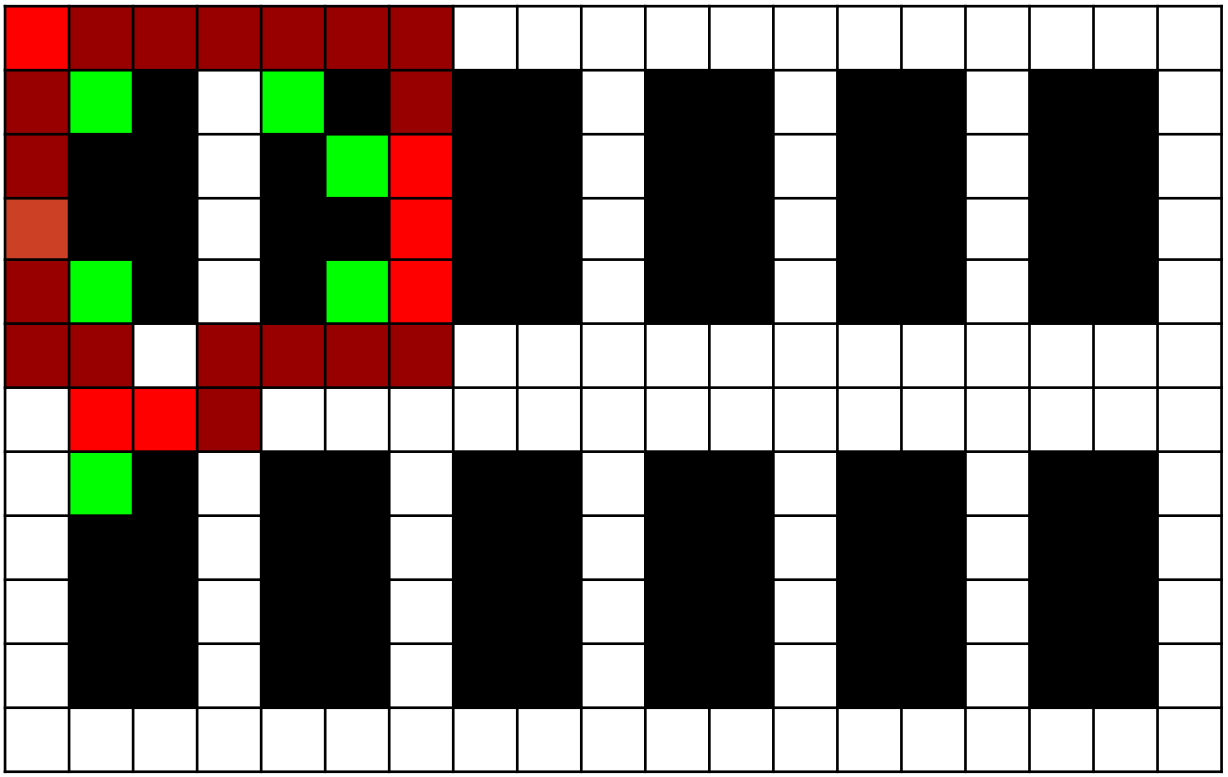
Rapport de soutenance 2

# Perfect Course

Votre allié pour optimiser votre temps et vos forces



https://perfectcourse.neocities.org/

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

— Groupe 4 —

# I. Les adaptations du planning

Nous avons révisé notre plan d'action pour atteindre plusieurs objectifs intermédiaires :

- **Objectif 1** : Notre première étape consiste à obtenir un catalogue des produits proposés dans la grande surface, avec les références associées (de 0 à n) pour faciliter notre travail. Nous demanderons donc à la grande surface de nous fournir ce catalogue sous forme de fichier texte, que nous convertirons ensuite en tableau où chaque élément représente le nom du produit.

- **Objectif 2** : Pour déterminer le chemin le plus court pour l'employé, nous avons besoin d'une liste de courses du client, représentée par un tableau de même taille que celui du catalogue, où chaque élément correspond au nombre de produits demandés pour chaque référence. Par exemple, pour un catalogue [pain, lait, bouteille] et un panier [1, 0, 4], le client a commandé 1 pain et 4 bouteilles.

- **Objectif 3** : Nous devrons déterminer les étagères (nœuds) que l'employé devra traverser pour récupérer les produits à partir de la liste de courses du client. Cela sera représenté par un tableau de taille G.order où chaque élément sera un 0 ou un 1, selon que l'employé doit ou non passer par ce nœud.

- **Objectif 4** : Nous devrons implémenter un algorithme capable de renvoyer dans un tableau les distances les plus courtes entre chaque nœud du graph et un nœud source. Nous avons choisi l'algorithme de Bellman parmi d'autres, et le résultat de notre fonction sera un tableau d'éléments Bellman (distance_min, precedentNode).

