Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2020 – 2021
DEV1 – JAVL

DEV1 – JAVL – Laboratoires Java

TD 13 – Mise en pratique : 3 in line

Dans ce travail, nous allons développer un jeu simple en essayant de reprendre tous les concepts de ce premier cours de développement.
Nous mettons en œuvre une approche *tests first* en utilisant JUnit. Nous utiliserons des tableaux à une dimension, des structures conditionnelles et répétitives. Nous gèrerons nos versions avec *git*. Nous prendrons également soin d'écrire la javadoc pour toutes nos méthodes.

Les codes sources et les solutions de ce TD se trouvent à l'adresse :
`https://git.esi-bru.be/dev1/labo-java/tree/master/td13-inline/`

Table des matières

Version du document
11 novembre 2020

Les règles du jeu

Ce jeu se joue seul. Le jeu est constitué d'un plateau pouvant recevoir vingt billes de différentes couleurs. Une valeur par couleur. Il existe cinq couleurs. Au début du jeu, deux billes de couleurs aléatoires sont placées à deux endroits libres aléatoires. Le plateau de jeu est à une dimension comme sur les illustrations.

À chaque tour de jeu, le joueur choisit quelle bille déplacer et à quel endroit la déplacer pour peu qu'il n'y ait pas plus d'une bille à *sauter*. À chaque déplacement, deux nouvelles billes de couleurs aléatoires sont placées à deux endroits libres aléatoires.

Lorsque trois billes de même couleurs sont alignées côte à côte, elles sont supprimées.

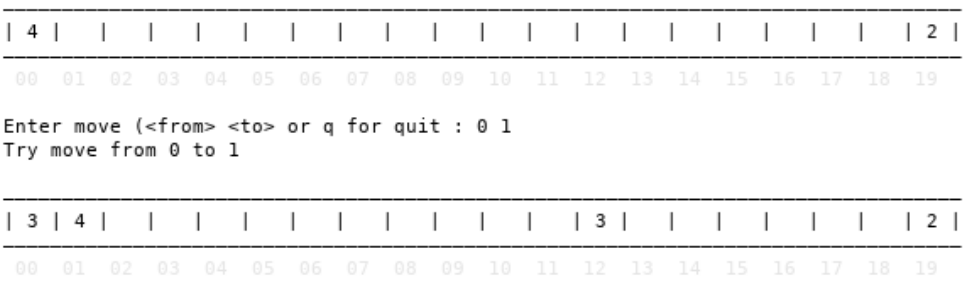
Le but du jeu est de jouer le plus longtemps possible.

Exemples

Exemple en fin de partie

- La bille ④ en position 7 peut être déplacée en position 5 ce qui aura pour effet de supprimer les billes en positions 3,4 et 5.
- La bille ⑤ en position 12 ne peut pas être déplacée en position 17 car il y a 3 billes entre la position de départ et la position d'arrivée. Par contre la bille ③ en position 14 peut être déplacée en position 11 par exemple pour pouvoir éventuellement déplacer la bille ⑤ au coup suivant.

Exemple d'un début de partie



- Le jeu est initialisé de manière aléatoire avec les deux billes ④ et ② aux positions respectives, 0 et 19 [1].
- Le joueur déplace la bille ④ de la position 0 à la position 1 et deux billes aux valeurs aléatoires sont ajoutées à des positions aléatoires ce qui donne le deuxième état du jeu pour cet exemple.

1. Le fait que ce soit au début et à la fin est du au hasard.

Exemple d'une suppression de 3 billes identiques

```
_____________________________________________________________________________
|   | 4 |   |   |   |   | 3 |   |   | 3 | 3 |   |   | 5 |   |   |   | 2 |
_____________________________________________________________________________
 00  01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19

Enter move (<from> <to> or q for quit : 7 10
Try move from 7 to 10

_____________________________________________________________________________
|   | 4 |   |   | 4 |   |   |   |   |   |   |   |   | 3 | 5 |   |   |   | 2 |
_____________________________________________________________________________
 00  01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19
```

- ▷ Le déplacement de la bille ③ en position 7 vers la position 10 est autorisé. Ce déplacement a pour effet d'aligner 3 billes de mêmes valeurs aux positions contigües 10, 11 et 12.
- ▷ Les billes étant : trois, de même valeur et contigües, elles disparaissent.
- ▷ Après chaque déplacement, deux nouvelles billes de valeur aléatoire et à une position aléatoire apparaissent. Il s'agit, dans cet exemple, des billes de valeur ③ et ④ aux positions respectives 14 et 4.

