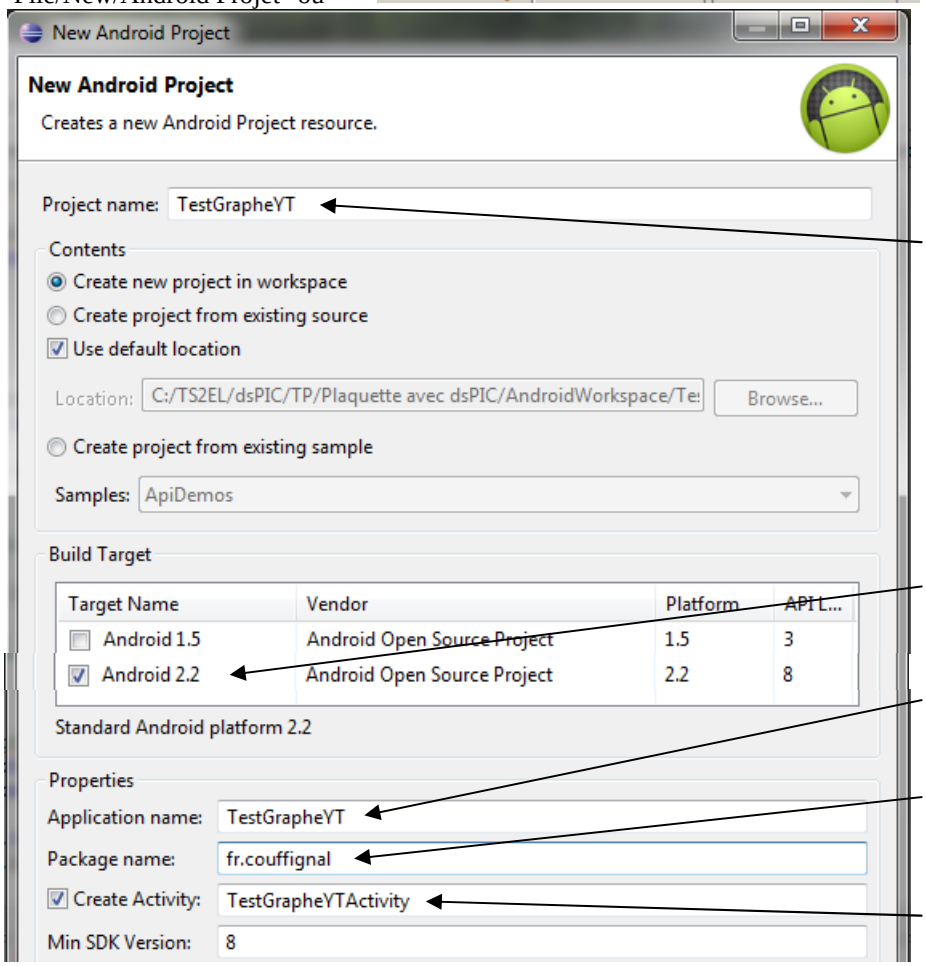
ANDROID & ECLIPSE

Tutorial pour une créer une application de graphe YT de type « Roll »

Avant propos : Eclipse, le plugin Android SDK et un émulateur de tablette AVD sont installés.

1. Créer l'application

- "File/New/Android Projet" ou



Nom du projet. Un dossier à ce nom est créé dans le "workspace"

Version d'Android de la tablette utilisée.

Recopie du nom du projet. Peut être modifié

"fr" et "couffignal" sont 2 dossiers imbriqués dans le dossier du projet dans lesquels seront placés les fichiers du projet.

Nom de la première activité de l'application

Le projet est créé avec "Finish". Il ne comporte pour le moment qu'un seul source : « TestGrapheYTActivity.java ».

