GROUPE 2

Penot Paul // Bories Thomas

CIR 2

ISEN Lille

Institut Supérieur de l'électronique et du Numérique

PROJET PLURIDISCIPLINAIRE

PROJET WEB

Professeur référent : Mme Zgaya M. Soulignac

Table des matières

[1-Introduction 3](#_Toc262478351)

[1.1-Projet 3](#_Toc262478352)

[1.2-plan du rapport 3](#_Toc262478353)

[1.3-Organisation 3](#_Toc262478354)

[2-Création Web 4](#_Toc262478355)

[2.1-Le type Graph 4](#_Toc262478356)

[2.1.1-Introduction 4](#_Toc262478357)

[2.1.2-Conception 4](#_Toc262478358)

[diagramme de classe 4](#_Toc262478359)

[2.1.3-Implémentation 4](#_Toc262478360)

[2.1.4-test du type Graph 4](#_Toc262478361)

[2.1.5-Conclusion 4](#_Toc262478362)

[2.2-Interface WEB 4](#_Toc262478363)

[2.2.1-Introduction 4](#_Toc262478364)

[2.2.2-Spécification 4](#_Toc262478365)

[2.2.3-Présentation de l'interface 4](#_Toc262478366)

[2.2.4-Conception 4](#_Toc262478367)

[2.2.5-Implémentation 4](#_Toc262478368)

[2.2.6-Conclusion 4](#_Toc262478369)

[3-Base de données 4](#_Toc262478370)

[3.1-Introduction 4](#_Toc262478371)

[3.2-Conception 4](#_Toc262478372)

[3.3-Implémentation 4](#_Toc262478373)

[3.3.1-sauvegarde 4](#_Toc262478374)

[3.3.1-Chargement 4](#_Toc262478375)

[3.4-Conclusion 4](#_Toc262478376)

# 1-Introduction

## 1.1-Projet

Dans le cadre de notre deuxième année à l'ISEN, nous devons réaliser un projet pluridisciplinaire par binôme. Celui-ci devra proposer à l'utilisateur une application permettant de manipuler des graphes.

Pour mener à bien ce projet nous allons en parallèle développer pour deux technologies distinctes :

Une qui interviendra pour l'utilisateur par l'intermédiaire d'une interface web, lui permettant de manipuler visuellement des graphes et d'y appliquer des algorithmes spécifiques au graphe qu'il aura créé tel que celui de Welsh-Powell, Dijkstra, Little.

L'exécution des algorithmes se fera en arrière plan par l'intermédiaire d'un serveur distant sur lequel sera implémenté en C++ pour optimiser le rendu des calculs.

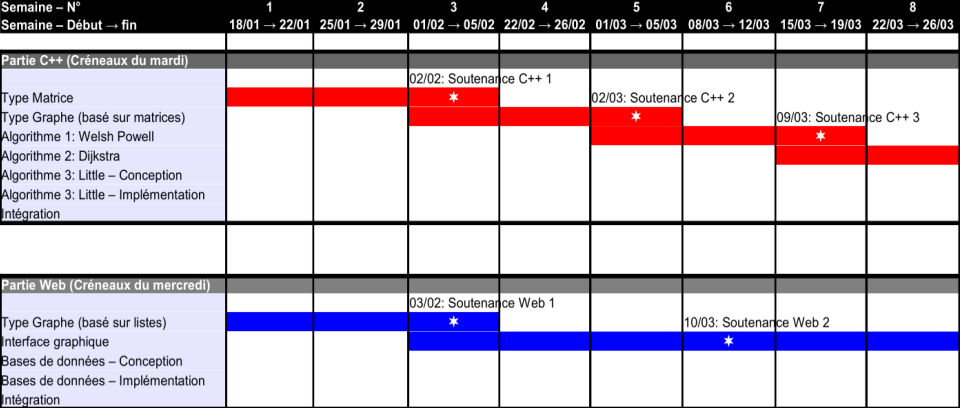
Une base de donnée nous permettra aussi de sauvegarder et charger des graphes par l'intermédiaire de l'interface web.

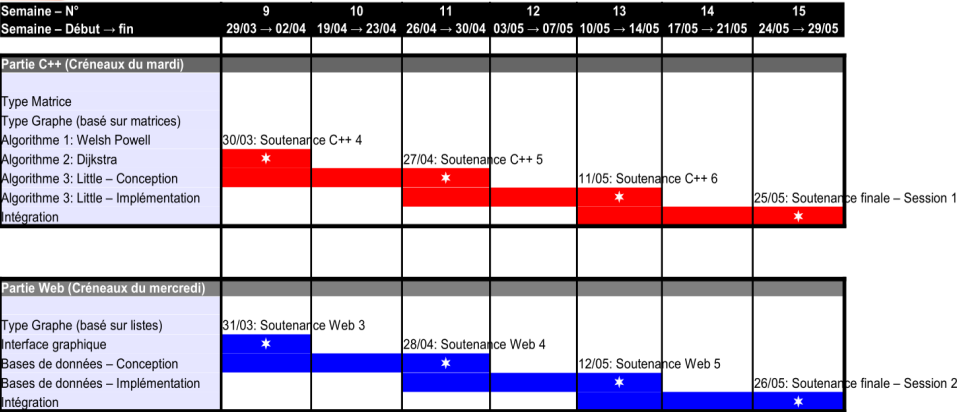
## 1.2-plan du rapport

Le rapport s'organisera en sous-partie résumant les différente étape du projet WEB il doit permettre de suivre l'évolution du projet, chacune des sous partie donnera les objectifs à réaliser pour chaque soutenances, notre choix de conception, implémentation, conclusion et divers éléments permettant la description approfondie du travail fournie.

