Rapport Projet IN104

Plus court chemin dans un itinéraire GPS

Réalisé par :

BELKAHLA Malek

BEN AMIRA Aziz

BOUAZIZ Elyes

Encadrante :

Mme ALOUANE Sonia

1A Techniques Avancées

Année universitaire : 2023/2024

Remerciment

Nous souhaitons exprimer nos sincères remerciements à toutes les personnes qui ont contribué au succès de notre projet et nous ont apporté leur aide et leurs conseils pratiques pour la réalisation de ce projet.

Nous tenons tout d'abord à remercier chaleureusement notre enseignante à l'Ecole Nationale d'Ingénieurs de Tunis, Mme. Sonia ALOUANE, pour son soutien et son expertise. Ses conseils et sa disponibilité ont grandement contribué à l'amélioration de notre projet et à notre choix judicieux de la direction à suivre.

Nous tenons également à remercier vivement Mme. CHAKER Hedia, responsable de la filière Techniques Avancées à l'ENIT, pour son soutien moral et ses encouragements, qui ont été une source de motivation essentielle pour nous tout au long du projet.

Enfin, nous tenons à remercier nos amis et camarades de classe qui nous ont soutenus et conseillés tout au long du projet.

Table des matières

Introduction

Dans le monde actuel en constante évolution, la navigation efficace est devenue un élément crucial de la vie quotidienne. Que ce soit pour se rendre au travail, à un rendez-vous important ou pour explorer de nouveaux endroits, disposer d'un système de navigation fiable est indispensable. C'est dans ce contexte que notre projet prend tout son sens : développer un programme capable de déterminer le plus court itinéraire entre deux points donnés sur une carte GPS.

Notre projet vise à répondre à un besoin universel et croissant d'optimisation des déplacements. En s'appuyant sur des algorithmes sophistiqués , notre programme permettra aux utilisateurs de gagner du temps précieux en empruntant les itinéraires les plus efficients.

La présente introduction pose les bases de notre projet, en mettant en lumière ses objectifs, sa portée et son importance. Nous décrirons ensuite les différentes composantes du programme, en expliquant les algorithmes sous-jacents et les technologies employées. Nous aborderons également les défis rencontrés et les solutions mises en œuvre pour les surmonter. Enfin, nous conclurons en présentant les perspectives d'avenir et les potentielles applications de notre programme.

En résumé, ce projet nous a permis d'explorer les avantages des algorithmes de recherche dans le domaine de la navigation GPS et de développer une solution pratique et innovante pour aider les utilisateurs à trouver rapidement le chemin optimal entre deux points sur une carte.

Chapitre 1

Contexte

De nos jours, dans notre société où la commodité est accessible d'un simple clic, le métier de livreur revêt une importance capitale. Il est impératif de prioriser l'optimisation des trajets de livraison et la rapidité pour répondre à ce besoin croissant.

Ce projet vise à fournir une solution qui permettra aux livreurs de naviguer de manière efficace et rapide pour satisfaire cette demande.

Ce rapport présente la conception d'un projet informatique visant à calculer le plus court chemin entre plusieurs points sur une carte, en utilisant l'algorithme de Dijkstra et le langage C. Le projet s'appuie sur la librairie osm-gps-map pour la représentation de la carte et utilise des structures de données de type graphes pour modéliser le réseau routier.

1.1 Analyse des besoins et étude de faisabilité

1.1.1 Analyse des besoins

Dans les premières phases de ce processus, il est essentiel de comprendre en profondeur les exigences des livreurs afin de développer une solution sur mesure.

Dans le cadre de notre application, les besoins sont clairs : les livreurs recherchent un moyen efficace pour planifier le trajet optimal entre différentes destinations. De surcroît, il est crucial

de prendre en compte des critères tels que la rapidité de calcul des itinéraires et l'exactitude des résultats.

1.1.2 Étude de faisabilité

Pour que notre projet d'application fonctionne, il faut examiner plusieurs points techniques, logistiques . Tout d'abord, il est crucial de vérifier si nous avons accès aux données géographiques nécessaires pour développer l'algorithme qui trouve le chemin le plus court. Il est également essentiel de prendre en compte les délais pour s'assurer que le projet peut être terminé dans un temps raisonnable.

En résumé, il est vital de bien comprendre les besoins et les possibilités du projet pour s'assurer qu'il réussit. Cela nous aidera à créer une solution qui satisfait les utilisateurs tout en étant réalisable avec nos ressources et nos contraintes.

1.2 Spécification

Définir avec précision les entrées, les sorties et le traitement des données de notre application est une étape essentielle de la spécification de notre projet. L'objectif est de trouver un chemin optimisé pour naviguer entre plusieurs déstinations.

1.2.1 Entrée

L'entrée du programme sera constituée du point de départ, du point d'arrivée et éventuellement de quelques endroits intermédiaires spécifiés par l'utilisateur.

1.2.2 Sortie

La sortie du programme consistera en un itinéraire optimal tracé sur la carte, indiquant le chemin à suivre pour naviguer de manière efficace entre tous les endroits spécifiés.

1.3 Choix technologiques

1. **Langage de programmation** : C : choix justifié par sa performance, sa portabilité et sa large adoption dans le domaine du développement embarqué.
2. **Librairie de carte** : osm-gps-map : permet de convertir des données OpenStreetMap en un format adapté à la navigation GPS. Elle offre des fonctionnalités de manipulation de graphes et de calcul de distances.
3. **Algorithme de recherche de chemin** : Dijkstra : algorithme reconnu pour son efficacité dans la recherche du plus court chemin dans un graphe pondéré.

