Potagibus
Rapport

Pierre GUYOT
Pierre-Yves JACQUIER
Arnaud KRAFFT
Titouan LANGLAIS

22 Mai 2023

Table des matières

1 Introduction

Ce document sert à résumer l'Etat de l'Art effectué au début du projet ainsi que de décrire les structures de données, algorithmes et fonctions utilisés lors de la création de l'application. Il sert également à étudier les divers tests effectués ainsi qu'à revenir sur l'aspect Gestion de Projet.

Tout d'abord, Potagibus est une application permettant aux usagers de voiture électrique de connaître l'itinéraire qu'ils doivent prendre pour arriver à leur destination sans que leur véhicule ne tombe à court de batterie. L'utilisateur va renseigner son point de départ, son point d'arrivée ainsi que le modèle de sa voiture. L'application lui renverra alors la liste des bornes de rechargement par lesquelles il doit passer pour arriver à destination le plus tôt possible.

Potagibus peut également simuler des utilisateurs afin d'étudier l'encombrement du réseau de bornes.

2 Etat de l'Art

Dès le début du projet, il nous a été clair que l'aspect le plus important du projet serait l'algorithme permettant d'obtenir le chemin le plus court entre le point d'arrivée et le point de départ de l'utilisateur. Nous avons donc fait des recherches sur les différents algorithmes permettant d'obtenir de tels chemins. Nous en avons trouvé cinq, Dijkstra, Bellman-Ford, A*, Anytime A* et Jump Point Search.

2.1 Dijkstra et Bellman-Ford

Nous avons rapidement abandonné les algorithmes Dijkstra et Bellman-Ford à cause de leur complexité.

En effet, pour E arrètes et V noeuds, la complexité est en :
- O(E + Vlog(V)) pour Dijkstra.
- O(V(V + E)) pour Bellman-Ford.

2.2 JumpPoint Search

Après avoir réalisé des recherches sur l'algorithme Jump Point Search, nous avons décidé de ne pas l'utiliser. Cet algorithme est fait pour fonctionner avec une carte séparée en cases. Ici, comme nous avons les coordonnées géographiques de chaque borne de rechargement, nous avons choisi une matrice d'adjacence, qui n'est pas utilisable avec cet algorithme.

2.3 A* et Anytime A*

L'algorithme A* est l'algorithme que nous avons utilisé. Cet algorithme a une complexité en O(E) avec E le nombre d'arètes et convenait parfaitement à notre problème.

l'algorithme Anytime A* est un dérivé du A*. Il permet de donner des résultats plus rapidement avec la même complexité. Nous avons décidé de le garder de côté afin de potentiellement l'utiliser à la place du A* si nous avions le temps, mais malheureusement ce dernier nous a manqué à ce niveau.

3 Parsing des données

Pour ce projet, nous avions accès à deux bases de données : d'une part un gargantuesque fichier recensant les bornes de recharge de voiture électrique en France, et d'une autre part un site internet donnant des chiffres concernant certaines voitures électriques. Nous devions traiter ces données pour extraire ce qui était important pour le projet.
Pour ce faire, nous avons utilisé des scripts python lisant des .csv pour les filtrer.
Pour les voitures, cela s'est en partie réalisé en copiant simplement la page web qui nous était donnée et l'arranger pour qu'elle soit lisible par un programme. Les données présentes sur les listes étant insuffisantes, nous avions dû utiliser une base de donnée faite main.

Concernant les bornes, nous avions un .csv comportant une cinquantaine de propriétés, certaines plus importantes que d'autres, mais aussi pas tous complètes. Le parsing de ce document était une aventure à elle seule. Beaucoup d'incohérences dans la rédaction du document, la présence de retours à la ligne dans certaines adresses et tant d'autres facteurs ont rendu la tâche très difficile. Au final nous avons dû créer une fonction lisant le fichier caractère par caractère pour le transformer en une liste de listes.
En suite, on utilise la fonction filtre du fichier filtrestations.py qui permet de générer un fichier .csv répondant aux besoins du programmeur. Cette fonction a plusieurs fonctionnalités :

- Elle ne prend que les paramètres qui sont souhaités (dans la BD utilisée dans Pota'Gibus, il s'agit de : ID, longitude, latitude, nombre de points de charge et puissance nominale) (paramètre "proprietes")
- Elle peut refuser des stations qui sont trop proches d'une autre (permettant de réduire le nombre de points : s'il y a une dizaine de stations dans un rayon d'1km, pourquoi toutes les garder ?) (paramètre "distmin")
- Elle peut refuser les stations qui ont pour ID "Non Concerné", nous ne savons pas ce que ça implique pour la station, mais nous préférons les retirer. (paramètre "concerne")
- Elle peut garder que les stations en accès libre. (paramètre "libre")
- Elle peut refuser les doublons : si deux stations ont les mêmes coordonnées, seule la première figurant dans la base de données sera conservée (paramètre "doublons")
- On peut choisir le nom du fichier de sortie (paramètre "nomsortie")
- On peut décider de convertir toutes les puissances nominales en watts (paramètre "watts")

Le fichier en sortie ne possède pas les impuretés de la base de données originale et peut être utilisée dans le programme C sans souci.

4 Structures de données

4.1 Unité des données

Mise à part contre indication, les données présentent ici suivent ces unités :
— Temps en heures
— Capacité en W.h
— Recharge en W
— Distance en km
— Angle (longitude et latitude) en degrés

4.2 Coordonnées

La première structures de données que nous avons créée est une structure nommée coord et définie par :

```
struct coord
{
    double x;
    double y;
} typedef coord;
```

Tous les coordonnées que nous avons utilisées sur ce projet son sous la forme de cette structure.

4.3 chemin

Cette structure de données n'est utilisée uniquement dans l'algorithme du A*. La sortie du A* est une structure de type **chemin_tab_struct**. Chemin est une liste chaînée d'éléments chemin. Lorsqu'elle est crée, elle doit être combinée avec la structure **garbage_chemin** afin d'être détruite à la fin du programme. Cette structure stocke différents éléments importants pour la recherche du plus court chemin dans le A*.

```
struct chemin
{
    struct chemin* suivant;
    int id; // Identifiant de la station, -1 si depart, -2 si arrivee
    double distance_total; // Distance parcourue depuis le depart
    double distance_prochain; // Distance a la prochaine station
    double capacite_avant;
    double capacite_apres;
} typedef chemin;
```

4.4 chemin_tab

Cette structure est un élément de la structure **chemin_tab_struct**. Elle permet de créer un seul élément du tableau dans lequel elle sera implémentée. Ce n'est pas cette structure chemin que le A* utilise dans son algorithme, c'est la structure qu'il renvoie.

```
struct chemin_tab
{
    int id;
    double distance_prochain;
    double capacite_avant;
    double capacite_apres;
} typedef chemin_tab;
```

La variable ID est l'identifiant de la station associée au chemin. Si c'est le départ, alors la valeur vaut -1 et si c'est l'arrivée alors elle vaut -2. distance_prochain est la valeur de la distance jusqu'à la prochaine station. Si c'est l'arrivée, la valeur est égale à -1. On stocke aussi la capacité de la voiture avant et après rechargement à la station. Pour le départ et l'arrivée, les valeurs de capacite_avant et capacite_après sont identiques.

4.5 chemin_tab_struct

Cette structure permet de stocker facilement tout ce qui est nécessaire pour établir le tableau du chemin. Cela signifie avoir besoin de la voiture qui a servi à créer ce chemin, et aussi le tableau des stations et des voitures. D'un point de vue rétrospectif, il aurait été aussi intéressant d'y ajouter les coordonnées du départ et de l'arrivée pour éviter de manipuler plusieurs structures à la fois.

```
struct chemin_tab_struct
{
    chemin_tab* tab;
    int taille;
    station_tab* s_tab; // Lien vers le tableau de stations
    int id_v;
    voiture_tab* v_tab; // Lien vers le tableau de voitures
} typedef chemin_tab_struct;
```

4.6 file

Cette structure est une file de priorité. Cette dernière est triée en fonction de la variable distance_approche et la valeur intéressante qu'elle stocke est le **chemin**. À chaque ajout d'élément, la file est automatiquement triée.

```
struct file
{
```

```
    struct file* suivant;
    struct file* precedent;
    chemin* chemin;
    double distance_approche;

} typedef file;
```

4.7 garbage_chemin

Cette structure est un tableau qui s'agrandit lorsqu'il n'est plus assez grand. Cela aurait très bien pu être une liste, car il n'est pas nécessaire de chercher des éléments dedans. Mais son design initial avait besoin de cette fonctionnalité, qui est devenu optionnelle par la suite.

```
struct garbage_chemin
{
    chemin** chemin;
    int taille;
    int taille_max;
} typedef garbage_chemin;
```

4.8 list_t

La structure de données list_t correspond à une liste chaînée qui a comme corps des élément de type coord. Elle est définie par :

```
struct list_t
{
    coord *element;
    struct list_t *next;
} typedef list_t;
```

4.9 matrice_inf

La structure de données matrice_inf sert à représenter la matrice d'adjacence qui sera utilisée par l'algorithme A*. Elle comporte un entier représentant la taille du tableau ainsi qu'un tableau de flottants représentant la matrice. Cette structure de données est définie par :

```
struct matrice_inf
{
    int taille;
    float** mat;
} typedef matrice_inf;
```

4.10 station

La structure de données station est celle qui permet de regrouper toutes les informations utiles sur les stations de rechargement après le parsing. On peut y retrouver l'ID de la station, ses coordonnées, le nombre de points de charge total, le nombre de points de charge disponibles ainsi que la puissance des bornes de la station. Elle est définie par :

```
struct station
{
    char id[LENGTH_ID];
    coord* coord;
    int nbre_pdc;
    int nbre_pdc_dispo;
    int puissance;
} typedef station;
```

