 2
  - *1 point* : **E** ×15, **A** ×9, **I** ×8, **N** ×6, **O** ×6, **R** ×6, **S** ×6, **T** ×6, **U** ×6, **L** ×5
  - *2 points* : **D** ×3, **M** × 3, **G** ×2
  - *3 points* : **B** × 2, **C** × 2, **P** × 2
  - *4 points* : **F** ×2, **H** ×2, **V** ×2
  - *8 points* : **J** × 1, **Q** × 1
  - *10 points* : **K** ×1, **W** ×1, **X** ×1, **Y** ×1, **Z** ×1

  Lorsqu'une lettre est piochée, son nombre d'instances dans « le sac » est décrémenté.

  Chaque lettre correspond à un certain nombre de points (voir liste ci-dessus).

  Lors de son tour, un joueur peut, soit poser un mot sur le plateau (de haut en bas et de gauche à droite), soit remettre l'entièreté ou une partie lettres dans « le sac », soit passer son tour.

  Lorsque le joueur réussit à poser toutes ses lettres, il réalise un « Scrabble » et marque immédiatement 50 points bonus.

  Les points sont comptés en fonction de la valeur des lettres (voir liste ci-dessus) et, éventuellement des bonus du plateau. Un bonus est activé si le mot est posé sur ce dernier. Un mot peut être posé sur plusieurs bonus, les effets sont cumulables mais les bonus ne peuvent être utilisés qu'une seule fois.

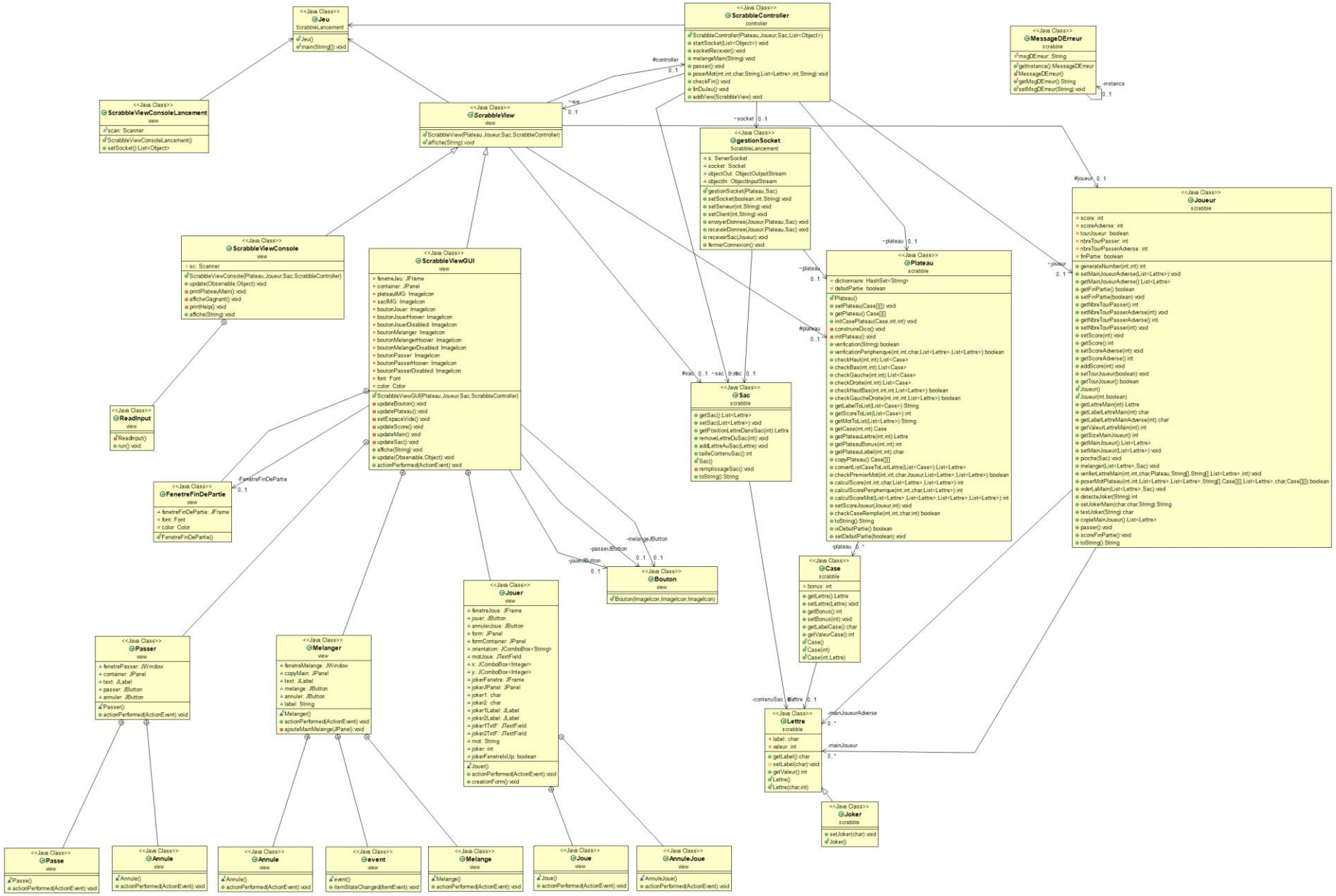
Un mot doit utiliser une lettre d'un mot déjà présent sur le plateau ou le complétant, cette lettre est comptée dans les points.

