# Rapport de soutenance finale

## Mai 2023

PerfectCourse

Votre allié pour optimiser votre temps et vos forces

Site web disponible ici:

Erwan Maigne-Montamat
Jean-Loup de Beauminy
Mathias Lecoeur
Thomas Tayrac

— Groupe 4 —

**Votre allié pour optimiser votre temps et vos forces** ..... **1**
**1. Introduction** ..... **3**
1.1. Problématique ..... 3
1.2. Objectif ..... 3
1.3. Résultats obtenus ..... 3
1.4. Fonctionnalités ..... 4
1.5. Technologies utilisées ..... 4
**2. Revue de la littérature et état de l'art** ..... **4**
**3. Processus de développement** ..... **5**
3.1. Analyse des besoins : ..... 5
3.2. Conception de l'architecture logicielle : ..... 6
3.3. Implémentation des fonctionnalités : ..... 7
3.3.1. Graphs et structures ..... 7
3.3.2. Récupération des données ..... 10
3.3.3. Algorithmes de plus courts chemins ..... 13
3.3.4. Implémentation graphique ..... 16
Chemin représenté graphiquement : ..... 20
3.3.5. Emplacement de l'employé: ..... 21
3.4. Site Web: ..... 24
**4. Tests et validation :** ..... **27**
**5. Documentation et maintenance :** ..... **28**
Maintenance : ..... 29

# 1. Introduction

## 1.1. Problématique

Après une rencontre avec un préparateur de commande d'une grande surface, un problème d'optimisation majeur a été identifié : l'absence de planification préalable des itinéraires en réserve entraîne une perte de temps significative, impactant la durée du travail et engendrant des pertes pour l'entreprise. Selon l'expérience du préparateur, une planification rapide permettrait de gagner entre 2 à 4 minutes par préparation de commande.

## 1.2. Objectif

Ce projet vise à développer une application qui propose des itinéraires optimisés dans les réserves des grandes surfaces pour les employés en préparation de commande ou en mise en rayon. L'objectif est d'améliorer leur productivité et de minimiser leurs déplacements dans l'entrepôt. Initialement, l'objectif était d'optimiser la circulation des clients en magasin. Cependant, suite aux conseils d'un responsable de grande surface, une modification de l'objectif a été décidée, axant le projet sur l'optimisation des déplacements des employés. Cette décision est basée sur la constatation que l'optimisation des déplacements des clients n'est pas nécessaire, car les vendeurs ont intérêt à maintenir une certaine confusion dans la localisation des articles.

## 1.3. Résultats obtenus

Le programme développé en langage C permet à l'utilisateur de fournir une liste d'articles, puis lui affiche le chemin le plus optimal dans le magasin pour récupérer ces articles.

### 1.4. Fonctionnalités

Optimisation des déplacements des employés : L'application génère des itinéraires rapides et efficaces en prenant en compte les caractéristiques des articles tels que leur poids, leur encombrement et leur fragilité.

### 1.5. Technologies utilisées

Algorithmes de plus courts chemins : langage **C**

Récupération des données : le site web permet à l'utilisateur de sélectionner des produits (**HTML**, **CSS**, **JavaScript**, **PHP**)

Référencement des articles et modélisation du magasin : base de données en format .txt.

## 2. Revue de la littérature et état de l'art

La recherche sur les algorithmes de plus courts chemins a connu des avancées significatives dans divers domaines, tels que la **logistique**, la gestion des entrepôts et la planification des itinéraires. Ces algorithmes visent à trouver le chemin le plus court ou le plus optimal entre deux points d'un réseau donné.

**L'algorithme de Dijkstra**, développé par Edsger W. Dijkstra en 1956, est l'un des premiers algorithmes utilisés pour résoudre le problème du plus court chemin. Il est basé sur la notion de la distance la plus courte entre les nœuds successifs et permet de trouver le chemin optimal dans un graphe pondéré.

Dans les années qui ont suivi, d'autres algorithmes de plus courts chemins ont été proposés, tels que l'algorithme de **Bellman-Ford** (1959) et l'algorithme de Floyd-Warshall (1962). L'algorithme de Bellman-Ford permet de trouver le plus court chemin dans un graphe pondéré même en présence d'arêtes négatives, tandis que l'algorithme de Floyd-Warshall trouve les chemins les plus courts entre toutes les paires de nœuds dans un graphe pondéré.

Plus récemment, des approches plus avancées ont été développées, telles que l'algorithme **A* (A-star)** qui utilise une heuristique pour guider la recherche du chemin optimal, et l'algorithme de recherche de chemin bidirectionnel qui explore le graphe à partir des deux extrémités pour accélérer la recherche du plus court chemin.

Dans notre projet, nous avons choisi d'implémenter l'algorithme de **Bellman** en raison de sa simplicité et de son efficacité pour résoudre le problème du plus court chemin dans un graph. Nous avons adapté cet algorithme en prenant en compte les contraintes spécifiques de notre contexte, telles que les articles lourds, légers, encombrants et fragiles.

Bien que peu de recherches aient été menées spécifiquement sur l'optimisation des déplacements des employés en préparation de commande ou mise en rayon dans un magasin, les travaux existants sur les **algorithmes de plus courts chemins** nous ont fourni une base solide pour développer notre approche algorithmique et concevoir notre système de gestion des déplacements.

# 3. Processus de développement

## 3.1. Analyse des besoins :

- Identification des problèmes et des défis liés à l'optimisation des déplacements des employés en préparation de commande ou en mise en rayon.
- Définition des objectifs spécifiques à atteindre, tels que l'amélioration de la productivité, la réduction des temps de déplacement et l'optimisation des itinéraires.

## 3.2. Conception de l'architecture logicielle :

- Développement d'une architecture logicielle se basant sur des **graphs** implémentés grâce au “struct” en C.
- Utilisation d'**algorithmes de plus court chemin** adaptés à l’implémentation de nos graphs.
- Gestion des **bases de données** dans des fichiers en .txt représentant la forme de l'entrepôt, la liste des articles et leurs références, ainsi que les articles sélectionnés pour être récupérés dans l'entrepôt.
- Utilisation de techniques de **modélisation** pour représenter le magasin, les rayons, les produits et les différentes contraintes liées aux déplacements des employés.