4.11 station_tab

La structure de données station_tab permet de regrouper en un tableau des pointeurs vers toutes les stations, ainsi qu'un entier représentant la taille du tableau. Elle est définie par :

```
struct station_tab
{
    int taille;
    station* tab;
} typedef station_tab;
```

4.12 corresp_station_tab

La structure de données corresp_station_tab regroupe dans un tableau des ID de stations après que celles-ci aient été triées par "remplir ici". Elle contient également dans un entier la taille du tableau. Elle est définie par :

```
struct corresp_station_tab
{
    int taille;
    int* tab_id;
} typedef corresp_station_tab;
```

4.13 utilisateur

La structure de données utilisateur est une liste chaînée permettant de regrouper les informations sur des utilisateurs générés aléatoirement pour la simulation. Elle regroupe un nom de modèle de voiture et deux structures coord correspondant aux points de départ et d'arrivée. Elle est définie par :

```
struct utilisateur
{
    char* voiture;
    coord* depart;
    coord* arrivee;
    struct utilisateur* next;
} typedef utilisateur;
```

4.14 utilisateur_info

La structure de données utilisateur_info permet de regrouper les informations concernant les utilisateurs pendant la simulation. Elle contient un entier ID_courrant qui correspond à l'ID du chemin sur lequel l'utilisateur se trouve, Nb_ticks_attente qui correspond à l'avancement dans la simulation ainsi qu'un pointeur vers un élément de la structure chemin_tab_struct qui correspond au chemin entre les point de départ et d'arrivée sur lequel l'utilisateur se trouve. Cette structure est définie par :

```
struct utilisateurinfo
{
    int ID_courrant;
    int Nb_ticks_attente;
    chemin_tab_struct* chemin;
} typedef utilisateurinfo;
```

4.15 utilisateurtrajet

La structure de données utilisateurtrajet est une liste chaînée permettant de regrouper toutes les informations sur chaque utilisateur pendant la simulation. Son corps est constitué d'un pointer vers un élément de la structure utilisateur_info. Elle est définie par :

```
struct utilisateurtrajet
{
    utilisateurinfo* info;
    struct utilisateurtrajet* next;
} typedef utilisateurtrajet;
```

Lors de la simulation on a besoin de connaître le premier utilisateur, et pour pouvoir nous arrêter au bon moment, nous avons besoin de connaître le nombre d'utilisateurs. Nous avons créé cette structure :

```
struct utilisateurtrajet\_header
{
    utilisateurtrajet* first;
    int size;
}typedef utilisateurtrajet_header;
```

4.16 visite_tab

Cette structure de données est un simple tableau permettant de savoir si une station a été visitée.

```
struct visite_tab
{
    int* tab;
    int taille;
} typedef visite_tab;
```

4.17 voiture

La structure de données voiture est celle qui permet de regrouper toutes les informations utiles sur les modèles de voitures après le parsing. On peut y retrouver le nom du modèle, sa capacité en kilomètres, son efficacité en Watt par heure par kilomètre et sa vitesse de charge. Elle est définie par :

```
struct voiture
{
    char name[LENGTH\_NAME];
    int range;
    int efficiency;
    int fast_charge;
} typedef voiture;
```

4.18 voiture_tab

La structure de données voiture_tab permet de regrouper en un tableau des pointeurs vers toutes les voitures, ainsi qu'un entier représentant la taille du tableau. Elle est définie par :

```
struct voiture_tab
{
    int taille;
    voiture* tab;
} typedef voiture_tab;
```

5 Algorithmes

5.1 Recherche du meilleur chemin - A*

Le but du projet était de trouver le chemin le plus avantageux en partant d'un point A pour arriver à un point B tout en passant par un certain nombre de stations. Au préalable, une matrice d'adjacence est créée permettant de regrouper toutes les distances entre les stations, pour éviter de les recalculer. De plus, à partir de la capacité de la batterie et de son efficacité, nous pouvons connaître la distance de parcours maximale par arêtes.
L'algorithme se divise en deux parties, une phase d'initialisation et une phase de recherche. Il fonctionne de la même manière que l'algorithme de Dijkstra, à la différence qu'il prend en compte la distance supposée qu'il lui reste jusqu'à l'arrivée. Cela permet d'éviter de calculer des possibles chemins n'allant pas vers la bonne direction.
Un pseudo-code est disponible en annexe, mais il date des premières versions, donc il n'est plus d'actualité.

Le code utilise quatre structures de données. Un vif rappel est fait ici, mais il est possible d'en connaître plus grandement les détails dans la partie structure de données. Une **file de priorité** qui va trier de la plus petite à la plus grande les valeurs. Cela permettra de savoir quels prochains **chemins** il faudra traiter, soit celui permettant de minimiser la distance, donc avec la valeur la plus faible.
Une structure **chemin** qui permet de construire une liste chaînée d'élément **chemin**. Chaque éléments chemin correspond à une station, et son prochain élément est la station précédente. Cette structure est spéciale car plusieurs **chemins** peuvent être reliés au même élément **chemin**. On peut le visualiser comme un arbre avec comme racine le **chemin** du point de départ, et se divisant en plusieurs **chemins** correspondant à toutes les stations parcourues.
Afin de facilement supprimer tout cela, tous les éléments **chemins** sont stockés dans une structure nommée **garbage chemin** qui se charge de collecter tous ces éléments puis les supprimer à la fin de l'algorithme. C'est une simple liste chaînée.
Enfin, une dernière structure du nom de **visite** est utilisée. Celle-ci est un simple tableau permettant de savoir de manière immédiate si un point a déjà été visité.

L'initialisation consiste à implémenter le premier **chemin** comme le point de départ dans la **file de priorité**.
La phase de recherche consiste en une boucle **while** qui ne s'arrête qu'une fois que la file est vide ou que le prochain élément dans la **file de priorité** est le point d'arrivée. Chaque itération de la boucle consiste en premier lieu à récupérer le premier **chemin** de la **file de priorité**. On extrait ensuite l'ID de la station associée à ce **chemin**. Si c'est une station déjà visitée par l'algorithme (à l'aide de la structure **visite**), alors on passe à une nouvelle itération de la boucle. Sinon on considère la station comme vue par l'algorithme et on effectue les prochaines instructions de la boucle. On calcule ensuite la distance maximale que peut effectuer l'usager depuis cette station. Cela va dépendre de la capacité de la voiture à la sortie de cette station, et de l'efficacité de la voiture. C'est

une valeur récupérée de la structure chemin.
En suite, on va récupérer la liste de tous les voisins de cette station qui ne sont pas encore visités et ayant une distance de notre station inférieure à la distance maximale calculée précédemment. Chaque voisin est alors mis dans un **chemin**. On calcule alors pour ce **chemin** la capacité à l'entrée de la station, et la capacité après rechargement dans la station.
Afin de pouvoir mettre le **chemin** dans la **file de priorité**, il va falloir trouver une valeur qui permet d'évaluer la pertinence d'un **chemin** par rapport à un autre. La pertinence d'un **chemin** se retrouve dans la distance parcourue depuis le début de **chemin**, soit le départ. Un autre facteur de pertinence est le temps et l'efficacité de la recharge dans une station, que l'on peut transformer en une distance de rechargement. Enfin, une estimation de la distance restante est également importante. Cette estimation est simplement la distance entre la station courante et l'arrivée le tout multiplié par un paramètre nommé lambda. Ce paramètre lambda est une constante qui permet d'améliorer la vitesse de l'algorithme au détriment de trouver le meilleur chemin. Afin de trouver le meilleur chemin, lambda doit être mis à la valeur 1. Afin d'avoir un algorithme plus rapide, il est possible d'augmenter la valeur (le plus souvent mis à 1.5). Cela permet à l'algorithme de privilégier les **chemins** qui viennent d'être ajoutés dans la **file de priorité** plutôt que les anciens **chemins**.
Cette valeur pour la **file de priorité** est au final la somme de la distance depuis le départ avec l'estimation de la distance restante, le tout soustrait par la distance de rechargement. On ajoute donc le **chemin** avec sa valeur à la **file de priorité**, puis on accède à la prochaine itération de la boucle.
L'algorithme se termine en renvoyant le chemin s'il a été trouvé, ou une valeur NULL. De plus, le chemin renvoyé est une transformation en tableau du nom de **chemin_tab_struct** de la stucture **chemin**.

La complexité de l'algorithme est assez importante en mémoire à cause de la matrice d'adjacence O((nombre de points)^2) par rapport aux autres structures qui sont simplement en O(nombre de points). Sa complexité algorithmique est un peu plus complexe. En effet celle ci dépend à la fois du tri de la file de priorité mais également si le point possède bien un chemin. Lorsque le point ne possède pas de chemin, l'algorithme se force à tester toutes les possibilités comme un Dijkstra. On a alors mis en place un système de Timeout pour éviter cela. On peut savoir que sa complexité temporelle maximale ne peut dépasser celle d'un Dijkstra. Mais sa vraie complexité temporelle dépendra du facteur lambda, de la structure du réseau et de la distance maximale de la voiture.

5.2 Simulation

Le but de la **simulation** est de simuler l'occupation du réseau de stations de rechargement en France. Elle se base sur une représentation temporelle discrète (que nous appellerons des "ticks"), avec chaque tick valant 10 minutes.
La simulation se lance depuis la ligne de commande en renseignant un nombre d'utilisateurs, notée N par la suite. Elle crée des fichiers numérotés d'après le numéro de tick, et qui

contiennent les coordonnées ainsi que l'occupation des stations non-vides. L'occupation étant calculée comme le nombre de points de charge disponible sur le nombre de points de charge de la station. À noter que nous ne pouvons pas savoir avec cette modélisation le nombre de personnes dans la file d'attente.