- **Objectif 5** : Nous devrons également implémenter un algorithme capable de renvoyer le chemin le plus court entre deux nœuds.

- **Objectif 6** : Enfin, nous devrons rassembler les cinq objectifs précédents pour atteindre notre objectif global.

# II. Présentation du groupe:

## A. Jean-Loup de Beauminy

Ma passion pour l'informatique a commencé en seconde, lors de mes cours de mathématiques où nous avons programmé sur nos calculatrices. Depuis, j'ai acquis des connaissances dans d'autres langages tels que CSS/HTML, JavaScript, Python et le langage C, ce qui me confère un avantage certain étant donné que le projet est en C. Cependant, les parties relatives aux interfaces graphiques et à la création de chemins constituent des défis que je suis impatient de relever.

## B. Mathias Lecoeur:

Ayant déjà effectué la spécialisation en informatique à l'EPITA une première fois, j'ai acquis des compétences solides en C. Ce projet est pour moi l'occasion d'acquérir de nouvelles compétences, de combler mes lacunes et de mettre en pratique les connaissances acquises lors de ma prépa classique.

## C. Erwan Maigne-Montamat:

Je suis passionné par l'informatique depuis l'âge de 10 ans, lorsque j'ai commencé à jouer à Minecraft. J'ai ensuite suivi le développement de plugins et de mods, avant de choisir un bac S spécialité Sciences de l'Ingénieur avec une option Informatique et Sciences du Numérique. Je suis particulièrement intéressé par le domaine de l'algorithmique, et la recherche de chemins, qui fait partie de ce projet, me motive tout particulièrement.

### D. Thomas Tayrac:

Je suis passionné de programmation et d'algorithmie depuis ma première année de lycée, où j'ai découvert le langage Python en cours d'ICN. La programmation est une façon de penser un peu spéciale et complexe, mais elle ouvre la voie à de nombreuses possibilités et solutions pour résoudre des problèmes. Ce projet nous permet justement de résoudre un problème de perte d'énergie et de temps pour un employé, ce qui est à mes yeux très utile. Si nous réussissons à mener à bien ce projet dans les moindres détails, cela me rendra très fier. C'est pourquoi j'ai une motivation particulière à réaliser ce projet de la meilleure manière possible.

# III. Répartition des tâches et planning

## A. Répartition des tâches

*Prendre également en compte les adaptation de planning vues précédemment

| Date (plus ou moins approximative) | Tâche ou sous problème à résoudre |
|---|---|
| 30 janvier 2023: | Remise des listes des groupes |
| 31 janvier 2023: | Validation définitive des groupes |
| | Formalisation du projet et création du cahier des charges |
| 06 février 2023: | Remise du cahier des charges |
| 08 février 2023 : | Validation théorique du cahier des charges |
| 17 février 2023 | Validation effective du cahier des charges |
| Semaine du 20 février 2023: | Implémentation des graphs en C<br><br>Création de l'outil de modélisation de l'entrepôt sous forme de graphe (via l'implémentation des graphs en C),<br><br>Codage de la base du site pour héberger les fichiers de formalité (code, rapports, …) |
| Semaine du 27 février 2023: | Implémentation d'une reconnaissance d'image pour lire les qr codes. Localisation de l'employé grâce au QR code. |
| Semaine du 6 mars 2023: | Création d'une interface de démonstration dans laquelle on peut faire se déplacer l'employé et simuler le chemin emprunté,<br><br>1 ère Soutenance |

| | |
|---|---|
| Semaine du 13 mars 2023: | Implémentation du premier algorithme de recherche de chemin, basé sur la distance parcouru (et donc le temps) |
| Semaine du 20 mars 2023: | Création d'un entrepôt type et archivage d'une liste de produits imaginaires.<br>Création de l'interface permettant d'enregistrer des références |
| Semaine du 27 mars 2023: | Création de l'algorithme de recherche de chemin selon le poids et la fragilité des objets |
| Semaine du 11 avril 2023: | Semaine de rattrapage des différents retards du projet, débogage d'éventuelles coquilles et/ou rectification de la feuille de route.<br>Création du design définitif du site et mise en place de ce dernier |
| Semaine du 18 avril 2023: | Mise en place de localisation "en temps réel" grâce à une simulation de balise bluetooth (amélioration de la précision comparé au QR code) |
| Semaine du 25 avril 2023: | Détection de l'égarement de l'employé et création mise en place du correcteur d'itinéraire.<br><br>Préparation de la seconde soutenance. |
| Semaine du 1er mai 2023: | 2éme soutenance (nous supposons) |
| Semaine du 8 mai 2023: | Mise en place de l'indication "en temps réel" de la direction à suivre pour l'employé (dans la simulation toujours). |
| Semaine du 15 mai 2023: | Mise en place d'un parser pour récupérer la liste des produits selon les références données |
| Semaine du 22 mai 2023: | Mise en place d'un générateur aléatoire de commande et récupération des dites commandes |

