Desarrollo de Aplicaciones MindWave en Android y Processing

Juan Hernández García,
Departamento de Arquitectura y Tecnología de Computadores,
Universidad de Granada

4 de diciembre de 2013

Resumen

Guía básica y tutoriales sobre el desarrollo de aplicaciones para *MindWave*.

Índice

1. Introducción **3**

2. MindWave Mobile y ThinkGear **3**

3. Medidas **3**
3.1. Valor de Atención 3
3.2. Valor de Meditación 4
3.3. Fuerza de Parpadeo 4
3.4. Divagación Mental 4
3.5. Bandas EEG 4
3.6. RAW EEG 5
3.7. Pobreza de la Calidad de Señal 5

4. Inicio al Desarrollo de Aplicaciones **5**
4.1. Desarrollo en Android 5
4.1.1. Requisitos Previos 5
4.1.2. MindWave Eye 6
4.1.3. Recursos 12
4.2. Desarrollo en Processing 14
4.2.1. Requisitos Previos 14
4.2.2. MindWave Star 14
4.2.3. Recursos 22

1. Introducción

En esta guía encontrará los conceptos básicos necesarios para comenzar a desarrollar aplicaciones para *MindWave Mobile*. Primero se le introducirán las características del hardware, cómo ponerlo en funcionamiento y emparejarlo con las herramientas de desarrollo. A continuación se comentarán las instrucciones y herramientas necesarias para la programación.

2. MindWave Mobile y ThinkGear

MindWave Mobile es un producto de *NeuroSky* capaz de obtener información cerebral del usuario. Esta información es adecuadamente disgregada y tratada para ser mostrada y utilizada en el desarrollo de aplicaciones.

ThinkGear es la tecnología que se encuentra dentro de cada producto de *NeuroSky* y nos permite dicho tratamiento de las ondas cerebrales. Es el conjunto de:

- Un **sensor** que hace contacto con la frente.
- Una pinza para enganchar en el lóbulo de la oreja, que sirve como **toma de tierra**.
- Un **chip integrado** que procesa todos los datos.

3. Medidas

Como se ha dicho en la sección anterior, *MindWave* es capaz de captar nuestras ondas cerebrales. No obstante, los datos no se almacenan en crudo, ya que el cerebro humano tiene demasiadas variaciones que hace que los datos sin tratamientos sean complejos de analizar. El chip integrado contenido en todos los dispositivos de *NeuroSky* realiza un procesamiento y filtrado de los datos, aplicando algoritmos evolutivos para estabilizarlos y normalizarlos. Realmente no es tan necesario saber qué está haciendo el dispositivo por dentro, sino cómo interpretar los datos que nos devuelve.

Finalmente, añadir que entre las medidas que se nos devuelven, hay también medidas de control que nos indican el estado del dispotivo físico. A continuación se procederá a explicarlas.

3.1. Valor de Atención

Este valor indicará la intesidad de la concentración mental del usuario. Se ve aumentado, por ejemplo, cuando pensamos con fuerza sobre un hecho concreto. Por el contrario, la perdida de concetración, ansiedad, o estrés harán que este valor decaiga.

Rango: se mueve en el rango entero [0,100], donde 0 es ninguna atención, y 100 concentración absoluta.

Actualización: una vez por segundo.

Activación: activa por defecto.

3.2. Valor de Meditación

Este valor indicará la calma o relajación mental del usuario. Esta medida refleja la relajación **mental** del usuario, no la física, aunque la relajación física puede influir en la mental. Este valor se verá aumentado, por ejemplo, al cerrar los ojos y respirar profundamente. Por el contrario, decrementará cuando nos encontremos agitados y/o ansiosos.

Rango: se mueve en el rango entero [0,100], donde 0 es ninguna relajación, y 100 relajación absoluta.

Actualización: una vez por segundo.

Activación: activa por defecto.

3.3. Fuerza de Parpadeo

MindWave Mobile también es capaz de reconocer cuándo parpadeamos y medir la fuerza, o intesidad, del parpadeo (¿Sorprendido?).

Rango: se mueve en el rango entero [1,255], donde 1 es un parpadeo apenas perceptible, y 255 uno realizado con mucha fuerza.

Actualización: cada vez que el usuario parpadea.

Activación: activa por defecto.

3.4. Divagación Mental

Este nivel indica la facilidad de «salto» entre pensamientos no relacionados. Si el usuario cambia rápidamente entre pensamientos sin llegar a centrarse en ellos, este valor será mas alto.

Rango: se mueve en el rango entero [0,10], donde 0 es N/A, y 10 divagación mental extrema.

Actualización: desconocida.

Activación: activa por defecto.

3.5. Bandas EEG

Esta medida muestra la intesidad de nuestras ondas cerebrales divididas en bandas según su frecuencia. Las bandas son delta, theta, alpha, beta, gamma.

Una correcta intepretación de estas bandas puede darnos mucha información acerca del usuario. Es aconsejable manejarlas si se quieren realizar aplicaciones expertas (apliaciones médicas, etc). El desarrollador medio no necesitará hacer uso de ellas, y con las otras medidas (calculadas a partir de las bandas EEG) tendrá más que suficiente.