2. L'application grapheYT de type « Roll »

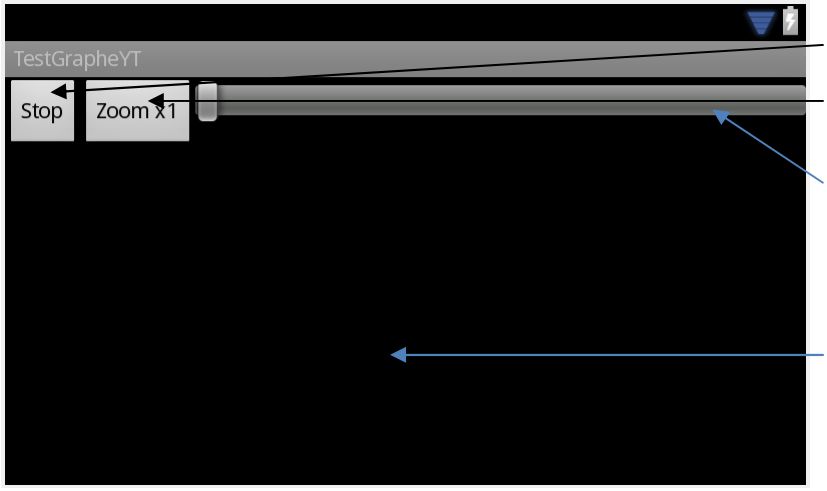
2.1 Description de l'application

Il s'agit de créer un graphe déroulant dont le fonctionnement est analogue à celui d'une table traçante ou d'un oscilloscope en mode « Roll ».
Le signal d'entrée (simulé dans cette application) est affiché sur l'écran comme un stylet fixe sur un papier déroulant. Un bouton « Run/Stop » permet de bloquer et de relancer l'acquisition. Un autre permet de changer l'échelle. Un curseur permet de déplacer dans la mémoire d'acquisition.
Spécification :

- Fréquence d'échantillonnage = 250Hz
- Profondeur mémoire = 2 minutes, soit 30000 échantillons
- Fenêtre d'affichage :
  - zoom x 1 : entre 2s et 3s suivant la résolution de l'écran
  - zoom x 2 : entre 4s et 6s suivant la résolution de l'écran

2.2 Ecran

L'écran est composé avec l'aide de l'éditeur graphique du fichier « res/layout/main.xml » :



Bouton « Run/Stop »

Bouton « Zoom x1/Zoom x2 »

Curseur de déplacement dans la mémoire d'acquisition

Ces 3 premiers objets sont placés dans un « LinearLayout » horizontal

Place réservée pour le graphe YT

Fichier « main.xml » :

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:orientation="vertical"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent"
6      >
7      <LinearLayout android:layout_height="wrap_content"
8                    android:id="@+id/linearLayout1"
9                    android:layout_width="match_parent">
10         <Button    android:text="Stop"
11                   android:id="@+id/btnRunStopId"
12                   android:layout_width="wrap_content"
13                   android:layout_height="wrap_content">
14         </Button>
15         <Button    android:text="Zoom x1"
16                   android:id="@+id/btnZoomId"
17                   android:layout_width="wrap_content"
18                   android:layout_height="wrap_content">
19         </Button>
20         <SeekBar   android:id="@+id/curseurGrapheId"
21                   android:layout_weight="1"
22                   android:layout_height="wrap_content"
23                   android:layout_width="match_parent">
24         </SeekBar>
25     </LinearLayout>
26 </LinearLayout>
```

2.3 Objet « Graphe YT »

Cet objet comporte la définition de la surface graphique, les constantes et variables nécessaires et les méthodes pour l'initialiser et le manipuler.

2.3.1 Création de la classe « GrapheYT.java »

On commence par créer la classe associée à l'objet « GrapheYT », héritée de la classe "SurfaceView" et implémentée de l'interface "SurfaceHolder.Callback" (méthodes de communication des réponses de retour). Pour l'instant elle ne comporte aucun traitement effectif, mais on y place les méthodes nécessaires :