1.4 Architecture logicielle

Le projet se compose des modules suivants :

1. **Module de lecture de carte** : Charge les données de la carte depuis un fichier au format .osm et les convertit en une représentation interne adaptée à l'algorithme de Dijkstra.
2. **Module de calcul de chemin :** Implémente l'algorithme de Dijkstra pour trouver le plus court chemin entre deux points spécifiés sur la carte.
3. **Module d'affichage :** Affiche le résultat de la recherche sur une interface graphique ou une console.

1.5 Fonctionnalités du projet

1. **Saisie des points de départ et d'arrivée** : L'utilisateur peut saisir les coordonnées des points de départ et d'arrivée manuellement ou les sélectionner sur une carte interactive.
2. **Calcul du plus court chemin** : Le projet calcule le chemin le plus court entre les points spécifiés en utilisant l'algorithme de Dijkstra. Affichage du résultat : Le projet affiche le chemin le plus court sur une carte, en mettant en évidence son tracé et sa distance totale.

3. **Options supplémentaires :** Le projet peut inclure des options supplémentaires, telles que la recherche de chemins optimal entre plusieurs points choisis par l'utilisateur (Exemple : Livreur)

1.5.1 Conclusion

En conclusion de ce chapitre, nous avons réalisé une analyse approfondie des besoins et des contraintes pour le développement de notre projet.

En comprenant les exigences des utilisateurs, notamment les livreurs, nous avons pu cerner les fonctionnalités essentielles à intégrer dans notre solution. De plus, l'étude de faisabilité nous a permis d'évaluer les aspects techniques et logistiques pour garantir la viabilité de notre projet.

À travers la spécification détaillée des entrées, des sorties et du traitement des données, ainsi que la sélection judicieuse des technologies, nous avons jeté les bases nécessaires pour concevoir une solution efficace et adaptée aux besoins spécifiques du domaine de la livraison.

Chapitre 2

Conception

2.1 Structures utilisées

Dans cette section on va discuter les différentes strcutures utilisées dans le programme :

```
typedef struct Point {
    double lat;
    double lon;
} Point;
```

Point : Représente un point géographique avec deux attributs, `lat` et `lon`, représentant respectivement la latitude et la longitude du point. Utilisé pour stocker des coordonnées géographiques dans le système de coordonnées GPS.

```
typedef struct Edge {
    int dest;
    double weight;
    struct Edge* next;
} Edge;
```

Edge : Représente une arête dans un graphe, avec trois attributs principaux. `dest` est l'identifiant du sommet de destination de l'arête. `weight` représente le poids de l'arête, qui est utilisé pour représenter la distance. `next` est un pointeur vers la prochaine arête dans la liste d'adjacence, permettant de représenter un graphe sous forme de liste d'adjacence.

```
typedef struct Graph {
    int numVertices;
    Point* vertices;
    Edge** adjLists;
} Graph;
```

Graph : Représente un graphe avec quatre attributs principaux. `numVertices` est le nombre de sommets dans le graphe. `vertices` est un tableau de pointeurs vers des structures `Point`, représentant les sommets du graphe. `adjLists` est un tableau de pointeurs vers des listes d'arêtes, permettant de représenter les listes d'adjacence pour chaque sommet du graphe.

```
typedef struct {
  Graph * graph;
  int * list;
  int void_pos;
  int* path;
}
CallbackData;
```

CallbackData : Utilisée pour stocker des données associées à un callback dans le contexte de l'utilisation de la bibliothèque `osm-gps-map`. Contient quatre attributs principaux. `graph`

est un pointeur vers une structure `Graph`, représentant le graphe sur lequel le callback est effectué. `list` est un tableau d'entiers, probablement utilisé pour stocker des indices ou des identifiants de sommets. `void_pos` est un entier qui pourrait représenter une position ou un index spécifique dans le contexte du callback. `path` est un pointeur vers un tableau d'entiers, probablement utilisé pour stocker un chemin ou une séquence de sommets dans le graphe.

```
1 Fichier CSV pour lire les donnees routiere
```

2.2 Librairies utilisées

1. **stdio.h** : Fournit des fonctions pour l'entrée et la sortie standard, permettant aux programmes de lire des données depuis l'entrée standard (clavier) et d'écrire des données vers la sortie standard (écran). Les fonctions incluent ‘printf()‘ pour l'impression formatée, ‘scanf()‘ pour la lecture formatée, ‘fopen()‘, ‘fclose()‘, ‘fread()‘, ‘fwrite()‘ pour la manipulation de fichiers, et bien d'autres.
2. **stdlib.h** : Offre une variété de fonctions générales, y compris la gestion de la mémoire (‘malloc()‘, ‘calloc()‘, ‘realloc()‘, ‘free()‘), la génération de nombres aléatoires (‘rand()‘, ‘srand()‘), la conversion de types (‘atoi()‘, ‘atol()‘, ‘atof()‘), et la terminaison du programme (‘exit()‘, ‘abort()‘).
3. **math.h** : Contient des fonctions mathématiques essentielles, telles que les fonctions trigonométriques (‘sin()‘, ‘cos()‘, ‘tan()‘), les fonctions exponentielles (‘exp()‘, ‘log()‘), les fonctions arithmétiques (‘sqrt()‘, ‘pow()‘, ‘fabs()‘), et les fonctions de comparaison (‘fmin()‘, ‘fmax()‘).
4. **glib.h** : Fournit des fonctions de base pour le développement d'applications, y compris la gestion de chaînes de caractères (‘$g\_strdup()$, $g\_strconcat()$‘), la gestion de listes (‘$g\_list\_append()$, $g\_list\_free()$‘), la gestion de tableaux (‘$g\_array\_new()$, $g\_array\_free()$‘), la gestion de hashmaps (‘$g\_hash\_table\_new()$, $g\_hash\_table\_destroy()$‘), et bien d'autres.

