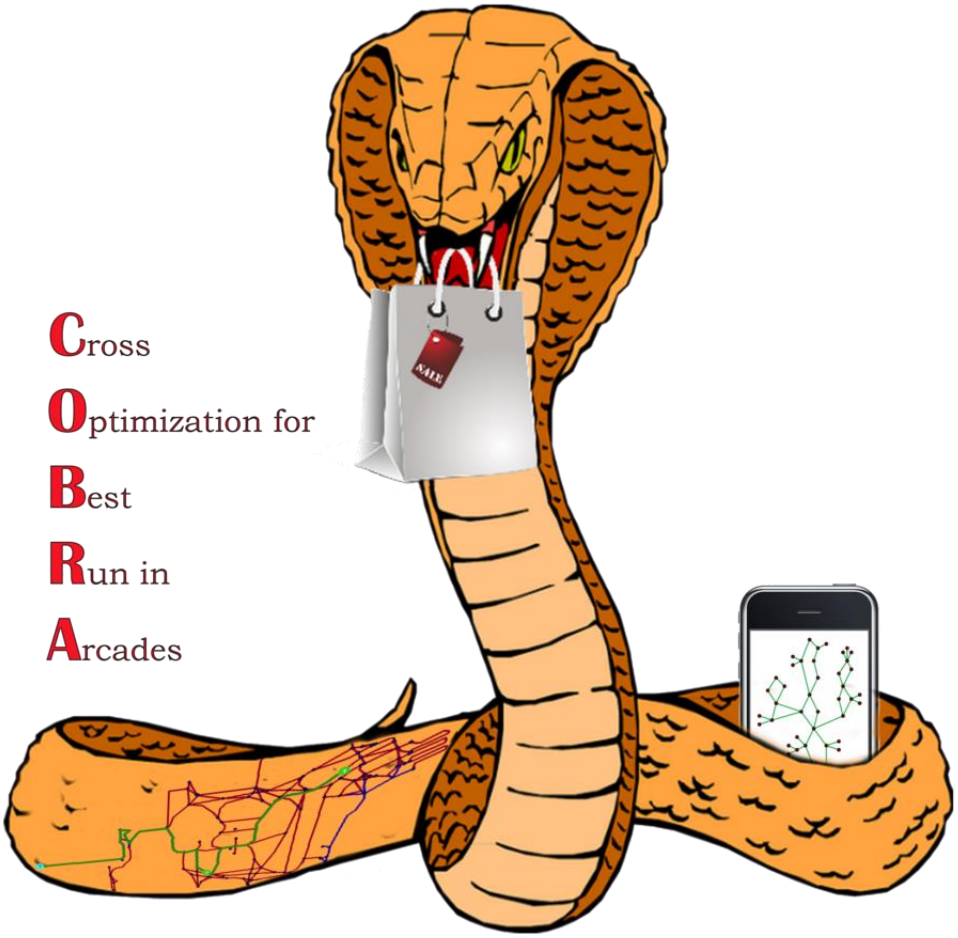
# MANUEL DE DEVELLOPEMENT DE L'APPLICATION C.O.B.R.A.



# Sommaire :

# 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.

Notre projet nécessite la modification du build.gradle et du manifeste respectivement placée dans app/manifests et Gradle Scripts/build.gradle(app), référez-vous à la partie consacré à ses modifications pour plus d'informations.

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

Vous trouverez nos adresses mails dans la partie contact, vous pouvez ainsi nous contacter si vous avez la moindre question ou commentaire.

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

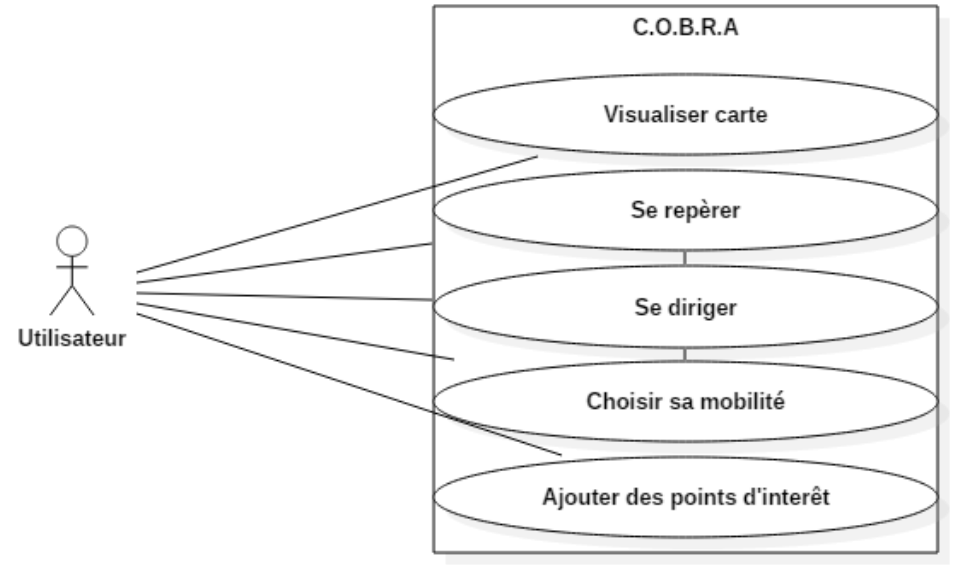
# 1. Modélisation :

