http://libgdx.badlogicgames.com/

Programación de xogos 2D/3D
Framework LIBGDX

PARTE 2D

Lugar: CE.FO.RE. FERROL
Docente: ANGEL DANIEL FERNÁNDEZ GONZÁLEZ

DO 2 Ó 6 DE SETEMBRO DO 2013

LIBGDX:

É un framework para o desenrolo de xogos coa linguaxe Java tendo coma principal vantaxe que é multiplataforma. Para nós iso significa que podemos desenrolar o xogo independentemente da plataforma onde se vaia a executar.

Así grazas a este framework podemos desenrolar xogos para:

- Móbiles con Android (versións a partires da 1.5).
- Windows.
- IOS.
- Linux.

Máis información sobre as vantaxes en:
http://libgdx.badlogicgames.com/features.html

Polo tanto imos poder desenvolver un xogo e probalo no PC sen necesidade de utilizar un dispositivo móbil (unicamente que o xogo se basee nalgún hardware específico deste tipo de dispositivos coma poda ser o acelerómetro)

LibGDX está composto por unha serie de compoñentes:

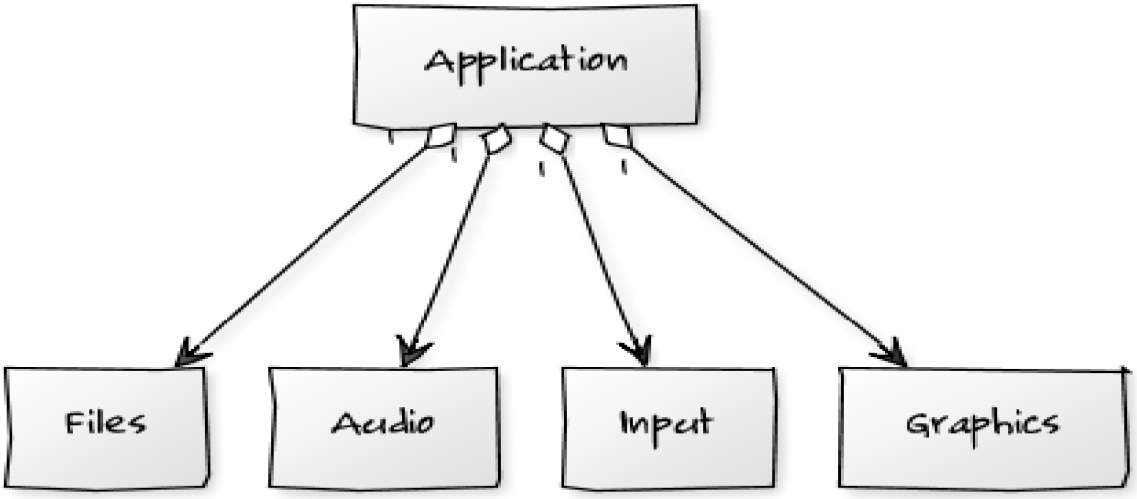
- Marco de **Aplicación**, que manexará o bucle principal y ademais estará encargado do ciclo de vida, é dicir, os eventos de creación, destrución, pausa e resume.

- Un compoñente de **Gráficos** que permitiranos xestionar a representación de imaxes e obxectos gráficos na pantalla.

- Un compoñente de **Audio**, que facilitaranos o acceso ós sons e música da aplicación.

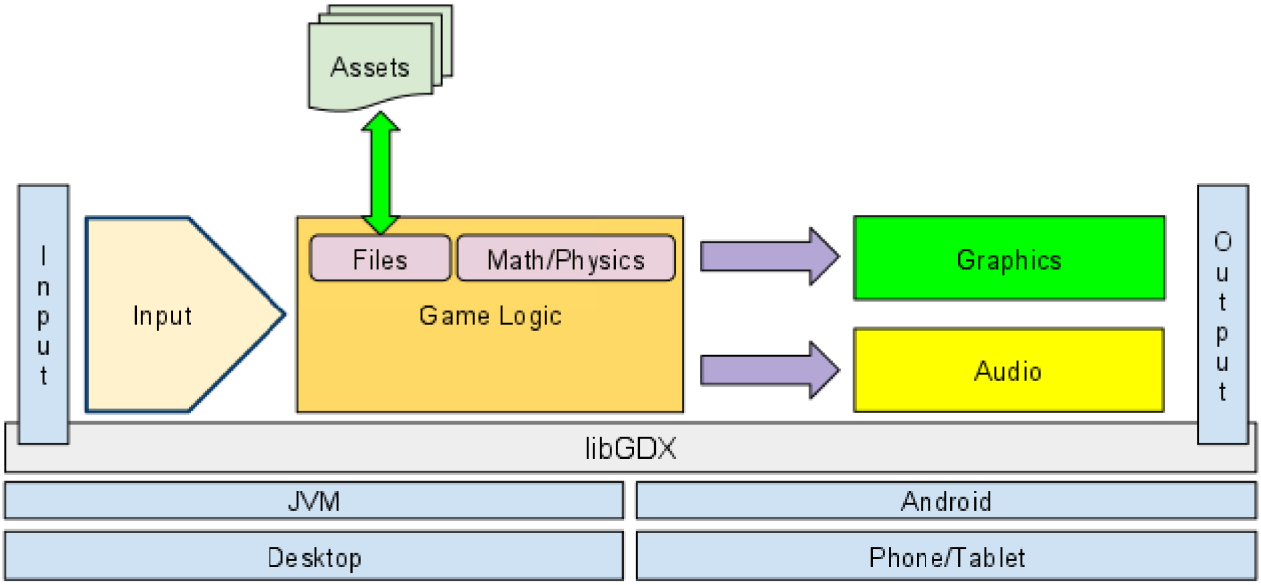
- Un compoñente de **Entrada e Saída** para ler e escribir os diferentes arquivos de datos como por exemplo, arquivos de configuración, sons, música, texturas,...

- Un compoñente de **Entrada** que xestionará a entrada a través do teclado, pantalla táctil ou acelerómetro.



Tamén temos módulos específicos para a xestión de cálculos matemáticos, coma Math e tamén módulos para a xestión de colisións entre os elementos gráficos dos xogos.

Outro gráfico que amosa de diferente forma a estrutura de módulos do framework LIBGDX:



Nota: Assets é o cartafol onde imos gardar todos os recursos (gráficos – modelos 3d– sons – música,...) do noso xogo. Nas aplicacións Android normalmente se gardaría no cartafol /res. Podemos ver neste enlace a diferenza entre os dous cartafois:
http://stackoverflow.com/questions/5583608/difference-between-res-and-assets-directories

INSTALACIÓN:

O que temos que ter instalado no noso equipo é:

- JRE (Java RunTime Environment) ou JDK (Java Development Kit).
- Android SDK.
- Eclipse + pluging aplicacións Android.
- Framework libgdx

O proceso de instalación será o seguinte:

Instalación do JRE.

O podemos descargar dende:
http://www.oracle.com/technetwork/java/javase/downloads/jre7-downloads-1880261.html

e escoller a versión correcta.

Normalmente se instala no cartafol **C:\Archivos de Programa\Java\JRE7**

Instalación do Android SDK

Podemos seguir este manual: http://developer.android.com/sdk/installing/index.html

O Android SDK vainos permitir desenrolar xogos para as diferentes versións de Android e dispoñer de máquinas virtuais que son emuladoras dun sistema operativo Android para facer probas sobre elas.

O SDK atópase nesta páxina:

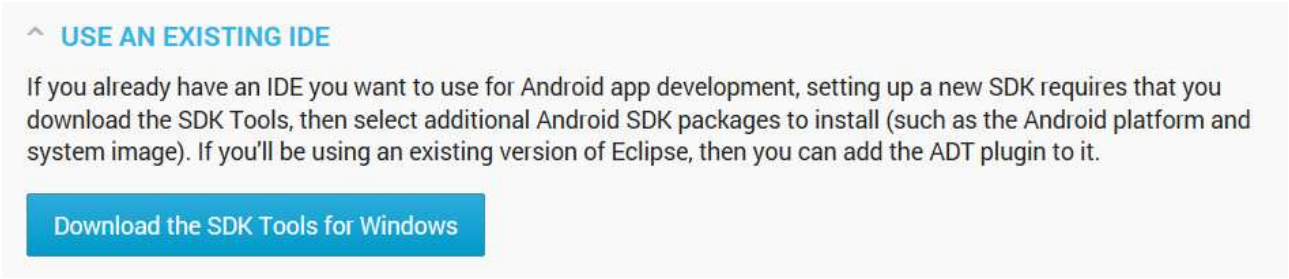
http://developer.android.com/sdk/index.html

Opción a)

Neste punto podemos descargar o ADT Bundle for Windows se queremos descargar nun só paso todo o software necesario para desenrolar aplicacións para móbiles (Eclipse, Android SDK,....).

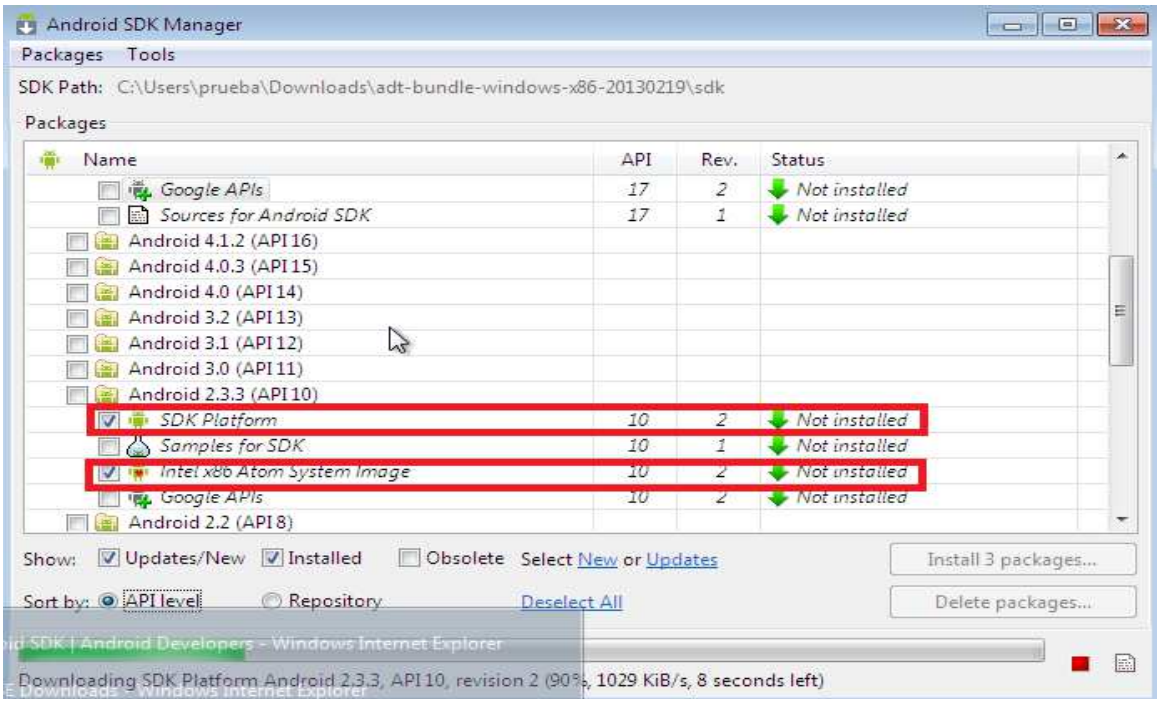
Opción b)

Se xa temos o IDE descargado podemos facelo por pasos e instalar só o SDK, premendo sobre a opción 'USE AN EXISTING IDE'



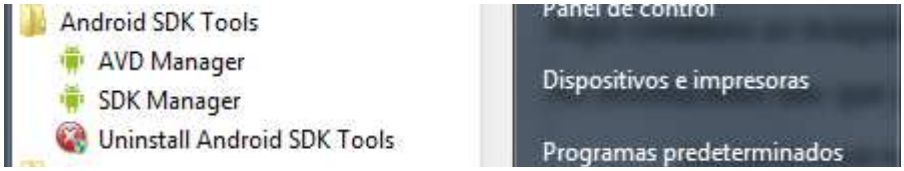
Ó executar o SDK indícanos o que temos instalado e se queremos instalar novas API´s.

Escoller “SDK Plataform-tools” e “SDK Tools” así coma as versións de Android 2.3.3 e a 4.0.3 (escoller SDK Plataform e unha System Image por cada versión) como se amosa a continuación e o Google Usb Driver en Extras :



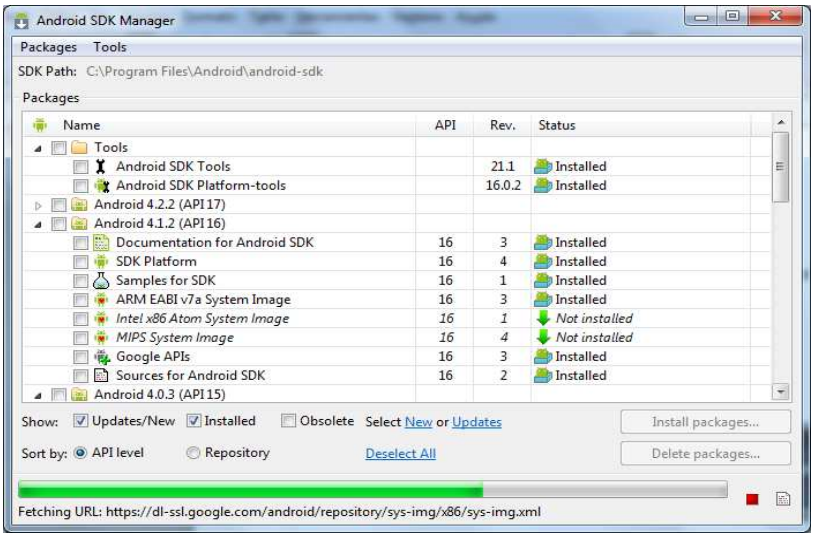
Nota: se instalamos o ADT Bundle For Windows trae descargada a versión de Android 4.2.2 (API 17) e a Android 2.3.3 (API 10).

Unha vez instalado temos acceso a dous programas (se escollemos a opción b, instalación independente):

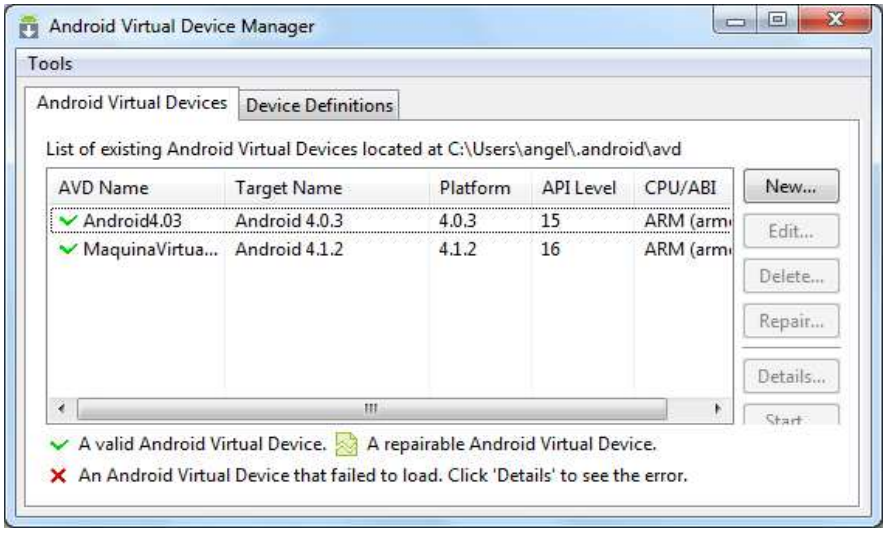


En caso de utilizar a opción a) (bundle) teremos no raíz do arquivo descomprimido o SDK Manager:

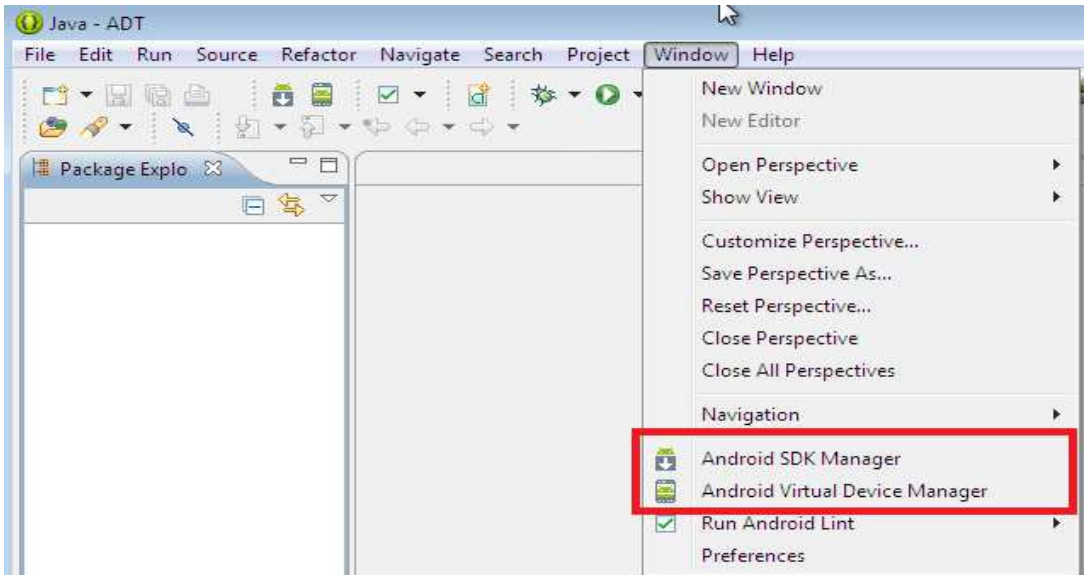
- SDK Manager: volvamos á pantalla da instalación, por se queremos instalar ou desinstalar versións API de Android instaladas.



- AVD Manager: para crear os emuladores de Android.

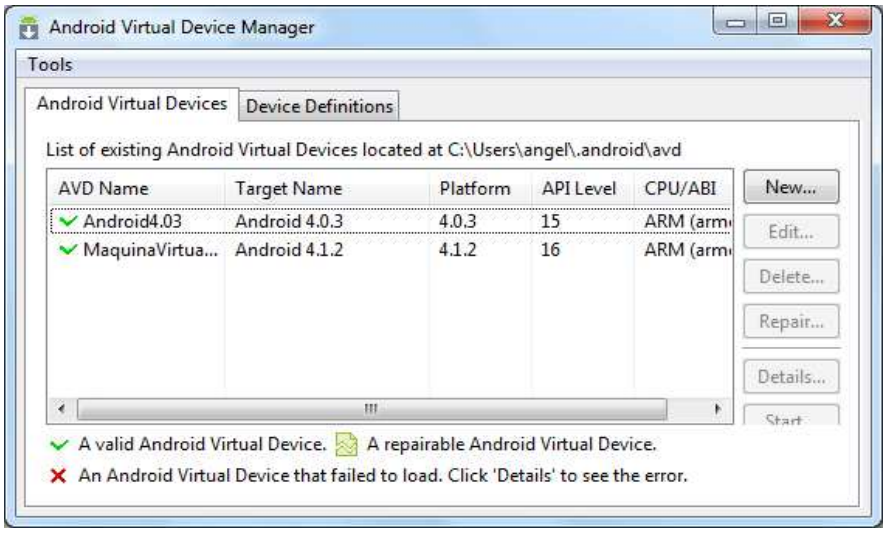


Dende o propio IDE do Eclipse podemos acceder a ambos programas:



Creación de novas máquinas virtuais Android.

