

# RAPPORT DU PROJET TRANSVERSE

Equipe Eyjafjallajökull

Guillaume CHEVALIER, Gaëtan FAUCHER, Tristan LE BRAS, Amaury PAQUIS-THONAT, Vincent ROCHE

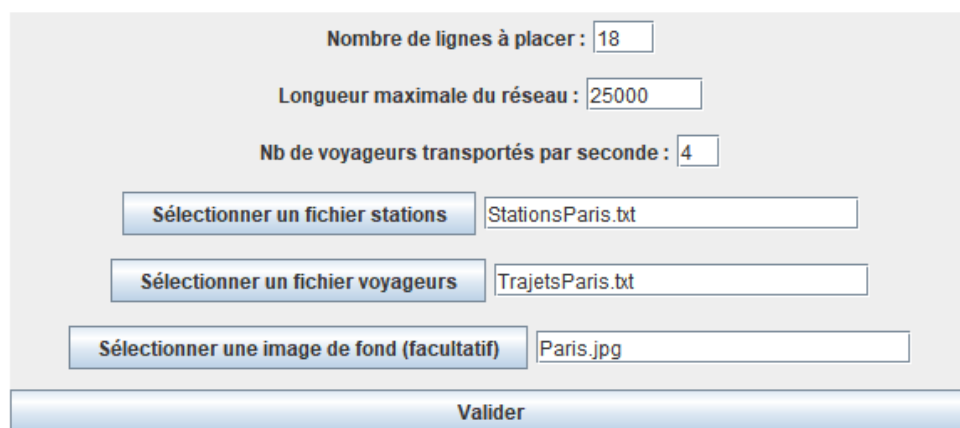
## But du programme

Le but de notre intelligence artificielle est, à partir d'une carte de stations de métro et d'une liste de trajets que souhaitent faire les voyageurs, de lier ces stations par des lignes de façon à obtenir le réseau le plus efficace possible, c'est-à-dire que la moyenne des temps de trajet des voyageurs soit la plus faible possible. Nous utilisons dans notre exemple le réseau de métro de Paris, mais n'importe quel réseau peut être créé, et cela peut aussi correspondre à d'autres moyens de transport que le métro.

Nous avons écrit notre programme en Java, son interface graphique utilise Swing.

## Fonctionnalités du programme

Dans un premier temps, l'utilisateur choisit les critères que le réseau créé par l'intelligence artificielle devra respecter, ainsi que les fichiers contenant les données sur lesquelles se baser pour les différents calculs.



The screenshot shows a Java Swing window for configuring the program. It contains several input fields and buttons. At the top, there are three labels with corresponding text boxes: 'Nombre de lignes à placer : 18', 'Longueur maximale du réseau : 25000', and 'Nb de voyageurs transportés par seconde : 4'. Below these are three rows, each with a button and a text box: 'Sélectionner un fichier stations' with 'StationsParis.txt', 'Sélectionner un fichier voyageurs' with 'TrajetsParis.txt', and 'Sélectionner une image de fond (facultatif)' with 'Paris.jpg'. At the bottom is a large 'Valider' button.

Le nombre de lignes à placer est le nombre de lignes que devra contenir le réseau final, après sa construction. Pour l'exemple, nous essayons de reprendre les caractéristiques du réseau de métro parisien afin de voir les différences entre les résultats donnés par notre intelligence artificielle et la réalité, le nombre de lignes est donc de 18 (lignes 1 à 14, 3 bis, 7 bis, et les branches des lignes 7 et 13 car notre programme ne prend pas en compte les branches).

La longueur maximale du réseau est la longueur maximale de rails qui devra être construite par les nouvelles lignes, cela permet en quelque sorte d'imposer une limite de budget à notre intelligence artificielle. L'unité de cette valeur est la même que celle qui sera utilisée par les coordonnées des stations, afin que l'utilisateur puisse utiliser l'échelle qu'il souhaite. La valeur 25 000 correspond environ aux 220 km du réseau parisien.

Le nombre de voyageurs transportés par seconde représente le débit des lignes. Il s'agit du nombre de voyageurs pouvant passer par un arc en une unité de temps. Cela permet de simuler les capacités maximales, les fréquences des trains et les saturations lors du calcul des temps de trajet. Sur le réseau parisien, il y a en moyenne 4 voyageurs par seconde et par ligne selon nos calculs.

L'utilisateur doit ensuite choisir l'emplacement des fichiers de stations et de voyageurs qui seront utilisés.

