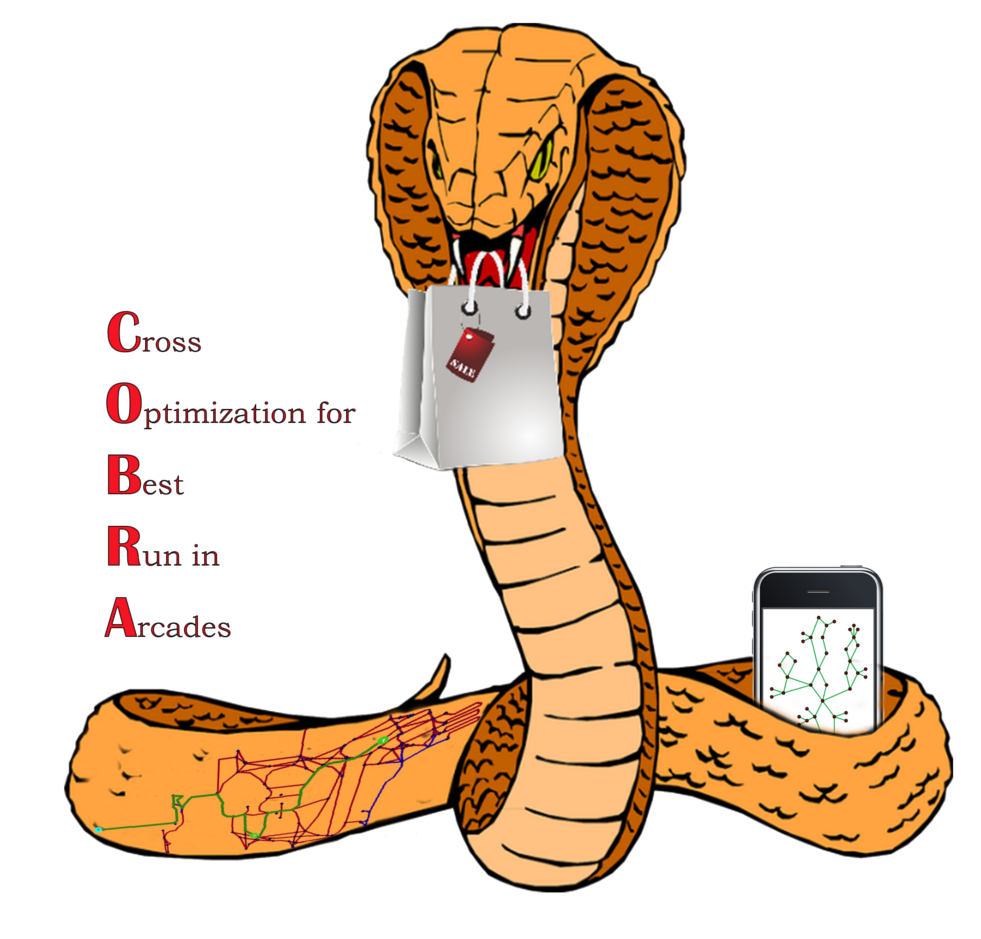
MANUEL DE DEVELLOPEMENT DE L’APPLICATION C.O.B.R.A**.**



# Introduction :

Nous allons vous présenter dans ce manuel l’organisation et le fonctionnement du code de notre application C.O.B.R.A. qui est une application mobile d'aide au déplacement dans le centre commercial des Arcades situé à Noisy-le-Grand qui a pour caractéristiques notamment d’être multi-niveau.

Le code de notre application est disponible sur git au lien suivant :

<https://github.com/hbaltz/projetDev.git>

Le code final se situe dans 2-Android\Assemblee\AndroidProjet

En annexe vous trouverez la liste de l’ensemble des variables globales avec leur utilité.

