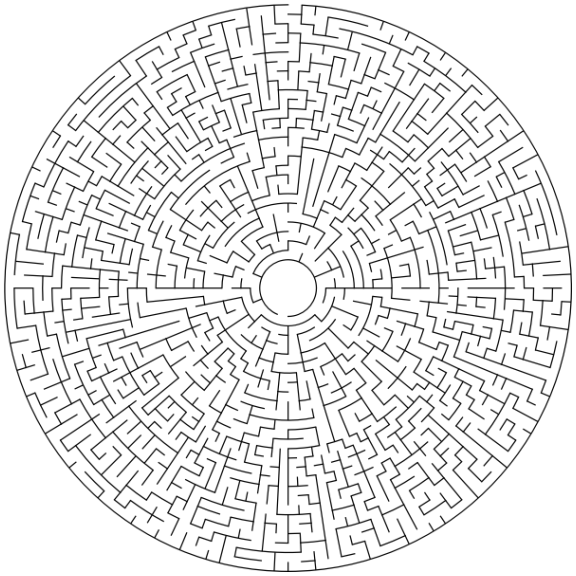
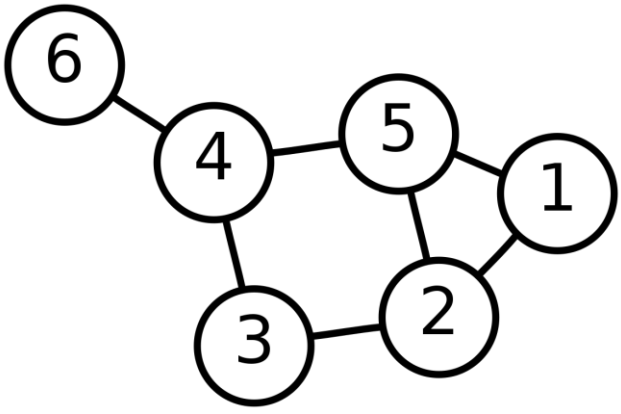
FELLOUH Corentin TG1



Documentation Projet 4 NSI

Parcours de Labyrinthes



Sommaire :

I. Fonctionnement général

Tout d'abord l'utilisateur choisit les dimensions du labyrinthe a créés et le taux de boucle au sein de ce dernier. Par la suite, le labyrinthe s'affiche a l'écran de l'utilisateur (il est possible que la fenêtre s'ouvre en arrière plan). Le menu principal apparait alors, avec 5 options possibles :

-Quitter le programme

-Parcours de labyrinthe

-Résolution de labyrinthe

-détection de cycle

-créer un nouveau labyrinthe

a) Quitter le programme

Si l'utilisateur décide de quitter le programme, toute sauvegarde d'un labyrinthe précédent est supprimée et le programme se ferme.

b) Parcours de labyrinthe

Si l'utilisateur sélectionne le parcours de labyrinthe, il est alors amener à un sous-menu consacré aux deux types de parcours de labyrinthes :

-Parcours en largeur (BFS)

-Parcours en profondeur (DFS)

Une fois le type de parcours sélectionné, une fenêtre s'affiche avec le parcours du labyrinthe en temps réel, il est fortement recommandé d'attendre la fin de l'animation avant de fermer la fenêtre. Une fois fini, l'utilisateur peut fermer la fenêtre et revient alors sur le sous-menu de parcours.

c) Résolution de labyrinthe

Si l'utilisateur sélectionne la résolution de labyrinthe, le programme va alors lui demander les coordonnées de la case de départ et de la case d'arrivée (N.B. : l'axe y est inversé.) puis une fenêtre va s'afficher et montrer comment aller de la case de départ a la case d'arrivée en temps réel. L'utilisateur peut ensuite fermer la fenêtre et revient alors sur le menu principal.

d) Détection de cycle

Si l'utilisateur sélectionne la détection de cycle, le programme va alors révéler instantanément si le labyrinthe possède des boucles ou s'il est acyclique. L'utilisateur revient alors au menu principal.

e) Créer un nouveau labyrinthe

Si l'utilisateur choisit de créer un tout nouveau labyrinthe, toute sauvegarde du labyrinthe précédent est supprimé, le programme se relance et redemande alors à l'utilisateur les dimensions du labyrinthe.