## 3.3. Implémentation des fonctionnalités :

### 3.3.1. Graphs et structures

**Implémentation du magasin :**

```
/**
 * Structure représentant un graphe.
 */
struct Graph {
    int order; // Nombre de noeuds dans le graphe
    struct Node *tableNodes[512]; // Tableau contenant les noeuds du
graphe
};

/**
 * Structure représentant un noeud du graphe.
 */
struct Node {
    int nodeID;  // Identifiant du noeud
    int boolShelf; // 0 si noeud extrémité d'une étagère, 1 si
                   // noeud produit, 2 si noeud entrée de
                   // l'entrepôt
    int shelf;   // Numéro de l'étagère (valide uniquement si
                 // boolShelf est égal à 0 ou 1)
    int department;  // Numéro du rayon (valide uniquement si
                     // boolShelf est égal à 0 ou 1)
    int boolTopBottom;          // 0 si noeud intermédiaire, 1 si
                                // noeud du haut, 2 si noeud du bas
    struct Node *adjTab[10];    // Tableau contenant les noeuds
                                // adjacents
    int nbAdj;                  // Nombre de noeuds adjacents
    int X;                      // Coordonnées X du noeud
    int Y;                      // Coordonnées Y du noeud
    int productsTab[3];         // Tableau contenant les références
                                // des produits du noeud
    int nbProducts;             // Nombre de produits dans le noeud
};
```

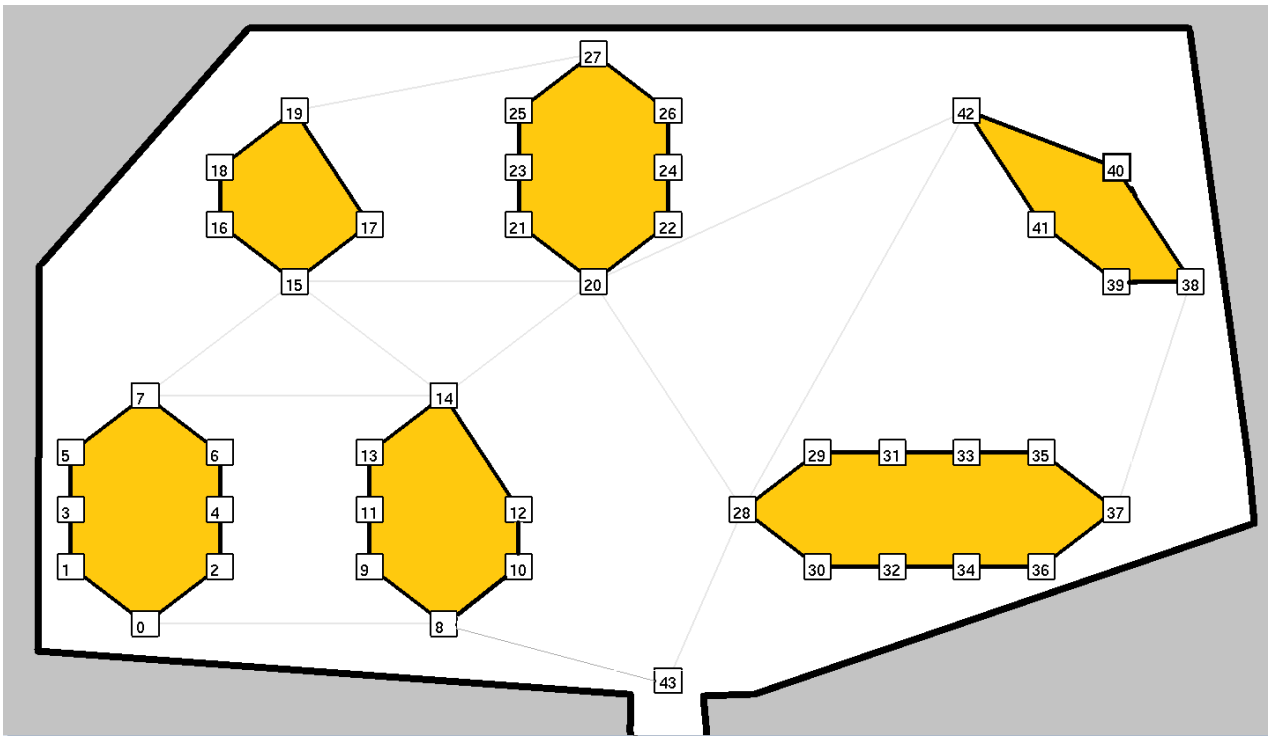
A l'aide des structures de données implémentées, nous avons créé une représentation du magasin sous forme de **graphe**. Chaque étagère du magasin est représentée par un nœud dans le graphe, utilisant la structure **"struct Node"** décrite précédemment. Les étagères sont interconnectées grâce à une liste d'étagères voisines, permettant ainsi de modéliser les connexions entre les différentes zones du magasin. Cette implémentation est basée sur une structure de liste d'adjacence.

Chaque nœud du graph possède des **coordonnées** (coordonnée x et coordonnée y) correspondant à l'**emplacement réel de l'étagère** dans le magasin. De plus, chaque nœud contient une liste d'articles associés à cette étagère, en utilisant la structure "struct Article" précédemment définie. Ces informations sur les articles seront utilisées ultérieurement lors de la création du plus court chemin dans le magasin.

**Représentation des produits :**

```
/**
 * Structure représentant un article.
 */
typedef struct Article {
    int ref;              // Référence de l'article
    char* nom;            // Nom de l'article
    float prix;           // Prix de l'article
    float poids;          // Poids de l'article (en kg)
    float volume;         // Volume de l'article (en m3)
    float resistance;     // Résistance de l'article entre 0 et 1
    int frozen;           // Indicateur indiquant si l'article
                          // est congelé (1) ou non (0)
} Article;
```

Pour représenter les produits dans l'application, nous utilisons la structure de données "struct Article" décrite précédemment. Chaque article est identifié par **une référence ou un code-barre unique**, et contient des informations telles que son nom, son prix et d'autres paramètres pertinents qui peuvent influencer l'ordre de récupération lors de la création du plus court chemin. Cette représentation des produits permet une gestion efficace des articles tout au long du processus d'optimisation des déplacements des employés.




Nous pouvons voir ci-dessus une représentation d'un entrepôt avec 6 rayons et 44 nœuds. **Chaque nœud est placé selon les paramètres Node.X et Node.Y**. et possède une liste de nœuds auxquels il est relié.
Le parcours des nœuds et le plus court chemin qui sera généré se feront le long des lignes entre les étagères et les rayons.

### 3.3.2. Récupération des données

Pour récupérer les données de modélisation du magasin, de la liste des produits proposés par le magasin et de la liste des produits du panier que nous devons récupérer, nous utilisons des bases de données en .txt disponible dans le projet à /src/bdd

Informations de **modélisation du magasin** en .txt :

```
department 1 top(2,1) bottom(2,5)
01(1,2) 0 1 2
02(3,2) 3 4 -1
03(1,3) 5 6 7

department 2 top(6,1) bottom(6,5)
01(5,2) 17 18 19
02(7,2) 20 -1 -1



department 3 top(4,7) bottom(4,10)
01(3,8) 27 28 29
02(5,8) 30 -1 -1

department 4 top(8,7) bottom(8,11)
01(7,8) 33 34 35
02(9,8) -1 -1 -1
03(7,9) 36 37 38

department 5 top(10,3) bottom(15,3)
01(11,4) 48 49 50
02(11,2) -1 -1 -1
03(12,4) 51 52 53

department 6 top(16,7) bottom(13,10)
01(15,7) -1 -1 -1
02(15,9) 69 -1 70
03(14,8) -1 -1 -1

Entry:
```

```
start(9,0)

1_bottom 2_bottom
1_top 2_top
1_bottom 3_top
3_top 4_top
3_bottom 4_bottom
3_top 2_bottom
4_top 2_bottom
4_top 5_top
6_top 5_bottom
6_bottom 4_top
6_bottom 5_top
start 5_top
start 2_top
```

**shop.txt** est séparé en plusieurs parties correspondant aux différents rayons. Dans ces rayons nous retrouvons les étagères avec 3 produits par etagères et des coordonnées graphiques pour pouvoir les situer dans l'espace. Le dernier paragraphe répertorie les liens entre les différents rayons.