Cet algorithme récupère la liste des stations et des voitures puis initialise N utilisateurs ayant chacun un trajet et une voiture, grâce aux fonctions **rdm_utilisateur** et **trajets** ; ceci est noté comme le tick 0. Grâce à cette initialisation, nous pouvons lancer l'hérédité qui lance le **traitement** qui va nous faire passer au tick suivant et créer un fichier de représentation du réseau (comme indiqué plus haut). Une fois qu'il n'y a plus d'utilisateur la simulation s'arrête.

La complexité de l'algorithme est vraiment importante, en effet, c'est elle qui va limiter notre simulation. Pour la complexité temporelle, nous parcourons la liste d'utilisateurs de façon linéaire, nous devrions donc avoir une complexité temporelle en O(N), avec N utilisateurs simulés. L'algorithme de simulation se base sur notre algorithme de recherche du meilleur chemin **A***, or chaque utilisateur garde en mémoire son propre chemin et les points de charge ne gardent en mémoire que le nombre de points de charges disponible. Par conséquent, nous avons une complexité spatiale, elle aussi en O(N), avec N utilisateurs simulés.

À noter que notre algorithme se base sur des fonctions qui sont en O(1) du nombre d'utilisateurs, mais qui prennent quand même beaucoup de temps et d'espace. Cela implique que nous avons un temps (resp. espace) minimum, indépendant du nombre d'utilisateurs, que nous ne pouvons pas ou peu diminuer.

6 Fonctions

6.1 Calcul de la distance

Pour calculer la distance entre deux points sur un globe, nous avions eu plusieurs idées. Tout d'abord, nous avons pensé à faire un simple calcul en supposant que la France pouvait être assimilée par un plan, une norme euclidienne. Nous nous sommes rendu compte que si nous voulions être bien plus précis, et éviter des problèmes par la suite, il valait mieux utiliser la formule de haversine qui permet de calculer la plus petite distance reliant deux points sur un globe.
Cette formule utilise plusieurs fonctions trigonométriques et prend en paramètre le rayon de la Terre, les latitudes et longitudes des deux points.

$$d = 2r \cdot arcsin\left(\sqrt{sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + cos\left(\varphi_1\right) cos\left(\varphi_2\right) sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Avec :

r le rayon de la terre
φ_i latitude du point i
λ_i longitude du point i

6.2 Sélection de points

Pour accélérer l'exécution de l'algorithme A*, nous avons décidé d'écrire une fonction **selec_point_list** qui permet de retirer des points jugés inutiles car bien trop éloignés des points de départ et d'arrivée pour être pertinent dans la simulation.
La fonction prend un ovale dont les deux foyers correspondent au point de départ et d'arrivée du chemin et garde uniquement les points situés dans l'ovale.
Bien que fonctionnelle, cette fonction nous a mis face à un problème, que nous avons nommé "l'effet lac". Il se produit lorsque les points de départ et d'arrivée sont séparés par quelque chose d'infranchissable, tel qu'une chaîne de montagne ou un lac. L'ovale ne comporte alors pas assez de point pour pouvoir avoir un chemin entre le point de départ et d'arrivée alors qu'il est possible d'en avoir un.
Nous avons résolu ce problème en ajoutant un paramètre "marge" à la fonction qui permet d'agrandir artificiellement l'ovale en cas de besoin et donc de surmonter "l'effet lac".
Outre l'ovale, nous avions envisagé de faire une sélection similaire en utilisant un rectangle à la place d'un ovale, deux coins opposés du rectangle correspondant au point de départ et d'arrivée. Après tests, nous nous sommes rendu compte que cette solution ne marchait pas car le rectangle se résumait à une ligne si les deux points étaient sur la même longitude ou la même latitude.

6.3 Génération d'utilisateurs

La fonction **rdm_utilisateur** permet de générer les utilisateurs fictifs qui seront utilisés pour la simulation.
Elle prend en argument un pointeur vers la liste des voitures, représentée par la structure de données **voiture_tab**, un pointeur vers la liste des stations, représentée par la structure de données **station_tab**, ainsi qu'un entier indiquant le nombre d'utilisateurs à générer.
Elle retourne la liste des utilisateurs sous la forme d'une liste chaînée faite avec la structure de données **utilisateur**.
Cette fonction crée la liste chaînée, puis pour chaque utilisateur jusqu'à en avoir le nombre voulu, crée un nouveau chaînon contenant un point de départ, un point d'arrivée et un modèle de voiture tous générés aléatoirement.

La fonction **trajets** permet d'obtenir les trajets utilisés par ces utilisateurs fictifs afin de pouvoir faire la simulation.
Elle prend en argument un pointeur vers la liste des utilisateurs fictifs, représentée par la structure de données **utilisateurs**, un pointeur vers la liste des voitures, représentée par la structure de données **voiture_tab** et un pointeur vers la liste des stations, représentée par la structure de données **station_tab**.
Elle retourne la liste des nouvelles informations sur les utilisateurs sous la forme d'une liste chaînée faite avec la structure de données **utilisateurtrajet**.
Cette fonction génère une matrice d'adjacence, puis pour chaque utilisateur de la liste chaînée, calcule leur chemin via la fonction a_star, qui applique l'algorithme A*.

6.4 Passage au tick suivant

Passage au tick suivant est totalement gérée par la fonction **traitement**.
Elle prend en argument un pointeur vers la tête de la structure **utilisateurtrajet**, cette tête comporte la taille de la liste chaînée et le premier élément de la liste.
Cette fonction permet de passer du tick n au tick n+1, elle parcourt de façon linéaire la liste d'**utilisateurtrajet**. Elle vérifie si l'utilisateur est à la fin du parcours, au début, à une station de rechargement ou s'il est en trajet. Si l'utilisateur est à la fin du trajet, alors elle le supprime, sinon elle continue le trajet/rechargement, ou si l'utilisateur est arrivé à une station, elle regarde s'il peut se recharger.

7 Visualisation

Nous avons construit, à l'aide de la libraire python Flask, un outil de visualisation des données, qui permet aussi d'utiliser le programme Pota'Gibus (si exécuté depuis une machine linux). Voici ses principales fonctionnalités :

7.1 Simulation : itinéraire

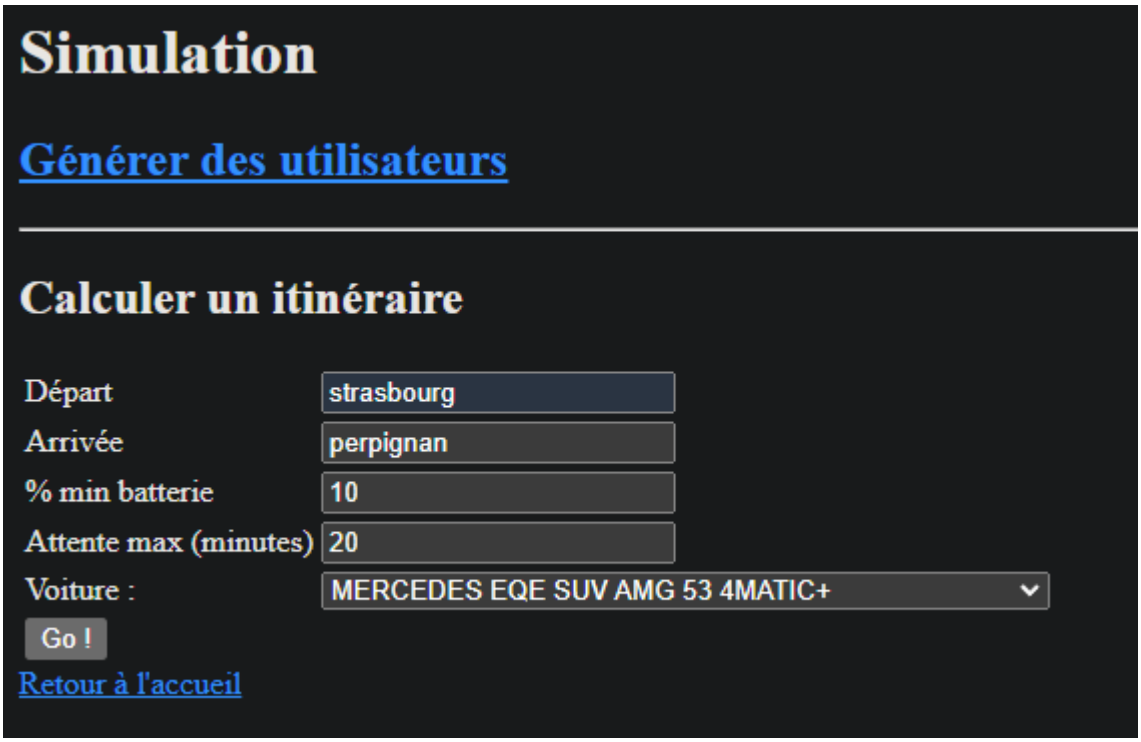


FIGURE 1 – Interface de calcul d'itinéraire

L'utilisateur peut renseigner les villes de départ et d'arrivée, le pourcentage minimal de batterie qu'il est prêt à accepter, le temps maximum qu'il est prêt à attendre à la borne et son modèle de voiture. Le site appelle en suite le programme pota_p1.exe grâce à la librairie OS de python.

7.2 Simulation : charge du réseau

FIGURE 2 – Interface de génération d'usagers

L'utilisateur peut renseigner un nombre d'usagers à simuler. Le site appelle en suite le programme simulation.exe grâce à la librairie OS de python.