| | |
|---|---|
| | pour tester l'application |
| à partir de maintenant les dates sont toujours à supposer puisque nous ne savons pas si seulement la première soutenance a été décalé ou non | |
| | Tests divers et variés et amélioration de l'interface graphique |
| | Semaine de rattrapage des différents retards du projet, débogage d'éventuelles coquilles.<br>Début de préparation du rapport de projet |
| | Débogage des éventuels bugs persistants, préparation du rapport de projet ainsi que de la soutenance |
| Semaine du 29 mai 2023: | Dernière soutenance (à voir si cette dernière sera décalé ou non) |

## B. Répartition des tâches :

| Tâche | Jean Loup | Thomas | Erwan | Mathias |
|---|---|---|---|---|
| Implémentation graph en C | X | X | X | X |
| Codage base du site | X | | | |
| Création outil de modélisation entrepôt | | X | X | X |
| Localisation grâce au QR code | X | | X | |
| Reconnaissance QR code | | X | | X |
| Interface de démonstration | X | X | X | X |
| Préparation | X | X | X | X |

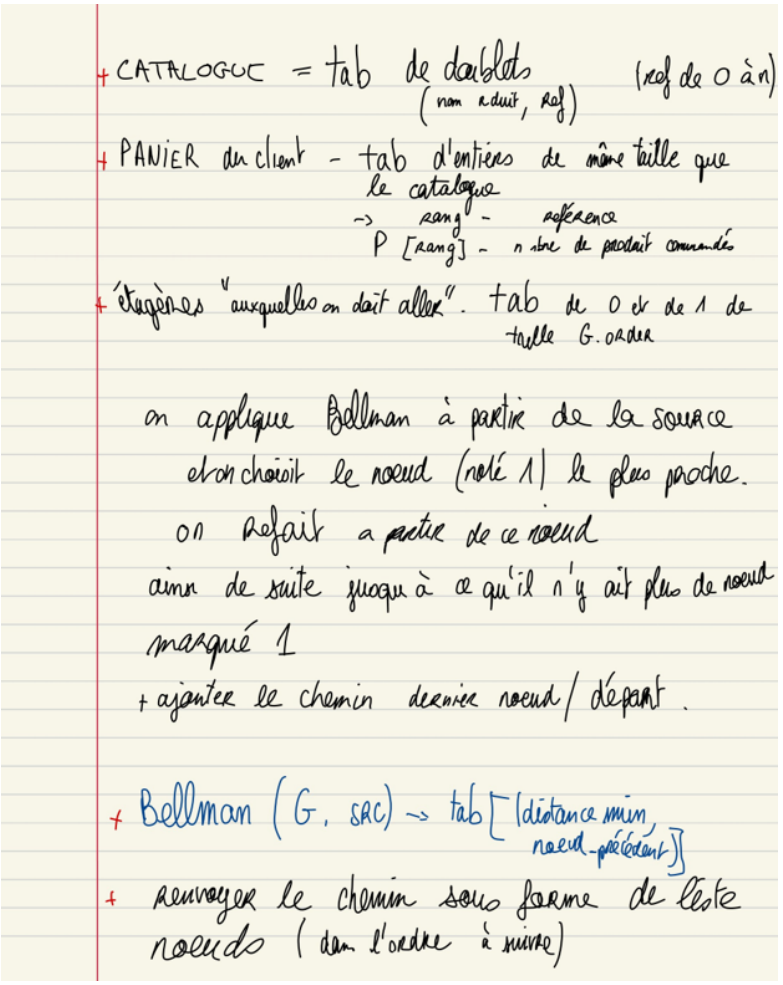
| | | | | |
|---|---|---|---|---|
| première soutenance | | | | |
| Algorithme plus court chemin | X | | X | |
| Interface d'ajout de référence | | X | | X |
| Création d'un entrepôt type | | X | | |
| Algorithme de recherche de chemin selon le poids | X | | X | |
| Création design définitif site | X | | X | |
| Codage du design définitif du site | X | | | |
| Débogage avant la seconde soutenance | | X | | X |
| Localisation "en temps réel" grâce à une simulation de triangulation bluetooth | X | X | X | X |
| Détection égarement employé | X | X | | |
| Préparation seconde soutenance | X | X | X | X |
| Correcteur d'itinéraire | | | X | X |
| Création du parser pour implémenter la liste de produit à aller chercher | | X | | X |
| Générateur aléatoire de commandes | X | | X | |

| | | | | |
|---|---|---|---|---|
| Tests de la version “finie” | X | X | X | X |
| Amélioration du design final | X | X | X | X |
| Débogage avant la dernière soutenance- | X | X | X | X |
| Préparation dernière soutenance | X | X | X | X |