```
1  package fr.couffignal;
2
3  import android.content.Context;
4  import android.util.AttributeSet;
5  import android.view.SurfaceHolder;
6  import android.view.SurfaceView;
7
8  public class GrapheYT extends SurfaceView implements SurfaceHolder.Callback
9  {
10
11  public GrapheYT(Context context, AttributeSet attrs)
12  {
13   super(context, attrs);
14   // TODO Auto-generated constructor stub
15  }
16
17  @Override
18  public void surfaceChanged(SurfaceHolder holder, int format, int width, int height)
19  {
20   // TODO Auto-generated method stub
21  }
22
23  @Override
24  public void surfaceCreated(SurfaceHolder holder)
25  {
26   // TODO Auto-generated method stub
27  }
28
29  @Override
30  public void surfaceDestroyed(SurfaceHolder holder)
31  {
32   // TODO Auto-generated method stub
33  }
34
35 }
```

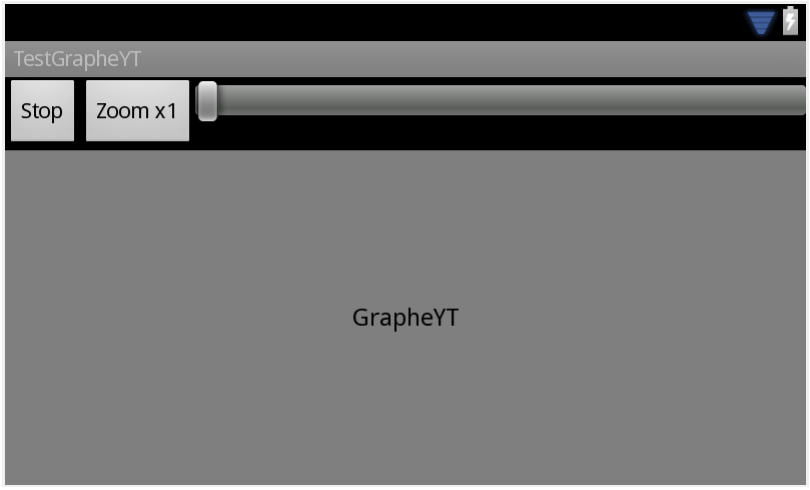
2.3.2 Insertion de la surface graphique dans l'écran

Il faut modifier le fichier « main.xml » comme suit :

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:orientation="vertical"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent"
6      >
7      <LinearLayout android:layout_height="wrap_content"
8                    android:id="@+id/linearLayout1"
9                    android:layout_width="match_parent">
10         <Button   android:text="Stop"
11                   android:id="@+id/btnRunStopId"
12                   android:layout_width="wrap_content"
13                   android:layout_height="wrap_content">
14         </Button>
15         <Button   android:text="Zoom x1"
16                   android:id="@+id/btnZoomId"
17                   android:layout_width="wrap_content"
18                   android:layout_height="wrap_content">
19         </Button>
20         <SeekBar  android:id="@+id/curseurGrapheId"
21                   android:layout_weight="1"
22                   android:layout_height="wrap_content"
23                   android:layout_width="match_parent">
24         </SeekBar>
25     </LinearLayout>
26     <fr.couffignal.GrapheYT
27                 android:src="@android:drawable/screen_background_light"
28                 android:id="@+id/surfaceGrapheYT"
29                 android:layout_width="match_parent"
30                 android:layout_height="match_parent"
31     />
32 </LinearLayout>
```

Ajouter

On obtient l'écran suivant :



La classe « GrapheYT » sera complétée après la description du « thread » de dessin du graphe.

2.4 « Thread » de dessin du graphe

2.4.1 Définitions

Thread : Souvent appelés processus légers, les *threads* peuvent être considérés comme des tâches dans lesquelles s'exécutent certaines fonctions d'une application. Dès lors qu'une application Android démarre, un *thread* est créé (le « *main thread* ») et sert de réceptacle pour les fonctions principales. Pour donner une illusion de parallélisme, il est possible de créer plusieurs *threads* dans une application, le système d'exploitation Android se chargeant de les exécuter de façon séquentielle et rapide d'où l'impression de parallélisme ou de simultanéité des traitements.
Le système d'exploitation Windows utilise un principe analogue, avec ses « processus » et son gestionnaire de tâches.