7.3 Carte : itinéraire

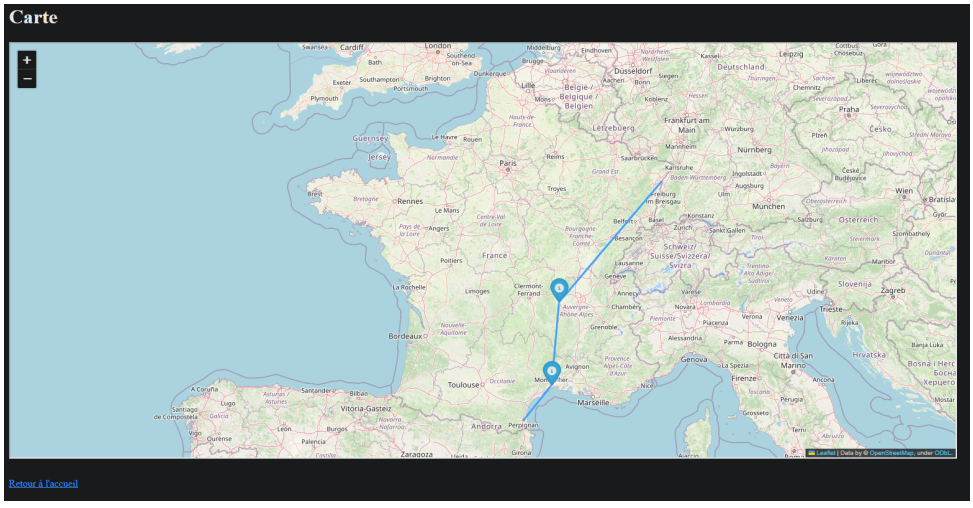


FIGURE 3 – Carte de l'itinéraire Strasbourg - Perpignan

Les marqueurs représentent les stations auxquelles l'utilisateur doit s'arrêter. Cliquer dessus affiche le temps d'arrêt en minutes.

7.4 Carte : charge du réseau

FIGURE 4 – Carte de la charge totale du réseau pour 45 usagers

Pour chaque tick, des marqueurs indiquent l'occupation d'une station, en suivant ce code couleur :

Couleur	Occupation
Vert foncé	0%
Vert clair	0% à 20%
Orange	20% à 40%
Rouge	40% à 60%
Rouge foncé	60% à 80%
Noir	plus de 80%

Le site permet aussi de visualiser tick par tick à l'aide d'un slider :

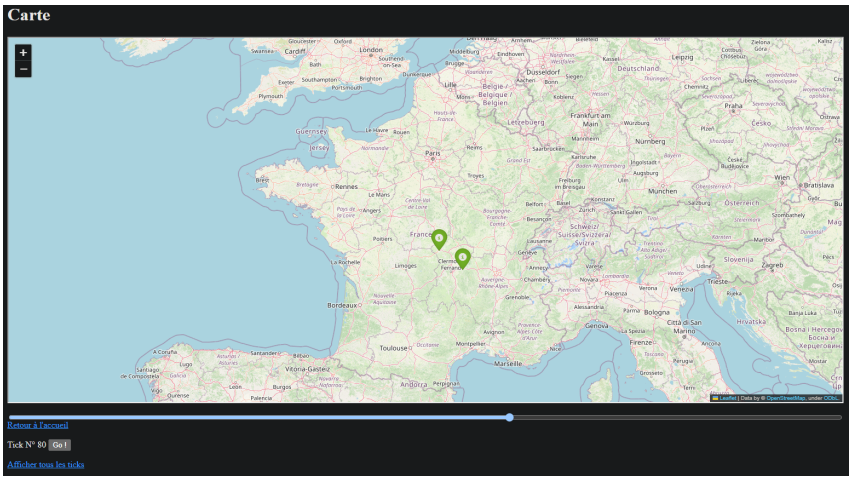


FIGURE 5 – Visualisation du tick N°80

8 Tests

8.1 Simulation

Lorsque nous avons commencé à esquisser la simulation, nous nous sommes mis d'accord sur le fait que la simulation fonctionne de façon "intuitive" avec une approche tick par tick. D'après la partie **Algorithme**, la simulation fonctionne en O(N), avec N utilisateurs.
Le graphique ci-dessous a été généré en faisant la moyenne sur 15 simulations aléatoires de N utilisateurs. Étant considéré comme "Temps de simulation", la récupération des stations et voitures, la génération des utilisateurs, la modélisation du graphe, la génération des trajets et la simulation de la charge du réseau. Le temps de calcul est le temps de simulation auquel est soustrait le temps de récupération des stations et voitures et la génération du graphe.

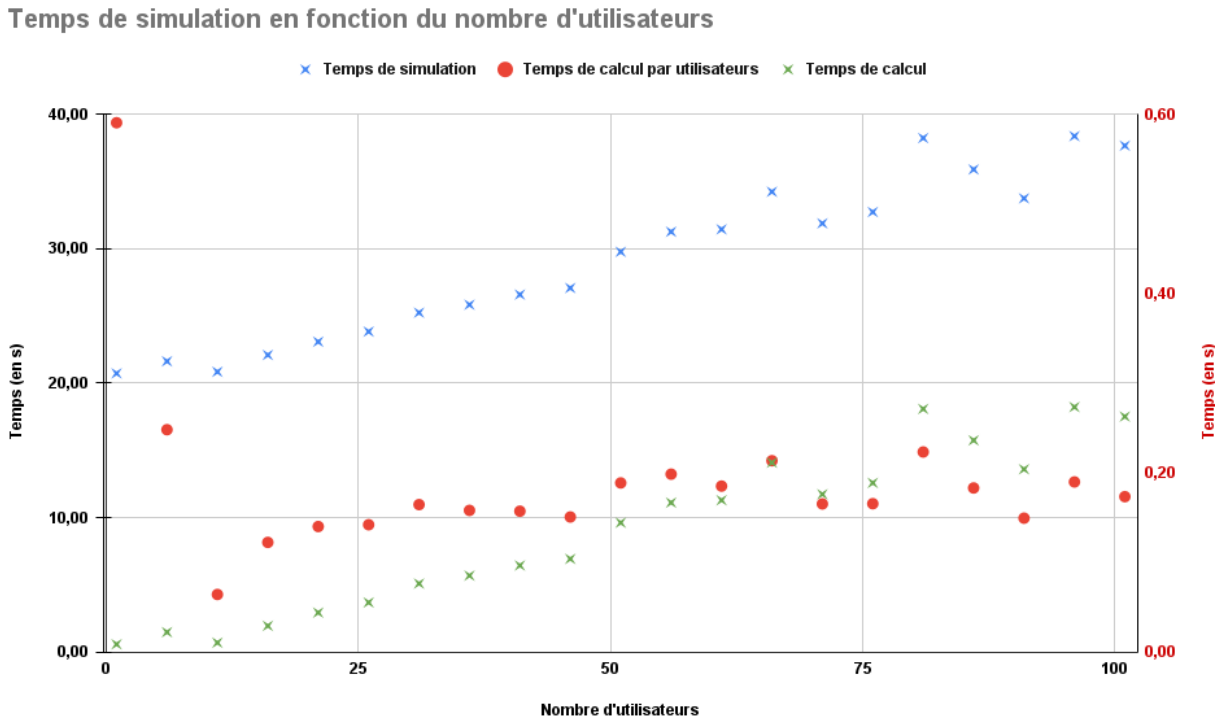


FIGURE 6 – Temps de simulation en fonction du nombre d'utilisateurs

Ici nous pouvons voir qu'avec un nombre d'utilisateurs assez faible (N < 100) le temps de simulation semble linéaire. Comme nous l'avions présumé dans la partie **Algorithme**, il y a un temps de simulation minimal dû aux différentes générations de structures. Par souci de clarté, ce temps a été retiré pour donner les points verts.

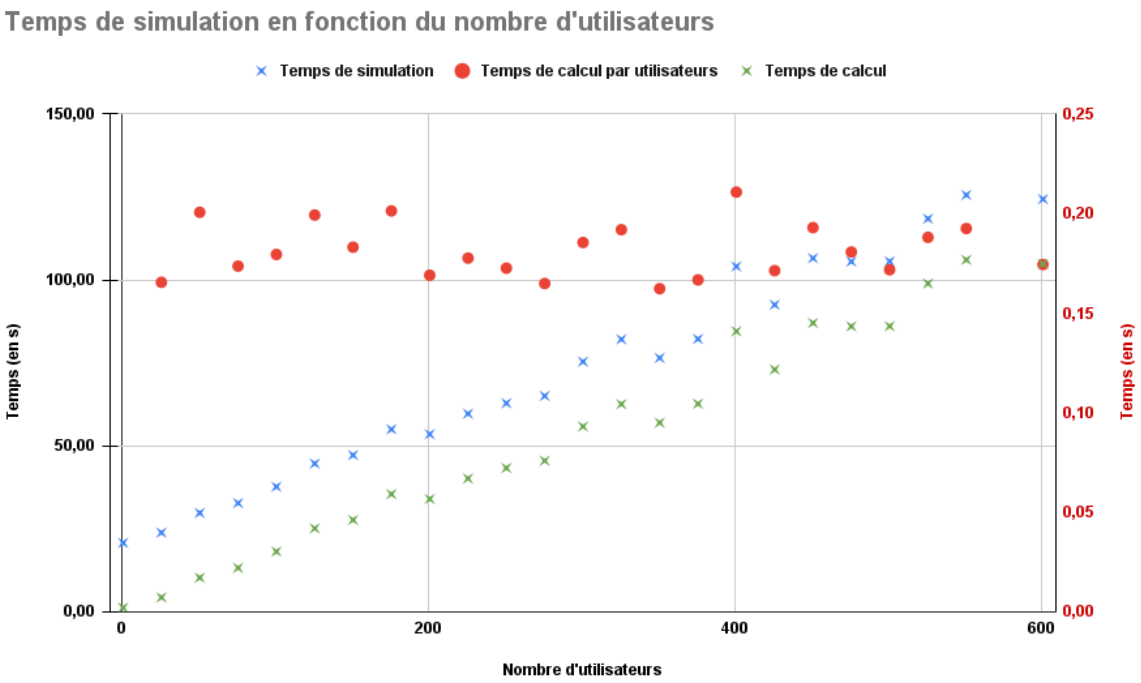


FIGURE 7 – Temps de simulation en fonction du nombre d'utilisateurs avec N > 100

Dans le graphique ci-dessus, nous avons testé le programme avec plus d'utilisateurs pour voir si ça avait un impact sur la forme de la courbe.