Para crear unha nova máquina virtual debemos de premer o botón de New de AVD Manager:



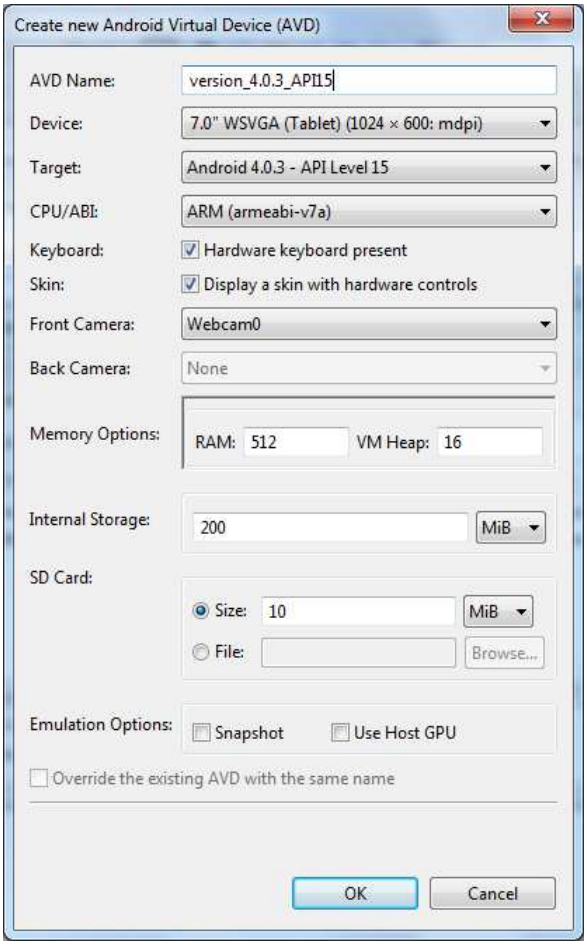
e escoller a versión de Android que imos a executar (as versións dispoñibles serán as instaladas no paso anterior).

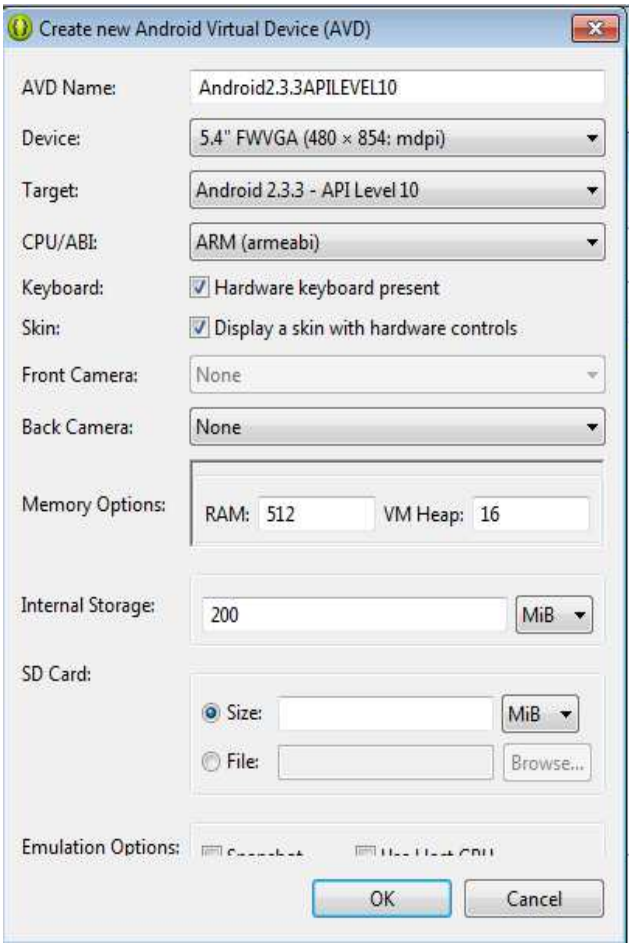
Podemos emular diferente hardware presente nun dispositivo Android, como a cámara web, a memoria, unha tarxeta SD Card externa, o tamaño do dispositivo,....

O lóxico será escoller unha versión de Android cun Hardware (sobre todo a resolución da pantalla => tamaño) que teñan unha maioría de dispositivos de tal forma que así nos aseguramos que a nosa aplicación vai a executarse sen problemas neses dispositivos.

Se estamos a usar o ADT Bundle For Windows do paso anterior teremos a posibilidade de crear dispositivos virtuais có S.O. Android 2.3.3 ou 4.2.2.

Crearemos polo tanto un dispositivo para cada versión de Android.





Podemos ver os cadros seguintes para facernos unha idea de como son os dispositivos que utilizan Android na actualidade.

As resolucións nas que podemos traballar están no seguinte cadro:

	Low density (120), *ldpi*	Medium density (160), *mdpi*	High density (240), *hdpi*	Extra high density (320), *xhdpi*
Small screen	**QVGA (240x320)**		480x640	
Normal screen	**WQVGA400 (240x400)** **WQVGA432 (240x432)**	**HVGA (320x480)**	**WVGA800 (480x800)** **WVGA854 (480x854)** 600x1024	640x960
Large screen	**WVGA800** (480x800)** **WVGA854** (480x854)**	**WVGA800* (480x800)** **WVGA854* (480x854)** 600x1024		
Extra Large screen	1024x600	**WXGA (1280x800)**† 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

Información sacada de http://developer.android.com/guide/practices/screens_support.html

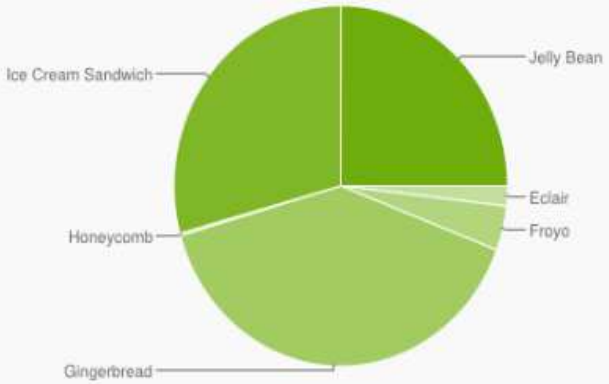
A cota de mercado dos dispositivos Android nas súas respectivas versións do S.O. en Abril do 2013:

Platform Versions

This section provides data about the relative number of devices running a given version of the Android platform.

For information about how to target your application to devices based on platform version, read Supporting Different Platform Versions.

Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.7%
2.2	Froyo	8	4.0%
2.3 - 2.3.2	Gingerbread	9	0.1%
2.3.3 - 2.3.7		10	39.7%
3.2	Honeycomb	13	0.2%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	29.3%
4.1.x	Jelly Bean	16	23.0%
4.2.x		17	2.0%

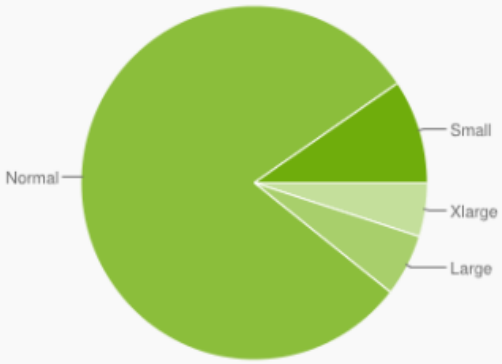


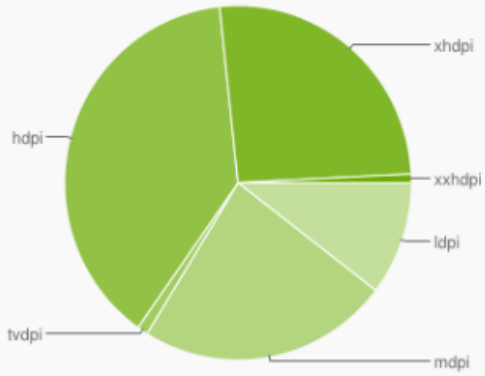
Data collected during a 14-day period ending on April 2, 2013.
Any versions with less than 0.1% distribution are not shown.

Información obtida de: http://developer.android.com/about/dashboards/index.html

Cota de mercado por resolución de pantalla:

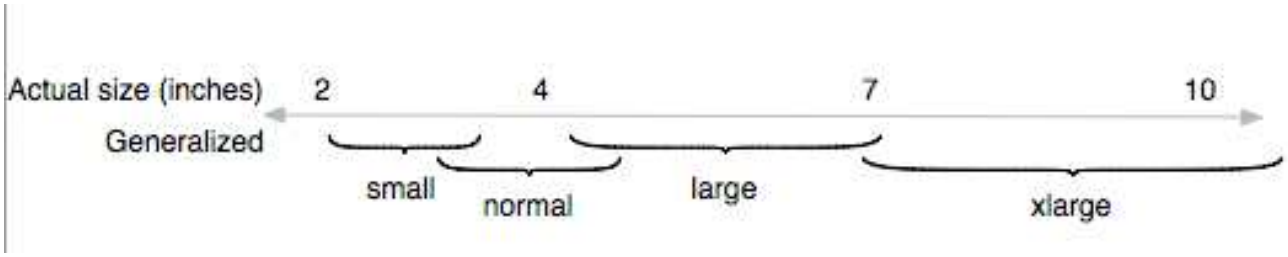
	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	9.5%						9.5%
Normal	0.1%	16.1%		37.9%	25.0%	0.8%	79.9%
Large	0.7%	2.7%	1.0%	0.5%	0.8%		5.7%
Xlarge	0.1%	4.6%		0.1%	0.1%		4.9%
Total	10.4%	23.4%	1.0%	38.5%	25.9%	0.8%	





Data collected during a 14-day period ending on April 2, 2013
Any screen configurations with less than 0.1% distribution are not shown.

Destas gráficas podemos deducir que un 80% dos dispositivos Android teñen un tamaño de pantalla Normal



e un 37,9% dos dispositivos con este tamaño traballan cunha resolución de hdpi (480x800) e un 25% xhdpi (640x960)

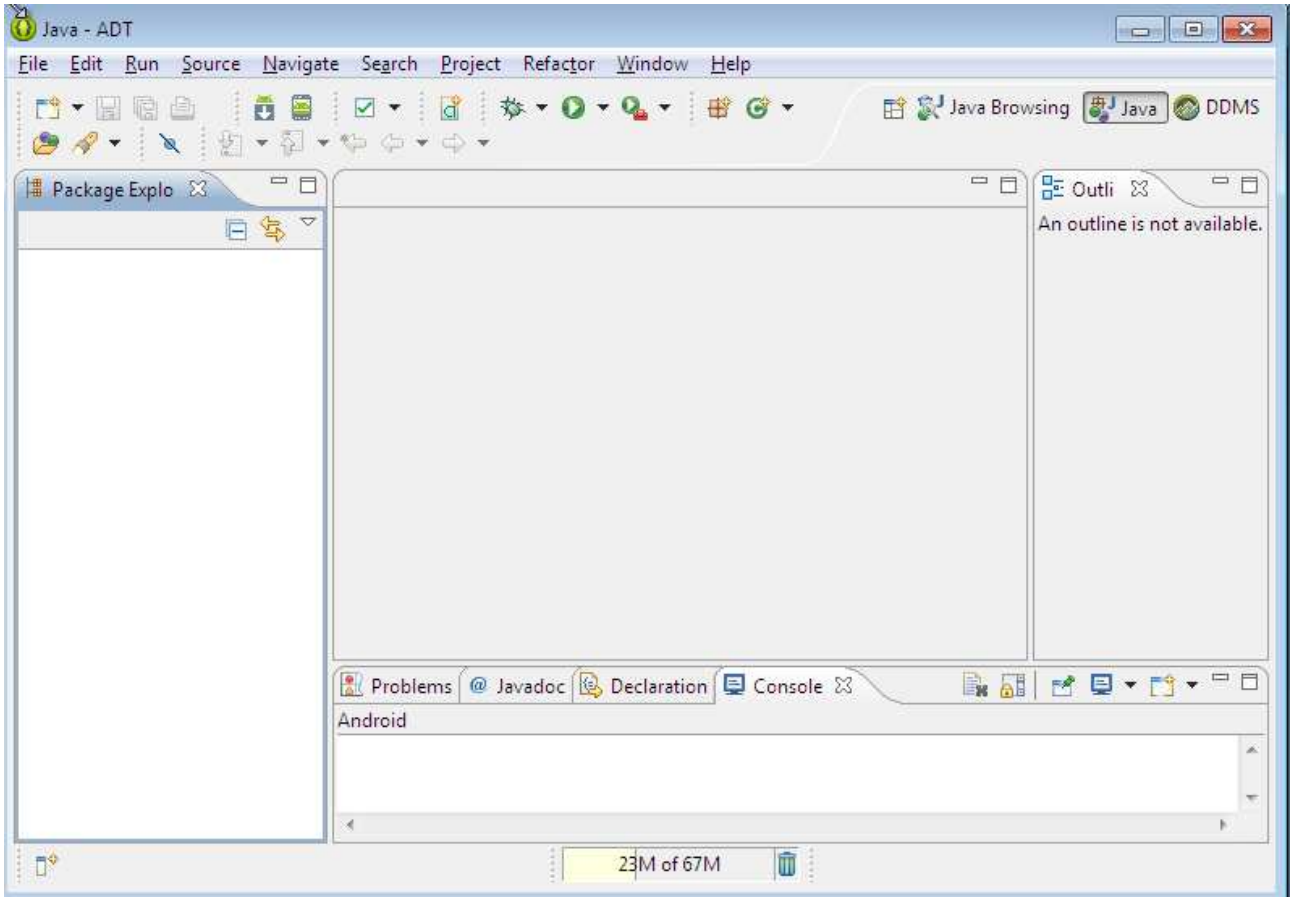
Instalación do IDE ECLIPSE.

Debemos de descargar e executar o entorno de desenrolo ECLIPSE.

Nota: Se usamos o ADT Bundle (\SOFTWARE\adt-bundle-windows-x86-20130219\) xa está configurado para usar o Android SDK e se atopa no cartafol Eclipse (nome eclipse.exe).

Tamén podemos descargar a última versión dende: http://www.eclipse.org/downloads/
e escollemos a opción Eclipse for Mobile Developers.

Unha vez descomprimido podemos executar o programa premendo dúas veces sobre o arquivo ECLIPSE.EXE (se estamos no S.O. Windows).

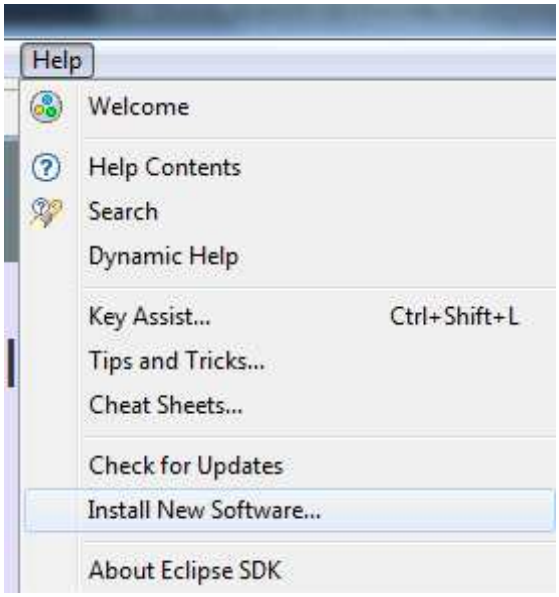


Instalar o plugin para ECLIPSE:
http://developer.android.com/sdk/installing/installing-adt.html

Este plugin o imos necesitar para desenrolar aplicacións Android dentro do IDE ECLIPSE.

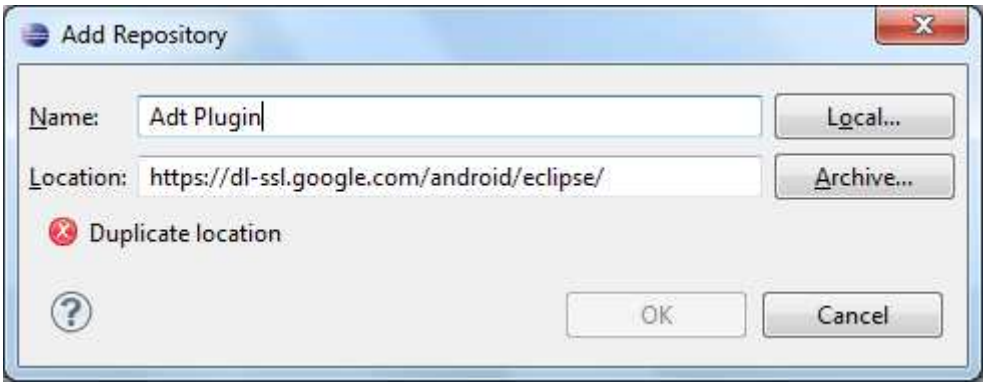
Nota: Se usamos o ADT Bundle (\SOFTWARE\ADT Bundle\) xa está instalado.

Para instalalo só temos que arrancar o eclipse e premer sobre o menú Help da parte superior:



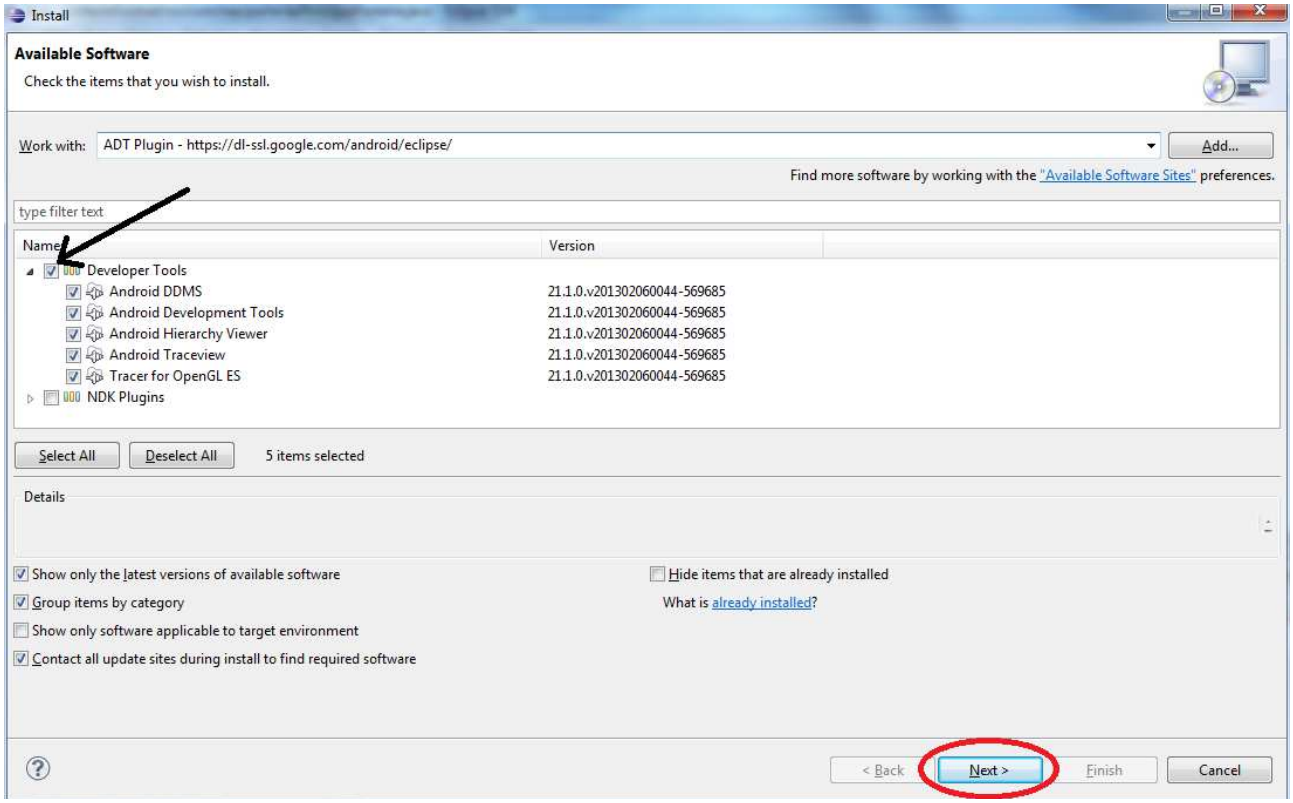
escollendo a opción 'Install New Software'.