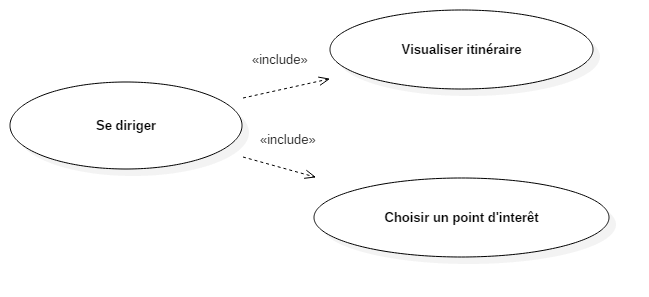
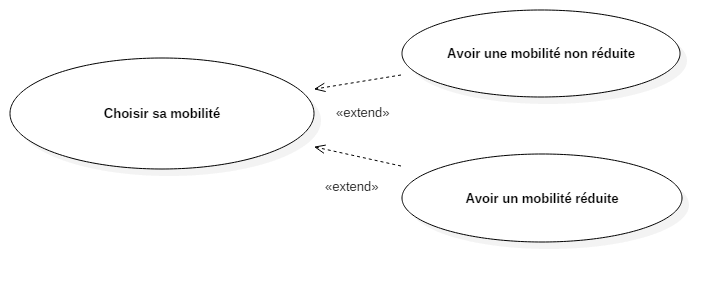
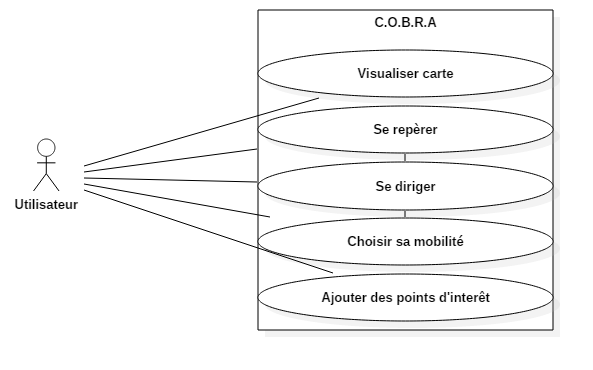
# Modélisation :

## Cas d’utilisation :

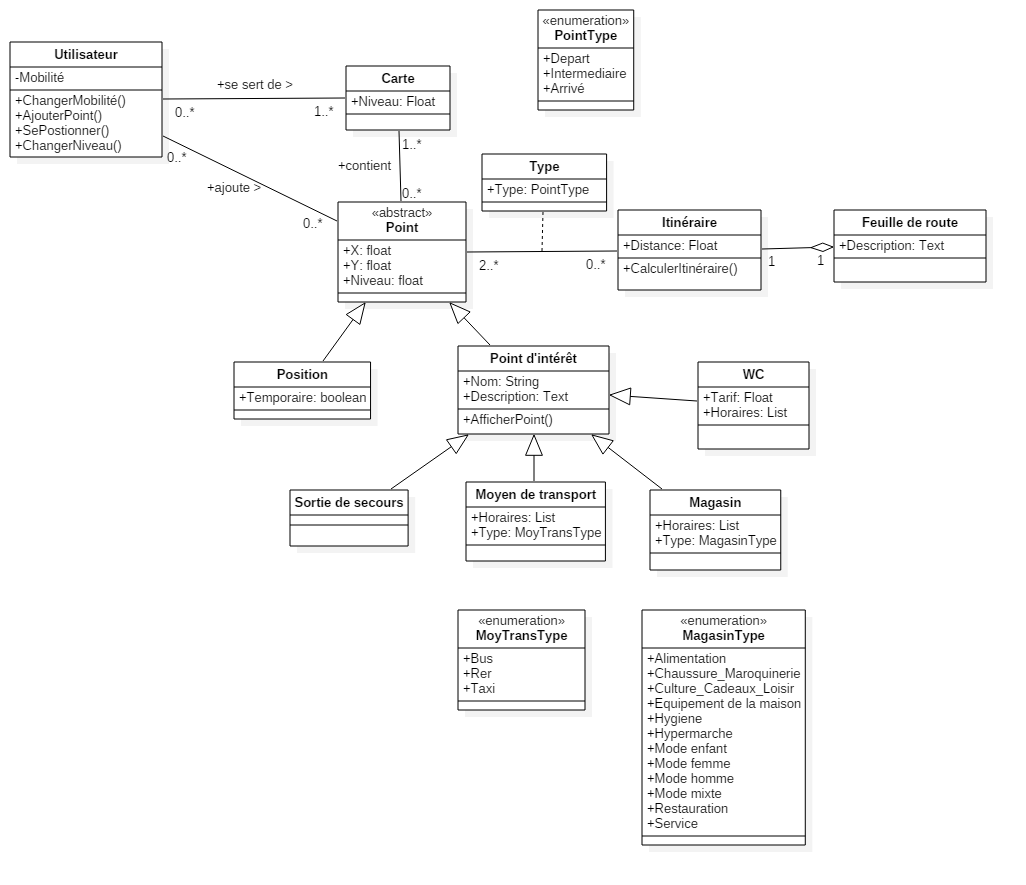
Notre application doit aider un utilisateur à effectuer différentes tâches dont les principales sont listées ci-dessous :

* Visualiser la carte du centre commercial
* Se repérer dans ce centre
* Se diriger à l’intérieur des Arcades
* Choisir sa mobilité, c’est-à-dire si il a une mobilité réduite (handicap, poussette, caddie) ou non
* Ajouter des points d’intérêt

Vous trouverez ci-dessous les diagrammes de cas d’utilisation représentant les fonctionnalités principales de notre application COBRA (Cross Orientation for Best Run in Arcades) :



## Diagramme de classes :



Nous avons identifiés 6 classes principales :

* Utilisateur
* Carte
* Point (Dont position et point d’intérêt est une classe enfant)
* Point d’intérêt (Dont WC, sortie de secours, moyen de transport, magasin sont des classes enfants)
* Itinéraire
* Feuille de route

Nous sommes partis de données préexistantes notre modélisation s’est donc basée sur ses données, notamment pour les classes Point et Point d’intérêt.