Choix techniques

1. Le plateau de jeu est représenté par un tableau d'entiers de taille 20.
2. Sur le plateau, les billes sont représentées par un entier numéroté de 1 à 5 correspondant à leur valeur.
3. Une case vide est représentée par l'entier 0.
4. La classe s'appellera `g12345.inline.InLine`. Elle fait donc partie du package `g12345.inline`, s'appelle `InLine` et se trouve dans un fichier `InLine.java`.

Pour l'exercice, vous aurez besoin d'un dépôt *git*. Vous pouvez travailler dans votre dépôt habituel contenant tous les exercices de vos tds ou en créer un pour l'occasion. À votre meilleure convenance.

1 Déplacer une bille (version 1)

La méthode `move` qui déplace une bille a besoin comme paramètres en entrée : le plateau de jeu représenté par un tableau d'entiers, un entier pour la position de départ et un entier pour la position d'arrivée.

Voici la signature de la méthode :

```
public static void move(int[] is, int from, int to)
```

Il s'agira bien d'écrire une méthode qui déplace un élément dans un tableau d'entiers.

Exercice 1 Écrire les tests

Commençons par demander à Netbeans de générer de code pour nous.

- ✎ Dans la classe `InLine` ajoutez la méthode `move` qui ne fait rien pour l'instant et d'un clic droit, demandez à Netbeans de générer les tests pour cette méthode.
- ✎ Écrivez des tests pour cette méthode.

Dans votre plan de tests, pensez à :

- ▷ un cas général où il n'y a pas de billes sur le chemin ;
- ▷ un cas général où il y a une bille sur le chemin ;
- ▷ un cas où la position de départ est hors tableau ;
- ▷ un cas où la position de départ ne contient pas de bille ;
- ▷ un cas où la position d'arrivée est hors tableau ;
- ▷ un cas où la position d'arrivée contient déjà une bille ;
- ▷ un cas où il y a plus d'une bille sur le chemin.

Lancez vos tests… et constatez qu'ils échouent tous. C'est normal. Continuons.

Exercice 2 Écrire la première version

- ✎ Écrivez la méthode `move` pour qu'elle déplace une bille sans se soucier de savoir combien de billes se trouvent entre la position de départ et la position d'arrivée. Votre méthode vérifiera par contre que le mouvement reste bien dans le tableau, qu'il y a une bille à la position de départ et que la position de destination n'est pas occupée.
  La méthode lance une `IllegalArgumentException` si les paramètres ne permettent pas de faire le déplacement.
- ✎ Lancez vos tests. Les six premiers tests doivent réussir. Le dernier sera pour plus tard.

Fin de fonctionnalité

Écrivez la javadoc si ce n'est déjà fait.
Lorsque votre code est propre, faites un commit avec un nom explicite.

2 Quelques méthodes simples

Exercice 3 Ajouter deux billes

- ✎ Écrivez une méthode `add2balls` ajoutant deux billes à deux endroits libres aléatoires du tableau de jeu.

Vous ne devez pas tester cette méthode mais bien écrire la javadoc.

Voici la signature de la méthode :

```
public static void add2balls(int[] is)
```

Remarque : Soyez attentif au fait que vous ne pourrez pas placer deux billes sur le tableau de jeu si celui-ci ne contient plus deux places libres. C'est donc mieux de vérifier à un moment donné.

Exercice 4 — Afficher le tableau de jeu

✎ Écrivez une méthode `display` qui affiche le tableau de jeu.