5. **gtk/gtk.h** : Contient des widgets et des outils pour créer des interfaces utilisateur graphiques. Inclut des widgets pour les boutons, les champs de texte, les listes, les arborescences, et bien d'autres. Fournit également des outils pour la gestion des événements de l'interface utilisateur, la gestion des fenêtres, et la personnalisation de l'apparence des widgets.
6. **gdk/gdkkeysyms.h** : Définit les symboles des touches du clavier, permettant aux applications de gérer les entrées clavier de manière plus intuitive. Les symboles incluent les touches alphanumériques, les touches de fonction, les touches de direction, et bien d'autres.
7. **osm-gps-map.h** : Fournit des widgets et des fonctions pour intégrer des cartes interactives dans les applications graphiques, permettant aux utilisateurs de visualiser des données cartographiques et d'interagir avec des éléments cartographiques. La bibliothèque supporte le chargement de données cartographiques à partir de sources externes, comme OpenStreetMap, et permet de personnaliser l'apparence des cartes et des éléments cartographiques. Elle offre également des fonctionnalités pour gérer des événements de l'interface utilisateur liés à la carte, comme les clics et les mouvements de la souris, permettant aux utilisateurs d'interagir avec la carte de manière intuitive.
8. **assert.h** : Fournit la macro 'assert()' qui permet de tester des conditions au moment de l'exécution. Si la condition testée est fausse, le programme se termine avec un message d'erreur. C'est utile pour le débogage et pour s'assurer que certaines conditions sont remplies avant de continuer l'exécution du programme.

2.3 Fonctions utilisées

— **Graph * createGraph(int numVertices)** :

Crée un graphe avec le nombre spécifié de sommets.

— **void addEdge(Graph *graph, int src, int dest)** :

Ajoute une arête entre deux sommets du graphe, en tenant compte de la distance calculée par la fonction haversine.

— **double haversine(double lat1, double lon1, double lat2, double lon2)** :

Calcule la distance en kilomètres entre deux points donnés par leurs latitudes et longitudes, en utilisant la formule de Haversine.

— **void show-vertices(Graph *graph)** :

Affiche les latitudes et longitudes de tous les sommets du graphe (utile à des fins de test et de vérification).

— **void dijkstra(Graph *graph, int src, int dest, OsmGpsMap *map, int* Path, double* distance)** :

Implémentation de l'algorithme de Dijkstra pour trouver le plus court chemin entre deux sommets donnés (src et dest).

- Utilise un tableau dist pour stocker les distances temporaires entre les sommets et le sommet source.
- Utilise un tableau sptSet pour marquer les sommets déjà visités dans le processus de Dijkstra.
- Utilise un tableau predecessor pour reconstruire le chemin optimal en remontant des sommets jusqu'au sommet source.
- Affiche la distance du plus court chemin.
- Peut également remplir les tableaux Path et distance avec le chemin et la distance trouvés.

— **void find-closest-point(double given-lat, double given-lon, const char * points-file, double * lat-result, double * lon-result)** :

Recherche le point le plus proche dans un fichier de points donné, en fonction des latitudes et longitudes.

— **OsmGpsMapTrack * link-points(double lat1, double long1, double lat2, double long2)** :

Crée une trace sur la carte reliant deux points donnés par leurs latitudes et longitudes.

— **int get-index(Graph *graph, double lat, double lon)** :

Recherche l'index d'un sommet dans le graphe en fonction de sa latitude et de sa longitude.

— **int get-empty-index(Graph *graph)** :

Recherche le premier index disponible dans le tableau de sommets du graphe pour ajouter de nouveaux points.

— **void file-plot(char *filename, OsmGpsMap *map, Graph *graph)** :

Lit un fichier de points et crée des points et des arêtes correspondantes dans le graphe, en ajoutant également des traces sur la carte.

— **void free-path(int * Path)** :

Réinitialise le tableau Path en mettant tous les éléments à 0.

— **void print-list(int * list,int size)** :

Affiche une liste d'entiers (utile pour la vérification et le débogage).

— **void sort-list(Graph* graph ,int* list,OsmGpsMap * map,int size)** :

Trie la liste d'indices de points pour optimiser le calcul du plus court chemin global en utilisant un algorithme de tri adapté (à implémenter).

2.4 Variables

`graph :` Un pointeur vers la structure `Graph` contenant les sommets et les arêtes.

`map :` Un pointeur vers l'objet `OsmGpsMap` représentant la carte.

`Path :` Un tableau d'entiers pour stocker les indices des sommets du plus court chemin trouvé.

`distance :` Un pointeur vers une variable double pour stocker la distance du plus court chemin.

2.4.1 Conclusion

En conclusion de ce chapitre sur la conception, nous avons examiné les structures, librairies, fonctions et variables essentielles de notre programme. Nous avons défini des structures telles que ‘Point‘, ‘Edge‘, ‘Graph‘ et ‘CallbackData‘ pour représenter les données géographiques et les graphes.

