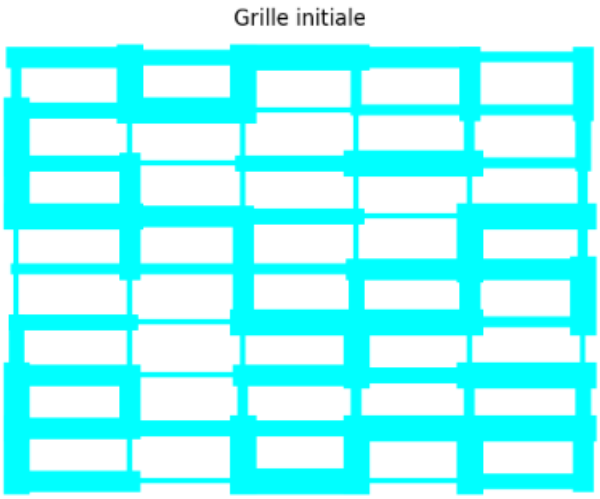
ISMAIL Saman **Groupe : IE-04a**

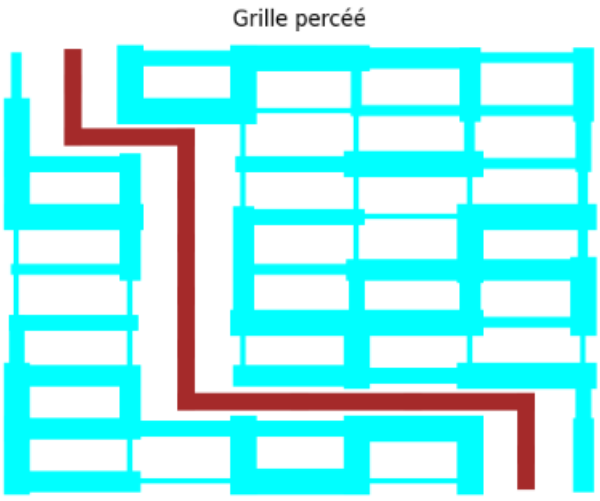
Rapport projet INFO3A
Construction au fond de la jungle

Pour faire tout ce projet j'ai utilisé l'IDE Intellij.

Démonstartion

```
Lignes : >? 8
Colonnes : >? 5
Cout: 93
Chemin: [(0, 0), (1, 0), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (6, 2), (6, 3), (6, 4), (7, 4)]
```





Tout d'abord, j'ai créé 3 fichiers (Main.py, Cellule.py, Grille.py), ces trois classes vont nous permettre respectivement d'exécuter le programme principal, modéliser la cellule et modéliser la grille.

La plupart de notre code sera présent dans la classe Grille

la classe Cellule contient les informations sur une cellule

Une Cellule contient cinq variables :

- 4 variables pour l'épaisseur de chaque mur (gauche, haut, droit et bas)
- Centre : un couple qui permet de connaitre le centre de la cellule initialisé à (0,0)

Elle contient également les méthodes qui permettent de récupérer les informations de la cellule

Les méthodes sont:

- __init__: initialise les variables de la classe(constructeur)

- les getters et setters de chaque attribut

- __str__: permet d'afficher les informations de la cellule(elle ne va jamais être appelée dans le programme principal)

Une Grille contient 3 variables de classe :

- lignes : un entier qui permet de stocker le nombre de lignes de la grille
- colonnes : un entier qui permet de stocker le nombre de colonnes de la grille
- grille : Une liste à deux dimensions qui contient les cellules de la grille

Elle contient également les méthodes suivantes :

- __init__(self, lignes, colonnes) : constructeur de la classe Grille
- les getters et setters des variables de classe
- creerGrille(self) : permet de créer la grille de cellules à partir de la classe Cellule, randomise l'épaisseur de chaque mur et retourne une liste à deux dimensions de cellules, les épaisseurs sont dans une liste [3, 6 ,9 12, 15]
- afficherGrille(self) : permet d'afficher la grille avec la bibliothèque matplotlib
- creationDictionnaire(self) : algorithme qui permet de créer un dictionnaire qui contient les sommets, leurs voisins et les poids(graphe pondéré)

- djikstra_pred(self) : algorithme Djikstra qui permet de calculer le chemin le plus court, cette méthode retourne deux dictionnaires : D qui contient les distances minimales et P qui contient les prédécesseurs

- minimum(self, dico) : une fonction calculant le minimum des valeurs d'un dictionnaire donné en paramètre et renvoyant la clé correspondante (utilisée dans l'algorithme Djikstra)

- chemin(self) :méthode qui donne le chemin optimal traversant la grille. Ce chemin est donné

sous forme d'une liste de couples comportant les coordonnées des cellules traversées dans l'ordre

- afficherChemin(self) : affichage du chemin optimal traversant la grille avec la bibliothèque matplotlib et aussi l'affichage de la grille percée

- calculCout(self) : permet de calculer le coût du chemin le plus court

Le fichier Main est le programme principal, c'est dans ce fichier que j'ai fait la plupart de nostests

Partie 1:

a)

Voir méthode creerGrille(self) dans le fichier Grille.py

J'ai choisi de construire la grille avec une liste à deux dimensions car c'est la meilleure méthode pour représenter une grille.

b)

Voir méthode afficherGrille(self) dans le fichier Grille.py

Partie 2 :

a)

J'ai choisi un dictionnaire comme structure de données associée à l'ensemble des cheminspassant par toutes les cellules de la grille, et tenant compte de l'épaisseur de chaque mur.

Comme j'ai vu en CM, le meilleur moyen de représenter un graphe pondéré est un dictionnaireb)

Voir méthode creationDictionnaire(self) du fichier Grille.py

c)

Nous partons du principe que 1 épaisseur a besoin de 1 litre de carburant pour être percée.

Nous utilisons l'algorithme de Djikstra pour traverser la grille avec un coût minimal (en termes de dépense d'énergie), il est adapté ici car cet algorithme va essayer plusieurs chemins pour trouver le plus court ce qui nous permettra d'économiser de l'énergie.

Voir méthode djikstra_pred(self) et minimum(self, dico) dans le fichier Grille.py

d)

Voir méthode chemin(self) dans le fichier Grille.py

e)

Voir classe Grille dans le fichier Grille.py

Partie 3 :

a)

Voir méthode afficherChemin(self) dans le fichier Grille.py

b)

Voir le fichier Main.py

Partie 4 :

J'ai réfléchi à mettre au point une version plus efficace (mais pas forcément optimale) de l'algorithme de Djikstra.

En effet c'est un algorithme qui s'appelle Astra, il s'agit d'une extension de l'algorithme de Dijkstra.

J'ai implémenté l'algorithme mais j'ai été bloque car nous ne savions pas commentgénérer l'heuristique de l'algorithme

Voir méthode Astar(self) du fichier Grille.py

Fin