Implementing Pinch zooming with Maps API for Java ME

This article explains how you could pinch zooming support to the map element implemented with Maps API for Java ME API.

Archived: This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

The article is believed to be still valid for the original topic scope.

Introduction

The Maps API for Java ME allows to easily embed Nokia Maps service into Java ME applications, however the API is missing implementation for Pinch zooming. This article illustrates one possible way on implementing on how you could add Pinch zooming to your applications using the Java Me Maps API. Note that this method is designed to work only with Symbian touch screen devices which are supporting multi touch, and will not work on other devices.

Enabling the Multi touch events

To enable to receive multi-touch events you need to set the Nokia-UI-Enhancement JAD attribute to EnableMultiPointTouchEvents. This is illustrated for example in Multipoint touch article. After settings this attribute, and running the application in a device which is supported, you can get the pointer index with `com.nokia.pointer.number system` property. This method exists only in the scope of the methods `pointerPressed()`, `pointerDragged()`, and `pointerReleased()` with `Canvas`. Outside these methods, the returned value is `null`.

General touch event handling

Generally there is two ways on handling touch events, one being on handling them with custom controls as shown in Handling map events with Maps API for Java ME article. However it appears that the `com.nokia.pointer.number` property is not existing on the scope where the touch event handlers are called, thus instead of handling the touch events with the custom control, you must do the handling inside the `MapCanvas` class, which would result `MapCanvas` class which would look something like the following:

```
public class MyMapCanvas extends MapCanvas implements ...{
 ...
protected void pointerPressed(int x, int y) {
        ...}
protected void pointerDragged(int x, int y) {
 ...}
protected void pointerReleased(int x, int y) {
 }
private int getPointerEventId() {
  String idString = System.getProperty("com.nokia.pointer.number");
         int id = -1;
  if (idString != null) {
   id = Integer.parseInt(idString);
  }
  return id;
 }
}
```

The method of calculating the pinch gesture generally depends on the preferences of the designers, with this one, calculations are simplified and the Pinch gesture is simply calculated by the distance variation of the two pointers, and the effect for the gesture is only handled when both pointers are released. If needed you could of course handle the pinch already with draw events.

(C) Copyright Nokia 2013. All rights reserved.

The Pinch gesture recognition is started with `pointerPressed()` method. Within this method the pointer values used in calculations are reset as well as {{Icode|Boolean} helpers indicating which pointers are currently pressed are also set to indicate which ones are pressed but not released yet:

```
protected void pointerPressed(int x, int y) {
 int PId = this.getPointerEventId();
 if(PId == 0){
         lastX1 = startX1 = x;
  lastY1 = StartY1 = y;
  onePres = true;
 }else if(PId == 1){
         lastX2 = startX2 = x;
  lastY2 = StartY2 = y;
  twoPres = true;
 }
 oneRel = twoRel = false;
 super.pointerPressed(x,y);
}
```

With `pointerDragged()` the only thing which would be needed to be handled is the disabling of the panning while the pinch zooming activity is handled. This is handled by not calling the `pointerDragged()` implementation of the base class. And with this example the identification on whether we are having pinch event going on is handed by checking that we have two pointers which are currently pressed, but are not released yet.

```
protected void pointerDragged(int x, int y) {
 if(twoPres && onePres){
         int PId = this.getPointerEventId();
         if(PId == 0){
   lastX1 = x;
   lastY1 = y;
  }else if(PId == 1){
   lastX2 = x;
   lastY2 = y;
             }
 }else{
  super.pointerDragged(x,y);
 }
    }
```

And to finalize the touch event handlings the `pointerReleased()` method needs to be implemented. With this example implementation, we simply mark down the end point where the pointer was released, and then also mark it released and not pressed. Then if both pointers were released, we would finalize the Pinch zooming.

```
protected void pointerReleased(int x, int y) {
 int PId = this.getPointerEventId();
 if(PId == 0){
  lastX1 = x;
  lastY1 = y;
  oneRel = true;
  onePres = false;
 }else if(PId == 1){
  lastX2 = x;
  lastY2 = y;
  twoRel = true;
  twoPres = false;
 }

 if(oneRel && twoRel){
         oneRel = twoRel = false;
             // do the actual Pinch zooming in here
 }
 super.pointerReleased(x,y);
}
```

The Actual Pinch zooming is then handled by calculating the difference of the distances between these two pointers from the pointer press until the pointer release. And use 100 pixels difference as one zoom level change. Before setting the zoom the max/min zoom values are checked as well as the original center is saved, so it can be re-set after the zooming:

```
int Distance1 = (int)Math.sqrt((((startX1 - startX2)* (startX1 - startX2)) + ((StartY1 - StartY2) * (StartY1 - StartY2))));
int Distance2 = (int)Math.sqrt((((lastX1 - lastX2)* (lastX1 - lastX2)) + ((lastY1 - lastY2) * (lastY1 - lastY2))));

int zoomOrg = getMapDisplay().getZoomLevel();
int zoomNew = zoomOrg + (int)((Distance2 - Distance1) / 100);

if(zoomNew > getMapDisplay().getMaxZoomLevel()){
 zoomNew = (int)getMapDisplay().getMaxZoomLevel();
}
if(zoomNew < getMapDisplay().getMinZoomLevel()){
 zoomNew = (int)getMapDisplay().getMinZoomLevel();
}

if(zoomNew != zoomOrg){
 GeoCoordinate orgZenter = getMapDisplay().getCenter();
        getMapDisplay().setZoomLevel(zoomNew, 0, 0);
        getMapDisplay().setCenter(orgZenter);
}
```

Resources

Full source codes for the example can be found from File:JavaMeMaps PinchZoom.zip

Summary

The Java ME Location API offers rich functionalities that allows to integrate all the main Ovi Maps features in a Java ME application, with just a few lines of code.

 (C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.