M1 MIAGE
2021-2022

Infinity Loops

Contents

M1 MIAGE
2021-2022

Membres :

- Sylvain THOR
- Erwan MOALIC
- Léonard MOREAU

Méthodes :

Checker

`public static Grid buildGrid(String inputFile)`

Cette méthode permet prend un fichier avec les informations d'une grille en argument (tailles, pièces) et retourne une grille de type Grid.

Pour résumer, elle permet de transformer un fichier de données en une grille.

`public static boolean isSolution(Grid grid)`

Prend une grille en argument et retourne true si la grille est solved, sinon retourne false.

Orientation

`public static Orientation getOrifromValue(int orientationValue)`

Prend une orientation sous la forme d'un entier en paramètre et renvoie l'orientation (de l'énumération) correspondante.

`public int getOriValueFromType(PieceType pt)`

Prend un type de pièce en paramètre et renvoie son orientation sous la forme d'un entier.

`public Orientation turn90()`

Effectue une rotation de 90° sur l'orientation à partir de laquelle la méthode a été appelée.

`public Orientation getOpposedOrientation()`

Renvoie l'orientation opposée.

`public int[] getOpposedPieceCoordinates(Piece p)`

Renvoie les coordonnées de la pièce opposée à l'orientation de la pièce dans un tableau de taille 2 (tab[0] = y, tab[1] = x).

PieceType

`public static PieceType getTypefromValue(int typeValue)`

Prend un type de pièce sous la forme d'un entier et le renvoie sous la forme d'un PieceType.

`public int getTypeValue()`

Retourne le type de la pièce sur laquelle la méthode a été appelée sous la forme d'un entier.

`public LinkedList<Orientation> setConnectorsList(Orientation ori)`

Prend une Orientation en paramètre et retourne la liste des connecteurs de la pièce sur laquelle la méthode a été appelée. Pour trouver cette liste, la méthode identifie d'abord l'orientation, puis le type de la pièce, et ajoute les connecteurs correspondants à la liste.

Par exemple : si on appelle la méthode sur un TTYPE et qu'il est orienté vers le NORTH, alors les Orientations EAST, NORTH et WEST seront ajoutés à la liste chaînée.

`public ArrayList<Orientation> getListOfPossibleOri()`

Retourne la liste des orientations possibles d'une pièce.

Par exemple : si on appelle la méthode sur un ONECONN, alors les orientations NORTH, WEST, SOUTH et EAST seront ajoutées à l'ArrayList. Cependant si on appelle la méthode sur on appelle la méthode sur une BAR, seulement les Orientations NORTH et EAST seront ajoutées à l'ArrayList car SOUTH et WEST sont déjà représentées par NORTH et EAST.

`public int getNbConnectors()`

Retourne le nombre de connecteurs d'une pièce en fonction de son type.

Generator :

`public static void generateLevel(String fileName, Grid inputGrid)`

Prend une grille vide avec une certaine taille en paramètre, la remplit de pièces, puis la stocke dans fichier passé en paramètre.

Dans un premier temps, la méthode vérifie si le fichier existe déjà, si oui elle lèvera une exception. Sinon elle crée un nouveau fichier dans lequel sera stocké la grille.

Dans un second temps, la méthode crée une HashMap qui permet de relier une paire de booléens avec une liste de paires de types de pièces et d'entiers :

```
HashMap<Pair<Boolean, Boolean>, List<Pair<PieceType,List<Integer>>>>
```

Pour comprendre à quoi sert la HashMap, il faut comprendre ce qu'on cherche à faire pour générer une grille.

Pour générer la grille, on cherche à ajouter les pièces une par une. Pour cela, on parcourt la grille de gauche à droite puis de haut en bas en commençant par le coin supérieur gauche. De cette manière, on identifie pour chaque case quelles sont les types de pièces de possibles et leurs orientations possibles. Comme on parcourt de gauche à droite et de haut en bas, on sait exactement si la case voisine située à gauche et la case voisine située en haut attendent un connecteur. Cependant, on ne sait pas si la case voisine située à droite (sauf pour la dernière colonne) et la case voisine située en bas (sauf si pour la dernière ligne) attendent un connecteur. Ainsi, nous ne pouvons uniquement avoir l'information si NORTH et EAST attendent ou non un connecteur. C'est pourquoi notre HashMap associe une paire de booléens à une liste de paires.