Nous pouvons donc faire une régression linéaire pour déterminer si notre courbe se rapproche d'une courbe en O(N).
Lorsque l'on fait une régression linéaire, on obtient une droite de la forme $y = ax + b$ avec $a = 0,183$ et $b = 19,2$. Et on peut aussi récupérer le coefficient de détermination linéaire de Pearson $R^2 = 0,985$. Par conséquent, chaque utilisateur prend environ 185ms à être simulé, la mise en place des données pour la simulation met environ 19, 2s et nous sommes très proche d'une droite.

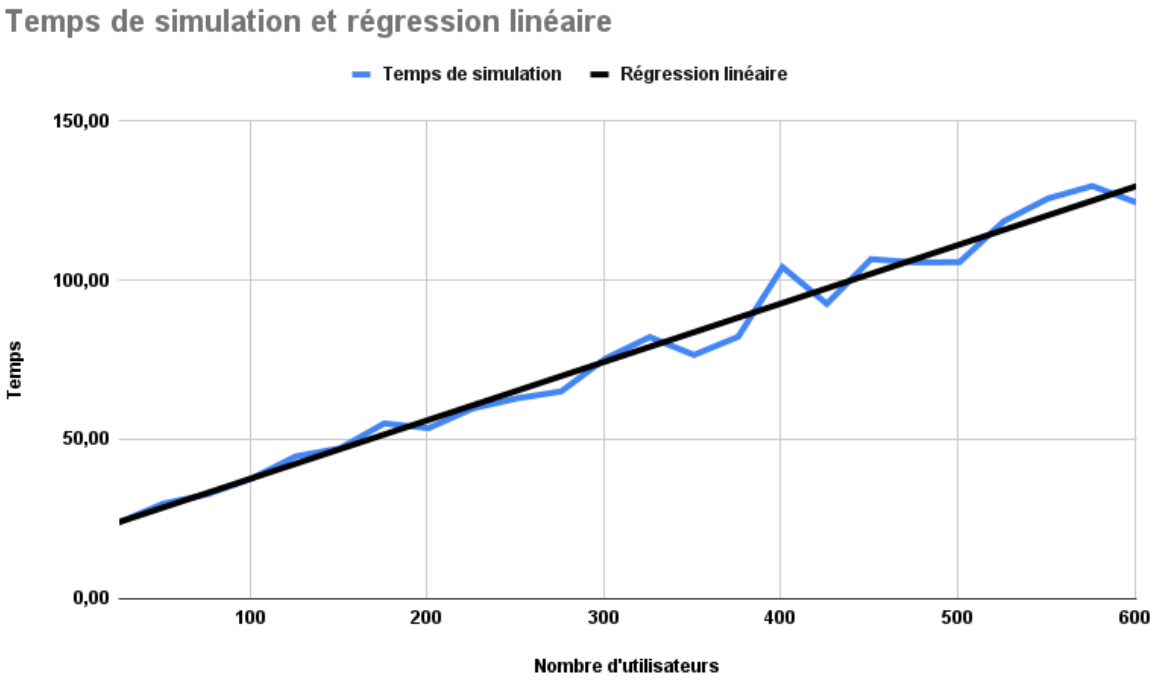


FIGURE 8 – Temps de simulation régression linéaire

9 Gestion de Projet

9.1 Matrice RACI

	Pierre-Yves	Arnaud	Titouan	Pierre
Cahiers des charges	I	RA	I	I
Gestion de projet	I	R	I	RA
Parsing	RA	I	I	I
Recherches Algos	C	A	R	R
Structure de données	R	RA	R	R
Implémentation Algos	R	RA	R	R
Interface	RA	I	I	I
Débug	I	R	R	RA
Simulation	I	R	RA	I
Sauvegarde simulation	I	I	RA	I
Visualisation simulation	RA	I	I	I
Guide Utilisateur	R	RA	R	R
Rapport et Diapo	R	RA	R	R

R : Responsable
A : Autorité
C : Consulté

I : Informé

On peut remarquer avec cette matrice RACI que nous avons été capables de bien se répartir le travail tout en coopérant sur presque tout.

9.2 Déroulé du projet

L'expérience du projet PPII1 nous a permis de nous mettre facilement et efficacement au travail. Ayant conservé le même groupe, nous étions conscients des points forts et des faiblesses de chacun, ce qui nous a permis de les mettre a profit. Notre groupe étant bien huilé, nous avons été capables de fixer un diagramme de Gantt (voir annexe) que nous avons respecté et qui nous a accompagné sur tout le projet. Nous avons de plus été capables de se projeter et de nous séparer le travail efficacement afin d'arriver au bout. Nous avons également appris à abandonner certaines idées qui, bien que bonnes, auraient été trop complexes ou ambitieuses. De telles idées auraient occasionné des délais qui nous auraient empêchés de terminer le projet.

10 Bilan global

Nous pouvons dire que le projet s'est globalement bien passé. Nous nous sommes mis au travail rapidement et les réunions hebdomadaires que nous avons faites systématiquement nous ont permis de progresser avec régularité et de terminer le projet. Chaque membre a fait le travail qui lui a été assigné et a été sérieux. Nous aurions pu cependant se répartir mieux le travail à faire (le lister par exemple) pour avancer plus rapidement et plus efficacement.

11 Bilans personnels

11.1 Pierre GUYOT

D'un point de vue personnel, je dirais que le projet m'a permis d'étoffer mes compétences de programmation à plusieurs. En particulier, de remarquer que la programmation en boîte noire, un peu comme peut le faire Java en POO, est plus intéressant en équipe. Cela permet de vraiment plus facilement se répartir les tâches, tout en ayant la possibilité de modifier des bases du code, sans avoir à coder de nouveau tout le reste en général !

Comme le C fut mon premier langage de programmation au collège, je pensais alors que ce projet ne serait pas si intéressant pour moi. Au final, j'ai pu m'améliorer dans la gestion de l'allocation de mémoire dynamique, et dans les structures de données.

D'un point de vue global, concernant la gestion d'équipe et de projet, faire des réunions de manière hebdomadaire fut au final très important. Même s'il n'y a pas forcément beaucoup de choses à se dire, car on discutait aussi en dehors des réunions, cela permettait de mettre tout le monde au courant en plus de cadencer le rythme de travail. La mise en place d'une réunion de brainstorming et d'un Gantt par la suite était vraiment intéressante. Même s'il peut être complexe de prévoir une gestion de tâches à faire dans le futur, cela permettait d'encadrer le projet dans le temps.

11.2 Pierre-Yves JACQUIER

Personnellement, je me vois un peu comme la voiture qui suit les cyclistes du Tour de France. Contrairement au premier PPII, j'ai pris la casquette du soutien technique, j'ai construit un parseur (nommé "monstre") d'une violence inégalée, ainsi que l'interface graphique. Bien que je n'ai pas énormément touché au code en C, mon travail y était quand même lié, car je devais fournir aux cyclistes (i.e. autres membres du groupe) tout ce dont ils avaient besoin, cela inclut devoir réécrire mon parseur deux fois car les données manipulées par le programme ont changé...

Sur le plan de la gestion de projet, la meilleure répartition des tâches a rendu ce projet bien plus agréable de mon côté, ma charge de travail ayant été approximativement divisée

par deux par rapport au premier projet.

Somme toute, les compétences que j'ai acquises au cours de ce projet n'étaient certes pas les mêmes que mes camarades, mais sont tout aussi importantes : devoir s'adapter à des besoins en changement constant et à une base de donnée truffée d'incohérences. J'ai aussi dû réfléchir à une solution pour permettre la communication entre le programme en C et le site web en HTML/Python (et ce dans les deux sens).

11.3 Arnaud KRAFFT

Le point le plus important de ce projet selon moi a été de voir à quel point une bonne organisation permet d'être efficace. Contrairement au premier PPII, nous avons cette fois ci réussi à travailler régulièrement, en faisant au moins une réunion par semaine pendant toute la durée du projet. Cela nous a permis d'éviter l'effet tunnel. Chaque membre a pu avoir du travail en permanent et nous avons pu nous répartir toutes les tâches à faire sans que personne se retrouve perdu, comme ça m'était arrivé lors du premier PPII.

Sur un point plus personnel, ce projet a été formateur en C et m'a permis de comprendre beaucoup de chose sur le langage. Ceci m'avait manqué dans le premier PPII car dû à une mauvaise répartition du travail de notre part, je m'étais retrouvé à faire principalement du SQL et de la gestion de projet.

Cet aspect m'a permis de comprendre réellement ce que c'est que de coder sur un projet en même temps que d'autres gens et de devoir utiliser leur travail pour faire fonctionner le sien, ainsi que toutes les modifications qui en découlent lorsque la fonction initiale se retrouve modifiée.
Cela souligne à quel point la communication entre membres est importante lors d'un travail en groupe.

11.4 Titouan LANGLAIS

Potagibus a été très formateur pour moi, la manipulation des pointeurs et le travail en groupe sur un projet C sont des choses qui m'étaient presque totalement inconnues, maintenant, j'y suis habitué et je les maîtrise assez pour pouvoir faire un projet. Hormis les connaissances que cela m'a apporté, j'ai beaucoup aimé ce projet, et pas juste car cela s'est bien passé, mais aussi car il y avait une grosse partie algorithmique qui m'a beaucoup plu.