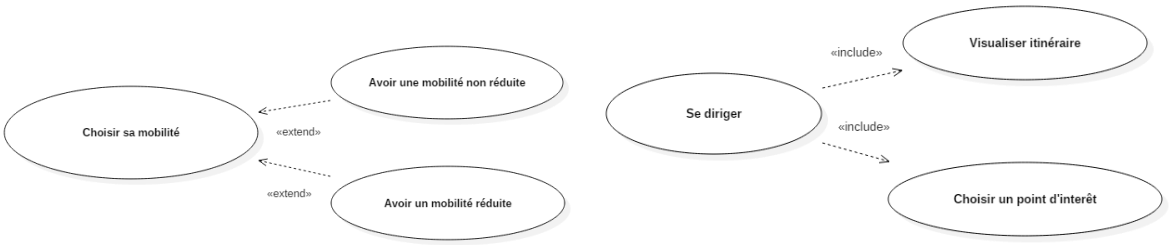
## 1.1 Cas d'utilisation :

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

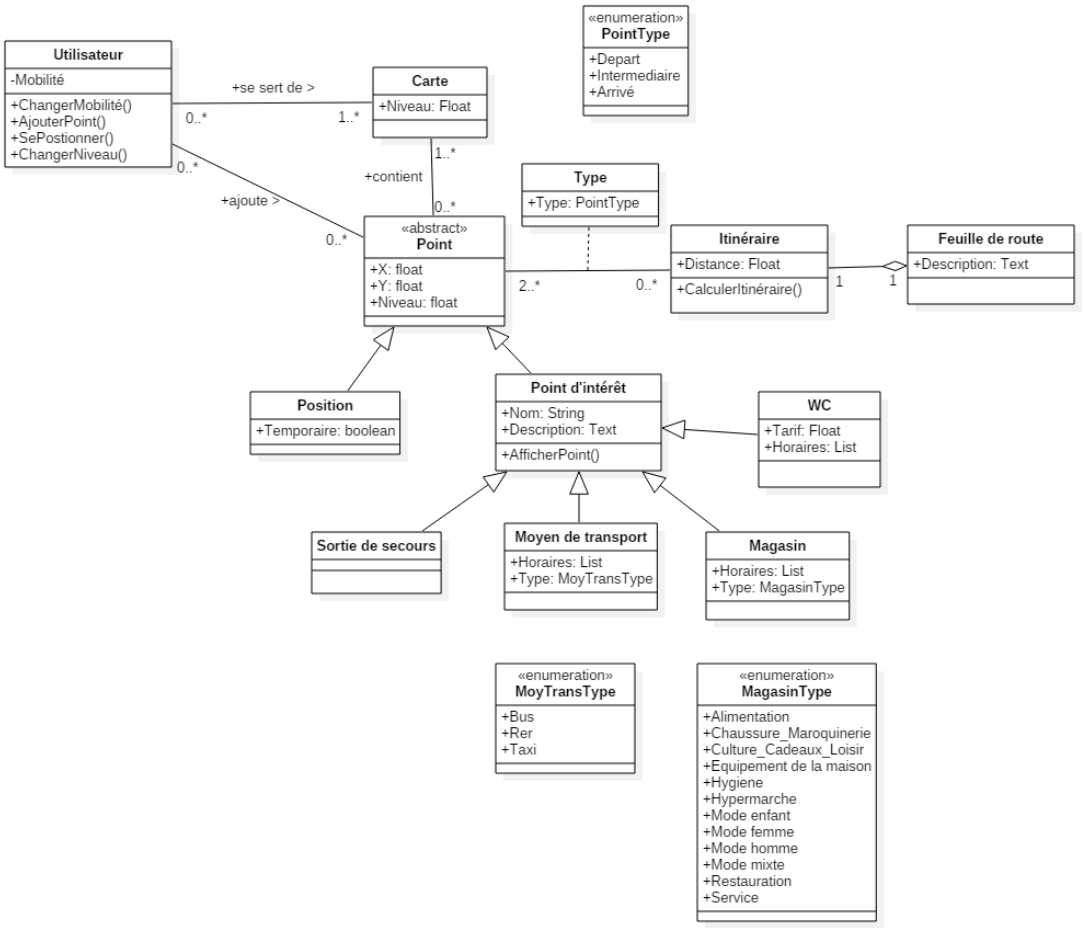
- Visualiser la carte du centre commercial
- Se repérer dans ce centre
- Se diriger à l'intérieur des Arcades
- Choisir sa mobilité, c'est-à-dire si il a une mobilité réduite (handicap, poussette, caddie) ou non
- Ajouter des points d'intérêt

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) :