# IV. Compte rendu individuel d'avancement

## A. Thomas:

Étant donné les imprévus du projet, j'ai dû modifier notre fiche de route en nous concentrant sur plusieurs objectifs intermédiaires, tels que la récupération du catalogue des produits, la création d'un tableau de liste de courses du client etc. Voici ci-contre la feuille de route que nous avons pu suivre pour préparer ce 2ème rapport.




Nous avons tous collaboré dans le cadre de ce projet et avons réparti les tâches entre nous. Mon rôle a été de mettre en place l'algorithme de Bellman pour analyser les distances les plus courtes entre un nœud source et les autres nœuds du graphe.

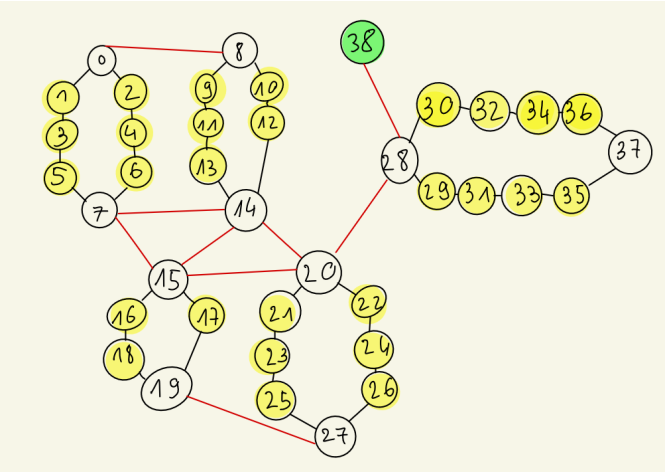
Tout d'abord, j'ai créé une **structure appelée eltBellman** qui stocke la **distance la plus courte depuis la source** ainsi que le **nœud précédent** correspondant. La présence de ce nœud précédent s'est avérée essentielle pour déterminer le chemin le plus court ultérieurement. J'ai ensuite implémenté l'algorithme qui permet de renvoyer un **tableau d'eltBellman de taille G.order**, en se basant sur le graphe et le nœud source fournis en entrée. Voici un aperçu de cette fonction :

```
/ Algorithme de Bellman retourne le tableau des distances et des prédécesseurs
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)poids(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++;
	}
}
```

Schéma du graph utilisé lors de nos test :




Et voici le résultat de notre print en prenant le graphe de gauche en entrée :

```
print bellman(38) :
node0: distance=17, precedent:8
node1: distance=16, precedent:3
node2: distance=16, precedent:4
node3: distance=15, precedent:5
node4: distance=15, precedent:6
node5: distance=14, precedent:7
node6: distance=14, precedent:7
node7: distance=13, precedent:14
node8: distance=13, precedent:10
node9: distance=12, precedent:11
node10: distance=12, precedent:12
node11: distance=11, precedent:13
node12: distance=11, precedent:14
node13: distance=10, precedent:14
node14: distance=9, precedent:20
node15: distance=11, precedent:20
node16: distance=12, precedent:15
node17: distance=12, precedent:15
node18: distance=13, precedent:16
node19: distance=14, precedent:18
node20: distance=7, precedent:28
node21: distance=8, precedent:20
node22: distance=8, precedent:20
node23: distance=9, precedent:21
node24: distance=9, precedent:22
node25: distance=10, precedent:23
node26: distance=10, precedent:24
node27: distance=11, precedent:26
node28: distance=3, precedent:38
node29: distance=4, precedent:28
node30: distance=4, precedent:28
node31: distance=5, precedent:29
node32: distance=5, precedent:30
node33: distance=6, precedent:31
node34: distance=6, precedent:32
node35: distance=7, precedent:33
node36: distance=7, precedent:34
node37: distance=8, precedent:36
node38: distance=0, precedent:-1
```