Si un mot en colle un autre, les points de tous les nouveaux mots sont comptés.

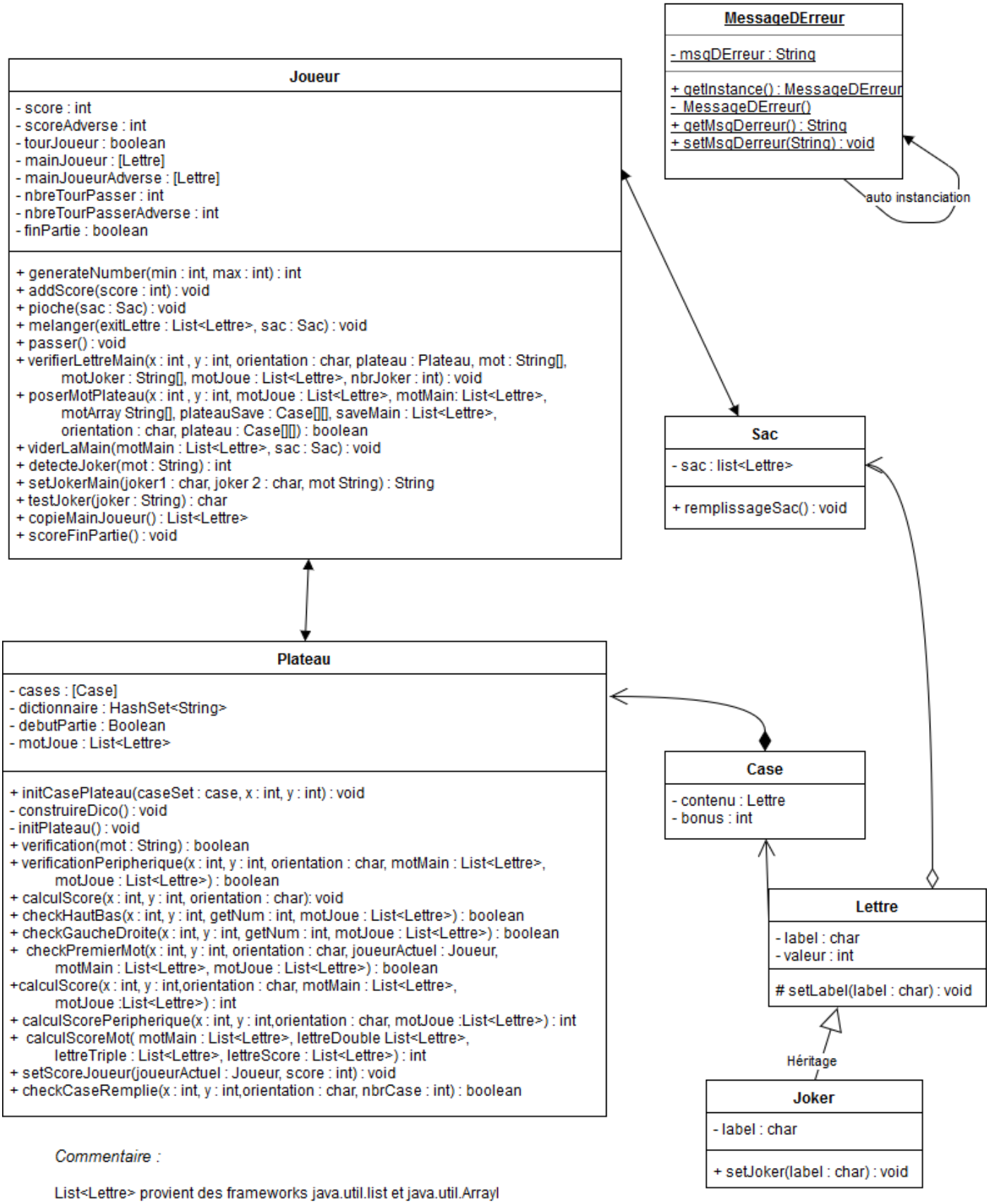
Les mots sont vérifiée grâce à un dictionnaire, les noms propres, les sigles et abréviations, les expressions composées et les symboles chimiques sont invalides à contrario, les verbes conjugués et les onomatopées sont autorisés. Les nouveaux mots formés au placement d'un mot sont aussi vérifiés.