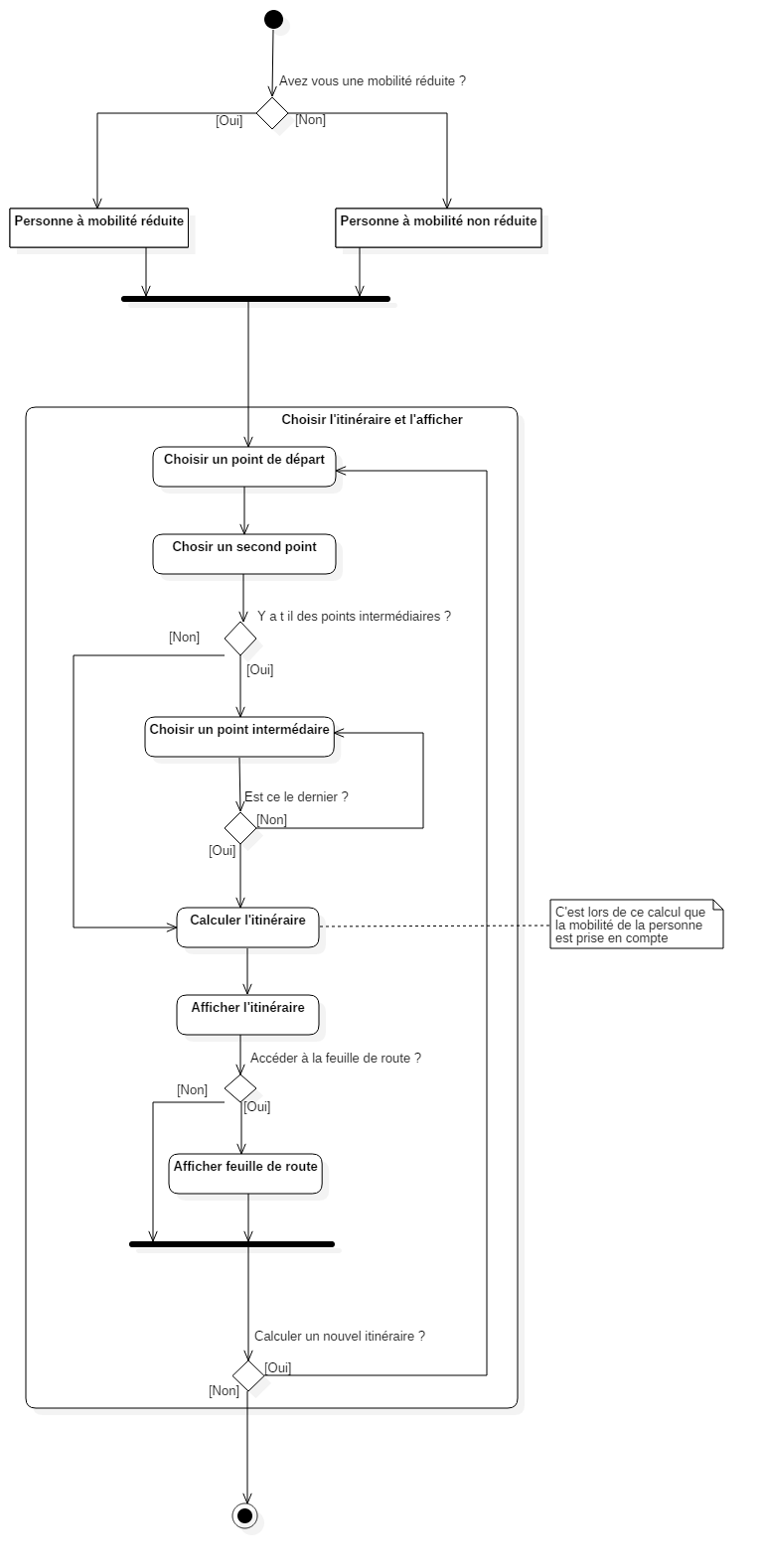
Les classes Itinéraires et Feuille de routes servent à afficher les informations nécessaires à l’Utilisateur pour se déplacer. (Feuille de route n’étant pour l’instant pas utilisée dans notre application).

La classe Carte sert à la gestion du multi-niveau, en effet le but de notre projet est de gérer un itinéraire sur du multi-niveau.

Enfin la classe Utilisateur sert à gérer la mobilité de ce dernier (s’il peut ou non monter des escaliers).