Affichez un espace lorsqu'il n'y a pas de bille et la valeur de la bille lorsqu'une bille est présente.

Vous ne devez pas tester cette méthode mais bien écrire la javadoc.

Voici la signature de la méthode :

```
public static void display(int[] is)
```

Fin de fonctionnalité

Écrivez la javadoc si ce n'est déjà fait.
Lorsque votre code est propre, faites un commit avec un nom explicite.

3 Retirer des billes

La méthode `remove3inline` parcourt le tableau de jeu et retire les billes dès lors qu'il y a trois billes de même couleurs côte à côte.

L'algorithme est un peu compliqué, il s'agit de chercher une suite de trois entiers identiques dans un tableau d'entiers. Mieux vaut y réfléchir un peu sur papier avant de se lancer tête baissée.

Voici la signature de la méthode :

```
public static void remove3inline(int[] is)
```

Exercice 5 — Écrire les tests

Redemandons à Netbeans de générer les tests pour nous.

✎ Dans la classe `InLine` ajoutez la méthode `remove3inline` qui ne fait rien pour l'instant et d'un clic droit, demandez à Netbeans de générer les tests pour cette méthode.

✎ Écrivez des tests pour cette méthode.

Cette fois, nous vous laissons réfléchir à votre plan de tests. Une fois vos tests écrits… ne les lancez pas. Ils échoueront ;-)

Exercice 6 — Écrire la méthode

Après avoir réfléchis à l'algorithme sur papier :

✎ écrivez la méthode `remove3inline`.

✎ Lancez vos tests.

Fin de fonctionnalité

Écrivez la javadoc si ce n'est déjà fait.
Lorsque votre code est propre, faites un commit avec un nom explicite.

4 Interface

Tout est en place pour écrire l'interface utilisateur et utilisatrice. Il s'agit de demander à l'utilisateur ou à l'utilisatrice quel mouvement iel veut faire et tant qu'iel ne désire pas s'arrêter, continuer.

Nous ne traiterons pas ici du tableau de jeu rempli ou ne permettant plus de faire un déplacement.

Exercice 7 Écrire l'interface

- Écrivez une méthode robuste de demande du mouvement ou de fin.
- Écrivez une méthode `main` permettant de jouer au jeu. En voici les étapes principales :
  - ▷ créer le tableau de jeu (le tableau d'entiers) ;
  - ▷ l'initialiser avec deux billes placées aléatoirement (via la méthode `add2balls`) ;
  - ▷ dans une boucle : lire le mouvement désiré, l'appliquer (via la méthode `move`) s'il est valide, ajouter deux billes placées aléatoirement (via la méthode `add2balls`, vérifier si trois billes sont en lignes et les supprimer (via la méthode `remove3inline`) [2] ;
  - ▷ signaler la fin du jeu et quitter.

Fin de fonctionnalité

Écrivez la javadoc si ce n'est déjà fait.
Lorsque votre code est propre, faites un commit avec un nom explicite.

5 Déplacer une bille (version 2)

Notre première version du déplacement ne vérifiait pas le nombre de bille entre la position de départ et la position d'arrivée. Il est temps d'ajouter cette dernière fonctionnalité.

Exercice 8 Écrire la méthode `move` (version 2)

- Revoyez la méthode `move` pour qu'elle gère correctement le déplacement.
- Relancez vos tests qui devraient tous réussir maintenant.

Fin de fonctionnalité

Écrivez la javadoc si ce n'est déjà fait.
Lorsque votre code est propre, faites un commit avec un nom explicite.

2. Le choix est fait d'ajouter d'abord les billes supplémentaires avant de supprimer les triplets pour le cas où une des deux billes supplémentaires serait une troisième.

6 Conclusion

Pour arriver au terme de cet exercice, c'est normal d'y consacrer deux heures au laboratoire et deux heures en dehors.

Bien qu'une solution soit fournie, nous espérons que vous avez pu mener à bien l'exercice sans la consulter. La solution fournie, n'est pas parfaite, le mieux pour vous est de demander un retour sur **votre** code à votre enseignant ou enseignante.