Debemos premer o botón Add:

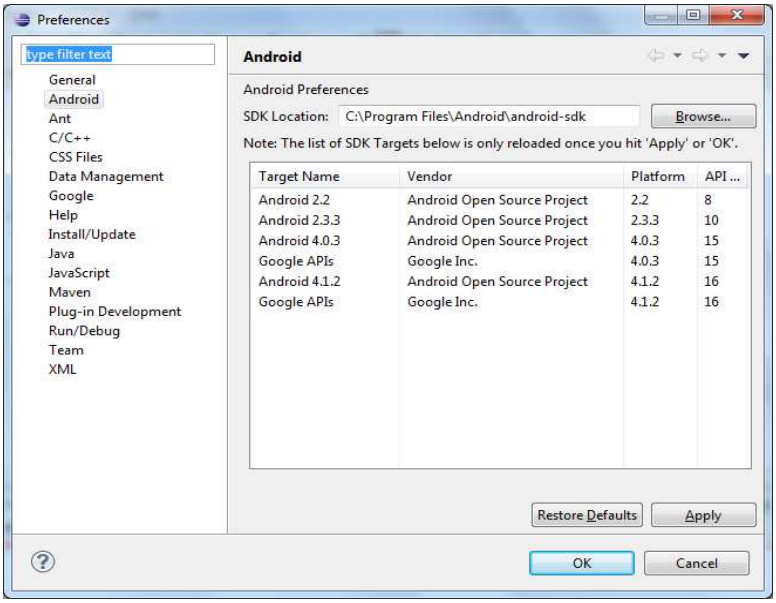


e premer o botón de OK.

Seleccionamos todos os paquetes dentro de Developers Tools e prememos o botón de Next ata o final.



Unha vez instalado, podemos ir o menú Window => Preferencias do Eclipse e seleccionar o cartafol onde se instalou o SDK do Android.



Se todo está correctamente instalado debería aparecer na parte superior do IDE Eclipse as iconas necesarias para executar proxectos Android:

Podemos consultar combinacións de teclas en Eclipse que nos pode ser útiles:

http://www.shortcutworld.com/en/win/Eclipse.html

Instalando o noso primeiro xogo usando libgdx.

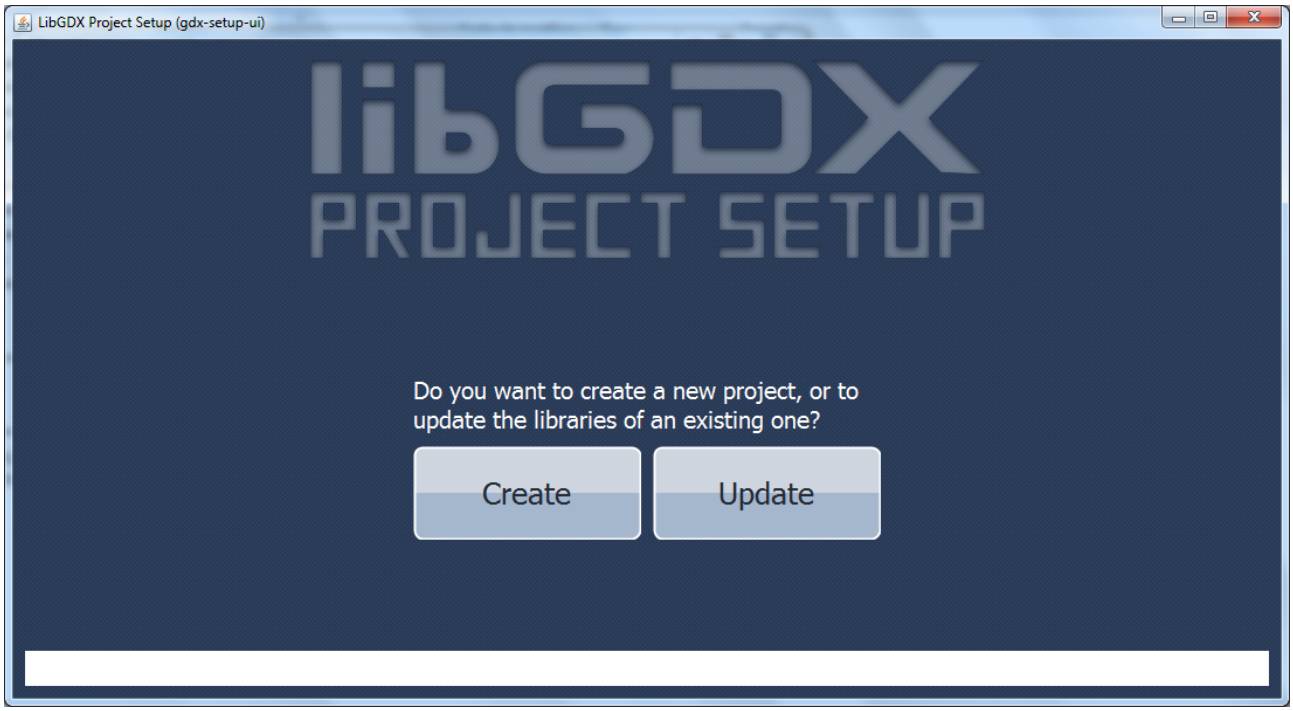
O primeiro será crear os proxectos necesarios para desenrolar os xogos.
Se o facemos 'manualmente' teríamos que crear un proxecto Android (versión Android do xogo), un proxecto Java cunha clase que implemente o método main (sería a versión desktop) e outro proxecto Java onde irían todas as clases necesarias para desenrolar o noso xogo.

Os dous proxectos creados inicialmente (móbil e desktop) chamarían ás clases do noso xogo (terceiro proxecto) utilizando as librerías do framework libdx.

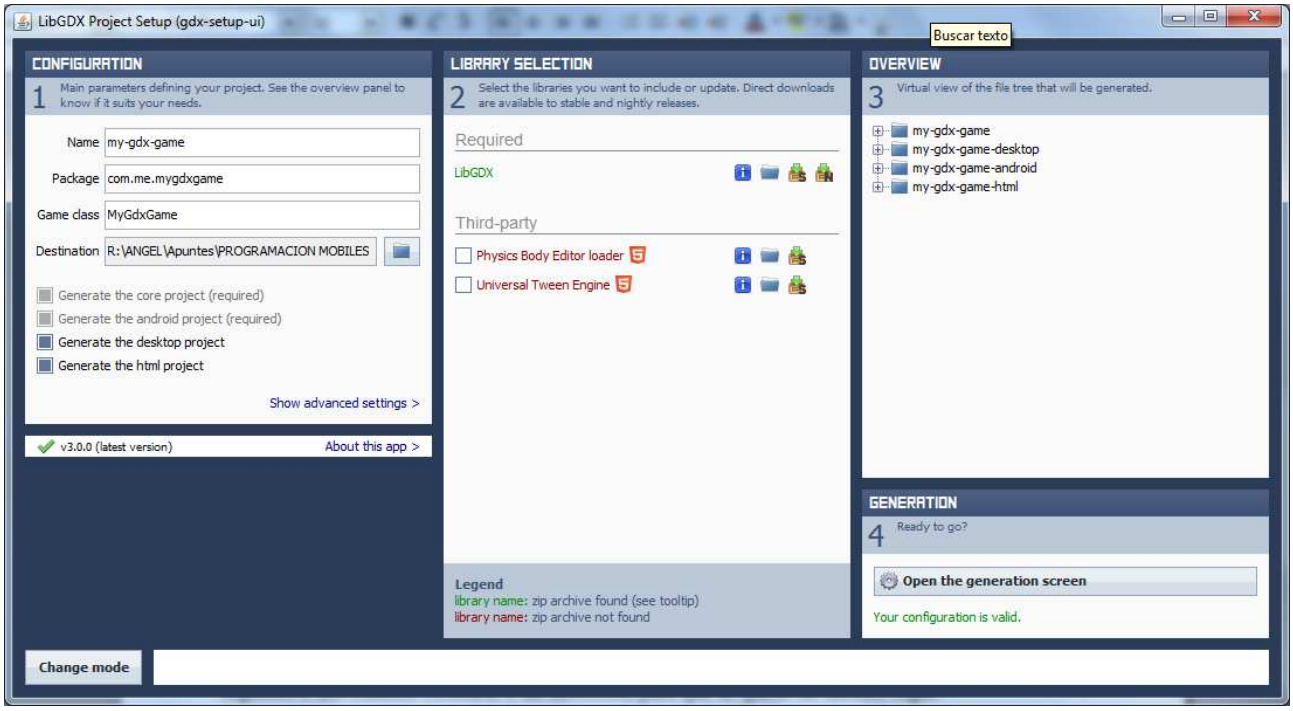
Este proceso (aínda que só se fai unha vez por cada xogo) e longo e tedioso, pero podémolo aforrar facendo uso dunha ferramenta desenrolada en Java que nos xerará a estrutura necesaria para desenvolver un xogo para diferentes plataformas.

Podemos descargar dita aplicación dende http://code.google.com/p/libgdx/wiki/ProjectSetupNew ou executala dende o DVD \SOFTWARE\Aplicación para crear proxectos libgdx\ executando o arquivo 'lanzar.bat' que está en dito cartafol.

O facelo aparecerá a seguinte pantalla:



Se queremos actualizar un proxecto xa creado actualizando as librerías do framework libgdx debemos premer o botón de Update. No noso caso imos crear un xogo novo e polo tanto debemos de premer o botón de Create



O primeiro que temos que facer é descargar as librerías que conforman o noso framework. Poderíamos descargalas dende aquí:

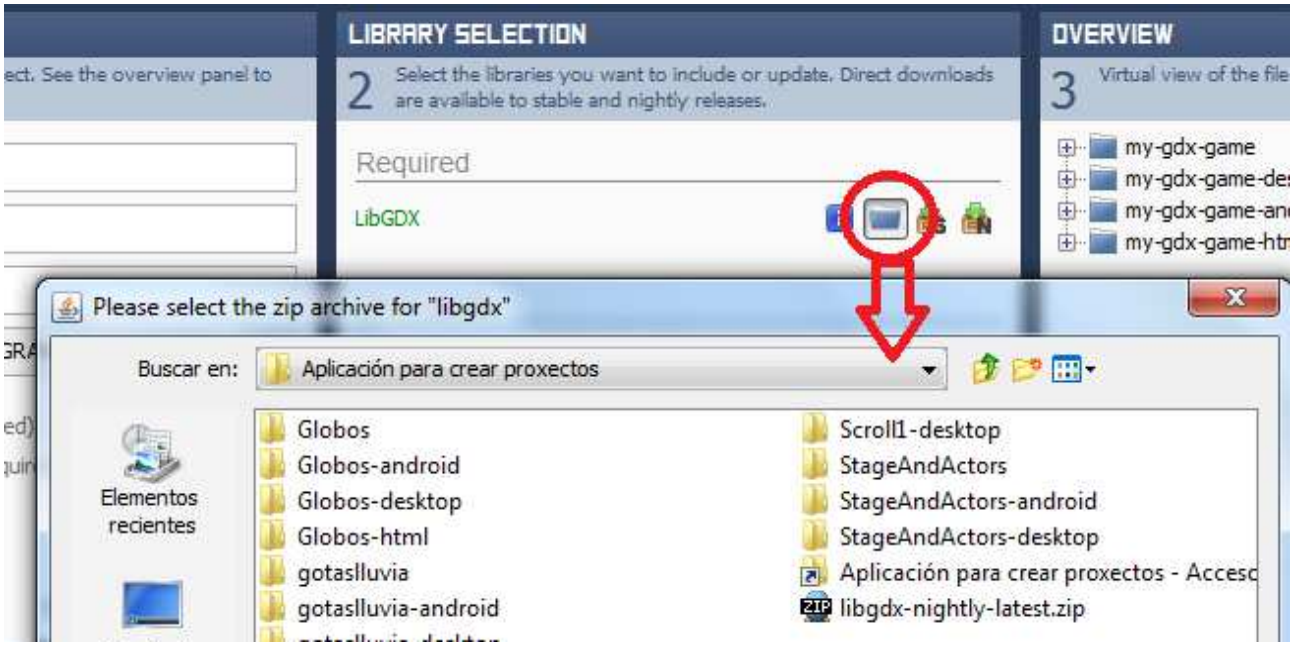
http://libgdx.badlogicgames.com/download.html

pero xa a propia ferramenta déixanos descargar a última versión:

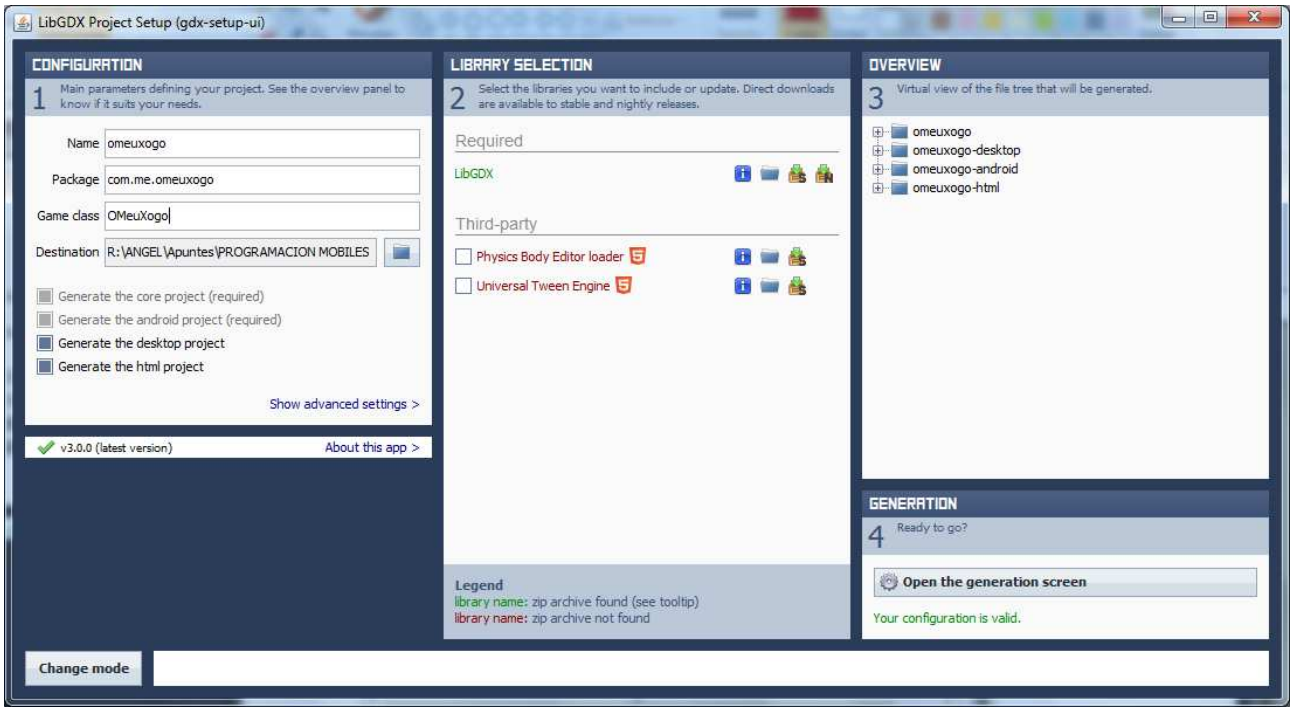
Debemos descargar a opción 'nightlies'. Dita versión ven ser como a 'stable' pero con bug´s e erros corrixidos.

Se xa a tivéramos descargada podemos indicarlle a aplicación onde se atopa premendo sobre o

cartafol e seleccionando o arquivo zip:

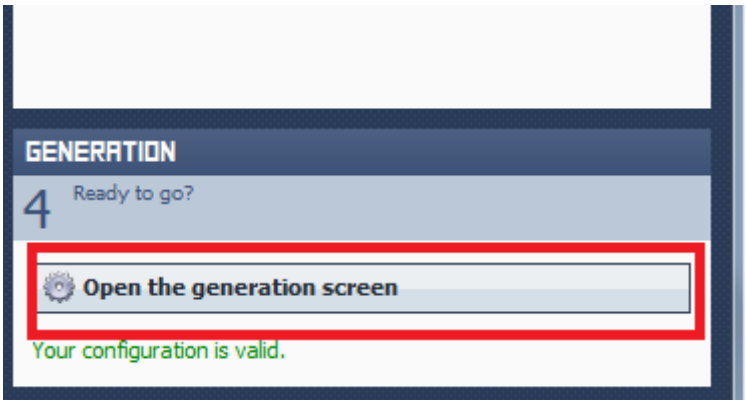


Unha vez escollido a versión do framework a utilizar podemos poñer o nome os nosos proxectos así como o nome da clase principal do xogo (pódese cambiar despois, xa o veremos).