## 1.3-Organisation

La réalisation de ce projet s'étend sur 7 étapes qui doivent être atteinte a intervalle régulier pour nous permettre de ne pas prendre de retard et nous mener jusqu’à un projet fonctionnel respectant les spécifications données.





# 2-Création Web

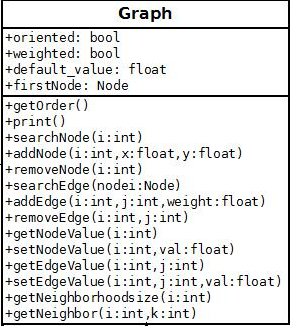
## 2.1-Le type Graph

### 2.1.1-Introduction

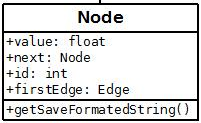
Cette première partie du projet consiste à implémenter le type graphe basée sur les listes chaînées pour ensuite être utilisée dans l'interface utilisateur. Il faut trouver les différentes structure nécessaire à construire pour modéliser le type graphe à partir d'une liste et créer les différentes fonctions qui seront utile pour manipuler le graphe, nous n'avons pour l'instant pas à créer les fonction et classe permettant la mise en place d'une interface graphique. Ce type graph est essentiel à la conception futur de notre projet, c'est sur celle-ci que se basera toute notre interface.

### 2.1.2-Conception

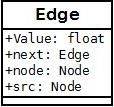
Pour la représentation en mémoire des matrices nous allons utiliser la méthode des listes chaînées, le langage JavaScript utilisé pour représenter le graphe ne permettant pas de créer des pointeurs nous avons utilisé la puissance du langage-objet fournie par celui-ci. Pour modéliser cette structure nous allons créés 3 classes.

 Une classe Graph qui nous permettra de créer un graphe en détaillant toutes ces propriétés, orienté, pondéré ou encore son nœud de départ. La classe graph à donc besoin des deux classes Node et Edge pour fonctionner, elle prendra en paramètre deux boolean dictant son état orienté et pondéré.

Elle contiendra aussi comme élément important le firstEdge qui sera le premier nœud nous permettant d'accéder à la classe Node.

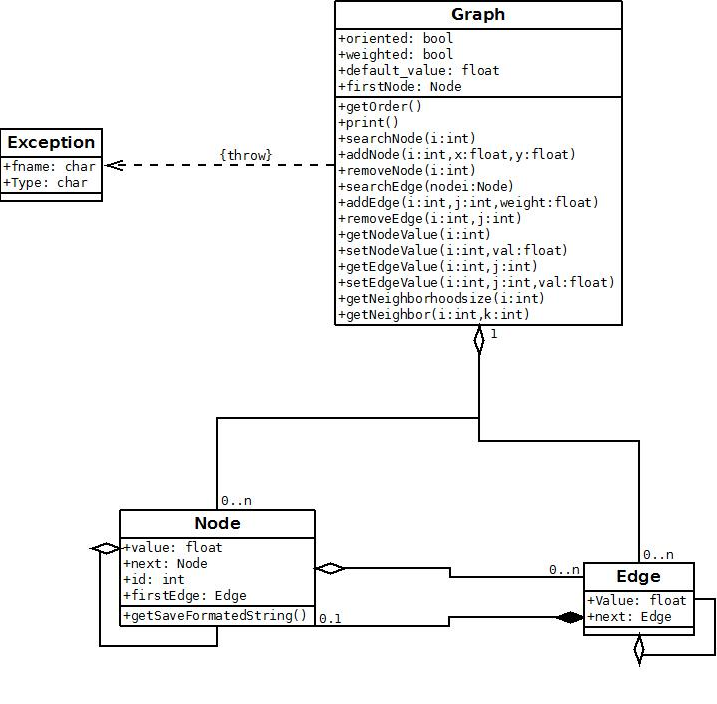
La classe Node pour nous permettre de construire en mémoire et graphiquement les nœuds du graph elle doit contenir pour répondre à nos attentes, son poids, sont id et sa première arrête et son nœud suivant s'il existe.

Pour simuler une liste chaînée nous ajoutons nextNode et first Edge pour accéder aux première arc du nœud.

La classe Edges représentant les arcs prenant comme paramètres un nœud de départ (src) et d'arriver (node), cette classe à une structure similaire a la classe Node.

Enfin pour continuer dans notre logique de listes chaînées il existe ici next pour créer la liste des arcs d'un nœud.

# diagramme de classe



### 2.1.3-Implémentation

L'implémentation nous permet de créer des graphes à partir de listes chaînées que nous avons recréé et de pouvoir grâce au fonction membre de graphe de parcours ce graphe pour ensuite le manipuler lors de sa création par l'interface graphique.