Après avoir vérifié le bon fonctionnement de l'algorithme, notamment la partie concernant le "**nœud précédent**", j'ai pu développer la fonction "**pathToTarget**". Cette dernière permet de retourner le **chemin le plus court entre deux nœuds** et est structurée de la manière suivante :

```
void pathToTarget(struct eltBellman tabBellman[], int sourceNode,
int targetNode, int pathToTarget[], int *nbPathNodes)
{
	int node=targetNode;
	int i=0;
	int pathToTargetInverse[512];
	*nbPathNodes=0;

	while (tabBellman[node].precedentNode!=-1)
	{
		pathToTargetInverse[i]=node;
		i++;
		*nbPathNodes=*nbPathNodes+1;
		node=tabBellman[node].precedentNode;
	}
	if (node!=sourceNode)
		printf("Error sourceNode in pathToNode\n");
	else
	{
		pathToTargetInverse[i]=node;
		*nbPathNodes=*nbPathNodes+1;

		/* Inversion du path */
		i=0;
		while (i<*nbPathNodes)
		{
			int node=*nbPathNodes-1-i;
			pathToTarget[node]=pathToTargetInverse[i];
			i++;
		}
	}
}
```

voici notre résultat sur le graphe lorsque l'on veut afficher le **chemin le plus court** entre le noeud de départ (**38** ici) et **12** :

```
path->Id38[-1,-1](-1,-1,-1)->Id28[5,-1](-1,-1,-1)->Id20[4,-1](-1,-1,-1)->Id14[2,-1](-1,-1,-1)->Id12[2,4](13,23,-1)
```

ce qui nous intéresse principalement sont les nombres après les "Id", on comprend donc que le chemin effectué est : **38 -> 28 -> 20 -> 14 -> 12.**

## B. Jean Loup

### Le catalogue :

Ci dessous le résultat de **void print_catalogue :**

```
Catalogue :

+------+----------------------------+------------+--------+--------+------------+
| Ref. | Nom                        |   Prix     |  Poids | Volume | Resistance |
+------+----------------------------+------------+--------+--------+------------+
|    0 | lait                       |       1.99 |   1.50 |   0.60 |       0.00 |
|    1 | fromage                    |       2.49 |   0.20 |   0.10 |       0.00 |
|    2 | yaourt                     |       0.99 |   0.20 |   0.10 |       0.00 |
|    3 | beurre                     |       1.89 |   0.25 |   0.20 |       0.00 |
|    4 | oeufs                      |       2.99 |   0.40 |   0.40 |       1.00 |
|    5 | pain                       |       1.29 |   0.50 |   0.50 |       0.00 |
|    6 | baguette                   |       0.99 |   0.40 |   0.40 |       0.00 |
...
|   66 | echalote                   |       0.99 |   0.20 |   0.20 |       0.00 |
|   67 | pomme_de_terre             |       0.99 |   0.20 |   0.20 |       0.00 |
|   68 | patate_douce               |       0.99 |   0.20 |   0.20 |       0.00 |
+------+----------------------------+------------+--------+--------+------------+
```

Le **catalogue** correspond à une **liste d'articles (struct Article)** extraite d'un fichier texte :

fichier texte (catalogue.txt) :

```
0 lait 1.99 1.5 0.6 0
1 fromage 2.49 0.2 0.1 0
2 yaourt 0.99 0.2 0.1 0
3 beurre 1.89 0.25 0.2 0
4 oeufs 2.99 0.4 0.4 1
5 pain 1.29 0.5 0.5 0
6 baguette 0.99 0.4 0.4 0
7 croissant 0.79 0.1 0.1 0
…
68 patate_douce 0.99 0.2 0.2 0
```

“**struct 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
    float volume;      // Volume de l'article
    float resistance;  // Résistance de l'article
} Article;
```

## Le panier :

Le panier représente la liste des nœuds que l’on doit visiter. Il s’agit d’une liste d’entier dont la taille est égale au nombre d’articles dans le catalogue.

```
int panier[69];
for (int i = 0; i < 69; i++) {
     panier[i] = -1;
}
panier[0] = 1; //  1 lait
panier[5] = 1; //  1 pain
panier[12] = 1; //  1 the_en_sachets
panier[25] = 1; //  1 haricots_blancs
panier[45] = 1;  //  1 oeufs_de_caille
panier[55] = 1; //  1 raisin
```