## 1.2 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.

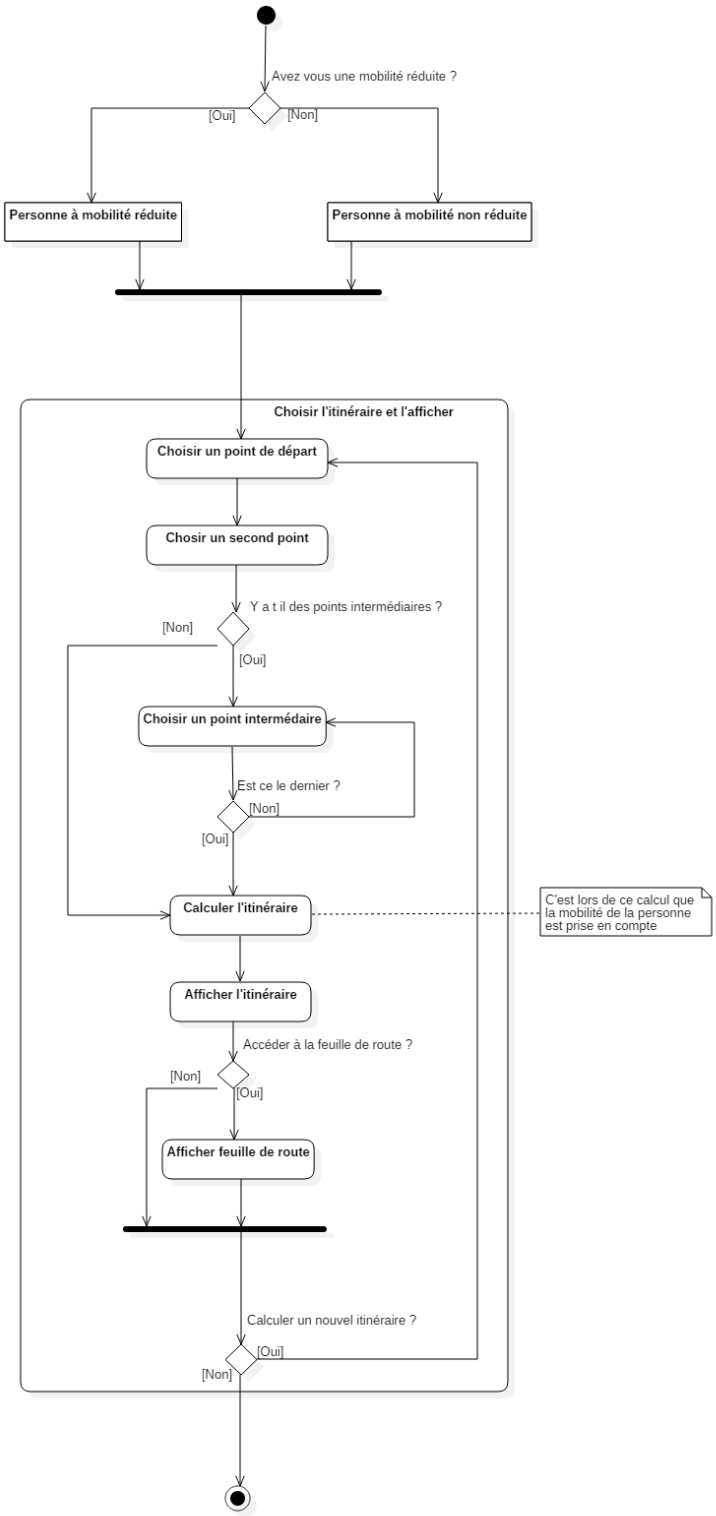
Les classes Itinéraires et Feuille de routes servent à afficher les informations nécessaires à l'Utilisateur pour se déplacer. (Feuille de route n'étant pour l'instant pas utilisée dans notre application).

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).

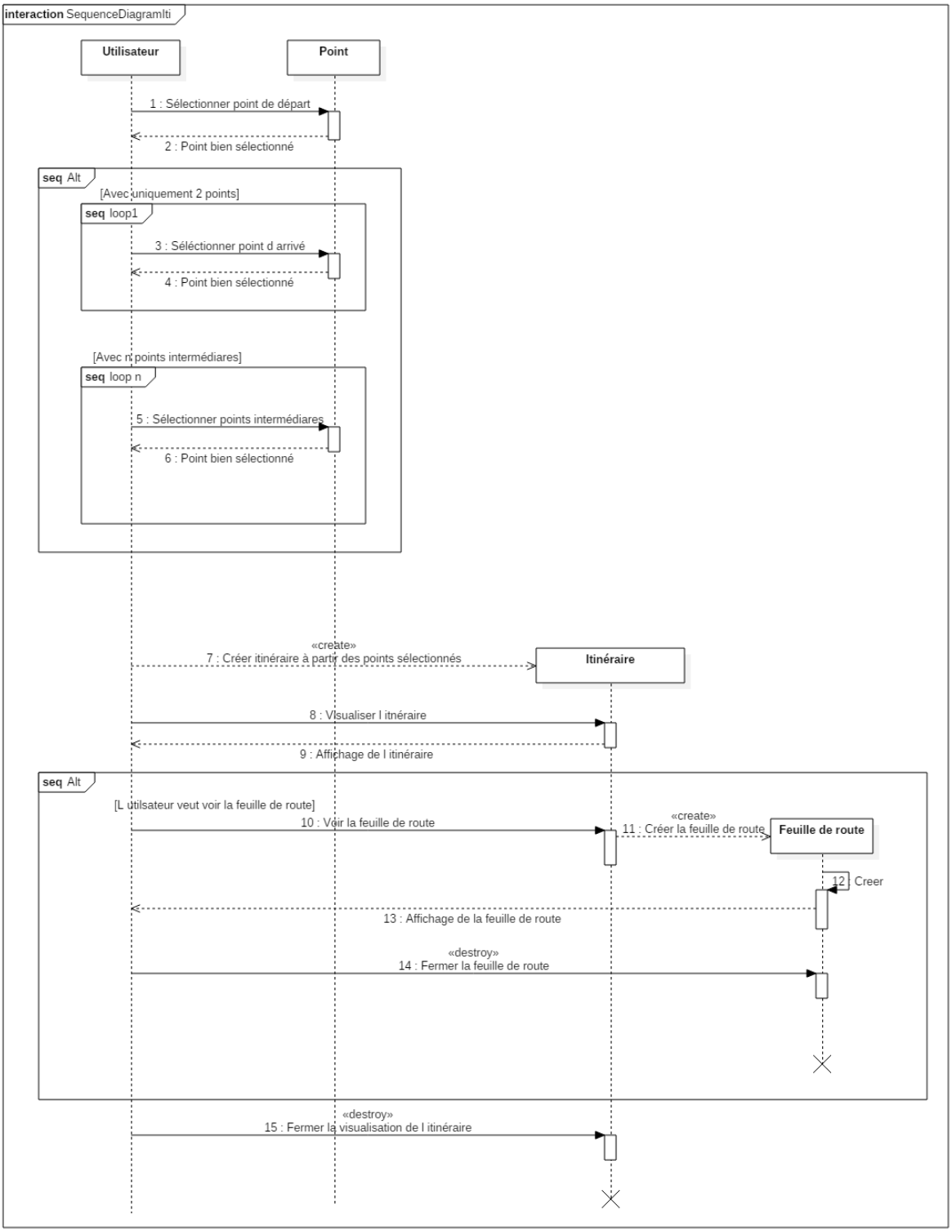
## 1.3 Diagramme d'activité :

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.