Si l'on regarde le groupe, notre gestion de projet était bien meilleure que lors de Pota'Gist (le premier projet) et je suis fier de pouvoir dire que notre groupe a bien géré cette partie. Nous n'avons pas fait de "crunch" et nous avons tout fait dans les temps en suivant notre gantt. Nous avons aussi corrigé beaucoup des erreurs que nous faisions lors de Pota'Gist, cependant, je trouve que l'on reste un peu rigides dans notre façon de gérer les choses à faire. Concernant la répartition des choses à faire, j'aurais aimé que tout le monde ait une vraie partie en C à faire, mais ça nous a permis d'avancer sans souci grâce

à notre mécanicien de Formule 1.

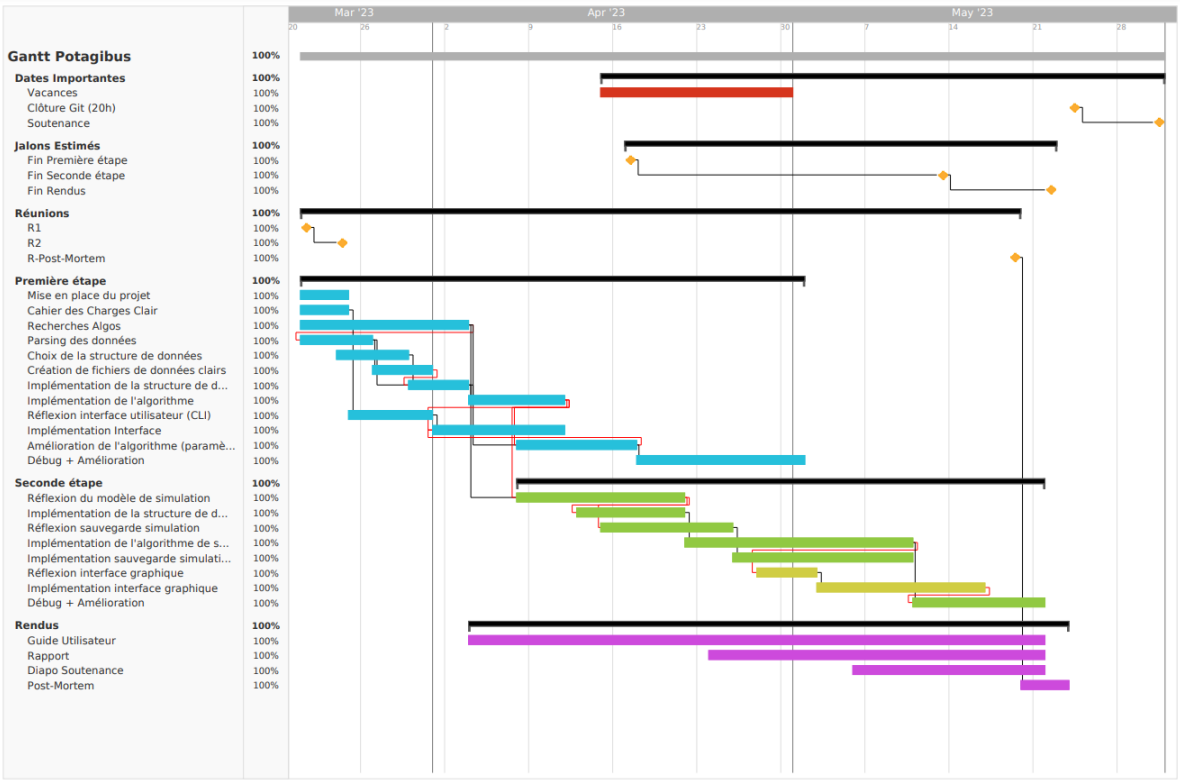


FIGURE 9 – Graphique Gantt Potagibus

12 Annexe

Pseudo-code du A* :

```
file_prochain = [] // De la plus petite a plus grande distance
chemin_depart = [id_depart] // Du d part a l'arrivee
vu = [id_depart] // Du plus petit au plus grand ID
lambda = 2 // lambda permet de choisir le style de chemin
    prefere pour une recherche plus rapide

pour i, voisin de id_depart :
    distance_total_appro_i = distance(id_depart, i) + lambda*
        distance(i, id_arrive)
    insert_croissant(file_prochain, [distance_total_appro_i, i,
        chemin_depart])


tant que file_prochain != [] ou point_plus_proche(file_prochain)
    == id_arrive :
```

```
12      [meilleur_distance, id_meilleur_point, meilleur_chemin] =
            retire_plus_petit(file_prochain)
13      si vu[id_meilleur_point] == Vraie :
14          continue
15      vu += [id_meilleur_point]
16      meilleur_chemin += [id_meilleur_point]
17
18      pour i, voisin de id_meilleur_point :
19          si vu[i] == Vraie :
20              continue
21          distance_total_appro_i = meilleur_distance + distance(
                id_meilleur_point, i) + lambda*distance(i, id_arrive)
22          insert_croissant(file_prochain, [distance_total_appro_i,
                i, meilleur_chemin])
23
24
25  si file_prochain != [] :
26      Pas de chemin
27
28  sinon :
29      retourne meilleur_chemin + [id_arrive]
```

13 Comptes rendus de réunion

Compte-rendu 1

Réunion de début de projet

21 Mars 2023

1 Participants

— Pierre GUYOT
— Pierre-Yves JACQUIER
— Arnaud KRAFFT
— Titouan LANGLAIS

2 Ordre du jour

1. Mise en place du projet
2. Distribution des rôles
3. Prise de conscience des attendues

3 Déroulement de la réunion

3.1 Mise en place du projet

Pour ce nouveau projet, il nous faut donc une gestion de projet avec un graphique Gant, différentes réunions et compte-rendus. De plus, le projet est uniquement en langage C.
Nous utiliserons Trello pour la ToDo list, et aussi pour le Gant du projet. De plus, un cahier des charges clair est nécessaire.

3.2 Distribution des rôles

Arnaud sera le chef de projet. Pierre sera le secrétaire. Pierre-Yves et Titouan seront ceux qui feront les recherches.

3.3 Prise de conscience des attendues

Notre problème peut se représenter par un graphe à parcourir. Les recharges étant des sommets. La distance entre chaque établissement étant le poids d'une arrête entre chaque sommet. On peut modéliser tout cela avec une matrice d'adjacence. De par le nombre important de points, on pense qu'il sera impossible d'appliquer l'Algorithme de Dijkstra.
De plus on fera différentes approximations pour la distance en se basant sur les coordonnées. La vitesse des voitures sera arrondi à 80 km/h constant.
Enfin, nous ferons différents exécutables en C pour différentes tâches, comme le parsing ou bien le formatage des données.

4 Prochaines étapes

Il faut pour la prochaine réunion :

1. Gantt + Trello - Pierre
2. Cahier des charges clair - Arnaud
3. Recherches approfondies sur les algos disponibles - Titoutan Pierre-Yves
4. Réfléchir au parsing et récuperer les données - Titoutan Pierre-Yves

5 Date et lieu de la prochaine réunion

Vendredi 24 Mars - Réunion à TN

Compte-rendu 2

Réunion de planification

24 Mars 2023

1 Participants

— Pierre GUYOT
— Pierre-Yves JACQUIER
— Arnaud KRAFFT
— Titouan LANGLAIS

2 Ordre du jour

1. Retour sur le Gantt et Trello
2. Présentation du cahier des charges et des algorithmes
3. Présentation des données et du parsing
4. Réflexion sur l'optimisation et la structure des données

3 Déroulement de la réunion

3.1 Retour sur le Gantt et Trello

Le site utilisé est TeamGantt qui est un site complet permettant d'éditer le Gantt ainsi que de l'exporter. Il est couplé avec Trello qui est un site permettant de faire une liste des tâches à faire. Les différents participants rajouteront des tâches à faire sur Trello, et Pierre s'occupera de les mettre dans le Gantt.
Il serait intéressant de faire une matrice RACI pour mieux donner les tâches aux différents membres.

3.2 Présentation du cahier des charges et des algorithmes

Arnaud présente en premier le cahier des charges disponible en annexe.
Titouan présente ensuite les différents algorithmes, disponible aussi en annexe. On retient des 3 algo : A*, Anytime A* et Jump Point Search.
On utilisera pour l'instant A* qui est le plus simple à comprendre. On essayera de l'améliorer si possible en Anytime A* permettant de trouver un chemin plus rapidement, et de s'améliorer petit à petit. Jump Point Search n'a pas assez de documentation, Pierre se propose de se documenter à son sujet.

3.3 Présentation des données et du parsing

Pierre-Yves présente les données intéressantes dans la base de données dont la différence entre les coordonnées XY et la latitude/longitude. Il pointe la possibilité de regrouper différentes bornes situés à la même adresse tout en retirant les données jugées inutiles. Pierre-Yves nettoiera donc la base de données.

3.4 Réflexion sur l'optimisation et la structure des données

En fonction du départ et de l'arrivée, il est intéressant de ne pas charger tous les points de recharge en mémoire. Il est décider de prendre une zone carré ou elliptique autour du départ et de l'arrivée dépendant d'un autre paramètre de rayon.
De plus, en fonction du modèle de voitures renseigné, on ne modélisera pas certains liens entre les stations les plus lointaines.
Enfin la solution retenu est d'utiliser une matrice d'adjacence en stockant les données et l'idée de chaque station dans une autre liste.