Les librairies comme ‘stdio.h‘, ‘stdlib.h‘, ‘math.h‘, ‘glib.h‘, ‘gtk/gtk.h‘, ‘gdk/gdkkeysyms.h‘, ‘osm-gps-map.h‘ et ‘assert.h‘ fournissent des fonctionnalités cruciales pour le bon fonctionnement des fonctions définies

Les fonctions sélectionnées remplissent des rôles spécifiques, du calcul d’itinéraires à la gestion des données et les variables telles que ‘graph‘, ‘map‘, ‘Path‘ et ‘distance‘ sont essentielles pour stocker et manipuler les données.

Ce chapitre fournitdonc une base solide pour l’implémentation de notre solution de navigation, en clarifiant les éléments clés de notre programme.

Chapitre 3

Implémentation

Ce rapport présente l'implémentation d'une solution pour calculer le chemin le plus court entre deux points d'une partie spécifique de certain pays en utilisant l'algorithme de Dijkstra et une interface graphique interactive "Map-viewer". L'utilisateur peut sélectionner les points de départ et d'arrivée sur une carte, et le programme affiche le chemin optimal en utilisant des marqueurs et des segments de ligne.

3.1 Structure de l'implémentation

La solution s'articule autour de deux modules principaux :

3.1.1 Calcul du chemin le plus court (C)

Ce module est implémenté en langage C et utilise l'algorithme de Dijkstra pour trouver le chemin optimal entre les points sélectionnés par l'utilisateur. Il prend en entrée les coordonnées des points de départ et d'arrivée, ainsi que la liste des cordonnées des points sur la carte formant un graphe. Le module retourne la séquence des nœuds constituant le chemin optimal et sa longueur.

3.1.2 Interface graphique (Map-Viwer)

Ce module est développé en Python en utilisant la bibliothèque OSM-GPS-MAP. Il crée une interface graphique interactive permettant à l'utilisateur de :

— Afficher la carte du monde.

— Sélectionner des points de départ et d'arrivée en cliquant sur la carte.

— Visualiser le chemin optimal calculé par le module C.

— Interagir avec l'interface pour zoomer et se déplacer sur la carte.

3.2 Fonctionnement détaillé

3.2.1 Algorithme de Dijkstra

L'algorithme de Dijkstra est implémenté comme décrit dans le template fourni. Il fonctionne de manière itérative en explorant les nœuds voisins du nœud actuel et en mettant à jour les distances les plus courtes connues. Le processus se poursuit jusqu'à ce que le nœud d'arrivée soit atteint ou que tous les nœuds aient été explorés.

3.2.2 Interaction avec l'interface graphique

L'interface graphique utilise les doubles cliques gauche pour capturer les interactions de l'utilisateur. Lorsque l'utilisateur clique sur la carte, les coordonnées du clic sont transmises au module C pour calculer le chemin le plus court entre le point de départ actuel et le nouveau point sélectionné. Le chemin calculé est ensuite affiché sur la carte en utilisant un traget marqué en rouge

3.3 Avantages et limitations

L'avantage principal de l'algorithme de Dijkstra est sa garantie de trouver le chemin le plus court dans un graphe pondéré avec des poids positifs. Cependant, sa complexité temporelle peut

être élevée dans certains cas, en particulier avec des graphes de grande taille. Dans notre application, nous utilisons l'algorithme de Dijkstra pour trouver efficacement le chemin le plus court entre deux points sélectionnés par l'utilisateur sur une carte, en optimisant sa mise en œuvre pour répondre aux exigences de performance spécifiques de notre application.

3.4 Tests et Validation

Dans cette section, nous présentons une série de photos illustratives de la carte que nous avons utilisée dans notre projet. Nous avons réussi à obtenir des données pour les villes suivantes : Hollywood, New Bury, New York, San Diego et UTC , notre travail s'inscrit dans un contexte où l'accès aux données sur les rues du monde est limité, en raison de leur nature secrète et exclusive. Les images mettent en évidence les différentes fonctionnalités et les étapes clés de notre application.