Liste des produits en txt :

```
0 lait 1.99 1.5 0.6 0 0
1 fromage 2.49 0.2 0.1 0 0
2 yaourt 0.99 0.2 0.1 0 0
3 beurre 1.89 0.25 0.2 0 0
4 oeufs 2.99 0.4 0.4 1 0
5 pain 1.29 0.5 0.5 0 0
6 baguette 0.99 0.4 0.4 0 0
```

**catalogue.txt** répertorie tous les produits du magasin sous la forme :
"référence nom prix poids volume résistance surgelé"

Panier en .txt :

```
4 1
8 1
15 2
39 1
64 4
70 1
```

**cart.txt** répertorie les produits du panier. Par souci de simplicité, le fichier n'enregistre que la référence du produit et sa quantité.

Pour récupérer les données des fichiers texte, nous avons fait une fonction permettant à la fois de lire le fichier texte, mais aussi de **remplir un graph** avec tous les nœuds du fichier texte.

```
int FileRead(struct Graph *G, char *filename , int print);
```

Cette fonction utilise donc le fichier shop.txt pour en extraire un graph avec l'implémentation vue dans la partie "Implémentation du magasin".

Nous avons ensuite fait deux fonctions différentes pour extraire les information de catalogue.txt et cart.txt :

```
void remplir_catalogue(Article* catalogue, char* nom_fichier);
void remplir_panier(int* panier, char* file, int nbArticles);
```

### 3.3.3. Algorithmes de plus courts chemins

La partie la plus importante du projet puisque le “défi” algorithmique reposait dessus, nous utilisons donc l’algorithme de **Bellman** vu en cours afin de déterminer le chemin le plus court.

L’ordre des articles à rejoindre avec l’algorithme de Bellman est définie par une fonction annexe “scoreArticle” qui prend en compte le poids le nombre la fragilité afin d’avoir le chemin évitant au maximum la casse.

```
void algoBellman(struct Graph *graph, int sourceNode, struct eltBellman *tab)
{
    int i=0;
    while (i<graph->order)
    {
       if (i == sourceNode)
       {
             tab[i].distance=0;
             tab[i].precedentNode=-1;
       }
       else
       {
             tab[i].distance=9999;
             tab[i].precedentNode=-1;
       }
       i++;
    }

    i=1;
    while (i<graph->order)
    {
       int N=0;
       while (N<graph->order)
       {
            if (N != sourceNode)
             {
                   int rgAdj=0;
                   while (rgAdj<graph->tableNodes[N]->nbAdj)
                   {
                         int P = graph->tableNodes[N]->adjTab[rgAdj]->nodeID;
                         int poidsArc= (int)distance(graph,N,P);
                         if (tab[N].distance > tab[P].distance+poidsArc)
                         {
                               tab[N].distance=tab[P].distance+poidsArc;
                               tab[N].precedentNode=P;
```

```
                        }
                        rgAdj++;
                    }
                }
                N++;
            }
            i++;
        }
}
```

Et on applique ensuite bellman un à un sur les nœuds dont l'**ordre de passage** a été décidé en fonction du **score de l'article** (exemple : des paramètres comme la fragilité d'un article nous pousse à le choisir en dernier). Plus le score d'un article est élevé, plus il va être choisi en dernier.

**Formule du score :**

```
// Calcul du score
double score = (articleA->fragility) * fragilityFactor +
               (articleA->frozen) * frozFactor -
               (articleA->volume) * volumeFactor -
               (articleA->poids) * poidsFactor;
```

**Trie du panier et application de Bellman sur la liste triée :**

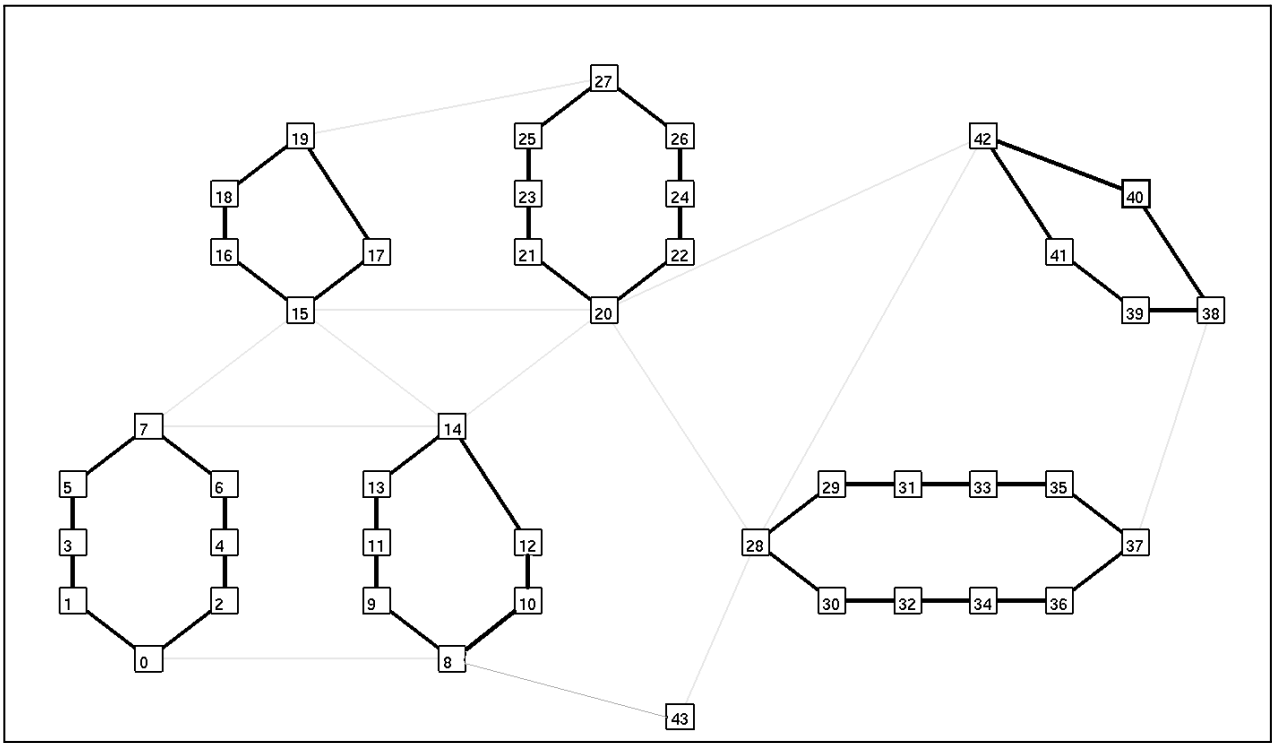
```
    // On trie les nodes du panier par ordre selon la fonction scoreArticle(const
void *a)
    for (int i = 0; i < nbNodes; i++) {
        for (int j = i + 1; j < nbNodes; j++) {
                if (scoreArticle(&catalogue[articles_sorted[i]]) >
                scoreArticle(&catalogue[articles_sorted[j]])) {
                      int temp = nodes[i];
                      nodes[i] = nodes[j];
                      nodes[j] = temp;

                      temp = articles_sorted[i];
                      articles_sorted[i] = articles_sorted[j];
                      articles_sorted[j] = temp;

                }
            }
      }
```

```
    // On applique l'algorithme de Bellman-Ford sur les noeuds triés
    for (int i = 0; i < nbNodes; i++) {
        algoBellman(graph, sourceNode, tab);
        targetNode = nodes[i];
        pathToTarget(tab, sourceNode, targetNode, path, &nbPathNodes);
        printPathToTarget(path, nbPathNodes, sourceNode, targetNode, print);

        for (int j = 0; j < nbPathNodes; j++) {
            path2[countPath] = path[j];
            countPath++;
        }

        sourceNode = targetNode;
    }
      return nodes;
}
```