## 1.4 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).

# 2. Organisation et fonctionnement du code :

## 2.1 Organisation général :

Nous allons vous présenter ici l'organisation générale du code, le fonctionnement de chacune des parties décrites sera développé 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.

## 2.2 Fonctionnement des différentes parties :

### 2.2.1 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() 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.

Nous commençons par initialiser les paramètres de notre calcul d'itinéraire et à placer les différents éléments (les stops et les paramètres de calcul) dans le référentiel spatial mapRef :

```
RouteParameters params = mRouteTask.retrieveDefaultRouteTaskParameters();
params.setOutSpatialReference(mapRef);
mStops.setSpatialReference(mapRef);
```

Nous gérons ensuite la restriction de mobilité de l'utilisateur, si il est à mobilité réduite on ajoute aux paramètres de calcul la restriction sur les arcs, pour que l'utilisateur à mobilité n'est pas un itinéraire proposé avec des escaliers mais uniquement avec des ascenseurs :

```
// Si l'utilisateur est à mobilité réduite, on ajoute la restriction au
paramètre
if(estRestreint){
    String[] restrictions = {"Restriction"};
    params.setRestrictionAttributeNames(restrictions);
} else{
    String[] restrictions = {""};
    params.setRestrictionAttributeNames(restrictions);}
```

Nous ajoutons alors les stops aux paramètres de l'itinéraire :

```
params.setStops(mStops);
params.setReturnDirections(true);
```

Nous calculons alors l'itinéraire et nous récupérons le résultat :

```
RouteResult results = mRouteTask.solve(params);
Route result = results.getRoutes().get(0);
```

Nous allons maintenant traiter ce résultat pour l'adapter à du multiniveau pour cela nous allons nous server de l'analyse spatial. Nous allons commencer par projeter la géométrie du graphe de chaque niveau dans le repère mapRef. Puis nous allons récupérons l'intersection de l'itinéraire avec la géométrie de chacun des étages pour permettre de n'afficher que cette partie à l'utilisateur. Nous allons vous présenter la chaîne de calcul pour le niveau 0, les autres étages étant gérer de la même façon :

```
// Nous projetons les arcs dans le repère local :
projection_niv0 = geomen.project(geometries_niveau0, WKID_RGF93, mapRef);
// Nous récupérons l'itinéraire golbal
geom = result.getRouteGraphic().getGeometry();
// Nous recoupons l'itinéraire avec les arcs de chacun des niveaux :
geom_intersect_niv0 = geomen.intersect(geom, projection_niv0, mapRef);
```

Nous possédons maintenant l'itinéraire globale ainsi que l'itinéraire sur chacun des étages. Nous gérons l'affichage à l'aide de la fonction afficherIti(). De plus nous affichons l'étage sur lequel le point de départ se situe

```
// Affichage de l'étage du point de départ :
spinnerEtgSel.setSelection(niveau_dep);
//Gestion affichage au moment du calcul d'itinéraire :
afficherIti();
```

#### 2.2.3.f afficherIti() :

Cette fonction sert à afficher l'itinéraire en fonction de l'étage sélectionné.

Nous commençons par effacer toutes itinéraires qui seraient déjà afficher :

```
mGraphicsLayer.removeGraphic(routeHandle);
```

Nous n'allons ensuite afficher que l'itinéraire qui correspond à l'étage sélectionné en arrière-plan, pour cela nous nous servons des variables globales de sélection d'étage qui sont modifié dans le listener BoutonEtageListener. Nous allons vous montrez comment cela est gérer pour un niveau, le niveau 0 par exemple.

Nous n'affichons l'itinéraire au niveau zéro uniquement si il y a une partie de l'itinéraire au niveau 0 et que c'est l'étage zéro qui est sélectionné :

```
// Nous ne visualisons que l'itinéraire au niveau selectionné :
if(geom_intersect_niv0 != null && etg0Selected) {
    if (!geom_intersect_niv0.isEmpty()) {
        routeHandle = mGraphicsLayer.addGraphic(new
Graphic(geom_intersect_niv0, ligSym));
    }
}
```

#### 2.2.3.g afficherPpv(SpatialReference mapRef) :

Cette fonction sert à afficher le magasin le plus proche du point de départ dans le référentiel mapRef pour permettre à l'utilisateur d'orienter sa carte.

Pour trouver le plus proche voisin, nous calculons la distance entre le point de départ et entre les magasins présents dans un buffer autour du point, le point le plus proche étant le point avec la distance la plus petite au point de départ. Pour améliorer le temps de calcul, nous ne parcourons calculons la distance du point de départ qu'au point présent à son étage et contenu dans un buffer autour du point de départ.

Nous allons vous présenter la chaîne de traitement pour le niveau 0, les autres étages étant géré de la même manière.

Nous commençons par projeter l'ensemble géométrique contenant l'intégralité des points d'un niveau dans le repère mapRef, puis nous retirons le point de départ de cet ensemble et nous calculons la distance entre cet ensemble au point de départ :

```
// Nous projetons les magasins en mapRef :
projection_mag_niv0 = geomen.project(mag_niveau0, WKID_RGF93, mapRef);
// Différence entre le point et les autres magasins
Geometry diff_niv0 = geomen.difference(projection_mag_niv0, depart,
mapRef);
// Nous calculons la distance géométrique entre depart et diff_niv
double distance_niv0 = geomen.distance(depart, diff_niv0, mapRef);
```

Pour pouvoir calculer des distances entre magasins nous avons besoin de définir une unité :

```
Unit meter = Unit.create(LinearUnit.Code.METER);
```

