JACOUD Bastien
REMOND Victor

Rapport de projet
Jeux de plateau et Modularité

I. Introduction

Dans le cadre d'un projet, nous avons dû réaliser en langage Java plusieurs jeux de plateau : un tetris, un jeu de casse-tête et le jeu du Blokus. Malheureusement nous n'avons pu présenter que deux jeux : le tetris et le casse-tête, le Blokus ne présentant pas de difficulté supplémentaire à réaliser par rapport aux deux premiers jeux.

Pour ce faire, nous avons en premier lieu réalisé le plateau, qui va être la base commune des différents jeux. Nous détaillerons ce dernier plus tard, dans notre première partie.

Dans un second temps, nous nous sommes partagés le travail afin de réaliser chacun l'un des deux jeux présentés. Bien entendu, nous nous sommes entraidés lorsque notre binôme rencontrait un problème quelconque, c'est pourquoi nous sommes tous les deux capables d'expliquer le fonctionnement de chaque jeu.

Afin de réaliser notre partie "vue", nous avons utilisé la librairie graphique JavaFX, nous avons aussi fait le choix d'utiliser des fichiers fxml pour nous simplifier la tâche au niveau réalisation graphique. Cependant, nous nous sommes rendus compte plus tard que l'utilisation de fichiers fxml ne présentaient pas de réel intérêt, compte tenu de la modularité de notre plateau. En effet, notre plateau doit être modulable car d'un jeu à l'autre son nombre de lignes et de colonnes va être modifié.

II. Réalisation du plateau

Ce que nous avons appellé le plateau est la partie commune à chacun des jeux, c'est donc le code qui nous a servi de base pour développer nos jeux par la suite.

A. Fonctionnalités

Pour ce qui est des fonctionnalités de notre plateau, nous avons créé toutes les fonctions nécessaires aux jeux dans le plateau, et nous les avons redéfinies si besoin dans le code des jeux eux-mêmes.

C'est aussi dans le code général que nous avons mis toutes les fonctions qui permettent de déplacer, poser et pivoter la pièce. Celles-ci effectuent en même temps les tests de collision afin d'éviter que deux pièces soient superposées, cette situation étant impossible dans chacun des trois jeux que nous avions à coder.

Nous avons aussi fait le choix de gérer les événements générés par le clavier dans la partie modèle de notre code grâce à deux fonctions redéfinies en fonction des événements à prendre en compte dans chaque jeu. Pour les événements par exemple, on a une fonction dans notre classe Plateau qui est appelée à chaque fois qu'une touche du clavier est enfoncée et une autre à chaque fois qu'une touche est relâchée, cependant aucune de ces deux fonctions n'effectue la moindre action, quelle que soit la touche en question. Ce n'est que par la suite que nous les redéfinissons dans chaque jeu afin de gérer les événements qui nous intéressent et ignorer les autres (dans le Tetris, les seules touches qui vont nous intéresser sont les 4 flèches directionnelles alors que dans le Casse-tête on utilise aussi la touche "C")

B. Modularité

Notre plateau étant l'élément central des jeux que nous devions réaliser, il se devait donc d'être modulable. En effet, le plateau du Casse-tête n'est pas de la même taille que celui du Tetris. De plus comme expliqué plus haut, chacun des jeux nécessite une gestion des événements mais celle-ci diffère d'un jeu à l'autre : la création d'une fonction de gestion des événements dans la classe mère complétée par une redéfinission dans chaque classe fille nous est donc apparue comme une obligation.

C. Échanges entre les parties Vue et Modèle

Dans un premier temps, nous avions effectué le lien entre les parties Vue et Modèle grâce aux "properties" de JavaFX. Cela fonctionnait très bien mais nous avons appris plus tard que l'import des classes de JavaFX n'était pas autorisé dans la partie modèle, nous avons donc adopté le patern observable/observer. Autre adaptation du même genre : la gestion des événements. En effet, les premières

fonctions de gestion des événements que nous avions prenaient en paramètres une variable de type “KeyCode”, un type provenant de JavaFX.Nous les avons donc adaptées en leur envoyant un type “String” contenant le nom correspondant l’événement, grâce à la fonction getName().

Grâce au pattern observable/observer, notre modèle n’a plus qu’à notifier le contrôleur en cas de changement et celui-ci met la vue à jour.

D. Diagrammes :

Les diagrammes de classe étant trop gros pour être ajoutés à notre rapport, veuillez les trouver en suivant les liens suivants :

Diagramme de l'ensemble du projet

Diagramme du Plateau

III. Casse Tête

Le jeu du casse tête ou du puzzle est un jeu qui consiste à faire sortir une pièce, cette dernière se trouvant bloquée parmi d'autres au début du jeu. Pour ce faire, nous avons le droit de déplacer les pièces uniquement dans le sens de la longueur, par exemple une pièce horizontale ne pourra se déplacer qu'horizontalement.