### 

### 2.1.4-test du type Graph

Pour tester notre type Graph, nous créons une batterie de tests pour vérifier la création et le bon établissement de tous les éléments :

Nous créons plusieurs graphes avec des ids continue et discontinue nous testons aussi l'ajout de nœud pour un id déjà existant.

Nous pouvons ensuite tester les fonctions liées aux arcs permettant leurs ajouts, nous ajoutons des arcs entre nœuds discontinue, sur deux nœud déjà relié ou nous recevons une exception pour nous prévenir que l'arc existe déjà.

Nous supprimons ensuite les nœud et arc en vérifiant que la vérification prend en compte la suppression d'élément n'existant pas et enfin nous terminons par supprimer un graph qui supprime en cascade touts les nœuds et touts les arcs.

### 2.1.5-Conclusion

L'implémentation du type graphe nous à permit d'utiliser nos connaissances dans le langage javascript et les possibilités objet qu'il nous propose pour créer nos classes, sans lesquelles il aurait été compliqué de construire notre classe graphe basée sur le principe des listes chaînées.

## 2.2-Interface WEB

### 2.2.1-Introduction

La finalité de cette étape est d'implémenter une interface graphique capable de créer, d'afficher un graphe et de réagir aux modifications apportées à celui-ci que ce soit par des ajouts, suppressions, mouvements ou modifications d'éléments dans le graphe ou encore à une application d'algorithme.

Il permettra à l'utilisateur de créer lui-même un graphe, de le sauvegarder et de le charger à partir d'une base de données construite plus tard. Il proposera un panel d'algorithmes (Welsh-Powell, Dijkstra, Little) applicables au graphe et un cadre affichant des informations concernant le graphe.

Cette interface devra respecter des spécifications strictes pour assurer un fonctionnement minimal et pouvoir continuer le projet.

Nous préciserons les détails de ces spécifications pour ensuite détailler nos choix de conception puis notre implémentation.

### 2.2.2-Spécification

Notre implémentation devra respecter certaines spécifications, elle devra permettre :

-La création de nœuds et d'arcs entre ceux-ci.

-Suppression d'un nœud avec les arcs le reliant à ses voisins et la suppression d'arcs.

-Identifier les nœuds sous la forme d'un label.

-D'afficher la valeur du nœud sous la forme d'une couleur.

-D'afficher la valeur des arcs sous la forme d'un label.

-D'avoir 3 modes de fonctionnement (sélection, ajout de nœud, ajout d'arc).

-De réinitialiser le graphe dans tous les modes.

-De sélectionner un arc ou un nœud en le mettant en valeur et afficher ses propriétés dans une zone texte.

-Informer l'utilisateur en cas de non-validité d'un élément modifié en précisant la nature du problème.

Nous devrons aussi faire en sorte que l'interface puisse afficher les informations permettant d'interpréter les algorithmes appliqués au graphe (flèches sur les arcs, coloration des nœuds, cadre affichant les informations relatives à l'algorithme). Nous devons penser aussi aux éléments graphique permettant la sauvegarde et le chargement de graphe à partir d'une base de données.

### 2.2.3-Présentation de l'interface

Nous avons mis du temps a nous décider pour un choix d'interface étant bien obliger de faire un choix nous nous somme arrêté sur une présentation de type enfantine . Celle-ci se décompose en élément simple qui sont

**-Un menu.**

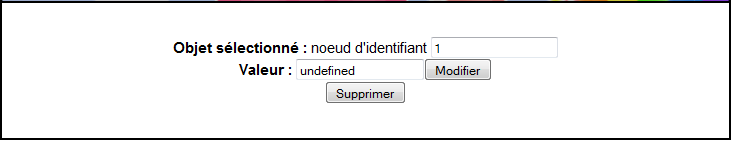
**-une zone de statue.**

**-une fenêtre de dessin.**

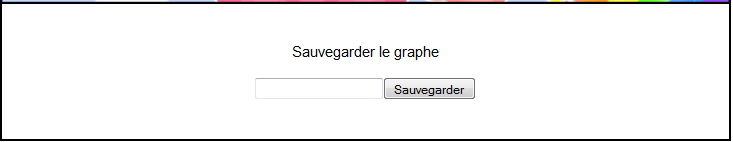


Le menu nous permettra d'effectuer différente action sur le graphe, et en fonction de l'option prit, la fenêtre de statut changera pour s'adapter à la sélection de l'utilisateur.

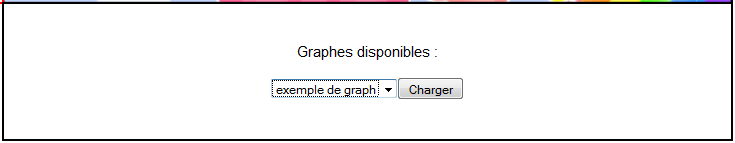
sélectionner:

 Un statut apparaît pour afficher l'objet sélectionné et sa valeur. Il est possible de modifier la valeur de l'objet en cliquant sur modifier.

sauvegarder:

L'utilisateur peut sauvegarder son graphe sous le nom qu'il aura préalablement indiqué dans le champ libre.

charger:

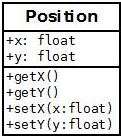
La liste déroulante est automatiquement mis à jour grace à AJAX et l'utilisateur n'a plus qu'à choisir dans le menu déroulant le graphe qu'il veut choisir et cliquer sûr charger.