4 Prochaines étapes

Il faut pour la prochaine réunion :

1. Exclusion de points inutiles (plusieurs méthodes) - Arnaud
2. Implémentation de la structure de données en matrice - Titouan

3. Nettoyage BDD - Pierre-Yves
4. RACI, Distance en XY et Jump Point Search - Pierre

5 Date et lieu de la prochaine réunion

Vendredi 31 Mars - Réunion à TN

Compte-rendu 3

Réunion de projet

31 Mars 2023

1 Participants

— Pierre GUYOT
— Pierre-Yves JACQUIER
— Arnaud KRAFFT
— Titouan LANGLAIS

2 Ordre du jour

1. Retour sur les tâches de la dernière fois
2. Réflexion sur les problèmes algorithmiques
3. Réflexion sur la structure de donnée utilisée
4. Free Talk

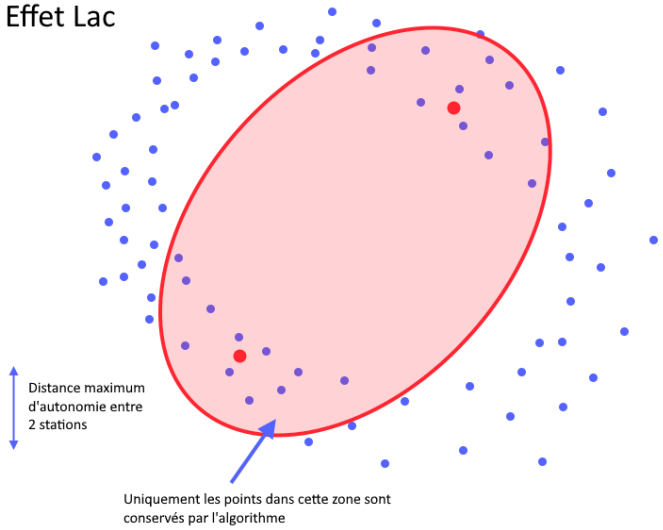
3 Déroulement de la réunion

3.1 Retour sur les tâches de la dernière fois

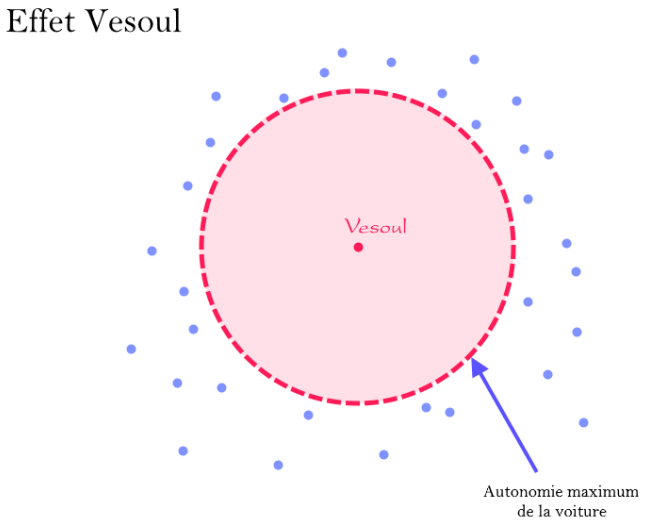
Pour les coordonnées X-Y, on utilisera la formule de Haversine permettant d'avoir une bonne approximation de la distance entre 2 points.
Jump Point Search n'est pas adapté pour notre problème, il est fait pour une carte en case avant tout. C'est une simple amélioration de A* de manière très précise.
Avec le nettoyage de la BDD, on remarque que cette dernière est très mal faite et comporte pas mal d'erreur pour permettre un bon parsing. On choisit de regrouper un certains nombres de points assez proche en une seule station.
Le système d'exclusion de points fonctionnent bien mais est soumis à divers problèmes expliqués plus tard.
La structure de données pour la matrice d'adjacence est donc bien un tableau de tableau.

3.2 Réflexion sur les problèmes algorithmiques

Avec le principe d'exclusion en forme d'ellipse ou bien de carrés, on peut arriver à l'effet lac. On exclut trop de point empêchant de trouver un seul chemin :



Pour éviter ce soucis, si aucun chemin n'est trouvé, on agrandit le rayon de l'ellipse et on recommence. Cela permet d'éviter l'effet lac.
De plus il y a aussi l'effet Vesoul, une ville au milieu de rien. Prendre en compte qu'il est possible qu'il n'existe aucun chemin, car l'autonomie de la voiture est trop petite jusqu'à la prochaine station. Il faut simplement prévoir cette possibilité dans le code.



3.3 Réflexion sur la structure de donnée utilisée

Pour le calcul de distance, il est intéressant d'utiliser le type float afin de garder une certaine précision. La matrice d'ajacence sera alors en float**.
En effet, on préfère donc travailler avec des tableaux car leur accès est en O(1) contrairement aux listes chaînées. Cependant, on ne peut pas tout faire en tableaux ne connaissant pas la taille des données que l'on va utiliser. On utilise donc en préprocessing (dans les fonctions) des listes chaînées que l'on transforme puis retourne en tableau.

3.4 Free Talk

Faire la spécification des fonctions dans le doc et dans la docu.
La fonction qui génère la matrice est une fonction récursive qui augmente la marge si elles ne trouvent pas de chemins (Effet lac). Si la marge est trop grosse, la fonction s'arrête.

4 Prochaines étapes

Il faut pour la prochaine réunion :

1. Préprocessiong liste chainée en renvoyant un tableau - Arnaud
2. Documentation et matrice adjacence - Titouan
3. Nouveau Filtrage de la base de données et formule de Haversine - Pierre-Yves
4. Écriture de l'algo A* et possiblement création en C - Pierre

5 Date et lieu de la prochaine réunion

Mercredi 5 Avril - Réunion à TN

Compte-rendu 4

Réunion de projet

5 avril 2023

1 Participants

— Pierre GUYOT
— Pierre-Yves JACQUIER
— Titouan LANGLAIS

2 Ordre du jour

1. Retour sur les tâches de la dernière fois
2. Spécification des fonctions
3. Problèmes puissance nominale
4. Free Talk

3 Déroulement de la réunion

3.1 Retour sur les tâches de la dernière fois

La BDD de l'état est vraiment mauvaise. Les données n'ont aucune unicité entre elles et rendent difficile sa lecture. Les informations sont séparées n'importe comment. Le fichier doit être filtré caractère par caractère. Le fichier filtré sera sous la forme : nom, longitude, latitude, nombre de bornes. Elle est modulable et on peut rajouter des éléments, ou changer la distance minimale entre les stations. Il faudrait pour la partie 2 rajouter le comptage des stations retirés, en les incluant dans les autres bornes.
Explication et présentation de l'algorithme A* en pseudo code, trouvable en dans les document Latex. Pour plus d'explication, car ce n'est pas clair, demander à Pierre.

3.2 Spécification des fonctions

Titouan a fait docs.txt qui explique les fonctions, des petites spécifications rapides. La matrice d'adjacence est strictement triangulaire supérieur, sur les colonnes, il n'y a pas la première colonne et il n'y a pas le dernière ligne. Il faut donc faire attention lorsque l'on prend les indices dans la matrice, puisqu'ils sont décalés. De plus, aussi prendre en compte des cas où l'on demanderai la case x,y avec x=y.

3.3 Problèmes puissance nominale

"d'enrichir la fonction précédente avec des paramètres supplémentaires, par exemple, ne jamais laisser le véhicule en dessous d'une charge de 30%, ou si votre modèle le permet, borner les temps de recharge (l'usager ne veut jamais rester plus de 20 minutes en charge)" Ceci est le deuxième point du projet qui suscite débat. Il est assez facile d'implémenter la possibilité que le véhicule ne se trouve jamais en dessous de 30% de batterie en changeant son autonomie. Par contre au niveau du maximum de 20 minutes de recharge, cela pose problème si on considère le paramètre de puissance nominale fourni par les stations. Les voitures ne se rechargent pas à la même vitesse et certaines stations sont alors plus intéressante que d'autres.
Ce sujet est encore en discussion pour savoir si on prend en compte la puissance nominale ou non. Si elle est prise en compte, il sera aussi important de collecter la puissance nominale maximale de chaque voiture, ou puissance max de rechargement.

3.4 Free Talk

Comment importe t-on la BDD en C ? Titouan propose de faire le parsing de données dans la BDD filtré de Pierre-Yves.
Pour l'algorithme A*, il serait intéressant de proposer de prendre le moins de stations possibles au cours du trajet.

4 Prochaines étapes

Il faut pour la prochaine réunion :

1. Continuer la fonction du préprocessing des points- Arnaud
2. Aide sur la génération de matrice et parsing en C - Titouan
3. Rajouter une 3ième fonction pour le nombre de bornes par station - Pierre-Yves
4. Implémenter et améliorer l'algo A* - Pierre

5 Date et lieu de la prochaine réunion

Mercredi 12 Avril - Réunion à TN

Compte-rendu 5

Réunion de projet

12 avril 2023

1 Participants

— Pierre GUYOT
— Pierre-Yves JACQUIER
— Arnaud KRAFFT
— Titouan LANGLAIS

2 Ordre du jour

1. Retour sur les tâches de la dernière fois
2. Entrée des paramètres
3. Amélioration du Parsing de station

3 Déroulement de la réunion

3.1 Retour sur les tâches de la dernière fois

Les tâches précédentes ont bien été faite. Nous avons bien une fonction de préprocessing des points, nous pouvons faire du parsing en C et récupérer la matrice. De plus, le parsing a été amélioré. L'algo A* a bien été implémenter et débugueur en partie. Mais il reste encore beaucoup de bug dû à l'implémentation de pas mal de fonctionnalité.

3.2 Entrée des paramètres

On a choisit d'utiliser les entrées par CLI en utilisant les coordonnées GPS et l'ID de la voiture. Il faudra donc faire un parsing des autonomies.

3.3 Amélioration du Parsing de station

Problème, on a des stations avec plus de 1600 points de recharge. Bug à corriger.

