ITI 1521. Introduction à l'informatique II
Hiver 2020

Devoir 3
(Dernière modification le 10 mars 2020)

Échéance: 22 mars, 2020, 23 h 30

Objectifs d'apprentissage

- **Itérer** à travers différents états
- **Appliquer** la technique de l'indirection

Introduction

Pour le devoir précédent, nous avions une question optionnelle (la question 3) où il fallait supprimer les symétries dans notre liste de jeux, pour des jeux 3×3. Ici, nous allons résoudre ce problème dans le cas général, et utiliser l'itération pour le faire.

Étape 1 : Symétries et itérateurs

Lorsque nous avons créé notre liste de jeux pour la deuxième question du devoir 2, nous avons ajouté beaucoup de solutions qui étaient essentiellement identiques, simplement une symétrie des autres jeux déjà énumérés. Par exemple, comme nous l'avons vu dans la question trois du devoir 2, il n'y a que trois façons différentes de jouer le premier coup dans une partie de Tic-Tac-Toe de 3×3. Les 6 autres coups d'ouverture sont équivalents par symétrie.

Examinons les symétries dans une grille de $n \times m$. Supposons d'abord que $n \neq m$, c'est-à-dire que la grille n'est pas carrée. Dans le cas d'une grille non carrée, nous avons essentiellement deux symétries : le basculement vertical et le basculement horizontal (Figure 1).

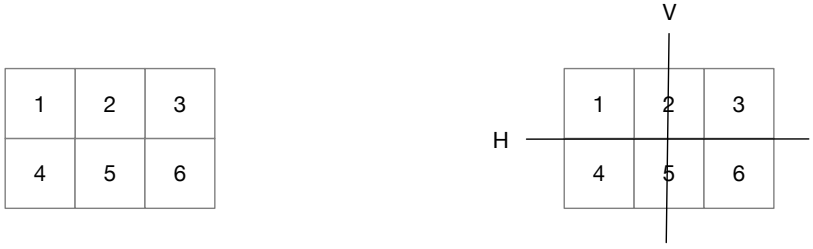

Figure 1 – Les grilles non carrées ont deux axes de symétrie.

Pour chaque grille non carrée $n \times m$, il y a jusqu'à trois grilles différentes mais symétriques : celle obtenue avec une symétrie verticale, celle obtenue avec une symétrie horizontale, et celle obtenue avec une combinaison des deux symétries (Figure 2).

Ce qui sera pratique, c'est qu'il est possible d'itérer à travers toutes ces symétries en appliquant des transformations répétées, par exemple la série symétrie horizontale, puis verticale, puis horizontale vous donnera les quatre grilles, comme le montre Figure 3.

Les choses sont un peu plus compliquées lorsque la grille est carrée. En plus des symétries horizontales et verticales, nous avons les deux diagonales, ainsi que la rotation (Figure 4). Chaque carré nous donne maintenant jusqu'à 7 autres grilles différentes mais symétriques, comme le montre la Figure 5.

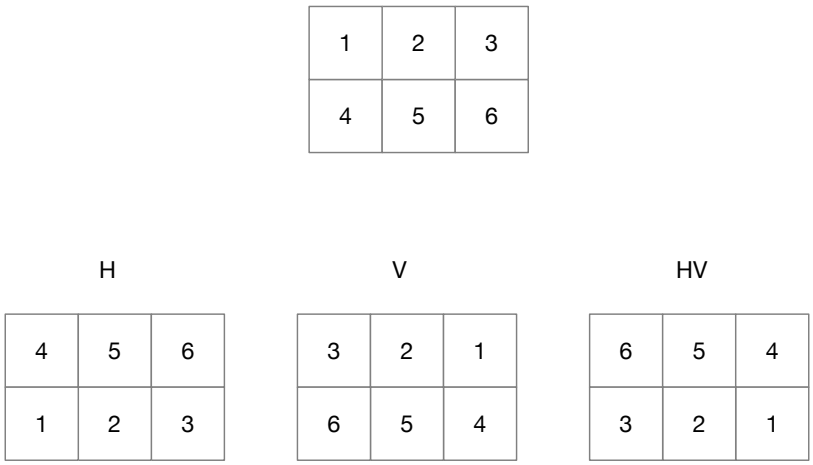

Figure 2 – Les grilles non carrées ont jusqu'à trois grilles symétriques équivalentes.