Looper : Le Thread est la base de la programmation concurrente … Malheureusement utilisé tel quel il n'a que peu d'utilité. En effet, un *thread* « normal » s'arrête dès lors qu'il a fini d'exécuter ses instructions pour donner la main au *thread* suivant. Par exemple, dans notre cas, un thread sera chargé de rafraichir l'affichage. Pour obtenir un affichage fluide, ce thread doit être

exécuté au moins 50 fois par seconde. Si les autres *thread* en cours d'activité sont nombreux, la simple commutation des tâches peut dégrader la qualité de l'affichage.
Pour éviter cela, il est possible de faire en sorte que ce *thread* boucle indéfiniment **en attente** d'autres *threads* à exécuter. On met alors en place un ***Looper*** qu'on appelle parfois également "boucle évènementielle". La classe *Looper* permet de préparer un *thread* à la lecture répétitive d'actions. Un tel *thread*, présenté dans la figure ci-dessous, est souvent appelé *looper thread*. Sous Android, le *main thread* est en réalité un *looper thread*. Un *Looper* étant propre à un unique *Thread*, il est implémenté sous la forme du *design pattern* TLS ou *Thread Local Storage* (les plus curieux pourront jeter un œil à la classe *ThreadLocal* dans la documentation Java ou Android).

Message : Les *Messages* sont les échanges entre les thread. Un *Message* peut être un message au sens propre (du texte ou des données) ou des fonctions exécutables (les *Runnables*).

Handler : Cet objet permet d'interagir avec d'autres *looper threads*. C'est par le biais d'un *Handler* qu'il est possible de poster des *Messages* ou des *Runnables* dans *le Looper* où ils seront traités ou exécutés.

2.4.2 Classe « ThreadGraphYT »

Sa fonction principale est le rafraichissement du graphe déroulant. Ce traitement est réalisé par la méthode "onDraw" de la classe "GrapheYT" et est donc appelée régulièrement par le thread "ThreadGraphYT".

```
1  package fr.couffignal;
2
3  import android.graphics.Canvas;
4  import android.view.SurfaceHolder;
5
6  // Thread utilisé par "GrapheXY"
7  public class ThreadGrapheYT extends Thread
8  {
9   private SurfaceHolder mySurfaceHolder;
10  private GrapheYT myGrapheYT;
11  private boolean runThreadGrapheYT; // Indicateur d'activité du "GrapheYT"
12
13  // Constructeur de l'objet "MyThread"
14  public ThreadGrapheYT(SurfaceHolder pSurfaceHolder, GrapheYT pGrapheYT)
15  {
16   mySurfaceHolder = pSurfaceHolder; // Objet "SurfaceHolder" du GrapheYT
17   myGrapheYT = pGrapheYT;           // Objet "GrapheYT"
18  }
19
20  // "Setter" de la variable "runThreadGrapheYT"
21  public void setRunning(boolean run)
22  {
23   runThreadGrapheYT=run;
24  }
25
26  // Méthode "run" du thread, appelée par le gestionnaire des tâches d'Android
27  @Override
28  public void run()
29  {
30   Canvas c;
31   while (runThreadGrapheYT)
32   {// Tant que le GrapheYT est actif
33    c=null;
34    try {
35         c=mySurfaceHolder.lockCanvas(null); // Le canvas retourné permet
36            // de dessiner sur la surface associée à "mySurfaceHolder",
37            // soit "GrapheYT". Tous les pixels doivent être rafraichis
38         synchronized(mySurfaceHolder) {myGrapheYT.onDraw(c);} // Appel "onDraw"
39        }
40    finally {
41             if (c!=null) {mySurfaceHolder.unlockCanvasAndPost(c);}
42            }
43   }
44  }
45
46 }
```

Attributs du thread :
Interface de maintien de la surface d'affichage
Instance de la classe "GrapheYT"
Indicateur …

Instanciation des classes du thread

Affectation de l'indicateur "runThreadGrapheYT"

La classe "Canvas" comporte toutes les méthodes de dessin.

Classe canvas "c" associée à la surface du "GrapheYT" et préparation de la surface à dessiner

Appel "synchronisé" de la méthode "onDraw"

Affichage de la surface modifiée

Note : on ne maîtrise pas le rythme d'exécution de la méthode "run". Il dépend du nombre de thread actifs et de la puissance de la machine. Ici, il doit être suffisant pour obtenir un affichage fluide.