La couleur de la pièce qui doit être sortie par l'utilisateur sera toujours rouge, tandis que les autres pièces auront une couleur aléatoire choisie parmi un panel de couleur défini dans PlateauController.

Parlons maintenant des fonctionnalités que nous avons implémentées dans ce jeu. La première fonctionnalité à avoir été implémentée est l'ajout d'un score. Le score compte tout simplement le nombre de coups joués avant d'avoir réussi à sortir la pièce de la grille. Il va ainsi donner un grand intérêt au jeu, le but étant d'utiliser le moins de coups possibles.

Lorsqu'une partie est terminée, c'est à dire lorsque la pièce principale (celle qui est toujours rouge) arrive à la sortie, une boîte de dialogue nous informe que le partie est terminée. Nous ne pouvons alors plus bouger de pièce.

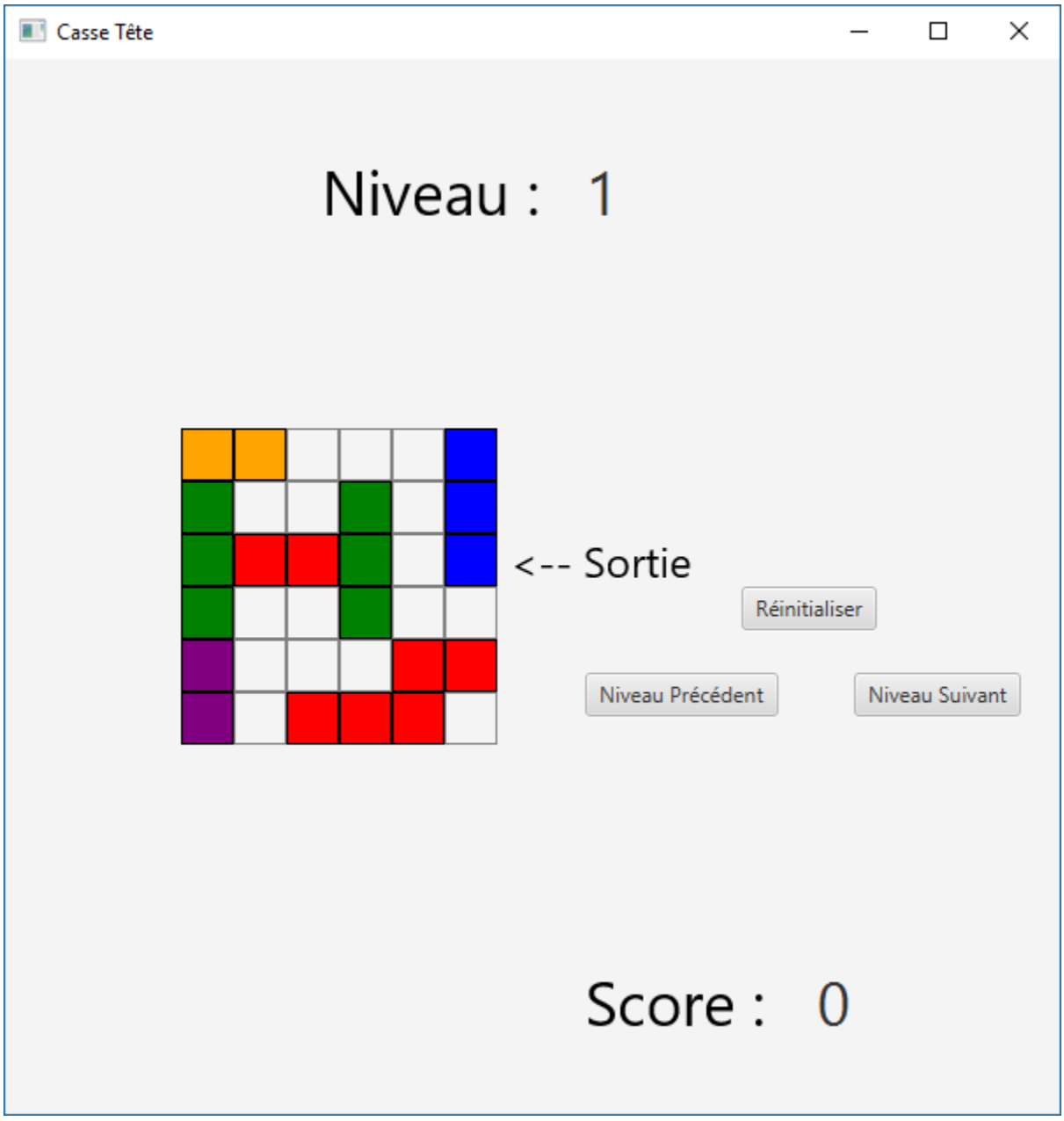
La possibilité de réinitialiser la partie a été mise en place grâce au bouton "Réinitialisation", permettant à n'importe quel moment de la partie de la recommencer. Toutes les pièces reprennent alors la place qu'elles avaient au début de partie et le score se remet à 0.