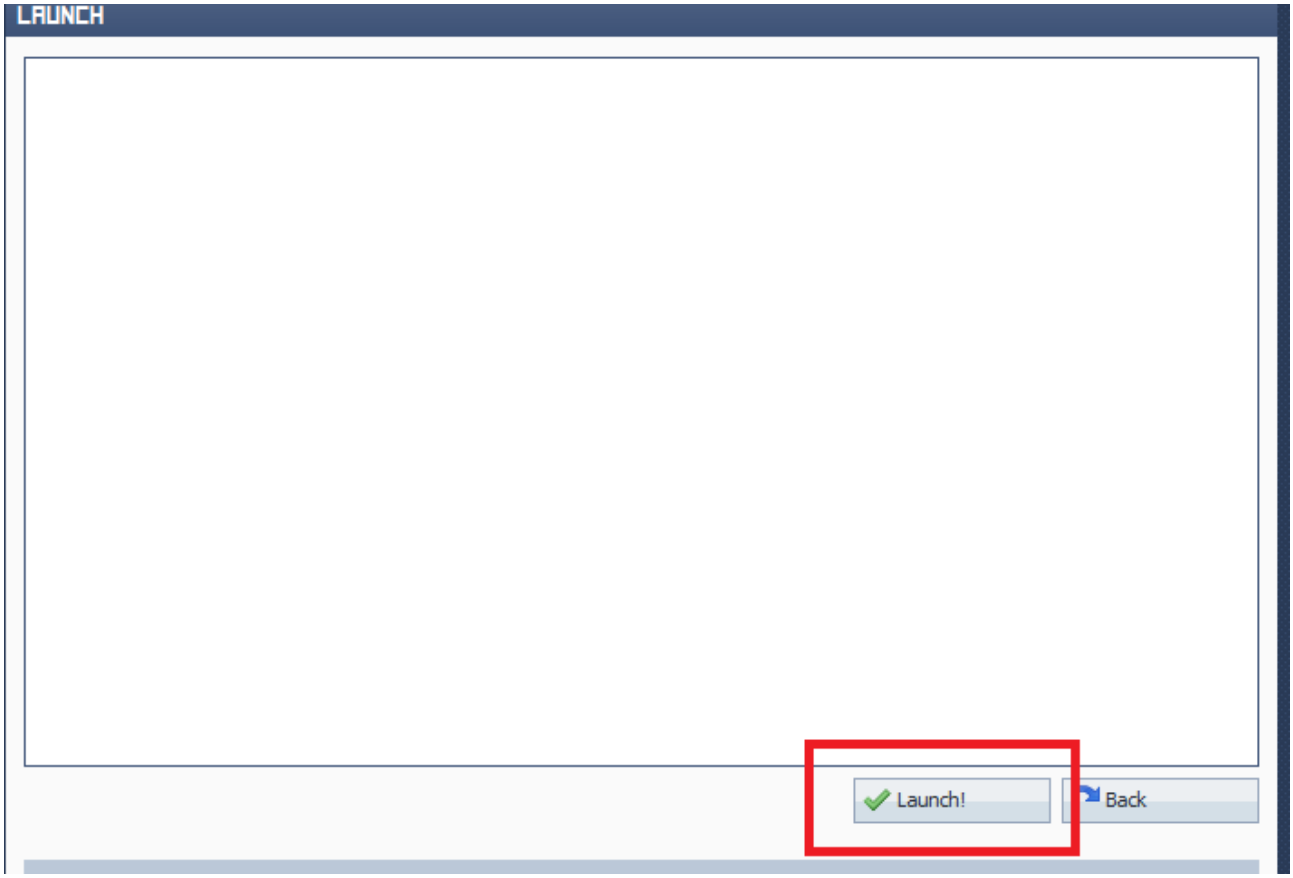


Nota: O nome da clase (Game Class) debe comezar con maiúscula.

Como vemos na parte dereita se amosa os nomes do proxectos que van xerarse.
Unha vez feito debemos premer o botón 'Open the generation screen'.



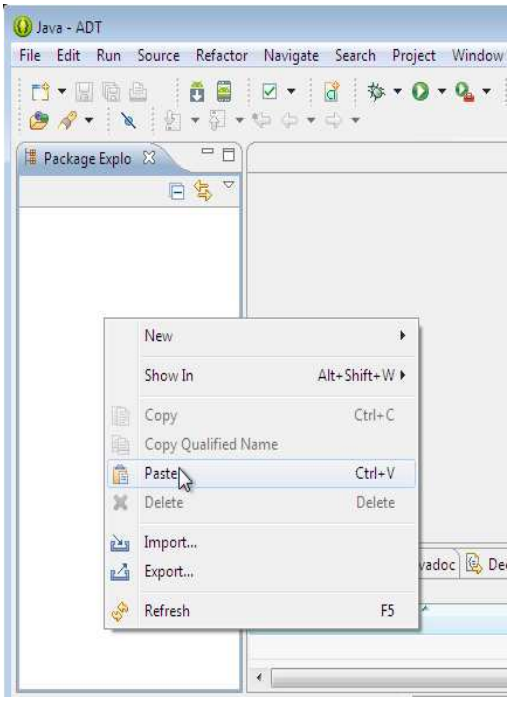
E premer o botón de 'launch'.



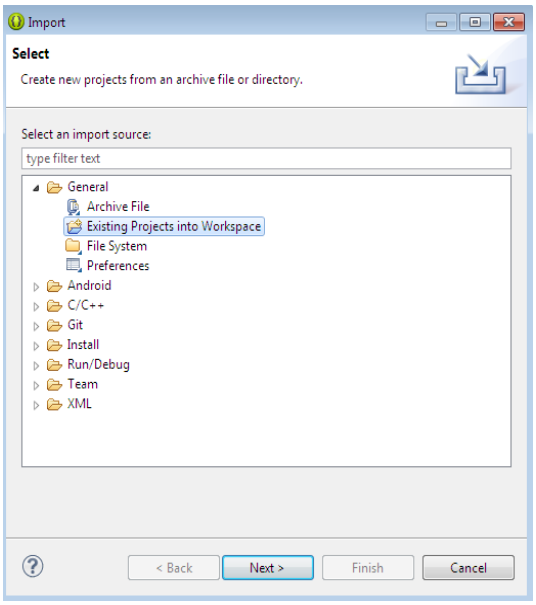
Xerándose desta forma a estrutura de proxectos que imos necesitar para desenvolver os xogos.

Agora só queda importar ditos proxectos o entorno de desenrolo Eclipse.

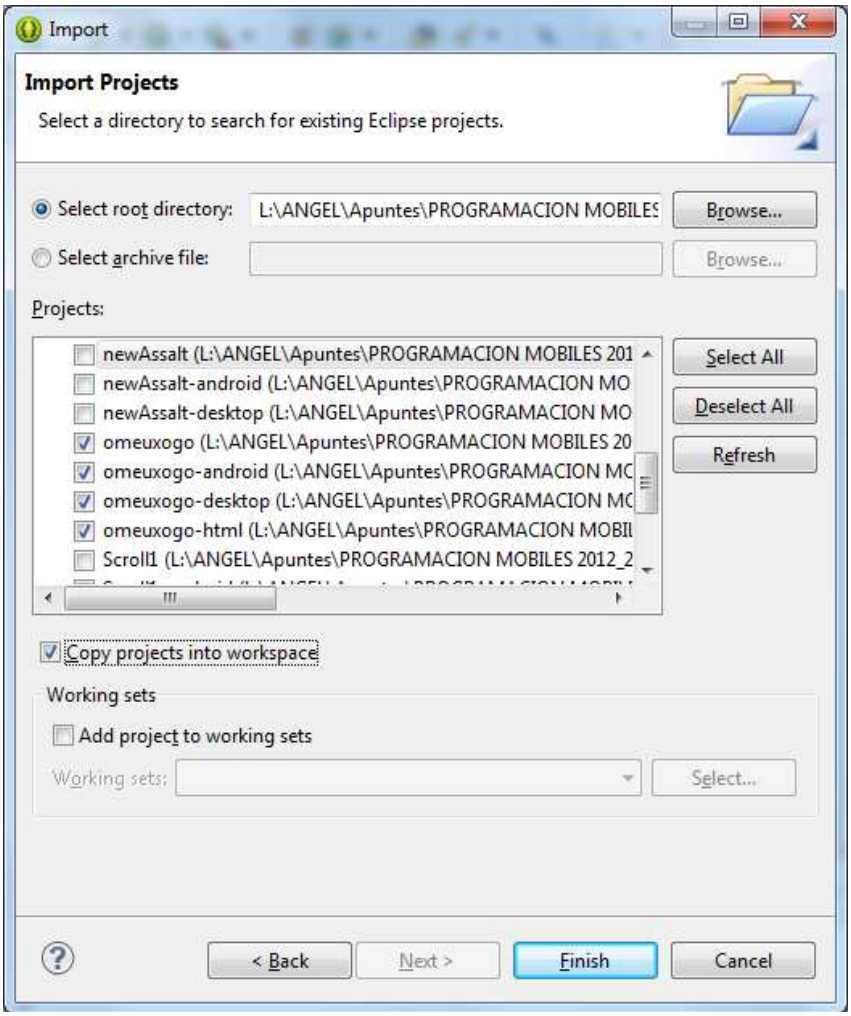
O podemos facer premendo o botón dereito do rato sobre o Package Explorer e facendo Import:



Escollemos a opción General => Existing Proyects Into Workspace:



Prememos o botón de Browse e nos posicionamos no cartafol onde se atopan os proxectos xerados pola ferramenta:



Nota: Lembrade marcar o opción 'Copy proyects into workspace' para que faga unha copia do proxecto o espazo de traballo do eclipse (por defecto C:\usuarios\usuarioconectado\workspace, o podemos cambiar indo ó menú File => Switch WorkSpace).

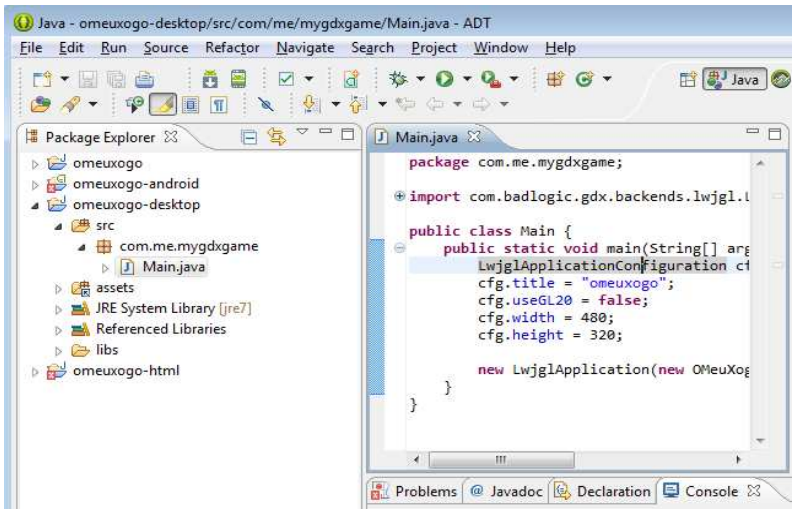
Prememos o botón de Finish.

Posiblemente vexamos un erro no proxecto de Android e o de HTML.
O de Android é debido á versión de Android na que se ten que executar a aplicación, a cal non se atopa instalada na nosa máquina (lembrar que escollimos as versións 2.3.3 e 4.0.3 ou se estamos co adt bundle teremos a 4.2.2 e a 2.3.3).

O erro do HTML é debido a que temos que ter instalado o Google Web Toolkit para o Eclipse se queremos probar a execución do xogo dentro dunha páxina web (non funciona co Internet Explorer).

Como podemos observar temos unha versión Android, outra HTML e outra Desktop (xa explicado anteriormente).

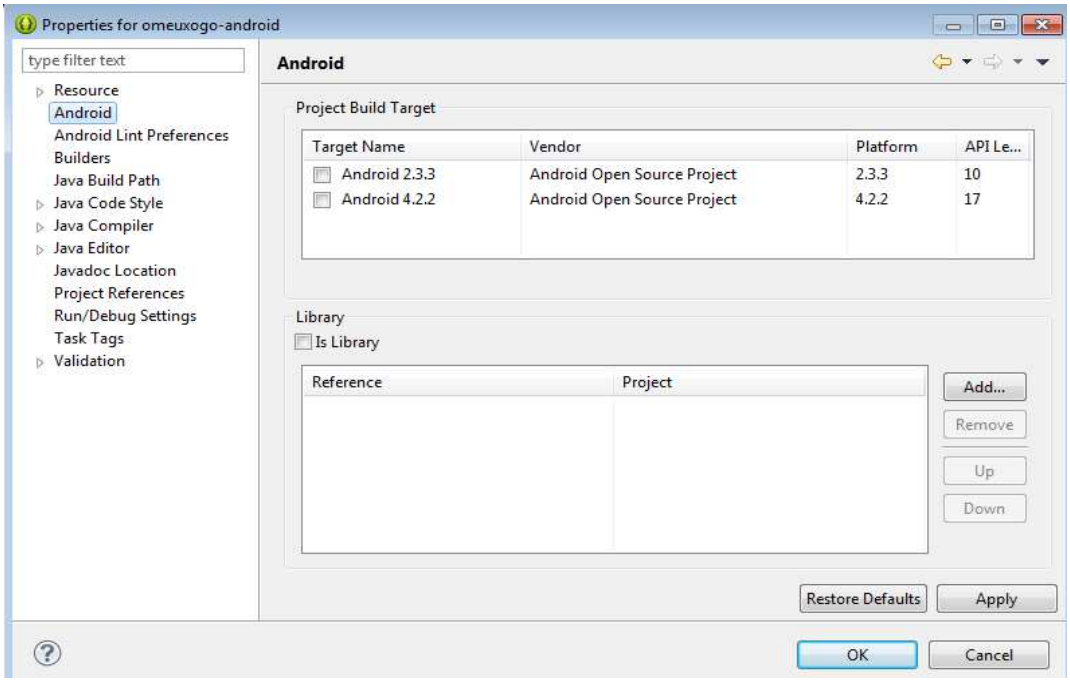
Podemos probar que as librerías están ben instaladas executando a aplicación de mostra.
Para isto imos a versión DESKTOP e nos movemos polo cartafol src e dentro deste imos o paquete com.me.mygdxgame e abrimos a clase Main.java.



Agora debemos executar premendo o botón de Run:

Ó facelo aparecerá unha xanela coma esta:

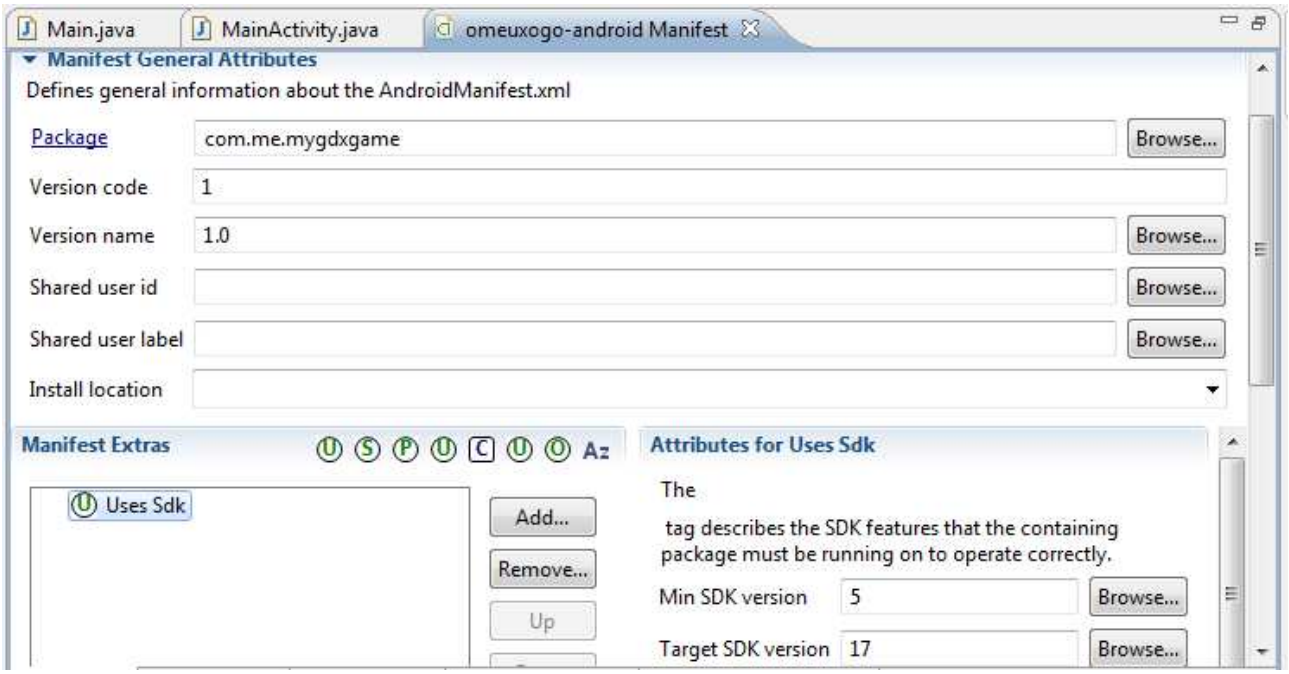
Podemos comprobar que podemos facer o mesmo coa versión Android. Primeiro imos cambiar a versión de Android sobre a que se vai a executar a aplicación de mostra.
Para isto temos que premer o botón dereito sobre o proxecto e escoller a opción de Propiedades e a rama de Android:



Na parte da dereita aparecerán as versións de Android descargadas polo SDK (pasos anteriores). Escollemos a versión 4.2.2 e prememos o botón de Ok. A versión 4.2.2 leva asociada a API Level 17. Imos modificar o arquivo AndroidManifiest.xml que se atopa no raíz do proxecto Android para indicarlle que a mínima versión no que pode correr a nosa aplicación sexa a 2.3.3. (API 10) e informarlle que imos desenrolar unha aplicación para a versión de Android 4.2.2. (API 17).

O arquivo AndroidManifiest.xml úsao Android como arquivo de configuración no que se garda información sobre os permisos que ten que ter a aplicación para poder executarse, o nome da mesma, a icona, actividade principal,....

Prememos polo tanto dúas veces en dito arquivo (AndroidManifiest.xml):



Escollemos a opción Uses Sdk e na parte dereita aparecerán dous campos (Min SDK e Target SDK). O primeiro é o que imos modificar escribindo o número 10 (API 10) en Min SDK e 17 en Target SDK, premendo o botón de Gardar.

NOTA: se tivésemos instaladas as imaxes doutras versións do Android poderíamos poñer como target a API correspondente a dita versión tendo en conta que o target non debería ser inferior a versión Android 3.2 debido a que por defecto o arquivo de Manifiesto incorpora esta liña:

....android:configChanges="keyboard|keyboardHidden|orientation|screenSize">

Esta liña vai indicar unha lista de 'eventos' que de non estar recollidos nesta configuración provocarían a destrución e creación da aplicación nun dispositivo Android. Xa veremos que dende o punto de vista do manexo de gráficos, isto suporía a destrución do contexto de Open Gl, o que se traduce en que teríamos que volver a cargar todos os gráficos que manexemos. Desta forma cando se produce dito evento a aplicación non se destrúe.

Se poñemos un target API menor ó 10 (se pode facer, xa que libgdx funciona a partires da API 6) teríamos que eliminar o screensize que foi incorporado posteriormente.

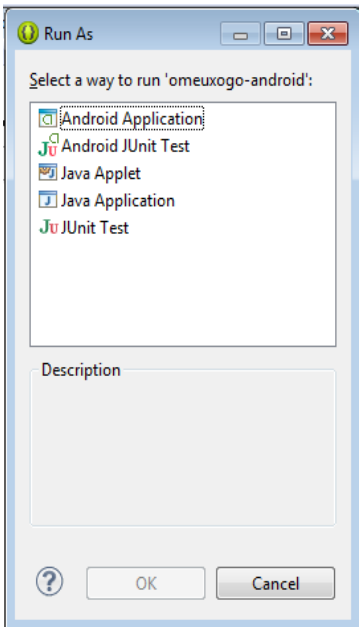
Máis información en:

- http://developer.android.com/guide/topics/manifest/activity-element.html
- http://code.google.com/p/libgdx/wiki/ApplicationConfiguration

Agora xa podemos abrir a clase MainActivity.java e executala.