2.5 Code de l'activité principale : classe "TestGrapheYTActivity"

```
package fr.couffignal;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.SeekBar;

public class TestGrapheYTActivity extends Activity implements OnClickListener
{
 // Déclaration des instances des widgets de l'écran "main"
 Button btnRunStop;
 Button btnZoom;
 GrapheYT monGrapheYT;
 public static SeekBar curseurGrapheYT;
 public static boolean grapheRun = true;

 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState)
 {
  super.onCreate(savedInstanceState);
  // Instanciation de l'objet "monServiceIntent" utilisé pour
  // démarrer "ServiceGrapheYT"
  Intent monServiceIntent=new Intent(this,ServiceGrapheYT.class);
  // Lancer les services de "ServiceGrapheYT"
  startService(monServiceIntent);
  // Affichage de l'écran principal
  setContentView(R.layout.main);
  // Création des instances des boutons, du curseur et du
  // graphe YT de l'écran "main"
  btnRunStop=(Button)findViewById(R.id.btnRunStopId);
  btnZoom=(Button)findViewById(R.id.btnZoomId);
  btnZoom.setText("Zoom x1");
  monGrapheYT=(GrapheYT)findViewById(R.id.surfaceGrapheYT);
  curseurGrapheYT=(SeekBar)findViewById(R.id.curseurGrapheId);
  // "Ecouteur" du bouton Start/Stop
  btnRunStop.setOnClickListener(this);
  // "Ecouteur" du bouton Zoom
  btnZoom.setOnClickListener(this);
  // Initialisation du curseur de position
  curseurGrapheYT.setMax(GrapheYT.maxSample); // Valeur max="maxSample"
  curseurGrapheYT.setProgress(GrapheYT.maxSample); // Valeur actuelle=
 }                                                 // "maxSample"

 public void onClick(View v)
 {
  // Test de la zone cliquée
  if (v==btnRunStop)
  {// Bouton Run/Stop
   if (grapheRun)
   {
    btnRunStop.setText("Run"); // Texte bouton="Run"
    grapheRun=false;           // Indicateur utilisé ds
   }                           // "ServiceGrapheYT"
   else
   {
    btnRunStop.setText("Stop"); // Texte bouton="Stop"
    grapheRun=true;
   }
  }
  if (v==btnZoom)
  {// Bouton de zoom
   if (btnZoom.getText()=="Zoom x1")
   {
       btnZoom.setText("Zoom x2");
       GrapheYT.zoom=2; // Attribut utilisé ds "GrapheYT"
   }
   else
```

Variables publiques car utilisées dans d'autres classes

La classe "ServiceGrapheYT" (voir §2.7) comporte un timer qui est chargé de simuler l'affectation des échantillons dans le buffer.

Le dernier échantillon du buffer correspond au dernier en date. Le curseur au max permet d'afficher le signal acquis, sans délai.

Méthode appelée par un clic sur une position quelconque de l'écran. Il faut donc déterminer la zone pour effectuer le traitement correspondant.

```
    {
        btnZoom.setText("Zoom x1");
        GrapheYT.zoom=1;
    }
   }
  }
 }
}
```

2.6 Code de la classe "GrapheYT"

Le code comprend :

