Rapport Projet More or Less

CHEVALIER Simon
LACROIX Florent

Introduction :

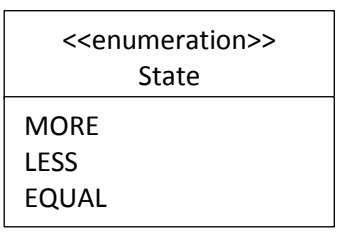
Ce projet a pour but de proposer à l'utilisateur de jouer au fameux jeu du **Plus ou Moins**. Le principe est simple : l'utilisateur entre un intervalle de 0 à un chiffre compris entre 1 et 10000000. L'ordinateur choisit ensuite un chiffre aléatoirement dans cet intervalle, et l'utilisateur doit le deviner en un minimum de coups, sachant que l'ordinateur lui fournit uniquement comme information si le chiffre entré est supérieur ou inférieur au chiffre à deviner.

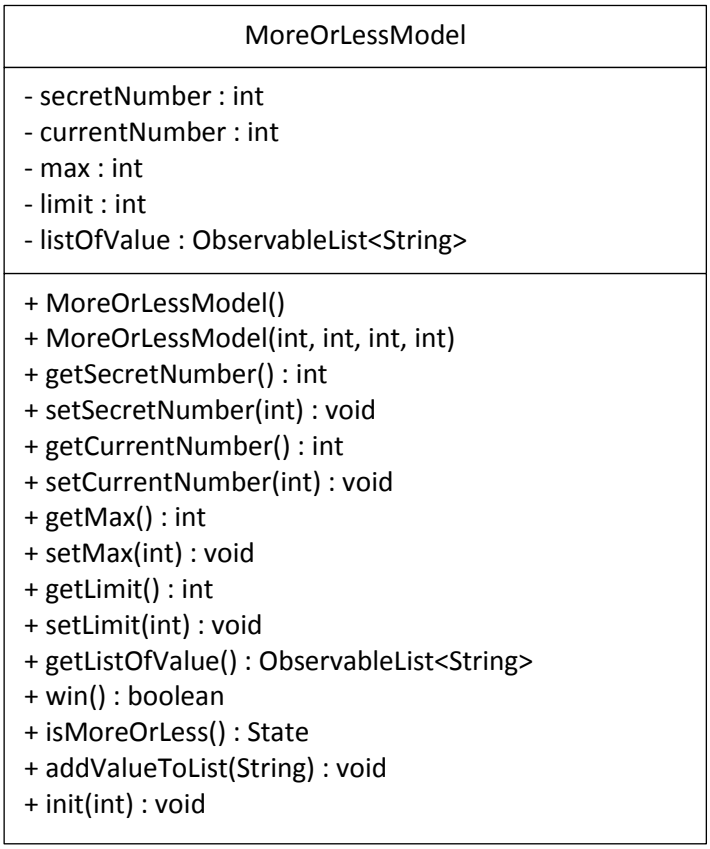
Dans notre jeu, l'utilisateur peut également choisir de voir ou non les valeurs qu'il a déjà testées.

Ce projet a été développé en JavaFX 2 dans l'objectif de fournir un jeu au projet **GameLoader**.

Ce que nous avons fait :

Nous avons dans ce projet 3 classes : *MoreOrLess*, *MoreOrLessModel* et *TransitionFactory*. *MoreOrLessModel* est le modèle du jeu, qui détermine le nombre secret dans l'intervalle et fournit diverses méthodes afin de savoir si le nombre que l'utilisateur entre est supérieur ou inférieur au nombre généré aléatoirement. Il possède 5 données membres privées : secretNumber, qui est le nombre que l'utilisateur doit deviner ; currentNumber, qui est le dernier nombre que l'utilisateur vient d'entrer ; max, qui est la limite haute de l'intervalle fixé par l'utilisateur (donc le secretNumber est fixé entre 0 et max) ; limit, qui est la valeur maximum de la variable max ; et enfin listOfValue, qui est une *ObservableList* de *String*, permettant de connaître l'ensemble des valeurs entrées par l'utilisateurs au cours de la partie. Nous avons également créé une énumération *State*, qui comprend 3 valeurs : **MORE**, **LESS** et **EQUAL**, que l'on utilise notamment dans la méthode isMoreorLess(). Ci-dessus le diagramme UML de l'énumération *State*, ainsi que le diagramme de classe de *MoreOrLessModel*.