Nous définissons ensuite la taille du buffer et nous initialisons la distance du plus proche voisin à un distance d'un kilomètre (distance très grande pour que les tests de recherche du plus proche voisin soit intialiser) :

```
String texte = null;
Geometry mag = null;
int taille = 14;
double dist_ref = 1000;
int color = Color.rgb(255, 1, 1); // Couleur texte d'affichage du nom du
magasin
```

Nous générons ensuite un buffer et nous cherchons le point le plus proche du point de départ se trouvant dans ce buffer. Nous stockons dans mag le ppv (plus proche voisin), pour cela nous appliquons un algorithme bien connu de minimum dès qu'un point est à une distance au point de départ inférieur à la distance minimale précédente au point de départ, alors ce point devient le ppv et sa distance au point de départ devient la distance minimale. Puis nous affichons sur la carte le ppv avec la couleur défini précédemment :

```
if (niveau_dep == 0){
    Polygon buff_niv0 = geomen.buffer(depart, mapRef, distance_niv0,
meter);
    Geometry magasin = geomen.intersect(buff niv0, projection mag niv0,
mapRef);

    // On cherhce le magasin le plus proche
    // c'est-à-dire à la distance minimale du point de départ
    for (int r=0; r<lst_mag_niveau0.size(); r++){
        Geometry mag_niv0_r = geomen.project(mag_niv0_geom[r], WKID_RGF93,
mapRef);
        double dist_mag0 = geomen.distance(mag_niv0_r,magasin, mapRef);
        if (dist_mag0 < dist_ref && dist_mag0!=0){
            texte = lst_mag_niveau0.get(r).toString();
            mag = geomen.project(mag_niv0_geom[r], WKID_RGF93, mapRef);
            dist_ref = dist_mag0;
        }
    }
    // Affichage du ppv :
    if (mag != null) {
        mGraphicsLayer.addGraphic(new Graphic(mag, new TextSymbol(taille,
texte, color)));
    }}
```

### 2.2.4 MainActivity.java : la gestion du formulaire:

Pour gérer le formulaire nous nous servons de la création d'une toolbar et du choix d'un élément dans cette toolbar pour lancer l'activité choix.java liée au formulaire.

#### 2.2.4.a onCreateOptionsMenu(Menu menu) :

Cette méthode permet de gérer la construction de la toolbar :

```
getMenuInflater().inflate(R.menu.menu_main, menu);
return true;
```

#### 2.2.4.b choix() :

Cette fonction sert à remplir l'intent qui sera envoyer à l'activité choix.java, on met dedans l'ensemble des magasins et des types de magasins :

```
Intent intent_choix = new Intent(MainActivity.this, Choix.class);
intent_choix.putExtra("Liste_mag0", lst_mag_niveau0);
intent_choix.putExtra("Liste_mag1", lst_mag_niveau1);
intent_choix.putExtra("Liste_mag2", lst_mag_niveau2);
intent_choix.putExtra("Liste_type0", lst_types_niveau0);
intent_choix.putExtra("Liste_type1", lst_types_niveau1);
intent_choix.putExtra("Liste_type2", lst_types_niveau2);
```

Nous lançon ensuite l'activité choix avec comme paramètre cet Intent :

```
startActivityForResult(intent_choix, 0);
```

#### 2.2.4.c onOptionsItemSelected(MenuItem item) :

Cette méthode lance à l'aide de choix l'activité lorsque que formulaire est choisi dans al toolbar.

### 2.2.5 Présentation du fonctionnement du formulaire :

Nous lançons d'abord Choix.java où nous avons le choix entre trouver un magasin par type pour le départ et l'arrivé, ou par scan de QrCode pour le point de départ.

Le choix par type lance Listetype.java qui affiche la liste des types de magasins, lorsque l'on choisit un type on lance Listemagasin.java qui affiche la liste des magasins de type le type sélectionné. Lorsque qu'un magasin est sélectionné, on détruit l'activité lié à Listemagain.java et on envoie le résultat à l'activité de Listetype.java et de même on renvoie le résultat à l'activité de choix.java. Lorsque l'on clique sur le bouton, on est renvoyé sur MainActivity.java, où si deux points ont été sélectionnés alors l'itinéraire est calculé et affiché à l'aide de la méthode onActivityResult().

### 2.2.6 Choix.java :

#### 2.2.6.a onCreate(Bundle savedInstanceState) :

Dans cette méthode, nous commençons par récupérer les informations que nous avons envoyé depuis MainActivity, ces informations sont placées dans un Intent (voir fonction choix() de MainActivity()). Voici par exemple comment nous récupérons la liste du nom des magasins du niveau 0 :

```
Intent intent_choix = getIntent();
final ArrayList lst_mag0 =
intent_choix.getStringArrayListExtra("Liste_mag0");
```

Cette méthode va ensuite gérer la définistion et l'affichage du texte dans les zones de textes et les boutons défini dans res/layout/choix_magasin.xml, par exemple ici nous affichons le texte par défaut « Magasin de départ » dans le TextView t_dep :

```
t_dep=(TextView)findViewById(R.id.mag_dep);
t_dep.setText("Magasin de départ");
```

Cette méthode va ensuite définir les différents listener sur chacun des boutons :

Tout d'abord nous définissons les listener sur les boutons de choix par type qui envoie un Intent à liste_type contenant les types et les noms des magasins de chaque niveau. Nous vous exposons ici comment on envoie à l'aide du bouton type_dep, la liste des noms et des types des magasisn du premier étage, sachant que pour type_arr cela se passe de la même manière (l'unique modification et le changement du deuxième paramètre de startActivityForResult(), pour le départ on met 3 et pour l'arrivée2 cela nous sera utile lors des tests pour savoir s'il s'agit du départ ou de l'arrivée, il s'agit du requestCode):

```
type_dep.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Nous lançons la liste des types de magasins lors du clic
        Intent intent_type = new Intent(Choix.this, Listetype.class);
        intent_type.putExtra("Liste_mag0", lst_mag0);
        intent_type.putExtra("Liste_type0", lst_type0);
        startActivityForResult(intent_type, 3);}
});
```