3.4.1 Le plus court chemin entre deux points A et B

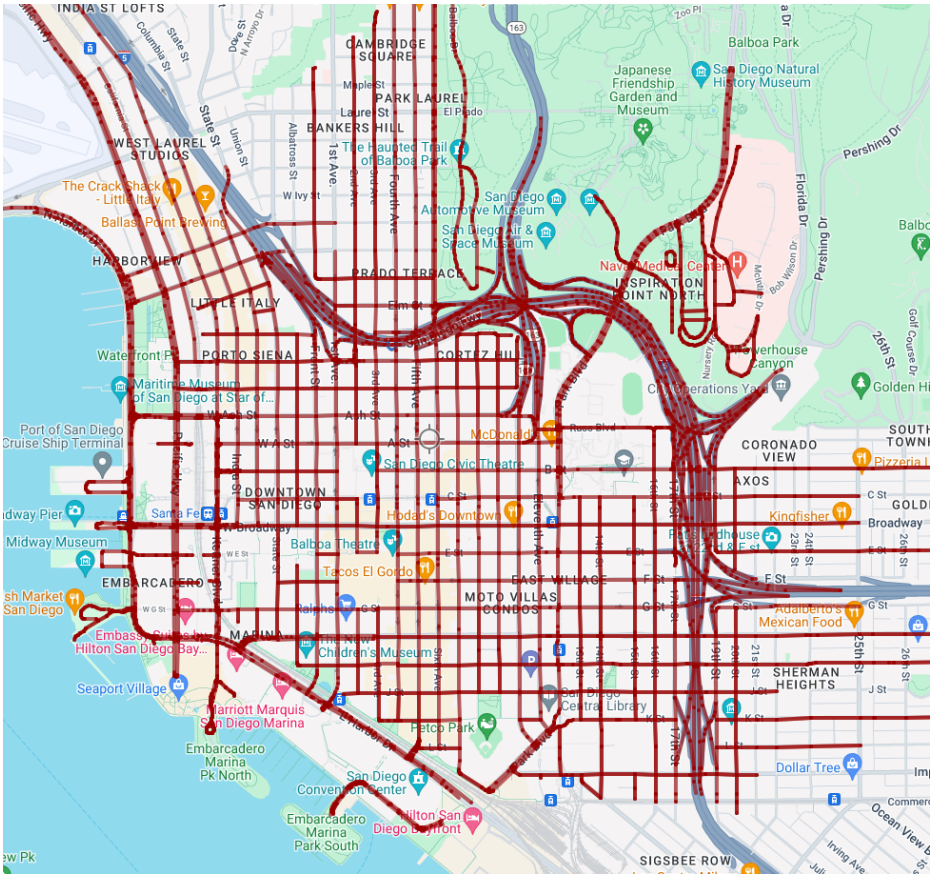

FIGURE 3.1 – La carte de la ville de San Diego

La Figure 3.1 illustre les différentes zones et points d'intérêt de la ville de San Diego. Les trajectoires rouges indiquent les rues accessibles, reflétant les données disponibles. Cette vue d'ensemble permet aux utilisateurs de se familiariser avec la zone géographique sur laquelle ils vont travailler.De plus, l'utilisateur peut naviguer sur la carte en utilisant la souris ou les flèches du clavier, tout comme sur une carte normale, offrant une expérience utilisateur intuitive et naturelle.

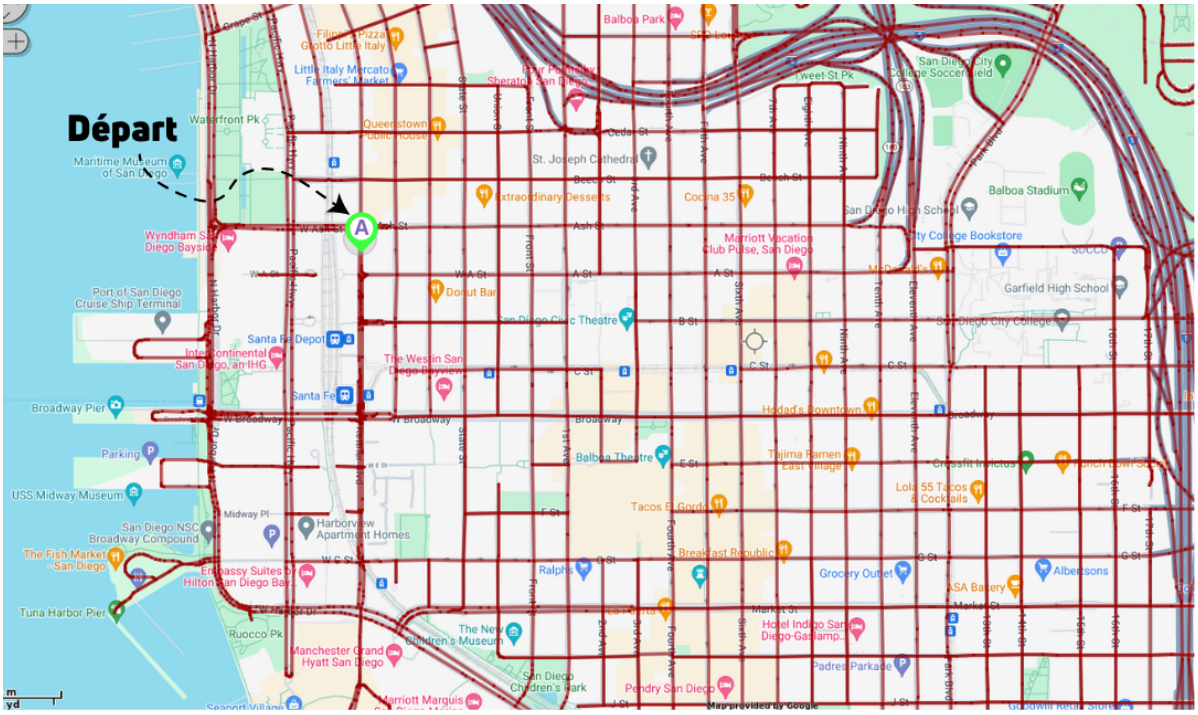

FIGURE 3.2 – Choix du point de départ

Dans la figure 3.2 l'utilisateur choisit le point de départ en cliquant deux fois avec le bouton gauche de la souris. Cette interaction simple et intuitive permet de sélectionner le point de départ pour le calcul du chemin optimal.