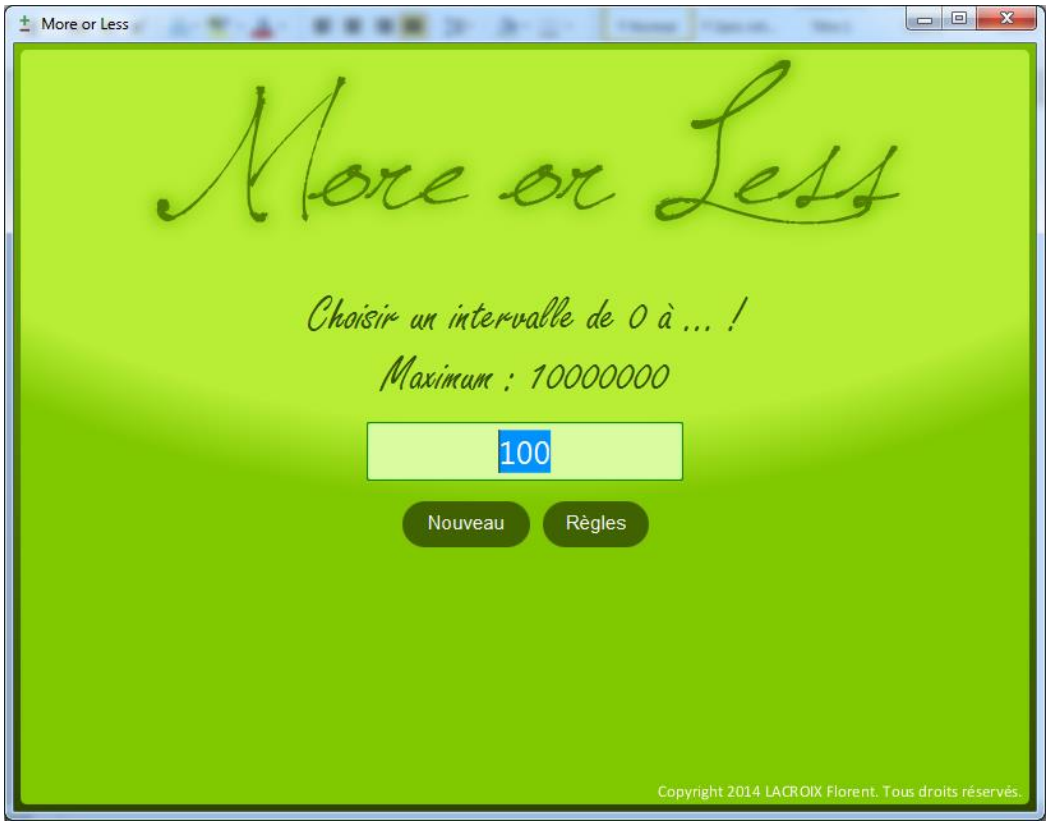


La liste observable permet de simplifier l'affichage des données qu'elle contient. En effet, à chaque fois qu'une action est menée sur la liste, celle-ci met à jour sa vue, ce qui permet de l'afficher facilement dans notre interface : à chaque fois que l'utilisateur entrera un chiffre, il sera ajouté à la liste, la vue de cette liste sera automatiquement mise à jour.

La classe *TransitionFactory* permet d'instancier des transitions, qui nous serons utiles plus tard dans le code. On lui fournit les paramètres adéquats, et elle nous retourne une *TranslateTransition* ou une *FadeTransition*. Aucune *TranslateTransition* n'est utilisée dans ce programme.

La classe *MoreOrLess* s'occupe quant à elle de la création de l'interface graphique et de l'interaction avec l'utilisateur (c'est-à-dire la gestion des différents boutons et de la case à cocher).

Une fois l'application compilée, voilà ce que nous avons :



En cliquant sur 'Nouveau', nous pouvons jouer au jeu (la validation d'un nombre se fait à l'aide la touche Entrée) :

Si on coche la *CheckBox*, on affiche la liste des valeurs déjà jouées (ce qui facilite le jeu en cas de très grand intervalle) :

Une fois la partie gagnée, on obtient (pour une partie entre 0 et 100000) :

En cliquant sur ‘Nouveau’, on retourne à la page d’accueil. Un clic sur le bouton ‘Règles’ permet d’avoir accès aux règles du jeu.

Des effets de transitions sont présents entre les différentes fenêtres du jeu. Un fichier CSS gère les graphismes (couleur, police, effet sur les boutons, barres de défilement,…). Les polices utilisées dans l’application, si elles ne sont pas standards (comme celles utilisées ici : Freestyle Script et Jellyka Saint-Andrew's Queen) sont chargées par l’application, afin d’être sûr que l’utilisateur puisse en profiter.

Si l’utilisateur, durant la partie, rentre un nombre en dehors de l’intervalle, il est averti de son erreur, et le nombre n’est pas pris en compte. Si, à un moment, l’utilisateur entre un chiffre dépassant la taille de int, celui-ci est rejeté.

Conclusion et améliorations possibles :

Il y a un petit problème au niveau de l’affichage des numéros dans la liste : à cause de la police utilisée, le premier chiffre du nombre est très légèrement rogné. Nous n’avons pas su comment résoudre le problème, en sachant qu’en utilisant une police standard, le problème disparaît. Aussi, il nous a fallu choisir : le premier champ de texte permettant de choisir l’intervalle se valide à l’aide d’un bouton, et le deuxième champ de texte qui permet de trouver le nombre généré par l’ordinateur se valide à l’aide de la touche Entrée. À moins de dupliquer le code, nous n’avons pas trouvé de moyen pour que les deux champs de texte se valident à la fois à l’aide d’un bouton et du clavier. Aussi, avec le recul, si cette application était à refaire, nous la ferions à l’aide de JavaFX Scene Builder, afin de coder le plus proprement possible un modèle MVC (notamment grâce à l’aide d’un fichier FXML). Cela aurait également permis de construire l’interface graphique plus rapidement.

Néanmoins, le jeu est totalement opérationnel ; de plus les graphismes et l’ergonomie sont agréables pour l’utilisateur. Le cahier des charges est donc rempli.

Ce rapport succinct ne présente pas tout ce qui a été fait dans l’application, mais les aspects principaux, à savoir comment nous avons procédé, ainsi que quelques points de l’interface graphique qui nous ont paru intéressants.