Diagramme UML final:

Global :



Modèle :



L'UML Global étant très grand vous pouvez le retrouver ici ou dans le PDF (inclus dans le dossier contenant ce rapport) pour avoir une vision plus claire de ce qu'il contient.

Choix d'implémentation :

Nous avons dû prendre plusieurs choix durant la conception du projet :

Le plateau de jeu est une matrice à double entrée, où chaque case contient un objet Case. Cet objet contient, lui, un éventuel objet Lettre.

Nous avons utilisé des fichiers .XML pour initialiser le plateau et le sac. Le projet utilise donc deux de ces fichiers : dataCase et dataLettre, qui permettent de créer respectivement tous les objets Case et tous les objets Lettre nécessaires. Ces fichiers sont lu lors de l'initialisation, dans le constructeur du sac et du plateau. Nous avons choisi le xml car c'est un format léger et facile d'utilisation avec Java.

Nous avons utilisé des ArrayLists pour la main du joueur et le sac de lettres car les méthodes disponibles avec les listes étaient utiles pour gérer les mouvements de lettre de l'un vers l'autre (« .add() », « .remove() », « .removeAll() »).

Le dictionnaire se trouvant dans la classe Plateau est une HashSet qui charge à l'initialisation un fichier .txt de ± 400 000 mots. Nous avons pris une HashSet pour supprimer instantanément les éventuels doublons. La méthode «.contains() » nous a évidemment été très utile pour la recherche de mot.

Nous avons choisi d'utiliser un Singleton pour le message d'erreur car contrairement aux classes abstraites, les variables du singleton peuvent changer d'état ce qui dans notre cas était exactement ce dont on avait besoin. Le message d'erreur s'initialise suivant des cas spécifiques dans des méthodes parfois comprises dans d'autres méthodes. Ces méthodes ne pouvaient pas se retrouver dans le contrôleur ou être appelées par lui car cela aurait manqué de cohérence. Dès lors, il nous est paru évident, pour plus de légèreté et de compréhension du programme, de ne pas retourner le message d'erreur à travers toutes ces méthodes mais plutôt de le placer dans le Singleton pour pouvoir le récupérer dans les affichages.

Pour l'utilisation des sockets, nous avons dû implémenter la sérialisation afin que les classes Lettre, Case et Joker puissent être envoyée avec le plateau.

La classe ScrabbleViewConsoleLancement permet de demander à l'utilisateur les données nécessaires à l'initialisation des sockets. Cette classe sort légèrement du pattern MVC, mais c'était plus agréable de pouvoir lancer l'application sans devoir définir les paramètres avant le lancement.

Difficultés rencontrées :

Modèle :

Plateau :

- Vérification des mots adjacents nouvellement créés
- Calcul du score et bonne gestion des bonus

Résolution des problèmes :

Une des grosses interrogations au début du programme était de trouver un moyen de vérifier les mots posés ainsi que les nouveaux mots formés.

Nous avons trouvé la solution ensemble, en écrivant des algorithmes de vérification en pseudocode. Ensuite, en se basant sur ces algorithmes et après de nombreux essais, la vérification était implémentée. Elle sera modifiée encore de nombreuses fois, au fil des bugs rencontrés grâce aux tests et aux JUnits.

Nous avons suivi le même mode opératoire pour l'élaboration du calcul de score ; en se basant sur des méthodes écrites en pseudocode.

