TP 1 - Recherche de solutions

L'objectif de ce TP est d'implémenter différentes recherches de solutions dans un labyrinthe.

1 Découverte du code fourni

Votre code devra permettre de trouver une solution (pas forcément optimale) permettant d'aller d'un point A à un point B, en utilisant un parcours soit en profondeur, soit en largeur.

À titre d'exemple, le labyrinthe affiché à la figure 1 montre le chemin trouvé en jaune depuis le point de départ (case rouge) vers le point d'arrivée (case verte). Les cases explorées pour trouver ce chemin sont quant à elles colorées en rouge sombre.

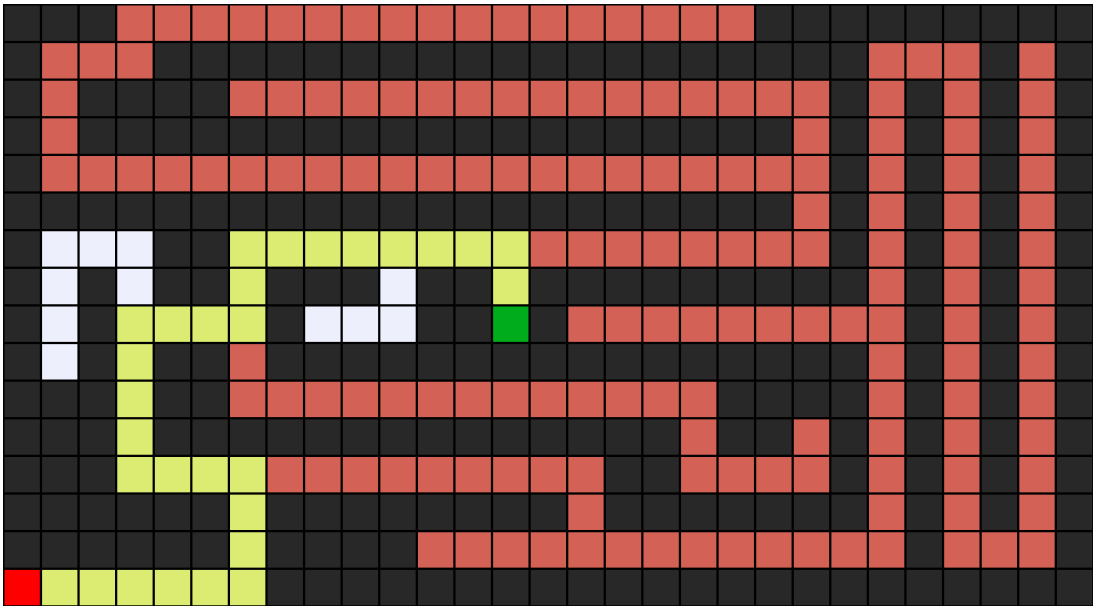

FIGURE 1 – Exemple de labyrinthe

Le programme en Python maze.py qui vous est fourni a un paramètre correspondant à un fichier textuel, contenant une description d'un labyrinthe (cinq exemples de fichier vous sont donnés). Vous disposez déjà d'une méthode fonctionnelle de recherche de solution ainsi que de deux modes d'affichage : une textuelle dans un terminal et une sous la forme d'une image dans un fichier nommé maze.png. Pour exécuter par exemple le programme sur le premier labyrinthe fourni, créez votre environnement python suivant la procédure du fichierCONFIG.txt, puis utilisez la commande :

```
python maze.py maze1.txt
```

L'implémentation donnée crée une frontière d'exploration des états, à l'aide de deux classes (Node et StackFrontier), ainsi qu'un ensemble des états explorés, au moyen de l'attribut explored de la classe Maze. Chaque état (state) est représenté par une paire d'entiers associés aux positions dans le labyrinthe. L'algorithme de recherche proprement dit est mis en œuvre par la méthode solve() de Maze. L'attribut parent de node permet de retrouver un chemin depuis l'état de départ (start) vers celui d'arrivée (goal).

Question

1. Examinez le code de solve() et de StackFrontier. La méthode de recherche codée correspond-elle à un parcours en profondeur ou en largeur ?

2 Ajout d'options pour choisir le mode de recherche

Pour choisir l'algorithme de recherche utilisé, nous souhaitons ajouter les trois options suivantes au programme maze.py :

```
usage: maze.py [-h] [-d | -b | -a] maze.txt

Search a solution in a maze.

positional arguments:
  maze.txt              text file with a description of a maze

options:
  -h, --help            show this help message and exit
  -d, --depth-first     depth first search
  -b, --breadth-first   breadth first search
  -a, --a-star          A* search
```

Pour insérer ces options, utilisez la bibliothèque argparse. Vous pourrez recourir à la fonction add_mutually_exclusive_group() pour rendre l'usage de ces options mutuellement exclusif.

Par défaut, le programme devra utiliser la mise en œuvre actuelle. La recherche de solution A* sera quant à elle développée lors de la section suivante.

Pour implémenter le parcours en profondeur ou en largeur, vous créerez une classe héritant de StackFrontier(), puis vous l'utiliserez dans la méthode solve() en changeant le type de l'objet frontière.

3 Recherche guidée par une heuristique

La dernière option à coder -a est basée sur l'heuristique suivante, estimant la distance à parcourir depuis un nœud N pour atteindre la case d'arrivée B :

$$h(N) = |x_B - x_N| + |y_B - y_N| \ .$$

Définissez cette fonction comme méthode de la classe Maze.

Le principe de la recherche A* est de sélectionner à chaque étape dans la frontière le nœud $N*$ qui a la valeur valeur($N*$) la plus faible. Cette fonction valeur prend en compte à la fois la distance (coût) parcourue depuis la case de départ et la fonction heuristique h :

$$\text{valeur}(N) = \text{coût}(N) + h(N) \ .$$

Définissez une nouvelle classe NodePriority héritant de Node et ayant deux nouveaux attributs : cost et value. De même définissez une classe PriorityQueueFrontier héritant de StackFrontier pour retourner le nœud avec la plus haute valeur.

Pour finir l'implémentation de A*, mettez à jour la méthode solve() de façon à incrémenter le coût à chaque création de nœud et à utiliser la classe PriorityQueueFrontier.

Questions

1. Exécutez les trois algorithmes de recherche de solution sur chacun des cinq labyrinthes fournis. Comparez les nombres de chemins explorés.
2. Les solutions trouvées sont-elles optimales ?

4 Travail à rendre

Déposez par binôme, sur l'espace de cours, votre programme `maze.py` et un fichier texte contenant les réponses aux questions.