Encore une fois, il est possible d'itérer à travers les 8 différentes grilles («carrées») mais équivalents et symétriques, par exemple avec la séquence rotation-rotation-rotation-symétrie horizontale - rotation-rotation-rotation-rotation, comme le montre la figure 6.

Étape 2 : Modifier Utils.java

Il ressort de la discussion ci-dessus que la mise en œuvre de la symétrie horizontale, de la symétrie verticale et de la rotation de 90 degrés (dans le sens des aiguilles d'une montre) est suffisante pour obtenir toutes les grilles symétriques possibles, qu'ils soient carrés ou non. Nous allons donc ajouter trois méthodes dans **Utils.java** pour ce faire. Conformément à notre approche précédente, les grilles vont être mémorisées à l'aide d'un tableau unidimensionnel. Pour des raisons qui deviendront très bientôt évidentes, nous utiliserons un tableau d'entiers pour notre grille. Chacune des trois méthodes aura trois paramètres : le nombre de lignes et le nombre de colonnes de la grille, et une référence au tableau d'entiers représentant la grille. Vous devez implémenter les trois méthodes de classe dans la classe Utils.java, c'est-à-dire :

- **public static void horizontalFlip(int lines, int columns, int[] transformedBoard)** : effectue une symétrie horizontale sur les éléments de la grille **columns** × **lines** enregistrés dans le tableau référé par **transformedBoard**. Les éléments du tableau référencé par **transformedBoard** sont modifiés en conséquence (voir l'exemple ci-dessous).
- **public static void verticalFlip(int lines, int columns, int[] transformedBoard)** : effectue une symétrie verticale sur les éléments de la grille **columns** × **lines** enregistrés dans le tableau référencé par **transformedBoard**. Les éléments du tableau référencé par **transformedBoard** sont modifiés en conséquence (voir l'exemple ci-dessous).
- **public static void rotate(int lines, int columns, int[] transformedBoard)** : pivote de 90 degrés dans le sens des aiguilles d'une montre les éléments de la grille **columns** × **lines** enregistrés dans le tableau référencé par **transformedBoard**. Les éléments du tableau référencé par **transformedBoard** sont modifiés en conséquence (voir l'exemple ci-dessous).

Les trois méthodes doivent valider les données fournies et traiter tous les cas possibles selon les besoins.

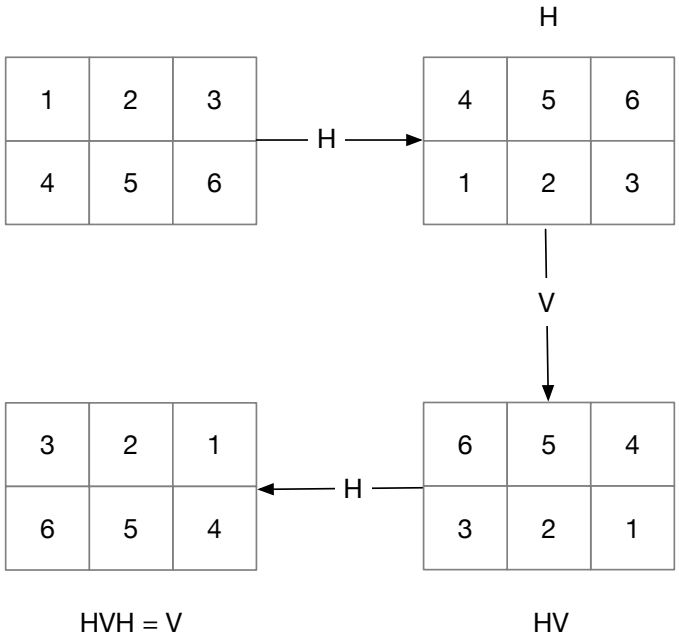

Figure 3 – Enumération de toutes les grilles symétriques non carrées.