II. Explication détaillée par fonction

Pour commencer, les classes Piles et Files sont initialisées, ces structures de données seront utiles pour les parcours de labyrinthe ainsi que la résolution.

a) Initialisation

La fonction d'initialisation est la pierre angulaire du code, c'est elle qui demande à l'utilisateur les caractéristiques du labyrinthe nécessaire à sa création et qui va communiquer ces informations aux autres fonctions tout en sauvegardant dans un fichier csv la configuration des murs au sein du labyrinthe. A la fin de son exécution, la fonction d'initialisation appelle la fonction de menu.

b) Menu

La fonction de menu est la fonction qui fait le lien avec les différentes opérations possibles sur le labyrinthe, comme expliqué précédemment, elle propose 5 options possibles a l'utilisateur (arrêt, parcours, résolution, cycles, réinitialisation).

c) Parcours

Parcours est donc un sous-menu faisant le lien vers les deux fonctions de parcours du labyrinthe, il affiche en fonction du choix de l'utilisateur un parcours en largeur ou en profondeur, il est aussi possible de revenir au menu principal. Cette fonction est celle qui va suivre la liste renvoyer par les fonctions de parcours bfs et dfs.

d) Voisins

La fonction Voisins, inspiré du cours, se sert de maze_map, une méthode qui renvoie un dictionnaire qui renseigne sur la présence des murs vers les 4 points cardinaux d'une case (syntaxe : maze_map = {(1, 1): {'E': 1, 'W': 0, 'N': 0, 'S': 0},

(2, 1): {'E': 0, 'W': 0, 'N': 0, 'S': 1}, (3, 1): {'E': 0, 'W': 0, 'N': 1, 'S': 1}, etc…}). Avec ces informations, la fonction voisins renvoie une liste des voisins d'une case en rajoutant ou en enlevant 1 a la coordonnée correspondante en fonction de la présence ou non d'un mur. Elle renvoie une liste contenant les coordonnées de chaque voisin de la case concernée.

e) Parcours_largeur

Cette fonction pars d'une case du labyrinthe (par défaut, le coin en bas à gauche), et effectue un parcours en largeur du labyrinthe, c'est-à-dire que l'on parcourt toute les cases a 1 de distance, puis toutes les cases a 2 de distance, etc…
Elle renvoie une liste de case dans l'ordre où il faut les parcourir.

f) Parcours_profondeur

Cette fonction pars d'une case du labyrinthe (par défaut, le coin en bas à gauche), et effectue un parcours en profondeur du labyrinthe, c'est-à-dire que l'on suit un chemin du début a la fin, puis, quand on ne peut plus le suivre, on suit un des chemins que l'on a oublié.
Elle renvoie une liste de case dans l'ordre où il faut les parcourir.

g) Resolution

Cette fonction sert à donner un chemin entre une case de départ et une case d'arrivée dont les coordonnées sont déterminées par l'utilisateur.
Elle affiche ensuite a l'écran le chemin à prendre en temps réel.

h) Dfs2

Cette fonction est un parcours en profondeur qui stocke les parents de chaque case (c'est-à-dire qu'on associe la case dont on vient a la case ou on arrive). Elle renvoie un dictionnaire appelé parents.

i) Solution

Cette fonction se sert du dictionnaire parents pour trouver un chemin entre la case de départ et la case d'arrivée. Elle renvoie une liste de case à parcourir pour arriver à la case ciblée.

j) Cycle

Cette fonction indique à l'utilisateur si le labyrinthe créé possède des cycles ou s'il est acyclique.

k) isCyclic

Cette fonction se sert du dictionnaire de parents pour détecter la présence de cycle dans le labyrinthe. Elle renvoie True ou False.

III. Parcours le plus efficace

Il faut savoir que, dans le code, le délai entre chaque action de l'agent parcourant le labyrinthe dépend de la taille de ce dernier et du type de parcours, ceci est volontaire, dans le but d'obtenir une vitesse à peu près constante. Cependant, à délai égal, le parcours le plus efficace est celui en profondeur, surtout sur les labyrinthes de grande taille.

IV. Problèmes rencontrés

Le module dont je me sers pour générer le labyrinthe (pyamaze) créé une sortie par défaut en vert en (1,1), pour le camoufler je me sers d'un agent immobile, appelé « b » de la même couleur que le labyrinthe, qui semble alors fermé.

Comment obtenir une vitesse constante peu importe la taille du labyrinthe ?
Je calcule la vitesse de délai en faisant 1/((largeur+longueur)/2) * n avec n une constante. Plus n est élevé et plus la vitesse de parcours est lente. Sachant que le parcours en largeur est moins efficace je lui accorde un délai moins élevé que le parcours en profondeur.