SEE
UNSTABLE
BUILDINGS

DEVELOPER MANUAL

SUB (See Unstable Buildings)

Hugo BALTZ

ENSG Géomatique
ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

uOttawa

8/16/2016

Table of Contents

Important notes:

This application has been developed on Android Studio 2.1.1.

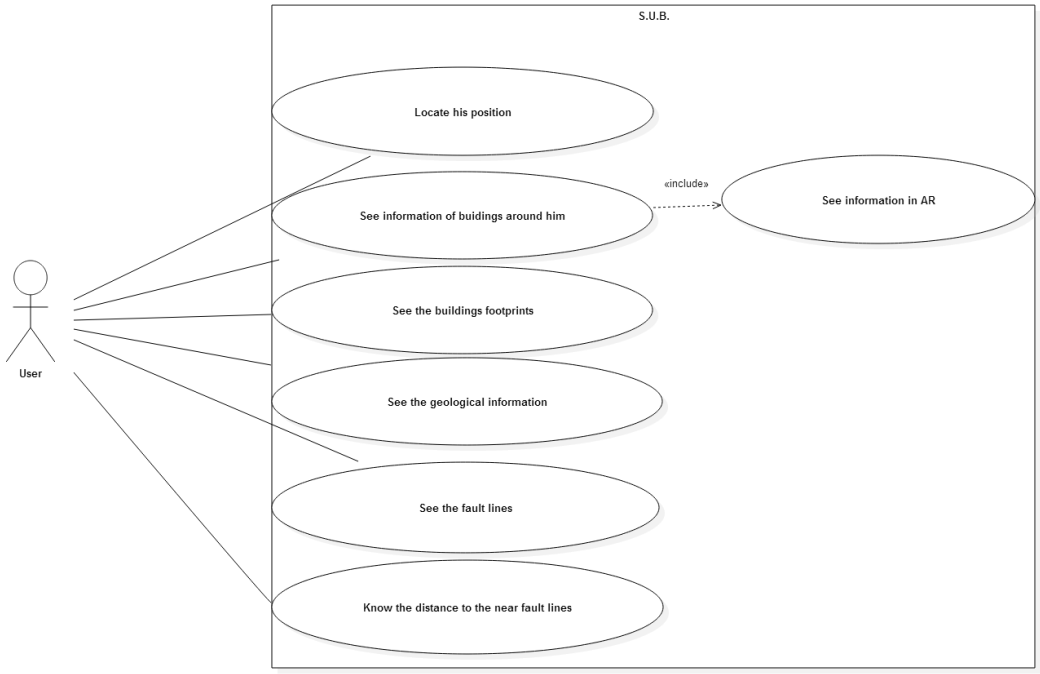
The minimum SDK version to launch the application is 15, and the target SDK is 23.

This application uses the library ArcGIS Runtime SDK for Android 10.2.7.

This application uses the multidex 1.0.1.

Introduction:

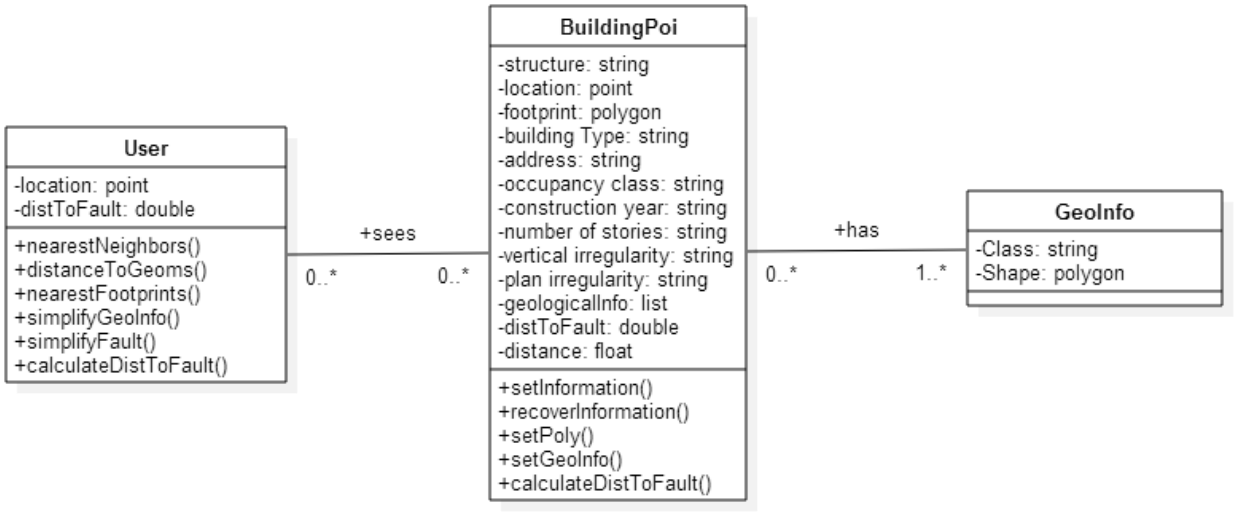
This application displays information about the structure of a building, the geological information, the footprints and the fault lines.



I. Code structure:

The code is divided in three parts:

- MainActivity.java: this file is the activity of the application, this is where we manage the application, we open the database, we create the listener, we manage the views.
- The folder View: each information that we can display on the screen has his own view, this way is more simple to decide what information is on the screen or not (we use the visibility of the views).
- The folder Class: contains the class like describe in the uml diagram and a class Utilities with several useful functions.



For those who wish to have more information about these class who can look the Javadoc of this project, or contact me at hugo.baltz@gmail.com.

II. How the application works:

In this section, I will explain how the different functionalities of the application work.

II.1 Locate the user:

To locate the user, we use a listener on the device's GNSS, when the location changes, the location of the user is updated:

```
public void onLocationChanged(Location location) {
    // We project the latLong in  WGS_1984_WMAS:
    locUser = geomen.project(location.getLongitude(), location.getLatitude(),
WGS_1984_WMAS);

    // We set the location:
    user.setLocation(locUser);
```

Then we project and send these information to the minimap to help the user to locate himself in updateNN():

```
// We update the map:
if(uoMap != null && displayMap) {
    uoMap.setUser(user.getLocation());
    uoMap.invalidate();
}
```

II.2 Open a geodatabase:

We use the SDK Arcgis runtime for Android to manage the geometry and the geodatabase created with ArcMap.

We declare at the beginning where the geodatabase is store on the device:

```
private final String extern = "/storage/sdcard1";
private final String chDb = "/sub";
```

Then in the onCreate() of the MainActivity, we call the function accessDb() which recovers all the information that we need. There an example of how we recover the information about the footprints:

```
// Get the external directory
String networkPath = chDb + "/uo_campus.geodatabase";
// Open a local geodatabase
Geodatabase gdb = new Geodatabase(extern + networkPath);
```

We start by store the features:

```
// Recover features from  db:
GeodatabaseFeatureTable footprints = gdb.getGeodatabaseTables().get(1); // We know
that the first table is the footprints table

long nbr_lig_ft = footprints.getNumberOfFeatures();
int nbr_int_ft = (int) nbr_lig_ft;

// We recover all the features in the gdb:
Feature[] features_footprints = new Feature[nbr_int_ft];

for (int r = 1; r <= nbr_lig_ft; r++) {
    features_footprints[r - 1] = footprints.getFeature(r);
}
```

Then we recover from these features the information that we need:

```
/// Recover Footprints:
// Initialize:
int len1 = features_footprints.length;
PoiFootprints = new Polygon[len1];

Polygon acFoot = new Polygon(); // useful if no object in the db
Feature Footprint;

for (int k = 0; k < len1; k++) {

    Footprint = features_footprints[k];

    // Recover information about buildings :
    if (Footprint != null) {
        PoiFootprints[k]=(Polygon) Footprint.getGeometry();
    } else {
        PoiFootprints[k] = acFoot;
    }
}
```

II.3 The menu:

For more visibility, the user can choose to display only the information that we want to see. So I develop a menu that appears when the user makes a movement from the up to the bottom of the device's screen and disappears with the inverse movement.