- la déclaration des attributs (ou variables) nécessaires au dessin du graphe déroulant, en particulier le buffer des échantillons.
- les méthodes de gestion du graphe dont notamment "onDraw" qui se charge de dessiner chaque image du graphe déroulant

```
package fr.couffignal;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class GrapheYT extends SurfaceView implements SurfaceHolder.Callback
{
 public static ThreadGrapheYT _thread; // Instanciation de la classe du thread graphique
 public int i,j;
 public Paint paint; // Classe pour définir les styles et les couleurs des dessins
 public static int[] datasBuffer;  // Buffer des échantillons
 public static int largeurPixels;  // Largeur de l'écran en pixels
 public static int hauteurPixels;  // Hauteur de l'écran en pixels
 public int x1, x2, y1, y2;        // Utilisées pour dessiner la courbe sur le graphe
 public static int indexSample;    // Index dans le buffer de l'échantillon représenté à droite de l'écran
 public static int indexGraphe;    // Index dans le buffer de l'échantillon dessiné de droite à gauche
 public static int oldReste1000, oldReste200; // Pour déterminer les périodes 200mS et 1000mS
 public static final int FE = 250; // Fréquence d'échantillonnage = 250Hz
 public static final int maxSample=FE*120; // Profondeur mémoire = 120s = 2mn
 public static int zoom=1;         // Valeurs possibles : 1 ou 2
 public static int posMemStart=0;  // Décalage du pixel de droite en nb d'échantillons (entre -maxSample et 0)

 public GrapheYT(Context context, AttributeSet attrs)
 {
  super(context, attrs);
  datasBuffer = new int[maxSample]; // Instanciation du buffer des échantillons
  getHolder().addCallback(this);    // Ajout de l'interface "Callback" à cette classe
  _thread=new ThreadGrapheYT(getHolder(),this);      // Instanciation du thread "ThreadGrapheYT"
  paint = new Paint();              // Instanciation de la classe "Paint" utilisée
 }

 public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3)
 {
  // TODO Auto-generated method stub
 }

 public void surfaceCreated(SurfaceHolder holder)
 {// Méthode appelée immédiatement après la création de la surface
  _thread.setRunning(true); // Appel du "setter" qui affecte l'attribut "runThreadGrapheYT"
  _thread.start();          // Démarrage du thread
 }

 public void surfaceDestroyed(SurfaceHolder holder)
 {// Méthode appelée juste avant la destruction de la surface
  boolean retry=true;
  _thread.setRunning(false); // Appel du "setter" qui affecte l'attribut "runThreadGrapheYT"
```

Nécessaire pour associer le thread "ThreadGrapheYT"

Constructeur de la classe appelé dans "TestGrapheYTActivity"

Déclaration obligatoire pour éviter une erreur

```
  // Attendre l'arrêt effectif du thread
  while (retry)
  {
   try {
         _thread.join(); // Blocage du thread à la fin du traitement en cours
               retry=false;    // Non exécuté si exeption
        }
   catch (InterruptedException e)
        {}
  }
 }

 /*
  * "onDraw" : appelée par la méthode "run" du thread "ThreadGrapheYT"
  * La méthode est chargée de redessiner complètement la surface :
  *   - repeindre la surface en noir (le fond)
  *   - dessiner la courbe en mode "roll" et en blanc
  *   - dessiner 2 echelles de temps en mode "roll"
  *      - toutes les 200mS : un pointillé vert sur la hauteur de l'écran
  *      - toutes les 1000mS : une droite bleue sur la hauteur de l'écran
  */
 @Override
 protected void onDraw(Canvas canvas)
 {
  largeurPixels=canvas.getWidth();  // Largeur actuelle de l'écran en pixels
  hauteurPixels=canvas.getHeight(); // Hauteur actuelle de l'écran en pixels
  // Calcul du décalage du pixel de droite en nb d'échantillons
  posMemStart=TestGrapheYTActivity.curseurGrapheYT.getProgress()-maxSample;
  if (posMemStart <= (-maxSample+zoom*FE)) posMemStart=-maxSample+zoom*FE;
  // Effacer l'écran
  canvas.drawColor(Color.BLACK);
  // Dessiner la courbe en mode "roll"
  // Index de l'échantillon actuel représenté à l'extrème droite de l'écran
  indexSample=ServiceGrapheYT.writeIndex+posMemStart;
  if (indexSample < 0) indexSample=indexSample+maxSample;
  // Coordonnées du pixel de la courbe à l'extrème droite
  x1=largeurPixels-1; y1=datasBuffer[indexSample];
  // Pour tous les pixels de l'écran et de droite à gauche
  for (indexGraphe=largeurPixels-1; indexGraphe > 0; indexGraphe--)
  {// Dessin de la courbe sur la largeur de l'écran
   // Calcul de l'index circulaire dans "datasBuffer" du prochain pixel
   indexSample=indexSample-zoom;
   if (indexSample < 0) indexSample=indexSample+maxSample;
   // Coordonnées du prochain pixel
   x2=indexGraphe; y2=datasBuffer[indexSample];
   // Détection période 200mS
   if ((indexSample % (FE/5)) > oldReste200)
   {// Traitement toutes les 200mS
    if ((indexSample % FE) > oldReste1000)
        {// Traitement toutes les 1000mS
          paint.setColor(Color.BLUE);
          canvas.drawLine(x2, 0, x2, hauteurPixels, paint);
        }
    else
    {// Traitement toutes les 200mS
     paint.setColor(Color.GREEN);
     for (i=1; i<(hauteurPixels/10); i++)
         {// Dessin d'un pointillé vert vertical sur la hauteur de l'écran
          canvas.drawPoint(x2, hauteurPixels-i*10, paint);
         }
    }
    oldReste1000=indexSample % FE;
   }
   oldReste200=indexSample % (FE/5);
   // Dessin partiel de la courbe en blanc
   paint.setColor(Color.WHITE);
   canvas.drawLine(x1, y1, x2, y2, paint);
   x1=x2; y1=y2;
  }
 }
}
```

