Conception Détaillée
Tétris

Duchesne Lucas

17 janvier 2014

Table des matières

1 Introduction

Ce manuel décrira comment le code à était pensé pour pouvoir le réutiliser dans la suite. Ainsi dans allions étudier chaque classe du programme et décrire les relations entrent-elles. L'ordre de description des classes sera dans l'ordre approximatif qu'elle apparaitra lors de l'exécution du programme.

2 La classe main et option

Nous allons ici présenter la classe main qui sera la classe lancé au début du programme, elle permet de gérer la fenêtre option et de lancer une partie de tétris. La fonction principale de la classe main et *startTatris* qui permet grâce au option que l'on aura validé dans la classe option de crée une instance de la classe Tétris avec les options et aussi de lancer le timer *timerAction* du model principales J'ai choisi de créer la classe *option* héritant de la classe *Jframe* pour pouvoir avoir une interface. Dans cette dernière, nous avons rajouté différent boutton , ratio boutton, des zones de texte (pour la sauvegarde et la lecture de fichier) et le choix de couleur. La fonction principale est la fonction qui s'active lors du clique sur le bouton start de la fenêtre *option* qui appelle la fonction *startTetris* de la classe *main*.

3 La classe Tétris

Pour programmer le tétris j'ai suivi le model : view, controleur et model et c'est trois classes sont abstract de façon à pouvoir les implémenter de différents moyens si vient le besoin de changer l'interface, le controleur et model pour une évolution du logiciel futur. Cette classe hérite de la classe Jframe pour permettre d'avoir un affichage en swing que l'on utilisera dans la classe view. Elle a comme attribut un model *modelTetris*, une view *viewTetris*, un model *controleurTetris* et ainsi que le timer d'affichage *timerAffichage* et le timer de la souris *timer-Souris* qui sert à effectuer une action pendant le temps ou l'utilisateur presse un bouton de la souris.

Le *timerAffichage* va appeler la fonction *afficher* de la classe *view* tous les 50 ms pour afficher le jeu à l'écran.

3.1 La classe Model

La classe *model* permet de gérer la gestion du jeu par rapport aux éléments reçu par la classe *controleur* et de stocker les données du jeu tétris. Nous allons parlez d'abord des attributs de cette classe et ainsi de comment sont stockées nos données de jeux. Pour cela j'ai stocké le tableau de jeux dans un tableau de classe *carre* à deux dimensions de taille fixe permettant de décrire le plateau de jeu appelé *matrix*. On stock une instance de la classe *Tétrominos* pour savoir quelle pièce on utilise et ou la positionnée. Le timer stocké *timerAction* permet de gérer la descente de la pièce lors du choix du jeu à la souris ou autre contrôleur

ou d'effectuer la prochaine action lors de la lecture du fichier enregistré. La classe *model* a les principales fonctions du jeu :

DeplacerTetrominosX : permet de déplacer le *tétrominos* courant, ou plutôt le coordonnée haut gauche du *tétrominos* en considérant que le *tétrominos* est dans un carré de 4x4 case. Pour cela nous appelons la fonction *mettreAJourX* de la classe *tétrominos*.

DeplacerTetrominosY : Fair pareil que la fonction précédant mais sur les coordonnées X et tests si on peut encore déplacer la pièce dans le cas contraire appel la fonction *placer* de *tétrominos*.

rotationTetrominos : permets de faire la rotation de la pièce courante en testant si la rotation peut se faire dans ce cas-là, on appelle la fonction *rotation* de la classe *tétrominos*.

testFinDeligne : permet de tester s'il existe des lignes complètes et de les supprimer du tableau de jeu. Pour cela, elle va tester toutes les lignes en partant du bas jusqu'à ce qu'elle trouve une ligne complètement ligne.

3.1.1 La classe Tétrominos

La classe *Tetrominos* permet de gérer chaque pièce et de stocker leur position sur le plateau. Pour cela j'ai eu besoin de stocker plusieurs variables pour cette classe. Pour savoir sa position j'ai crée un point fictif, pour chaque tétrominos, en haut à gauche d'un carré de 4x4 *carre* et on stocke cette positon dans la variable *coordonne* de type *Point*. Pour connaitre la composition du tétrominos j'ai stocké dans un liste de types *ArrayList¡carre¿* nommé *vecCarre* et la classe *carre* permet de positionner chaque carré par rapport au centre du tétrominos. On garde aussi en mémoire l'angle de rotation dans la variable*angle*.