The menu's design is located on res/layout/menu.xml and the animations is located in res/anim/.

The menu offers several possibilities:

- See the points of interest (POIs)
- See the footprints
- See the geological information
- See the fault lines
- See the minimap

The menu uses a checkbox for each possibility. The listeners on these checkboxes are define in MainActivity.java. When a checkbox is checked we send to the view that corresponds to the chose possibility, the important information for this view (for example the list of polygon of the nearest footprints to the FtDrawSurfaceView) and the visibility of the view's visibility is set to visible. When the checkbox is unchecked we don't send this information and the view's visibility is set to invisible. For example, how the footprints are managed on the menu:

We start by declare the listener:

```
/**
 * Footprint's Listener
 */
private View.OnClickListener checkedFtListener = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Footprints:
        if(checkFt.isChecked()){
            displayFootprint = true;
            FtDrawView.setVisibility(View.VISIBLE);
```

```
        }else if(!checkFt.isChecked()) {
            displayFootprint = false;
            FtDrawView.setVisibility(View.INVISIBLE);
        }
    }
};
```

In updateView(), the function that updates the views:

```
// We update the display:
if (FtDrawView != null && displayFootprint) {
    FtDrawView.setVariables(NN, orientationVals, user);
    FtDrawView.invalidate();
}
```

II.4 The views:

As I explain to you in the previous paragraph, the application uses a superposition of views that have their visibility sets by the menu.

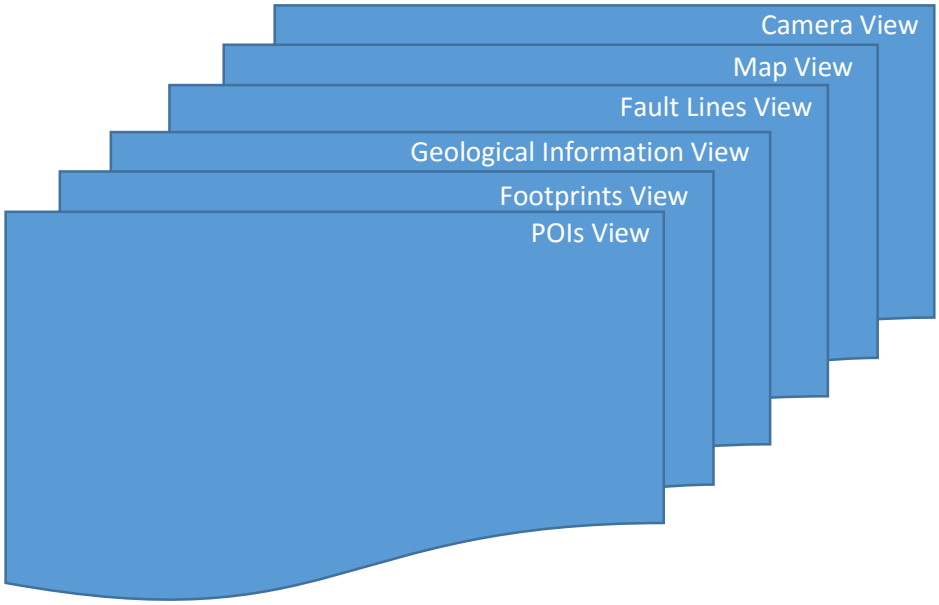


Figure 1. The superposition of views

We update the views when the sensors which give the orientation of the device send new information.

Expect the camera and the map view, we draw on the other views the information in augmented reality.