### 3.3.4. Implémentation graphique

Nous avons réalisé une interface graphique pour visualiser le magasin et les parcours optimisés. Pour cela, nous avons utilisé **OpenGL**, une bibliothèque graphique multiplateforme.

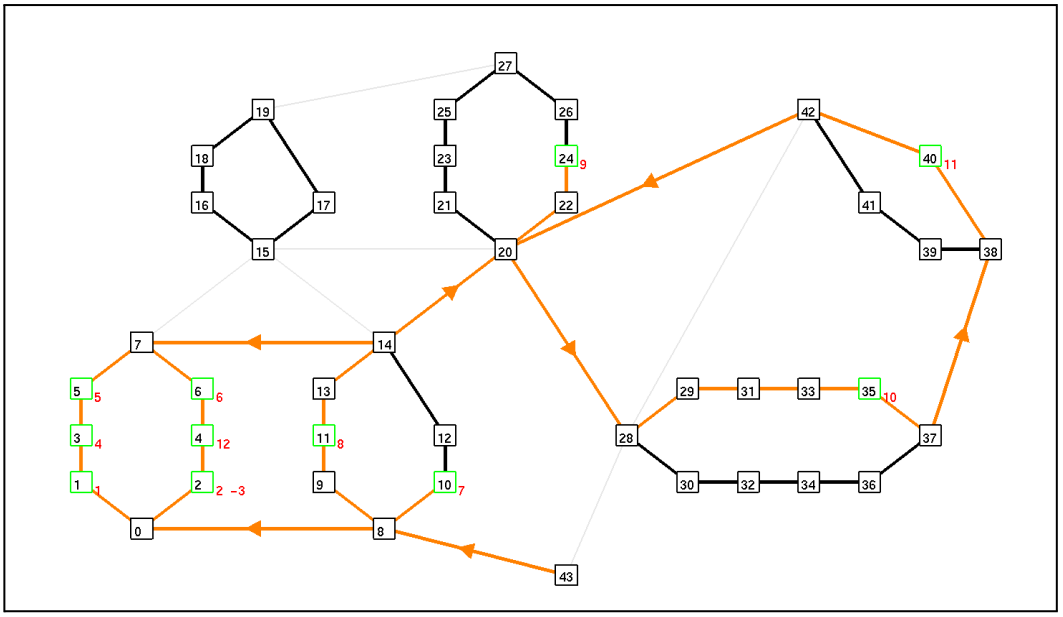
En utilisant les fonctionnalités d'OpenGL, nous avons pu créer une représentation visuelle du magasin avec ses étagères, ses rayons et ses connexions. Chaque élément du magasin est représenté par des formes géométriques appropriées, telles que des carrés ou des lignes.




L'implémentation graphique nous a permis d'afficher le magasin à l'écran. Par exemple, nous avons utilisé des fonctions OpenGL pour dessiner des carrés représentant les étagères et des lignes pour représenter les connexions entre les étagères.

Exemple, drawSquare **affiche un carré** vert si l'article est dans le panier et noir sinon :

```
void drawSquare(int x, int y, int size, int bool_panier) {
    size = size *1.4; // Demi-taille du carré

    glBegin(GL_QUADS);
    glColor3f(1.0f, 1.0f, 1.0f); // Blanc
    glVertex2i(x - size, y - size);
    glVertex2i(x + size, y - size);
    glVertex2i(x + size, y + size);
    glVertex2i(x - size, y + size);
    glEnd();


     if (bool_panier) {
          glLineWidth(2.0); // Épaisseur du contour
          glColor3f(0.0f, 1.0f, 0.0f); // Vert
     } else {
          glLineWidth(2.0); // Épaisseur du contour
          glColor3f(0.0f, 0.0f, 0.0f); // Noir
     }
    glBegin(GL_LINE_LOOP);
    glVertex2i(x - size, y - size);
    glVertex2i(x + size, y - size);
    glVertex2i(x + size, y + size);
    glVertex2i(x - size, y + size);
    glEnd();
}
```




Nous avons également utilisé **OpenGL** pour afficher le chemin le plus court généré par l'algorithme d'optimisation des déplacements. Ce chemin est tracé à l'écran sous la forme de lignes oranges fléchées et d'indices rouges indiquant l'ordre par lequel les articles doivent être pris.

L'implémentation graphique avec OpenGL a apporté une dimension visuelle à notre application, permettant aux utilisateurs de visualiser le magasin et les résultats de l'optimisation des déplacements de manière intuitive. Cette interface graphique améliore l'expérience utilisateur en offrant une **représentation visuelle claire et interactive** du magasin et des **parcours optimisés**.

