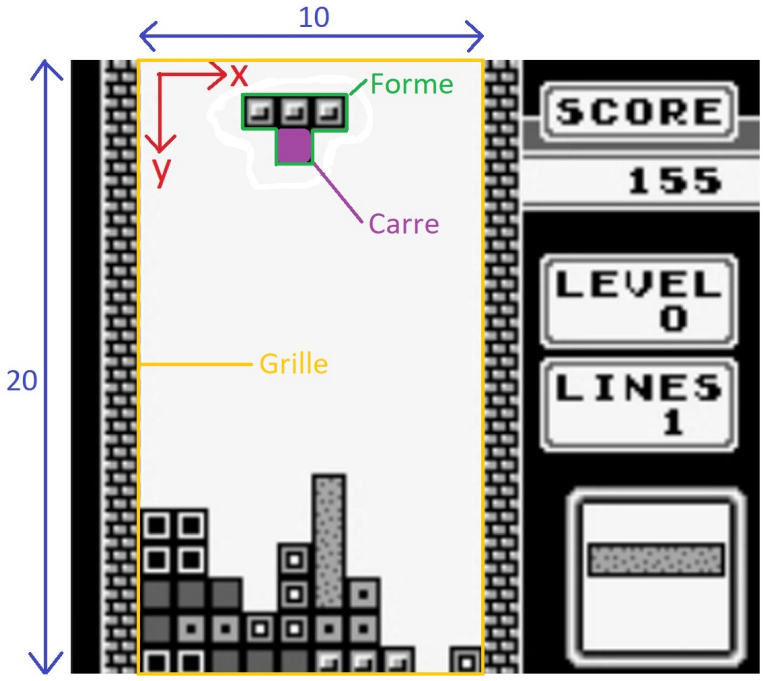
Programmation orientée objet
TP2 : Tétris, partie 1

Romain Marie

Février 2019

3iLINGENIEURS
</les sciences informatiques>

Dans ce TP, vous allez suivre un processus qui va construire une application Tétris de manière très progressive. Ne vous contentez pas de simplement suivre les instructions une à une, essayez de comprendre pourquoi elles sont réalisées dans cet ordre, et en quoi chaque étape permet d'avancer vers le résultat final.



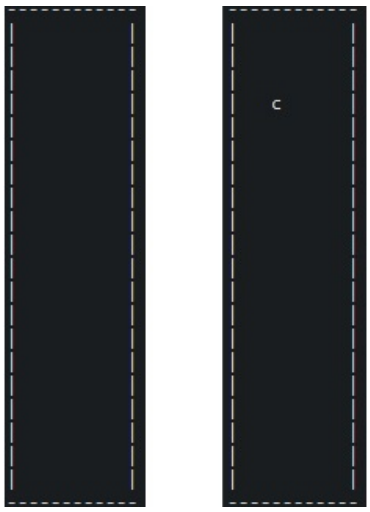
1 Etape 1 : Mise en place du projet

1. Dans votre workspace, créez un projet que vous nommerez "Tetris".
2. Importez dans votre projet les deux fichiers "MainTetris.java" et "ZoneSaisie.java" que vous trouverez sur TOUSCOM. Ces fichiers définissent le point d'entrée de votre programme (méthode *main(???)*), et contiennent le code qui permet de voir lorsque les flèches du clavier sont pressées.
3. Exécutez votre programme. Normalement, lorsque vous appuyez sur une flèche du clavier, un texte affichant laquelle devrait apparaitre à l'écran.
4. Ajoutez à votre projet une classe ***Carre***. Cette classe servira a représenter les petits carrés formant les formes du tétris, et occupant chacun une case dans l'espace de jeu. Les propriétés composant un ***Carre*** seront donc :
   - un entier **x** : La position horizontale du ***Carre*** compris entre 0 et 9
   - un entier **y** : La position verticale du ***Carre*** compris entre 0 et 19
   - un caractère **c** : L'aspect visuel du ***Carre*** (pour affichage)
5. Ajoutez une classe ***Grille***. Cette classe servira à représenter la zone de jeu, et permettra en particulier de détecter lorsqu'une ligne sera complète, ou quand une forme aura fini sa descente. L'unique propriété d'une ***Grille*** sera **carres**, une matrice de ***Carre*** de dimensions [20][10] (20 lignes, 10 colonnes).
6. Ajoutez un constructeur par défaut à votre classe ***Grille*** qui se contente de créer la matrice de 10x20 ***Carre***. Puisqu'aucun ***Carre*** n'est présent dans la grille au début du jeu, chaque case devra être initialisée à **null** (mot-clé indiquant qu'une référence n'est associé à aucun objet).
7. Ajoutez un constructeur avec paramètres à votre classe ***Carre***. Elle initialisera ses 3 paramètres, c'est à dire ses coordonnées **x** et **y**, ainsi que le caractère **c** qui le représente. Vous ferez attention à ne pas pouvoir placer un ***Carre*** hors de la grille !
8. Ajoutez les méthodes *int getX()*, *int getY()* et *char getC()* à votre classe ***Carre***. Ces méthodes s'appellent des accesseurs, et doivent permettre de récupérer les valeurs des propriétés privées d'un Carre.