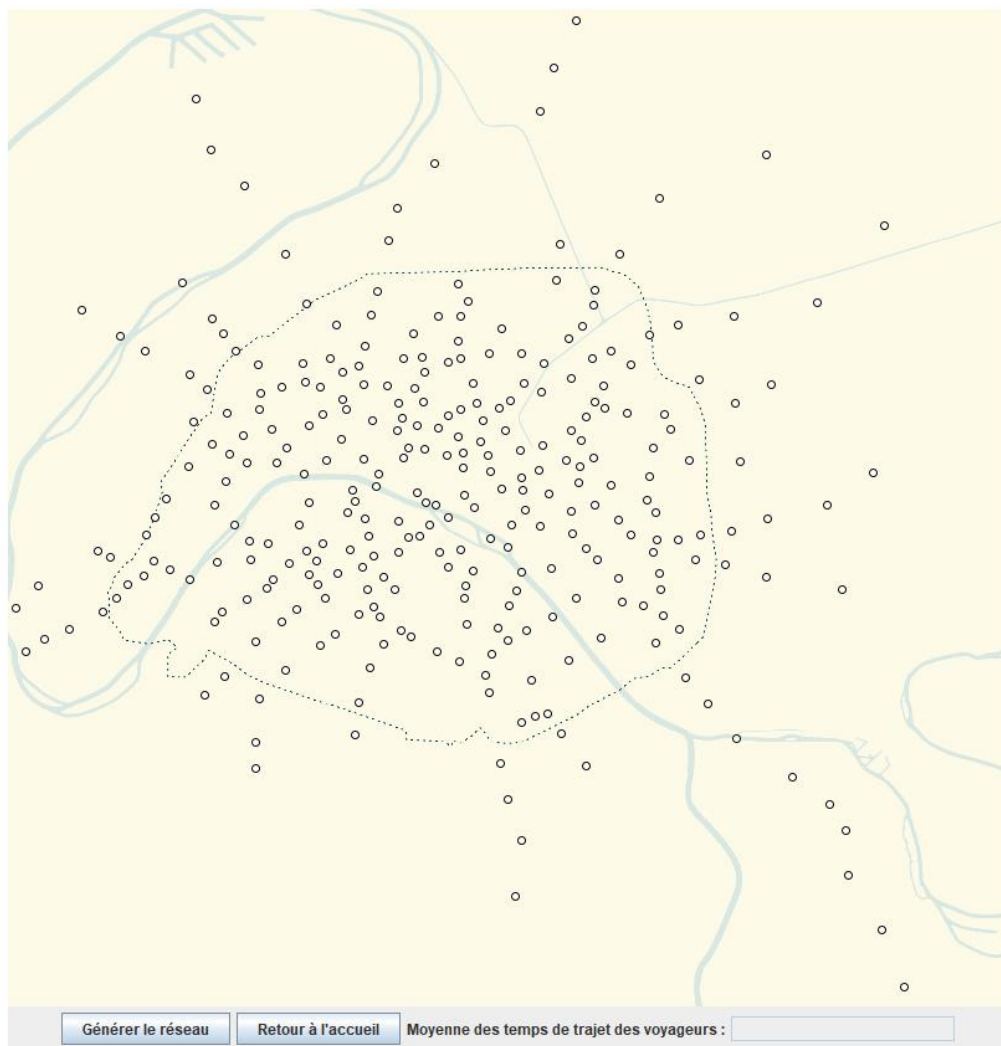
Le fichier stations contient à chaque ligne le nom de la station (ils doivent tous être différents), suivi de sa coordonnée X, puis de sa coordonnée Y (en pixels depuis le coin supérieur gauche de la fenêtre, la taille de la fenêtre s'adaptera automatiquement).

Le fichier voyageurs contient à chaque ligne le nom d'une station de départ du trajet, suivi du nom d'une station de destination, suivi du nombre de voyageurs souhaitant faire ce trajet. Chaque station doit avoir au moins un voyageur qui en part ou qui s'y rend.

Pour notre exemple concernant le métro de Paris, nous avons utilisé les coordonnées réelles des stations, ainsi que les données sur les nombres de voyageurs entrant à chaque station en 2017. Ces données proviennent du site <https://opendata.stif.info/explore/dataset/validations-sur-le-reseau-de-surface-nombre-de-validations-par-jour-2e-sem/>. Seules les données sur l'origine des voyageurs sont disponibles, nous avons donc comparé les tranches horaires du matin et du soir en semaine afin d'obtenir des trajets entre ce qui est de manière générale le domicile et le lieu de travail des voyageurs. Nous obtenons donc finalement la proportion de voyageurs se rendant à chaque station ce qui nous permet de créer les trajets et de remplir notre fichier.