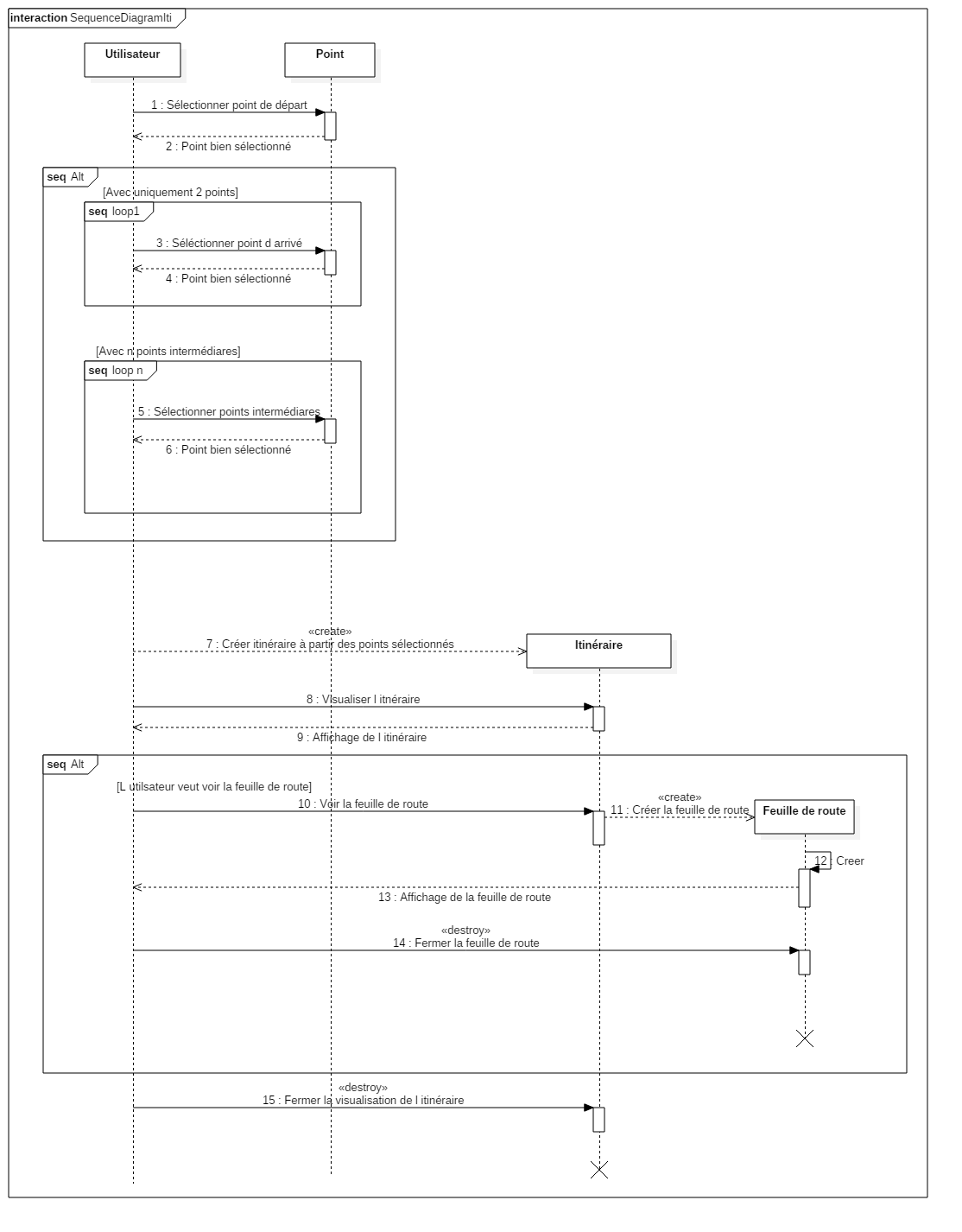
## Diagramme d’activité :

Ce diagramme présente le déroulement de la création d’un itinéraire sur notre application



Notez cependant que le choix de point intermédiaire et l’affichage de feuille de route n’est pas gérer dans la version actuelle de notre code.

## Diagramme de séquence :



Dans notre application, nous commençons par choisir un point de départ, puis nous choisissons le point d’arriver, (le choix de point intermédiaire n’étant pas encore implémenter). A ce moment le calcul d’itinéraire est effectué. Puis la feuille de route est créée et affichée (Non implémentée dans la version actuelle de notre application).

# Organisation et fonctionnement du code :

## Organisation général :

Nous allons vous présenter ici l’organisation général du code, le fonctionnement de chacune des parties décrites sera développée par la suite.

* Dans AndroidProjet\app\src\main\java\com\example\formation\androidprojet\_v1 nous avons les fichiers java qui gèrent le fonctionnement de notre application:
  + MainActivity.java : gère la majeure partie fonctionnement de notre application (récupération de la base de données, calcul et affichage d’itinéraire par niveaux, formulaire avec autocomplétion).
  + Choix.java, Listetype.java, Listemagasin.java : servent à gérer le formulaire par choix du type de magasin (et le point de départ par QrCode).
* Dans AndroidProjet\app\src\main\res\layout nous avons les fichiers liées à l’affichage des différentes activités, les différents fichiers correspondent aux différents fichiers java décrits ci-dessus.

