UNIVERSITÉ PARIS 8

LICENCE INFORMATIQUE & VIDÉOLUDISME

PROJET TUTEURÉ

Rapport du projet : Dessin de graphe

Projet conduit par :

Nom prénom	**Classe**	**N°Étudiant**
Ichou Aymane	L3B	21007668

Projet tuteuré par : Sven Defelice

Sommaire :

1.Mon projet

Dans le cadre d'un projet tuteuré, mon projet à consisté en la création d'un moyen de visualisation d'un graphe non orienté, et le projeter de manière dîtes « jolie ».

Nous avons choisi ce projet, car le côté créatif, artistique, et arbitraire nous a plu. Cet aspect nous a permis de pouvoir agencer notre projet comme nous l'avons voulu, et expliquer simplement et logiquement nos choix. A titre d'exemple, l'un des choix que nous avons arbitrairement pris, est que la génération des nœuds soit au maximum à 100 nœuds pour des raisons d'esthétisme, et de lisibilité. Un autre choix a été de pouvoir observer l'évolution des points et des arêtes à travers une animation.

2.Les stacks.

2.1.Stack technique.

Pour mener à bien ce projet, nous avons été libre de choisir les bibliothèques et les langages utilisés.
Pour ce faire, nous avons utilisé le langage Python, ainsi que les bibliothèques :

- matplotlib, pour l'affichage du graphe via l'interface ainsi que l'animation.
- dataclasses, pour la création de classes.
- csv, pour la création de fichier .csv .
- copy, pour la copie en profondeur de liste et de dictionnaire.
- random, pour la génération de nombre aléatoires.
- sys, pour get les arguments en ligne de commande.
- math, pour l'utilisation de fonctions mathématiques.

2.2.Stack théorique.

Ceci étant côté technique.Pour le côté théorique, l'agencement ainsi que le positionnement des points, ont été géré grâce à l'algorithme de force dirigé de EADES. Nous avons choisi cette algorithme de part sa simplicité en terme de calcul, de son efficacité, et de sa malléabilité. Avoir utiliser cette algorithme permet d'apporter à l'utilisateur un côté ludique et expérimental, et que cela ne dépende pas seulement d'un programme qui exécute et qui donne le résultat. Ici, l'utilisateur possède un vrai pouvoir de décision sans que ceci ne soit compliqué à comprendre et à exécuter.

3.Structure du projet.

3.1.Les différents fichiers/dossier.

Dans ce projet, nous allons retrouver des fichiers d'une part, et les programmes Python de l'autre part.
En terme de fichiers, nous avons les fichiers :
- "config.txt" , ce fichier qui va permettre certaines configurations du projet. Il contient 2 configurations tel que edit_mode, ainsi que l'identifiant de fichier.
Le mode éditeur permet l'édition du graphe grâce à la souris, l'identifiant va permettre un bon nommage des fichiers.

- "generated_graph.txt" , ce fichier contient le résultat d'une génération aléatoire d'un graphe produit par le programme gen_graph.py. Il contient l'ensemble des liens crée formaté de cette manière :

"nœud_parent → nœud_enfant"

Puis se termine par le nombre de nœud unique présent lors de la génération.

- Le dossier "csv", c'est le dossier qui va venir accueillir les exports qui vont être fait durant la démo si l'utilisateur en fait.
- " README.txt", ce fichier contient les explications pour l'utilisation du programme.

3.2.Les différents sous programmes.

