Conception Détaillée
Tétris

Duchesne Lucas

26 janvier 2014

Table des matières

1 Introduction

Ce manuel décrira comment le code a été pensé pour pouvoir le réutiliser dans la suite. Ainsi nous allions étudier en premier lieu la conception générale du tétris grâce à des diagrammes UML et ainsi avoir une idée générale du logiciel que nous allons présenter. Par la suite nous étudierions plus le code pour décrire chaque détails de la programmation du logiciel. Pour cela, nous allons décrire classe par classe du code java et l'expliquer. L'ordre de description des classes sera approximatif celle de l'exécution du programme.

Tetris est un jeu vidéo de puzzle conçu en 1984 par Alekseï Pajitnov. Bâti sur des règles simples et exigeant intelligence et adresse, il est l'un des jeux vidéo les plus populaires au monde.

Des pièces de couleur et de formes différentes descendent du haut de l'écran. Le joueur ne peut pas ralentir ou empêcher cette chute mais peut l'accélérer ou décider à quel angle de rotation (0, 90, 180, 270) et à quel emplacement latéral l'objet peut atterrir. Lorsqu'une ligne horizontale est complétée sans vide, elle disparaît et les blocs supérieurs tombent. Si le joueur ne parvient pas à faire disparaître les lignes assez vite et que l'écran se remplit jusqu'en haut, il est submergé et la partie est finie.

2 UML et conception

2.1 Présentation du projet

Avant de commencer la description de la conception post programmation du logiciel, nous allons explicités les spécifications du logiciel voulue dans le cahier des charges.

Nous voulons créer un tétris avec les fonctionnalités normal d'un tétris comme décrit en introduction. Mais nous voulons implémente une autre version tétris pour des fins d'utilisations pour voir comment les personnes réagissent pendant une partie de tétris. Pour cela, nous voulons pouvoir déplacer les pièces du tétris avec une précision de l'ordre du pixel et de même pour la rotation sera au degré prêt. De plus, nous aimerons pouvoir enregistrer les mouvements d'une partie pour pouvoir la rejouer plus tard pour l'analyser et pour affecter les décisions des utilisateurs nous voudrions pouvoir tous modifier, tels que avoir plus qu'une seule couleur jusqu'à pouvoir commencer le tétris avec des briques déjà en place en bas de l'espace de jeu.

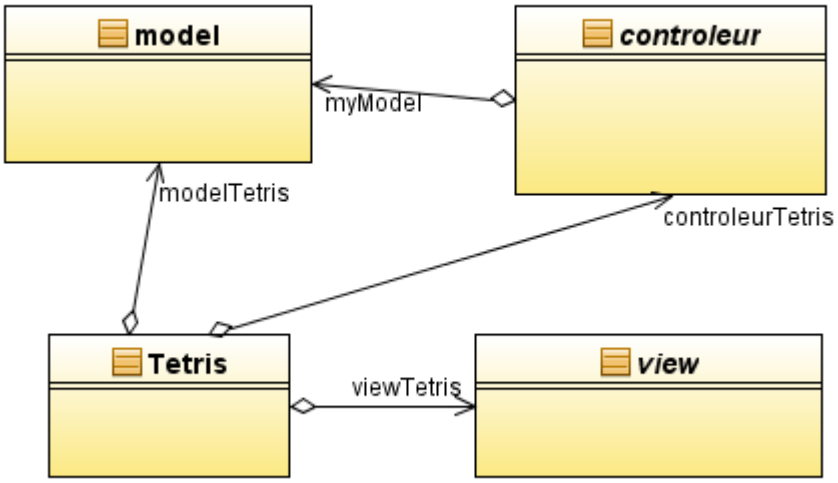
2.2 Conception

Le choix du langages, java, et de la librairies graphiques, swing, étant suggérer dans le cahier des charges nous avons utilisé celui ci. Pour programmer le tétris, nous voulions une conception permettant l'extension futur du logiciel pour cela nous sommes partie sur une conception du logiciel sur la base vue, modèle et contrôleur qui seront standard pour pouvoir les ré-implémenter facilement

chaque une séparément sans avoir à retoucher les autres pour que logiciel fonctionne.

