ASINSA gr.96
ROSART Vittore
PASCAL Antoine
OLMETA Vincent
CHAPARD Clément

RISK - Fonction intéressante

Projet d'informatique - 2ème année INSA de Lyon

VUE D'ENSEMBLE

Dans le cadre de notre projet, nous avons décidé d'implémenter l'algorithme de Dijkstra. Cet algorithme est très efficace quand il s'agit de trouver le chemin le plus court entre deux sommets d'un graphe pondéré. Il permet ainsi de trouver le chemin entre deux territoire nécessitant de faire face au moins de régiments possible. Nous allons détailler les grandes étapes de son fonctionnement. L'algorithme est implémenté dans la fonction *suggestion_trajet*. Cette fonction prend en paramètre un territoire de départ et un territoire d'arrivée, et renvoie la liste des étapes composant le chemin le plus facile pour aller du premier au deuxième territoire.

Lors de la réalisation de l'algorithme, nous nous sommes beaucoup appuyé sur l'algorithme de Dijkstra tel que nous l'utilisons en mathématique, pour pouvoir se représenter plus facilement les différentes étapes logique. Cela explique certains choix de conception. Nous allons détailler les grandes étapes de son fonctionnement.

GRANDES ÉTAPES

Initalisation

Nous initialisons d'abord les variables nécessaires. Nous avons la liste des candidats, qui est une copie de la liste de tout les territoire. Il faut bien faire attention utiliser la méthode *copy()* pour ne pas endommager la liste qui est utiliser à de nombreuses reprises dans le programme. J'initialise ensuite une representation des adjacences en dictionnaires de dictionnaires. Pour chaque territoire, on a associe un dictionnaire qui contient tout les territoires adjacents ainsi que le nombre de troupe sur ces territoires. Le nombre de troupe sur les territoire adjacent représente la pondération des arrêtes.

Ensuite, on crée un dictionnaire *chemin*, qui a chaque territoire associe le sommet depuis lequel, pour le moment, a le moins de poid, ainsi que le poids pour l'atteindre.

On crée ensuite une liste *parcouru*, qui représente les territoire que l'algorithme n'a plus à traiter, c'est donc l'inverse de la liste *candidats*. Les élements de cette liste sont ceux pour lesquels on aurait rayé la colonne dans une application mathématique de l'algorithme.

Enfin, on met un poids infini à tout les territoires autre que le territoire initial pour commencer.

Boucle principale

On répète toutes les étapes décritent dans les paragraphes ci-dessous tant qu'on a pas atteint le territoire d'arrivée, ou tant qu'il y a encore des candidats disponible. Cette dernière condition n'est pas nécessaire mais pourrait éviter un problème de *list out of range* en cas d'un problème quelconque.

On parcours d'abord tout les sommets qui n'ont pas encore été visités. On calcule ensuite le poids pour atteindre ce sommet depuis le dernier visité. Si ce poid est plus faible que le précédent poid calculé pour atteindre ce sommet, on garde en mémoire cette valeur pour ce sommet, ainsi que son prédessesseur.

On récupère ensuite le sommet avec le poids le plus faible. On donne une priorité au territoire que l'on veut atteindre dans le cas ou il y aurait plusieurs chemins possible à égalité. Ce n'est pas forcément nécessaire pour le bon fonctionnement, mais permet d'arrêter la boucle plus tôt, ce qui est intéressant du point de vu de la complexité.

On enlève ensuite le candidat séléctionné de la liste *candidats* et on l'ajoute à la liste *parcourus*.

Obtention des étapes

Une fois qu'on a parcouru notre graphe jusqu'au territoire d'arrivée, il faut remonter à la liste des étapes. Pour cela, on remonte les antécedents de chacun des territoires depuis le territoire d'arrivée, jusqu'à ce qu'on arrive au territoire de départ.

La fonction retourne alors la liste des étapes