Maintenant que les sous fichiers ont été expliqué, nous avons les programmes parallèles qui vont venir faciliter et épauler le bon fonctionnement du programme principal.
Nous avons les programmes :
- "set_config.py", ce fichier va venir faciliter l'écriture dans le fichier "config.txt" , et proposer à l'utilisateur les choix possibles.
Pour edit_mode, les valeurs possibles sont soit 0 ou 1, respectivement, désactivé ou activé. Ce paramètre va venir activer ou non, le mode éditeur.
Pour l'id de fichier, c'est l'identifiant qui va être concaténé au nom du fichier pour permettre un nommage sans écraser les fichiers déjà présent. Il prend en valeur un entier, qu'elle qu'il soit. Puis s'incrémentera lorsque l'utilisateur fera des exports.
-"gen_graph.py", ce fichier est celui qui va calculer le graphe de manière aléatoire. Il prend en argument le nombre de nœud (maximum 100) souhaité dans la ligne de commande.
Exemple : python3 gen_graph.py 15
Il va pour chacun des nœuds, le tester avec tous les autres nœuds avec le test suivant :

On tire un nombre aléatoire, et si le nombre est plus petit que la probabilité de 1/n (où n, est le nombre de nœud) , l’arête entre les deux nœuds va être crée et écrite dans le fichier "generated_graph.txt".

3.3.Programme principale.

Le code principal est contenu dans le fichier "graph_drawing.py", c'est lui qui va venir calculer l'ensemble des forces attractives et répulsives entre les points, et appliquer un vecteur distinct sur chacun des points.
Ce programme prend deux arguments dans la ligne de commande.
Il prend le "nb_iterations", qui est le nombre de fois que le calcul des forces va être calculé pour l'ensemble des points.
Ainsi que le cooling factor aussi appelé facteur de refroidissement. Il doit être inférieur strictement à 1. Ce facteur va venir amenuiser la force qui sera appliqué sur chacun des nœuds d'une part, et éviter des nombres aberrants d'autre part.
Une erreur est indiqué dans la console lorsque le facteur donnée est trop grand.

4. Deux grands axes, le calcul puis l'animation.

4.1.Calculs.

4.1.1. Algorithme de EADES.

Lors du calcul, c'est l'algorithme de force dirigé de EADES qui est mis en œuvre. Cet algorithme considère les nœuds comme des anneaux, et les arêtes comme des ressorts et voit l'ensemble du graphe comme un système mécanique. Une force attractive est appliqué si un nœud est relié par une arête avec un autre nœud, et une force répulsive pour tous les autres nœuds est appliqué à un nœud donné. Puis calculé sur chacun des nœuds. Les forces sont calculé de la manière suivante :

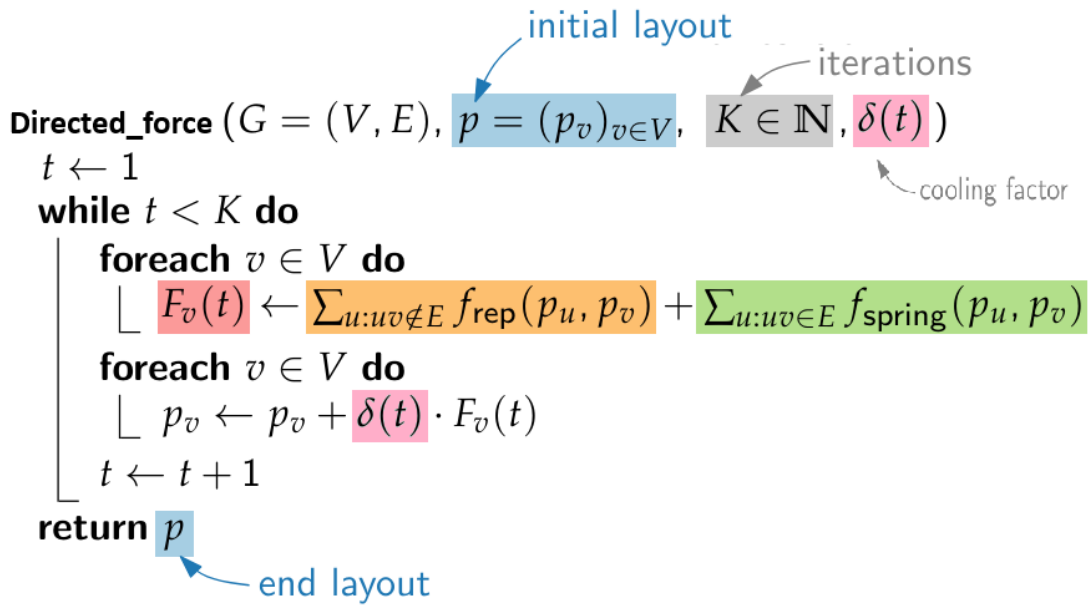
$$f_{\text{rep}}(p_u, p_v) = \frac{c_{\text{rep}}}{||p_v - p_u||^2} \cdot \overrightarrow{p_u p_v}$$