Nous associons ensuite au bouton du QrCode l'initialisation du scan de QrCode :

```
qr.setOnClickListener(new View.OnClickListener() {
    //QR code
    @Override
    public void onClick(View v) {
        //QR code
        // Nous lançons le scanner au clic sur notre bouton
        IntentIntegrator integrator = new IntentIntegrator(Choix.this);
        integrator.initiateScan();}
});
```

Nous gérons ensuite l'évènement sur le bouton d'envoie des résultats au MainActivity, pour cela nous mettons les noms des points de départ et d'arrivée dans l'intent que nous envoyons ensuite à MainActivity. Nous appelons enfin la fonction finish() pour détruire l'activité choix.

```
ok.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View v) {
        // Envoie au click des données à mainActivity
        if (mag_dep!= null&& mag_ar!=null ){
            Intent intent_req = new Intent(Choix.this, MainActivity.class);
            intent_req.putExtra("Depart", mag_dep);
            intent_req.putExtra("Arrivee", mag_ar);
            setResult(RESULT_OK, intent_req);
            finish();}}
});
```

#### 2.2.6.b onActivityResult(int requestCode, int resultCode, Intent intent_req) :

Cette méthode commence par gérer le résultat que le scan du QrCode envoie, nous le gérons exactement de la même manière que dans le onActivityResult() de MainActivity.

Cette méthode gère ensuite la récupération et l'affichage du nom du magasin de départ et d'arrivée, pour différencier s'il s'agit du départ et de l'arrivée on se sert du requestCode que nous avons défini dans le onCreate() de choix :2 désigne l'arrivée et 3 le départ . Voici par exemple comment nous traitons l'affichage et le stockage du nom du magasin d'arrivée :

```
if (String.valueOf(requestCode) == String.valueOf(2)) {
    if (String.valueOf(resultCode) == String.valueOf(RESULT_OK)) {
        t_arr.setText(intent_req.getStringExtra("mag"));
        mag_ar = intent_req.getStringExtra("mag");
    }
}
```

### 2.2.7 Listetype.java :

#### 2.2.7.a onCreate(Bundle savedInstanceState) :

Dans cette méthode, nous commençons par récupérer les informations que nous avons envoyé depuis l'activité Choix, ces informations sont placées dans un Intent (voir onCreate() de l'activité Choix). La récupération s'effectue de la même manière que dans le onCreate() de l'activité Choix.

Nous créons ensuite une liste de type général pour l'ensemble des types, car il est plus facile de les afficher ainsi. Voici par exemple l'ajout des types de magasins du niveau0 à cette liste :

```
for (int l=0; l<lst_type0.size(); l++) {
    Object obj = lst_type0.get(l);
    if (!lst_types.contains(obj)) {
        lst_types.add(obj);}
}
```

Nous affichons ensuite l'ensemble des types à l'aide d'une ListView et nous y définissons un Adapter qui nous permettra de connaître la position du type de magasins dans la liste cliqué par l'utilisateur et ainsi de récupérer le type de magasin cliqué.

```
mListView = (ListView) findViewById(R.id.listetype);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(Listetype.this,
        android.R.layout.simple_list_item_1, liste_type);
mListView.setAdapter(adapter);
```

Nous définissons ensuite le listener sur cette ListView : quand un élément est cliqué, nous avons la position de l'élément cliqué dans la liste des types, nous pouvons donc récupérer sa valeur. Nous envoyons au click les informations suivantes à l'activité Listemagasin : le type du magasin recherché ainsi que l'ensemble des types et des noms des magasins. Et nous lançons l'activité Listemagasins. Voici par exemple comment nous transmettons le type sélectionné et la liste des types du niveau 0 :

```
mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {
        Object type = liste_type.get(position);
        Intent intent = new Intent(Listetype.this, Listemagasin.class);
        intent.putExtra("type", type.toString());
        intent.putExtra("Liste_mag0", lst_mag0);
        startActivityForResult(intent, 1);}
});
```

#### 2.2.7.b onActivityResult(int requestCode, int resultCode, Intent intent_req) :

Cette méthode gère ensuite la récupération et l'envoie du nom du magasin sélectionné dans l'activité Listemagasin (cela est très similaire à l'action dans onActivityResult() de l'activité Choix). Nous appelons enfin la fonction finish() pour détruire l'activité Listetype :

```
// Nous renvoyons le magasin choisi à l'activité choix :
if (String.valueOf(requestCode)==String.valueOf(1)) {
    if (String.valueOf(resultCode)==String.valueOf(RESULT_OK)) {
        Intent intent = new Intent(Listetype.this, Choix.class);
        intent.putExtra("mag", intent_req.getStringExtra("mag"));
        setResult(RESULT_OK, intent);
        finish();}
}
```

### 2.2.8 Listemagasin.java :

#### 2.2.8.a onCreate(Bundle savedInstanceState) :

Dans cette méthode, nous commençons par récupérer les informations que nous avons envoyé depuis l'activité Listetype, ces informations sont placées dans un Intent (voir onCreate() de l'activité Choix). La récupération s'effectue de la même manière que dans le onCreate() de l'activité Choix.