2 Validation étape 1

Avant d'aller plus loin, vous allez vérifier que votre programme permet de correctement créer une ***Grille***, d'y insérer un ***Carre***, et d'afficher le tout en mode texte

1. Ajoutez une méthode *void ajouterCarre(Carre c)* à votre classe ***Grille***, qui prend un ***Carre*** en paramètre, et le place à la bonne case dans la ***Grille***. Si la case est déja occupée, la méthode ne fera rien du tout.
2. Ajoutez une méthode *void affichage()* à votre classe ***Grille*** qui affiche la grille au format texte. Une case vide (null) sera représentée par un " ", tandis qu'une case occupée par un ***Carre*** sera représentée par la propriété **c** du ***Carre***. Ci-dessous, un exemple d'une ***Grille*** vide, ainsi qu'une ***Grille*** avec un unique ***Carre*** en coordonnées (3,3).



3. Dans votre méthode *main()*, en vous plaçant avant la boucle principale du programme, créez une instance de **`Grille`** que vous nommerez g. Utilisez la méthode *affichage()* pour l'afficher. Vous devriez normalement voir la figure de gauche ci-dessus.

4. Juste en dessous, créez une instance c de **`Carre`** en coordonnées (3,3), représenté par le caractère 'c'.

5. Ajoutez c à g, puis afficher de nouveau g. Vous devriez normalement voir la figure de droite ci-dessus.

3 Etape 2 : Déplacer un Carré

le fil conducteur de Tétris est de déplacer des **`Carre`** (par groupe de 4), jusqu'à ce qu'ils rencontrent un obstacle (bas de la zone de jeu ou autre **`Carre`**). La première étape consiste donc à considérer un unique **`Carre`** et à lui permettre de se déplacer dans la **`Grille`**.

Faire déplacer un **`Carre`** dans la **`Grille`** implique 3 actions consécutives :

- Il faut d'abord vérifier dans la **`Grille`** si la case est libre
- Puis, si c'est le cas, il faut déplacer le **`Carre`** dans la **`Grille`**, c'est à dire vider la case courante, et remplir celle d'arrivée.
- Enfin il faut modifier les coordonnées **x** et **y** du **`Carre`** pour qu'elles correspondent à la case d'arrivée.

1. Ajoutez une méthode *boolean estLibre(int x,int y)* à votre classe **`Grille`**. Cette méthode devra vérifier si une case de la **`Grille`** est occupée par un **`Carre`**. Dans le cas où les coordonnées sont hors de la zone de jeu, il faudra bien sûr renvoyer false.

2. Ajoutez une méthode *void viderCase(int x,int y)* à votre classe **`Grille`**, qui met à **null** la case occupée de coordonnées (x, y). Bien sûr, si les coordonnées sont hors de la grille, la méthode n'effectuera aucune action.

3. Ajoutez une méthode *boolean peutDescendre(Grille g)* à votre classe **`Carre`**. Cette méthode devra interroger la grille pour savoir si la case se situant sous le **`Carre`** est libre.

4. Ajoutez une méthode *void descendre(Grille g)* à votre classe **`Carre`**. Cette méthode devra réaliser les 3 actions suivantes :
   - utiliser la méthode *peutDescendre(Grille g)* pour savoir si la case de destination est libre
   - Si c'est le cas, modifier les propriétés du **`Carre`** pour qu'elles correspondent à ses nouvelles coordonnées dans la grille.

- Toujours si c'est le cas, déplacer le **Carre** dans la **Grille**, c'est à dire vider la case courante et remplir celle d'arrivée.

5. De la même façon, ajoutez les méthodes *boolean peutGauche(Grille g)*, *boolean peutDroite(Grille g)*, *void gauche(Grille g)* et *void droite(Grille g)* à votre classe **Carre**.

4 Validation étape 2

Pour valider cette étape, vous allez maintenant faire un petit programme qui va simplement adapter la boucle principale du programme pour faire se déplacer un **Carre**.

1. Dans la boucle principale de votre méthode *main()*, modifiez le switch/case, pour que, lorsque la flèche "bas" est pressée, le **Carre** tente de descendre. Vous utiliserez bien sûr la méthode descendre() de la classe **Carre**.

2. Faites de même pour les flèches gauche et droite !

3. Affichez la **Grille** à chaque passage dans la boucle principale du programme. En appuyant sur les flèches, vous devriez maintenant être capable de faire bouger le **Carre** dans la **Grille**.

5 Etape 3 : Supprimer les lignes complètes

A ce stade, vous êtes en mesure de faire descendre des **Carre** dans la **Grille**, donc de remplir cette dernière. Il faut maintenant faire en sorte qu'une ligne complète soit détectée et supprimée.

