TP4 - AG Labyrinthe

A lire attentivement!

Ce travail est noté. Il doit être fait individuellement. Toute aide externe (telle que poser des questions sur des forums ou utiliser des assitants génératifs de type Copilot) est interdite.

Description

Vous devez résoudre un problème de labyrinthe en utilisant Python et les algorithmes génétiques avec le framework DEAP.

Le labyrinthe est représenté par une matrice A de taille NxN, où

- $a_{ij} = 0$ si la cellule est pratiquable
- $a_{ij} = 1$ si la cellule est un mur (non-pratiquable)

$$A_0 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

La cellule de départ est située en $(0, 0)$ (en haut à gauche), et la cellule d'arrivée aléatoirement. Les déplacements autorisés sont les quatre points cardinaux, à savoir: nord, sud, est, ouest.

Dossier

L'archive `student.zip` contient :

- Le module `labyrinth.py`
  - coeur logique de l'application
- Le notebook `explore.ipynb`
  - terrain d'exécution et d'expérimentation, qui utilise le module `labyrinth.py`
- Le fichier `grid10.npy`
  - sérialisation Numpy d'une grille exemple de 10x10

Un début d'implémentation a été réalisé pour vous aider à comprendre la structure du TP.

But

Implémenter la méthode `solve_labyrinth(...)` du module `labyrinth.py`. La signature de cette fonction **ne doit pas être modifiée**.

Rendu

A rendre comme devoir sur Cyberlearn avant le **dimanche 27 novembre 2022, 23h59**:

- Code : seul le fichier `labyrinth.py` **complété** est à rendre.
- Documentation : ajouter un fichier de texte (.md, .docx, .pdf, .txt) expliquant et justifiant **étape par étape** votre solution (encodage d'un chromosome, fonction de fitness, sélection, crossover, mutation, critére(s) d'arrêt et autres spécificitées).

Évaluation

La note sera calculée selon les critères suivants:

- Qualité de l'approche de l'algorithme génétique (60%)
- Qualité générale du code, de la documentation, et de l'exploration (40%)

Les dimensions suivantes seront observées pour les critères:

- Qualité du code et de la documentation
- Considérations pour l'algorithme génétique

Votre implémentation sera ensuite testée sur plusieurs grilles au travers de la méthode `solve_labyrinth(...)`.

Les conditions suivantes seront vérifiées :

- Est-ce que l'algorithme fonctionne se termine normalement (sans crash) ?
- Est-ce que l'algorithme parvient à retourner une solution ? (après un temps max.)
- Est-ce que la solution est possible ?
- Est-ce que la solution est bonne ? (proximité avec la solution optimale)

Les grilles suivantes seront utilisées :

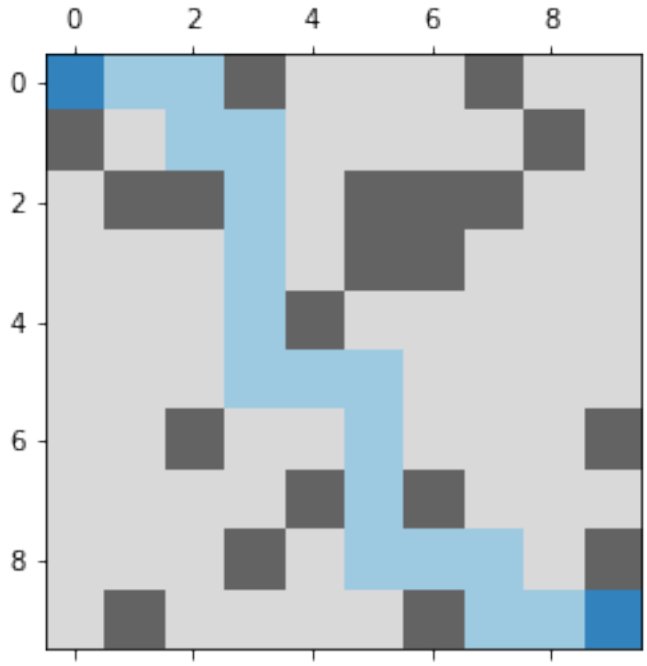
Taille	Temps max.
10x10	10s
15x15	15s
20x20	30s
30x30	60s
40x40	90s
30x30*	180s
30x30**	10s

* grille non-aleatoire

** grille impossible

Exemples

Easy (10x10)



Non aléatoire (30x30, ici pas une solution valable)