Rango: cada banda se mueve en un rango según su frecuencia. Consultar documentación específica.

Actualización: al menos una vez por segundo.

Activación: activa por defecto.

3.6. RAW EEG

También se nos ofrece la posibilidad de tratar con las ondas cerebrales en crudo, sin la disgregación anterior. Puede tener utilidad para trabajar aún a más bajo nivel. En un principio tampoco será necesario tener estar medida en cuenta.

3.7. Pobreza de la Calidad de Señal

Esta es una utilísima medida de control. Nos indica la calidad de la comunicación entre *MindWave Mobile* y la aplicación. Esta medida tendrá un valor más bajo cuanto mejor sea la calidad de señal, y mayor cuando encontremos más ruido en la misma.

Rango: rango entero [0,255]. El valor especial 200 indica que no hay contacto con la piel de una persona, es decir, un fallo en el circuito.

Actualización: una vez por segundo.

Activación: activa por defecto.

4. Inicio al Desarrollo de Aplicaciones

En esta sección se introducirá al desarrollo de aplicaciones que hacen uso de los datos suministrados por *MindWave Mobile*, mediante dos tutoriales. El primero de ellos será sobre la tecnología *Android*, y el segundo sobre *Processing*(*Java*based).

En estos tutoriales se asume un control básico de los lenguajes. No deben mirarse como una guía de aprendizaje de los mismos. No obstante, se detallarán los aspectos que puedan generar más conflicto, para facilitar su realización.

4.1. Desarrollo en Android

NeuroSky ofrece soporte directo para el desarrollo en *Android*. A su vez, *Android* suministra el pack completo de desarrollo en esta tecnología desde su web. Estas dos cosas hacen que sea muy sencillo comenzar a desarrollar aplicaciones para dispositivos *Android*.

4.1.1. Requisitos Previos

- **Descargar las herramientas de desarrollo de *NeuroSky* para *Android*:** Developer Tools 3: Android `http://store.neurosky.com/products/developer-tools-3-android`. En el archivo comprimido encontraremos la biblioteca «ThinkGear.jar», que debe ser incluida en nuestro proyecto.
- **Descargar el SDK de *Android*:** Android SDK `https://developer.android.com/sdk/index.html`. El SDK incluye una versión de eclipse lista para comenzar a programar. [1]

[1] Se recomienda instalar con «SDK Manager.exe» la versión 18 de la API. La versión descargada contiene la 19, que tiene algunos problemas con Eclipse

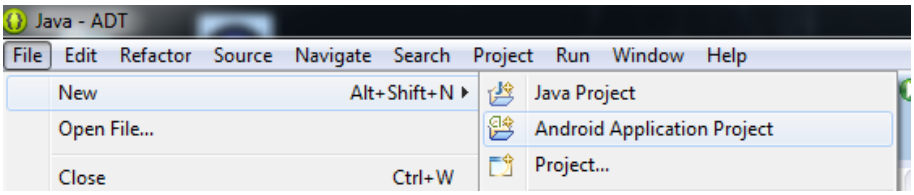


Figura 1: Nuevo Proyecto

4.1.2. MindWave Eye

Descripción MindWave Eye es una aplicación que permite capturar los datos enviados por el dispositivo *MindWave* y representar las variaciones en la meditación y atención mediante datos numéricos, y el parpadeo mediante la iluminación del iris de un ojo (imagen).

Creación del Proyecto Una vez se han cumplido los requisitos previos, abrimos eclipse y creamos un nuevo proyecto *Android*, haciendo:

- **File >new >Android Application Project 1**

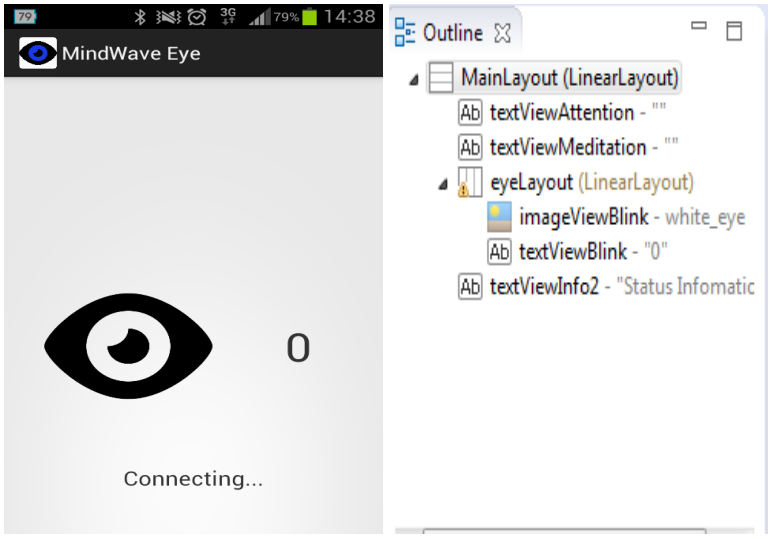
A continuación introducimos como nombre de aplicacion «MindWaveEye», y pulsamos **Next:** hasta que nos solicite introducir el nombre de la actividad principal. La actividad principal suele llamarse igual que el proyecto más el sufijo «Activity». Nombramos a la actividad principal «MindWaveEyeActivity». Finalizamos con la creación del proyecto.