### 2.2.4-Conception

**Classes**

Lors de l'implémentation de la partie graphique de ce projet, nous avons été amenés à ajouter un certain nombre de classes à notre programme. Tout d'abord car cela nous a permis d'appliquer les concepts vus en cours, et aussi car cela simplifiait grandement l'utilisation de l'interface. Nous avons de plus modifié les classes **Node** et **Edge** existantes afin de faciliter leur manipulation.

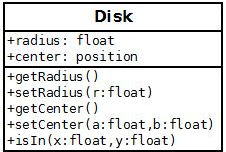
#### 2.2.4.1-Position



Cette classe représente une position dans un plan. Elle possède les propriétés x et y, et prend ces paramètres lors de la construction. Ils représentent les coordonnées du curseur dans la page et donc dans le canvas.

Les seules méthodes de la classe sont les accesseurs aux membres x et y. Cette classe nous permettra a tout moment de connaitre la position d'un curseur. pour positionner nos éléments .

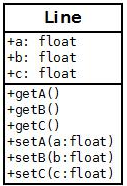
#### 2.2.4.2-Disk

 La classe **Disk** représente un disque dans l'espace, de centre (a,b) et de rayon r. Le constructeur prend donc ces différents paramètres.

Elle possède les propriétés radius et center. Center est une position, c'est le centre du disque, radius est son rayon.

Les méthodes de **Disk** sont les accesseurs permettant d'accéder au centre et au rayon, ainsi qu'une fonction (isIn) qui prend des coordonnées x et y en paramètre, et renvoie true si le point appartient au disque, false sinon.

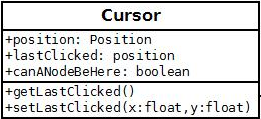
#### 2.2.4.3-Line

 Elle représente une droite dans le plan. Elle est caractérisée par une équation cartésienne de la forme ax+by+c=0. Son constructeur prend les coordonnées (x,y) de deux points dans le plan et calcule une équation de la droite, en résolvant de simples équations avec le vecteur directeur et un des deux points.

Ses propriétés sont a, b et c, les coefficients de l'équation cartésienne de la droite.

Ses méthodes sont des accesseurs pour ces différentes propriétés.

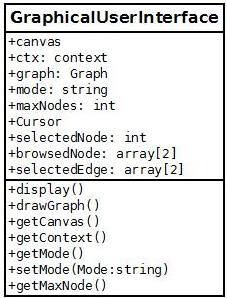
#### 2.2.4.4-Cursor

 **Cursor** représente le curseur de la souris. Il hérite de **Position**, donc des propriétés x et y, la position du curseur actuelle dans le canvas.

Elle possède également les propriétés lastClicked et canANodeBeHere. LastClicked est une position. Elle sert à stocker la position du dernier point cliqué. CanANodeBeHere est un entier utilisé pour savoir si un noeud peut être ajouté à l'endroit où pointe le curseur. Il vaut 0 si oui, et une valeur positive sinon.

Ses méthodes sont des accesseurs.

#### 2.2.4.5-GraphicalUserInterface

C'est la classe qui va modéliser l'interface graphique du projet.

Propriétés :

- canvas : le canvas de la page, où sera dessiné le graphe

- ctx : le contexte du canvas, qui est notre zone d'affichage nous le choisissons en 2D cela sera suffisant pour l'affichage de graphe.

- graph (Graph) : le graphe dessiné à l'écran

- mode (string): le mode de fonctionnement en cours. Il vaut 'addNode' pour l'ajout d'un nœud , 'addEdge' pour l'ajout d'arcs, et 'select' pour la sélection d'un élément. Nous aurions pu choisir un entier pour stocker le mode mais avoir une chaîne nous semble plus clair lors de la compréhension du mode.

- maxNodes (number): stocke l'id maximal du graphe. Il est utilisé lors de l'ajout d'un nœud.

- cursor (Cursor): le curseur de la souris

- selectedNode (number): l'id du noeud actuellement sélectionné

- browsedNode (Array[2]): tableau de deux cases dont la première contient l'id du nœud actuellement survolé par le curseur, et la deuxième un booléen qui vaut true si au moins un nœud est survolé (mis à jour lors de l'affichage du graphe), false sinon.

- selectedEdge (Array[2]): tableau de deux cases contenant les id des nœuds reliés par l'arc

Méthodes :

- display : dessine l'état actuel de l'interface graphique. Elle fait appel à la méthode drawGraph. Lorsque l'interface est en mode d'ajout de nœud, elle affiche un nœud vide centré sur la position du curseur. Un système d'anticollision d'écris avec la classe Node, est utilisé, afin de ne pouvoir ajouter un nœud sur un autre ou sur les bords du canvas.

- drawGraph :dessine le graphe en mémoire à l'écran. Elle parcourt les nœuds et leurs arcs un par un et fait appel à leurs méthodes display.