$$f_{\text{spring}}(p_u, p_v) = c_{\text{spring}} \cdot \log \frac{||p_u - p_v||}{\ell} \cdot \overrightarrow{p_v p_u}$$

Légende :

- f_rep = force repulsive
- f_spring = force attractive
- u, v : deux nœuds ou points, liés dans f_spring, et séparé dans f_rep
- c_spring = constante valant 1
- c_rep = constante valant 2
- L = valeur d'un ressort au repos
- || a || = Notation, distance euclidienne de a

4.1.2 Fonctionnement de la fonction principale :

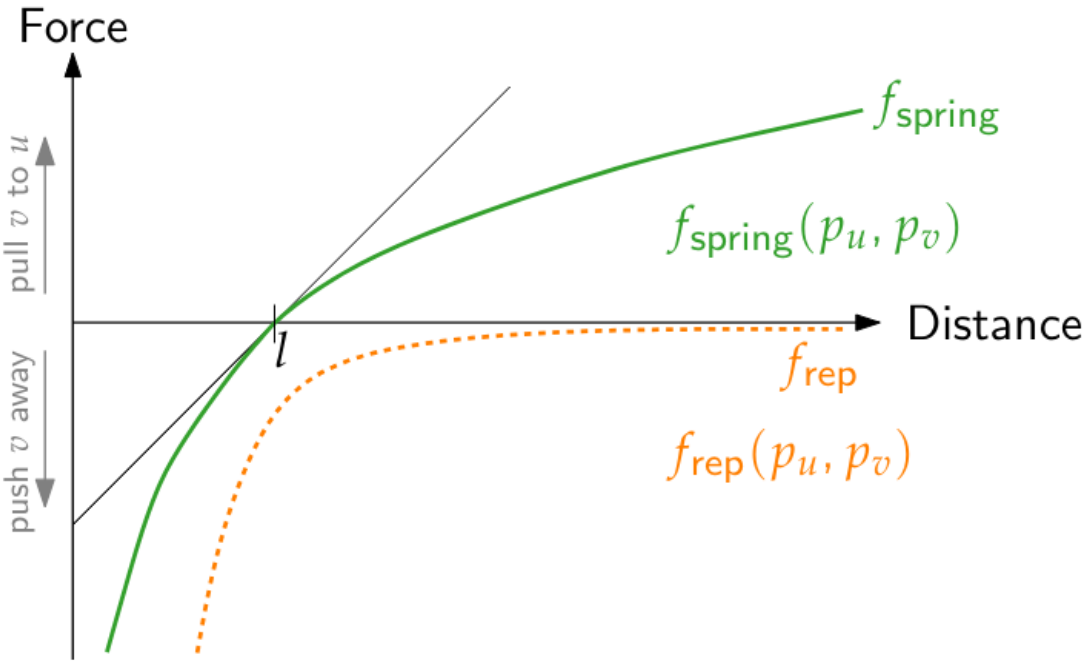


4.1.3.Explications.

La fonction, prend un graphe constitué de Vertices (points ou nœuds) et de Edges (arêtes), avec des positions initialisé au hasard, un nombre d'itérations puis un cooling factor. A chaque tour, la somme des forces répulsive et la somme des forces attractives vont être additionné, et ajouté à une liste. Cela renvoie une liste de longueur du nombre de nœud, car le calcul est effectué pour chaque nœud.
Ce sont les fonctions nommées "sum_frep_to_1_point" et "sum_fspring_to_1_point" dans le code, qui vont calculer la somme des forces répulsive et attractive à un point donné.
Puis une deuxième boucle, qui va appliqué les forces à chaque point en appliquant le facteur de refroidissement.
Et cette boucle va être répété selon le nombre d'itérations données.