Le jeu a été réalisé de telle sorte que plusieurs niveaux puissent être mis en place. Grâce aux boutons "niveau précédent" et "niveau suivant", l'utilisateur peut choisir son niveau. S'il n'existe pas de niveau suivant ou de niveau précédent, une fenêtre d'information va s'ouvrir et indiquer que c'est impossible. Nous n'avons cependant réalisé que deux niveaux pour le moment car nous ne voyons pas d'intérêt supplémentaire à rajouter des niveaux, étant donné qu'ils s'implémenteront de la même manière que les niveaux déjà existants.

Pour déplacer une pièce l'utilisateur devra tout d'abord la sélectionner avec la touche "C", puis la déplacer par la suite avec les flèches du clavier. Lorsqu'une pièce est sélectionnée, sa couleur deviendra un peu plus foncée pour que le joueur sache quelle pièce il est en train de déplacer. Par défaut, aucune pièce n'est sélectionnée.

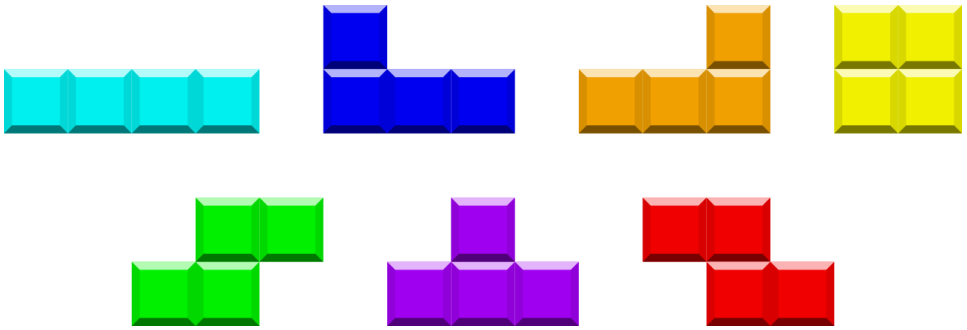
C'est cette dernière partie qui a été la plus difficile à réaliser car au début, lorsque l'on réinitialisait une partie et que l'on souhaitait sélectionner une pièce, les 2 ou 3 premières pièces sélectionnées n'apparaissaient pas dans une couleur plus foncée. Heureusement, ce petit soucis a été résolu depuis.

Certaines améliorations telles que la sauvegarde du meilleur score pour chaque niveau auraient pu être mises en place. Malheureusement, par manque de temps, nous n'avons pas pu les implémenter.



IV. Tetris

Le second jeu que nous avons implémenté, le Tetris, est un jeu bien connu dont le but est de compléter des lignes sur un plateau d'une largeur de 10 cases et d'une hauteur de 20 grâce à des pièces aléatoirement sélectionnées parmi 7 formes différentes :



Dans la version originale du jeu, chaque forme possède sa couleur propre (toutes les lignes et uniquement elles seront bleues par exemple), nous avons fait le choix d'assigner une couleur aléatoirement choisie parmi celles préalablement définies dans le code, ainsi dans notre version il n'y a aucun lien entre la forme d'un pièce et sa couleur.

Les principes de Tetris que nous avons dû conserver sont qu'une fois qu'une ligne est complétée, celle-ci disparaît et toutes les lignes au-dessus descendent d'un rang. La partie se termine quand on ne peut plus poser la pièce suivante.