Interface graphique :

- Tentative d'implémentation du drag'n drop finalement abandonné après avoir réussi à rendre les pièces déplaçables.

Résolution des problèmes :

Le problème était que n'ayant pas assez de connaissances en swing les méthodes utilisées n'ont pas été écrites par nous et malgré la compréhension des méthodes et le déplacement possible des éléments graphiques, nous n'arrivions pas à empêcher les pièces de se superposer ce qui avait pour conséquence de détruire l'image de la lettre.

Nous avons donc décidé d'abandonner cette possibilité, le projet n'étant pas encore fini. Nous trouvions qu'il était préférable de faire une interface graphique certes moins interactive mais fonctionnelle plutôt que d'essayer, pour finalement se retrouver à court de temps et ne pas arriver à finir le projet.

Les seules choses qu'il restait à faire pour rendre cette drag'n drop fonctionnel étaient d'empêcher les lettres de se superposer et d'arriver à lire le layout, sans doute de manière similaire que la méthode utilisée pour lire le plateau, pour récupérer les modifications une fois que l'on appuyait sur le bouton « Jouer ».

Compilation :

- Inclure toutes les ressources dans le .jar (.txt, .xml)

Résolution des problèmes :

Quand on compile le projet, les ressources externes ne sont pas inclues dans le .jar. Il est évident que l'application ne peut se lancer sans ces ressources (nécessaire lors de l'initialisation).

Le fichiers .jar doit donc se trouver dans le même dossier que les ressources pour pouvoir les appeler.

Améliorations éventuelles :

Global :

- Permettre de jouer à trois et quatre joueurs.
- Permettre de jouer dans d'autres langues.
- Permettre une sauvegarde du jeu pour pouvoir reprendre la partie plus tard.
- Implémenter le ScrabbleViewConsoleLancement et le ScrabbleViewGUILancement dans le pattern MVC pour avoir une cohérence globale par rapport à ce dernier mais par manque de temps, il n'a pas été possible de l'implémenter.

Interface console :

- Ajout de fonctionnalité pour permettre de relancer directement une partie à la fin d'une, d'abandonner la partie et de pouvoir sauvegarder la partie.

Interface graphique :

- Implémenter le Drag'n Drop pour rendre l'interface graphique plus agréable à l'utilisation.
- Ajout d'un menu pour pouvoir relancer directement une partie lorsque l'on en termine une, abandonner la partie et permettre la sauvegarde de la partie.
- Utilisation de ScrabbleViewGUILancement lors du démarrage du programme pour permettre l'utilisation du programme de manière complète aussi bien en interface graphique qu'en console

Conclusion personnelle :

Simon Fauconnier :

Personnellement, j'ai beaucoup appris grâce à ce projet. Nous avons réussi à résoudre de nombreuse interrogation grâce aux nombreux tutoriels disponibles sur le Net (ArrayList, Socket…)

Le fait de recréer un jeu physique déjà existant nous a imposé des contraintes, en effet nous devions rendre un jeu de scrabble fonctionnel, avec toutes les règles officielles implémentées. C'était donc un défi d'arriver à faire fonctionner l'application avec toutes les fonctionnalités décrites dans le cahier des charges.

Nous avons délégué le travail dès le début du projet. GitHub s'est révélé très pratique pour suivre l'évolution du projet en restant attentif à la progression de l'autre. Nous avons beaucoup discuté ensemble des problèmes rencontrés afin des trouvés des solutions adaptées.

Le projet m'a également permis de travailler Java tous les jours, ce qui est une bonne chose lorsque l'on apprend un langage. C'est dans de tel projet que l'on se rend compte de l'importance de la JavaDoc (pour comprendre une méthode que l'autre a implémenté) et des test unitaires (qui m'ont permis de découvrir de nombreux bugs de vérification de mot).

Steve Henriquet :

Ce projet a été très enrichissant car il nous a permis de véritablement pratiquer ce que l'on nous enseigne, et ce comme dans une situation réelle avec un cahier des charges et des échéances à respecter, une gestion de groupe pour une bonne répartition du travail, l'apprentissage via différents tutoriels disponibles en ligne pour atteindre les objectifs ainsi que pour compléter ce qui a été vu en cours et d'apprendre à se débrouiller par soi-même en cas d'erreur en allant chercher les réponses ou les pistes de réponses en ligne et non via le professeur uniquement, contrairement aux séances de TP.