```
panier[61] = 1; //  1 carotte
panier[64] = 1; //  1 oignon
panier[68] = 1; //  1 patate_douce
int nbart = 9;
```

Ici, il y a 9 produits dans le panier. Le rang correspond à la référence du produit et la valeur correspond à la quantité du produit.

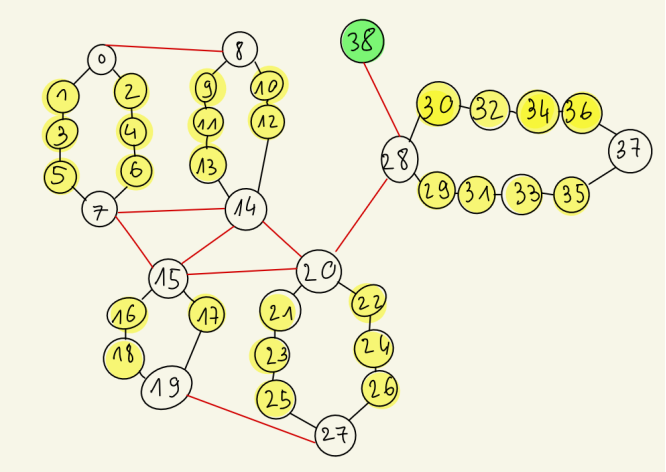
Ci dessous le résultat de la fonction **void print_panier(panier)** :

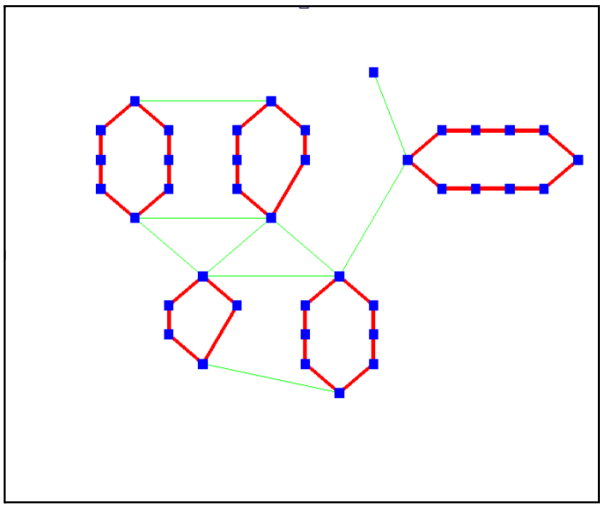
```
Print panier :
+------+----------------------------+-----------+-------------+
| ref  | Article                    | Quantité  | N° de noeud |
+------+----------------------------+-----------+-------------+
| 0    | lait                       | 1         | 30          |
| 5    | pain                       | 1         | 3           |
| 12   | the_en_sachets             | 1         | 5           |
| 25   | haricots_blancs            | 1         | 13          |
| 45   | oeufs_de_caille            | 1         | 26          |
| 55   | raisin                     | 1         | 32          |
| 61   | carotte                    | 1         | 34          |
| 64   | oignon                     | 1         | 35          |
| 68   | patate_douce               | 1         | 36          |
+------+----------------------------+-----------+-------------+
```

Comme nous pouvons le voir ci-dessus, à partir de la **référence** d'un produit il est possible de le retrouver dans le catalogue ainsi que dans les nœuds du graph. Nous avons ainsi **toutes les informations sur un produit** en indiquant seulement sa référence.

# Plus court chemin

Schéma d'une réserve sous forme de graph :



En utilisant l'algorithme implémenté par Thomas (Bellman), nous pouvons déterminer le plus court chemin à utiliser .
Pour cela, nous appliquons donc Bellman avec :

- point de départ 38 (choix arbitraire, correspond au noeud de l'entrée de la réserve)
- point d'arrivée : 30, le nœud le plus proche (déterminé par l'algorithme de Bellman) contenant un produit du panier. En effet, comme nous pouvons le voir ci dessus, 30 est très proche de l'entrée et contient bien le lait qui est dans le panier.
- On prend ensuite 30 comme point de départ et on applique Bellman en ce point
- Ainsi de suite jusqu'à avoir parcouru tous les nœuds du panier.

On a donc le résultat de **void print_path** qui montre le chemin optimal à parcourir pour récupérer tous les articles du panier :

| N° de noeud | Article | ref |
|---|---|---|
| 38 | | |
| 28 | | |
| 30 | lait | 0 |
| 32 | raisin | 55 |
| 34 | carotte | 61 |
| 36 | patate_douce | 68 |
| 37 | | |
| 35 | oignon | 64 |
| 33 | | |
| 31 | | |
| 29 | | |
| 28 | | |
| 20 | | |
| 14 | | |
| 13 | haricots_blancs | 25 |
| 14 | | |
| 7 | | |
| 5 | the_en_sachets | 12 |
| 3 | pain | 5 |
| 5 | the_en_sachets | 12 |
| 7 | | |
| 15 | | |
| 20 | | |
| 22 | | |
| 24 | | |
| 26 | oeufs_de_caille | 45 |

## Implémentation graphique :

Suite à de nombreux problèmes rencontrés avec les bibliothèques graphiques nous avons fini par choisir openGL, étant moins répandue mais plus simple d'utilisation.

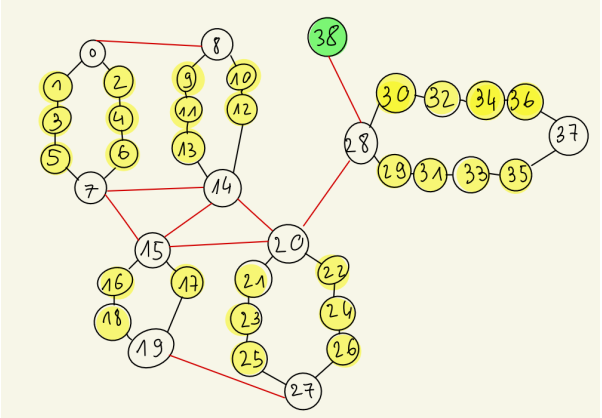
Etant donné que chaque noeud du graph possède des coordonnées dans l'espace :