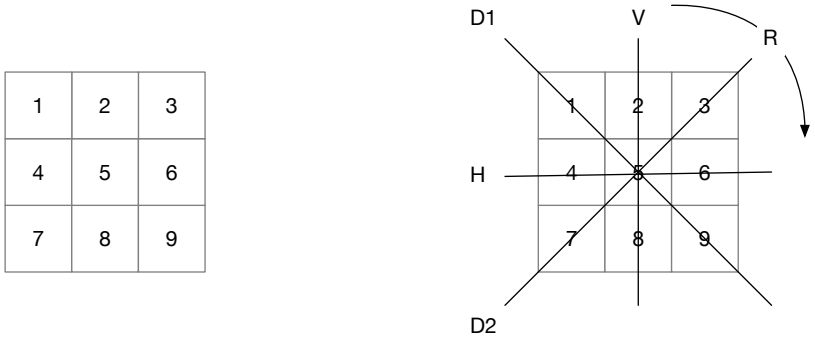

Figure 4 – Les grilles carrées ont 4 axes de symétrie, et peuvent également être tournées de 90 degrés.

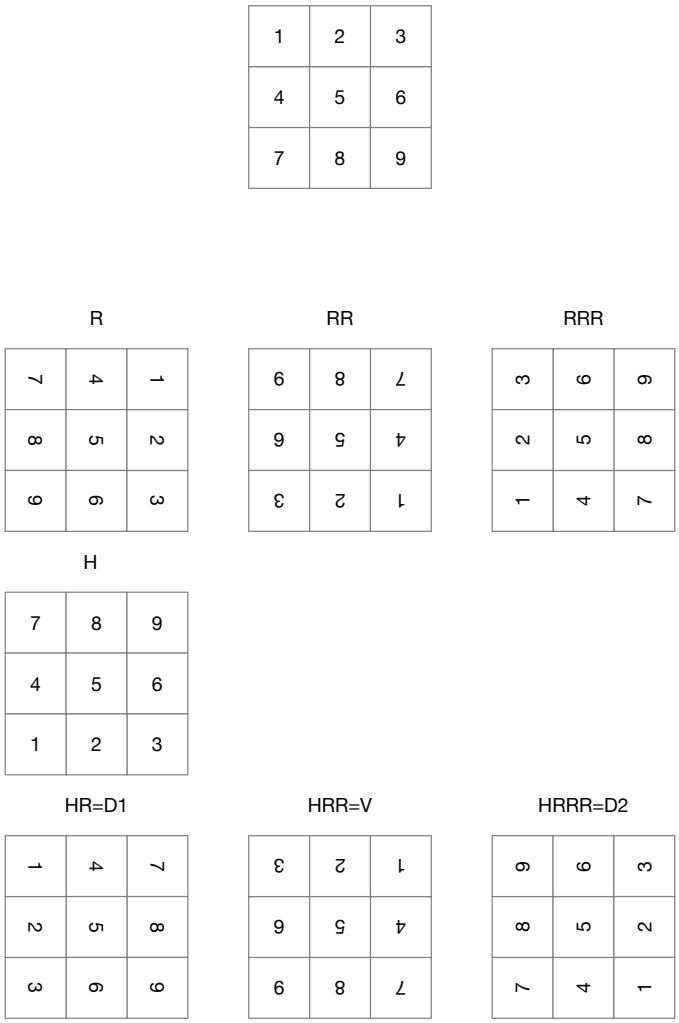

Figure 5 – Les grilles carrées ont 7 grilles symétriques équivalents.