## Fonctionnement des différentes parties :

### MainActivity.java : les méthodes :

#### 2.2.1.a OnCreate() :

Nous commençons par gérer les éléments graphiques. La gestion des différentes parties s’effectue toujours de la même façon, nous allons expliquer le déroulement de la gestion d’un élément graphique complexe (une zone de saisie d’autocomplétion), les autres se gérant de la même façon :

* // Nous ouvrons ici un objet de AutoCompleteTextView défini dans le layout:
* textViewDep = (AutoCompleteTextView)  
   findViewById(R.id.*dep\_magasin*);
* // Nous récupèrons la valeur dans string.xml qui correspond au texte que l’on souhaite affiché par défaut dans la zone de saisie
* String depTxt = getResources().getString(R.string.*dep*);
* // Nous mettons le texte par défaut dans le zone de saisie
* textViewDep .setHint(depTxt);
* // Nous commençons la recherche automatique dès la première lettre écrite
* textViewDep .setThreshold(1);
* // Nous associons à la zone de saisie un listener
* textViewDep .setOnItemClickListener(new BoutonSaisieAutomatiqueDepListener());

Nous ouvrons ensuite l’intégralité des cartes de fond qui seront affiché dans notre application et nous mettons leur visibilité à false pour les cache, ainsi les images sont toutes chargés et il suffit de changer la visibilité pour afficher les images de fond. Nous ajoutons de plus une couche graphique par-dessus les images de fond c’est à cet endroit que nous dessinerons nos points et notre itinéraire :

* // Retrieve the map and initial extent from XML layout  
  mMapView = (MapView) findViewById(R.id.*map*);  
  // Mise en place des fonds et visibilité = false :  
  mMapView.addLayer(mTileLayer0);  
  mTileLayer0.setVisible(false);  
  // Ajout couche graphique :  
  mMapView.addLayer(mGraphicsLayer);

Nous gérons ensuite la définition des marqueurs des points d’arrivés et de départ :

* // Création symbole point départ/arrivé :  
  marqueur = getResources().getDrawable(R.drawable.*ic\_action\_marqueur*);  
  symStop = new PictureMarkerSymbol(marqueur);

Enfin nous récupérons les informations dans la base de données à l’aide de la fonction accesBdd() dont le fonctionnement sera détaillée par la suite

#### 2.2.1.b onActivityResult(int requestCode, int resultCode, Intent intent) :

Fonction qui sert à la gestion des informations reçu de la part du QrCode et du formulaire

* Nous commençons par gérer les informations reçues lorsque qu’un QrCode est scanné. / Nous utilisons la classe IntentIntegrator et sa fonction parseActivityResult pour parser le résultat du scan.
* IntentResult scanningResult = IntentIntegrator.*parseActivityResult*(requestCode, resultCode, intent);

Si le résultat est non nul, alors nous traitons les informations obtenus par le scan du QrCOde. Nous récupérons le contenu, le format du code barre, et nous affichons le résultat dans nos TextView. Ensuite nous testons si le contenu de notre QrCode correspond à une valeur connue et si oui on ajoute pour le cas où il s’agit du Qr Code 01 un point sur la carte (ce point n’est pour l’instant pas ajouté au stop).

* // Nous récupérons le contenu du code barre :  
  String scanContent = scanningResult.getContents();  
  // Nous récupérons le format du code barre :  
  String scanFormat = scanningResult.getFormatName();
* if(scanContent.equals( "QR code 01" ) ){  
   // On marque la geometrie du QR code sur la carte  
   // Rappel, on teste avec le magasin "La grande recre"  
   Geometry projection = geomen.*project*(geom\_QR\_code, WKID\_RGF93, mapRef);  
   mGraphicsLayer.addGraphic(new Graphic(projection, new SimpleMarkerSymbol(Color.*RED*, 10, STYLE.*CROSS*)));}
* Nous gérons ensuite les informations renvoyé par le formulaire (normalement point de départ et point d’arrivé). Nous commençons par récupérer le nom des magasins sélectionnés dans le formulaire. Si il y a deux stops ou plus nous les réinitialisons. Nous récupérons à l’aide de leurs noms les géométries des magasins sélectionnés et nous les ajoutons au stop. Nous lançons ensuite le calcul d’itinéraire.
* // Récupération des noms du magasin de départ et d'arrivée  
  final String mag\_dep = intent.getStringExtra("Depart");
* final String mag\_arr = intent.getStringExtra("Arrivee");
* // Nous comptons le nombre de points présents dans mStops :  
  int tStop = mStops.getFeatures().size();
* // Si il y en a plus de deux nous réinitialisons les stops :  
  if( tStop >=2 ) {  
   mStops.clearFeatures();  
   clearAffich();} // Fonction expliquée plus bas
* // Nous retrouvons les points de départ et d'arrivé à l'aide de leurs noms dans la liste de magasin :  
  Geometry ptDep = trouverPtSel(mag\_dep, true);  
  depart = geomen.*project*(ptDep, WKID\_RGF93, mapRef);  
  ajouterPoint(depart, symStop);
* // Nous effectuons la même chose pour l’arrivée (non écrit ici)
* // On récupére à nouveau le nombre de stop :  
  tStop = mStops.getFeatures().size();  
  // Si on a 2 stops on calcule et on affiche l'itinéraire :  
  if( tStop >= 2) {  
   calculerIti(mapRef);  
   afficherPpv(mapRef);}