La classe a comme fonction principale, *testDeplacement* qui permet de tester si le tétrominos peut se déplacer à cette position sur le plateau. Pour cela, elle va tester pour chaque *carre* du plateau si elle touche une *carre* du tétrominos actuelle avec la fonction *intersects* de la classe *rectangle* implémenté par la classe *carre*.

La fonction *placer* est une fonction principale qui permet de placer la pièce sur le plateau de jeu. Pour cela, en premier lieu elle remet la pièce à l'endroit si la pièce n'est pas dans une rotation normale du jeu tétris, après rotation de la pièce on vérifie si on ne peut pas déplacer la pièce encore dans ce cas la, on rend la main à l'utilisateur. Dans l'autre cas, on met les *carre* de la pièce dans le tableau à l'endroit où sont les *carre* de *vecCarre*.

Les autres fonctions *mettreAJourX*, *mettreAJourY* et *rotation* permet de déplacer et d'effectuer la rotation du tétrominos pour cela on parcours chaque carre du vecteur et on déplace les coordonnées de chaque *cassse* ou de faire la rotation avec la fonction *rotationCenter* de *carre*.

La classe tetrominosFactory Cette classe permet de gérer la création des tétrominos de différent moyen possible. J'ai créé la fonction de telle façon qu'elle gérer la gestion de différent type de création pour savoir quel type de création on est on stocke dans la variable *modeCreation*. Les différents types de création

ont chaque une fonction pour une création aléatoire *randTetrominos*, on doit préalablement spécifier un fichier de sauvegarde de la séquence pour pouvoir rejouer la partie avec la fonction *setNameFileSav* et pour la création par rapport à une séquence enregisté *seqTetrominos*. Pour cette dernière on doit spécifié le nom du fichier à lire avec la fonction *setNameFileSeq*.

La fonction *setTetrominos* permet de crée un tétrominos du type passé en paramètre et le renvoie.

3.1.2 La classe carre

Cette classe permet de gérer chaque case du jeu. Pour cela elle extends la classe *rectangle* pour avoir facilement la gestion graphique et des déplacements des cubes sur le jeu. J'ai stocké en plus comme variable la couleur du cube *couleur*, une variable pour savoir si le carré est plein *plein* et une variable de type *Point2D* pour stocker le centre du carre. Les fonctions principales de cette classe sont :

rotationCenter : permet de faire la rotation du *carre* grâce à la fonction *AffineTransform* qui va effectue la transformation du de la variable *center*.

changerCoordonne : permet de faire la translation du *carre* grâce à la fonction *translate* de la classe *rectangle*.

changeCarre : permet de déplacer un *carre* lorsque on veut descendre une ligne.

3.2 La classe Controleur

La classe *controleur* est une classe abstract qui permet d'implémenter les différents contrôleurs du tétris plus facilement. J'ai développé cette classe de façon quel garde en variable la classe *savFile* qui contient le nom du fichier de sauvegarde des commandes et qui permet d'effectué des actions dessus. Elle garde aussi en variable le type de contrôleur utilisé lors de la section dans la variable *type*.

La principale fonction est *gestionTouche* qui est abstract et qui permet de l'utiliser lors du développement des différents types de contrôleur.

3.2.1 La classe souris

Cette classe implémente la classe *controleur*, elle permet de gérer les déplacements de la souris. Comme on a développé le programme sous *swing* pour capter les mouvements de la souris on l'a mis dans la *tétris* qui est notre *Jframe* principale et qui en revient par la création des trois fonctions suivant dans la classe *tétris* : *formMousePressed*, *formMouseMoved* et *formMouseReleased*.

La fonction *formMouseMoved* permet de récupéré les coordonnées de la souris lors du déplacement et de l'envoyé à la fonction *gestionTouche* de la classe *souris*. Cette dernière va appeler la fonction *deplacerTetrominosX* du *model* avec en paramètres la position selon X de la souris.