Antes de continuar, debemos asegurarnos de agregar a las bibliotecas de nuestro proyecto **«ThinkGear.jar»**, contenida en el Developer Tools 3: Android.

Interfaz Si buscamos en el subdirectorio de carpetas que se genera con el proyecto, podremos encontrar un archivo llamado **activity_mind_wave_eye.xml** dentro de **MindWaveEye/res/layout**. En este archivo se va a crear la interfaz principal que veremos al iniciar el programa en nuestro dispositivo android.

Una vez abramos el fichero, se iniciará el modo edición gráfico. Procedemos a borrar el **TextView** creado por defecto y nos disponemos a crear una intefaz básica:

- Añadimos un **LinearLayout** (vertical) o refactorizamos el que viene por defecto, lo renombramos como «MainLayout» y pulsando botón derecho sobre él, escogemos las opciones **Layout Height >Match Parent y Layout Width >Match Parent**.
- Dentro de ese **LinearLayout** añadimos tres **TextView**. Para todos ellos hacemos **Layout Width >Match Parent**, de esta manera deben quedar los cuatro abarcando una porción proporcional de la pantalla, uno debajo de otro.
- Renombramos los **TextView** como «TextViewMeditacion», «TextViewAtencion», «TextViewInfo».



(a) Main Layout (b) Outline

Figura 2: Interfaz Gráfica

- Establecemos como texto por defecto para los **TextView**s «Attention: 0» «Meditation: 0» y «Status Info» (o los dejamos como una cadena vacía).
- Dentro también del «MainLayout» creamos otro **LinearLayout**(horizontal) y lo llamamos «eyeLayout».
- En «eyeLayout» añadimos un **ImageView**y un **TextView**.
- Renombramos el nuevo **ImageView** como «ImageViewBlink» y el nuevo **TextView** como «TextViewBlink». [2]
- Añadimos dentro de la carpeta de recursos nuestras imagenes, en este caso «white_eye.png» y «blue_eye.png» y establecemos como predefinida para el **ImageView** «white_eye.png».

Una vez hecho todo lo anterior, la interfaz quedaría como muestra la Figura2

Lógica de la Apliación Ya que tenemos lista la interfaz, podemos pasar a programar la funcionalidad de la aplicación. Abrimos la actividad principal y declaramos las siguientes variables:

```
1 //Declaracion de variables
2 BluetoothAdapter bluetoothAdapter;
```

[2]En *Android* es aconsejable predefinir los strings utilizados, dentro de la carpeta «values». Si no se hace, el entorno nos advertirá mediante warnings

```
3  TextView tMeditation, tAttention, tBlink, tOthers;
4  ImageView iBlink;
5  TGDevice tgDevice;
6
7  final boolean rawEnabled = false;
8  boolean closedEye = false;
```

La primera de ellas será la variable a la que le asignaremos el *bluetooth*. Con los **TextView** e **ImageView** controlaremos los elementos gráficos que hemos definido en la interfaz. Con «tgDevice» interactuaremos con nuestro dispositivo *MindWave Mobile*. Finalmente «closedEye» lo utilizaremos para controlar el parpadeo del ojo. Nos falta por declarar e inicializar el manejador de eventos del dispositivo. Se hará más adelante.

Lo inmediatamente siguiente es completar el método **onCreate**()de la siguiente manera:

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState){
3     super.onCreate(savedInstanceState);
4     setContentView(R.layout.activity_hello_mind_wave);
5
6     //Asociamos los elementos de la interfaz con nuestras
          variables de programa
7     tAttention = (TextView)findViewById(R.id.textViewAttention);
8     tMeditation = (TextView)findViewById(R.id.textViewMeditation
          );
9     tBlink = (TextView)findViewById(R.id.textViewBlink);
10    iBlink = (ImageView)findViewById(R.id.imageViewBlink);
11    tOthers = (TextView)findViewById(R.id.textViewInfo2);
12
13    //Obtenemos el bluethooth por defecto
14    bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
15
16   if(bluetoothAdapter == null) {
17       // Alerta al usuario de que el bluetooth no esta
             disponible
18       Toast.makeText(this, "Bluetooth no disponible", Toast.
             LENGTH_LONG).show();
19       finish();
20       return;
21   }else {
22       //Creamos el TGDevice
23       tgDevice = new TGDevice(bluetoothAdapter, handler);
24   }
25       //Si el dispositivo se encuentra en el estado adecuado,
             entonces conecta.
26       if(tgDevice.getState() != TGDevice.STATE_CONNECTING &&
             tgDevice.getState() != TGDevice.STATE_CONNECTED)
27          tgDevice.connect(rawEnabled);
28 }
```

El método **onCreate()** se ejecuta cuando se inicia la actividad[3]. Suele ser el encargado de asociar variables con elementos de la interfaz, e inicializar todos los valores necesarios para el buen funcionamiento de la aplicación. En nuestro caso podemos ver que, haciendo uso del *bluetooth*, inicializa el dispositivo y lo connecta para que comience la transmisión. También hace uso de «handler», manejador de eventos del dispotivo que definimos más adelante.

El método **onDestroy()**será mucho más simple. Es el encargado de cerrar el dispositivo cuando se destruya la actividad.