**Exemple concret** d'un chemin généré par l'algorithme de Bellman et affiché grâce à la bibliothèque openGL :

**Panier** :

| ref | Article | Quantité | N° de noeud |
|---|---|---|---|
| 4 | oeufs | 1 | 2 |
| 7 | croissant | 1 | 3 |
| 31 | saumon_fume | 1 | 18 |
| 41 | saucisses_de_Montbeliard | 1 | 24 |
| 53 | framboise | 1 | 31 |
| 70 | boeuf_bourguignon | 1 | 40 |

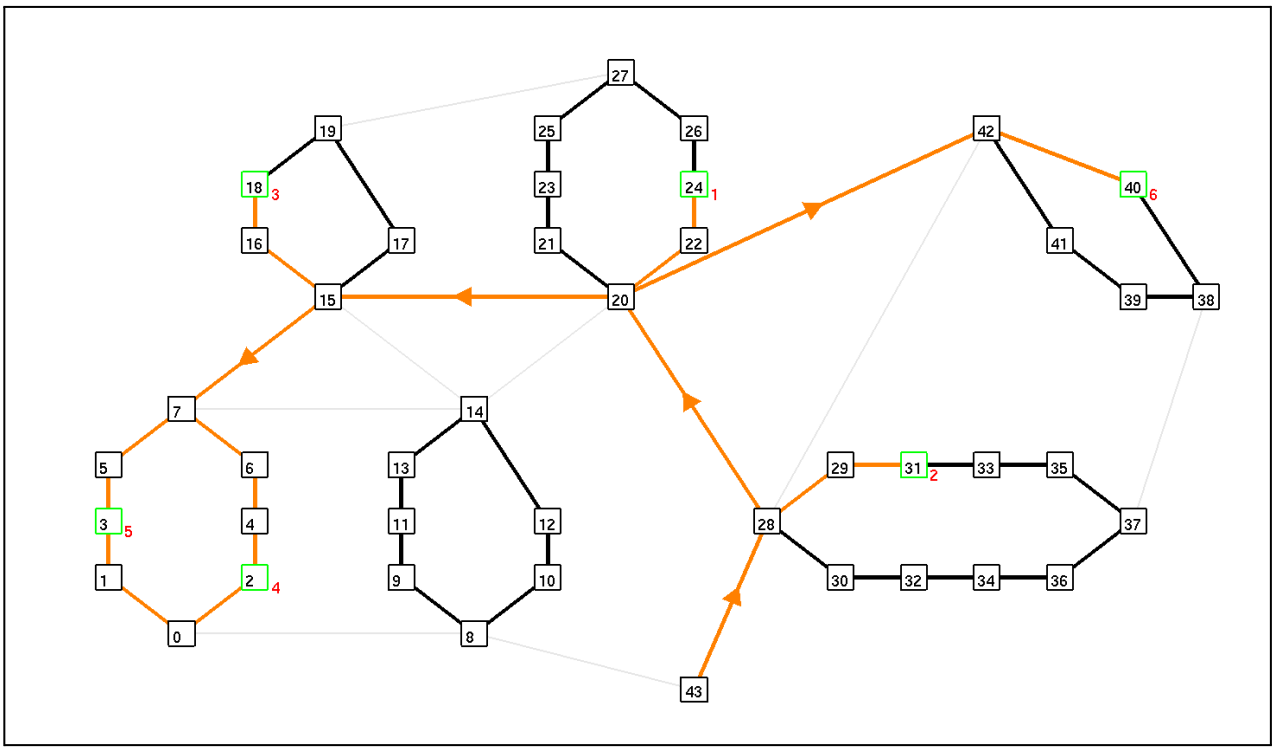
**On applique l'algorithme** de plus court chemin,
Ordre dans lequel les articles doivent etre recuperes :

| Article | ref | Priorité |
|---|---|---|
| saucisses_de_Montbeliard | 41 | 1 |
| framboise | 53 | 2 |
| saumon_fume | 31 | 3 |
| oeufs | 4 | 4 |
| croissant | 7 | 5 |
| boeuf_bourguignon | 70 | 6 |

Ordre des nœuds à parcourir pour faire le **chemin le plus optimisé**.

| N° de noeud | Article | ref |
|---|---|---|
| 43 | | |
| 28 | | |
| 20 | | |
| 22 | | |
| 24 | saucisses_de_Montbeliard | 41 |
| 22 | | |
| 20 | | |
| 28 | | |
| 29 | | |
| 31 | framboise | 53 |
| 29 | | |
| 28 | | |
| 20 | | |
| 15 | | |
| 16 | | |
| 18 | saumon_fume | 31 |
| 16 | | |
| 15 | | |
| 7 | | |
| 6 | | |
| 4 | | |
| 2 | oeufs | 4 |
| 0 | | |
| 1 | | |
| 3 | croissant | 7 |
| 5 | | |
| 7 | | |
| 15 | | |
| 20 | | |
| 42 | | |
| 40 | boeuf_bourguignon | 70 |

Chemin représenté **graphiquement** :



### 3.3.5. Emplacement de l'employé:

L'emplacement de l'employé était une donnée importante du projet puisqu'à terme l'objectif final était de guider pas à pas l'employé, il y a donc eue deux principale solutions imaginée:

Le localisation par QRcode:
Le concept était simple: chaque employé dans le suivi de sa préparation de commande va devoir scanner un **QR code** nous allons donc le récupérer de là déduire la position de l'article qu'il a pris et ensuite avec ceci nous allons déterminer sa position. Néanmoins dans la pratique cela n'a pas été complété pour une difficulté à comprendre les différents masque et l'ordre de stockage des données dans un QR code, voici ce qui a cependant été fait:

```
//Lecture des QRcode:

/*int [2] coordsfromQR(char* filename, struct Graph *G){
    int[-1, -1] coords;

    //Lecture du fichier:
    int ref = readQRcode(filename);
    if (ref == -1){
        return coords;
    }
    else{
    //On récupère les coordonnées:
    Nnode = ref_to_node(struct Graph *G, int ref);
    Node = G->nodes[Nnode];
    coords[0] = Node->x;
    coords[1] = Node->y;
    return coords;

    }

}

int readQRcode(char* filename){
    //TODO: OCR du QRcode et renvoie de la référence
    return (-1);
}
*/
```

Enfin pour la partie “décryptage” du QRcode nous avions quelque chose comme ceci :

```c
//On transforme la sdl image en matrice de 0 et 1 selon si le pixel est noir ou blanc
struct Matrix* imageToMatrix(SDL_Surface* image)
{
    struct Matrix* matrix = initMatrix(image->w, image->h);
    Uint8 r, g, b;
    Uint32 pixel;
    for (int i = 0; i < image->h; i++)
    {
        for (int j = 0; j < image->w; j++)
        {
            pixel = getpixel(image, j, i);
            SDL_GetRGB(pixel, image->format, &r, &g, &b);
            if (r == 0 && g == 0 && b == 0)
            {
                matrix->matrix[i][j] = 1;
            }
            else
            {
                matrix->matrix[i][j] = 0;
            }
        }
    }
    return matrix;
}
```

Structure Matrix qui était de cette forme :

```c
struct Matrix
{
    int width;
    int height;
    int** matrix;
};

struct Matrix* initMatrix(int width, int height)
{
    struct Matrix* matrix = malloc(sizeof(struct Matrix));
    matrix->width = width;
    matrix->height = height;
    matrix->matrix = malloc(sizeof(int*) * height);
    for (int i = 0; i < height; i++)
    {
        matrix->matrix[i] = malloc(sizeof(int) * width);
    }
    return matrix;
}
```

Les étapes manquantes sont donc:

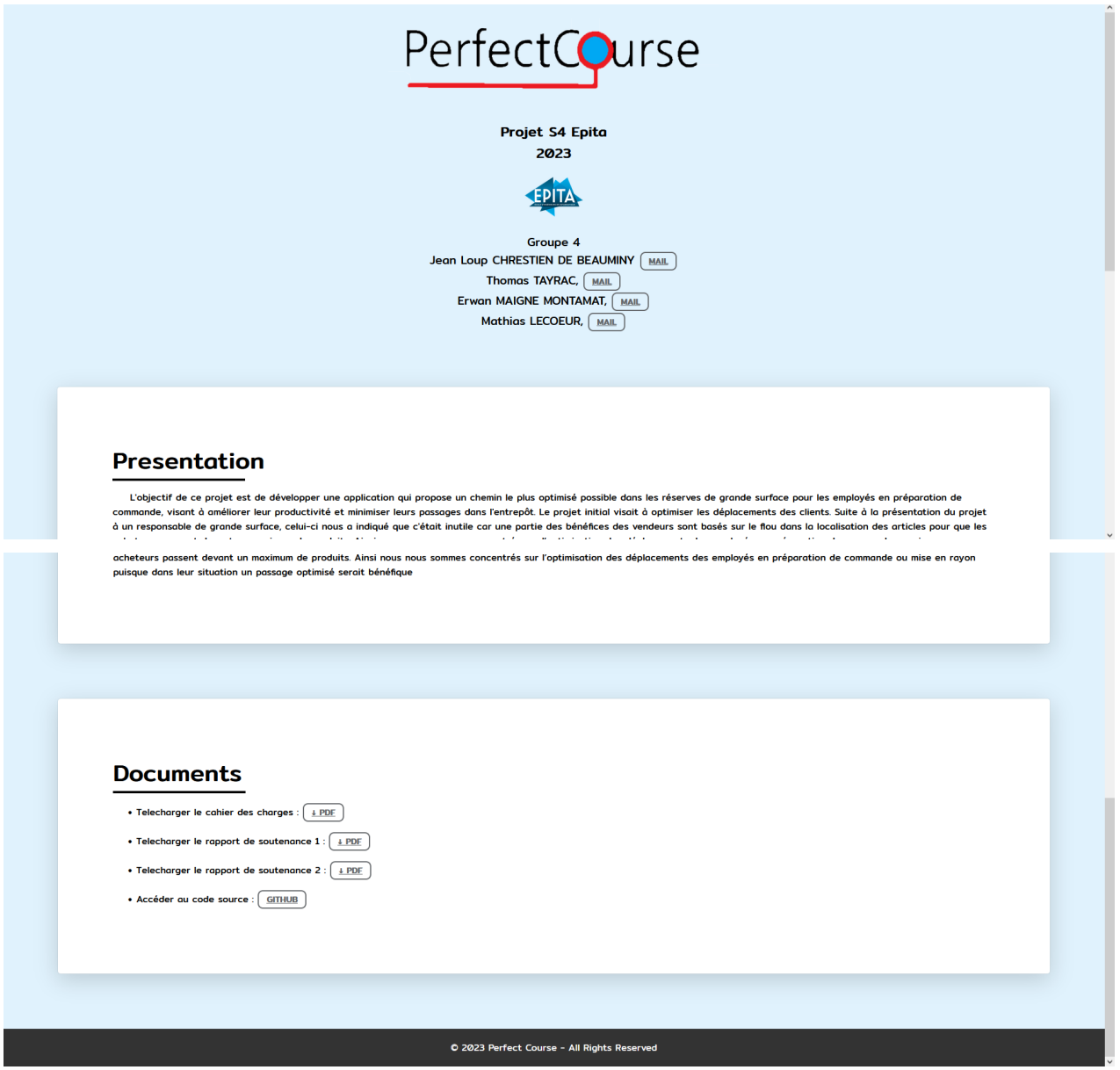
- La détermination du masque appliqué, l'application du reverse de ce masque, la lecture des données.

- La deuxième possibilité de localisation aurait été via une triangulation de position avec des balises bluetooth (symbolisé par des zone couverte par des balises dans le logiciel) néanmoins n'ayant pas eu le temps de finir les QRcode nous ne nous sommes pas attaqués à ceci.

Enfin notons que comme précisé dans le dernier rapport la localisation est passé **au second plan après le test et la vérification de toutes les fonctionnalités**, or nos effectifs ayant été réduit à deux personnes nous avons eue du mal à finir les fonctionnalités principales et à assurer notre conformité avec les consignes (aucun warning lors de la compilation par exemple)

## 3.4. Site Web:

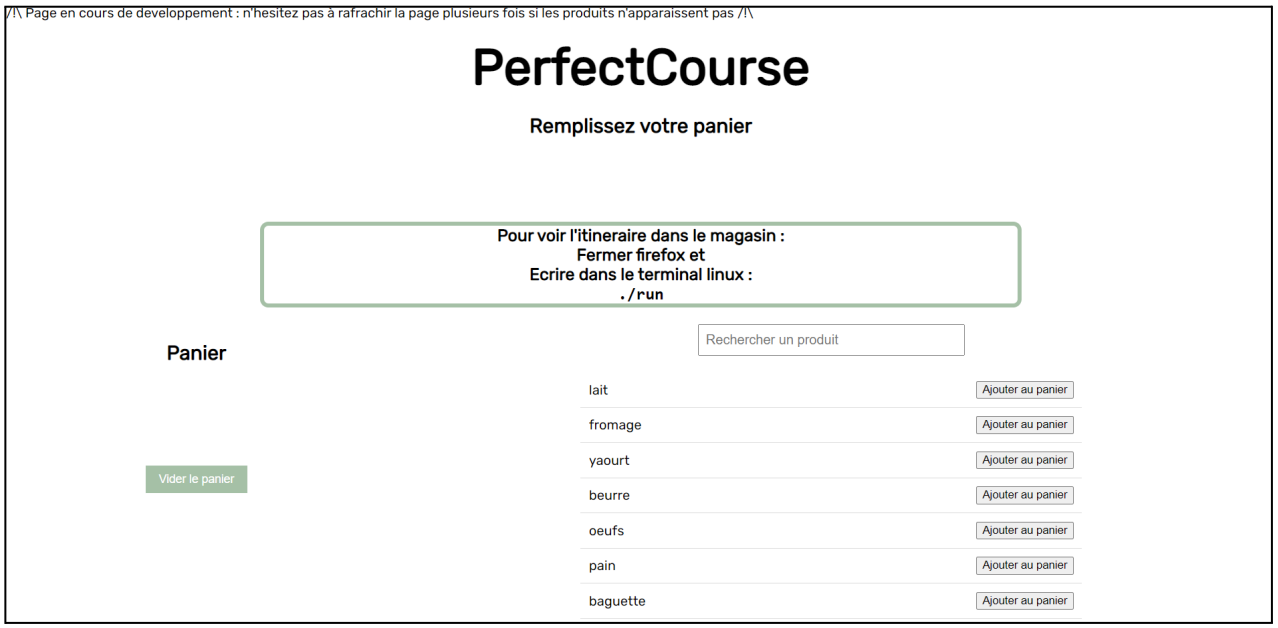
Le site web est toujours hébergé sur Néo Cities et contient l'accès à toutes les données nécessaire au suivi du projet comme notamment les différents rapports de soutenances le repos github du projet et il ressemble à ceci:
(Il est trouvable à l'adresse: https://perfectcourse.neocities.org/ )