Notre programme permet également d'ajouter un fond à la carte des stations, celui-ci est donc une image à choisir sur cette première fenêtre. Cela est facultatif. Le fichier *Paris.jpg* est un fond de carte de la région parisienne qui correspond aux stations utilisées dans notre exemple.

Après avoir validé ces critères, l'utilisateur arrive sur la fenêtre de la carte du réseau, avec chaque station placée sur le fond choisi :



Le nom des stations s'affiche au passage de la souris.

Le bouton « générer le réseau » exécute l'intelligence artificielle qui place les lignes entre les stations. L'utilisateur peut constater le fonctionnement de l'intelligence artificielle en direct car la carte se met à jour en permanence durant son fonctionnement. Une fois les calculs terminés, l'utilisateur est averti du nombre de stations reliées et le calcul des temps de trajet des voyageurs commence afin d'en afficher la moyenne en bas à droite de la fenêtre.



L'utilisateur a maintenant la possibilité de retourner à l'accueil du programme s'il veut tester à nouveau l'intelligence artificielle avec des critères différents.

## Implémentation

Nous avons mis en œuvre le modèle MVC afin que les diverses classes soient rangées de manière logique.

### Représentation du réseau en mémoire

Cinq classes représentent le réseau :

- Réseau : avec la liste des stations, arcs, voyageurs et lignes
- Station : avec ses coordonnées, son nom et la liste des chemins les plus courts vers toutes les autres stations (remplie après exécution de l'algorithme de Dijkstra)
- Arc : avec les stations à ses extrémités et la ligne dont il fait partie, ainsi que des informations sur les flux de voyageurs utilisées lors du calcul des temps de trajet

- Voyageur : avec une station d'origine, une station de destination, et diverses données utilisées lors du calcul des temps de trajet qui représentent par exemple la position du voyageur sur le réseau
- Ligne : avec son numéro et sa couleur (couleurs assignées automatiquement, nous avons repris les couleurs du réseau parisien)

## Intelligence artificielle

Nous avons prévu la possibilité d'implémenter plusieurs algorithmes permettant de placer les lignes, c'est pourquoi une classe abstraite *IACreationLignes* est présente. Finalement, nous n'avons qu'une intelligence artificielle d'implémentée, dans la classe *IACriteresPonderes*.

Notre algorithme fonctionne en plusieurs étapes :

1. Création d'un réseau reliant toutes les stations, sans arcs qui se croisent.
2. Calcul des chemins les plus courts pour chaque voyageur sur ce réseau complet avec l'algorithme de Dijkstra, ce qui donne le flux de chaque arc (le nombre de voyageurs dont le trajet y passe).
3. Placement des lignes
  - a. Le premier arc de la ligne est placé sur l'arc au flux maximal parmi les arcs pas encore placés dans le réseau.
  - b. La ligne est prolongée à chacune de ses extrémités, en choisissant le meilleur arc. Pour cela, divers critères sont pris en compte comme le flux des arcs, l'angle formé avec le dernier arc placé (pour éviter les virages trop serrés), la longueur des arcs, le nombre de correspondances déjà existantes. Ces critères ont plus ou moins d'importance, cela est défini par les valeurs définies au début du fichier *IACriteresPonderes.java*.
  - c. Arrêt du prolongement de la ligne lorsqu'aucun arc ne répond aux critères (surtout d'angle).
4. Fusion des lignes qui se trouvent bout à bout (qui ont un terminus à une même station) afin de pouvoir gagner une ligne supplémentaire à placer.
5. Raccordement des stations encore isolées
  - a. Poursuite des lignes vers les stations proches des terminus, avec une plus grande tolérance d'angle pour les virages.
  - b. Exécution d'un algorithme permettant de lier aux lignes existantes les stations isolées qui en sont proches, sans faire de trop grands détours.

## Calcul des temps de trajet

*Méthode « evaluer » de la classe « Reseau »*

Une fois le réseau créé, l'algorithme de Dijkstra est exécuté afin d'obtenir les arcs par lesquels passe chaque voyageur. Une boucle est ensuite exécutée tant que tous les voyageurs ne sont pas arrivés. À chaque itération, chaque voyageur avance d'une unité de distance. Un temps supplémentaire est appliqué lorsque le voyageur doit changer de ligne, et il doit attendre si le nombre de voyageurs déjà passés par un arc est supérieur au débit maximal défini au début du programme.

À la fin, on obtient le nombre d'itérations qu'il a fallu à chaque voyageur pour arriver à destination et on en calcule la moyenne.