### 2.2.2 MainActivity.java : les listener :

#### 2.2.2.a checkedListener :

Cette classe défini un listener utilisé pour modifié la valeur de estRestreint (mobilité réduite ou non), elle ne possède donc qu’une méthode onClick qui change cette valeur

* // Si la checkbox est cochée on met le booléen estRestraint à true, sinon à false  
  if (((CheckBox) v).isChecked()) {estRestreint = true;}  
  else {estRestreint = false;}

#### 2.2.2.b BoutonEtageListener :

Cette classe défini un listener sur une liste déroulante à choix, plus précisément sur le choix de l’étage. Cette classe implémente deux méthodes onItemSelected(AdapterView<?> parent, View view, int position, long id) et onNothingSelected(AdapterView<?> arg0).

* Nous ne faisons rien dans onNothingSelected.
* Dans onItemSelected, nous récupérons l’objet sélectionné et en fonction nous changeons les variables globales de visibilité des images de fond. Puis nous n’affichons que l’étage sélectionné et à l’aide de la fonction afficherIti(), nous affichons l’itinéraire au niveau sélectionné. Voici pour exemple ce qu’il se passe lorsque l’étage 0 est sélectionné.
* // On selecting a spinner item  
  String etageSelec = parent.getItemAtPosition(position).toString();  
    
  // Nous récupèrons les noms des étages qui sont stockés dans ressources.strings.values  
  String[] nom\_etage = getResources().getStringArray(R.array.*etage\_array*);  
    
  // Test suivant la sélection de l'utilisateur:  
  if (etageSelec.equals(nom\_etage[0])) {  
   etgsSelected = false;  
   etg0Selected = true;  
   etg1Selected = false;  
   etg2Selected = false;}
* // Nous affichons l'étage sélectionné :  
  mTileLayer.setVisible(etgsSelected);  
  mTileLayer0.setVisible(etg0Selected);  
  mTileLayer1.setVisible(etg1Selected);  
  mTileLayer2.setVisible(etg2Selected);

#### 2.2.2.c BoutonQRcodeListener :

Cette classe défini le listener utilisé sur le bouton de scan du QrCode, elle implémente une méthode onClick(View v). Cette méthode lance le scan par QrCode à l’aide d’un IntentIntegrator :

* // Nous lançon le scanner au clic sur notre bouton  
  IntentIntegrator integrator = newIntentIntegrator(MainActivity.this);  
  integrator.initiateScan();

#### 2.2.2.d BoutonSaisieAutomatiqueDepListener et BoutonSaisieAutomatiqueARrrListener :

Nous devons définir deux classes de listener pour les zones de saisie automatique de magasins (l’un pour le départ, l’autre pour l’arrivée), en effet récupérer facilement l'id de l’AutoCompleteTextView sur laquelle nous cliquons. Ces classes implémentent la méthode onItemClick(AdapterView<?> parent, View view,int position, long id). Cette méthode fonctionne similairement dans les deux classes. Le fonctionnement de cette méthode est très proche du fonctionnement de [onActivityResult()](#_2.2.1.b_onActivityResult(int_reques) pour la gestion du formulaire.

Cette méthode commence par remettre l’affichage à zéro, si il y a déjà plus de deux stops, lorsqu’un nouveau point de départ ou un nouveau point d’arrivée est sélectionné, tout en rajoutant celui qui n’est pas modifié au stop, c’est-à-dire qui si l’on modifie l’arrivée le point de départ n’est pas changé il faut donc penser à l’ajouter aux stops.

* // Remise à zéro des stops :  
  // Si il y a plus de deux stops au départ  
  // Nous réinitialisons la vue et nous remettons en fonction du bouton sélectionné le départ  
  // ou l'arrivé (on remet le départ si on modifie l'arrivé et inversement)  
  if( tStop >=2 ) {  
   mStops.clearFeatures();  
   clearAffich();  
   ajouterPoint(depart, symStop);}

Nous récupèrons ensuite le nom du magasin sélectionné dans la liste des magasins générée par la saisie automatique. Puis on récupère la géométrie de ce point dans la liste des géométries des magasins.