Nous récupérons ensuite l'intégralité des magasins ayant pour type le type sélectionné dans ListeType, voici par exemple comment nous récupérons les magasins de type Type pour le niveau 0, les niveaux 1 et 2 étant traité de la même manière :

```
for (int s=0; s<lst_type0.size(); s++) {
    if (lst_type0.get(s).equals(Type)){
        lst_magasin.add(lst_mag0.get(s));}
}
```

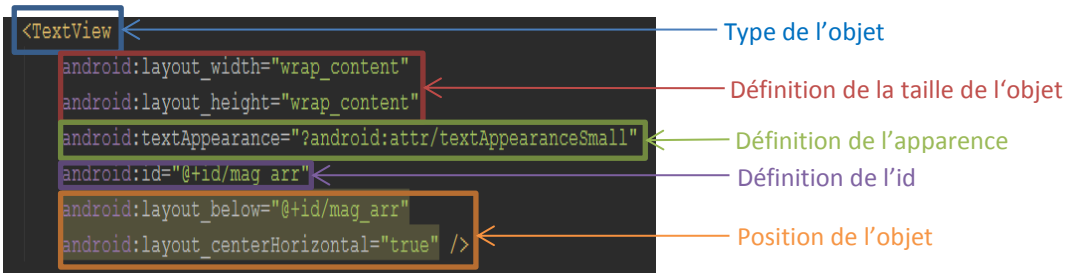
L'affichage de ces magasins s'effectue de la même manière que pour les types dans le onCreate() de Listetype.

Nous définissons ensuite le listener sur cette ListView : quand un élément est cliqué, nous avons la position de l'élément cliqué dans la liste des types, nous pouvons donc récupérer sa valeur. Nous envoyons au click les informations suivantes à l'activité Listetype : le magasin sélectionné. Et nous retournons à l'activité Listetype. Nous appelons enfin la fonction finish() pour détruire l'activité Listemagasin :

```
maliste.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
            Object magasin = liste_magasin.get(position);
            Intent intent = new Intent(Listemagasin.this, Listetype.class);
            intent.putExtra("mag", magasin.toString());
            setResult(RESULT_OK, intent);
            finish();
        }
    });
}
```

## 2.3 Les layout:

Les layout gèrent la partie visuelle d'une application android, les fichiers layout sont écrit en xml qui est un langage facilement lisible et compréhensible par des êtres humains, nous allons vous donner un exemple pour que vous compreniez facilement comment lire du xml :




# 3. Modification du manifeste et du build.gradle :

## 3.1 Le manifeste :

Dans le manifeste nous avons dû autoriser notre application à accéder à différentes fonctionnalités de notre appareil android : la lecture et l'écriture sur la mémoire du téléphone, ainsi que l'accès à internet :

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
<uses-permission android:name="android.permission.INTERNET" />
```

Ayant plusieurs activités, il faut donc bien pensée à toutes les déclarer dans le manifeste de l'application:

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    <activity
        android:name=".MainActivity"
        android:label="@string/app_name"
        android:theme="@style/AppTheme.NoActionBar">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".Listemagasin" />
    <activity android:name=".Listetype" />
    <activity android:name=".Choix" />
</application>
```

## 3.2 build.gradle(app):

Comme nous nous servons de librairies externes, il faut penser à les déclarer dans le build.gradle de l'application, celui qui se situe donc dans le dossier app.

Il faut ajouter dans android{} des packingOptions :

```
android {
    compileSdkVersion 23
    buildToolsVersion "23.0.1"
    defaultConfig {…}

    packagingOptions{
        exclude 'META-INF/LGPL2.1'
        exclude 'META-INF/LICENSE'
        exclude 'META-INF/NOTICE'
        exclude 'LICENSE.txt'
    }
    buildTypes {…}
```

Il faut de plus ajouter toujours dans android{} des dependencies et des repositories, correspondant aux librairies(Arcgis et lecture de QrCode) dont nous nous servons :

```
dependencies {

    compile fileTree(dir: 'libs', include: ['*.jar'])

    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.1.1'
    compile 'com.esri.arcgis.android:arcgis-android:10.2.7' //Pour Arcgis
    compile 'com.android.support:design:23.1.1'

    compile 'com.journeyapps:zxing-android-embedded:3.2.0@aar' //QR code
    compile 'com.google.zxing:core:3.2.1' // QR code

}

repositories {

    jcenter()

    maven{

        url 'https://esri.bintray.com/arcgis' // Pour Arcgis
    }

}
```

# Contact:

Si vous souhaité plus d'information, vous pouvez contacter les différentes personnes ayant participé au projet. Voici leur coordonnée :

Hugo Baltz : hugo.baltz@gmail.com

Elsa Darroman : elsa.darro@gmail.com

Sylvain Jourdana : sylvain.jourdana@gmail.com

Cédric Menut : cedric.menut1805@gmail.com

# Annexe : Liste des variables

| Type : | Nom : | Utilité : |
|---|---|---|
| | | |
| MapView | mMapView; | Objet visuel auquelle on ajoute des layer pour les visualiser |
| | | |
| String | extern | Chemin du lieu de stockage du téléphone |
| String | chTpk | Chemin où se trouve les données |
| | | |
| | | |
| String | tpkPath | Chemin du tpk du graphe |
| String | tpkPath0 | Chemin du tpk de l'étage 0 |
| String | tpkPath1 | Chemin du tpk de l'étage 1 |
| String | tpkPath2 | Chemin du tpk de l'étage 2 |
| | | |
| TiledLayer | mTileLayer | Layer du graphe qui va recevoir le tpk |
| TiledLayer | mTileLayer0 | Layer de l'étage 0 qui va recevoir le tpk |
| TiledLayer | mTileLayer1 | Layer de l'étage 1 qui va recevoir le tpk |
| TiledLayer | mTileLayer2 | Layer de l'étage 2 qui va recevoir le tpk |
| | | |
| GraphicsLayer | mGraphicsLayer | Le layer sur lequelle on dessine |
| | | |
| RouteTask | mRouteTask | Paramètre de l'itinéraire |
| NAFeaturesAsFeature | mStops | On y stock les stop pris en compte dans le calcul d'itinéraire |
| | | |
| | | |
| Drawable | marqueur; | Varaible pour ouvrir l'image correspondant au marqueur du point |
| Symbol | symStop; | Symbole des stops |
| | | |
| | | |
| Spinner | spinnerEtgSel; | Le spinner permmettant le choix des étages |
| boolean | etgsSelected | Vraie si le graphe est selectionné, utile à l'affichage multin-niveau des tpk et de l'itinéaire |
| boolean | etg0Selected | Vraie si l'étgae 0 est selectionné, utile à l'affichage multin-niveau des tpk et de l'itinéaire |
| boolean | etg1Selected | Vraie si l'étgae 1 est selectionné, utile à l'affichage multin-niveau des tpk et de l'itinéaire |
| boolean | etg2Selected | Vraie si l'étgae 2 est selectionné, utile à l'affichage multin-niveau des tpk et de l'itinéaire |
| | | |
| | | |
| Geometry | geom_QR_code | Contient la géométrie récupèrer par le QrCode |
| | | |
| | | |

