Tutorial Arduino parte 4

Hola a todos .

Este es el cuarto tutorial sobre Arduino , en este tutorial vamos a continuar el tutorial 3 , creando una aplicación 3d para Android , que encienda la bombilla de forma remota .

Creamos un proyecto Android nuevo en eclipse con Android SDK.

Continuando como teníamos en el tutorial 3 , en la parte del servidor php , el fichero que recibía una variable GET y encendía o apagaba la bombilla si la variable es high o low.

En este primer código podemos ver como desde el proyecto Android podemos realizar la conexión con el servidor PHP para que envíe la señal a la placa Arduino.

El fichero PHP ha de estar en modo servicio , con las lineas //die… descomentadas , para que nos devuelva una respuesta.

```
private void lighting()
{
ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
if (cm != null) {
boolean connected=false;

NetworkInfo[] info = cm.getAllNetworkInfo();
if (info != null) {
for (int i = 0; i < info.length; i++) { if (info[i].getState() == NetworkInfo.State.CONNECTED) { connected = true; } } } if(connected) { String destiny="http://192.168.0.131/arduino/service.php?check=on"; String missatxe="Signal CHECK sended. Click again to switch ON"; cambiarTextoBoton("Switch ON"); switch(estado) { case 0: destiny ="http://192.168.0.131/arduino/service.php?hight=on"; missatxe="Signal ON sended. Click again to switch OFF";
```

```
cambiarTextoBoton("Switch OFF"); break; case 1: destiny ="http://192.168.0.131/arduino/service.php?low=on"; missatxe="Signal OFF sended. Click again to switch ON"; cambiarTextoBoton("Switch ON"); break; } escribir(missatxe); InputStream is = null; //initialize String result = ""; try{ HttpClient httpclient = new DefaultHttpClient(); HttpPost httppost = new HttpPost(destiny); HttpResponse response = httpclient.execute(httppost); HttpEntity entity = response.getEntity(); is = entity.getContent(); //convert response to string try{ BufferedReader reader = new BufferedReader(new InputStreamReader(is,"UTF-8"),8); StringBuilder sb = new StringBuilder(); String line = null; while ((line = reader.readLine()) != null) { sb.append(line + "\n"); } is.close(); result=sb.toString(); ArrayList message=this.parse( result);
String status=message.get(0).toString();
String signal=message.get(1).toString();
String mensage=message.get(2).toString();
if(signal.compareTo("1")==0)
{
checkOK=1;
estado=0;
}
else
{
if(signal.compareTo("2")==0)
{
checkOK=2;
estado=1;
}
else
{
if(signal.compareTo("-1")==0)
{
checkOK=-1;
estado=-1;
}
```

```
}

}
}catch(Exception e){
Log.e("log_tag", "Error converting result "+e.toString());
escribir("Error converting result "+e.toString());
}
}catch(Exception e){
Log.e("log_tag", "Error in http connection "+e.toString());
escribir("Error in http connection "+e.toString());
}
}
}

}
```

Bien , ya podemos comunicarnos con el servidor , pero bueno un poco rancia la aplicación , no?

¿Falta un botón , un poco de color o algo no?

Puesto a investigar un poco pués encontré una librería que permitía crear aplicaciones OPENGL de manera un poco más sencilla de lo normal , con un lenguage más para un principiante como yo que ir directamente a OPENGL de Android , que es un poco más rudo para alguien que no lo usa muy a menudo.

Esa librería es Min3D.

Min3D nos permite crear escenas 3D e importar objetos 3DS o OBJ , así como texturas y objetos básicos como esferas y cubos.

Tras descargar min3d de su repositorio , incluimos la carpeta al proyecto y podemos ver y probar los ejemplos.

Partiendo del ejemplo ExampleLoadObjFileMultiple.java , donde podemos ver como cargar un objeto desde un fichero .OBJ .

En este fichero podemos ver que hereda de la clase extends RendererActivity , y lo aplicaremos a la clase que estemos utilizando .

El método initScene() es el encargado de crear la escena , en este método es donde colocaremos todos los objetos de la escena , cargaremos todos los ficheros necesarios , .OBJ , .PNG , y crearemos las esferas y rectángulos que necesitemos.

Este método es llamado al iniciar la aplicación y cada vez que el dispositivo entra en modo PAUSE , o se bloquea la pantalla.

Como en toda aplicación 3D , además de una escena necesitamos una cámara y luces para iluminar nuestros objetos.