```
struct Node
{
    int nodeID;
    int boolShelf;
    int aisle;
    int shelf;
    int boolTopBottom;
    struct Node *adjTab[10];
    int nbAdj;
    int X;                  //Coordonnée X
    int Y;                  //Coordonnée Y
    int productsTab[3];
    int nbProducts;
};
```

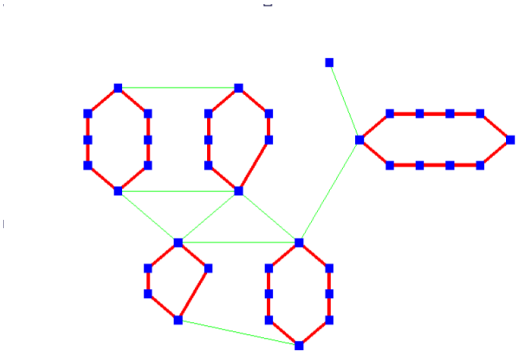
```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
```

Il est donc possible de représenter plutôt facilement la réserve dans l'espace.
Il suffit de placer un carré à l'emplacement des nœuds et les relier par une droite s' ils sont liés.

Graph attendu :



Graph obtenu à l'aide de openGL :



# c) Erwan

## QR codes et localisation:

Comme noté dans la liste de tâches nous avions pour projet de proposer une localisation de l'employé à l'aide de deux choses : Du bluetooth/wifi pour trianguler sa position dans l'entrepôt, et des QR codes qui permettent en scannant la référence de garder le fil de la commande ainsi que la position de l'employé grâce à celle de l'article scanné. J'ai donc commencé par la génération de ces QR codes dont voici quelques exemples:

Il ne restait "plus" que la partie reconnaissance du QR code, le souci étant qu'un OCR est une tâche complexe, c'était d'ailleurs notamment l'entièreté du projet de s3, en l'état il n'y a donc que les QRcode déjà générés. Notons également qu'actuellement la génération des QR codes est faite grâce à une librairie.

Exemple de la récupération d'une référence (et des coordonnées grâce à un QR code.

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

```
//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 au vu de la complexité de la tâche il est probable que cela ne soit relégué qu'au second plan (après que les autres fonctionnalités aient été implémentée testées et retestées)