* // Nous sélectionnons le magasin dans la liste de saisie automatique  
  String item = parent.getItemAtPosition(position).toString();  
  Log.*v*("mag\_selectionne",item);
* // A l’aide de la fonction trouverPtSel nous récupèrons le point dans la liste des géométries  
  ptTest = trouverPtSel(item, true);  
  if (ptTest !=null){  
   trouve = true;}

Nous ajoutons ensuite le point au stop en gérant le fait que ce soit le point de départ ou d’arrivée.

* // Lorsque que nous avons trouvé un point  
  // Nous gérons le fait que ce soit le départ ou l'arrivé  
  // Dans tous les cas nous l'ajoutons au stop et nous l'affichons à l’aide de la fonction ajouterPoint()  
  if(trouve){  
   depart = geomen.*project*(ptTest, WKID\_RGF93, mapRef);  
   ajouterPoint(depart, symStop);}

Si il y a deux stops ou plus, nous lançons le calcul d’itinéraire et nous affichons le plus proche voisin du point de départ pour aider l’utilisateur à orienter la carte.

* // Si nous avons 2 stops nous calculons et nous affichons l'itinéraire  
  if( tStop >= 2) {  
   calculerIti(mapRef);  
   afficherPpv(mapRef);}

### 2.2.3 MainActivity.java : les fonctions :

#### 2.2.3.a accesBdd() :

Cette fonction sert à récupérer les données relatives au graphe et au magasin dans la géodatabase

Nous commençons par récupérer le chemin où se trouve notre géodatabase  ainsi que le nom du graphe:

* String networkPath = chTpk + "/Routing/base\_de\_donnees.geodatabase";  
  String networkName = "GRAPH\_Final\_ND";

Nous ouvrons ensuite localement la geodatabase :

* // Ouverture locale de la geodatabase  
  Geodatabase gdb = new Geodatabase(extern + networkPath);

Nous récupérons ensuite un magasin pour tester notre QrCode :

* // Magasin test : La grande recre  
  geom\_QR\_code = gdb.getGeodatabaseTables().get(0).getFeature(1).getGeometry();

Nous récupérons ensuite les tables contenant l’ensemble des arcs pour les différents niveaux, nous allons explicitez la chaîne de traitement pour l’étage 1, les étages 0 et 2 se traitant de la même manière. Il faut déjà noter que dans la déclaration des variables il faut définir le nombre d’objet qui sera contenu dans nos array, il faut donc connaître le nombre d’arc au préalable. De plus comme nous allons regroupez l’ensemble des arcs en une seule géométrie, il faut que les array n’est pas un taille trop grande car la fonction de fusion se limite à 512 géométries, dans notre cas nous avons dû couper les arcs des étages 1 et 2 en deux groupes que nous fusionnons ensuite.

GeodatabaseFeatureTable tab\_niv1 = gdb.getGeodatabaseTables().get(12);

// Définition du nombre d'arcs :

int l1 = array\_geom\_niv1\_1.length;  
int l2 = array\_geom\_niv1\_2.length;

Geometry poubelle = new Polyline(); // Varaible utile si pas d'objet dans la base

// Etage 1\_1 :  
for(int j=1; j<=l1; j++){  
 if (tab\_niv1.checkFeatureExists(j)) {  
 array\_geom\_niv1\_1[j-1] = tab\_niv1.getFeature(j,WKID\_RGF93).getGeometry();  
 } else {array\_geom\_niv1\_1[j-1] = poubelle;}}  
  
// Etage 1\_2 :  
int k1 = 0;  
double longTot = 0;  
for(int j=l1+1; j<=l1+l2; j++){  
 if (tab\_niv1.checkFeatureExists(j)) {  
 array\_geom\_niv1\_2[k1] = tab\_niv1.getFeature(j,WKID\_RGF93).getGeometry();  
 } else {array\_geom\_niv1\_2[k1] = poubelle;}  
 k1 = k1+1;}

Nous possédons maintenant l’ensemble des arcs du niveau 1, ils sont stockés dans deux array différents, nous allons maintenant fusionner l’ensemble de ses géométries pour n’en former plus qu’une seule.

array\_geom\_niv1[0] = geomen.*union*(array\_geom\_niv1\_1, WKID\_RGF93);  
array\_geom\_niv1[1] = geomen.*union*(array\_geom\_niv1\_2, WKID\_RGF93);  
geometries\_niveau1 = geomen.*union*(array\_geom\_niv1, WKID\_RGF93);

Cette fonction récupère aussi l’intégralité des magasins présents dans la géodatabase, nous récupérons leurs géométries, leurs noms et leurs types. Pour accélérer le temps de traitement nous commençons par récupérer les features des différents magasins puis de ses feature nous récupérons la géométrie, le nom et le type de chaque magasin.