Les fonctions emphformMousePressed et *formMouseReleased* permettent de gérer le clique de la souris. La première s'activera lorsque qu'on cliquera sur un des boutons (bouton droit tourné à droite et inversement pour le gauche), elle déclenchera un timer qui effectuera dans chaque un des cas l'appel de la fonction *rotation* du *model* et sauvegardera le mouvement dans le fichier de la classe *savFile* tous les 100 ms.

3.2.2 La classe lecteurSeq

Cette classe implémente la classe *controleur*, elle permet de gérer la lecture d'un fichier de jeu et effectuer la partie enregistrer. Pour cela, elle va charger le fichier voulu dans la classe *savFile* lors de la création de l'instance de la classe.

J'ai implémenté la fonction *gestionTouche* de la classe *controleur* pour quel gère la lecture du fichier, pour cela elle appelle la fonction *getNextMove* de la classe *savFile* qui permet de récupérer la prochain mouvement écrit dans le fichier sauvegarde et d'appeler la fonction correspondante dans la classe *model* pour quel rejoue la partie sauvegardé.

Cette fonction implémentée, sera appelée dans le *timerAction* tous les 10 ms, comme on a sauvegardé le temps pour chaque action on pourra modifier cet appel pour que cela puisse effectué le jeu sauvegardé en temps normal en effectuant toutes les actions passé entre chaque 10 ms.

3.3 La classe View

La classe *view* est une classe qui permet d'implémenter les différentes vues de notre *tétris* dans le cas où l'on voudra avoir différents affichages. Cette classe a comme fonction abstract *afficher* qui est appeler pour afficher le tétris. Dans notre cas, on l'implémente pour qu'elle puisse utiliser la bibliothèque *Swing* du java.

3.3.1 La classe panel

Pour implémenter l'affichage en *Swing* nous avons besoin d'implémenter la classe *JPanel* et aussi *view* pour notre programme. Pour cela, cette fonction fera la liaison entre l'implémentation des deux. Elle a comme variable *monPanel* qui est de type *persoPanel* qui n'est d'autre n'autre implémentation du *Jpanel*.

afficher est implémenté dans cette classe pour appeler le *repaint* de la classe *persoPanel*

3.3.2 La classe persoPanel

Cette classe permet d'implémenté la classe *jPanel* pour avoir un affichage en *swing*. Cette classe a comme variable *matrix* et *tetrominos* pour savoir ce qui faut afficher et de la variable *sameColor* qui permet de savoir si l'affichage doit se faire avec une seule couleur.

4 Conclusion

En conclusion, le programme implémente les fonctions essentielles du cahier des charges :

1. Reproduction du comportement du jeu standard
   (a) Même pièce, les 7 pièces différentes, **Oui**.
   (b) Même actions = tourner/déplacer/tomber, **Oui**.
   (c) Possibilité de mettre en pause, **Non**.
2. Déplacement avec une précision arbitraire des pièces, **Oui**.
3. Rotation avec précision arbitraire, **Oui**.
4. Contrôle du jeu à la souris, **Oui**, la classe *souris*.
5. Contrôle du jeu au clavier, **Non**, classe existente mais pas implémenté.
6. Gestion des conflits entre commandes de l'utilisateur et contraintes du jeu, **Oui**, la fonction *tetrominos.testDeplacement()*.
7. Recaler les Tetrominoes sur la grille des blocs une fois qu'ils sont stabilisés/tombés, **Oui**, la fonction *tetrominos.placer()*.
8. Détermination de la séquence des Tetrominoes, **Oui**, la classe *tetrominosFactory*.
9. Sauvegarde du comportement de l'utilisateur, **Oui**, la classe *savFile*.
10. Contrôle du jeu directement depuis un fichier de données, **Oui**, la classe *lecteurSeq*.
11. Limitation possible de la vitesse maximale de déplacement des Tetrominos, **Oui**.

Le code corespont au les exigences techniques voulue :

1. Implémentation en Java, **Oui**.
2. Interface graphique Swing, **Oui**.
3. Finalisation sous forme de package exécutable avec code source intégré, **Oui**.
4. Indépendance à des bibliothèques/packages sont standards , **Oui**.
5. Modèle de conception permettant de produire plusieurs vues/contrôles, **Oui**.
6. Rendre le code le plus modulaire et extensible possible, **Casi**.