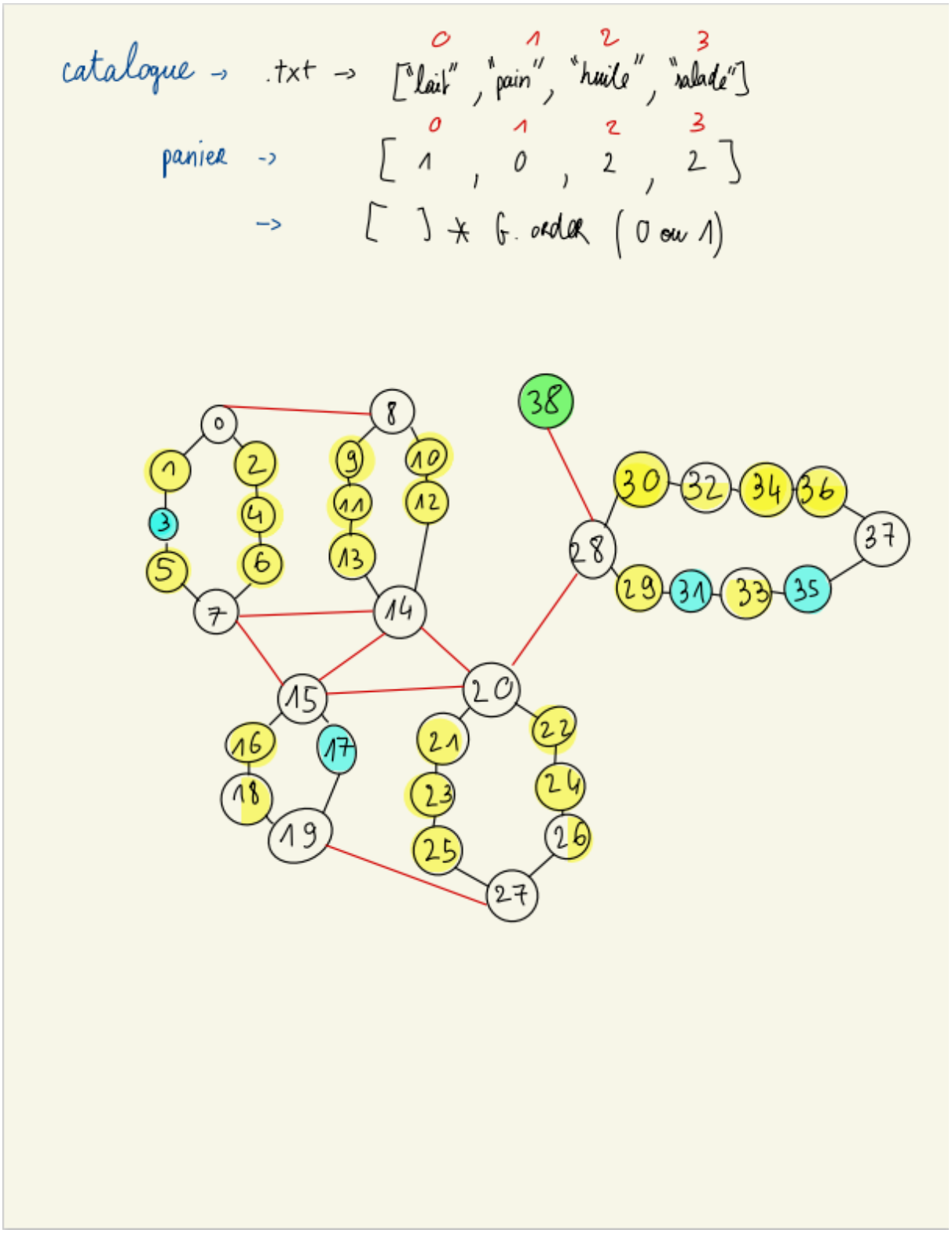
Contribution au cahier des charges :

La seule autre contribution effective de ma part est celle à ce cahier des charges notamment en termes de mise en page et de rédaction des parties communes.

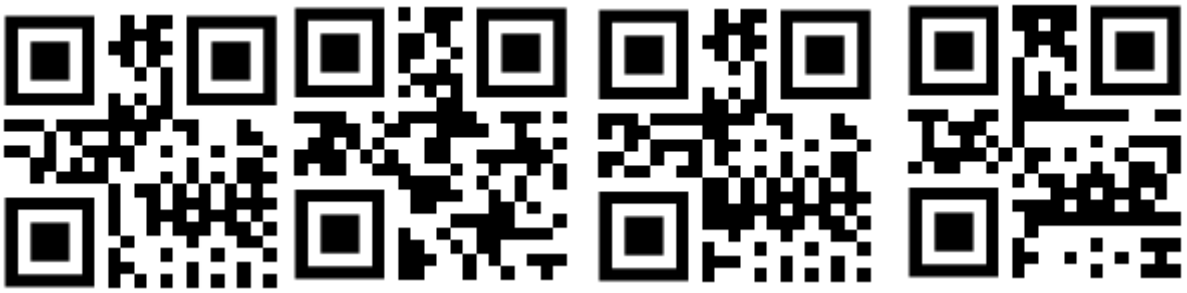
# V. Conclusion

L'application de recherche du chemin le plus optimisé pour les employés en préparation de commande est un projet visant à améliorer la productivité et à optimiser le travail des employés dans l'entrepôt. Ce rapport numéro 2 témoigne de l’avancement du projet et des défis qu'il nous reste à surmonter si nous voulons nous tenir à ce qui est dit dans le cahier des charges, il est important de noter que nous sommes globalement dans les délai ce qui est une bonne chose à ce stade d’avancement du projet

# VI. Annexes:


catalogue -> .txt -> ["lait", "pain", "huile", "salade"]
0 1 2 3
panier -> [1, 0, 2, 2]
0 1 2 3
-> [ ] * 6. order (0 ou 1)

Exemple de QRcode:



Rappel du site : https://perfectcourse.neocities.org/