Accesseurs :

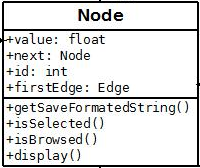
- getCanvas : retourne le canvas

- getContext : retourne le contexte du canvas

- get/setMode : retourne/modifie le mode de fonctionnement

- getMaxNodes : retourne l'id maximal du graphe

#### 2.2.4.6-Node

 Nous avons donc apporté des modifications à la classe Node précédemment créé afin de faciliter l'utilisation de l'interface graphique.

Node hérite maintenant de Disk. Il a donc un centre et un rayon, lequel est défini grâce à la variable globale NODE\_RADIUS.

Nous avons rajouté les méthodes isSelected, isBrowsed et display. IsSelected renvoie true sur le noeud est sélectionné, c'est-à-dire que le dernier point cliqué par le curseur appartient au disque représentant le noeud.

IsBrowsed renvoie true sur la position actuelle du curseur appartient au disque, false sinon.

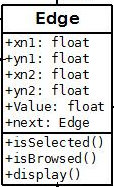
Display permet de dessiner le noeud à l'écran.

Ainsi, chaque noeud sait maintenant où il se trouve dans le canvas, s'il est sélectionné ou survolé, et peut se dessiner lui-même.

L'anticollision est appliquée dans la méthode display de la classe Node. Il crée un disque temporaire de centre le centre du noeud, et de rayon deux fois plus grand. Si la position du curseur actuelle appartient à ce disque, cela signifie qu'en rajoutant un noeud à cet endroit, les deux vont se chevaucher. De plus, elle calcule si l'ajout d'un noeud sur les bords du canvas ne fera pas déborder le noeud. Si c'est le cas, la propriété canANodeBeHere de l'interface est incrémentée.

De cette façon, si canANodeBeHere vaut plus de 0, on ne peut ajouter de nouveau noeud à l'endroit où pointe le curseur.

#### 2.2.4.7-Edge



Pour les mêmes raisons que pour la classe Node, nous avons rajouté à Edge des propriétés et des méthodes. Edge hérite à présent de Line.

De plus, edge possède une nouvelle propriété src, une référence sur le noeud source. Cela rend le dessin de l'arc plus aisé, connaissant les coordonnées des deux points (ie les centres des noeuds) à relier.

Tout comme node, edge possède les méthodes isSelected, isBrowsed et display. IsSelected renvoie true si la distance du dernier point cliqué par rapport à l'arc (c'est-à-dire à la droite) est inférieure ou égale à 10, false sinon.

IsBrowsed renvoie true si la distance du curseur par rapport à l'arc est inférieure ou égale à 10.Display permet à chaque arc de se dessiner tout seul.

La présence des même nouvelles méthodes dans Node et Edge nous a amené à penser à créer une nouvelle classe, qui représentera un élément graphique. Cependant, faute de temps, nous ne l'avons pas implémentée.

### 2.2.5-Implémentation

Pour permettre l'interaction de tous les éléments entre eux et un appel correct des fonctions nous utilisons la puissance du JavaScript pour détecter et repérer les actions dans notre page. Le fichier interface.js contient les fonctions en lien direct avec l'interface utilisateur.

Lors de l'arriver sur la page, la premier fonction à être appelé est la fonction main qui initialise la page en créant une nouvelle interface d'utilisateur (GraphicalUserInterface) ( , et la stock dans une variable GUI, il contient une instance de Graph nous permettant de garder le Graphe en mémoire. Nous initialisons aussi le mode à ajout d'un nœud.

Pour pouvoir interpréter les clics et déplacements de l'utilisateur dans notre interface nous faisons appel à une batterie d'écouteurs. Nous avons un écouteur sur "clic" qui fera appel à la fonction clickReact pour déterminer dans quel mode est l'utilisateur et le retourner dans une variable globale MODE.une fois le mode connu les écouteurs peuvent réagir selon le mode sélectionné. Pour traiter la position du curseur nous avons 3 écouteurs "mousemove" sur celui-ci permettant de dessiner ou non l'ombre d'un nœud suivant sa position dans la page et bien sûr retourner sa position en temps réel.

Nos écouteurs sur les boutons changent value, del, clear, nous permettre dans l'ordre de changer la valeur d'un élément sélectionné, de supprimer l'élément sélectionné, et de démarrer une nouvelle instance de Graph.