4.1.4.Illustration des forces.

Voici la puissance des forces en fonction de la distance entre deux points :



4.2.L'animation.

4.2.1.Fonctionnement de l'animation.

Lorsque la fonction directed_force a fini de calculer, elle renvoie un ensemble d'état, où un état est l'ensemble du graphe à un tour donné.
L'animation va être joué grâce à la FuncAnimation de la bibliothèque Matplotlib.
"FuncAnimation :
(fig, update, frames=len(positions_history), interval= 5, repeat=False)"
Elle prend en paramètre, une interface, une fontion callback d'update, la liste des états, un intervalle en millisecondes, et un paramètre repeat qui défini si l'animation boucle.
Durant cette animation plusieurs entrées claviers sont disponible :

- La barre d'espace : Elle joue ou met en pause l'animation.
- La touche t : Termine l'animation uniquement si l'animation est en pause.
- La touche e : Elle permet l'export si l'animation est terminé.
- La touche m : Si le mode éditeur est activé et que l'animation est en pause, elle va permettre la transposition du graphe dans la fenêtre éditeur.

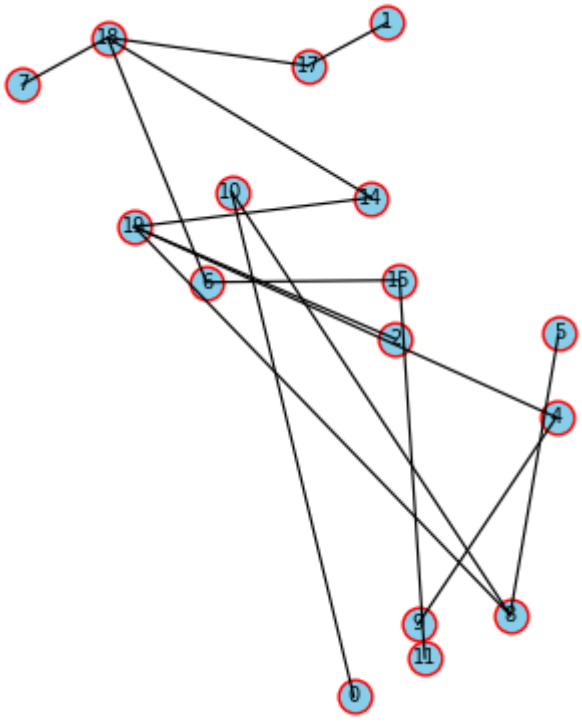
4.2.2.Mode éditeur.

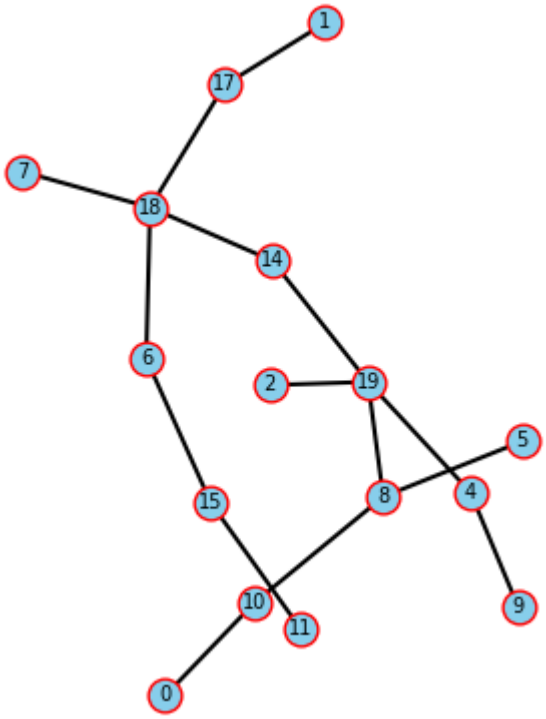
Qu'est ce que le mode éditeur ?
Le mode éditeur est une deuxième fenêtre, en plus de la fenêtre principale, qui va venir s'ouvrir en arrière plan si le paramètre a été activé dans le fichier de configuration.
Dans ce mode éditeur, les nœuds sont déplaçable avec le curseur de la souris en mode drag & drop.
Les transpositions depuis la fenêtre principal y sont illimités, ainsi que les exports.
Ce mode a été mis en place afin que l'utilisateur puisse peaufiner ou détailler les résultats qu'il peut obtenir dans la fenêtre principale.
Subtilité à prendre en compte, la dernière fenêtre cliqué (sélectionné) est celle qui va être relier aux entrées clavier.