Le traitement de la méthode "onDraw" est assez lourd : un processeur rapide sera nécessaire pour obtenir un graphe déroulant fluide. On pourra aussi tester la librairie OpenGL pour Android.

2.7 Classe "ServiceGrapheYT"

Dans une application réelle (électrocardiographe par exemple), le flux d'échantillon provient de l'extérieur. Le module "Bluetooth" est tout indiqué pour cet usage.
Pour tester notre application, on crée une classe "Service" qui va instancier un "Timer". Ce timer va exécuter une tâche de fond périodique qui va simuler l'arrivée des échantillons.
La période choisie est de 4mS, soit un rythme de 250Hz. Cela peut correspondre à la fréquence d'échantillonnage d'une chaîne d'acquisition ECG.

```
package fr.couffignal;

import java.util.Timer;
import java.util.TimerTask;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class ServiceGrapheYT extends Service
{
 // Instanciation de l'objet monTimer de type Timer
 public static Timer monTimer = new Timer();
 // Autres attributs
 public static int writeIndex=0; // Index d'écriture ds le buffer
 public int dataY = 0;           // Valeur simulée de l'échantillon

 // Attention : le timer est démarré dans "onCreate" car la méthode "onStart"
 // est lancée à chaque fois que la tablette est tournée ce qui multiplierait
 // les instances de la classe "TimerTask"
 @Override
 public void onCreate()
 {
  super.onCreate();
  // Configuration du timer :
  //    - association de la classe "myTimerTask"
  //    - période = 4mS, soit rythme = 250Hz
  monTimer.scheduleAtFixedRate(new myTimerTask(), 0, 4);
 }

 // Classe argument de la fonction "monTimer.scheduleAtFixedRate"
 private class myTimerTask extends TimerTask
 {
  @Override
  public void run()
  {// Traitement exécuté à chaque période du timer
   if (!TestGrapheYTActivity.grapheRun) return;  // Ne rien faire si le graphe est stoppé
   dataY++; // Incrémentation de l'échantillon simulée à chaque période du timer
   if (dataY >= GrapheYT.hauteurPixels) dataY=0; // modulo la hauteur de l'écran
    GrapheYT.datasBuffer[writeIndex]=dataY;             // Affectation du buffer
    if (writeIndex==GrapheYT.maxSample-1) writeIndex=0; // Index circulaire
                                     else writeIndex++;
  }
 }

 @Override
 public void onDestroy()
 {
  super.onDestroy();
  // Arrêt du timer
  monTimer.cancel();
 }

 @Override
 public IBinder onBind(Intent arg0)
 {
  return null;
 }
}
```

Ne pas oublier de déclarer ce service dans "AndroidManifest.xml"

Méthode exécutée à chaque période du timer (ici toutes les 4mS)

Note : la courbe simulée est simple (dents de scie) pour obtenir un traitement rapide qui ne perturbera pas le graphe déroulant.

QUELQUES COPIES D'ÉCRAN