Diagramme de cas d'utilisation

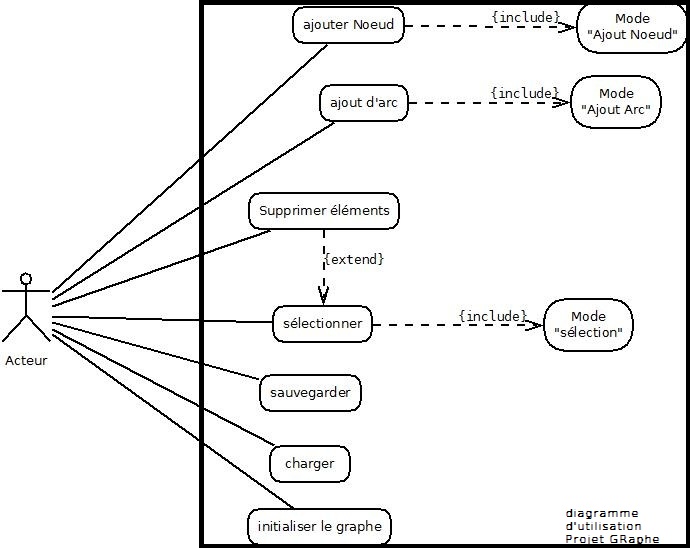
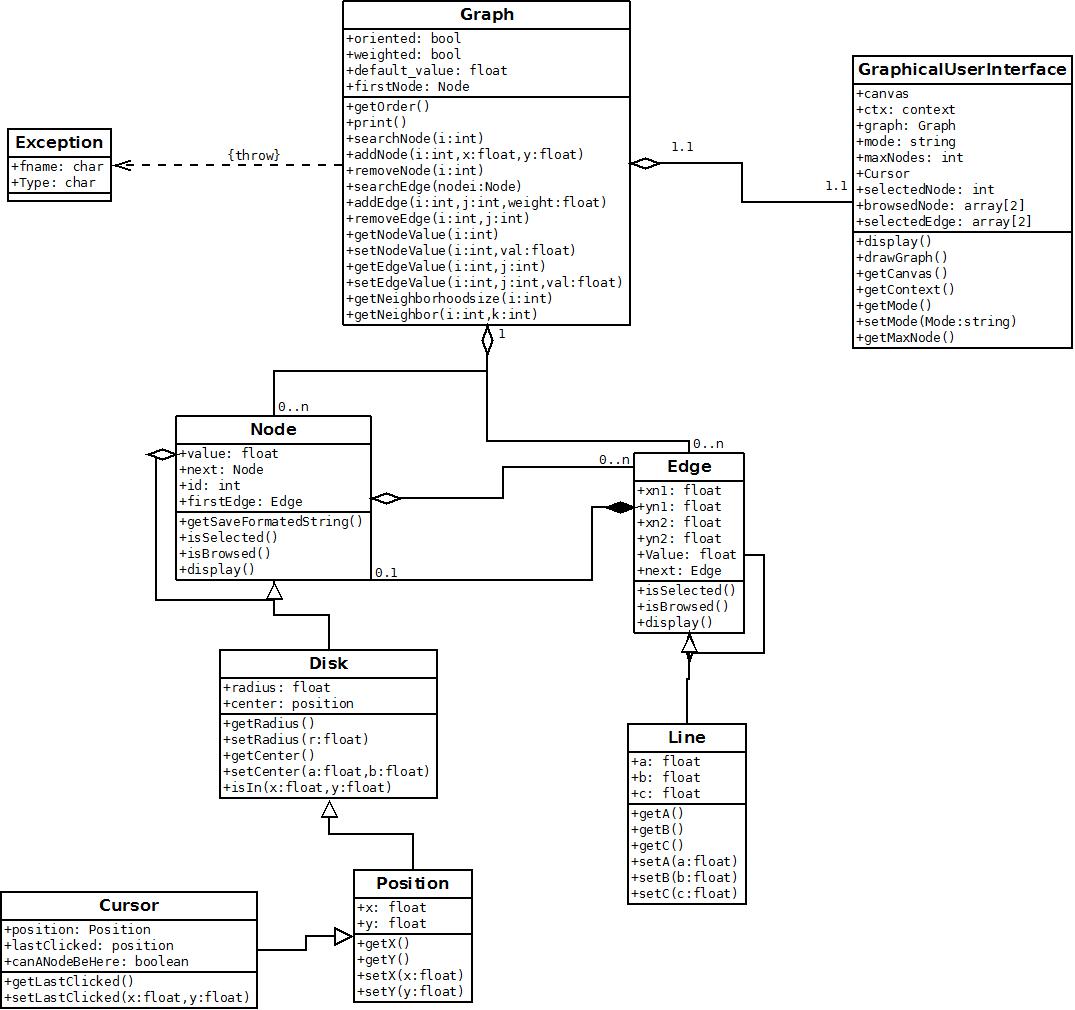


Diagramme UML



### 2.2.6-Conclusion

Nous avons commencé à coder l'interface sans utiliser les concepts Objet, puis, suite à une remarque de M. Soulignac et aux fonctionnalités offertes par le langage javascript nous avons tout repensé en conséquence, d'où la création d'une classe interface graphique qui supprime toute les variable global précédemment utilisées et facilite grandement l'utilisation de l'interface.

# 3-Base de données

## 3.1-Introduction

Après avoir créé une interface Web permettant de créer nos graphes, il devient important de pouvoir les sauvegarder et les charger pour l'utilisateur nous nous intéressons donc maintenant à une partie consistant à la mise en place des différentes structures pour sauvegarder et charger nos graphes sur notre serveur par exemple.

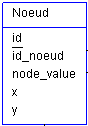
Cette demande nous conduit donc naturellement vers la mise en place d'une base de données avec tous les ajouts que nous allons devoir réaliser pour relier notre BDD à notre interface graphique et cela implique l'utilisation de langages qui n'ont pas était utilisé jusqu'ici mais aussi de nouvelle capacité offerte par le javascript pour nous permettre d'échanger entre ces différent langages.