5.Résultats.

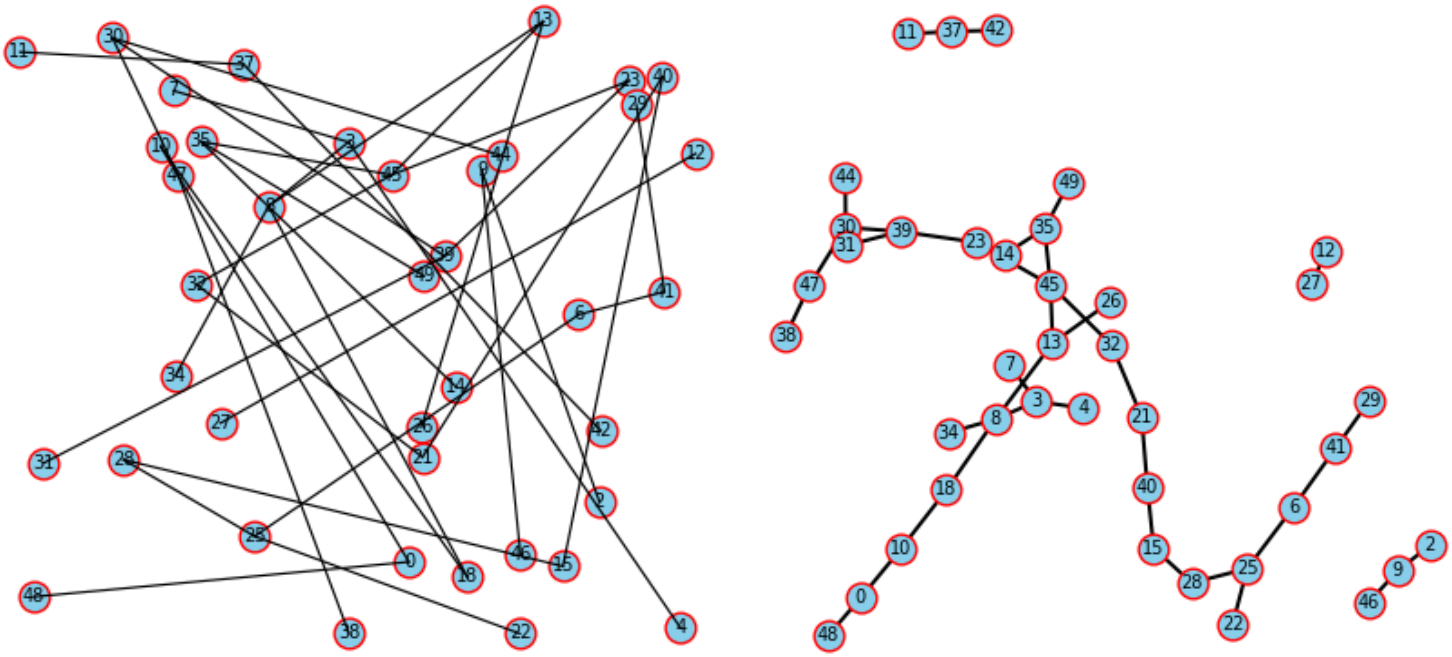
Voici les résultats obtenus avant et après, pour un petit, un moyen, un grand échantillon, accompagné du nombre d'itérations, du cooling factor, ainsi que le pourcentage de l'animation représentant le moment où le résultat a été récupéré.
Il faut noté que, dû à cet algorithme, le résultat final n'est pas forcément la meilleure visualisation. Si le programme tourne durant trop longtemps, les points auront tendance à se rapprocher plus que nécessaire. Malheureusement, c'est un défaut de l'algorithme. Il se peut donc que la meilleure visualisation soit, par exemple, à 60 % de l'animation en fonction des paramètres données lors du lancement du programme. C'est en prenant en compte ce facteur, qu'il a été arbitrairement choisi de ne pas ajouter une touche qui permettait d'aller à la fin de l'animation directement.