| | | |
|---|---|---|
| Geometry[] | array_geom_niv0 | Liste de la géométries des arcs du niveau 0 |
| Geometry[] | array_geom_niv1_1 | Liste de la géométries des arcs du niveau 1 (découpé en deux car trop grand nombre d'arcs) |
| Geometry[] | array_geom_niv1_2 | Liste de la géométries des arcs du niveau 1 (découpé en deux car trop grand nombre d'arcs) |
| Geometry[] | array_geom_niv2_1 | Liste de la géométries des arcs du niveau 2 (découpé en deux car trop grand nombre d'arcs) |
| Geometry[] | array_geom_niv2_2 | Liste de la géométries des arcs du niveau 2 (découpé en deux car trop grand nombre d'arcs) |
| Geometry[] | array_geom_niv1 | Regroupement des listes array_geom_niv1_1 et array_geom_niv1_2 |
| Geometry[] | array_geom_niv2 | Regroupement des listes array_geom_niv2_1 et array_geom_niv2_2 |
| | | |
| Geometry | projection_niv0 | Projection dans le système local de array_geom_niv0 |
| Geometry | projection_niv1 | Projection dans le système local de array_geom_niv1 |
| Geometry | projection_niv2 | Projection dans le système local de array_geom_niv2 |
| | | |
| Geometry | geometries_niveau0 | Ensemble géométrique des géométries du niveau 0 |
| Geometry | geometries_niveau1 | Ensemble géométrique des géométries du niveau 1 |
| Geometry | geometries_niveau2 | Ensemble géométrique des géométries du niveau 2 |
| | | |
| Geometry | geom | Géométrie de l'itinéraire entier |
| Geometry | geom_intersect_niv0 | Géométrie de l'itinéraire entier sur le niveau 0 |
| Geometry | geom_intersect_niv1 | Géométrie de l'itinéraire entier sur le niveau 1 |
| Geometry | geom_intersect_niv2 | Géométrie de l'itinéraire entier sur le niveau 2 |
| | | |
| | | |
| GeometryEngine | geomen | Objet utile pour l'analyse spatial |
| | | |
| | | |
| SpatialReference | WKID_RGF93 | Système de référence |
| | | |
| | | |
| | | |
| Feature[] | mag_niv0 | Ensemble des features des magasisn du niveau 0 |
| Feature[] | mag_niv1 | Ensemble des features des magasisn du niveau 1 |
| Feature[] | mag_niv2 | Ensemble des features des magasisn du niveau 2 |
| | | |
| Geometry[] | mag_niv0_geom | Ensemble des géométries des magasisn du niveau 0 |
| Geometry[] | mag_niv1_geom | Ensemble des géométries des magasisn du niveau 1 |
| Geometry[] | mag_niv2_geom | Ensemble des géométries des magasisn du niveau |

| | | |
|---|---|---|
| | | 2 |
| | | |
| Geometry | projection_mag_niv0 | Projection dans le système loclale de mag_niveau0 |
| Geometry | projection_mag_niv1 | Projection dans le système loclale de mag_niveau1 |
| Geometry | projection_mag_niv2 | Projection dans le système loclale de mag_niveau2 |
| | | |
| Geometry | mag_niveau0 | Un élément de mag_niv0_geom |
| Geometry | mag_niveau1 | Un élément de mag_niv1_geom |
| Geometry | mag_niveau2 | Un élément de mag_niv2_geom |
| | | |
| ArrayList | lst_mag_niveau0 | Liste des noms des magasisn du niveau 0 |
| ArrayList | lst_mag_niveau1 | Liste des noms des magasisn du niveau 1 |
| ArrayList | lst_mag_niveau2 | Liste des noms des magasisn du niveau 2 |
| | | |
| ArrayList | lst_types_niveau0 | Liste des noms des magasisn du niveau 0 |
| ArrayList | lst_types_niveau1 | Liste des noms des magasisn du niveau 1 |
| ArrayList | lst_types_niveau2 | Liste des noms des magasisn du niveau 2 |
| | | |
| Geometry | pt_fnac | Géométrie du point correspondant à la fanc utile pour la fnac |
| | | |
| | | |
| int | routeHandle | Augmente de 1 pour chaque itinéraire calculé correctement |
| | | |
| | | |
| CheckBox | checkBoxRes | Objet qui gère la restriction de mobilité |
| boolean | estRestreint | Vraie si la personne est à moblité réduite |
| | | |
| | | |
| List | lst_nom_mag | Liste de l'ensemble des noms des magasins |
| AutoCompleteTextView | textViewArr; | Formulaire d'autocomplétion d'arrivée |
| AutoCompleteTextView | textViewDep; | Formulaire d'autocomplétion de départ |
| | | |
| | | |
| int | niveau_dep | Niveau de l'étage de départ |
| int | niveau_arr | Niveau de l'étage d'arrivée |
| | | |
| | | |
| Geometry | depart; | Géométrie du point de départ |
| Geometry | arrive; | Géométrie du point d'arrivée |