Ó facelo aparecerá unha xanela na que se nos pedirá que indiquemos que tipo de aplicación imos a

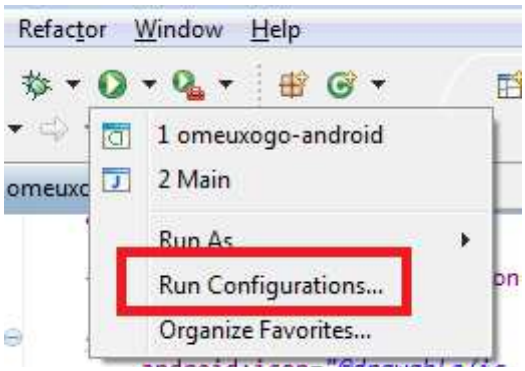
executar. Nos elixiremos 'Android Application':



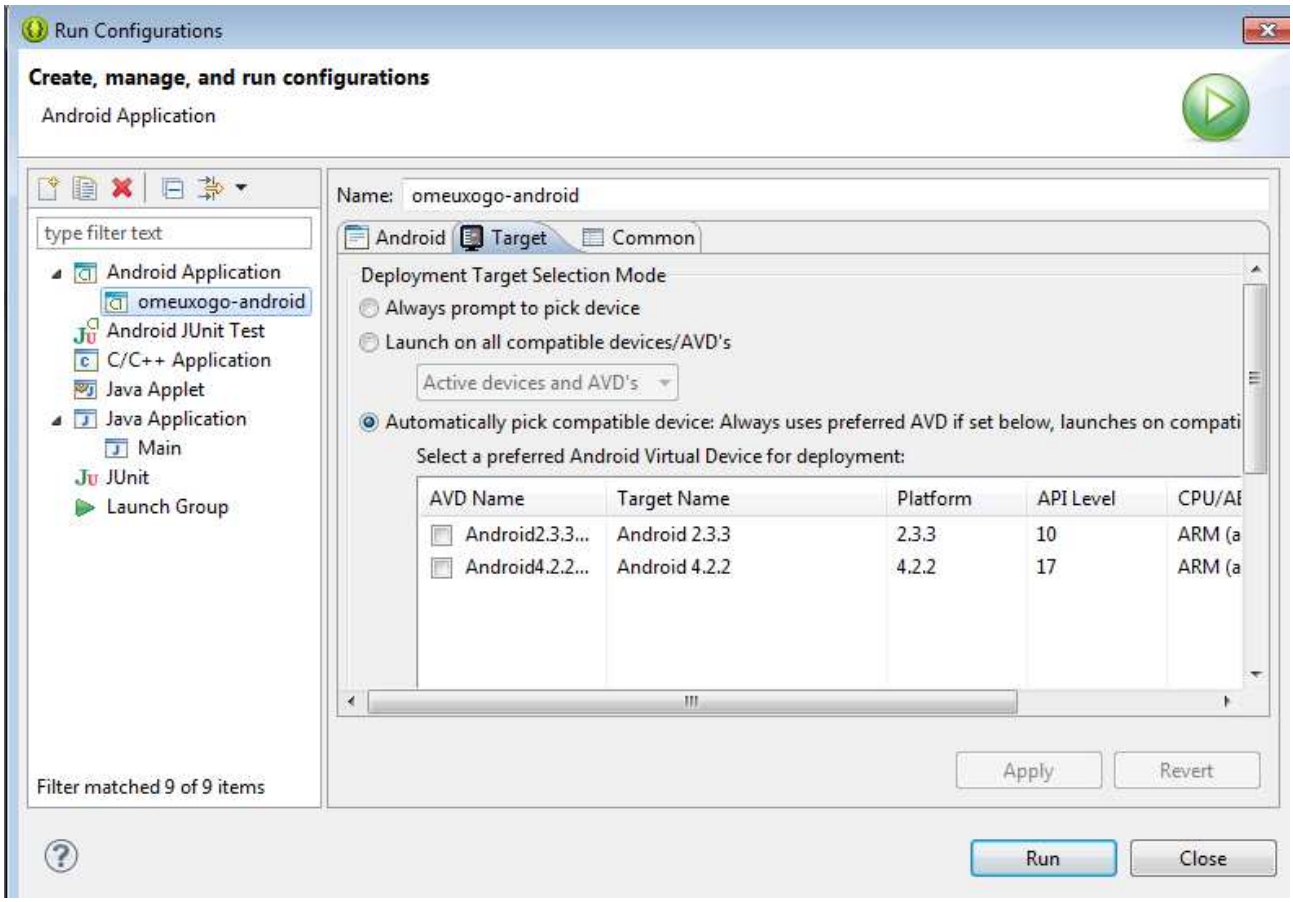
Isto vai levar consigo que se lance unha máquina virtual có S.O. Android indicado nas propiedades do proxecto (no noso caso a versión 4.2.2):

Nota: No emulador de Android podemos rotar a pantalla premendo as teclas Ctrl+F11

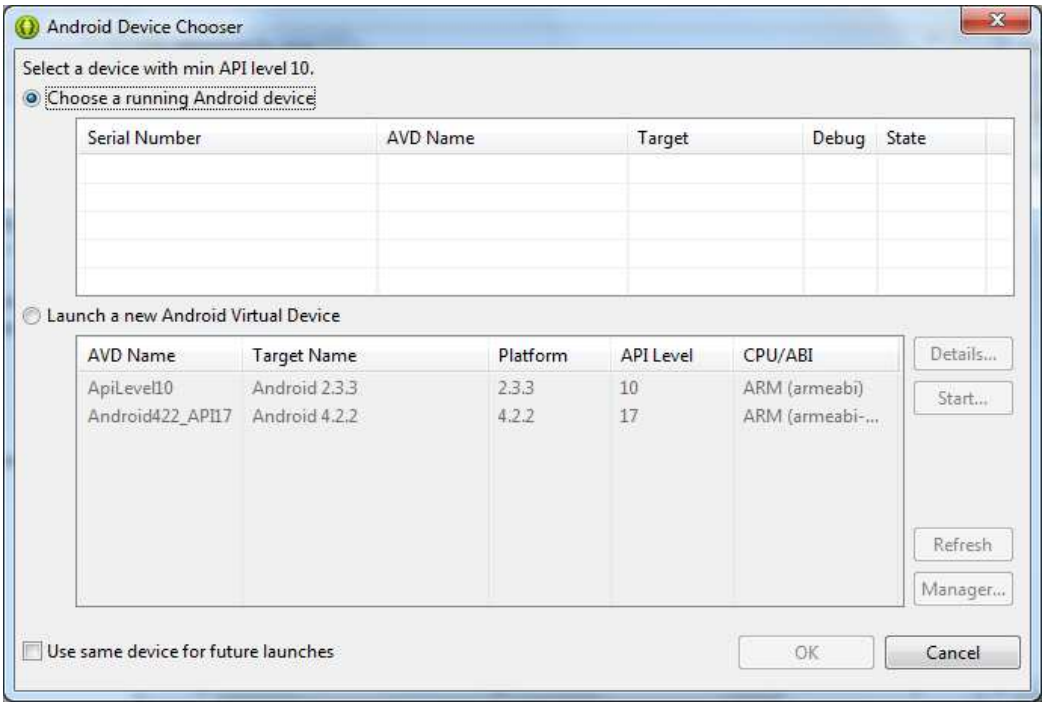
Cando executamos por primeira vez unha aplicación créase unha 'configuración de execución'. Podemos editar dita configuración premendo sobre a frecha que está ó carón da icona de execución e escollendo a opción 'Run Configuration'.



No caso da configuración do proxecto Android podemos indicar na pestana 'Target' que dispositivo virtual terá que lanzarse cando executemos a aplicación Android. Imos modificala para que sempre pregunte, escollendo a opción 'Always prompt to pick device'.



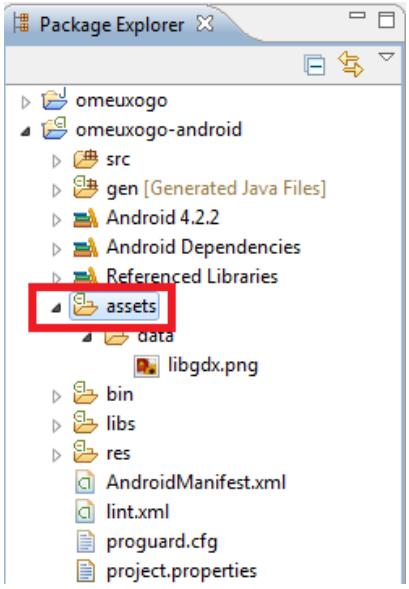
Se agora executamos a aplicación pediranos sobre que dispositivo virtual / real (se hai algún conectado) vai executar a aplicación.



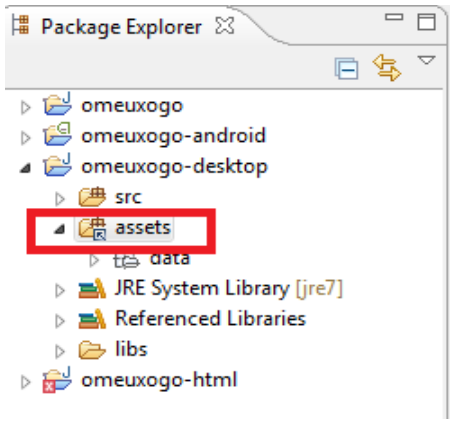
Se non existe un dispositivo xa en execución podemos lanzar un novo escollendo a opción 'Launch a new Android Virtual Device'.

Imos analizar a estrutura dos proxectos.

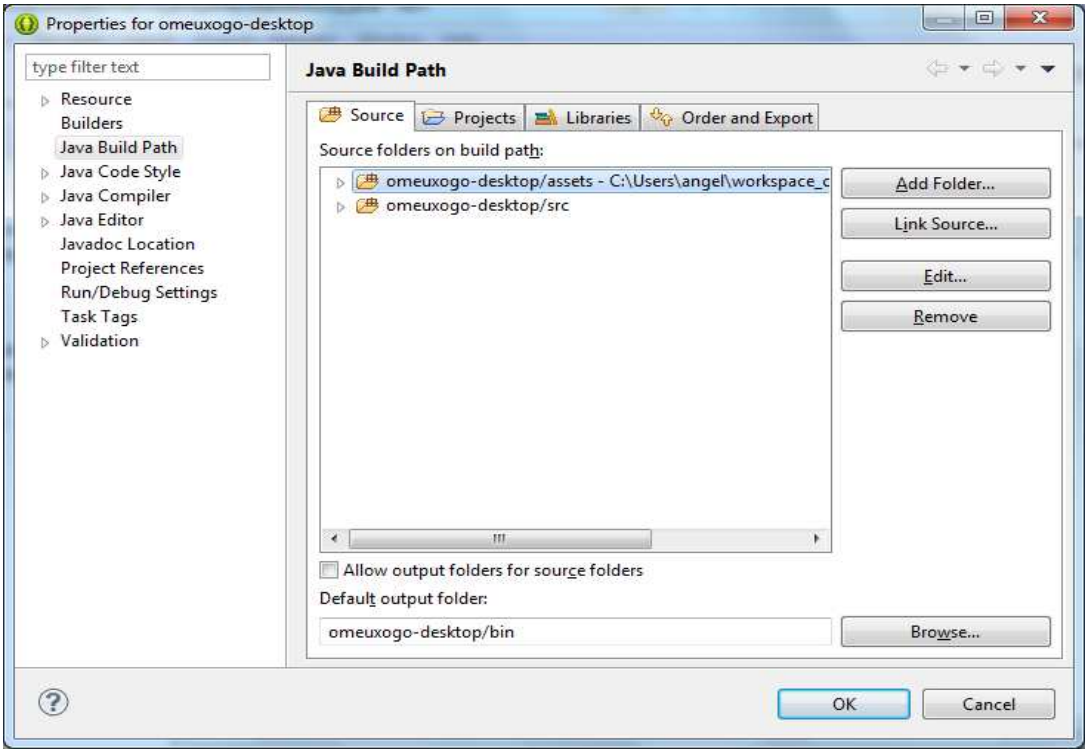
Na versión Android da nosa aplicación se atopa o cartafol Assets. É dentro deste cartafol onde temos que gardar todos os nosos recursos gráficos, música, efectos de son,....xa que a este cartafol é a onde vai dirixirse o resto de versións.



Podemos comprobar coma na versión Desktop o cartafol assets non é máis que un 'acceso directo' o cartafol creado na versión Android.

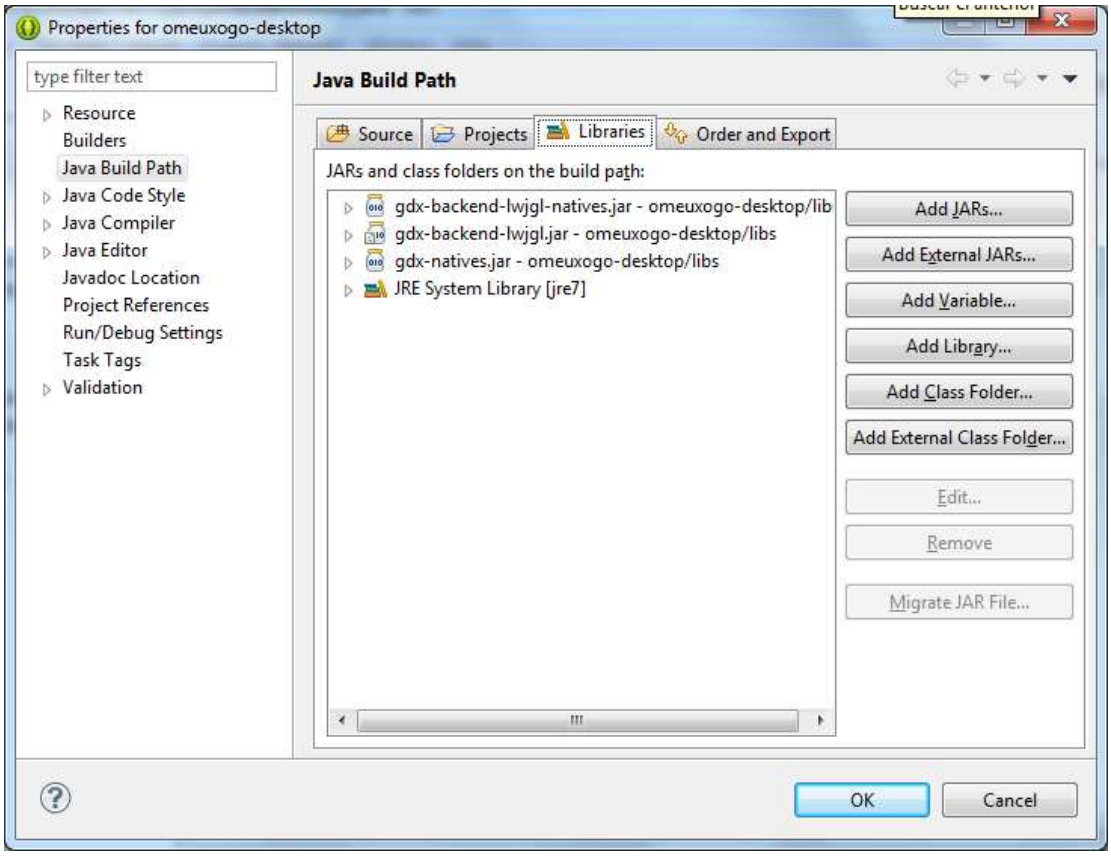


Se imos as propiedades do proxecto e marcamos a opción 'Java Build Path' podemos comprobar que dito cartafol é un 'Link Source' apuntando o cartafol Assets da versión Android:

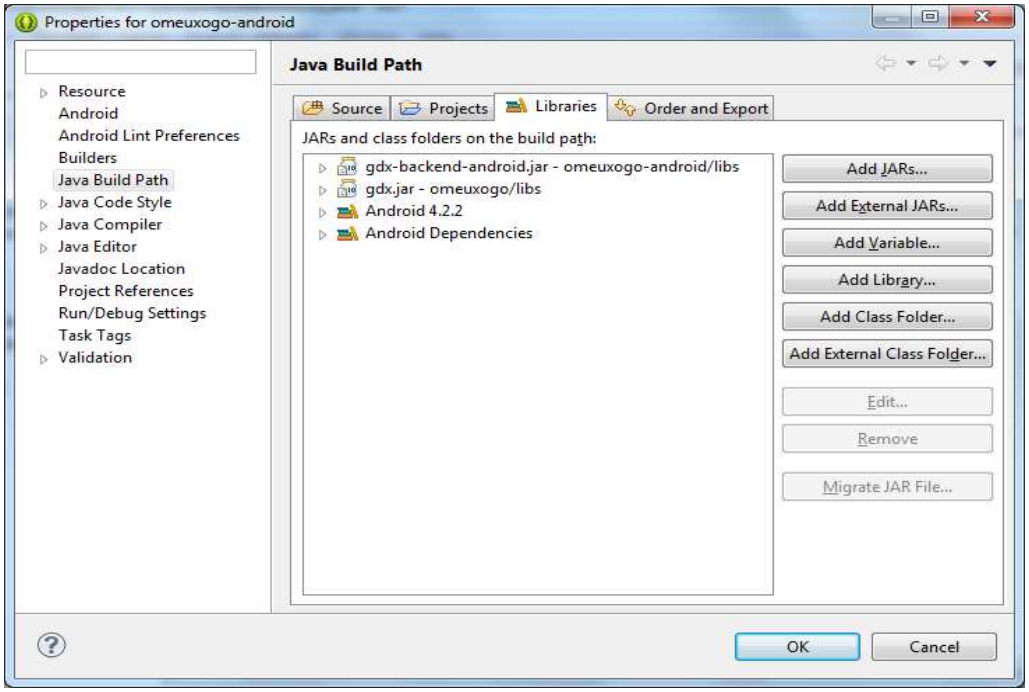


NOTA IMPORTANTE: cando facemos a importación de proxectos ó eclipse pode suceder que o enlace non funcione e sexa necesario volver a seleccionar ó cartafol da versión Android (premendo o botón Edit da pantalla anterior).
En Windows7 os arquivos '.project' e '.classpath' que están no cartafol raíz nas diferentes versións (desktop,html,android,...) aparecen como ocultos. É necesario quitarlle dito atributo.

Na mesma pantalla (se prememos na pestana Libraries) podemos ver que librerías do motor de xogos libgdx son necesarias para desenrolar os xogos:



O mesmo o podemos facer coa versión Android.



Agora analicemos as clases que inician o xogo.
No caso da versión Desktop é unha clase Java cun método Main.