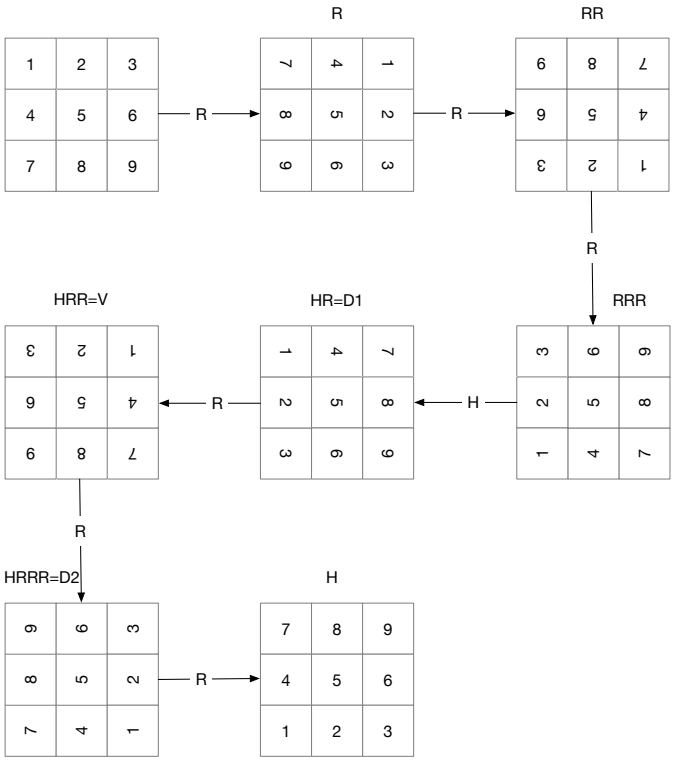

Figure 6 – Enumération de tous les grilles carrées symétriques.

La classe **Utils.java** est accompagnée des tests suivants :

```java
public class Utils {
    // code omis
    private static void test(int lines, int columns) {
        int[] test;
        test = new int[lines*columns];
        for(int i = 0; i < test.length; i++) {
                test[i] = i;
        }
        System.out.println("testing " + lines + " lines and " + columns + " columns.");
        System.out.println(java.util.Arrays.toString(test));
        horizontalFlip(lines,columns,test);
        System.out.println("HF => " + java.util.Arrays.toString(test));
        horizontalFlip(lines,columns,test);
        System.out.println("HF => " + java.util.Arrays.toString(test));
        verticalFlip(lines,columns,test);
        System.out.println("VF => " + java.util.Arrays.toString(test));
        verticalFlip(lines,columns,test);
        System.out.println("VF => " + java.util.Arrays.toString(test));
        if (lines == columns){
                for(int i = 0; i < 4; i++) {
                        rotate(lines,columns,test);
                        System.out.println("ROT => " + java.util.Arrays.toString(test));
                }
        }
    }

    public static void main(String[] args) {
        int[] test;
        int lines, columns;

        test(2,2);
        test(2,3);
        test(3,3);
        test(4,3);
        test(4,4);
    }
}
```

L'exécution du test ci-dessus devrait produire le résultat suivant :

```
> javac Utils.java
> java Utils
testing 2 lines and 2 columns.
[0, 1, 2, 3]
HF => [2, 3, 0, 1]
HF => [0, 1, 2, 3]
VF => [1, 0, 3, 2]
VF => [0, 1, 2, 3]
ROT => [2, 0, 3, 1]
ROT => [3, 2, 1, 0]
ROT => [1, 3, 0, 2]
ROT => [0, 1, 2, 3]
testing 2 lines and 3 columns.
[0, 1, 2, 3, 4, 5]
HF => [3, 4, 5, 0, 1, 2]
HF => [0, 1, 2, 3, 4, 5]
VF => [2, 1, 0, 5, 4, 3]
VF => [0, 1, 2, 3, 4, 5]
testing 3 lines and 3 columns.
[0, 1, 2, 3, 4, 5, 6, 7, 8]
HF => [6, 7, 8, 3, 4, 5, 0, 1, 2]
HF => [0, 1, 2, 3, 4, 5, 6, 7, 8]
VF => [2, 1, 0, 5, 4, 3, 8, 7, 6]
VF => [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