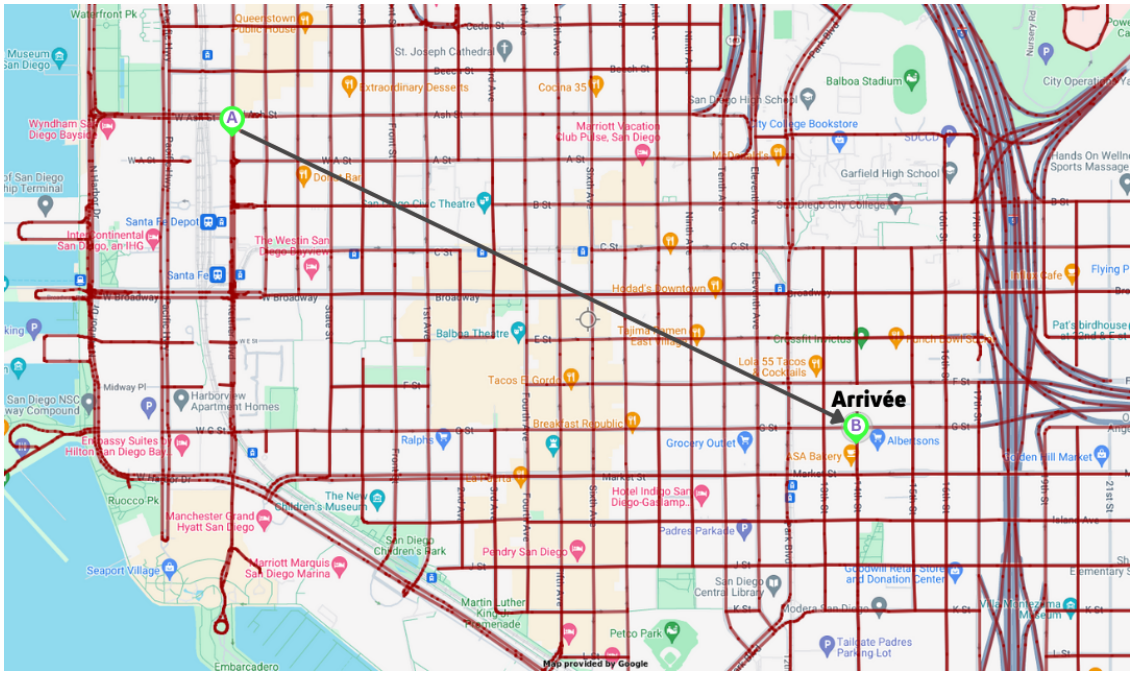

FIGURE 3.3 – Choix du point d'arrivé

Dans la figure 3.3 l'utilisateur choisit le point d'arrivée en cliquant deux fois avec le bouton gauche de la souris. De la même manière que pour le point de départ, cette interaction permet de sélectionner le point d'arrivée pour le calcul du chemin optimal. Après un double-clic droit,

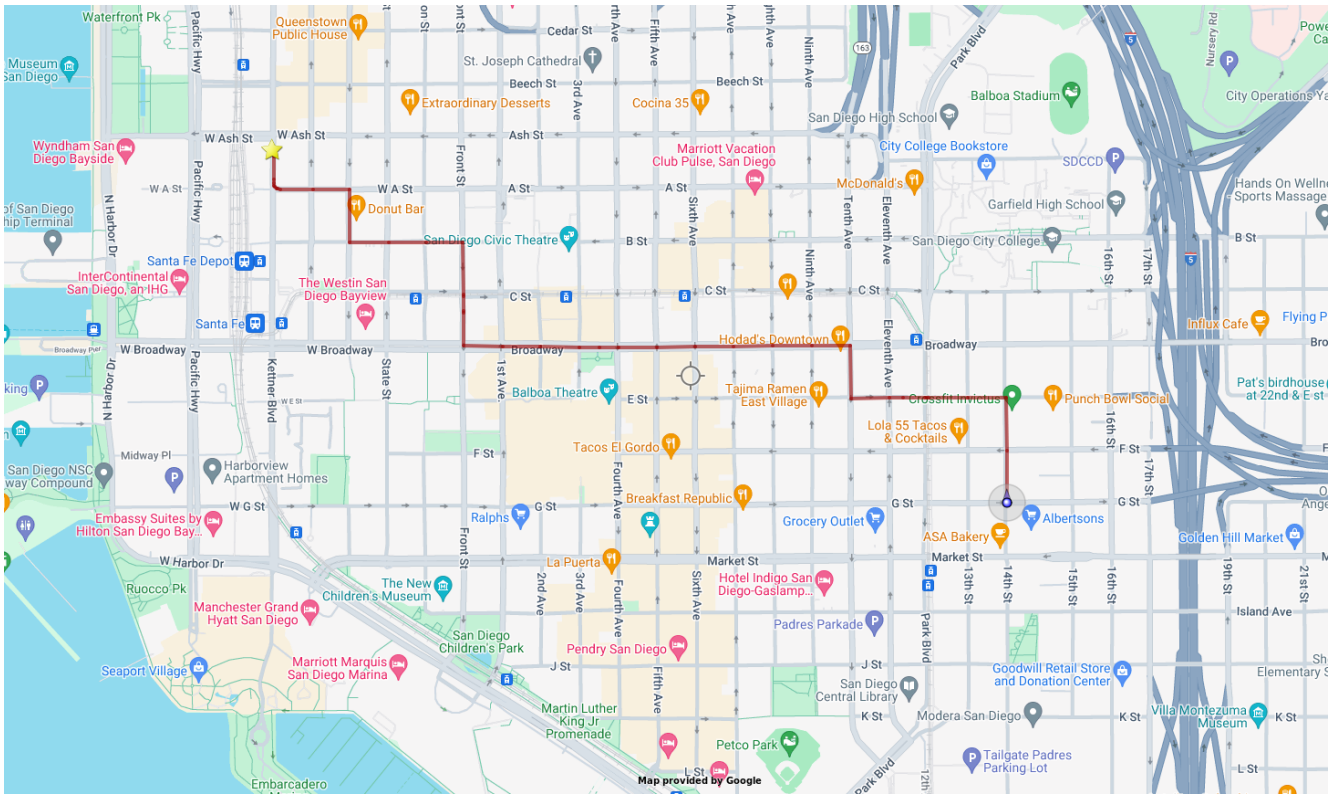

FIGURE 3.4 – Calcul de chemin optimal

le chemin optimal est représenté sur la carte. L'algorithme de Dijkstra est utilisé pour calculer le chemin le plus court entre le point de départ et le point d'arrivée, en tenant compte des différentes routes et obstacles présents sur la carte. Cette représentation visuelle du chemin optimal aide les utilisateurs à visualiser et à comprendre le parcours proposé.