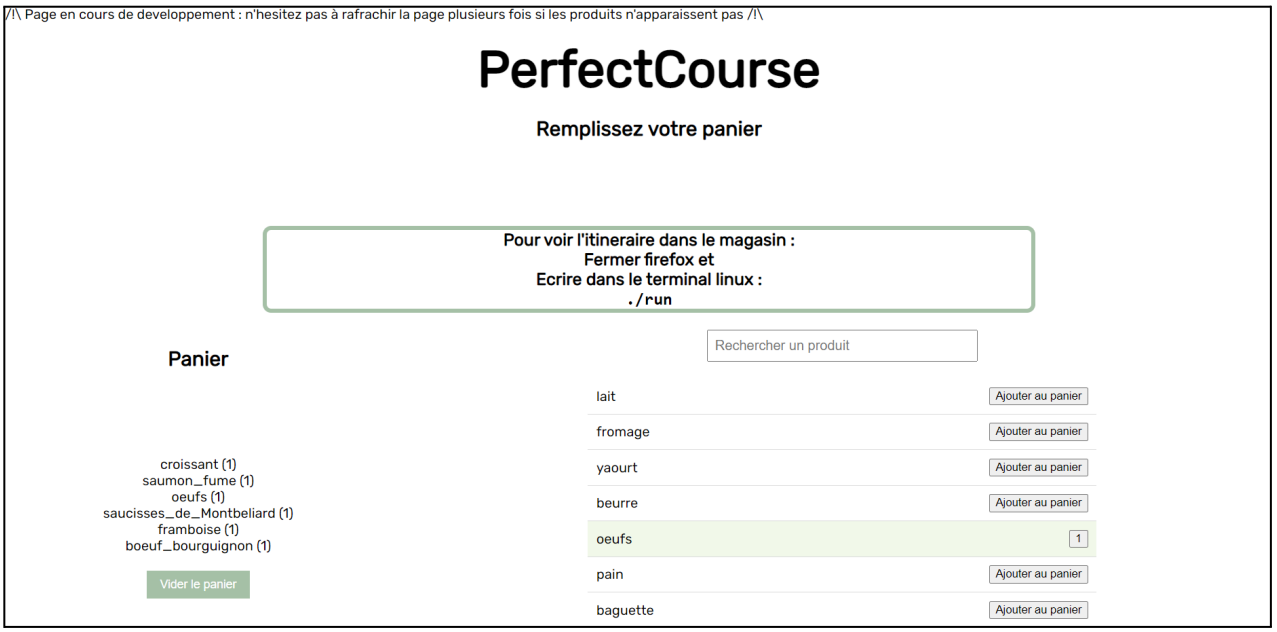

(html css)

Nous avons également créé un **second site qui permet de créer son panier** en choisissant à la main les produits souhaités sans avoir à écrire soi-même son fichier "cart.txt" celui-ci étant disponible à l'adresse:
https://perfectcourse.000webhostapp.com/ (Lien disponible en annexe également)

Ou en faisant ./create_cart afin d’avoir une liste de course utilisable par le projet



Chaque produit sélectionné avec sa quantité se voit ajouté à la liste sur le côté gauche comme ceci:




à partir de là il suffit de faire **./run** et de là votre machine récupérera toute la liste que vous venez de faire et lancera la recherche du **chemin le plus optimisé** pour vous.

Ce site est codé en html, css, php et javascript. Le **code javascript** permet de récupérer tous les articles de catalogue.txt hébergés à la racine du site, pour afficher la liste de choix qui permet de remplir à son tour cart.txt (le panier).

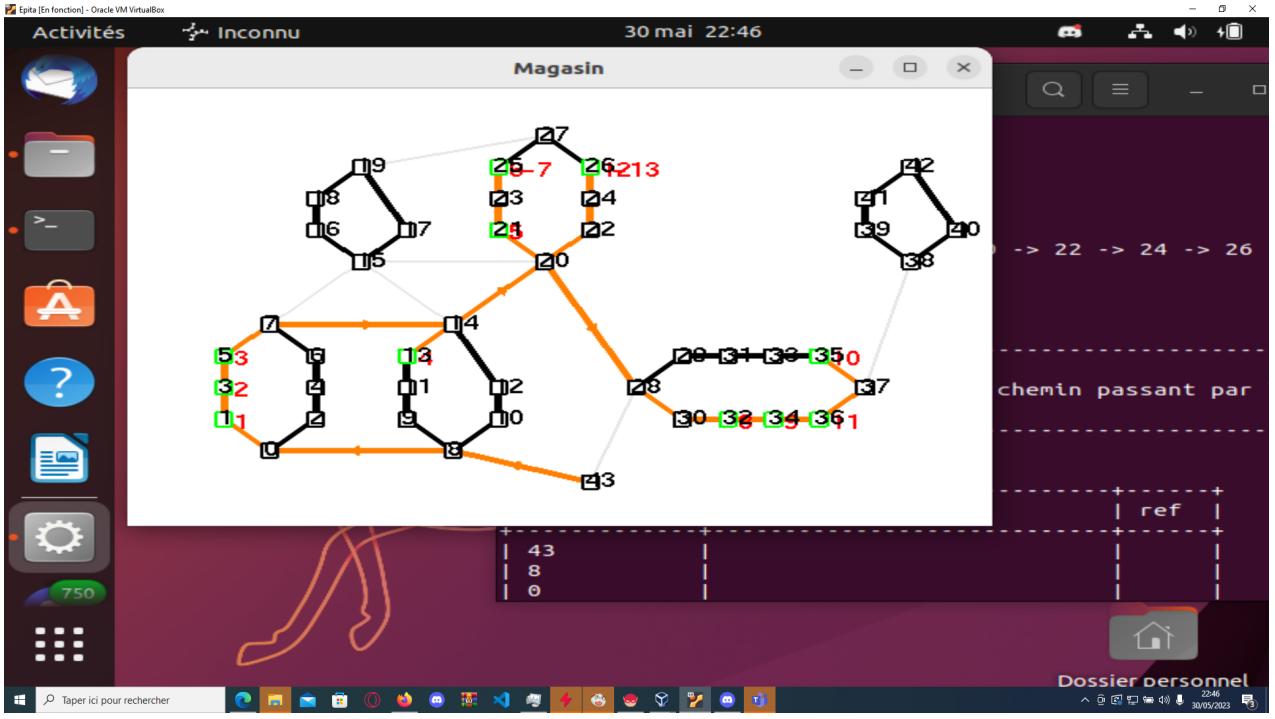
le fichier ./create_cart du projet va ensuite télécharger le fichier cart.txt hebergé à l'adresse https://perfectcourse.000webhostapp.com/cart.txt. Ce fichier est téléchargé directement dans le répertoire /src/bdd qui est utilisé pour générer le plus court chemin.