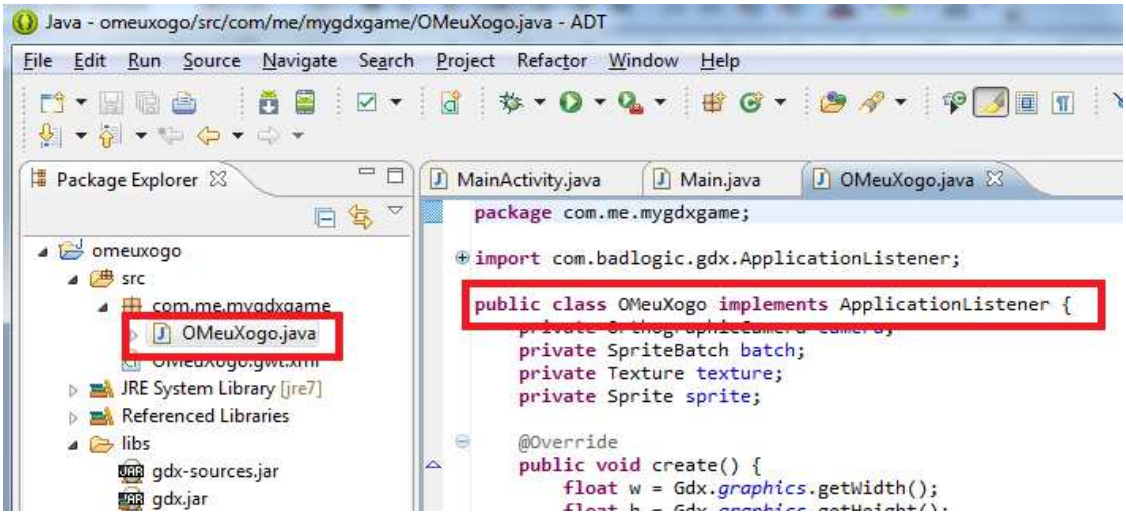
```
public static void main(String[] args) {
	LwjglApplicationConfiguration cfg = new LwjglApplicationConfiguration();
	cfg.title = "O meu xogo";
	cfg.useGL20 = false;
	cfg.width = 480;
	cfg.height = 320;

	new LwjglApplication(new OMeuXogo(), cfg);
}
```

O que fai dito método é preparar un obxecto cos datos necesarios para executar o xogo, no noso caso indica que:

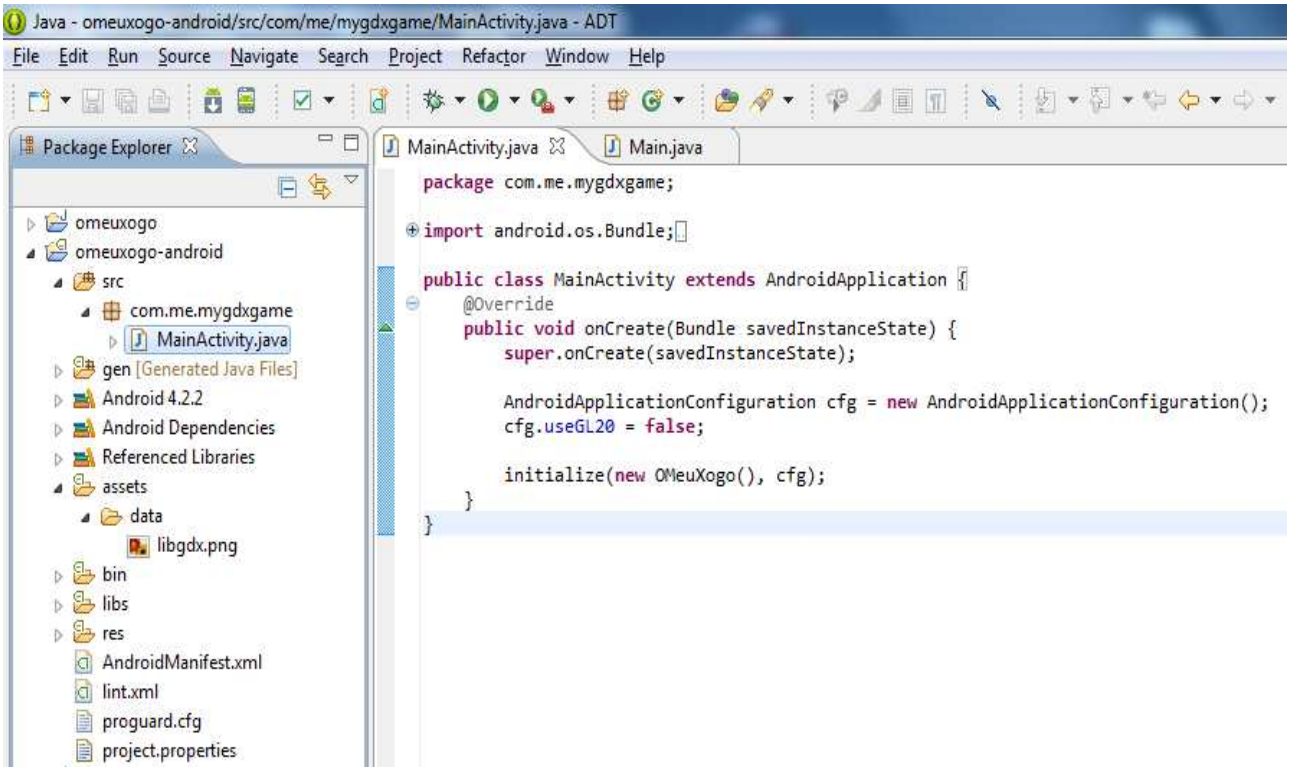
- O título do xogo será: O meu xogo (aparece na parte superior da xanela cando se executa o xogo)
- Non utilizará a versión de open gl 2.0
- Indica o ancho e alto en pixeis da pantalla onde vai executarse a aplicación.

Despois crea un obxecto da clase `LwjglApplication` mandando coma parámetros a configuración indicada anteriormente e un obxecto da clase OmeuXogo.

Dita clase está creada no proxecto 'COMUN' das diferentes versións (Desktop / Android / Html / IOS). No noso caso dita clase está definida no proxecto 'omeuxogo':



Vexamos agora a versión Android:



O código que lanza a aplicación é diferente xa que o que estamos a desenrolar é unha aplicación Android (a clase deriva de AndroidApplication e se inicializa no método onCreate) pero os conceptos son os mesmos: creamos un obxecto coa configuración a utilizar e lanzamos un obxecto da clase OmeuXogo con dita configuración.

Algo parecido temos na versión HTML.
A partires deste punto só temos que desenrolar o noso xogo no proxecto común (omeuxogo).

Antes analicemos os parámetros de configuración na versión Android.

Entro outras opcións temos as seguintes:

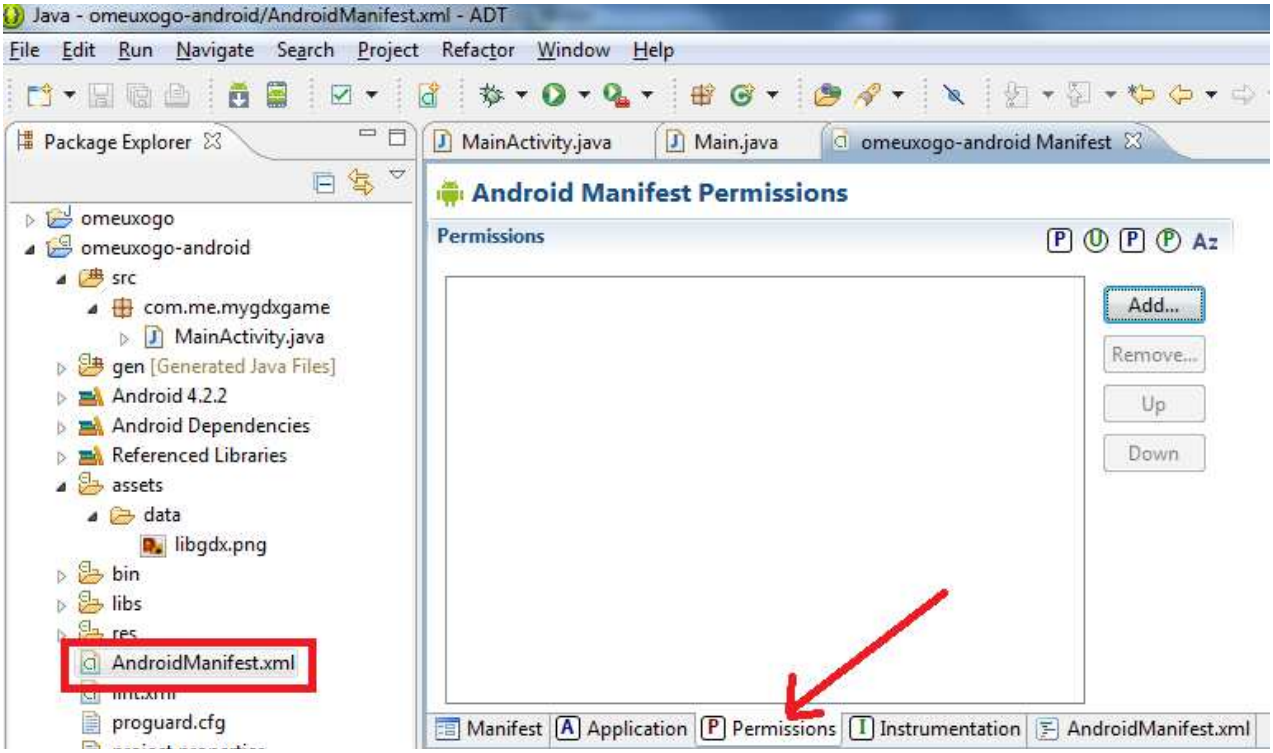
```
cfg.useAccelerometer=false;
cfg.useCompass=false;
cfg.useWakelock=true;
```

As dúas primeiras fan referencia ó uso do acelerómetro (http://es.wikipedia.org/wiki/%C3%B3metro) e o compás . Se o noso xogo non vai facer uso deste hardware é recomendable desactivalo na configuración para aforrar batería.

A terceira opción é necesaria para que o S.O. Android non pase ó estado de 'durmido' cando non recibe ningunha sinal por parte do xogo.

En Android dita opción leva consigo ter dito permiso por parte do S.O. Para indicarlle ó S.O. que a aplicación vai usar dito permiso será necesario modificar o arquivo AndroidManifiest.xml que se

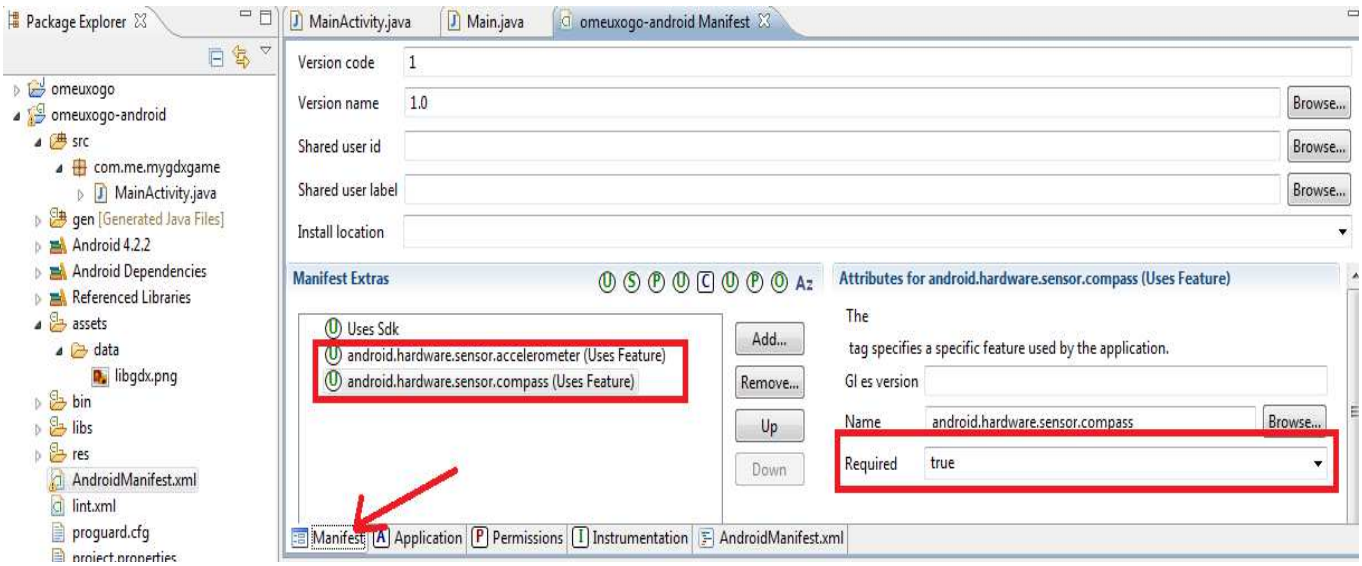
atopa no raíz do cartafol Android.



Premer o botón Add e escoller o permiso de tipo 'User Permission'.

Da lista que aparece escoller 'android.permission.WAKE_LOCK' e premer o botón de 'Gardar' en Eclipse.

As outras dúas opcións non necesitan permisos, pero podemos indicar ó S.O que imos utilizar dito hardware e se temos a aplicación / xogo no Market podemos facer que non se poida instalar se o dispositivo non dispón de dito hardware.



Podemos ver a lista completa de dispositivos en:

http://developer.android.com/guide/topics/manifest/uses-feature-element.html#hw-features

Se escollemos a opción 'Required' indicamos que é necesario dito hardware para que o xogo funcione.
No noso caso polo de agora non imos necesitar especificar dito hardware.

Agora imos analizar a clase que usamos en todas as versión do noso xogo: OmeuXogo.java.

Dita clase incorpora a interface `ApplicationListener`. Por medio de dita interface dispoñemos dos seguintes métodos:

```
@Override
public void create() {          => Cando se crea
}

@Override
public void dispose() {         => Cando se libera
}

@Override
public void render() {          => Chamado continuamente para facer o render
}

@Override
public void resize(int width, int height) {
                                => Cando se cambia a resolución da pantalla
}

@Override
public void pause() {           => O pausar (ou saír) da aplicación
}

@Override
public void resume() {          => O reanudar (ou iniciar) a aplicación
}
```

Podemos comprobar como se executan ditos eventos, informando ó log.
Cando se fai o log se indican dous datos: unha etiqueta e o texto do log. A etiqueta se usa para facer un filtro xa que no log aparecen moitas mensaxes que nos non xeramos e molestan.
Definimos polo tanto dita etiqueta na clase OMeuXogo da seguinte forma:

```
private static final String LOG = OMeuXogo.class.getSimpleName();
```

En negrilla indico que ese é o nome da clase onde está definida a propiedade.

Agora definimos un método para imprimir a mensaxe no log:

```
public static void imprimirLog(String mensaxe){
      Gdx.app.log(OMeuXogo.LOG,mensaxe);
}
```

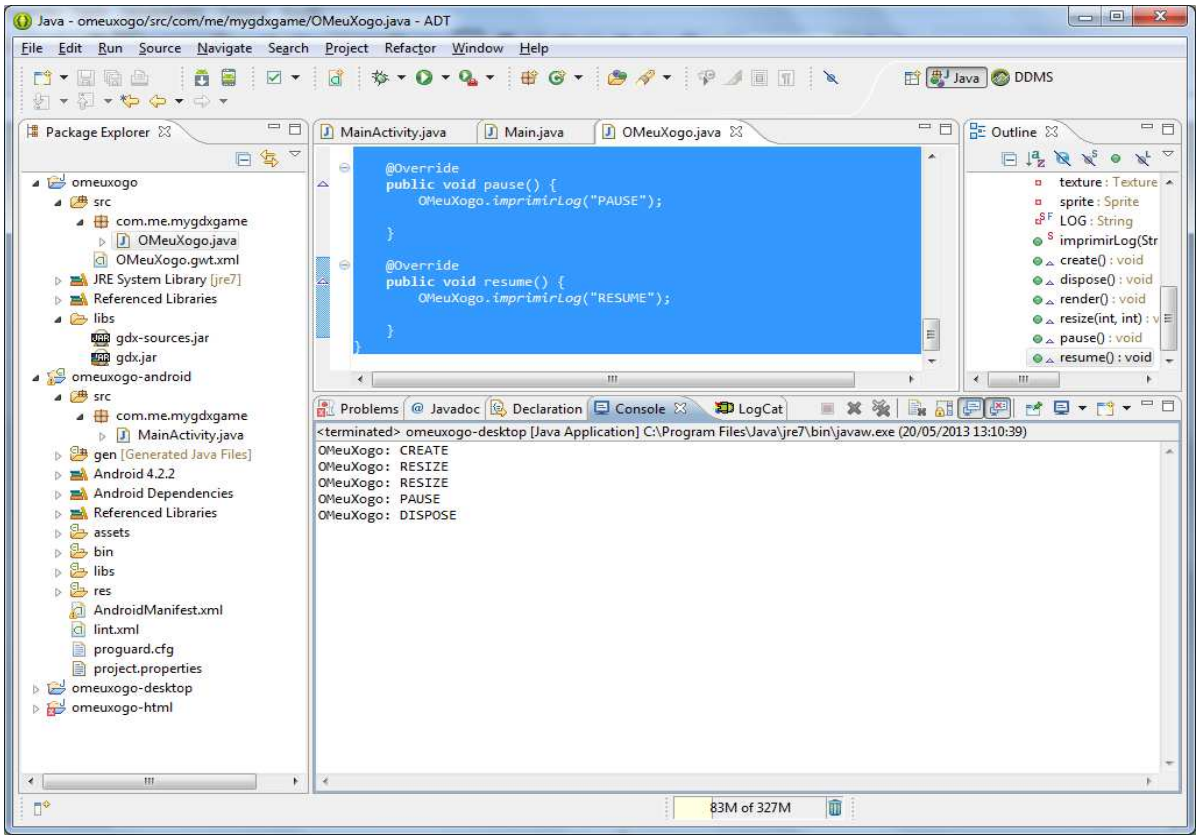
Podemos agora usar dito método para ver como pasa polos diferentes eventos (non poñer nada no

método render xa que se chama continuamente):

```
public class OMeuXogo implements ApplicationListener {
	private OrthographicCamera camera;
	private SpriteBatch batch;
	private Texture texture;
	private Sprite sprite;

	private static final String LOG = OMeuXogo.class.getSimpleName();

	public static void imprimirLog(String mensaxe){
		Gdx.app.log(OMeuXogo.LOG,mensaxe);
	}

	@Override
	public void create() {

		OMeuXogo.imprimirLog("CREATE");

		float w = Gdx.graphics.getWidth();
		float h = Gdx.graphics.getHeight();

		camera = new OrthographicCamera(1, h/w);
		batch = new SpriteBatch();

		texture = new Texture(Gdx.files.internal("data/libgdx.png"));
		texture.setFilter(TextureFilter.Linear, TextureFilter.Linear);

		TextureRegion region = new TextureRegion(texture, 0, 0, 512, 275);

		sprite = new Sprite(region);
		sprite.setSize(0.9f, 0.9f * sprite.getHeight() / sprite.getWidth());
		sprite.setOrigin(sprite.getWidth()/2, sprite.getHeight()/2);
		sprite.setPosition(-sprite.getWidth()/2, -sprite.getHeight()/2);
	}

	@Override
	public void dispose() {
		OMeuXogo.imprimirLog("DISPOSE");

		batch.dispose();
		texture.dispose();
	}

	@Override
	public void render() {

		Gdx.gl.glClearColor(1, 1, 1, 1);
		Gdx.gl.glClear(GL10.GL_COLOR_BUFFER_BIT);

		batch.setProjectionMatrix(camera.combined);
		batch.begin();
		sprite.draw(batch);
		batch.end();
	}

	@Override
	public void resize(int width, int height) {
		OMeuXogo.imprimirLog("RESIZE");
	}

	@Override
	public void pause() {
		OMeuXogo.imprimirLog("PAUSE");

	}

	@Override
	public void resume() {
		OMeuXogo.imprimirLog("RESUME");
```