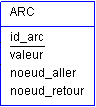
## 3.2-Conception

Pour sauvegarde notre Graphe nous avons besoin de sauvegarder 3 élément principaux avec les propriétés soit:

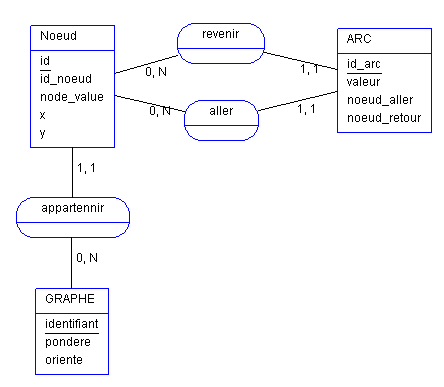
 Une entité GRAPHE contenant Le nom du graph, qui sera choisi par l'utilisateur, et ses conditions de pondération et d'orientation.

Cette entité peut contenir de 0 à n nœud et si elle est détruite les noeud sous-jacent devront être détruit.

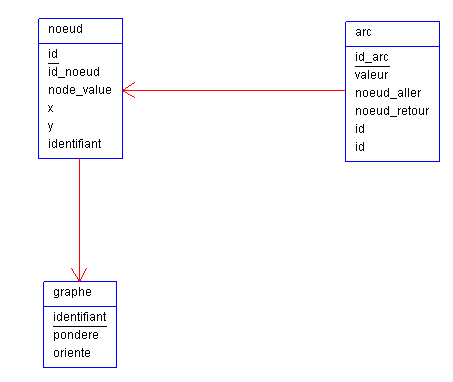
 Une entité Nœud permettant de stocker chacun des nœuds d'un graphe, il faut faire attention à leur stockage car leurs propriétés peuvent entraîner des conflits 2 nœud ayant le même ID peuvent appartenir à 2 graphes différents, nous importons donc les identifiants des graphes dans cette entité. Un nœud peut appartenir à un seul et unique graphe et il peut avoir de 0 à n arc stocker dans une nouvelles entité.

 Une entité Arc permettant de stocker cette fois-ci les arcs de notre graphe. Pour modéliser l'orientation des arcs nous créons deux relations "aller" et "revenir" qui permettront aussi de savoir à quel graphe ils appartiennent en les attachant à des nœuds uniques.

**MCD**



**MLD**



Dans le MLD nous remarquons que les identifiant sont importé dans la table nœud et que arc importe 2 "id" l'un correspondant aux nœud d'aller et l'autre aux nœud d'arriver de l'arc.

## 3.3-Implémentation

Pour implémenter cette base de données nous avons choisi d'utiliser un logiciel déjà vu l'an dernier qui est mySQL query Browser. Il nous permet d'implémenter notre base de données en mySQL, celui-ci pose cependant quelque problème dans la gestion des clés étrangère nous utilisons donc si nécessaire le moteur innoDB pour gérer celle-ci. Le script de création de table parle de lui-même quand aux types des données enregistré.

CREATE TABLE GRAPHE (identifiant VARCHAR(25) NOT NULL PRIMARY KEY,

oriente bool,

pondere bool

)ENGINE=InnoDB;

CREATE TABLE NOEUD (id INT PRIMARY KEY AUTO\_INCREMENT,

id\_noeud INT,

node\_value float,

x INT,

y INT,

identifiant VARCHAR(25) NOT NULL

check(id\_node>0),

FOREIGN KEY (identifiant) REFERENCES GRAPHE(identifiant) ON DELETE CASCADE

)ENGINE=InnoDB;

Ici id représente une valeur auto incrémenter pour pouvoir différencier chaque nœud selon leur graphe c'est cette valeur qui sera utilisée dans la table Arc pour stocker le nœud de départ et d'arriver.

id\_noeud est quand a lui l'id réel du nœud .

CREATE TABLE ARC (id\_arc int PRIMARY KEY AUTO\_INCREMENT,

valeur float,

noeud\_aller int NOT NULL,

noeud\_retour int NOT NULL,

check(aller>0),

check(retour>0),

FOREIGN KEY (noeud\_aller) REFERENCES NOEUD(id) ON DELETE CASCADE,

FOREIGN KEY (noeud\_retour) REFERENCES NOEUD(id) ON DELETE CASCADE

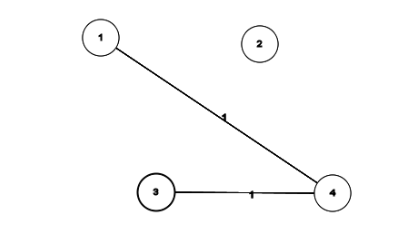
)ENGINE=InnoDB;

Une fois la base de données créé il faut la relier à notre interface WEB pour résoudre ce problème nous utilisons une solution appelé AJAX. Nous allons avoir besoins pour sa d'une page PHP sur laquelle nous allons envoyer et traiter les informations du graphe ces informations circuleront par l'intermédiaire de l'URL grâce à des requêtes créées en JavaScript.

### 3.3.1-sauvegarde

Pour réaliser la sauvegarde nous devons récupérer le nom du graph qu'aura rentré l'utilisateur mais surtout les informations du graphe actuelle. Nous avons donc ajouté une fonction membre de Node "getSaveFormatedString" pour créer une chaîne capable de passer dans l'URL.