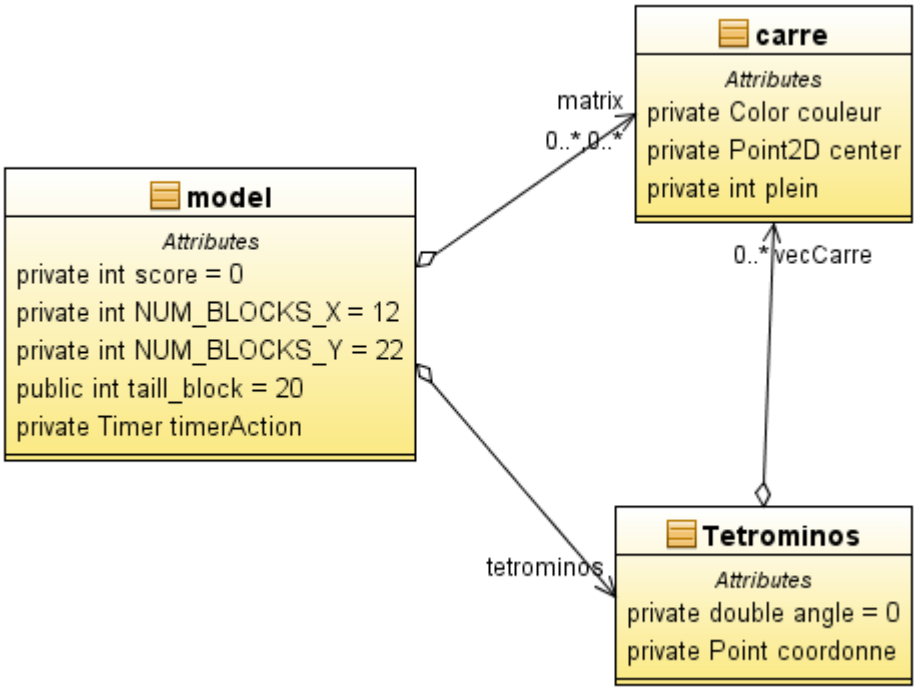
Cette conception sera implémenté en trois classe abstract, la classe model sera compensé des fonctions pour gérer la gestion du jeu, la classe view pour afficher les donnée du jeu, la classe controleur permettant de gérer les événements de l'utilisateur pour le jeu et de plus nous avons pensé à une classe en plus pour avoir les trois gérer les trois classes tels le montre le graphique suivant :

FIGURE 1 – Les classes de base de notre tétris



Le jeu tétris est basé sur la gestion de carré sur un tableau avec des carrés qui forme le décor et d'autre qui forme les pièces du tétris qui tombe. Nous avons donc pensé à gérer c'est carré à l'aide d'une classe carré permettant de définir les détails de chaque carré de l'écran tels que leurs positions et couleurs. Ainsi pour gérer la pièce descendante, nous avons pensé à une classe tétrominos permettant de gérer cette dernière avec les fonctions s'effectuent dessus et un tableau de carre pour définir la pièce. Et pour définir le tableau de jeu, nous avons définis un tableau de carre dans la classe model. Le jeu a besoin de créer les tétrominos de différent moyen pour cela nous avons pensé à une classe, *terominosFactory* implémentent différentes méthodes, soit pour la création aléatoire des tétrominos soit la création suivant une liste donné en paramètre de notre logiciel.

FIGURE 2 – Les classes tétrominos, carre et model



L'affichage choisit pour ce logiciel et l'utiliation de la bibliothéque *Swing* de java pour cela nous avons besoin de réuttiliser les classe jFrame et jPanel permettant l'affichage de notre tétris grâce à des fenêtres de base. De plus, ces classes en java permettant de capeter les actions de l'utilisateur sur notre logiciel que ce soit mouvement de la souris, touche du clavier... Il faudra donc que notre controleur, pour la souris, permet de récupérer ces valeurs et pour la classe view permette d'utilisé le *jPanel*. Nous avons donc décidé de faire la classe principale *tétris* héritant de la classe *jframe* permettant ainsi d'etre notre classe principale et de relié le controleur, model et vue. Les actions, ainsi capté, ferons appel aux fonctons abstract de la classe *controleur*. Pour le problème de l'affichage, nous devons réimplémenter la classe *view* et a classe *jPanel*, ici en java, nous avons choisi de faire une classe *panel* héritant de la classe *view* et contenant en variable une classe *persoPanel* héritant de la classe *jPanel*.

