VISITE ÉTUDIANT·E·S EPTM

A* pathfinding

Informatique et systèmes de communication ISC
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

1ère partie – Afficher le labyrinthe

Dans cette première partie, vous allez prendre connaissance du code source qui a été préparé pour vous ainsi que de la structure du code Java correspondante. À la fin de cette partie, vous aurez de quoi afficher un labyrinthe sous forme de texte et d'image. Vous pourrez également afficher sur l'image le chemin de n'importe quel point vers la sortie.

Tâche 0 – Importation du projet

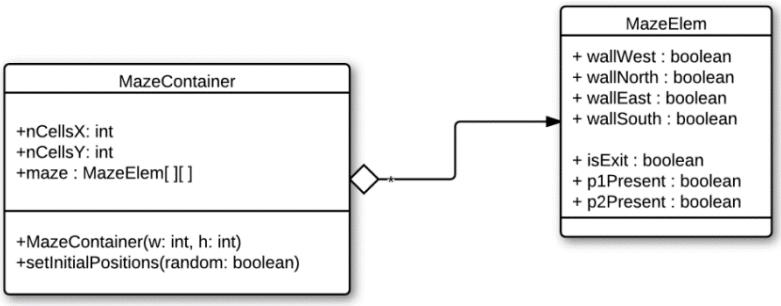
Nous avons préparé pour vous un projet *Github* contenant les éléments de base du projet. Pour l'importer :

- Lancez *IntelliJ*, faites *Get from VCS* ou *File → New → Project from version control*
- Entrez le nom de la repository https://github.com/ISC-HEI/maze-students et validez avec *Clone*
- S'il vous est demandé, choisissez un JDK 11 dans la liste.

Tâche 1 – Dessin du labyrinthe sous forme de texte

Cette première tâche va vous permettre de comprendre comment est codé le labyrinthe dans le projet en vous demandant de l'afficher sous forme de texte dans la console. Cela vous sera car nous réutiliserons toujours la même structure de donnée.

1. Ouvrez le fichier `TextDisplay.java` qui se trouve dans `maze/display` et observez le fichier.
2. Remarquez qu'il y a une méthode `main` dans le fichier qui constitue un point d'entrée dans le programme.
3. Dans le `main`, un `MazeContainer` est créé. Cette classe fabrique le labyrinthe et vous permet d'y accéder sous forme d'un tableau bi-dimensionnel de `MazeElem`. Dans la classe se trouvent également les informations sur les dimensions du labyrinthe (champs `nCellsX`, `nCellsY`).
4. Ouvrez la classe `MazeElem` et observez-la. Cette dernière classe contient déjà toutes les informations concernant ce qui se trouve dans une case du labyrinthe (murs, joueurs, sortie…).
5. Le diagramme de classes UML, c'est-à-dire comment les classes sont organisées, est le suivant :



6. Lancez le programme correspondant (clic droit sur *TextDisplay → Run*, là où se trouve la flèche verte).
7. Changez le programme de sorte qu'un labyrinthe de taille 5x5 soit affiché.
8. Comme vous le voyez, le labyrinthe n'est pas affiché complètement, seulement les croisements et les murs du bas et de tout à droite sont affichés. Il manque donc les murs en haut et à gauche. Complétez le code aux endroits notés *TODO Task 1* afin d'afficher le labyrinthe correctement comme ci-dessous (p1 correspond à la position du premier joueur et e à la sortie). Votre affichage devrait donner exactement :

```
*---*---*---*---*---*
| p1|                 |
*   *   *---*---*   *
|   |   |       |   |
*   *---*   *   *   *
|           |       |
*---*---*---*---*   *
|       |           |
*   *---*   *---*---*
|         e         |
*---*---*---*---*---*
```

Tâche 2 – Dessin du labyrinthe sous forme graphique

Maintenant que nous sommes capables d'afficher correctement un labyrinthe sous forme de texte, il est temps de rajouter un peu de graphisme à notre programme. Une classe fonctionnant de manière semblable à `TextDisplay` vous est fournie dans le fichier `GraphicDisplay`.