4 Prochaines étapes

Il faut pour la prochaine réunion :

1. Debug - Arnaud
2. Debug - Titouan
3. Parsing des voitures - Pierre-Yves
4. Debug - Pierre

5 Date et lieu de la prochaine réunion

19 Avril 14h - Réunion sur Discord

Compte-rendu 6

Réunion de projet

19 avril 2023

1 Participants

— Pierre GUYOT
— Pierre-Yves JACQUIER
— Arnaud KRAFFT
— Titouan LANGLAIS

2 Ordre du jour

1. Retour sur les tâches de la dernière fois
2. Free Talk

3 Déroulement de la réunion

3.1 Retour sur les tâches de la dernière fois

La gestion des bugs se passent bien. Toujours problème au niveau de la BDD qui est mal faites (celle donné par l'état).

3.2 Free Talk

On avance vers la fin de la première étape. Il faudrait prendre en compte la vitesse de rechargement pour l'inclure dans l'algo de A*. On décide de commencer la simulation. Les usagers seront des triplets. Possiblement faire un deuxième A* plus rapide si on veut faire une vraie simulation à temps réel (les usagers changent de déplacements en fonction de la circulation)

4 Prochaines étapes

Il faut pour la prochaine réunion :

1. Création des usagers aléatoires - Arnaud
2. Mise en place de la simulation - Titouan
3. Tableur - Pierre-Yves
4. Finir la partie 1 - Pierre

5 Date et lieu de la prochaine réunion

26 Avril - Réunion à TN

Compte-rendu 7

Réunion de projet

25 avril 2023

1 Participants

— Pierre GUYOT
— Pierre-Yves JACQUIER
— Arnaud KRAFFT
— Titouan LANGLAIS

2 Ordre du jour

1. Retour sur les tâches de la dernière fois
2. Free Talk

3 Déroulement de la réunion

3.1 Retour sur les tâches de la dernière fois

La structure de projet et la remise en place des fichiers étaient nécessaire et a été fait. Il manque encore quelques fonctions comme destroy pour la structure utilisateur. Il faudra faire un parcours entier de la liste utilisateur en une fois.
Explication de la simulation par Titouan, système par tick. Pierre doit rajouter la fonciton pop au chemin. La simulation renvoie le nombre de personnes à chaque moment en train de se charger dans les stations (tableau de tableau, matrice quoi). Transformer l'id relatif chemin en id station pour le chemin. Rajouter des nouveaux éléments dans le chemin comme la capacité, le temps de charge et toutes les informations liés à la voiture.
Le parsing est terminé. Encore du travail sur la partie 1.

3.2 Free Talk

Il faudrait une fonction qui donne son trajet, son temps de trajet et sa batterie pour chaque maillon de la liste chainée d'utilisateurs.

4 Prochaines étapes

Il faut pour la prochaine réunion :

1. Fonction Trajet Détaillé dans la doc- Arnaud
2. Simulation - Titouan
3. Visualisation Graphique - Pierre-Yves
4. Finir Partie 1 - Pierre

5 Date et lieu de la prochaine réunion

3 Mai - Réunion à TN

Compte-rendu 8

Réunion de projet

3 mai 2023

1 Participants

— Pierre GUYOT
— Pierre-Yves JACQUIER
— Arnaud KRAFFT
— Titouan LANGLAIS

2 Ordre du jour

1. Retour sur les tâches de la dernière fois
2. Free Talk

3 Déroulement de la réunion

3.1 Retour sur les tâches de la dernière fois

PYJ n'a pas pu faire le travail car les sorties n'ont pas encore été implémenté. Il a donc pensé à utiliser Open Street Map pour l'afficher. Arnaud a fait ce qu'il fallait concernant la simulation. On garde le chemin tel quel au final, on ne le transforme pas comme une file. Il faut retirer les données en double dans la simulation (distance_pro, et taux_de_charge et le next) RAPPEL : on travaille en heure et non en minute

3.2 Free Talk

Pierre n'a pas besoin d'inverser le chemin. Ajouter dans info un entier ID_courrant qui sera égal à la taille du chemin-1 Ajouter un entier Nb_ticks_attente égale (distance divisé par la vitesse)*nb de tiks par seconde. Rajouter un macros vitesse et tics par seconde

4 Prochaines étapes

Il faut pour la prochaine réunion :

1. Modifier la fonction de simulation - Arnaud
2. Calcul Tick - Titouan
3. Carte intéractive - Pierre-Yves
4. Prendre en compte la puissance nominale - Pierre

5 Date et lieu de la prochaine réunion

10 Mai - Réunion à TN

Compte-rendu 9

Réunion de projet

10 mai 2023

1 Participants

— Pierre GUYOT
— Pierre-Yves JACQUIER
— Arnaud KRAFFT
— Titouan LANGLAIS

2 Ordre du jour

1. Retour sur les tâches de la dernière fois

3 Déroulement de la réunion

3.1 Retour sur les tâches de la dernière fois

Titouan : Site web, choisir sa voiture et faire la simulation Il faut d'abord finir Gantt on l'a bien suivi Arnaud a faiy don boulot et régler les erreur de compil, il y a juste à finir le a_star titouan calcul tick fait pas tester, mais globalement bien passé Pas beau mais commenter Il y a une valeur spécial pour le nombre de tick avant d'arriver, à 1 on est arrvier PYJ a improviser quelques choses pour fire le sit web, n'ayant pas les données de nos sorties Il l'affiche avec falsk et ça fonctionne Pierre est encore sur le A_star, et doit faire sa doc une bonne fois pour toute un slider pour les ticks ou des pages différentes

4 Prochaines étapes

Il faut pour la prochaine réunion :

1. Fonction Destroy + Readme + Début du rapport + Complétez Doc - Arnaud
2. Simulation - Titouan
3. Site Web - Pierre-Yves
4. Doc + A_Star - Pierre

5 Date et lieu de la prochaine réunion

16 Mai - Réunion à TN

Compte-rendu 10

Réunion de projet

16 mai 2023

1 Participants

— Pierre GUYOT
— Pierre-Yves JACQUIER
— Arnaud KRAFFT
— Titouan LANGLAIS

2 Ordre du jour

1. Retour sur les tâches de la dernière fois
2. Précision tâche

3 Déroulement de la réunion

3.1 Retour sur les tâches de la dernière fois

Retirer les mains inutiles
2 mains, pour la partie 1 et pour la partie 2
macros temps_recharge_max=1/3
Explication du a_star
macros minimum_percent_battery = 1
macros capacite_depart = -1
Rapport en création avec l'état de l'art etc
PYJ doit parler du parsing de données dans le rapport
Pierre fait chemin_tab et chemin_tab_struct
Mettre le gantt
Pierre => explique les algos fonction partie 1
Pierre => explication des structures
Faire explication partie 2
Readme : ajouter sur le readme : ajouter clone, make
Faire readme : Expliquer comment cloner, make et les arguments pour exécuter la commande
compléter la fonction utilisateur
rediger une fct qui passe de modèle à ID
Titouan a commencer la simulation
Titouan donne les fichiers comme demandé
Squelette de site web fait
Mais bug UTF 8
Le site permet de calculer un itinéraire
Pierre => fait un autre p1 pour le site

3.2 Précision tâche

Readme $Expliquer comment cloner, make et les arguments pour exécuter les commandes$
$Expliquer comment lancer et utiliser le site Web dans le readme$
$Site Web : 2 possibilits, soit on peut lancer la simulation et l'afficher. On a un slider pour afficher les diffrents ticks. Soit on peut a$
$Dans les deux cas, python appelle un programme C qui renvoie dans un fichier txt.$

4 Prochaines étapes

Il faut pour la prochaine réunion :

1. Faire sa partie sur le rapport - Tout le monde
2. Readme, Debug, Finir simulation - Arnaud

3. Finir Simulation - Titouan
4. Finir Site Web - Pierre-Yves
5. Fonction ID nom voiture, nouveau exe partie 1, gant et Trello - Pierre

5 Date et lieu de la prochaine réunion

20 Mai - Discord

Compte-rendu 11

Réunion de projet

20 mai 2023

1 Participants

— Pierre GUYOT
— Pierre-Yves JACQUIER
— Arnaud KRAFFT
— Titouan LANGLAIS

2 Ordre du jour

1. Retour sur les tâches de la dernière fois

3 Déroulement de la réunion

3.1 Retour sur les tâches de la dernière fois

décalage inutile
Pierre => GANTT TRELLO
Readme => Comment lancer le site web et l'utiliser
Rapport en complétion
Site Web : seulement sur Linux
Site Web : ./runsite.sh
127.0.0.1 :5000
Rajouter la liste des voitures
Post mortem
Rapport => Finir parsing, plus les algos et les fonctions
La fonction random n'est pas random
Calcul complexité A_star
Titouan => Fonction random C
Pseudo Code pour le A_Star
Pierre Yves => Visualisation Rapport
Test => Titouan 10k
Bilan personnel => pour tout le monde

4 Prochaines étapes

Il faut pour la prochaine réunion :

1. Post Mortem, finir rapport - Arnaud
2. Post Mortem, rapport algos/fonctions/sd, fonction random, rapport test - Titouan
3. Rajouter liste voiture site web, Post Mortem, rapport Parsing, visualisation rapport - Pierre-Yves
4. Gantt Trello Readme, Post Mortem, rapport algos/fonctions/sd, calcul complexité A* pseudo code A* - Pierre

5 Date et lieu de la prochaine réunion

20h 23 mai - Réunion Discord

Compte-rendu 12

Réunion de projet

23 mai 2023

1 Participants

— Pierre GUYOT
— Pierre-Yves JACQUIER
— Arnaud KRAFFT
— Titouan LANGLAIS

2 Ordre du jour

1. Post Mortem

3 Déroulement de la réunion

3.1 Post Mortem

Review du rapport, RAS
Prochaine réunion, faire le diapo

4 Prochaines étapes

Il faut pour la prochaine réunion :

5 Date et lieu de la prochaine réunion

25 Mai 16h - Réunion à TN