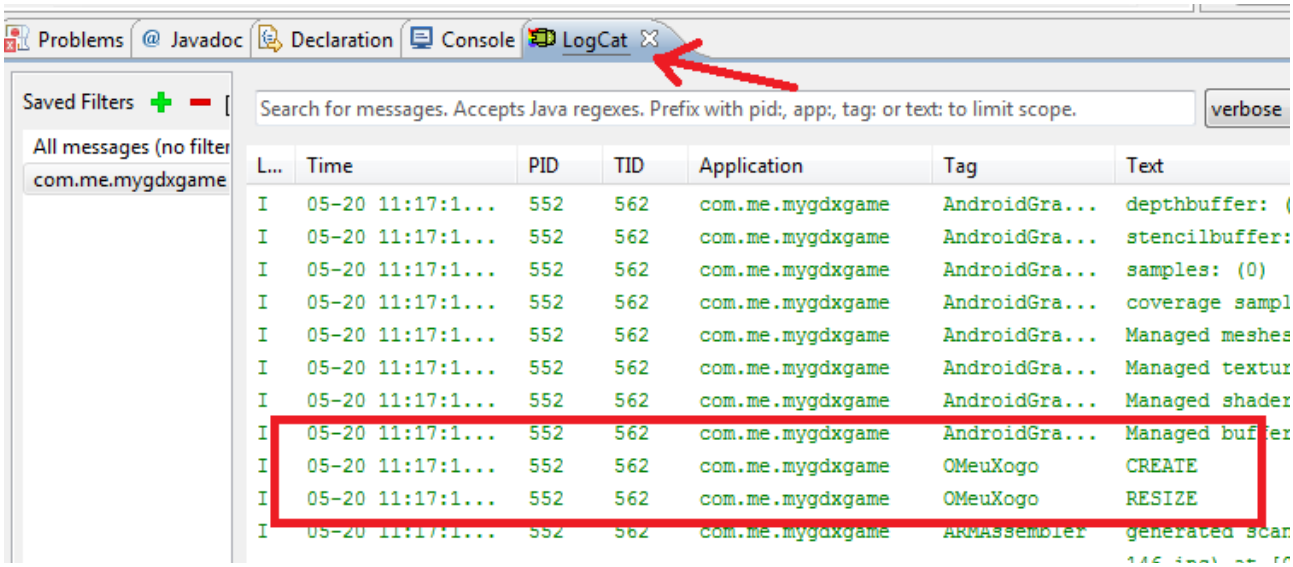
```
        }
}
```

Se executamos a versión Desktop, modificamos o tamaño e saímos podemos ver a secuencia de log na ventá 'Console' do Eclipse:

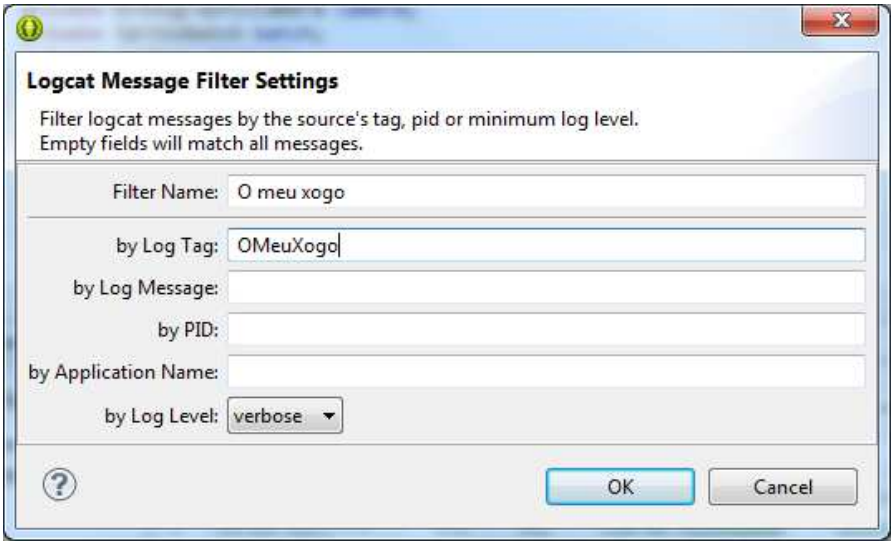


Nota: a ventá Console se pode amosar indo o menú de Eclipse Window => Show View => Console

Se o facemos na versión Android as mensaxes non aparece na pestana Console senón na pestana LogCat.

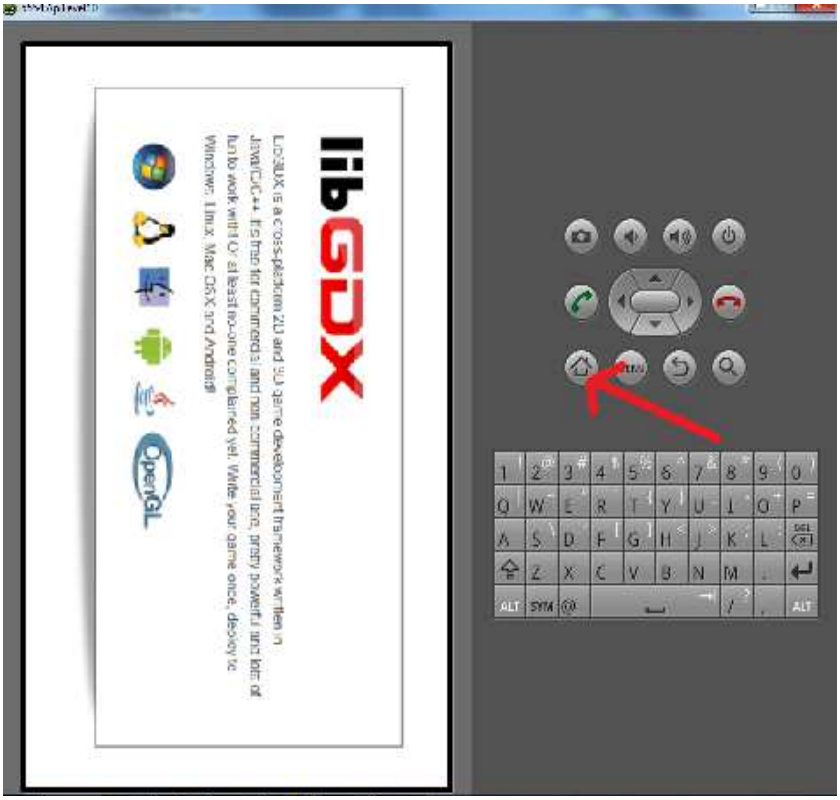


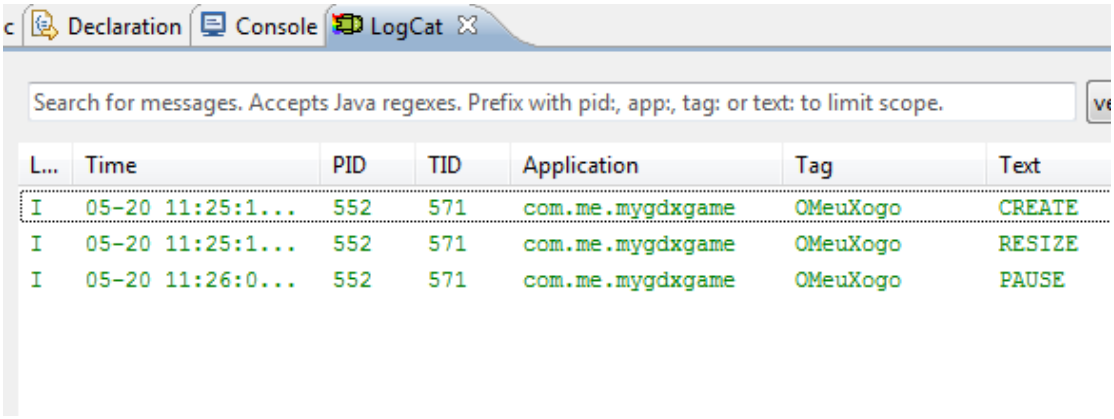
Podemos agora filtrar as mensaxes creando un novo filtro, premendo na parte esquerda no símbolo + (cor verde):



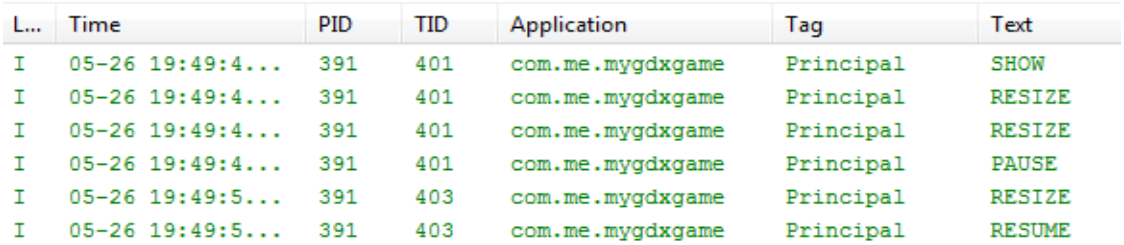
e poñendo como Log Tag o tag que definimos na clase (que se corresponde co nome da clase).

Se coa aplicación en execución prememos o botón de 'Casa' veremos como se executa o método PAUSE pero non o dispose.
Lembrar que cando se sae dunha aplicación Android esta queda en memoria ata que o sistema operativo a necesite que será cando realmente liberará a memoria da mesma.





Exemplo de log´s cando prememos 'Casa' no dispositivo Android.



Exemplo de log´s cando volvamos á aplicación dende Pause.

Isto o imos utilizar (facer log) bastantes veces para comprobar o funcionamento do noso xogo.

Outra utilidade que imos ter para desenrolar os nosos xogos é comprobar a que velocidade (fotogramas por segundo = fps) se executa o noso xogo.

Definimos a seguinte propiedade:

```
private FPSLogger fps;
```

O poñer dita liña dará un erro xa que a clase FPSLogger non está importada. Premer a combinación de teclas Ctrol+Shift+O para importalo automaticamente.

```
@Override
public void create() {

        OMeuXogo.imprimirLog("CREATE");

        fps = new FPSLogger();

        .............

@Override
public void render() {

        fps.log();
        ..............
```

Agora no log (escoller o filtro sen tag) aparecerán os FPS ó que se debuxa o noso xogo.

Nota importante: se non modificamos o proxecto ANDROID non vai compilar as novas modificacións cando executemos a versión de Android. Lembrade que ditas modificacións as estamos a facer sobre o proxecto común. É necesario por tanto modificar a clase de Android se queremos que instale a nova versión. Para isto chega, por exemplo, con poñer un espazo en branco na clase MainAcitivity.java, gardar e executar.

O DESENROLO DO XOGO.

Para desenrolar o xogo previamente a codificación do mesmo temos que ter en conta unha serie de cuestións.

Temos que ter:

- Un nome.
- Unha historia.
- Ten que pertencer a unha categoría.

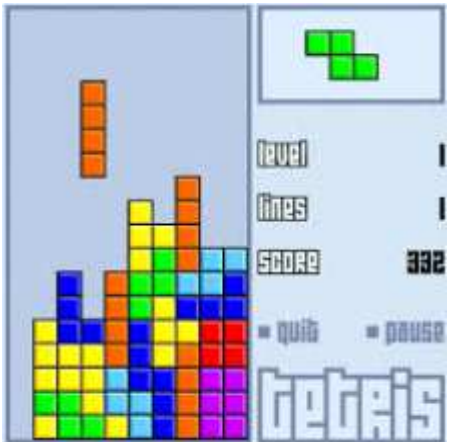
Neste derradeiro punto podemos atopar as seguintes:

- Xogo casual. Aquí poden entrar moitos tipos de xogos pero case todos eles teñan algunhas características comúns: son moi accesibles, incluso para aquelas persoas que non son etiquetadas como 'xogadores' habituais polo que aumenta o grupo de xogadores; cada sesión de xogo dura uns poucos minutos pero ó ser moi aditivos os xogadores poden estar horas xogando. Poden ser xogos de plataforma, de disparos ou de lanzar unha pelota a unha canastra.

Exemplo de xogo casual Archery


Exemplo de xogo casual Angry Birds

- Xogo de puzzle:



Exemplo de xogo de puzzle Tetris

- Xogo de acción e arcade.

Exemplo de xogo de acción Doom

Este xénero inclúe sub-xéneros como os xogos de plataforma, xogos de carreiras, xogos de disparos en primeira ou terceira persoa.

Exemplo de xogo de disparo Space Invader

- Xogos de Torres de defensa (Towers-Defense ou TD). O obxectivo deste tipo de xogos é lograr que as unidades inimigas non cheguen a cruzar o mapa. Para logralo débense construír torres que ataquen os inimigos ó pasar.

Exemplo de xogo Torres de defensa 'Desktop Tower Defense'

Máis información: http://es.wikipedia.org/wiki/Tower_defense
Xogos deste tipo: http://www.playtowerdefensegames.com/categories/1/defense-tower-games.html

- Innovación: Este tipo de xogos utilizan as novas funcionalidades que nos ofrecen os dispositivos Android (cámara, gps,...) para desenrolar xogos con novas experiencias.

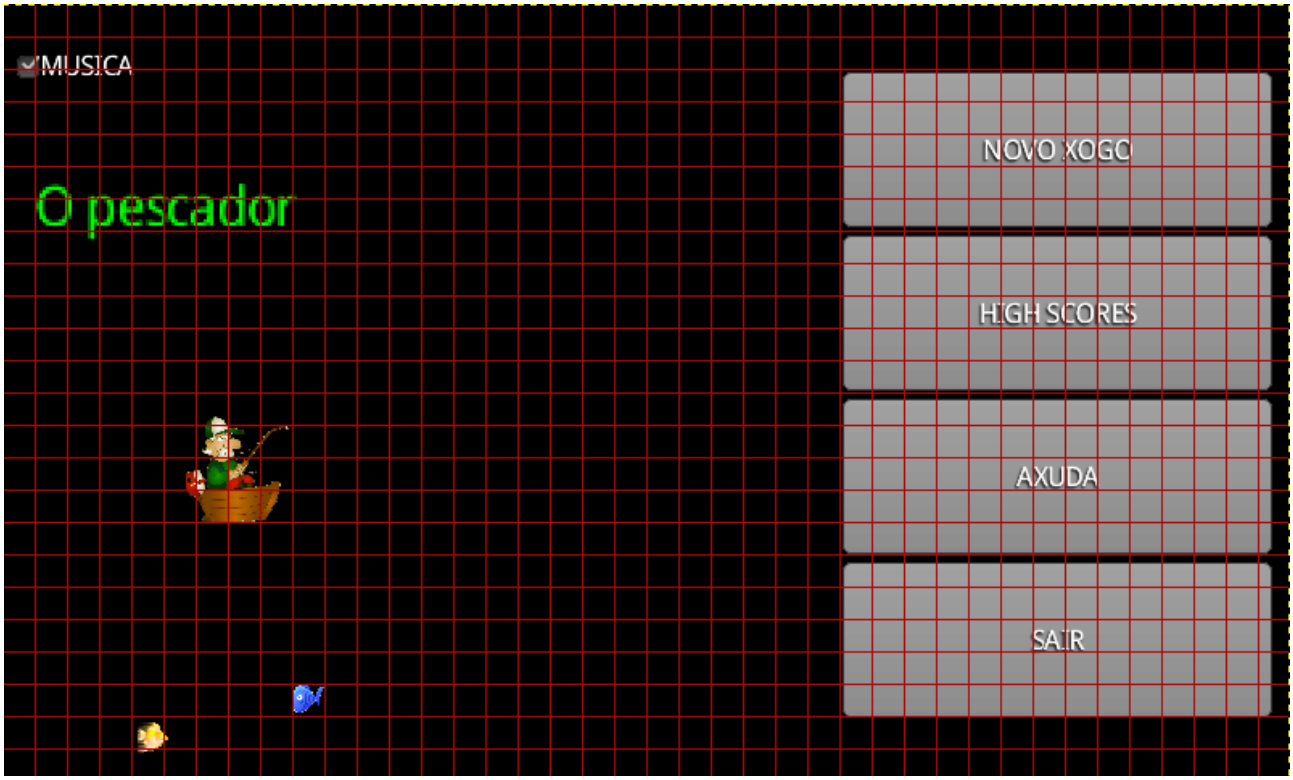
Exemplo de xogo innovador SpecTrek de SepecTreking.com

- Papel (cuadriculado) e lapis para:
  - Definir os elementos que integran as pantallas.
  - Definir as pantallas.
  - Crear un diagrama de transición entre pantallas.