1. Ouvrez le fichier `GraphicDisplay`.
2. Observez la méthode `main` de cette classe et exécutez-la.
3. Modifiez le code afin d'afficher un labyrinthe 10x10.
4. Il est possible de changer la taille de chaque petit carré en modifiant un paramètre du constructeur de `GraphicDisplay`. Essayez d'autres valeurs plus grandes et plus petites.
5. Il est possible d'afficher le labyrinthe sans les bordures de la fenêtre. Trouvez comment et exécutez votre programme. Vous obtiendrez quelque chose de similaire à ce qui se trouve à droite sur l'image ci-contre. Vous pouvez décider d'utiliser plutôt cette méthode ou l'autre selon votre préférence. Essayez également de faire apparaître le grillage d'arrière-plan (une variable à changer).

6. Afin de vous assurer que l'affichage est correct, affichez également depuis le `main` de `GraphicsDisplay` une version textuelle du même labyrinthe.

2ème partie – Sortir du labyrinthe

Tâche 3 – Algorithme de propagation de Lee (A*)

Dans cette phase, vous allez implémenter un moyen de trouver automatiquement la sortie du labyrinthe à l'aide d'un algorithme de routage.

1. Ouvrez la classe `AStar.java` qui contient le squelette de la classe du solveur pour le labyrinthe selon la méthode de Lee qui a été présentée sur les slides. La phase de propagation de l'algorithme (quand le chemin grandit dans toutes les directions depuis le point de départ) peut être décrite formellement comme suit :

Algorithme 1 Propagation de Lee (algorithme A^*)

Marquer le point de départ avec 1
$m \leftarrow 1$
// Propagation en vague
répéter
  Marquer tous les voisins non marqués des points marqués m avec $m+1$
  $m \leftarrow m+1$
jusqu'à destination trouvée **or** plus aucun point ne peut être marqué

2. Pour tester votre méthode, un `main` a également été créé ici. Vous pouvez l'utiliser pour débugger votre code plus facilement.
3. Observez la méthode `solve`. Elle commence par créer une solution vide (qui est en fait un tableau d'entiers). Elle continue en faisant la propagation de Lee puis le backtracking. Le but de tout l'algorithme est de mettre des 1 là où se trouve le chemin de la solution entre le point passé en argument et la sortie. Par exemple, le labyrinthe :

```
*---*---*---*---*
| p1|           |
*   *   *---*   *
|   |   |       |
*   *---*   *   *
|           |   |
*---*---*---*   *
|     e         |
*---*---*---*---*
```

donnera au final

```
1 - 0 - 0 - 0
1 - 0 - 1 - 1
1 - 1 - 1 - 1
0 - 1 - 1 - 1
```

4. Si on enlève le backtracking (en commentant la méthode `backtrace()` dans `solve`), on peut voir les étapes de la propagation ce qui est très pratique pour débugger son code. Dans le cas ci-dessus, cela donnera :

```
01 - 10 - 09 - 08
02 - 11 - 06 - 07
03 - 04 - 05 - 08
00 - 11 - 10 - 09
```

5. Pour pouvoir obtenir ce résultat, vous devez implémenter la méthode `expansion`. Celle-ci est appelée par `solve` tant qu'elle ne retourne pas *true* (c'est-à-dire tant que la propagation de la vague n'a pas atteint la destination). Notez également que la méthode reçoit à chaque appel l'état de *m* qui est le numéro de l'étape en cours. A vous de trouver comment faire (avec notre aide) ! Si nécessaire, n'hésitez pas à utiliser la méthode `access_solution()`, en comprenant à quoi elle sert.

Tâche 4 – Afficher la solution sur la fenêtre graphique

Votre implémentation du point précédent vous donne un tableau contenant une solution à partir d'un point donné. Vous avez la possibilité de l'afficher directement sur la représentation graphique du labyrinthe. C'est ce que vous allez réaliser dans cette tâche.

1. En vous inspirant de ce qui a été fait auparavant, affichez depuis le `main` de la classe `AStar` la version graphique du labyrinthe.
2. L'objet de la classe `GraphicDisplay` que vous avez créé possède une méthode `setSolution` qui prend comme argument un tableau d'entiers correspondant à une solution depuis un point. Utilisez cette méthode pour afficher la solution calculée avec l'algorithme A* créé à l'étape précédente. Si vous désirez enlever la solution de l'affichage, utilisez la méthode `clearSolution()`;

Nous vous souhaitons bien du plaisir
dans la réalisation de ce mini labo !