```
ROT => [6, 3, 0, 7, 4, 1, 8, 5, 2]
ROT => [8, 7, 6, 5, 4, 3, 2, 1, 0]
ROT => [2, 5, 8, 1, 4, 7, 0, 3, 6]
ROT => [0, 1, 2, 3, 4, 5, 6, 7, 8]
testing 4 lines and 3 columns.
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
HF => [9, 10, 11, 6, 7, 8, 3, 4, 5, 0, 1, 2]
HF => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
VF => [2, 1, 0, 5, 4, 3, 8, 7, 6, 11, 10, 9]
VF => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
testing 4 lines and 4 columns.
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
HF => [12, 13, 14, 15, 8, 9, 10, 11, 4, 5, 6, 7, 0, 1, 2, 3]
HF => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
VF => [3, 2, 1, 0, 7, 6, 5, 4, 11, 10, 9, 8, 15, 14, 13, 12]
VF => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
ROT => [12, 8, 4, 0, 13, 9, 5, 1, 14, 10, 6, 2, 15, 11, 7, 3]
ROT => [15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
ROT => [3, 7, 11, 15, 2, 6, 10, 14, 1, 5, 9, 13, 0, 4, 8, 12]
ROT => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>
```

Génération de tous les jeux non symétriques $n \times m$

Dans le devoir 2, nous avons déjà créé une méthode qui génère tous les jeux possibles pour une taille de grille donnée. Cette méthode ajoutait un jeu à la liste uniquement si le jeu n'était pas **égale** à un jeu qui était déjà là. De ce point de vue, il suffit donc de modifier légèrement cette méthode pour n'ajouter un jeu que s'il n'est pas *égale ou symétrique* à un jeu qui existe déjà. La nouvelle méthode d'instance **equalsWithSymmetry** de **TicTacToeGame** fournit cette information. Il suffit de l'implémenter, et la méthode **generateAllUniqueGames** qui est fournie dans la classe **ListOfGamesGenerator** générera la liste des listes que nous recherchons.

Indirection

Afin d'implémenter **equalsWithSymmetry** dans **TicTacToeGame**, nous devrons passer en boucle tous les jeux symétriques possibles pour voir si nous avons une correspondance. Bien sûr, nous pourrions simplement appliquer les symétries sur le jeu lui-même. Nous appliquerions ainsi des transformations sur la grille jusqu'à ce qu'elle corresponde à la grille du jeu auquel nous la comparons (auquel cas elle est symétrique) ou que nous n'ayons plus de jeux symétriques (auquel cas elle n'est pas symétrique).

Cependant, changer le tableau lui-même peut avoir des effets secondaires non désirés. Par exemple, imaginez que nous imprimons le jeu à l'utilisateur. Ce qui se passerait, c'est que, puisque le jeu est basculé vers des jeux symétriques équivalents, le jeu présenté à l'utilisateur pourrait être un jeu différent mais symétrique à chaque fois, ce qui ne serait clairement pas souhaitable.

C'est pourquoi nous allons introduire un niveau d'**indirection** pour calculer nos symétries. Le tableau lui-même restera inchangé, mais nous utiliserons un autre tableau qui fera correspondre les indices du tableau à leurs emplacements symétriques actuels. Nous utiliserons une variable d'instance, le tableau d'entier **transformedBoard** pour enregistrer l'indirection. Au départ, comme il n'y a pas de transformation, nous avons toujours **board[i]==board[transformedBoard[i]]**. Mais après avoir appliqué quelques symétries au jeu, **transformedBoard[i]** enregistre où l'index i du jeu est mis en correspondance dans la symétrie.

Par exemple, imaginez un jeu de 2×2. Avant toute symétrie, nous avons **transformedBoard[i]==i** pour $0 \leq i < 4$, et donc **board[i]==board[transformedBoard[i]]**. Si nous appliquons une symétrie verticale par exemple, l'indice 0 est mis en correspondance avec 1, 1 est mis en correspondance avec 0, 2 est mis en correspondance avec 3 et 3 est mis en correspondance avec 2. Le tableau désigné par **board** reste inchangé, et **transformedBoard** $= \{1, 0, 3, 2\}$. Donc si on veut savoir ce qui se trouve à l'index 1 du jeu après la symétrie, il faut accéder à **board[transformedBoard[1]]**, qui est donc **board[0]** avant la symétrie.

Itérer sur des tableaux symétriques

La deuxième chose à faire est de décider comment nous allons systématiquement passer en revue toutes les symétries d'un tableau donné. Chaque jeu a quatre ou huit positions symétriques selon qu'il est carré ou non. Strictement parlant, il y a parfois moins de choix différents car certaines des symétries sont en fait identiques, mais nous ne nous inquiéterons pas de cela ici. Nous proposons un mécanisme pratique permettant d'itérer toutes les positions possibles, en utilisant de nouvelles méthodes d'instance dans **TicTacToeGame** : **hasNext**, **next**, et **reset**.

- **hasNext()** retourne **true** si et seulement si un appel à la méthode **next** réussirait, et **false** sinon.
- Chaque appel à la méthode **next** trouve le prochain tableau symétrique équivalent dans **transformedBoard**. Il lance une exception **IllegalStateException** si le nombre d'appels à **next** est plus grand que le nombre de symétries existantes entre deux appels à la méthode **reset**.
- La méthode **reset** remet **transformedBoard** dans son éta original, correspondant au tableau inchangé.

Le programme Java suivant illustre l'utilisation souhaitée pour **hasNext**, **next**, et **reset** (la méthode **toStringTransformed** de **TicTacToeGame** retourne une représentation en chaîne du jeu transformé).