# 4. Tests et validation :

Nous avons pu faire differents tests pour attester du bon fonctionnement de notre projet :

- tests unitaires pour vérifier le bon fonctionnement des différentes parties de l'application sur chaque machine des personnes sur le projet ainsi qu'à Epita.

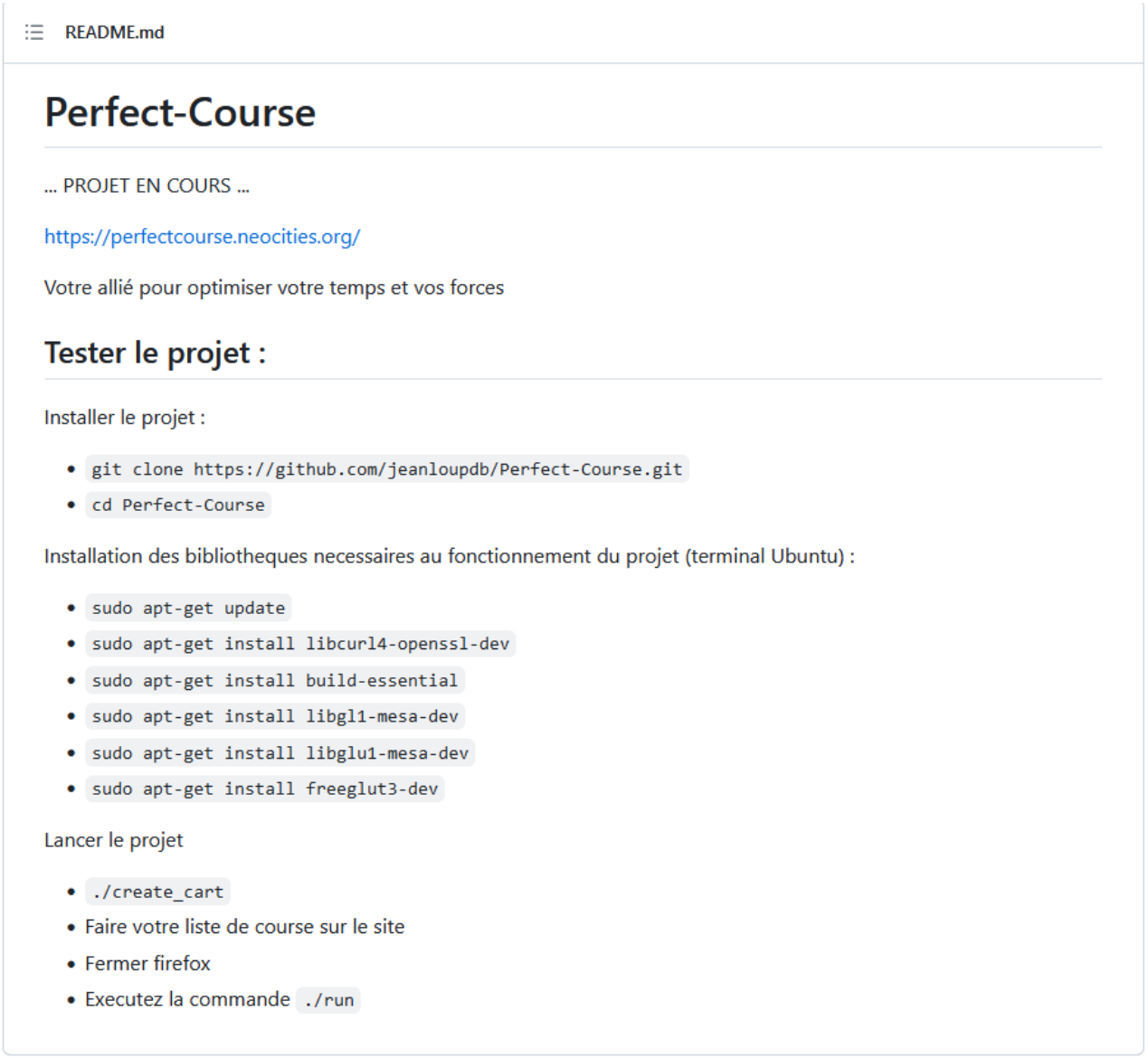
Donnant des résultats satisfaisant tel que vous le voyez ci-dessous :




- Utilisation de scénarios de test représentatifs pour évaluer les performances de l'application en termes d'optimisation des itinéraires sur une multitude d'itinéraires différents.
- Validation de l'application auprès d'utilisateurs finaux, tels que des employés de préparation de commande ou de mise en rayon, pour recueillir des retours et améliorer l'expérience utilisateur en nous adressant notamment aux connaissances qui travaillent dans la préparation de commandes et qui nous avaient soufflé l'idée.

# 5. Documentation et maintenance :

Rédaction d'un **fichier graph.h** contenant toutes les informations nécessaires à la compréhension du projet (commentaires utiles et précisions). Notons également que **l'entièreté du code a été commenté** lorsque nous avons perdu notre 3éme membre de manière à ce que nous puissions reprendre son travail de manière fluide.
Rédaction d'un readme précisant étape par étape la compilation du projet, incluant la mise à jour de toutes les bibliothèques nécessaires (si ces dernières ne sont pas déjà installées).



README.md

## Perfect-Course

... PROJET EN COURS ...

https://perfectcourse.neocities.org/

Votre allié pour optimiser votre temps et vos forces

### Tester le projet :

Installer le projet :

- `git clone https://github.com/jeanloupdb/Perfect-Course.git`
- `cd Perfect-Course`

Installation des bibliotheques necessaires au fonctionnement du projet (terminal Ubuntu) :

- `sudo apt-get update`
- `sudo apt-get install libcurl4-openssl-dev`
- `sudo apt-get install build-essential`
- `sudo apt-get install libgl1-mesa-dev`
- `sudo apt-get install libglu1-mesa-dev`
- `sudo apt-get install freeglut3-dev`

Lancer le projet

- `./create_cart`
- Faire votre liste de course sur le site
- Fermer firefox
- Executez la commande `./run`

## Maintenance :

Etant donné que le projet nous tient à coeur nous envisageons sérieusement de continuer à travailler dessus, pour pouvoir à terme proposer le projet finit à des clients privés, ce qui inclut notamment la mise en place d'un processus de maintenance pour assurer la pérennité de l'application et la prise en compte d'**éventuelles évolutions** ou améliorations futures.
Dans lesquelles figurent déjà:

La mise à jour de la localisation en temps réels
La création d’un créateur d’entrepôt assisté (et simplifié)
Des tutoriels d’utilisation du projet

## Annexes:

Rappel du site de suivi du projet:
https://perfectcourse.neocities.org/
Ou scannez le QRcode ci-dessous

Site de création de panier:
https://perfectcourse.000webhostapp.com/