POO – Programmation Orientée Objet en Java

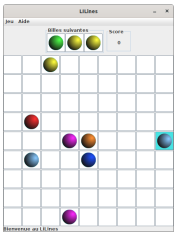
Fiche de TP numéro 4

Des balles, des lignes et des interfaces...

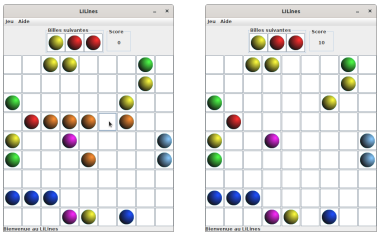
On souhaite développer une application Java reprenant les règles du jeu *Glines*. *Glines*, ou "Cinq et plus" est un jeu de logique où vous devrez aligner 5 billes de même couleur. Le jeu commence avec 5 billes placées aléatoirement sur un plateau.

Vous pouvez déplacer la bille de votre choix vers une autre case, du moment qu'il existe un chemin libre faits de lignes horizontales et verticales de la bille jusqu'à la case choisie. Ici, le joueur a choisi de déplacer la bille bleu clair vers la position du pointeur.

Le déplacement s'effectue puis l'ordinateur place de nouveau 3 billes au hasard sur le plateau. Les couleurs de ces billes étaient affichées auparavant comme les *"Billes suivantes"* (une bleue et deux roses).



Le joueur doit alors déplacer une nouvelle bille, puis les billes suivantes tomberont, etc etc. Lorsque le joueur réussit à aligner 5 billes de la même couleur, elles disparaissent, et le joueur a même un peu de répit, car les billes suivantes ne tombent pas.



Dans l'image ci-dessus, on a déplacé une bille orange pour faire une ligne de 5. Lorsque la ligne a disparu, les *billes suivantes* ne sont pas tombées.

Le but du jeu est donc de survivre le plus longtemps possible : la partie s'arrête lorsque le plateau est recouvert de billes. Le joueur doit faire attention, car une bille peut être coincée par les autres, et ne pas pouvoir être déplaçable partout !

L'application a été divisée en deux parties distinctes :

— une partie graphique, qui permet d'interagir avec l'utilisateur en affichant la grille de billes et en permettant de déplacer celles-ci par de simples clics ;

— une partie modèle, qui contient la modélisation du jeu.

L'interface graphique du jeu vous est fournie par l'intermédiaire d'un `package` sous la forme d'un fichier `guilines.jar`. Votre travail sera donc de réaliser la modélisation de l'application, c'est-à-dire d'implémenter les différentes classes permettant de contenir les informations de la grille, puis d'agir sur ces données pour les mettre à jour, et effectuer ou refuser au joueur un déplacement de billes.

Exercice 1 : Démarrage

Récupérez le fichier `guilines.jar` sur Moodle. Ce fichier, contient le `package` de même nom, et dans le paquetage `guilines` se trouve l'ensemble des classes et interfaces nécessaires au bon fonctionnement de l'interface graphique. Enregistrez-le dans le répertoire `bin`.

Notez que l'exécution de la commande `jar -tvf guilines.jar` vous permettra de lister les différentes classes présentes dans le fichier `.jar`. Mais la seule chose qui vous importe pour le moment est de savoir qu'il contient l'interface `guilines.IJeuDesBilles` que vous allez devoir implémenter.

Exercice 2 : Consulter la documentation de l'interface

Puisque l'interface graphique est déjà implémentée, il ne s'agit pas de se lancer au hasard dans la conception du modèle. Les concepteurs de l'interface graphique ont fourni dans le package une interface java reprenant l'ensemble des méthodes nécessaires au bon fonctionnement du jeu (et surtout à une bonne communication interface graphique vers modèle). La documentation de cette interface vous est fourni sur Moodle. Consultez-la.

Votre travail va être maintenant de réaliser le contrat de cette interface java, c'est-à-dire d'écrire une classe java implémentant cette interface et les méthodes nécessaires au bon fonctionnement du jeu. Les exercices suivants vous guident pour cette implémentation.

Exercice 3 : Une première implémentation de l'interface

Nous allons dans un premier temps réaliser une implémentation simpliste du modèle. Celle-ci ne permettra que de représenter la grille de billes. Pour cela :

— Déclarez une classe `MonJeu.java` par exemple, qui déclare implémenter l'interface qui vous est fournie.

— L'interface `IJeuDesBilles`, puisqu'elle se trouve dans le package `guilines`, doit être importée `import guilines.IJeuDesBilles;`

— La grille de billes sera représentée par un tableau d'entiers à deux dimensions. Chaque entier représente la couleur de la bille présente dans la case de coordonnées `[i][j]`. La valeur `-1`

représente une case vide. Le package qui vous est fourni propose huit couleurs différentes, codées sur des entiers de 0 à 7.

— Une fois les attributs déclarés, écrivez les constructeurs et accesseurs nécessaires à l'implémentation de l'interface.

— Écrivez une méthode qui va initialiser la grille : toutes les cases sont vides, sauf 5 cases qui contiennent une valeur dans [0;7]. La classe `java.util.Random` vous fournira la méthode `nextInt(borne)` nécessaire à la génération aléatoire des couleurs et positions de ces billes. Pensez à utiliser la documentation de l'API ! Utilisez cette méthode pour initialiser la grille.

— La méthode `deplace` retourne toutes les cases modifiées si le déplacement souhaité par le joueur est possible. Le retour est une liste (`java.util.List`) de `java.awt.Point` (i.e. `List<Point>`). Consultez la documentation de la classe `Point` et de l'interface `List`. Dans un premier temps, nous allons refuser tout déplacement. La méthode `deplace` va donc systématiquement retourner une liste vide :

```
List<Point> maListe = new ArrayList<Point>();
return maListe;
```

N'oubliez pas les importations nécessaires !