for(int v=0; v<=2; v++){  
 GeodatabaseFeatureTable mag = gdb.getGeodatabaseTables().get(v);  
  
 long nbr\_lignes = mag.getNumberOfFeatures();  
 for(int l=1; l<=nbr\_lignes; l++){  
 if (v==0) { // Si v=0 on est au niveau 0 on remplit donc la liste associée  
 if (mag.checkFeatureExists(l)) {// Si le magasin existe on l’ajoute sinon, on met l’élément à nul  
 mag\_niv0[l-1] = mag.getFeature(l);  
 } else {mag\_niv0[l-1] = null;}  
 } else if (v==1) {  
 if (mag.checkFeatureExists(l)) {  
 mag\_niv1[l-1] = mag.getFeature(l);  
 } else {mag\_niv1[l-1] = null;}  
 } else if (v==2) {  
 if (mag.checkFeatureExists(l)) {  
 mag\_niv2[l-1] = mag.getFeature(l);  
 } else {mag\_niv2[l-1] = null;}  
 }  
 }  
}

Nous possédons maintenant l’ensemble des feature des magasins, nous allons maintenant récupérer la géométrie, le nom et le type de chacun des magasins, nous allons vous présenter la chaîne de traitement pour l’étage 1, les autre étages se traitant de la même manière :

// Etage 1  
int len1 = mag\_niv1.length;  
for(int k=0; k<len1; k++) {  
  
 Feature Mag = mag\_niv1[k];  
  
 // Récupération géométrie :  
 mag\_niv1\_geom[k] = mag\_niv1[k].getGeometry();  
  
 // Récupération nom et type :  
 Map<String, Object> lignes = Mag.getAttributes();  
 Object type = lignes.get("TYPE");  
 Object nom\_mag = lignes.get("NOM");  
 lst\_types\_niveau1.add(type);  
 lst\_mag\_niveau1.add(nom\_mag);  
 lst\_nom\_mag.add(nom\_mag);}

Enfin lors du calcul du plus proche voisin du point de départ, nous avons besoin d’une géométrie contenant l’ensemble des magasins donc nous faisions l’union de la géométrie des magasins pour chaque étage :

mag\_niveau0 = geomen.*union*(mag\_niv0\_geom, WKID\_RGF93);  
mag\_niveau1 = geomen.*union*(mag\_niv1\_geom, WKID\_RGF93);  
mag\_niveau2 = geomen.*union*(mag\_niv2\_geom, WKID\_RGF93);

#### 2.2.3.b trouverPtSel(String item, boolean estDepart):

Cette fonction sert à retrouver le magasins ayant pour nom item dans la liste des magasins que nous avons récupérez à l’aide d’accesBdd(), le booléen d’entrée estDepart sert à définir si le point que nous allons récupérer est le point de départ ou non (ainsi nous ne modifions la variable globale de niveau dupoint de départ ou du point d’arrivé).

Pour récupérer un magasin, nous avons allons parcourir la liste des magasins de chaque étage jusqu’à ce que nous trouvions une géométrie ayant pour nom item, si nous n’en trouvons pas la fonction renvoie une géométrie null. Pour améliorer le temps de calcul, nous nous servons d’un booléen trouve qui se met à vrai si nous trouvons une géométrie, ainsi si le magasin est trouvé à l’étage 0 pas besoin de parcourir les étages 1 et 2 par exemple.

// Nous parcourons la liste récupérer au début dans la geodatabase et nous récupérons la géométrie correspondante au magasin choisie par l'utilisateur  
for(int k=0; k< mag\_niv0.length; k++) {  
 Feature Mag = mag\_niv0[k];  
 Map<String, Object> lignes = Mag.getAttributes();  
 Object nom\_mag = lignes.get("NOM");  
 if(nom\_mag.equals(item)){  
 ptTest = mag\_niv0[k].getGeometry();  
 trouve = true;  
 if(estDepart) {niveau\_dep = 0;} // Nous modifions la variable de niveau correspondant soit au point de départ soit au point d’arrivée  
 else{niveau\_arr = 0;}}}

Nous effectuons le même traitement pour les étages 1 et 2

#### 2.2.3.c ajouterPoint(Geometry point, Symbol symbol) :

Cette fonction sert à ajouter une géométrie point aux stops et sur le graphe avec le symbole symbol

mGraphicsLayer.addGraphic(new Graphic(point, symbol));  
StopGraphic stop = new StopGraphic(point);  
mStops.addFeature(stop);

#### 2.2.3.d clearAffich()°:

Cette fonction sert à réinitialiser l'affichage :

mGraphicsLayer.removeAll();  
mMapView.getCallout().hide();

#### 2.2.3.e calculerIti(SpatialReference mapRef)°:

Cette fonction sert à calculer l’itinéraire dans le référentiel spatial mapRef.