En este método asignaremos un color de fondo , o una textura , situaremos las luces , los objetos y situaremos y enfocaremos la cámara , para obtener la perspectiva adecuada.

El método updateScene() se ejecuta en cada frame , sería el equivalente al típico evento draw() de repintado de la pantalla en aplicaciones 2D , osea que se ejecuta cada vez que se pinta la pantalla.

Éste método es el encargado de las instrucciones de las animaciones , por ejemplo si queremos rotar un objeto , en este método calculamos el valor nuevo de la rotación y se le asigna.

Para cargar los ficheros .OBJ en la aplicación Android es necesario modificar los nombres de los archivos.

Los ficheros .OBJ suelen ir acompañados de un fichero con el mismo nombre con extensión .mtl con la definición de los materiales de los grupos de los objetos que hayan definidos en el fichero .OBJ , además de las imágenes de las texturas.

Osea que el fichero .OBJ tiene objetos , valores de vértices y objetos y el fichero .mtl los materiales.

Las imágenes de las texturas las colocaremos en la carpeta res/drawable… .

Los ficheros .OBJ y .MTL los renombramos sustituyendo el punto por un «_» osea un guión bajo , para evitar los conocidos conflictos de ficheros con el mismo nombre y diferente extensión que tiene la programación con Android.

Para este tutorial hemos creado una farola , con una esfera que hace de bombilla , y una esfera con una textura con transparencia .PNG , que hará el efecto de que la farola está encendida.

También se ha creado una especie de bullofa que emite unas esfera con transparencia , emulando el envío de ondas , dando a entender que se está comunicando con el servidor , una esfera nos indicará según la textura que tenga el texto ON de color verde , o el texto OFF de color roja o de color ambar si no hay conexión con el servidor ,a además hay otra bola que nos indicará según su textura , si estamos a más de cierta distancia de la bombilla , si hay cobertura GPS .

Bueno abrimos blender y creamos una lámpara y una bullofa (algo que de el efecto de ser un emisor de ondas) , y las exportamos en formato wavefront .OBJ .

Si no queremos utilizar blender o otro software para crear objetos 3D , los podemos descargar de alguna página , por ejemplo ésta , que ofrezcan objetos 3D en formato .OBJ .

Podemos ver como asignar el color de fondo:

```
scene.backgroundColor().setAll(0xfff2d533);
```

Asignar una textura a un rectángulo:

```
Bitmap b = Utils.makeBitmapFromResourceId(this, R.drawable.scuraki);
float w = 20f;
float h = w * (float)b.getHeight() / (float)b.getWidth();
Rectangle suelo = new Rectangle(w, h, 1,1, new Color4());
```

```
suelo.doubleSidedEnabled(true); // ... so that the back of the
plane is visible
suelo.normalsEnabled(false);
scene.addChild(suelo);

Shared.textureManager().addTextureId(b, "scuraki", false);
suelo.textures().addById("scuraki");

Asignar una textura a una esfera:
b = Utils.makeBitmapFromResourceId(R.drawable.bolaroja);
Shared.textureManager().addTextureId(b, "bolaroja", false);
bolarojaTexture = new TextureVo("bolaroja");
esfera.textures().addReplace(bolarojaTexture);

Iluminar la escena:
luzobj = new Light();
luzobj.ambient.setAll(new Color4 (128,128,128,128));
luzobj.diffuse.setAll(new Color4 (64,64,164, 128));
luzobj.emissive.setAll(new Color4 (0,0,0,255));
luzobj.specular.setAll(new Color4 (0,0,0,255));
luzobj.type(LightType.POSITIONAL);
scene.lights().add(luzobj);
luzobj.position.setAll(0.65f, -0.85f, 3.5f);

Añadir un objeto .OBJ a la escena:
parser2 = Parser.createParser(Parser.Type.OBJ, getResources(),
"adictosalainformatica.min3DAdictos:raw/fanal_obj", true);

parser2.parse();

fanal = parser2.getParsedObject();
fanal.scale().y = 0.25f;
fanal.scale().z = 0.25f;
fanal.scale().x = 0.25f;
fanal.shadeModel(ShadeModel.SMOOTH);
fanal.vertexColorsEnabled(true);
fanal.normalsEnabled(true);
fanal.colorMaterialEnabled(false);
```

```
scene.addChild(fanal);
fanal.position().x=0.0f;
fanal.position().y=1.6f;
fanal.position().z=-5f;
fanal.rotation().x=25f;
```