```
// onDestroy() es llamado cuando la actividad es destruida.
// Cerramos en el la conexion con el dispositivo.
@Override
public void onDestroy() {
   tgDevice.close();
   super.onDestroy();
}
```

Ahora le toca el turno del ya citado manejador de eventos del dispositivo. Es el encargado de establecer el qué pasará cuando recibamos un mensaje del TGdevice. Su código asociado es el siguiente:

```
/**
   * Manejador de mensajes del TGDevice
   */
private final Handler handler = new Handler() {
   @Override
   public void handleMessage(Message msg) {
      switch (msg.what) {
         case TGDevice.MSG_STATE_CHANGE:
            //Mensajes relacionados con la conexion
            switch (msg.arg1) {
               case TGDevice.STATE_IDLE:
                  break;
               case TGDevice.STATE_CONNECTING:
                  tOthers.setText("Connecting...");
                  break;
               case TGDevice.STATE_CONNECTED:
                  tOthers.setText("Connected.\n");
                  tgDevice.start();
                  break;
               case TGDevice.STATE_NOT_FOUND:
                  tOthers.setText("Can't find\n");
                  break;
               case TGDevice.STATE_NOT_PAIRED:
```

[3]Información sobre el ciclo de vida de la actividad puede ser encontrada en: `http://developer.android.com/reference/android/app/Activity.html`

```
24                      tOthers.setText("not paired\n");
25                      break;
26                  case TGDevice.STATE_DISCONNECTED:
27                      tOthers.setText("Disconnected mang\n");
28              }
29                  break;
30              //Baja senyal
31              case TGDevice.MSG_POOR_SIGNAL:
32                  tOthers.setText("PoorSignal: " + msg.arg1 + "\n");
33                  break;
34              //Mensaje de Atencion
35              case TGDevice.MSG_ATTENTION:
36                  tAttention.setText("Attention: " + msg.arg1 + "\n")
                        ;
37                  break;
38              //Mensaje de Meditacion-Relajacion
39              case TGDevice.MSG_MEDITATION:
40                  tMeditation.setText("Meditation: " + msg.arg1 + "\n
                        ");
41                  break;
42              //Detectado Parpadeo
43              case TGDevice.MSG_BLINK:
44                  iBlink.setImageResource(R.drawable.blue_eye);
45                  tBlink.setText(String.valueOf(msg.arg1));
46                  // Ejecucion de tarea asincrona para refrescar el
                        ojo
47                  new SleepTask().execute(300);
48              break;
49              //Poca Bateria
50              case TGDevice.MSG_LOW_BATTERY:
51                  Toast.makeText(getApplicationContext(), "Low
                        battery!", Toast.LENGTH_SHORT).show();
52              break;
53              default:
54              break;
55          }
56      }
57  };
```

Aquí está implementado el corazón del programa. Como se puede ver (y a pesar de poder parecer aparatoso) es un código simple. Se recibe un mensaje y, mediante un **switch** (con otro anidado) se comprueba qué tipo de mensaje se ha recibido para actuar en consecuencia. El **switch interno**, que comienza en la linea **9**, se encarga de los casos relacionados con la conectividad del dispositivo. El **switch externo** detecta cuándo se recibe una actualización en la atención y/o meditación y actualiza la información de la interfaz. El caso que difiere un poco del resto es el parpadeo. Cuando se recibe un mensaje de parpadeo se hacen tres cosas:

- Se actualiza el **ImageView** con un ojo con iris coloreado.

- Se actualiza el **TextView** que lo acompaña para que indique la fuerza del parpadeo.
- Se crea una tarea asincrona, que espera unos milisegundos y devuelve el ojo a su estado orginal.

¿Por qué necesito crear un tarea (o hebra) asíncrona? En *Android* no se permite que la hebra principal realice preocesos potencialmente bloqueantes que puedan dejar colgada la aplicación. Por lo tanto, si queremos esperar unos segundos (con un **Thread.sleep(milisegundos)**), y después devolver el ojo a su estado (imagen) original, tendremos que hacerlo en segundo plano. El código asociado a la definición de la tarea asíncrona es la siguiente:

```
1  public class SleepTask extends AsyncTask<Integer, Void, Void> {
2     @Override
3     protected Void doInBackground(Integer... params) {
4        try {
5           Thread.sleep(params[0]);
6        } catch (InterruptedException e) {
7           e.printStackTrace();
8        }
9        return null;
10    }
11
12    protected void onPostExecute(Void result) {
13       iBlink.setImageResource(R.drawable.white_eye);
14    }
15 }
```

Un **AsyncTask**[4] tiene dos métodos principales:

- **doInBackground(...)** En él debemos realizar todas aquellas tareas y cálculos potencialmente bloquantes que nos han obligado a dejar la hebra principal.
- **onPostExcute(...)** Este método recibe los datos devueltos por el anterior. En él se actualizarán los elementos de la interfaz que necesiten ser actualizados. **Si se intenta actualizar la interfaz desde doInBackground(...) el programa abortará**.

Como se puede ver, cuando se ejecuta la tarea asíncrona, esta duerme durante el número de milisegundos pasados por parámetro, y pasado ese tiempo se actualiza la imagen del **ImageView**.

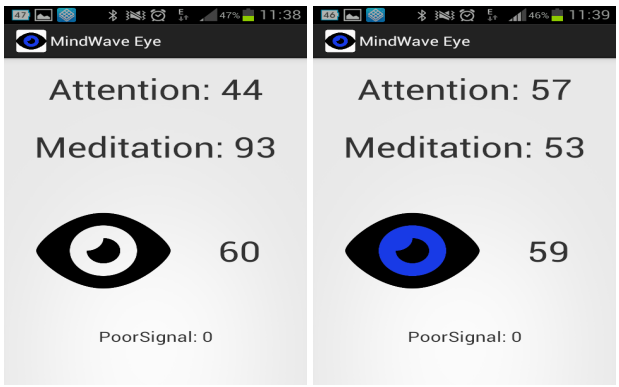
Podríamos pensar que nuestra aplicación está lista, y en parte tendremos razón, pero en *Android* hay algo que nunca podremos olvidar y que, si no tenemos en cuenta, nos dará más de un quebradero de cabeza. El archivo **AndroidManifest.xml**

[4]Puede encontrar toda la información que necesite sobre AsynTask en: `http://developer.android.com/reference/android/os/AsyncTask.html`

Android Manifest [5] Muchas aplicaciones *Android* necesitan tener acceso a recursos del dispositivo como la conexión wifi, el bluetooth, tu lista de contactos, etc. Para que esto sea posible, hay que decir de forma explícita que se quiere acceso al recurso, y esto se hace en **AndroidManifest.xml**. Por lo tanto, ya que hacemos uso en nuestro código del *bluetooth* del dispotivo *Android*, debemos solicitar el permiso añadiendo la siguiente linea:

```
1 <uses-permission android:name="android.permission.BLUETOOTH" />
```

El programa, una vez ejecutado, muestra el aspecto de la figura3



(a) Main Layout (b) Outline

Figura 3: MindWave Eye

4.1.3. Recursos

- **Código fuente:** `https://github.com/juanhg/MindWaveEye`
- **Aplicación:** ¡¡Descárgala desde **PlayStore** de Google!! MindWave Eye
- **Documentación sobre la API:** en el fichero **Development Tools** indicado en el apartado « Requisitos»
- **Eclipse «Intellisense»:** *Eclipse* cuenta con un «Intellisense» que es de gran ayuda para navegar entre funciones y atributos. Actívalo esperándo unos segundos tras poner una sentencia del tipo «TGDevice.» y aparecerá solo. También puedes forzar que aparezca pulsando la combinación de teclas «Ctrl+Space» en cualquier momento.

[5] Puede encontrar toda la información relacionada con AndroidManifest en: `http://developer.android.com/guide/topics/manifest/manifest-intro.html`

4.2. Desarrollo en Processing

Processing es un lenguaje de programación basado en *Java* que hace sencilla la tarea de dibujo de elementos gráficos. *NeuroSky* **no** ofrece soporte directo para el desarrollo en *Java*. Esto se traduce en que **no** ofrece soporte directo para *Processing*. Se nos plantean dos posibilidades:

- Trabajar con los **paquetes «en bruto»**[6] que nos suministra el dispositivo.
- Apoyarnos en alguna biblioteca de terceros que nos ofrezca cierto nivel de abstracción.

Para la realización de este tutorial se opta por la **segunda opción**.

4.2.1. Requisitos Previos

- Descargar e instalar **Processing:** `https://processing.org/download/`
- Desargar e instalar una versión de **Eclipse** con soporte para *Java*: `http://www.eclipse.org/downloads/`
- Instalar **Proclipsing**: `https://code.google.com/p/proclipsing/wiki/GettingStarted`
- Descargar e instalar **ThinkGear Connector**: `http://developer.neurosky.com/docs/doku.php?id=thinkgear_connector_tgc`
- Descargar e instalar **ThinkGear-Java-socket**: `https://github.com/borg/ThinkGear-Java-socket`
- Disponer de un **dispositivo *bluetooth*** habilitado en su equipo de trabajo.

Eclipse es un buen entorno de desarrollo que nos facilitará ciertos aspectos de la programación. *Proclipsing* es un *plugin* para *Processing* en *Eclipse*. *ThinkGear Connector* es necesario para la comunicación del dispositivo con el *bluetooth*. Por último, la biblioteca «ThinkGear-Java-socket» nos suministrará una abstracción de alto nivel para trabajar más cómodamente.

4.2.2. MindWave Star

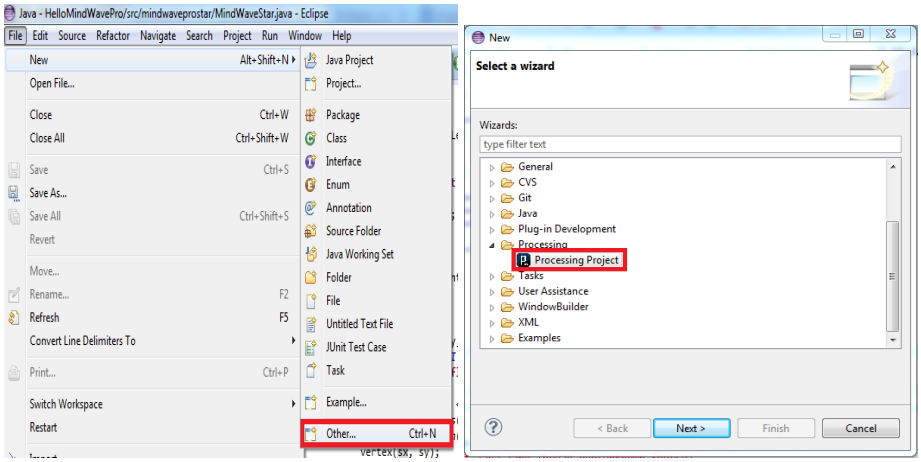
Descripción MindWave Star es una aplicación que permite capturar los datos enviados por el dispositivo *MindWave* y representar las variaciones en la meditación, atención, y parpadeo mediante figuras estrelladas.

[6] Información sobre el protocolo de comunicación de *MindWave*: `http://developer.neurosky.com/docs/doku.php?id=thinkgear_communications_protocol`

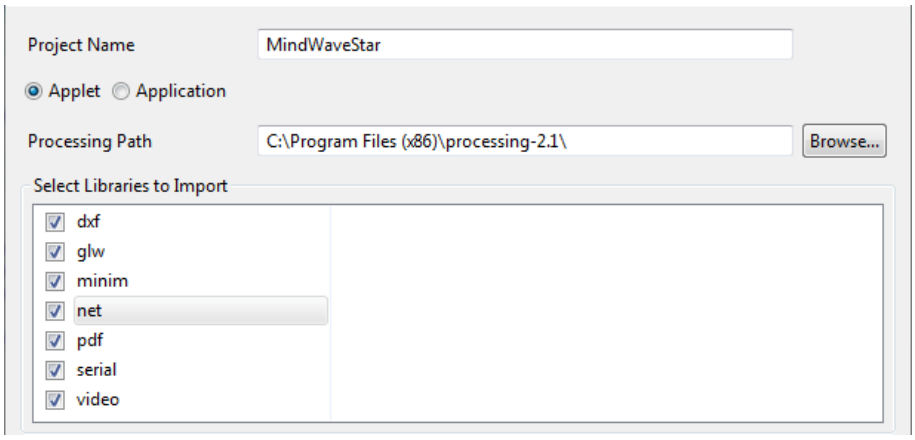
Creación del Proyecto Una vez se han cumplido los requisitos previos, abrimos *Eclipse* y creamos un nuevo proyecto Processing, haciendo:

- File >new >Others >Processing >Processing Project

A continuación pulsamos sobre siguiente, escogemos todas las bibliotecas, ponemos como nombre a nuestro proyecto «MindWaveStar», y pulsamos sobre finalizar. El proceso comentado se muestra en la figura 4



(a) Main Layout (b) Outline



(c) Outline

Figura 4: Nuevo Proyecto Processing

Creamos un **nuevo** paquete en nuestro proyecto que se llame «mindwaveprostar», y en él agregamos una **nueva clase** a la que llamaremos «MindWaveStar». Sobre ella programaremos la lógica de nuestra alicación. Antes de continuar, debemos importar a nuestro proyecto **ThinkGear-Java-socket**.

Código de la Apliación Esta vez, vamos a crear los gráficos y la lógica de la aplicación en el mismo documento, no por separado como hacíamos en *Android*. Processing trabaja con clases de *Java*, por lo que su sintáxis no debería ser un problema. *Processing* tiene dos métodos principales:

- **void setup():** inicializa los elementos de la aplicación o applet.
- **void draw():** bucle de dibujo que se invoca continua y automáticamente.

Con estos dos elementos se pueden conseguir aplicaciones muy vistosas, de una manera bastante simple.

Pasemos a ver el código.

```
1  package mindwaveprostar;
2
3  import java.net.ConnectException;
4
5  import processing.core.PApplet;
6  import processing.core.PFont;
7  import processing.video.Capture;
8
9  import neurosky.*;
10
11 public class MindWaveStar extends PApplet {
12 //Codigo de la clase
13 }
```

Se han de realizar los **import**s especificados, que contienen todas las funciones con las que vamos a trabajar. Por defecto, al crear el proyecto, deben haberse auto-incluido la mayoría. Dentro de la clase, donde aparece el comentario «Código de la clase», es donde introduciremos todo el resto del código de la aplicación. Lo primero es declarar las variables que necesitamos.

