PROJET INFO 3A EN 2019-2020

Votre projet est à réaliser par groupes de **4 étudiants**, en Java, mais vous pouvez utiliser d'autres langages pour vos premiers essais. Votre démonstration aura lieu lors des TP de la semaine 49. Chaque étudiant devra présenter une partie du projet. Lors des quelques TP précédents, vous pourrez travailler sur votre projet, mais c'est une mauvaise idée de commencer par le projet : faites d'abord des exercices plus faciles. Ce projet est le même que celui de l'an dernier, sauf que le 2D arbre et le raffinement itératif n'étaient pas demandés l'an passé, et que les étudiants travaillaient en binôme.

Enoncé : Votre programme doit calculer le plus court chemin entre deux points donnés A et Z dans un environnement encombré 2D. Cet environnement est un rectangle contenant des obstacles simples (rectangles, cercles, etc). Chaque obstacle peut être représenté par une fonction (ou une méthode) calculant si le point passé en paramètre est à l'intérieur de l'objet. Vous échantillonnerez l'environnement avec des points aléatoires, et vous rejetterez les points à l'intérieur d'un obstacle. Vous construirez le graphe dont les sommets sont les points échantillons, A et Z, et où deux sommets sont reliés par une arête quand ils sont distants de moins d'un certain seuil R ; vous fixerez empiriquement ce rayon seuil pour que le graphe soit suffisamment dense. Vous calculerez le plus court chemin entre A et Z par la méthode de Dijkstra, en utilisant un tas (ou « heap »). Vous mettrez en place trois optimisations :

- vous utiliserez un tas pour la méthode de Dijkstra.
- Vous utiliserez un 2D-arbre (« 2D-tree ») pour accélérer la génération du graphe. Cet arbre vous évite de calculer la distance entre les n points, ce qui est en temps quadratique, O(n^2).
- Vous utiliserez du raffinement itératif : après avoir calculé un plus court chemin de A à Z, vous calculerez l'écartement des sommets du graphe à ce plus court chemin AZ. Pour les sommets peu écartés, par exemple de 1.15 AZ, vous générerez des points à l'intérieur de leur disque de visibilité, et vous relancerez l'algorithme. Après 2 ou 3 raffinements, vous obtiendrez un chemin visuellement lisse.

Vous testerez votre programme avec des données de plus en plus volumineuses, et vous vérifierez sa complexité en traçant une courbe log-log (vu en CM et TD).

Cette méthode est utilisée dans l'industrie pour planifier les mouvements des robots ou des bras articulés dans les ateliers, sur les chaînes de montage pour l'assemblage des voitures ou des camions. La différence avec votre projet est le nombre de coordonnées par point. Il n'y en a que 2 dans votre projet : le x et le y de chaque point échantillon. Il y en a une dizaine pour un bras articulé saisissant un moteur et allant le placer dans une carrosserie, et une cinquantaine pour un robot humanoïde qui marche en poussant ou en tirant une brouette : chaque coordonnée est, typiquement, un angle ; il peut y avoir quelques longueurs, pour les vérins hydrauliques. Jean-Paul Laumond (page wikipedia) est un expert français de ce domaine. Cette méthode est un exemple de méthode probablement approximativement correcte, PAC (en anglais : « probably approximately correct »).

Barême : Chacune des trois optimisations est notée sur 6 points, au plus. Par défaut, vous avez 2/20 (si vous rendez votre projet à temps). La note maximale possible est donc 20/20.

Que devez-vous rendre ? Il faut envoyer une archive de votre projet à votre encadrant en TP, semaine 48 au plus tard. Cette archive doit porter le nom des étudiants de votre groupe. La ligne de commande suivante permet l'archive :

```
$  tar czvf dupont_durand_martin_tartampion.tgz LE_REPERTOIRE_DE_VOTRE_PROJET
```

Pour vérifier que votre archive est correcte, copiez la dans un répertoire temporaire TMP et décompressez la par :

```
$ mkdir TMP
$ cp  dupont_durand_martin_tartampion.tgz TMP
$ cd TMP
$ tar xzvf  dupont_durand_martin_tartampion.tgz
$ ls #pour vérifier que tous les fichiers de votre projet sont bien présents
```

Votre archive ne doit contenir qu'un seul et unique programme java. Donc n'y placez pas une hiérarchie de fichiers fabriquée par un AGL/EDI/IDE tel Eclipse. Votre archive peut contenir aussi un fichier texte LISEZMOI.txt, où vous pouvez expliquer comment lancer votre programme, par exemple dans le cas où il faut lui passer des paramètres (tel qu'un numéro de fonction de démonstration). Votre programme doit se compiler par

```
$ javac *.java
```

et se lancer par :

```
$ java Dijkstra  #si tel est le nom de votre programme
```

Il est conseillé que votre programme écrive, lors du lancement, les prénoms et noms de ses auteurs. N'utilisez pas de librairie Java exotique : c'est un module d'algorithmique, et pas d'IHM ou de graphique.

Erreurs à ne pas commettre :

- commencer le projet au dernier moment ;
- commencer le projet sans avoir fait suffisamment d'exercices de difficulté intermédiaire ; il faut être à l'aise avec les listes (par exemple, le graphe est représenté par un tableau de listes de paires : (indice du sommet voisin, longueur de l'arc) ; une feuille du 2D-arbre contient la liste des indices de sommets), les arbres (le tas est un arbre, le 2D-arbre aussi), les graphes, la programmation dynamique, la récursivité ;
- oublier qu'il faut mettre à jour la position d'un sommet dans le tas quand sa distance à la source diminue. Il faut donc une structure de données (par exemple un tableau) permettant de gérer efficacement la position d'un sommet d'indice donné dans le tas ;
- oublier de vérifier que votre programme fonctionne bien sur les machines en salle de TP ; certains programmes graphiques en java ne marchent pas en salle A119.
- ne pas venir en cours (CM, TD, TP). C'est en CM qu'est décrit le 2D-arbre par exemple.
- Perdre votre temps avec une IHM graphique qui n'est pas demandée. Faites-vous plaisir seulement une fois que votre programme fonctionne.

Pourquoi Java ? Python est certes plus facile mais il n'est pas compilé, et son environnement d'exécution est peu performant.