Controlar la rotación de los objetos en el método updateScene:

```
bombilla.rotation().y=yrot;
bombilla.rotation().x=xrot;
receptor.rotation().y=yrot;
receptor.rotation().x=xrot;

xrot += xspeed;
yrot += yspeed;
_count++;
```

Para dar un toque de color la bullofa cambia el valor de escalado cada cierto tiempo , y se crean unas esferas con transparencia , que viajan desde la bullofa (que está cerca de la cámara) hasta la lámpara (que está al fondo de la escena), emulando el envio de ondas.

Un array de esferas ameniza la pantalla , desplazandose .

La clase Burbuja es la encargada de controlar estas esferas:

```
package adictosalainformatica.min3DAdictos;

import android.graphics.Bitmap;
import min3d.Shared;
import min3d.Utils;
import min3d.core.Scene;
import min3d.objectPrimitives.Sphere;
import min3d.vos.Color4;
import min3d.vos.TextureVo;
public class Burbuja {
private float velocidad_x=.000f;
private float velocidad_y=0.02f;
private float velocidad_z=0.04f;
private float posicion_x=0.45f;
```

```
private float posicion_y=-0.25f;
private float posicion_z=0.3f;
private float variacion_x=0.45f;
private float variacion_y=-0.25f;
private float variacion_z=0.6f;
private float limite_x=4.01f;
private float limite_y=4.01f;
private float limite_z=4.8f;
private Color4 color=null;
private Sphere esfera =null;
long _index;
private float contador=0;
public Burbuja(long index)
{
this._index=index;
}
public void make(Scene scene )
{
if(esfera!=null)
{
esfera.clear();
}
esfera=null;
TextureVo textura=null;
esfera = new Sphere(1.5f, 20,20);
esfera.scale().x = esfera.scale().y = esfera.scale().z = .1f;
esfera.position().setAll(posicion_x, posicion_y, posicion_z);
Bitmap                              b                              =
Utils.makeBitmapFromResourceId(R.drawable.tscuraki);
Shared.textureManager().addTextureId(b, "burbuja0" + _index,
false);
b.recycle();
textura = new TextureVo("burbuja0" + _index);
esfera.textures().addReplace(textura);
esfera.colorMaterialEnabled(false);
esfera.vertexColorsEnabled(false);
esfera.lightingEnabled();
```

```
scene.addChild(esfera);

//Log.v(Min3d.TAG, "ReCrea Burbuja=" + _index);

}

public int mover()
{
int moviendo=1;
posicion_x=variacion_x + ( contador * velocidad_x);
posicion_y=variacion_y + ( contador * velocidad_y);
posicion_z=variacion_z - ( contador * velocidad_z);
esfera.position().setAll(posicion_x, posicion_y, posicion_z);
if((posicion_x>limite_x)||(posicion_y>limite_y)||(posicion_z *
-1 > limite_z ))
{
esfera.clear();

moviendo=0;
}
//Log.v(Min3d.TAG, "Mueve Burbuja=" + _index + " posiciónx="+
posicion_x + " posicióny=" + posicion_y + " posiciónz=" +
posicion_z);
esfera.rotation().x=contador * -1.3f ;
esfera.rotation().y=contador * -1.3f;
esfera.rotation().z=contador * -1.3f;
esfera.scale().x = esfera.scale().y = esfera.scale().z = .1f +
(contador * 0.001f);
contador++;
return moviendo;

}

public float getLimite_x() {
return limite_x;
}
public void setLimite_x(float limite_x) {
this.limite_x = limite_x;
}
```

```
public float getLimite_y() {
return limite_y;
}
public void setLimite_y(float limite_y) {
this.limite_y = limite_y;
}
public float getLimite_z() {
return limite_z;
}
public void setLimite_z(float limite_z) {
this.limite_z = limite_z;
}
public Color4 getColor() {
return color;
}
public void setColor(Color4 color) {
this.color = color;
}
public float getVelocidad_x() {
return velocidad_x;
}
public void setVelocidad_x(float velocidad_x) {
this.velocidad_x = velocidad_x;
}
public float getVelocidad_y() {
return velocidad_y;
}
public void setVelocidad_y(float velocidad_y) {
this.velocidad_y = velocidad_y;
}
public float getVelocidad_z() {
return velocidad_z;
}
public void setVelocidad_z(float velocidad_z) {
this.velocidad_z = velocidad_z;
}
public float getPosicion_x() {
```

```
return posicion_x;
}
public void setPosicion_x(float posicion_x) {
this.posicion_x = posicion_x;
}
public float getPosicion_y() {
return posicion_y;
}
public void setPosicion_y(float posicion_y) {
this.posicion_y = posicion_y;
}
public float getPosicion_z() {
return posicion_z;
}
public void setPosicion_z(float posicion_z) {
this.posicion_z = posicion_z;
}
public float getVariacion_x() {
return variacion_x;
}
public void setVariacion_x(float variacion_x) {
this.variacion_x = variacion_x;
}
public float getVariacion_y() {
return variacion_y;
}
public void setVariacion_y(float variacion_y) {
this.variacion_y = variacion_y;
}
public float getVariacion_z() {
return variacion_z;
}
public void setVariacion_z(float variacion_z) {
this.variacion_z = variacion_z;
}
public Sphere getEsfera() {
return esfera;
```

```
}
public void setEsfera(Sphere esfera) {
this.esfera = esfera;
}
public long getIndex() {
return _index;
}
public void setIndex(long _index) {
this._index = _index;
}

}
```