```
1  public class MindWaveStar extends PApplet {
2
3     //Declaracion de variables
4     ThinkGearSocket neuroSocket;
5     //Medidas
6     int attention = 0;
7     int meditation = 0;
8     int blinkSt = 0;
9     int poorSignal = 0;
10    int blink = 0;
11    //Fuente de letra
12    PFont font;
13    ...
14    ...
15    ...
16 }
```

Los nombres de las variables son auto-explicativos, y cumplen las mismas funciones que en el tutorial en *Android*. Lo inmediatamente siguiente será definir los eventos de captación de datos del dispositivo.

```
1
2     //Se ejecutan al recibir un mensaje del dispositivo
3     public void captureEvent(Capture _c)
4     {
5       _c.read();
6     }
7
8     public void attentionEvent(int attentionLevel)
9     {
10      attention = attentionLevel;
11    }
12
13    public void meditationEvent(int meditationLevel)
14    {
15      meditation = meditationLevel;
16    }
17
18    public void blinkEvent(int blinkStrength)
19    {
20      blinkSt = blinkStrength;
21      blink = 1;
22    }
23
24    public void poorSignalEvent(int signalLevel){
25       poorSignal = signalLevel;
26    }
27
28    //Eventos vacios.
29    public void eegEvent(int delta, int theta, int low_alpha,
          int high_alpha, int low_beta, int high_beta, int
          low_gamma, int mid_gamma){}
30    public void rawEvent(int [] valor){}
```

Estos eventos vienen definidos por la biblioteca de terceros que estamos utilizando. Se puede ver que se tiene un evento por cada una de las medidas que queremos capturar, y que asignan la medida a las variables que hemos definido en el paso previo. Aunque no vamos a utilizar las medidas eeg, se incluyen los **eegEvent(...)** y **rawEvent(...)** vacíos para que la biblioteca en cuestión no nos lance un mensaje de error advirtiéndonos de que no están definidos.

Vamos a pintar estrellas, así que definamos la función que se encargará de, dados unos sencillos parámetros, dibujarlas.

```
1   /**
2       * Encargada de dibujar estrellas
```

```
     * @param x Posicion espacial x
     * @param y Posicion espacial y
     * @param radius1 Primer radio
     * @param radius2 Segundo radio
     * @param npoints Numero de puntas
     */
    void star(float x, float y, float radius1, float radius2,
        int npoints) {
       float angle = TWO_PI / npoints;
       float halfAngle = (float) (angle/2.0);
       beginShape();
       for (float a = 0; a < TWO_PI; a += angle) {
          float sx = x + cos(a) * radius2;
          float sy = y + sin(a) * radius2;
          vertex(sx, sy);
          sx = x + cos(a+halfAngle) * radius1;
          sy = y + sin(a+halfAngle) * radius1;
          vertex(sx, sy);
       }
       endShape(CLOSE);
    }
```

Es un método de dibujo geométrico simple. Dentro del bucle **for** va creando los vertices, y con las funciones **beginShape()** y **endShape(CLOSE)** indica que quiere rellenar el contenido encerrado entre esos vertices.

Ya tenemos todo lo que necesitamos para crear los dos métodos principales de *Processing*. Comencemos por **setup()**:

```
public void setup()
   {
     //Establecemos el tamanyo de la aplicacion
     size(640, 360);

     //Creamos el nuevo socket
     ThinkGearSocket neuroSocket = new ThinkGearSocket(this);
     try
     {
       //Comenzamos la conexion
       neuroSocket.start();
     }
     catch (ConnectException e) {
       e.printStackTrace();
     }

     //Para dibujar suavizados los elementos geometricos
     smooth();
     //Cargamos fuente de letra
     font = loadFont("D:\\EclipseWorkSpace\\Processing\\
         HelloMindWavePro\\data\\TimesNewRomanPSMT-20.vlw");
```

```
21        textFont(font);
22        //Indicamos los frames por segundo
23        frameRate(30);
24        //Deshabilitamos el dibujado de bordes
25        noStroke();
26      }
```

En él se inicializan todos los elementos básicos. El código comentado es auto-explicativos. Ya que estamos abriendo una conexión con el dispositivo, deberíamos cerrarla cuando la aplicación termine. Podemos hacerlo creando el siguiente método:

```
1       //Sera invocado cuando se destruya la aplicacion
2       public void stop() {
3        neuroSocket.stop();
4        super.stop();
5       }
```

Por último, queda completar el método **draw()**, que será el corazón de nuestra aplicación. Su código asociado es el siguiente:

```
1   //Bucle de dibujo
2      public void draw()
3      {
4        //Fondo negro
5        background(0);
6
7        //Calculos para correcta visualizacion y eliminacion
8        //del valor del parpadeo
9        if (blink>0)
10       {
11         if (blink>15)
12         {
13           blink = 0;
14         }
15         else
16         {
17           blink++;
18         }
19       }
20       else{
21          blinkSt = 0;
22       }
23
24       //Mostramos los valores por pantalla
25       //con color blanco.
26       fill(255, 255, 255);
27       text("Attention: "+attention, 75, 60);
28       text("Meditation: "+meditation, 275, 60);
29       text("Blink: " + blinkSt, 480, 60);
```