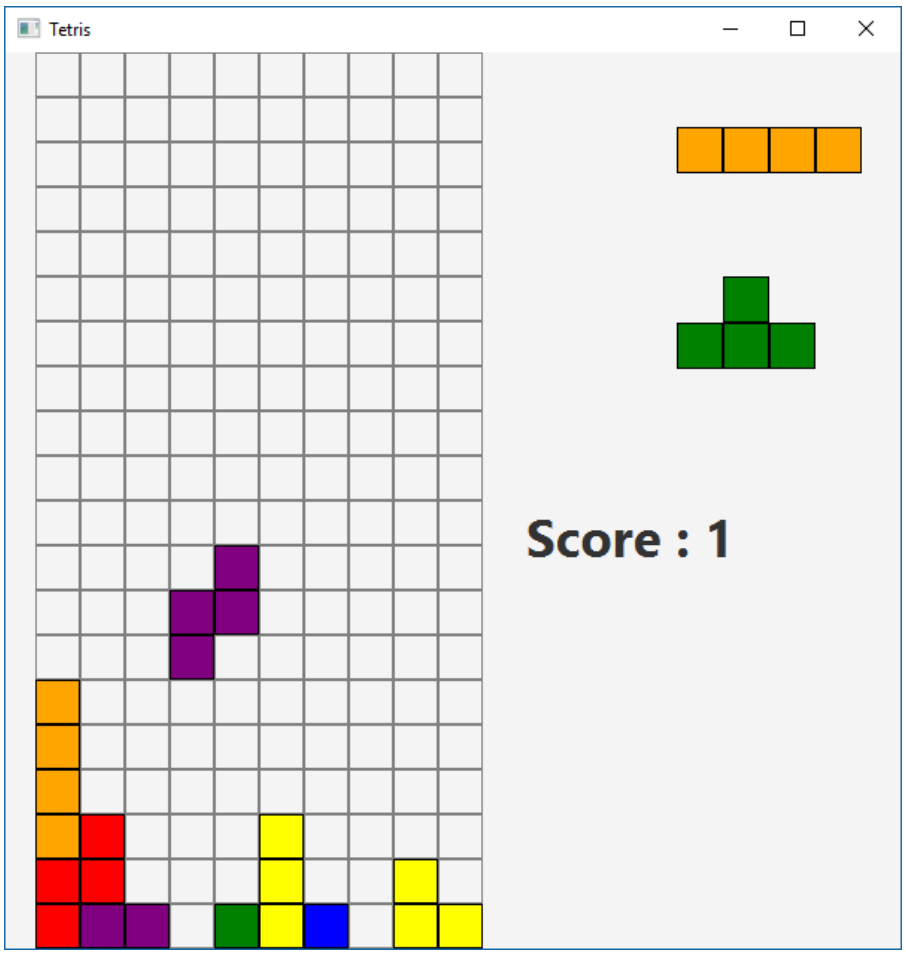
Pour permettre au joueur d'anticiper, nous avons vu que selon la version du Tetris utilisée, nous avions la possibilité de visualiser à l'avance jusqu'à quatre pièces. Nous avons fait le choix d'en afficher deux. Nous affichons aussi le score de la partie : chaque ligne détruite rapporte 1 point, la destruction de plusieurs lignes en une seule fois rapportent plus de points, par exemple un combo de 4 lignes rapporte un total de 15 points. Une fois la partie terminée, une fenêtre s'affiche donnant au joueur son score et le nombre de lignes qui ont été détruites : les deux informations peuvent être intéressantes étant donné qu'avec les bonus on peut avoir en deux parties deux scores différents avec par exemple en seconde partie plus de point mais moins de lignes détruites.

Parmi les fonctionnalités supplémentaires implémentées, la seule qui nous a vraiment paru importante est l'accélération de la chute de la pièce lors de l'appui sur une touche. Nous avons choisi de faire correspondre cette fonctionnalité avec l'appui sur la flèche directionnelle pointant vers le bas afin de rendre son utilisation plus intuitive. Il nous a aussi paru logique de faire en sorte que lorsque le joueur relâche la touche, la pièce reprenne sa vitesse de chute normale. Pour rendre cette

modification possible, nous avons utilisé un thread qui nous permet de faire descendre la pièce d'une ligne, avant de se mettre en pause pendant un certain temps et de recommencer. Ils nous a ensuite suffit de créer deux fonctions permettant de passer d'un temps de pause long à un temps court et inversement.

