Le projet c'est des graphes
Problématique : Comment optimiser un réseau de transport dans une ville ? A un but d'optimisation et d'automatisation.
Le projet est de prendre en compte plusieurs lignes de transport, des différents obstacles, des destinations obligatoires, des "embouteillages" et ce avec une plus courte distance (soit temporelle, soit métrique)

Actuellement la réponse à ce projet repose sur les algorithmes de Dijkstra et A*. Ils sont implémentés.
On a ajouté des concepts pour la modélisation et adapté les programmes en question.
Les dits concepts sont : les obstacles (représentés par l'absence d'arêtes); les points d'intérêts où l'algorithme est contraint de passer.
Le programme est basé sur du traitement de graphes avec des étiquettes uniques et les arêtes sont grossièrement au nombre de 8 autour de chaque nœud. (spécifiquement si le nœud est sur un ou deux bords)
→ Pour mieux visualiser tout ça, il y a une fonction `display_graph` et un exemple de configuration est déjà écrit.

Mon but prochain est d'implémenter des zones d'intérêts où le plus court chemin devra y passer d'où les matrices "circulaires". Dans cette optique il reste à faire en sorte que le programme prenne en compte ces zones pour le plus court chemin.
Mon but prochain est d'aussi implémenter une modification de la fonction `weight` (représentant un temps d'une arête, une distance, un ralentissement de circulation) en utilisant un gradient.
Ce gradient sera "superposé" au graphe et on prendra des valeurs de ce gradient pour déterminer le poids de l'arête concernée. Il n'y aura pas de modification à apporter aux programmes A* et dérivés.
Une autre chose pour alléger la création de graphe sera de faire une fonction aléatoire qui génère mes config.

Dictionnaire des Fonctions : *(du programme principal)*
• `lattice` génère mes nœuds
• `edges` génère mes arêtes
• `obstacles` retire des arêtes pour créer des obstacles
• `label_to_coordinates`
• `coordinates_to_label`
• `weight` donne le poids d'une arrête
• `a_star`
• `multi_a_star` est un A* modifié qui prend en compte les points d'intérêts
• `circular_matrix` renvoie les points à coordonnées entières comprises dans un rayon (entier)
• `display_graph` affiche le graphe avec plein de paramètres pour faire joli
• `super_print` elle est faite pour afficher les très gros plus courts chemins les rendant lisible sur le graphe. Elle renvoie le premier point, la direction vers le prochain point où ça change de direction puis renvoie ce point et ainsi de suite jusqu'au dernier.

La version la plus à jour est le fichier PlusCourteDistanceMulti.ipynb. Les autres fichiers sont intéressants car ils ciblaient d'autres étapes.