FIGURE 3 – Les classes view et controleur en Swing

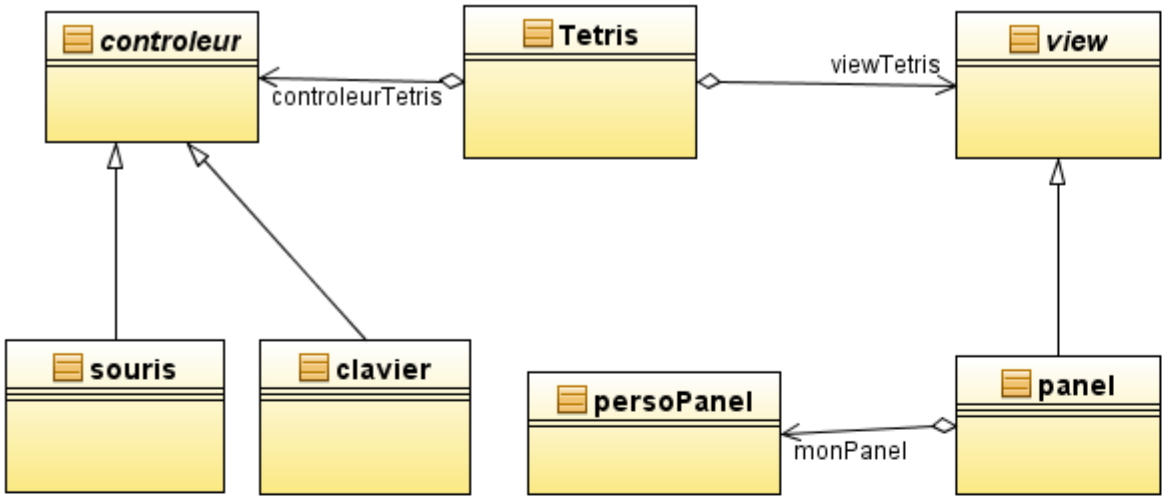
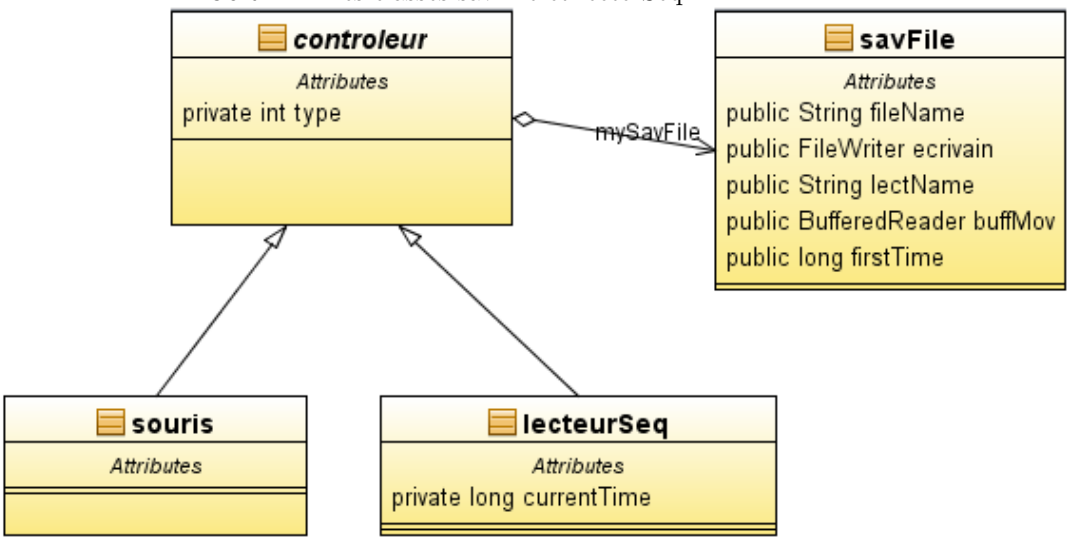