```
1  public class Test {
2
3          public static void printTest(TicTacToeGame g) {
4                  g.reset();
5                  while (g.hasNext()) {
6                      g.next();
7                      System.out.println(g.toStringTransformed());
8                  }
9                  System.out.println("reset:");
10                 g.reset();
11                 while (g.hasNext()) {
12                     g.next();
13                     System.out.println(g.toStringTransformed());
14                 }
15         }
16         public static void main(String[] args) {
17
18                 TicTacToeGame g;
19
20                 System.out.println("Test on a 3x3 game");
21                 g = new TicTacToeGame();
22                 g.play(0);
23                 g.play(2);
24                 g.play(3);
25                 printTest(g);
26                 System.out.println("Test on a 5x4 game");
27                 g = new TicTacToeGame(4,5);
28                 g.play(0);
29                 g.play(2);
30                 g.play(3);
31                 printTest(g);
32          }
33 }
```

L'exécution produit le résultat suivant :

```
> java Test
Test on a 3x3 game
 X |   | O
-----------
 X |   |
-----------
   |   |

   | X | X
-----------
   |   |
-----------
```

```
   |   | O

   |   |
-----------
   |   | X
-----------
 O |   | X

 O |   |
-----------
   |   |
-----------
 X | X |

 X | X |
-----------
   |   |
-----------
 O |   |

 O |   | X
-----------
   |   | X
-----------
   |   |

   |   | O
-----------
   |   |
-----------
   | X | X

   |   |
-----------
 X |   |
-----------
 X |   | O

reset:
 X |   | O
-----------
 X |   |
-----------
   |   |

   | X | X
-----------
   |   |
-----------
   |   | O

   |   |
-----------
   |   | X
-----------
 O |   | X
```

```
 O |   |
-----------
   |   |
-----------
 X | X |

 X | X |
-----------
   |   |
-----------
 O |   |

 O |   | X
-----------
   |   | X
-----------
   |   |

   |   | O
-----------
   |   |
-----------
   | X | X

   |   |
-----------
 X |   |
-----------
 X |   | O

Test on a 5x4 game
 X |   | O | X |
-------------------
   |   |   |   |
-------------------
   |   |   |   |
-------------------
   |   |   |   |

   |   |   |   |
-------------------
   |   |   |   |
-------------------
   |   |   |   |
-------------------
 X |   | O | X |

   |   |   |   |
-------------------
   |   |   |   |
-------------------
   |   |   |   |
-------------------
   | X | O |   | X

   | X | O |   | X
-------------------
```

```
   |   |   |   |
------------------
   |   |   |   |
------------------
   |   |   |   |

reset:
 X |   | 0 | X |
------------------
   |   |   |   |
------------------
   |   |   |   |
------------------
   |   |   |   |

   |   |   |   |
------------------
   |   |   |   |
------------------
   |   |   |   |
------------------
 X |   | 0 | X |

   |   |   |   |
------------------
   |   |   |   |
------------------
   |   |   |   |
------------------
   | X | 0 |   | X