II.5 Display information about the buildings:

We have already recover the information in the geodatabase (see II.2).

II.5.1 The nearest neighbors(NN):

The MainActivity recovers only the nearest points of interest around the user. Indeed, we don't draw the information about all the POIs in the database for more visibility. To recover only the nearest neighbors (NN), we use the geometry engine of the SDK Runtine Arcgis for Android. The function calls is a method of the class user: nearestNeighbors(). This is how this function works:

```
public ArrayList<BuildingPOI> nearestNeighbors(GeometryEngine geomen,BuildingPOI[]
builds, ArrayList<Polygon> footprints,GeoInfo[] geoInfos, SpatialReference
spaRef,double radius, Unit unit){

    ArrayList<BuildingPOI> NN = new ArrayList<>();

    Point loc = this.getLocation();
```

We define a buffer of *radius unit* around the user:

```
    Geometry buffer = geomen.buffer(loc, spaRef, radius, unit);
```

If a POI intersects the buffer, we store him with these information: the distance to the user, the footprints that corresponds to the POI and the geological information (the type A, B, C, D or E of the geological underground).

```
    for (BuildingPOI build : builds) {
        if (build != null) {
            Geometry locPOI = build.getLocation();
            if (locPOI != null) {
                if (geomen.intersects(buffer, locPOI, spaRef)) {
                    build.setDistance(geomen.distance(loc, build.getLocation(),
spaRef));
                    build.setPoly(geomen,footprints,spaRef);
                    build.setGeoInfo(geomen,geoInfos,spaRef);
                    NN.add(build);
                }
            }
        }
    }

    // Sort (useful when we draw): the closer the bigger when we draw
    Collections.sort(NN);

    return NN;
}
```

This function is call when the location of the device changes.

II.5.2 Draw these information:

The MainActivity send the NN to the DrawSurfaceView that draws the POIs and the information boxes about them. We send these information in the function updateView():

```
// We update the display:
if (DrawView != null && displayPoi) {
    DrawView.setVariables(NN, orientationVals, user);
    DrawView.invalidate(); //Force the onDraw()
}
```

In the DrawSurfaceView, we recover all the NN (for loop), and if the POI isVisible (Boolean fill when the sensor's information change, see the class BuildingPOI), then we project the POI on the screen using a perspective projection (see my report for the theoretical explanation). The function is a method of the class Utilities: positionMatOr(). When the projection is done, we have to apply a translation to change the system coordinates: function positionScreen() in Utilities.

```
// We calculate where the point have to be draw
posScreen = Utilities.positionMatOr(user.getLocation(), POI.getLocation(), orMat, 0f);
posScreen = Utilities.positionScreen(posScreen, (float) screenWidth, (float)
screenHeight);
```

I decide to always draw the point in the middle of the y-axis of the screen.

```
xPosScreen = posScreen.get(0);
// We draw in the middle of the y-axis
yPosScreen = (float) ((screenHeight / 2));
// We calculate the radius of the circle and of the text regarding the distance
radius = (float) (2000 / dist);
////////////////////////////////// Recover info: ////////////////////////
// We recover the information about the Poi that we want to display
strct = POI.getStructure();
information = POI.recoverInformation();
sizeStrings = Utilities.lengths(information); // Useful for the size of the rectangle
```

The color of the POI depends of the structure of the wall:

```
// We initialize the paint regarding the structure field:
paint = initializedPaint(strct);
paint.setTextSize(radius);
```

Then we draw with the canvas the circle and the information box:

```
////////////////////////////////// Draw: ////////////////////////////////
// We draw the circle:
canvas.drawCircle(xPosScreen, yPosScreen, radius, paint);
// We define the size of the rectangle:
w = radius * Collections.max(sizeStrings) / 2;
// We draw the rectangle and the texts:
drawInformation(canvas, information, xPosScreen, yPosScreen, radius, w, paint,
paintRect);
```