Exemple:



Pour ce graphe sauvegarder sous le nom "exemple" nous renverrons un URL du type:

save.php?val=exemple&&node1=0&coordx1=150&coordy1=63&edge1-4=1&node2=0&coordx2=325&coordy2=70&node3=0&coordx3=211&coordy3=233&edge3-4=1&node4=0&coordx4=405&coordy4=234&edge4-1=1&edge4-3=1&nb=4

val=exemple : val sera le nom du graph donné par l'utilisateur.

node1=0 : node1 représente le nœud d'id 1 et le 0 est sa valeur .

coordx1=150&coordy1=63 : coordonnées x et y du nœud d'id égale à 1 .

edge1-4=1 : Arc du nœuds 1 vers 4 de poids 1.

Une fois renvoyer sur la page PHP grâce aux requête de la page JavaScript nous récupérons chaque éléments pour les stocker 1 à 1 dans la base de données grâce aux langages SQL. Nous utiliserons des boucles simples pour récupérer dans les tables retourner en GET tous les éléments (nœud et arcs) pour ensuite les insérer avec des commandes telles quelles :

mysql\_query("insert into noeud (id\_noeud,node\_value,x,y,identifiant) values ('$n','$nodeid','$X','$Y','$selection');")or die (mysql\_error());

Lors de la sauvegarde d'un graphe une requête et également envoyer pour mettre a jour la liste déroulante des graphe enregistrer en mémoire .

### 3.3.1-Chargement

Pour permettre le chargement nous devons récupérer en PHP les informations du graph demander, la requête a donc seulement à envoyer le nom du graph sélectionné par l'utilisateur. Ensuite nous retournons chaque élément du graphe en questions sous forme d'arbre XML qui sera construit en PHP en incluant aux débuts de la page header ("Content-type : texte/xml") ;

L'arbre XML est de la forme :

<graph>

<weighted> true </weighted>

<oriented>false</oriented>

<node>

<node\_id>1</node\_id>

<label>12</label>

<x>34</x>

<y>455</y>

</node>

...

<edge>

<weight>45</weight>

<id\_source>1</id\_source>

<id\_dest>12</id\_dest>

</edge>

...

</graph>

Pour récupérer cet arbre XML il faudra spécifier le type de retour en récupérant la requête avec un var xml = requete.responseXML

La variable xml sera notre arbre XML aux même titre que document pour une page html .

remarque:Les balises id sont les id des nœuds/arcs dans la base de donnée, et les balises label les labels des nœuds dans le graphe.

Pour une meilleur compréhension, nous avons choisi de créer un tableau 'nodes' de 4xN (N est l'ordre du graphe), contenant l'élément XML correspondant à :

ligne 0 : les id des nœuds

ligne 1 : les labels des nœuds

ligne 2 : les abscisses

ligne 3 : les ordonnées

Les informations relatives à un noeud se trouvent donc en ligne.

Nous créons tout d'abord un nouveau graphe avec en paramètre les valeurs des balises weighted et oriented, que l'on converti en booléen grâce à la fonction getBool créé pour nos besoins dans la fonction loadTheGraph(les valeurs récupérées étant des chaînes de caractères).

Ensuite, on parcourt nodes et, pour chaque colonne, on récupère les informations des lignes 1, 2 et 3, et l'on ajoute les noeuds un à un. Nous avons dû modifier la fonction addNode pour pouvoir ajouter les coordonnées en paramètre (optionnels).

**

Pour les arcs, on reprend le même principe : construction d'un tableau 3xM (M nombre d'arcs), de cette forme :

**

Les valeurs dans les tableaux Edge pour id\_source et id\_dest ne correspondent pas directement aux id des nœuds. Pour les récupérer il suffit d'utiliser les propriétés firstChild et nodeValue sur les chaque edge.

Dans ce tableau, on a donc les id des noeuds, non leurs labels, ce qu'on préférerait. Il a donc fallu créer une fonction, getLabelById, qui prend un id en paramètre, et cherche le label correspondant dans le tableau nodes.

Ensuite, il suffit de parcourir le tableau edges et d'ajouter des arcs avec les valeurs récupérées.

Nous avons là aussi dû modifier la fonction addEdge, puisqu'elle ne prenait initialement pas le poids en paramètre, et le mettait à 1 par défaut. Le poids est toutefois optionnel.

De cette façon, à partir d'un arbre XML, nous avons recréé un graphe stocké en base de données, et nous sommes maintenant en mesure de l'afficher à l'écran.

## 3.4-Conclusion

Cette Partie d'implémentation d'une base de données nous permet maintenant de faire vivre en dehors de notre page les graphes créés. Elle nous a aussi permis de manipuler des notions Logique pour créer une BDD compatible avec nos exigences.

# 4- Conclusion

Ce projet nous à permis d'explorer tous les éléments vus en cours de l'année dans différent domaine il nous a permis de découvrir le travail que représente un projet avec tous les éléments qu'ils lui sont associés, présentation de l'avancement lors de soutenance, Rapport régulier, Batterie de tests complet sur nos Implémentation. Grâce à cette Partie Web nous avons pu découvrir la POO en langage WEB pour améliorer notre code mais aussi de nombreuses méthodes et technologie nous permettant de faire vivre une application WEB que ce soit par la POO, les méthodes comme l'AJAX ou encore une mise en place de BDD.