Maintenant expliquons la liste de paires. Peu importe où nous nous situons, la paire de booléens nous permet de savoir tout ce dont on a besoin. Lorsque nous sommes sur une case qui n'attend ni un connecteur en haut ni un connecteur en bas, nos 2 booléens sont à false. Quand on sait que ces 2 booléens sont à false, on sait qu'il n'est pas possible d'avoir une pièce de type FOURCONN, TTYPE, etc. De cette manière nous ajoutons simplement les types de pièces possibles ainsi que leurs orientations dans la HashMap, c'est pourquoi le deuxième argument de la HashMap est une liste contenant les PieceType qui sont chacun associés à la liste de leurs orientations possibles (compte tenu de 2 booléens) :

```
List<Pair<PieceType,List<Integer>>
```

Il ne reste plus qu'à tirer au hasard pour chaque case, un type de pièce possible et une orientation possible. Une fois la pièce choisie, il ne reste plus qu'à écrire la ligne correspondante dans le fichier sans oublier qu'il faut tirer une orientation au hasard pour pas que le fichier contienne une grille déjà résolue.

```
public static void generateFileFromGrid(String fileName, Grid inputGrid)
```

Prend une grille en paramètre et la stocke dans un fichier.

GUI :

L'interface graphique est interactive, il est possible de jouer en cliquant sur les pièces qu'on veut tourner (voir section Usage).

Le listener récupère les coordonnées du clic de l'utilisateur, puis divise la fenêtre en width*height cases afin d'identifier dans quelle case l'utilisateur a cliqué.

Lorsque le clic est effectué, la pièce correspondante est tournée d'une rotation de 90° dans le sens des aiguilles d'une montre.

Lorsque la grille est résolue, un message de victoire apparaît dans le terminal.

M1 MIAGE
2021-2022

Solver :

2 méthodes de solving ont été implémentées :

- SolveExhaustiveSearch()
- SolveNonExhaustiveSearch()

Ces 2 méthodes utilisent respectivement une méthode de backtracking permettant de revenir en arrière en cas d'erreur. Elles sont inspirées des algorithmes de parcours en profondeur (Depth-First Search).

Dans les deux algorithmes de solving, le solveur va visiter une par une de gauche à droite puis de haut en bas les cases de notre grille. À chaque case, le solveur va choisir une orientation possible de la case, puis passer à la prochaine, en cas d'erreur (plus aucune solution possible), grâce au backtracking, le solveur reviendra en arrière tester pour la dernière case ayant une orientation pas encore testée l'orientation suivante, et reprendre son parcours. Ainsi, l'algorithme est récursif et respecte une logique LIFO (Last In Last Out).

```
public static boolean solveExhaustiveSearch(Grid grid)
```

Et

```
public static boolean backtrack_v1(Grid grid, Piece p)
```

Ce premier algorithme va tester pour chaque case chaque orientation jusqu'à la fin, sans prendre en compte son environnement. Cet algorithme n'est pas du tout optimisé ce qui fait qu'il n'est même pas capable de solve une grille ayant une taille supérieure à 3x3.

```
public static boolean solveNonExhaustiveSearch(Grid grid)
```

Et

```
public static boolean backtrack_v2(Grid grid, Piece p, HashMap<Pair<Boolean, Boolean>, List<Pair<PieceType,List<Integer>>>> hm)
```

Contrairement au premier, cet algorithme va éliminer chaque orientation inutile à tester, et ce grâce à la HashMap que nous avons utilisée pour le Generator. Le solver va reprendre le même principe des booléens et va uniquement tester les orientations possibles pour la case courante, ce qui va économiser un grand nombre d'opérations et permettre de gagner en complexité.

Usage :

```
-g 10x10 -o outputfile
```

Génère une grille de taille 10x10.

```
-s outputfile -o solvedfile
```

Solve l'outputfile et stocke la grille solved dans solvedfile. Par défaut l'algorithme de solving est le solveNonExhaustiveSearch(). Pour changer d'algorithme, il suffit de changer le nom de la méthode appelée dans le Main.

`-c solvedfile`

Vérifie si la grille contenue dans solvedfile est bien solved. Ouvre aussi l'interface graphique pour visualiser la grille.

`-p outputfile`

Une option –p a été rajoutée pour utiliser l'interface graphique. Elle prend un fichier en argument, et écrit un message dans la console lorsque l'utilisateur gagne la partie. Cette option n'est pas seulement destinée à jouer mais elle peut servir à visualiser une grille (contenue dans outputfile).