FIGURE 4 – Les classes savFile et lecterSeq

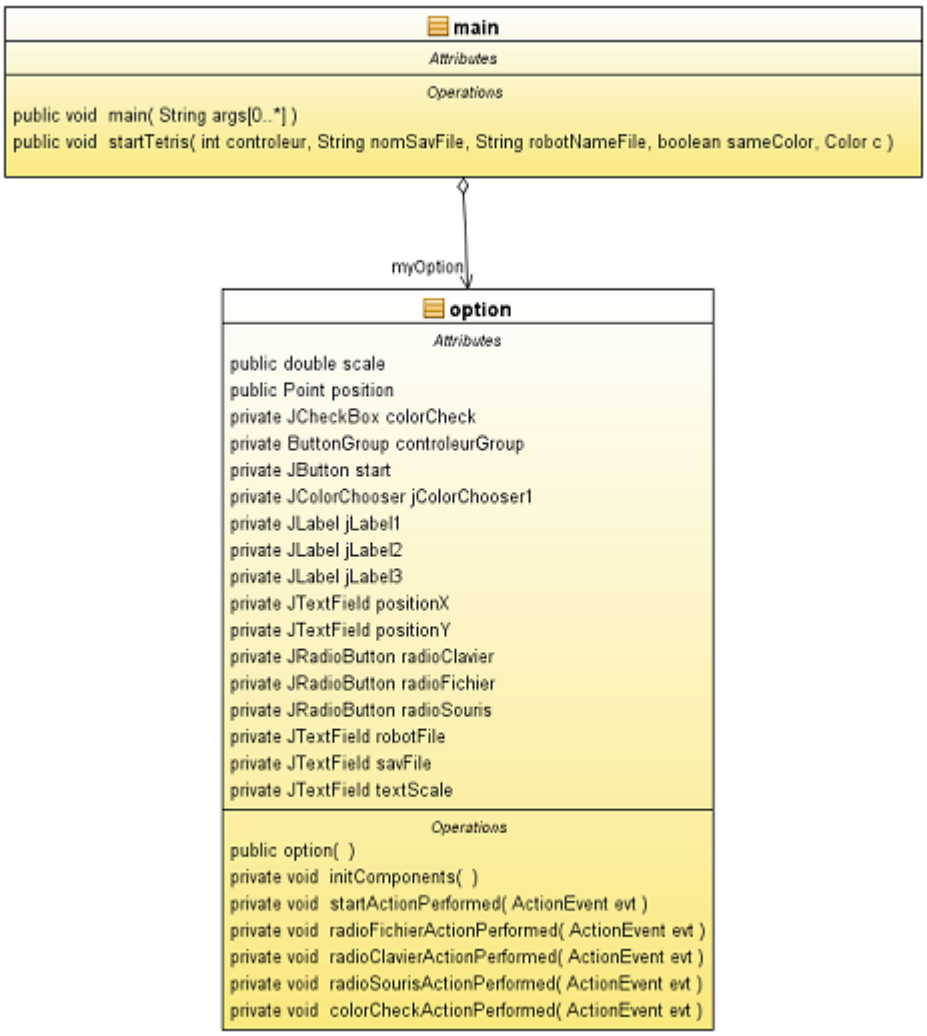


Pour notre logiciel, nous avons besoin de sauvegarder les données de jeu pour pouvoir rejouer la partie dans le besoin, pour cela, nous avons besoin d'une classe qui nous permetteras de sauegarder les actions *savFile* et une classe permettant de les lires *lecterSeq*. Cette dernière sera une classe déduit de la classe controleur car nous la considérons comme un controleur gérant le jeu comme la souris ou le clavier.

3 La classe main et option

Nous allons ici présenter la classe main, la première classe au début du programme, elle permet de gérer la fenêtre *option* et de lancer une partie de tétris.

FIGURE 5 – Les classes main et option détaillé



La fonction principale de la classe main et *startTatris* permettant d'intensifier la classe Tétris pour la partie avec les options donnée en paramètre. Elle lance le timer *timerAction* principales de la partie, gérant la chute de la pièce ou la lecture du fichier de sauvegarde.

Nous avons créer la classe *option* héritant de la classe *Jframe* pour pouvoir avoir une interface. Dans cette dernière, nous avons rajouté différents boutons, ratio bouton, des zones de texte (pour la sauvegarde et la lecture de fichier) et le choix de couleur. La fonction principale s'active lors du clic sur le bouton *start* de la fenêtre *option*. Elle appelle la fonction *startTetris* de la classe *main*.

4 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 *timerSouris* 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.

4.1 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.

4.1.1 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 enregistré *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.

4.1.2 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.

4.2 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.

4.2.1 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.

4.2.2 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.

4.3 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.

4.3.1 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*

4.3.2 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.

5 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 existante 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**.