Finalmente controlamos la distancia a la bombilla gracias al sensor GPS , activándolo si es necesario.

Podemos decidir activar la bombilla a cierta distancia , útil para luces de garages , por ejemplo que se encienda cuando falta 2 kilómetros para llegar con el coche.

En nuestro caso tenemos la esfera que nos indica la distancia , modificamos la textura para que nos muestre FAR si está lejos y NEAR si está cerca.

```
private void loadJipiEs()
{
// Acquire a reference to the system Location Manager
this.locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);

if
(!this.locationManager.isProviderEnabled(LocationManager.GPS_P
ROVIDER))
{
createGpsDisabledAlert();
}
else
{
```

```
// List all providers:
List providers = this.locationManager.getAllProviders();

Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_FINE);

this.bestProvider                                    =
this.locationManager.getBestProvider(criteria, false);

Location                  mylocation                 =
this.locationManager.getLastKnownLocation(this.bestProvider);

if(mylocation!=null)
{
this.getLocation(mylocation);
}
else
{
//this.afegir("Posición inicial vacía.");
}
}

}
private void getLocation(Location location)
{
if(location!=null)
{

boolean hasAltitude=false;
boolean hasAccuracy=false;
boolean hasBearing=false;
lat=location.getLatitude();
lon=location.getLongitude();
alt=location.getAltitude();

hasAltitude=location.hasAltitude();
hasAccuracy=location.hasAccuracy();
hasBearing=location.hasBearing();
```

```
distance=location.distanceTo(coords);
if(distance<5000) { distanceState="near"; } else { distanceState="far"; } } else { distanceState="Unknown"; } //Toast.makeText(getApplicationContext(), "Distance is : " + distance, Toast.LENGTH_LONG).show(); /*runOnUiThread(new Runnable() { public void run() { Toast.makeText(getApplicationContext(), "Distance is : " + distance, Toast.LENGTH_LONG).show(); missatger.setText("Distance is : " + distance); } });*/ escribir("Distance is : " + distance); } private void createGpsDisabledAlert(){ AlertDialog.Builder builder = new AlertDialog.Builder(this); builder.setMessage("Your GPS is disabled! Would you like to enable it?") .setCancelable(false) .setPositiveButton("Enable GPS", new DialogInterface.OnClickListener(){ public void onClick(DialogInterface dialog, int id){ showGpsOptions(); } }); builder.setNegativeButton("Do nothing", new DialogInterface.OnClickListener(){ public void onClick(DialogInterface dialog, int id){ dialog.cancel(); } }); AlertDialog alert = builder.create(); alert.show(); } private void showGpsOptions(){ Intent gpsOptionsIntent = new Intent(
android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS);
startActivity(gpsOptionsIntent); } }
```

En nuestro caso la luz la hemos activado a través de un botón externo a la escena , esta señal modifica la textura de la esfera que nos indica el estado de la bombilla , según sea la respuesta del servidor , ON , OFF o sin conexión , si la bombila está ON la esfera que emula la bombilla encendida , que está en la lámpara , se hace visible rodeando una esfera más pequeña que hace de núcleo de la bombilla.