3.4.2 Un chemin optimal entre N points

Cette solution est particulièrement adaptée pour les personnes effectuant des livraisons, permettant de choisir tous les points de livraison. L'algorithme s'occupe ensuite de calculer le chemin optimal, en tenant compte de la proximité des points sélectionnés.

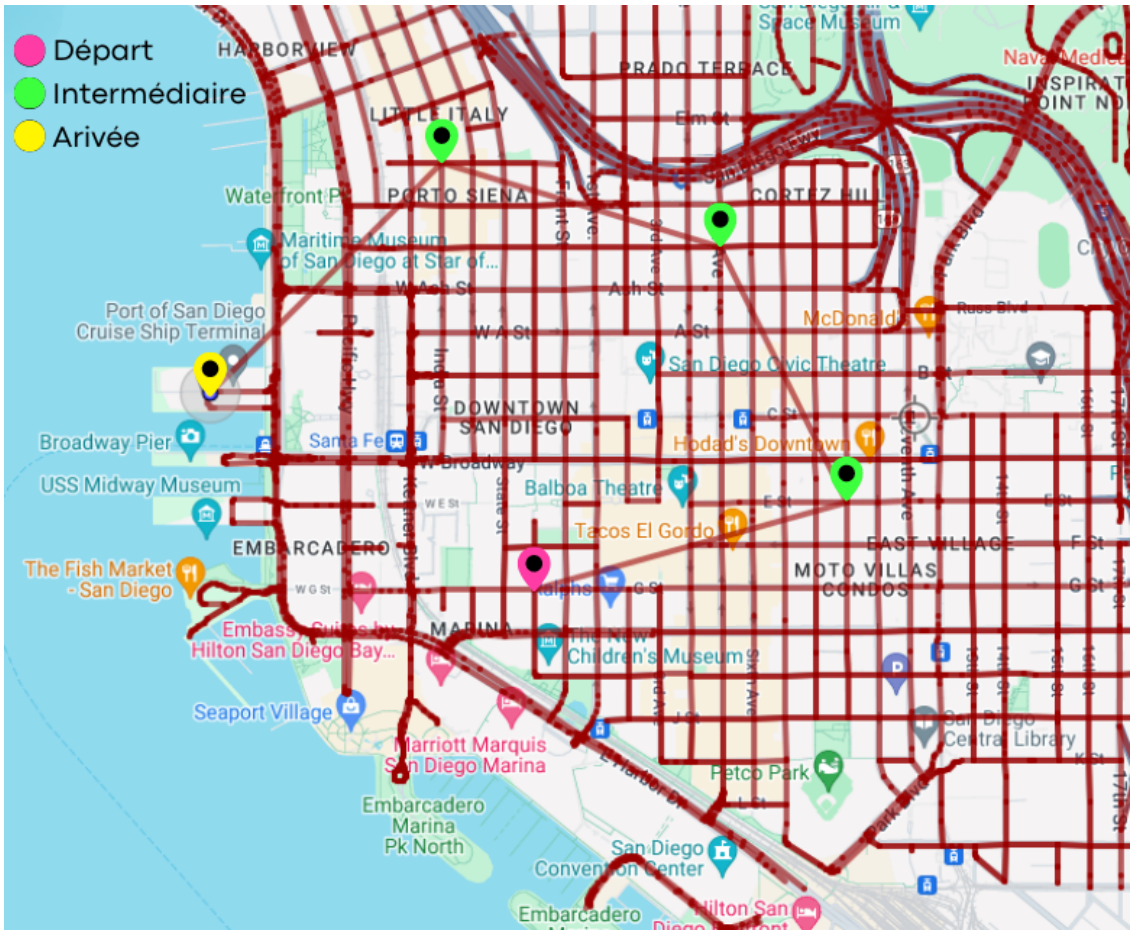

FIGURE 3.5 – Choix des points intermediaires

Le figure 3.5 illustre tous les points sélectionnés par l'utilisateur, y compris les points intermédiaires. Bien que ces points soient choisis de manière aléatoire par l'utilisateur, notre code s'occupe de les trier selon la distance de route, garantissant ainsi que le chemin optimal est calculé en tenant compte de la proximité des points. Cette fonctionnalité permet aux utilisateurs de visualiser et de comprendre les points sélectionnés et leur ordre sur le chemin optimal.