— La méthode `getNouvellesCouleurs` attend les valeurs pour les couleurs des *billes suivantes*. Retournez un tableau avec 3 valeurs choisies arbitraitement.

— La méthode `partieFinie` retournera toujours faux.

Exercice 4 : Une classe principale

Après avoir débuté l'implémentation de l'interface, vous devez tester votre première implémentation. Il suffit pour cela de réaliser une classe `DemoLines` qui contiendra la méthode `main` dont la tâche est de :

— créer une instance du modèle, donc de `MonJeu`

— puis de lancer l'interface graphique `Lines` en lui passant le modèle en paramètre.

Voilà le résultat :

```
import guilines.Lines;
import guilines.IJeuDesBilles;

public class DemoLines {
    public static void main(String[] args) {
    IJeuDesBilles monJeu = new MonJeu();
    // creer la fenetre - on utilise l'interface graphique implementee dans guiline
    Lines fenetre = new Lines("LILines",monJeu);
    }
}// DemoLines
```

Pour la compilation et l'exécution, il faudra spécifier à la machine virtuelle dans la variable `classpath` qu'elle doit utiliser le fichier `guilines.jar` :

```
javac -d bin -cp src:bin:bin/guilines.jar src/*.java
java -cp bin:bin/guilines.jar DemoLines
```

Exercice 5 : Un peu d'organisation

Dans votre répertoire `tp4`, créez un répertoire `etape1`, et un répertoire `etape2`. Déplacez les répertoires `src`, `bin` et `doc` dans le répertoire `etape1`. Puis copiez-les dans `etape2`.

```
moi@maMachine:java/tp4$ mkdir etape1       # creer le dossier etape1
moi@maMachine:java/tp4$ ls                 # lister le contenu du rep. courant
```

```
bin    doc    etape1   src
moi@maMachine:java/tp4$ mv * etape1/       # mv : move - tout déplacer dans etape1
moi@maMachine:java/tp4$ ls
etape1
moi@maMachine:java/tp4$ ls etape1/
bin    doc    src
moi@maMachine:java/tp4$ mkdir etape2
moi@maMachine:java/tp4$ cp -rf etape1/* etape2/   # tout copier das etape2
```

Vous allez procéder par étapes pour la construction de ce jeu. Maintenant que a première version du jeu tourne, nous allons travailler sur l'étape 2.

```
moi@maMachine:java/tp4$ cd etape2
```

Attention : Quittez tous les fichiers ouverts avec votre éditeur (ce sont les fichiers qui sont dans `etape1`). Vous allez maintenant travailler avec ceux dans `etape2`.

Exercice 6 : On déplace une bille

Deuxième étape, pour se familiariser avec les listes. On accepte maintenant systématiquement le déplacement d'une bille, du moment qu'elle arrive sur une case vide. Modifiez la méthode `deplace`. La liste retournée doit maintenant contenir un point avec les coordonnées de la bille au départ et un point avec ses coordonnées à l'arrivée. La grille doit être à jour.

Lorsque cette étape fonctionne, créez un répertoire `etape3`, copiez-y tous vos fichiers. Nous allons passer à l'étape 3.

Exercice 7 : Contrôler le déroulement du jeu

Déplacer les billes, c'est rigolo... mais si personne ne s'occupe de l'alignement des billes, le jeu risque de vite devenir lassant. Par définition, il est du devoir du modèle de réaliser ce genre de contrôle sur le jeu. C'est donc dans votre partie ! À vous maintenant de vérifier s'il y a bien `nbBillesAlignees` après un déplacement, puis, si c'est le cas, de mettre à jour la grille. Attention, la bille déplacée peut-être au milieu de deux alignements (un vertical et un horizontal). N'oubliez pas que lorsqu'on a `nbBillesAlignees`, les nouvelles billes ne tombent pas.

Vous pourrez profiter de la possibilité qui est donnée au modèle de choisir le nombre de billes alignées (cela ne concerne pas l'interface graphique) : la règle du jeu c'est 5 billes alignées, mais 3 c'est plus agréable pour tester.

Exercice 8 : Pour aller plus loin...

Vous avez maintenant réalisé un jeu fonctionnel, mais basique et limité. À vous maintenant de l'améliorer (au moins pour respecter les règles du jeu !) :

- Pour le moment, votre modèle ne contient qu'une seule classe (`MonJeu`). Ce n'est pas une conception très objet...Implémentez une classe `Bille` qui modélisera un élément de la grille. `MonJeu` contiendra maintenant un tableau à deux dimensions de `Bille`. Cela vous aidera par la suite.
- Dans la version implémentée actuellement, un déplacement de bille est possible si la case d'arrivée est libre. Dans la version originale, le déplacement n'est possible que s'il existe un chemin de cases vides entre la bille choisie et la case d'arrivée (une bille ne se déplace qu'horizontalement ou verticalement). Modifiez le code pour prendre en compte cette règle ;
- La mise à jour de la grille graphique est assez lourde : pour le moment, à chaque déplacement, c'est la totalité de la grille qui est rafraîchie. Réfléchissez à une façon de ne fournir à l'interface graphique que les coordonnées des cases qui doivent réellement être redessinées (la bille déplacée, les nouvelles billes, ...)

— Calculez un score (2 points par bille supprimées)
— ...
Chaque item ci-dessus correspond à une nouvelle étape de développement.