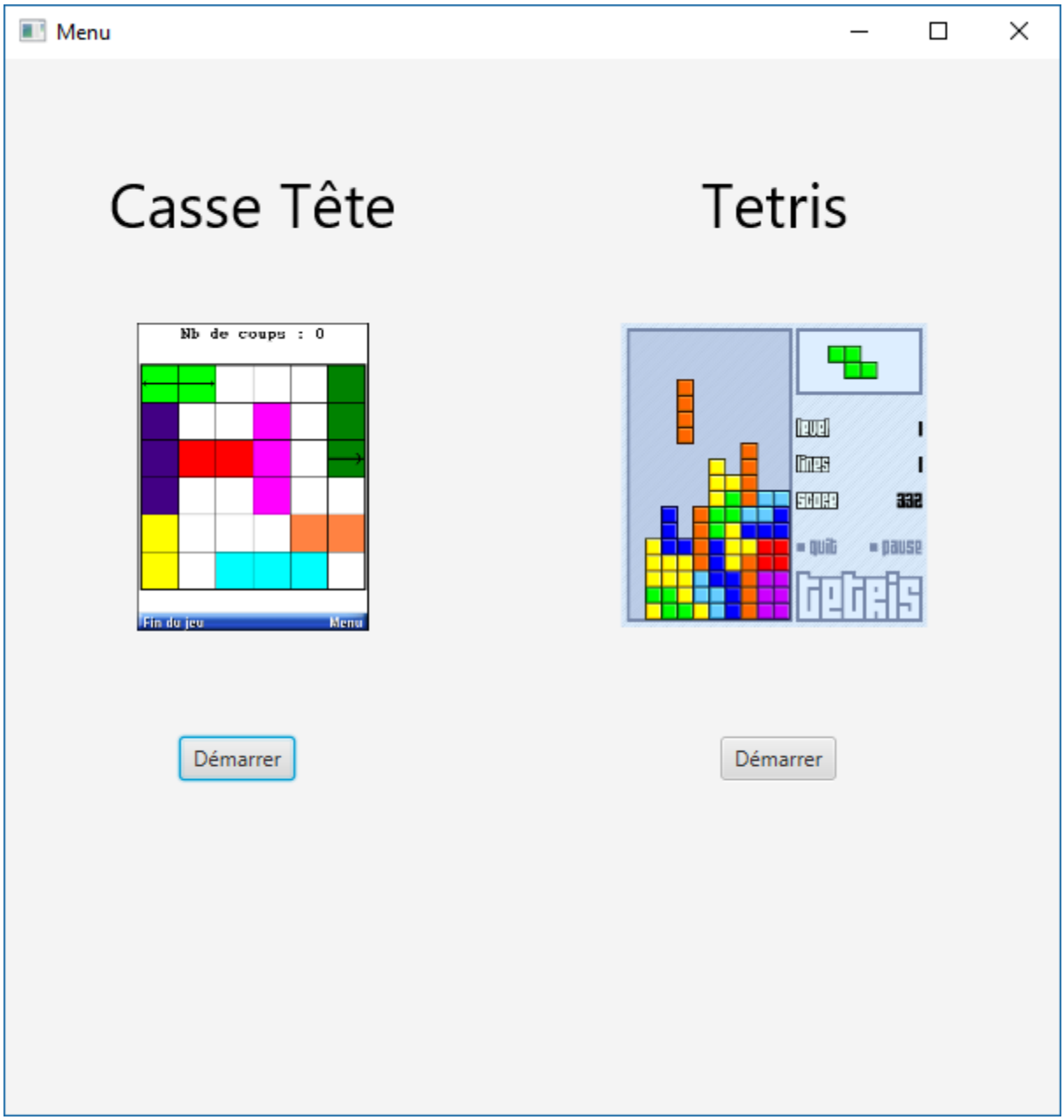
La difficultée majeure que nous avons rencontré durant la conception de ce jeu a été d'afficher le score : nous avions une erreur qui nous indiquait que seul le thread JavaFX pouvait modifier l'affichage et nous avons mis un moment avant de comprendre que l'utilisation de la fonction Platform.runLater(...) pouvait nous débloquer.

Depuis l'oral, nous avons effectué dans le code de ce jeu quelques modifications telles que la modification de la gestion des événements (nous utilisons maintenant une variable de type String pour savoir à quel événement on a affaire au lieu d'une variable de type KeyCode utilisée précédemment, un type issu de JavaFX). De même nous avions codée notre Tetris en utilisant des Properties pour faire le lien entre la vue et le modèle, nous les avons supprimé car elles étaient aussi issues de JavaFX et nous utilisons maintenant le pattern Observable/Observer entre les deux éléments.



V. Extensions proposées

_____Pour permettre aux utilisateurs de choisir le jeu auquel ils souhaitent jouer, nous avons fait en sorte que le jeu démarre sur un menu proposant nos deux jeux :



VI. Conclusion

En conclusion, ce projet nous aura beaucoup appris sur la librairie JavaFX dans le cadre de la création et de l'utilisation une interface graphique comme dans l'utilisation du logiciel SceneBuilder. De plus nos erreurs nous ont permi de comprendre comment utiliser les properties avant d'utiliser le pattern Observable/Observer, nous ne nous en sortons donc pas perdant.

Nous avons de même pu progresser dans le domaine de la mise en place d'architecture MVC dans des projets.