# Introduction

## Énoncé

Analyse et développement d’un

algorithme de recherche du plus court chemin

## Principe

Un fichier Excel est fourni. Celui-ci contient trois données :

* Une carte de sommets.
* Une liste de sommets stratégiques (rouge).
* Une liste de sommets d’intérêts (vert).

La carte met en évidence le coût de déplacement pour se rendre à chaque sommet, à partir de son voisin. Il est à noter que la carte contient des sommets inaccessibles, repérés par des cases noires, et le poids « -1 ».

Le but est de trouver le chemin qui rapporte le plus de points en passant par des points obligatoires, et des points bonus.

## Calcul des points

La « distance » d’un chemin est calculée en cumulant la somme des poids des sommets, qui constituent ce chemin.

Le « bonus » est l’addition des points décrits ci-après :

* 30 points par sommet stratégique.
* Un nombre de points, définit par le fichier Excel, pour chaque sommet d’intérêt.

Les « points » de chaque chemin sont donc calculés de cette manière :

**Points = Bonus - Distance**

## Cahier des charges

Il a été demandé que seuls les sommets stratégiques 1, 3, 6 et 7 constituent obligatoirement le chemin retourné.

Les déplacements en diagonales étaient eux, interdits.

## Algorithmie de base

Dans un premier temps, nous calculerons les chemins les plus courts entre chaque sommet « spécial » (sommet de départ, sommets stratégiques et sommets d’intérêts).

Nous utiliserons pour cela un algorithme de calcul du plus court chemin : Dijkstra.

Nous garderons également en mémoire la distance minimale entre chaque sommet.

A picture containing diagram, square, line, pixel

Description automatically generated

Lorsque cela sera fait, nous créerons une liste d’ordres possibles de ces sommets. Nous générerons donc toutes les permutations de ces ordres. Celles-ci considèreront évidemment lorsque les sommets sont obligatoires ou non.

Lorsque cela est fait, nous calculons pour chaque ordre possible les points produits par le chemin.

Nous sélectionnons alors le chemin qui génère le plus de points.

Ce chemin est alors affiché de deux manières.

Premièrement, il est affiché sur la console. Chaque sommet s’affiche, et des statistiques (distance, bonus et points) sont également affichées.

Secondement, une copie du fichier Excel est produite. Celle-ci mets en surbrillance le chemin utilisé, en changeant la couleur de fond des cellules. Les cellules qui possèdent déjà une couleur ne sont pas concernées.

# Relations entre entités

A picture containing screenshot, square, rectangle, post-it note

Description automatically generated

# Pseudo-code

# Conclusion

Nous avons choisi d’utiliser le langage JavaScript pour l’algorithmie de ce projet pour deux raisons.

La première est évidemment la facilité avec laquelle ce langage peut être exécuté chez les clients. En effet, il suffit à l’utilisateur d’avoir un navigateur pour pouvoir l’exécuter.

La seconde est la simplicité de sa syntaxe. En effet, si ce langage n’est pas toujours adapté à de plus gros projets, celle-ci est « minimaliste ». Cela la rend beaucoup plus simple à lire et à écrire.

Le projet est complètement fonctionnel.

À la fin de l’exécution, voici la sortie produite sur la console.

Chemin [(x, y), ...] : [(10, 19), (9, 19), (8, 19), …, (18, 1), (19, 1), (19, 2)]

Distance : 137

Bonus : 188

Points : 51

Temps d'exécution : 643.792 secondes

Le fichier produit affiche, lui, cette grille.

A picture containing screenshot, text, pixel, colorfulness

Description automatically generated

Cependant, j’ai identifié des hypothèses permettant une optimisation majeure du temps d’exécution.

Il est théoriquement par exemple possible d’éliminer certaines permutations. Celles qui incluent des paires de sommets trop éloignés, par exemple.

Il est aussi possible de multi-threader le calcul des points de chaque chemin prétendant ; le JavaScript étant mono-thread par défaut.

Il est à remarquer que le temps d’analyse, ainsi que de développement, relatif à ce projet s’élève à plus ou moins 35 heures.