INF2010 - Structures de données et algorithmes

Travail Pratique #6
Ensembles disjoints et parcours de graphe
Hiver 2016

Contexte du laboratoire

Les ensembles disjoints permettent de générer de façon élégante des labyrinthes aléatoires. C'est précisément ce que nous ferons dans ce laboratoire. Le labyrinthe sera modélisé au moyen d'un graphe.

Les sources qui vous sont remises incluent les classes: Main.java, DisjointSet.java, MyPannel.java, MyFrame.java et Maze.java. Tout le travail à réaliser se situe dans Maze.java. Vous ne devez en aucun cas modifier les autres classes. Maze.java utilise un objet de type `DisjointSet` (une implémentation des ensembles disjoints) pour construire des labyrinthes. Les autres classes utilisent Maze.java pour afficher le résultat à l'écran.

Exercice 1 : Génération du labyrinthe (3 points)

Maze dispose d'un graphe modélisation le labyrinthe, d'une liste des murs dans le graphe (un mur est identifié par les pièces qu'il sépare) ainsi que d'une liste identifiant le chemin reliant le début (le coin haut à gauche du labyrinthe) à la sortie (le coin en bas à droite du labyrinthe). La fabrication du labyrinthe se fait au moyen de `disjointSet` qui sert à savoir si deux pièces sont connectées. Deux pièces sont connectées si elles sont dans le même ensemble. Il suffit d'appeler `areInSameSet(…)` de la classe `disjointSet` pour le savoir.

Le constructeur de Maze procède alors en créant tous les murs possibles (liste `walls`) et le graphe qui lui est associé (`mazeGraph`). Il permute ensuite de manière aléatoire les murs du labyrinthe (on permute aléatoirement les éléments de `walls`) qui seront ensuite « détruits » un à un.

Vous devez donc compléter la méthode `generate()` qui détruira judicieusement certains murs du labyrinthe. Pour ce faire, on parcourt les éléments de `walls`. Si un mur sépare deux pièces non connectées (utilisez `areInSameSet(…)` de `DisjointSet` pour le savoir), il vous faudra détruire ce mur : pour ce faire, vous devez appeler la méthode `union(…)` de la class `DisjointSet` sur les deux pièces ainsi que la méthode `connect(…)` de `graphMaze` sur les deux mêmes pièces. Si les deux pièces sont déjà connectées, le mur est maintenu.

Pour que le labyrinthe s'affiche correctement, seuls les murs qui ne sont pas détruits doivent demeurer présents dans la liste `walls` à la fin de l'appel à `generate()`. Veuillez faire le nécessaire pour respecter cette contrainte.

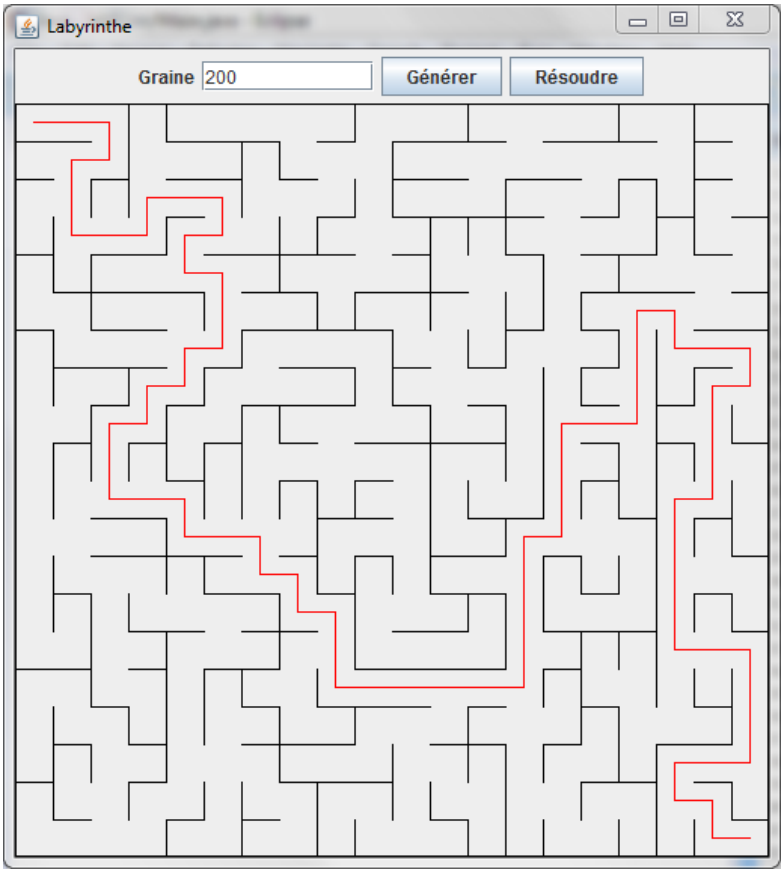
À la fin de cet exercice, cliquer sur le bouton Générer permettra de générer un nouveau graphe. Changez la valeur de la Gaine (seed du générateur aléatoire) pour voir le graphe changer.

Note : chaque pièce – instance de la classe `Room` – du graphe possède une liste d'adjacence nommée `neighbours`; la classe `Room` possède une variable `id` qui identifie par un numéro les sommets du graphe et un membre `source` qui servira à identifier le chemin quand il s'agira de résoudre le labyrinthe (exercice 2). Vous n'avez pas à altérer `id` et `source` dans cette partie du travail.

Exercice 2 : Résolution du labyrinthe (2 points)

La méthode `solve()` résout le labyrinthe en utilisant l'algorithme du plus court chemin (qui exploite un parcours BFS du graphe). Cette méthode initialisera donc les membres `distance` et `source` de l'ensemble des pièces du graphe. Il générera ensuite une liste `path` (membre de la classe `Maze`) contenant l'`id` des salles à parcourir pour résoudre le labyrinthe (vous pouvez mettre les pièces du chemin en ordre inverse). Pour vous aider, vous pouvez appeler la méthode `getFinish()` qui renvoie la pièce d'arrivée.

Un clic sur le bouton Résoudre dans la fenêtre appellera la méthode `solve()` que vous venez d'implémenter et indiquera le chemin d'un trait rouge.



Instructions pour la remise :

Le travail doit être fait par équipe de 2 personnes et doit être remis via Moodle au plus tard le 15 avril avant 23h50.

Veuillez envoyer vos fichiers .java **seulement**, dans un **seul répertoire**, le tout dans une archive de type *.**zip** (et seulement **zip**, pas de ~~**rar**~~, ~~**7z**~~, etc) qui portera le nom :

inf2010_lab6_MatriculeX_MatriculeY.zip, où MatriculeX < MatriculeY.

Les travaux en retard seront pénalisés de 20 % par jour de retard. Aucun travail ne sera accepté après 4 jours de retard. Si votre dépôt ne respecte pas la nomenclature définie ci-dessus, 0.5 point de pénalité sera appliqué.