Graphe constitué de 20 nœuds, 100 itérations et un cooling factor de 0.01, coupé à 100 % de l'animation :

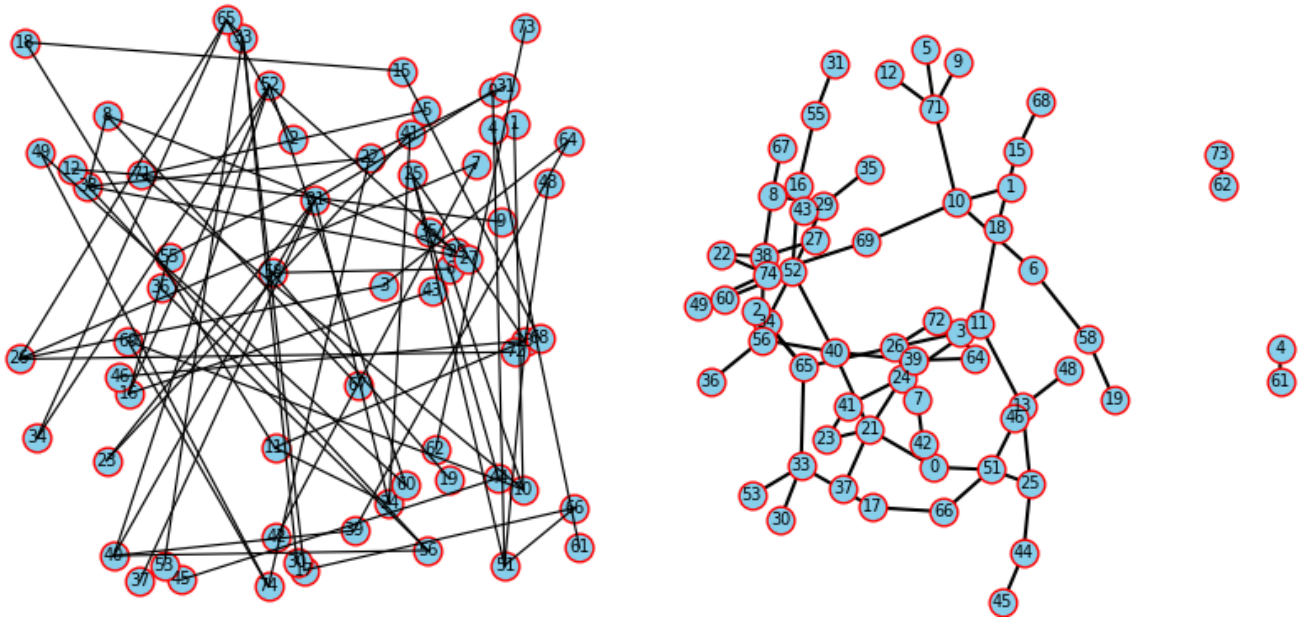




Graphe constitué de 50 nœuds, 200 itérations et un cooling factor de 0,1, coupé à 67 % de l'animation:



Graphe constitué de 75 nœuds, 800 itérations, cooling factor de 0.01, coupé à 100 % de l'animation :



6.Résumé du code.