1. Commencez par adapter votre programme pour que, lorsque le **Carre** courant ne puisse plus descendre, la référence *c* soit associée à un nouveau **Carre** initialisé en haut de la **Grille**. Vous pourrez décommenter la ligne *z.descenteAuto()* qui simulera l'appui sur la touche bas à un rythme de 1fois par seconde.

2. Dans votre classe **Grille**, ajoutez une méthode *boolean lignePleine(int l)* qui renvoie un booléen indiquant si une ligne (passée en paramètre) est pleine.

3. Ajoutez une méthode *supprimerLigne(int l)* qui supprimera la ligne passée en paramètre. Bien sûr, quand une ligne disparait, il faudra non seulement mettre à **null** toutes les cases qui la composent, mais aussi faire descendre une à une, toutes les lignes situées au dessus.

4. Modifiez votre boucle principale d'exécution pour que, chaque fois qu'un **Carre** finisse sa descente, chaque ligne complète de la **Grille** soit supprimée. Cette étape devra avoir lieu avant qu'un nouveau **Carre** soit créé !

6 Validation étape 3

Pour valider cette étape, il suffit de faire descendre suffisamment de **Carre** pour former une ligne, et de vérifier que sa suppression se passe comme il faut. Normalement, aucune ligne de code supplémentaire ne devrait être ajoutée.

7 Etape 4 : Déplacer une Forme

Pour finir, maintenant qu'il est possible de déplacer un **Carre** à volonté, il reste à en manipuler 4 à la fois, en les regroupant dans une **Forme**.

1. Ajoutez une classe **Forme** à votre projet. Elle contiendra un unique paramètre **carres**, qui sera un tableau de 4 **Carre**.

2. Ajoutez un constructeur avec paramètres qui permet d'initialiser une **`Forme`** de telle sorte que tous les **`Carre`** qui la composent aient le même caractère **c**. Ce constructeur prendra en paramètres un entier **id** servant d'identifiant permettant de savoir quel type de forme doit être créée, ainsi que le caractère **c** associé à la **`Forme`**. Bien sûr, la création d'une **`Forme`** implique la construction des 4 **`Carre`** qui la composent. Leurs coordonnées respectives devront être automatiquement déduites de l'identifiant passé en argument.

3. Ajoutez une méthode *void ajouterForme(Forme f)* à votre classe **`Grille`**. Cette méthode devra ajouter à la **`Grille`** chacun des 4 **`Carre`** de la **`Forme`** f. Vous utiliserez pour cela la méthode *ajouterCarre(Carre c)* qui existe déjà.

4. Ajoutez les méthodes *boolean peutDescendre(Grille g)*, *boolean peutDroite(Grille g)*, *boolean peutGauche(Grille g)* qui vérifient si **toute** la **`Forme`** peut se déplacer dans une direction. Attention : vous devrez faire en sorte que les **`Carre`** de la **`Forme`** ne se gènent pas entre eux.

5. Ajoutez les méthodes *void descendre()*, *void droite()* et *void gauche()* qui feront descendre / aller à droite / aller à gauche chacun des 4 **`Carre`** de la **`Forme`**. Attention : l'ordre dans lequel vous déplacerez les **`Carre`** risque d'influer sur la validité de votre algorithme.

8 Validation étape 4

Pour valider cette étape, vous allez adapter votre programme pour que le **`Carre`** qui descend soit remplacé par une **`Forme`**. Bien sûr, puisque **`Forme`** est une classe abstraite, il va falloir commencer par créer les classes filles

1. Modifiez votre boucle principale pour que ce soit une **`Forme`** qui descende au lieu d'un **`Carre`**. Au moment de la création d'une **`Forme`**, un tirage au sort aura lieu pour en connaitre la nature exacte.

9 Etape 5 : faire tourner une Forme

Normalement, votre Tetris devrait être opérationnel (et jouable). Il reste toutefois une des fonctionnalités les plus importantes à mettre en place : la rotation. Pour rappel, une rotation dans le plan peut être modélisée par une matrice M qui permet de passer des coordonnées (x, y) d'un point vers ses coordonnées (x', y') après transformation :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

En choisissant une rotation $\theta = \Pi/2$, et en rappelant que $cos(\Pi/2) = 0, sin(\Pi/2) = 1$, cette relation devient :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Autrement dit, on trouve $x' = -y$ et $y' = x$. C'est cette relation très simple que vous allez programmer pour faire tourner votre pièce.

1. Ajoutez à votre classe **`Forme`** deux propriétés xc et yc correspondant au centre de rotation de votre forme. Il devra être cohérent avec la disposition des 4 **`Carre`** qui la composent.

2. Mettez à jour le constructeur et les méthodes *descendre(), gauche() et droite()*, pour faire en sorte que le centre de rotation se déplace en même temps que la **`Forme`**.

3. Ajoutez la méthode *boolean peutTourner()* qui teste si la **`Forme`** peut tourner, c'est à dire si la destination de chaque **`Carre`** est libre.

4. Ajoutez la méthode *void tourner()* qui applique une rotation à la **`Forme`**.

5. En associant la rotation à la flèche du haut de votre clavier, mettez à jour votre boucle principale d'exécution.