```
30        text("PoorSignal: " + poorSignal, 275, 300);
31
32
33        //Dibujamos una estrella de tres puntas
34        //de color rojo
35        fill(255, 0, 0);
36        pushMatrix();
37        translate((float)(width*0.2),(float)(height*0.5));
38        rotate((float) (frameCount / 200.0));
39        star(0, 0, 5, 40, 3);
40        popMatrix();
41
42        //Dibujamos una estrella de veinte puntas
43        //de color azul
44        fill(0, 0, 255);
45        pushMatrix();
46        translate((float)(width*0.5),(float)(height*0.5));
47        rotate((float) (frameCount / 50.0));
48        star(0, 0,20, 40, 20);
49        popMatrix();
50
51        //Dibujamos una estrella de cinco puntas
52        //de color amarillo
53        fill(255, 255, 0);
54        pushMatrix();
55        translate((float)(width*0.8),(float)(height*0.5));
56        rotate((float) (frameCount / -100.0));
57        star(0, 0, 10, 40, 5);
58        popMatrix();
59
60        //Dibujamos las sombras, con los mismos colores
61        //que las estrellas originales pero transparentes
62
63        fill(255, 0, 0,160);
64        pushMatrix();
65        translate((float)(width*0.2),(float)(height*0.5));
66        rotate((float) (frameCount / 200.0));
67        star(0, 0, 5 + attention/2, 40 + attention/2, 3);
68        popMatrix();
69
70        fill(0, 0, 255, 160);
71        pushMatrix();
72        translate((float)(width*0.5),(float)(height*0.5));
73        rotate((float) (frameCount / 50.0));
74        star(0, 0,20 + meditation/2, 40 + meditation/2, 20);
75        popMatrix();
76
77        fill(255, 255, 0, 160);
78        pushMatrix();
79        translate((float)(width*0.8), (float)(height*0.5));
```

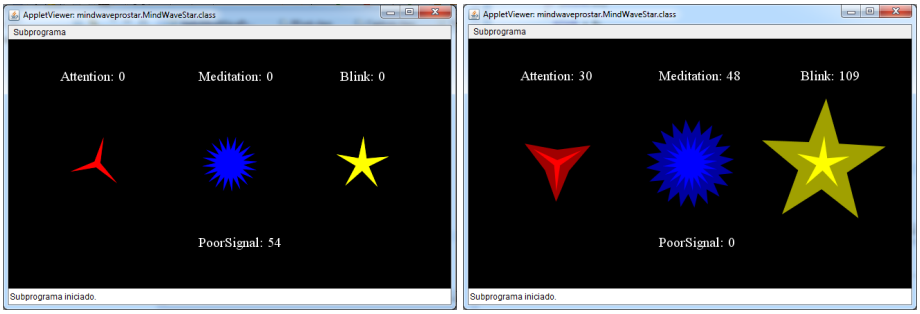
```
80          rotate((float) (frameCount / -100.0));
81          star(0, 0, 10 + blinkSt/4, 40 + blinkSt/2, 5);
82          popMatrix();
83        }
```

Hagamos unos pequeños comentarios:

- El método **fill(...)**[7] se encarga de establecer el color y la transparencia del los elementos que se van a pintar después de ser invocado.
- **translate(...)**[8] y **rotate(...)**[9] establecen las transformaciones de translación y rotación aplicadas al elemento dibujado. Su orden es importante. Si has trabajado previamente con *OpenGL* u otras bibliotecas gráficas estarás acostumbrado a esto.
- **pushMatrix()**[10] y **popMatrix()**[11] meten y sacan las transformaciones en la pila. Este proceso es necesario para la correcta aplicación de las transformaciones

Con estos métodos se crean tres estrellas de distintos colores no ligadas a las medidas. Posteriormente se superponen a ellas estrellas transparentes que varían en función de la medida pertinente. También se muestra mediante texto las medidas recibidas.

El programa, una vez ejecutado, muestra el aspecto de la figura5



(a) Sin Recibir Datos (b) Recibiendo Datos

Figura 5: MindWaveStar

[7] fill(...): `http://processing.org/reference/fill_.html`
[8] translate(...): `http://processing.org/reference/translate_.html`
[9] rotate(...): `http://processing.org/reference/rotate_.html`
[10] pushMatrix(): `http://processing.org/reference/pushMatrix_.html`
[11] popMatrix(): `http://processing.org/reference/popMatrix_.html`

4.2.3. Recursos

- **Código fuente:** `https://github.com/juanhg/MindWaveStar`
- **Documentación sobre *Processing*:** `http://processing.org/`
- **Eclipse «Intellisense»:** *Eclipse* cuenta con un «Intellisense» que es de gran ayuda para navegar entre funciones y atributos. Actívalo esperándote unos segundos tras poner una sentencia del tipo «TGDevice.» y aparecerá solo. También puedes forzar que aparezca pulsando la combinación de teclas «Ctrl+Space» en cualquier momento.