Les grandes étapes du programme principal "graph_drawing.py" :
- Classes :
    - vecteur et point : qui sont tout deux constitué d'un x et d'un y, mais ils existent tout les deux pour un soucis de clarté de code.
    - DraggablePoint : rend les points déplaçable dans le mode éditeur. Constitué d'une fonction on_release, on_press, on_motion, update_plot.

- Récupération des arguments donnés dans la ligne de commande avec get_config().

- Récupération du graphe, une "liste_gauche" va être crée pour les éléments à gauche de la flèche, et une "liste_droite" va être crée pour les éléments à gauche de la flèche. Je rappelle que le fichier "generated_graph.txt" est formaté comme ceci : 1→ 2 , soit l'arête entre le nœud 1 et 2

- Je crée l'arête dans l'autre sens (cf gen_graph.py), afin de m'assurer que les nœuds à droite existe.

- Je crée une liste de tuple edge, qui associe à un indice donnée, la liste gauche et droite.

- Je crée un dictionnaire node_positions, qui associe une clé de type string qui est le nœud, à un tuple de float que sont les coordonnées, et je place aléatoirement les nœuds

- Création d'une liste d'élément unique appelé all_node regroupant tous les nœuds.

- Création d'un dictionnaire d'adjacence appelé dic_adjacence. Il associe un nœud à une liste d'autre nœuds selon si ils sont relié au nœud donnée en clé.
Exemple : Considérons que 0 dans generated_graph.txt est relié à 3, 2, 7.
Nous allons obtenir {0 : [3, 2,7] },

- Calcul de l'ensemble des états dans la variable positions_history par la fonction directed_force(). Cette fonction est la fonction principal, elle va venir faire la somme des fonctions sum_frep_to_1_point(), et sum_fspring_to_1_point(), et stocker le résultat dans temp, puis append le resultat à liste_force.
Puis nous appliquons les vecteurs à chacun des nœuds en amenuisant les résultats avec le cooling factor. Et fait une copie du dictionnaire qui va être append à la liste positions_history (élément qui va être return). Et cela autant de fois que le nombre d'itérations donnée.

- On crée une sous interface si le mode éditeur est activé.

- On ouvre une interface, puis nous dessinons.

Ensuite, FuncAnimation est appelé, soit la fonction qui va itérer au sein de position_history, et à chaque itération, update() va être appelé, rafraîchissant ainsi l'animation.

-fig.canvas.mpl_connect('key_press_event', modify) : cette fonction va permettre l'association d'une clé clavier à l'interface, et si la touche donnée est appuyé, la fonction donnée va être appelé, ici modify.

plt.show() : ouverture de l'interface.

7.Bibliographie.

- Tutoriels pour la compréhension de l'algorithme, et plus loin :
https://www.youtube.com/watch?v=WWm-g2nLHds&t=222s&ab_channel=PhilippKindermann

https://www.youtube.com/watch?v=gJiSvGbH0CA&t=11s&ab_channel=PhilippKindermann

https://www.youtube.com/watch?v=JAe7Oscsp98&t=15s&ab_channel=PhilippKindermann

- Théorie de la visualisation de graphe, ainsi qu'inspiration sur les illustrations :
https://i11www.iti.kit.edu/_media/teaching/winter2016/graphvis/graphvis-ws16-v6.pdf

https://wuecampus.uni-wuerzburg.de/moodle/pluginfile.php/2319968/mod_resource/content/2/gv-ss20-09-force-directed-algorithms.pdf

- Démo de l'algorithme disponible en ligne gratuitement :
https://www1.pub.informatik.uni-wuerzburg.de/demos/forceDirected.html

- Pour les diverses recherches :
https://fr.wikipedia.org/wiki/Wikip%C3%A9dia:Accueil_principal

8.Remerciements.

Mes remerciements vont à :

- Sven Defelice, pour son encadrement du projet.
- Farès Belhadj, pour son aide à comprendre les formules mathématiques.
- Louis Falissard, pour son aide à mieux comprendre Matplotlib.
- Ainsi qu'à toute l'équipe pédagogique de l'université Paris 8, qui ont pu m'aider de près ou de loin.