   | X | 0 |   | X
------------------
   |   |   |   |
------------------
   |   |   |   |
------------------
   |   |   |   |
>
```

Comme indiqué plus haut, il suffit de trois transformations pour itérer à travers tous les jeux symétriques : la symétrie verticale (**VSYM**), la symétrie horizontale (**HSYM**) et la rotation (**ROT**). Nous devons également pouvoir réinitialiser le jeu à son état initial, la transformation de l'identité (**ID**).

A la suite d'un appel à la méthode **reset**, chaque appel à la méthode **next** modifie l'orientation du jeu selon la liste d'opérations suivante :

- pour une grille non carrée : **ID**, **HSYM**,**VSYM**, **HSYM**
- pour une grille carrée : **ID**, **ROT**, **ROT**, **ROT**, **HSYM** ,**ROT**, **ROT**, **ROT**

Vous devez ajouter toutes les variables d'instance nécessaires pour implémenter les méthodes : **hasNext**, **next** et **reset**.

Enfin, vous devez implémenter la méthode d'instance **boolean equalsWithSymmetry(TicTacToeGame other)**, qui retourne **true** si et seulement si **cette** instance de **TicTacToeGame** et **autre** sont identiques par symétrie.

La classe **ListOfGamesGenerator** a une méthode **generateAllUniqueGames** (déjà implémentée) qui s'appuie sur **equalsWithSymmetry** pour générer la liste des jeux. Voici quelques exemples :

```
> java TicTacToe
************************************************************
```

```
*                                                          *
*                                                          *
*                                                          *
*                                                          *
************************************************************

======= level 0 =======: 1 element(s) (1 still playing)
======= level 1 =======: 3 element(s) (3 still playing)
======= level 2 =======: 12 element(s) (12 still playing)
======= level 3 =======: 38 element(s) (38 still playing)
======= level 4 =======: 108 element(s) (108 still playing)
======= level 5 =======: 174 element(s) (153 still playing)
======= level 6 =======: 204 element(s) (183 still playing)
======= level 7 =======: 153 element(s) (95 still playing)
======= level 8 =======: 57 element(s) (34 still playing)
======= level 9 =======: 15 element(s) (0 still playing)
that's 765 games
91  won by X
44  won by O
3 draw
It took 116ms.
> java TicTacToe 2 2 2
************************************************************
*                                                          *
*                                                          *
*                                                          *
*                                                          *
************************************************************

======= level 0 =======: 1 element(s) (1 still playing)
======= level 1 =======: 1 element(s) (1 still playing)
======= level 2 =======: 2 element(s) (2 still playing)
======= level 3 =======: 2 element(s) (0 still playing)
that's 6 games
2  won by X
0  won by O
0 draw
It took 38ms.
> java TicTacToe 2 2 3
************************************************************
*                                                          *
*                                                          *
*                                                          *
*                                                          *
************************************************************

======= level 0 =======: 1 element(s) (1 still playing)
======= level 1 =======: 1 element(s) (1 still playing)
======= level 2 =======: 2 element(s) (2 still playing)
======= level 3 =======: 2 element(s) (2 still playing)
======= level 4 =======: 2 element(s) (0 still playing)
that's 8 games
0  won by X
0  won by O
2 draw
It took 40ms.
> java TicTacToe 4 3 2
```

```
************************************************************
*                                                          *
*                                                          *
*                                                          *
*                                                          *
************************************************************

======= level 0 =======: 1 element(s) (1 still playing)
======= level 1 =======: 4 element(s) (4 still playing)
======= level 2 =======: 36 element(s) (36 still playing)
======= level 3 =======: 172 element(s) (98 still playing)
======= level 4 =======: 439 element(s) (245 still playing)
======= level 5 =======: 1006 element(s) (167 still playing)
======= level 6 =======: 639 element(s) (91 still playing)
======= level 7 =======: 379 element(s) (13 still playing)
======= level 8 =======: 50 element(s) (2 still playing)
======= level 9 =======: 6 element(s) (0 still playing)
that's 2732 games
1285  won by X
790  won by O
0 draw
It took 235ms.
> java TicTacToe 4 4 2
************************************************************
*                                                          *
*                                                          *
*                                                          *
*                                                          *
************************************************************