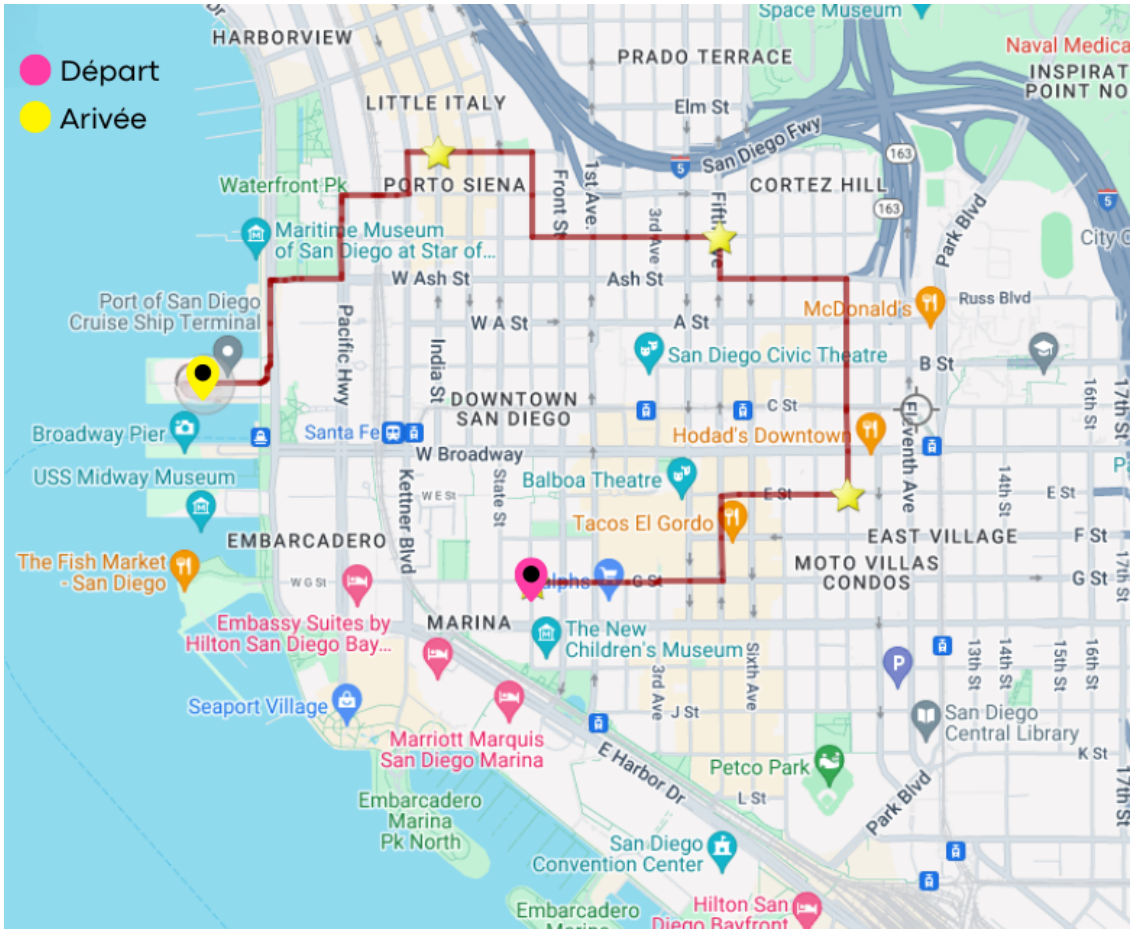

FIGURE 3.6 – Chemin optimal calculé

La figure 3.6 illustre un chemin optimal qui relie le point de départ et le point d'arrivée, en passant par les points intermédiaires sélectionnés par l'utilisateur. Le chemin est calculé en utilisant l'algorithme de Dijkstra, qui prend en compte la distance de route entre les points pour déterminer le chemin le plus court. Cette représentation visuelle du chemin optimal aide les utilisateurs à visualiser et à comprendre le parcours proposé, en mettant en évidence les points de départ, d'arrivée et intermédiaires.

3.4.3 Conclusion

En conclusion de ce chapitre sur l'implémentation, nous avons détaillé la mise en œuvre de notre solution visant à calculer le chemin le plus court entre deux points dans une région spécifique, en utilisant l'algorithme de Dijkstra et une interface graphique interactive "Map-viewer". Cette section a mis en lumière deux modules principaux qui constituent notre solution : le module de calcul du chemin le plus court en langage C et le module d'interface graphique en Python utilisant la bibliothèque OSM-GPS-MAP.

L'algorithme de Dijkstra a été implémenté pour trouver le chemin optimal entre les points sélectionnés par l'utilisateur. Son fonctionnement itératif a été expliqué, ainsi que son interaction avec l'interface graphique pour capturer les coordonnées des points sélectionnés.

La section sur les tests et la validation a fourni des captures d'écran illustrant le fonctionnement de notre application. Ces captures d'écran ont montré le processus de sélection des points de départ et d'arrivée, le calcul du chemin optimal, ainsi que la prise en charge de plusieurs points intermédiaires pour les livraisons. Ces tests et validations visuelles ont permis de démontrer l'efficacité et la convivialité de notre solution.

Chapitre 4

Conclusion

En conclusion, notre projet a mis en lumière l'importance des données cartographiques dans le développement d'applications pratiques et innovantes. En fournissant une interface utilisateur interactive pour la navigation sur des cartes spécifiques de villes comme Hollywood, New Bury, New York, San Diego et UTC, nous avons offert une solution flexible et intuitive pour la planification d'itinéraires. Cette application, accessible à la fois via la souris et les flèches du clavier, est particulièrement utile pour les livraisons, permettant aux utilisateurs de visualiser et de choisir facilement leurs points de livraison et de voir le chemin optimal calculé par l'algorithme. Notre travail souligne la puissance des données géographiques pour améliorer l'efficacité et l'accessibilité des applications de cartographie. Nous sommes fiers des réalisations de ce projet et restons ouverts à l'exploration de nouvelles fonctionnalités et améliorations.