Projet Python 1ère année

Sujet : Exploration d'un labyrinthe en aveugle

Encadrant : Arnaud Guibert (Z007, arnaud.guibert@enac.fr)

Contexte

Tous les ans, dans la communauté de recherche de robotique, une compétition a lieu : la « Micromouse Competition ». Vous pouvez trouver les règles détaillées ici. Cette compétition requiert des compétences diverses et complémentaires, telles que la programmation sur microcontrôleur (souvent en assembleur ou langage équivalent) et la conception de microvoitures (les « souris »).

Ce projet s'intéresse uniquement à la partie logicielle de la souris (en Python plutôt qu'en assembleur), c'est-à-dire l'exploration de labyrinthe en aveugle, et la recherche du plus court chemin (mais pas que, comme nous le verrons plus tard).

Règles générales pour les labyrinthes

- Le labyrinthe a une structure carrée (chaque cellule admet *au maximum* 4 voisins)
- Le point d'arrivée est symbolisé par un carré bleu.
- Le point de départ est symbolisé par un carré rouge.

Règles spécifiques de la « Micromouse Competition »

- Chaque souris parcourt le labyrinthe 5 fois, en commençant à l'aveugle.
- Chaque parcours est chronométré, rendant une exploration complète en une fois impossible.
- Le temps final retenu est le plus court de tous les parcours.

Partie I : génération de labyrinthe

- Créez une structure de données (orientée objet) permettant de créer un labyrinthe parfait « non trivial » de taille nxn. Plusieurs algorithmes de génération sont possibles pour générer un tel labyrinthe, tels que les algorithmes de Prim ou Wilson (cf Figure 1).

- Créez une interface graphique pour afficher le labyrinthe que vous venez de générer. Vous pouvez utiliser, par ordre de préférence, Qt / Tkinter (1) ou le prompt de commandes (2).

- Figure 2 montre un des labyrinthes utilisés pendant une des compétitions de la communauté. Implémentez-le en utilisant votre structure de données.

Partie II : premiers algorithmes d'exploration

- Implémentez l'algorithme suivant : « parcours aléatoire ».
    - Affichez-le sur votre interface

- Proposez un algorithme « semi-optimisé » en deux étapes qui se base sur le parcours aléatoire (en première étape). Remarque : il est possible que l'ordre des cases visitées lors du premier parcours soit intéressant.
    - Affichez-le sur votre interface

- Implémentez l'algorithme suivant : « méthode de la main gauche ».
    - Affichez-le sur votre interface

Partie III : algorithmes d'exploration plus complexes

- Implémentez l'algorithme suivant : « algorithme de Trémeaux ».
    - Affichez-le sur votre interface

- Implémentez l'algorithme suivant : « flood fill » (cf. Pseudocode 1).
    - Affichez-le sur votre interface

Partie IV : pour aller plus loin

- Contactez-moi pour aller plus loin (faites-le aussi si vous êtes bloqués d'ailleurs).

Annexes

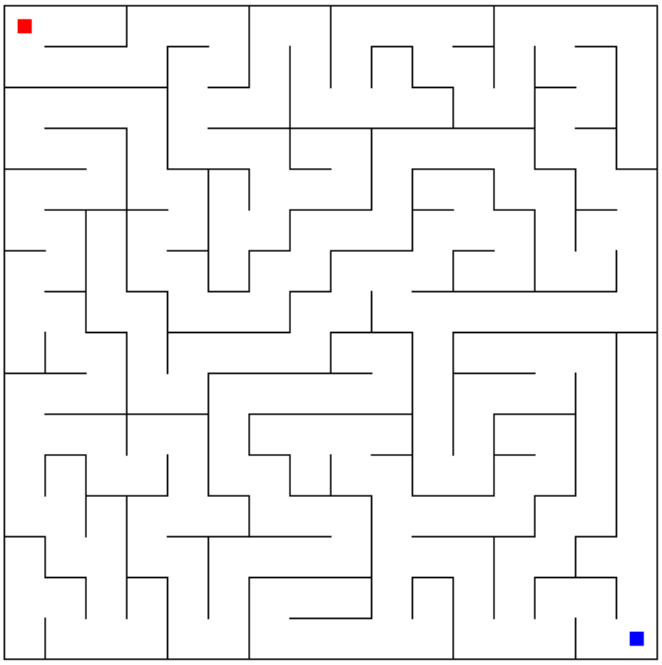

Figure 1 : Labyrinthe parfait

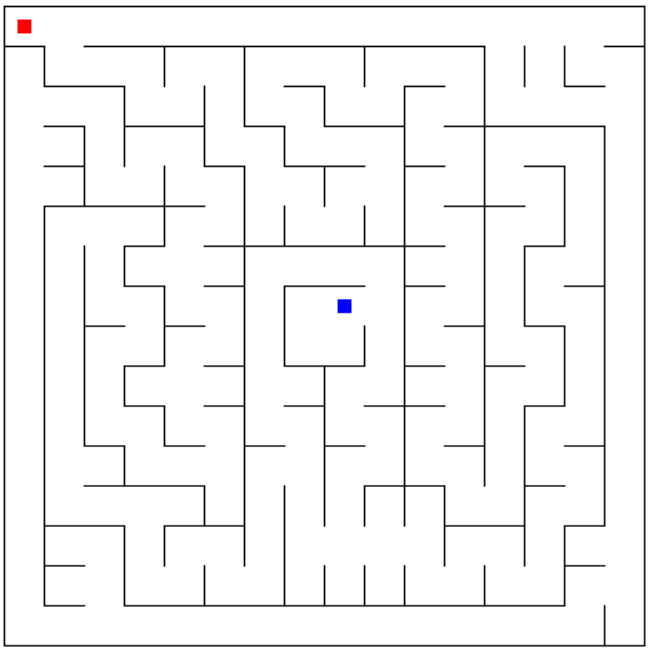

Figure 2 : Micromouse Contest – Japan 2017

Floodfill()

1. Décider d'un ordre de décision. Exemple : N > S > E > W.
2. Remplir (*) le labyrinthe.
3. Mémoriser les murs de la case courante.
4. Regarder les voisins. L'un d'eux est-il moins loin que moi de la sortie ?
   Si oui, faire (5.) Sinon, remplir le labyrinthe puis faire (5.)
5. Aller vers le voisin le plus proche de la sortie.
   Respecter l'ordre de décision en cas de conflit.
6. Si la case courante est la case de fin, stop. Sinon faire (3.)

Remplissage()

- *(Rappel : par défaut, les cellules non visitées n'ont aucun mur)*
- Assigner à la case de sortie la valeur 0
- Tant que toutes les cases n'ont pas de valeur faire pour chaque case *c* sans valeur
    - Si *c* a un voisin accessible avec une valeur v
        - *c* reçoit la valeur v+1

Pseudocode 1 : algorithme floodfill