The function drawInformation(), draw the information about the building on the canvas enclosed in a rectangle. The rectangle is placed upper the circle of the point:

```
public void drawInformation(Canvas c, ArrayList<String> information,
                            float x, float y, float r, float w, Paint paint, Paint
paintRect){
    int sizeInfo = information.size();
    // We draw the rectangle
    c.drawRect(x-r, y-sizeInfo*(r+1) - r, x + r + w, y-(r+1), paintRect);
    String infoTemp;
    // We draw the texts:
    for(int k=0; k < sizeInfo; k++){
        infoTemp = information.get(sizeInfo-k-1); // To draw the first the array in
the ArrayLIst the higher
        c.drawText(infoTemp, x-r, y-(k+1)*(r+1), paint);
    }
}
```

II.6 Display the footprints:

Like for the POI, to accelerate the calculation, we recover the nearest footprints.

The footprints are associated to the POI, it's a field of the POI, so to draw the footprints, the MainActivty send the NN (Nearest Neighbors see II.5.1) to FtDrawSurfaceView which recovers the footprints and draw them.

As for the POIs, if the POI is visible then we draw the footprints. The footprints is a polygon (com.esri.core.geometry.Polygon). To draw the footprints, we project every point that constituted the polygon with the perspective projection and the coordinate change used on the POIs (see II.5.2). We add each projected point to a path. At the end we draw the path on the screen.

```
// If the POI is visible by the user, we draw his footprint
draw = false;
// We recover the footprint
footprint = POI.getFootprint();
// We count the number of point in the footprint
countPoint = footprint.getPointCount();
//We initialize the path (which will served to draw the footprint)
wallpath = new Path();
wallpath.reset(); // only needed when reusing this path for a new build
// We project each point of the footprint on the screen with a perspective projection
for (int j = 0; j < countPoint; j++) {
    pointTemp = footprint.getPoint(j);
    pos = Utilities.positionMatOr(user.getLocation(), pointTemp, orMatTest, zDef);
    posScreenTemp = Utilities.positionScreen(pos, (float) screenWidth, (float)
screenHeight);
    if (posScreenTemp != null) {
        xPos = posScreenTemp.get(0);
        yPos = posScreenTemp.get(1);

        if (j == 0) {
            wallpath.moveTo(xPos, yPos);
            draw = true;
        } else wallpath.lineTo(xPos, yPos);
    }
}
```

When all the point of the polygon is projected then we draw the path.

```
if (draw) {
    wallpath.close();
    canvas.drawPath(wallpath, paint);
}
```

II.7 Display the geological information:

Like for the POI, it is the MainActivity that send the data to the GeoDrawSurfaceView.

The information about the geological underground is a polygon with a type (from A to E), but as the polygon are huge (one has an area of 2^E7 m^2), we cannot draw the all polygon, so we draw the intersection between the polygon and a buffer around the user. (see the methods simplifyGeoInfo() of the User class). This intersection is a polygon.

So for draw the polygon, we use the same method than the one use to draw the footprints (see II.6).

II.8 Display the fault lines:

Like for the geological information, the polylines of the fault lines are too long, so we simplify them (see the method simplifyFault() of the class User).

The method to draw the fault lines is the same than the one to draw the footprints, expect that in this case we use a line and not a path.

II.9 Distance to the fault lines:

We use the geometry engine of the SDK Arcgis runtime for Android to calculate the distance between the user and the nearest fault lines. We display the result on a TextView in the up-right corner of the screen.

```
displDist.setText("Distance to the nearest fault: " + (int)(user.getDistToFault()) + "
m")
```

Furthermore, we add to the information about the POI, the distance between the POI and the nearest fault lines (this is done in accesDb() when we recover the POI).

III. Notes:

I think that the rest of the code is easily comprehensible, if I am wrong, please contact me by e-mail at hugo.baltz@gmail.com,.