======= level 0 =======: 1 element(s) (1 still playing)
======= level 1 =======: 3 element(s) (3 still playing)
======= level 2 =======: 33 element(s) (33 still playing)
======= level 3 =======: 219 element(s) (141 still playing)
======= level 4 =======: 913 element(s) (587 still playing)
======= level 5 =======: 3338 element(s) (883 still playing)
======= level 6 =======: 4702 element(s) (1217 still playing)
======= level 7 =======: 7048 element(s) (511 still playing)
======= level 8 =======: 2724 element(s) (194 still playing)
======= level 9 =======: 1119 element(s) (0 still playing)
that's 20100 games
10189  won by X
6341  won by O
0 draw
It took 23865ms.
>
```

Intégrité académique

Cette partie du devoir vise à sensibiliser les étudiants au plagiat et à l'intégrité académique. Veuillez lire les documents suivants.

- https://www.uottawa.ca/administration-et-gouvernance/reglement-scolaire-14-autres-informations-impor
- https://www.uottawa.ca/vice-recteur-etudes/integrite-etudes

Les cas de plagiat seront traités conformément au règlement de l'université. En soumettant ce travail, vous reconnaissez :

1. J'ai lu le règlement académique sur la fraude académique.
2. Je comprends les conséquences du plagiat.
3. À l'exception du code source fourni par les instructeurs pour ce cours, tout le code source est le mien.
4. Je n'ai collaboré avec aucune autre personne, à l'exception de mon partenaire dans le cas d'un travail d'équipe.
    - Si vous avez collaboré avec d'autres personnes ou obtenu le code source sur le Web, veuillez alors indiquer le nom de vos collaborateurs ou la source de l'information, ainsi que la nature de la collaboration. Mettez ces informations dans le fichier README.txt soumis. Des points seront déduits proportionnellement au niveau de l'aide fournie (de 0 à 100%).

Directives

- Suivez toutes les directives disponibles sur la page Web Consignes pour la remise de tous vos travaux pratiques.
- Soumettez votre devoir par le biais du système de soumission en ligne campus virtuel.
- Vous devez préférablement réaliser le travail en équipe de deux, mais vous pouvez également le faire individuellement. Toutefois, si vous effectuez le travail en équipe, votre devoir ne doit être soumis qu'une seule fois sur Brightspace. Si les deux partenaires soumettent le devoir, une pénalité de 30 % pour le devoir 3 et de 40 % pour le devoir 4 sera appliquée.
- Vous devez utiliser les classes de modèles fournies ci-dessous.
- Si vous ne suivez pas les instructions, votre programme fera échouer les tests automatisés et, par conséquent, votre devoir ne sera pas noté.
- Nous utiliserons un outil automatisé pour comparer toutes les travaux les uns par rapport aux autres (y compris les sections française et anglaise). Les soumissions qui sont signalées par cet outil recevront la note 0.
- Il vous incombe de vous assurer que BrightSpace a bien reçu votre travail. Les soumissions tardives ne seront pas notées.

Fichiers

Vous devez soumettre un fichier **zip** (aucun autre format de fichier ne sera accepté). Le nom du répertoire principal doit avoir le format suivant : **a3_3000000_3000001**, où 3000000 et 3000001 sont les numéros d'étudiant des membres de l'équipe qui soumettent le travail (il suffit de répéter le même numéro si votre équipe compte un seul membre). Le nom du dossier commence par la lettre "a" (minuscule), suivie du numéro du devoir, ici 3. Les parties sont séparées par le trait de soulignement (et non par le trait d'union). Il n'y a pas d'espace dans le nom du dossier. L'archive a3_3000000_3000001.zip contient les fichiers que vous pouvez utiliser comme point de départ. Votre soumission doit contenir les fichiers suivants.

- README.txt
    - Un fichier texte qui contient les noms des deux partenaires pour les devoirs, leurs numéros d'étudiant, la section et une courte description du devoir (une ou deux lignes).
- CellValue.java
- GameState.java
- ListOfGamesGenerator.java
- StudentInfo.java
- Test.java
- TicTacToe.java
- TicTacToeGame.java
- Transformation.java
- Utils.java

Questions

Pour toutes vos questions, veuillez consulter le site web de la Piazza du cours :

- https://piazza.com/uottawa.ca/winter2020/iti1521/home

Dernière modification : 10 mars 2020