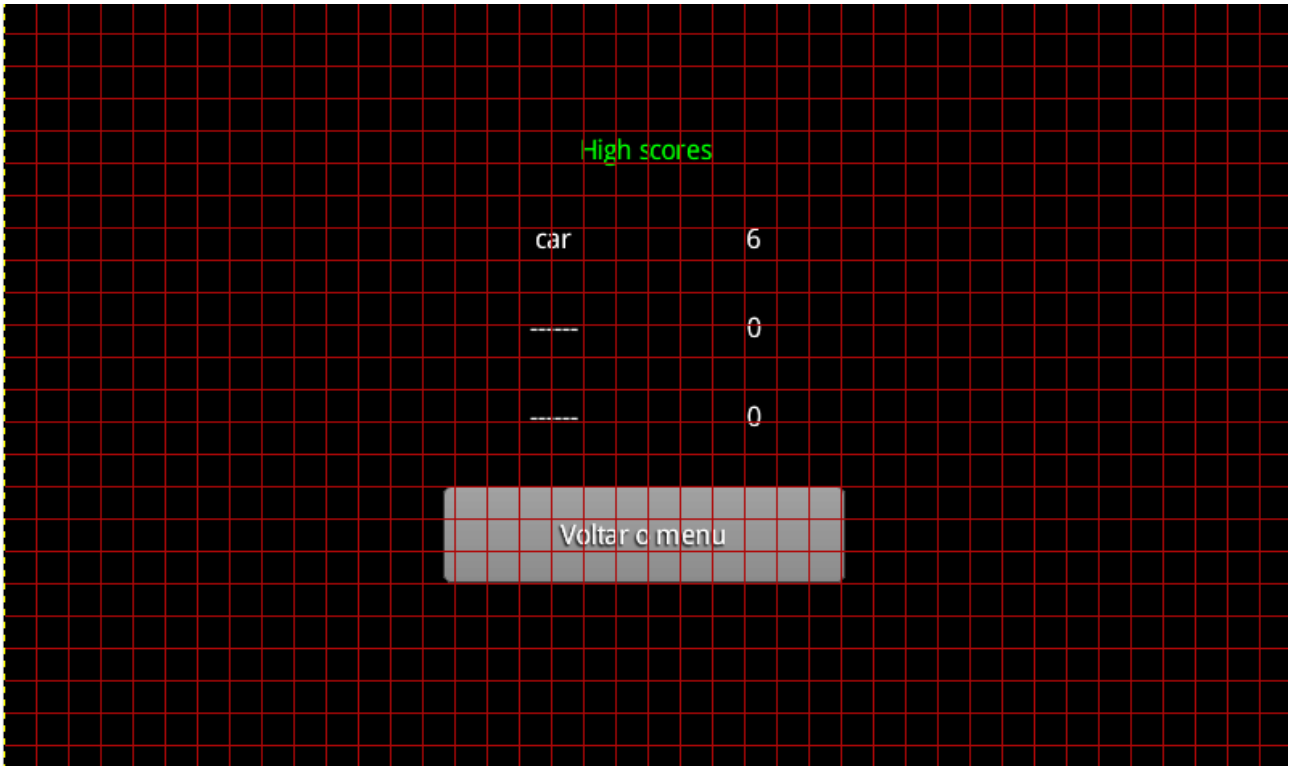
No noso caso:

Un nome: Caza ó peixe.
A historia: Xoán o neno pequeno dunha familia pobre ten que pescar moitos peixes para alimentar a súa familia. Estes van moi rápidos polo que só o mellor pescador pode pescalos.
Categoría: casual

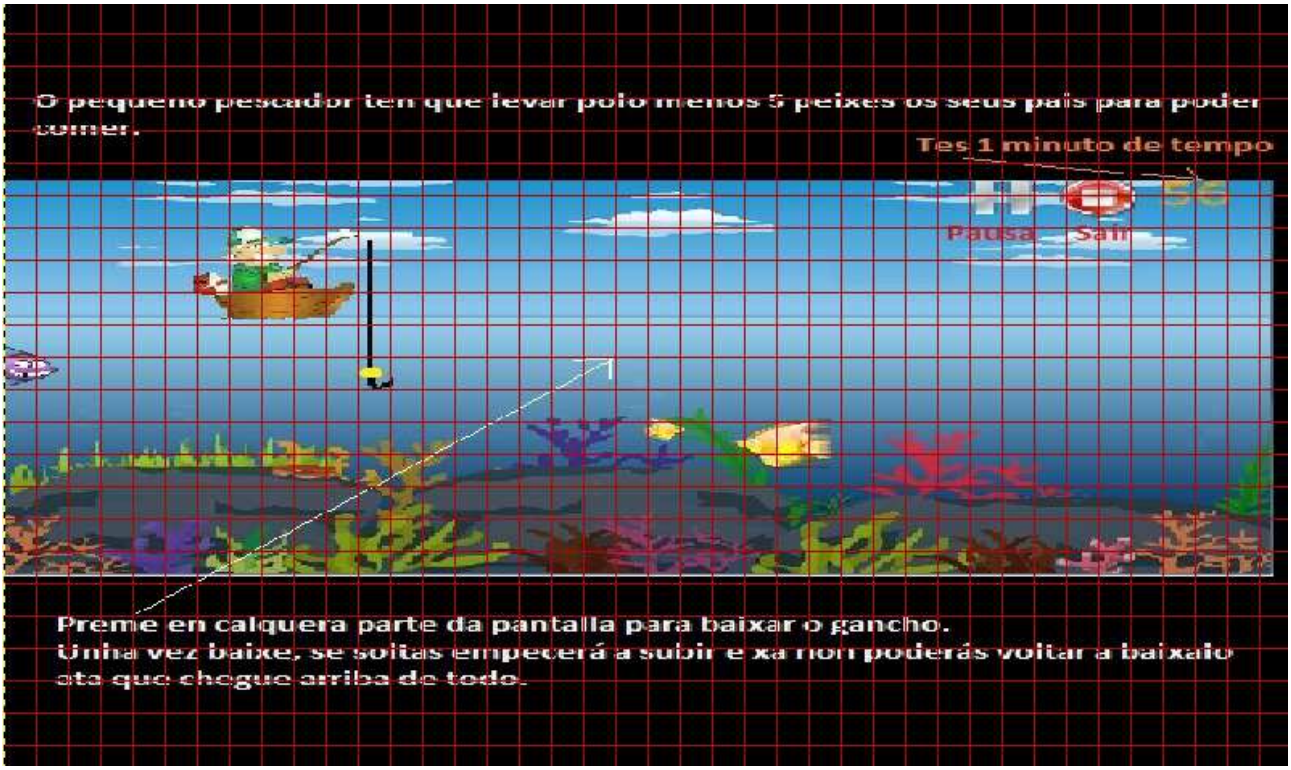
Papel: debemos de escribir en papel cuadriculado as diferentes pantallas e a transacción dunha a outra cun diagrama. O papel cuadriculado vainos servir para o tamaño-proporción dos compoñentes do noso xogo.



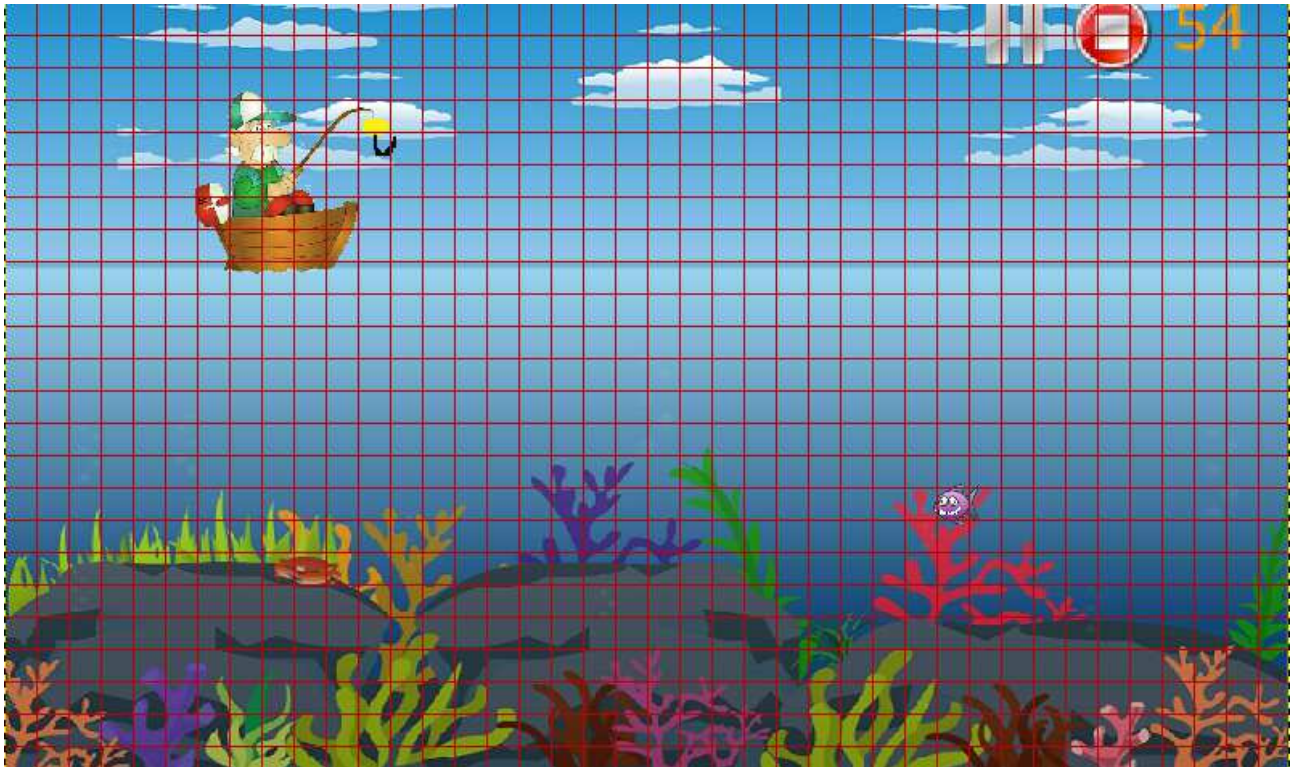
Pantalla Principal



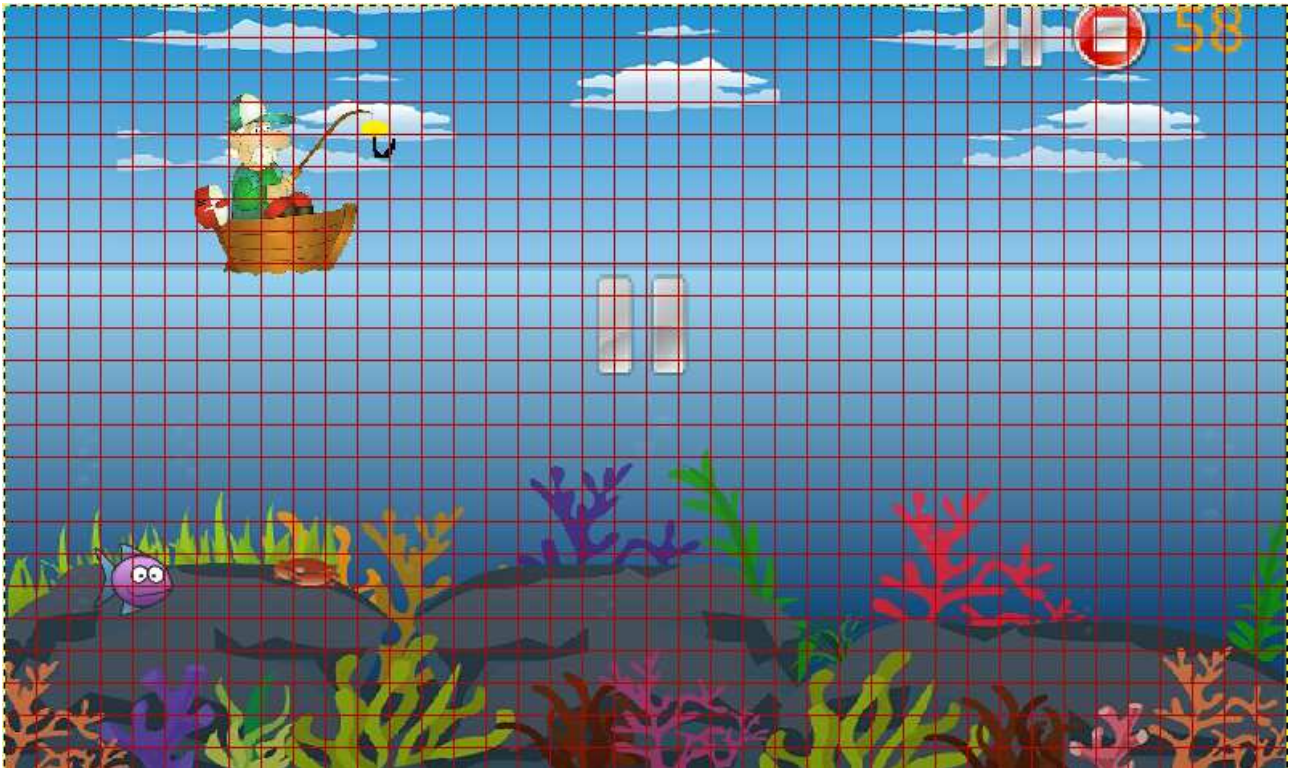
Pantalla High Scores



Pantalla Axuda

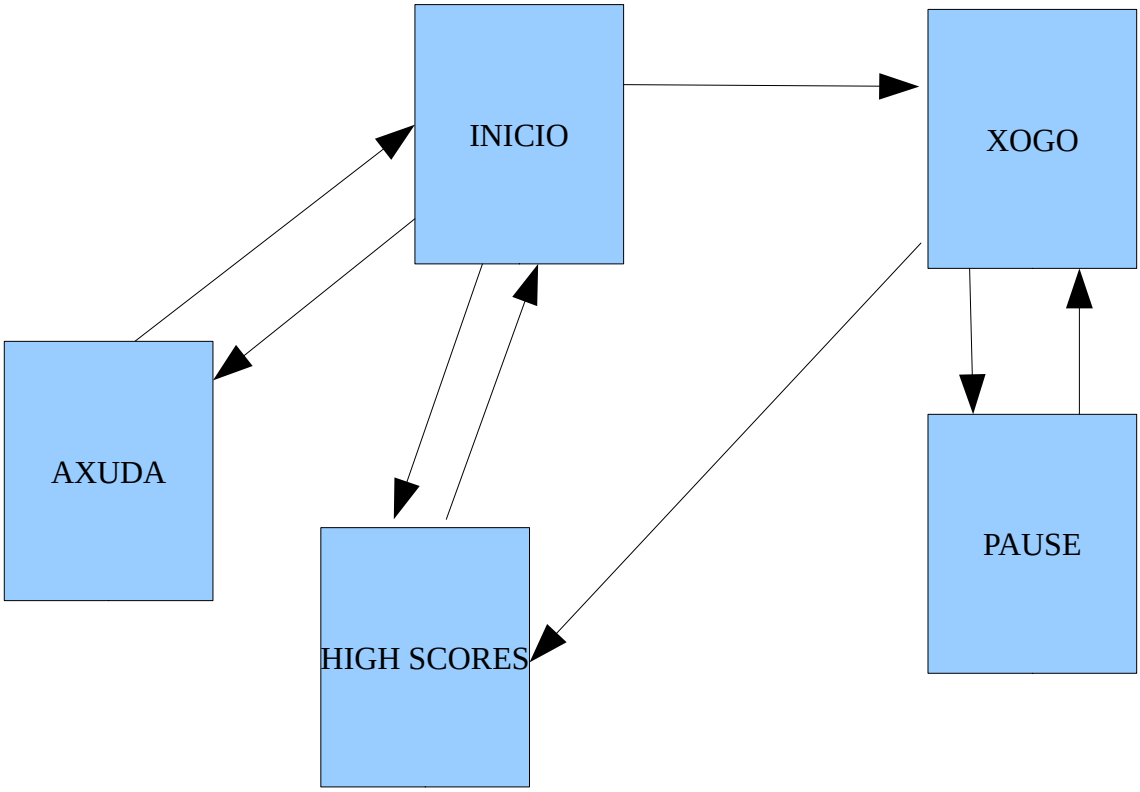


Pantalla do Xogo



Pantalla de Pause

Transaccións entre pantallas:



Como apuntamos anteriormente imos desenrolar o xogo para unha resolución das máis utilizadas, de 480x800. Podemos poñer esta mesma na versión Desktop para ver como queda.

DESENROLANDO O XOGO.

Imos crear a pantalla onde se vai desenrolar o xogo.
Para iso primeiro temos que facer as seguintes modificacións:

Creamos unha nova clase de nome PrincipalXogo que implemente a interface ApplitacionListener e que sexa chamada dende as diferentes versións:

Arquivo PrincipalXogo.java

```
public class PrincipalXogo implements ApplicationListener{

	@Override
	public void create() {
		// TODO Auto-generated method stub

	}

	@Override
	public void resize(int width, int height) {
		// TODO Auto-generated method stub

	}

	@Override
	public void render() {
		// TODO Auto-generated method stub

	}

	@Override
	public void pause() {
		// TODO Auto-generated method stub

	}

	@Override
	public void resume() {
		// TODO Auto-generated method stub

	}

	@Override
	public void dispose() {
		// TODO Auto-generated method stub

	}

}
```

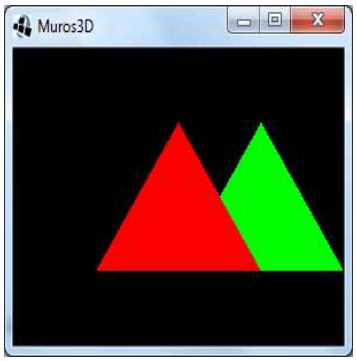
Exemplo: Main.java en versión Desktop:

```
public class Main {
	public static void main(String[] args) {
		LwjglApplicationConfiguration cfg = new LwjglApplicationConfiguration();
		cfg.title = "O Meu Xogo";
		cfg.useGL20 = false;
		cfg.width = 800;
		cfg.height = 480;

		new LwjglApplication(new PrincipalXogo(), cfg);
	}
}
```

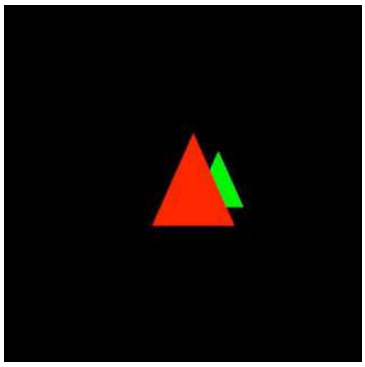
Agora imos cargar o fondo do noso xogo.
Para isto temos que facer uso dunha cámara.

No caso dos xogos 2D, a cámara é de tipo ORTOGRÁFICA frente a cámara utilizada en 3D que é unha cámara en PERSPECTIVA. A diferenza principal entre os dous tipos de cámara é a perspectiva. En 3D os obxectos mais afastados vense máis pequenos que os próximos.

Por exemplo:



Exemplo de cámara Ortográfica



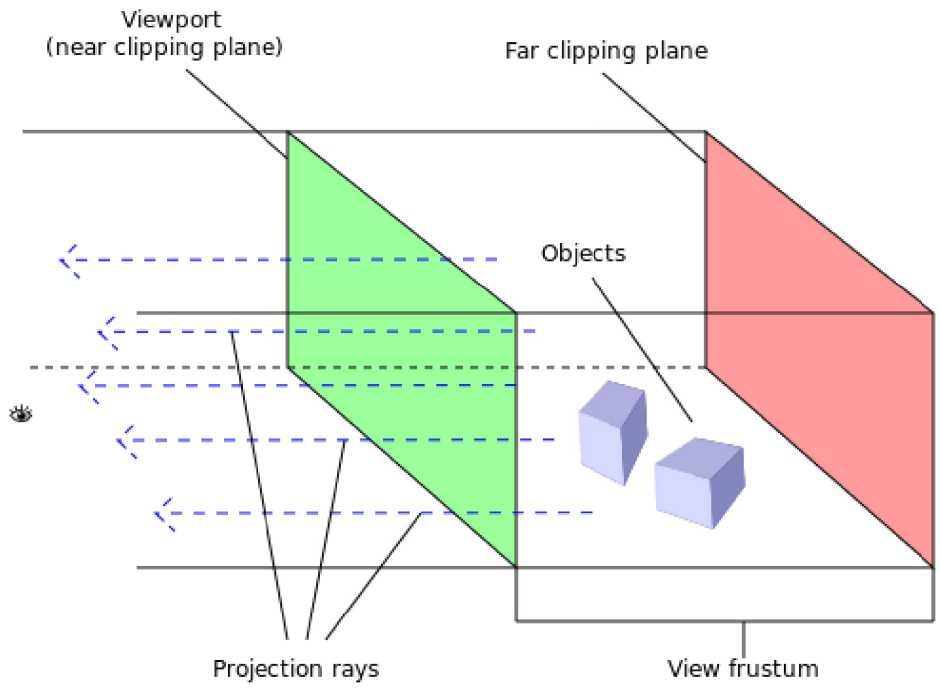
Exemplo de cámara en Perspectiva

Definimos dentro da clase a cámara que imos utilizar facendo uso da clase OrthographicCamera. A instanciamos no método create.

Arquivo PrincipalXogo.java:
Definimos a cámara.

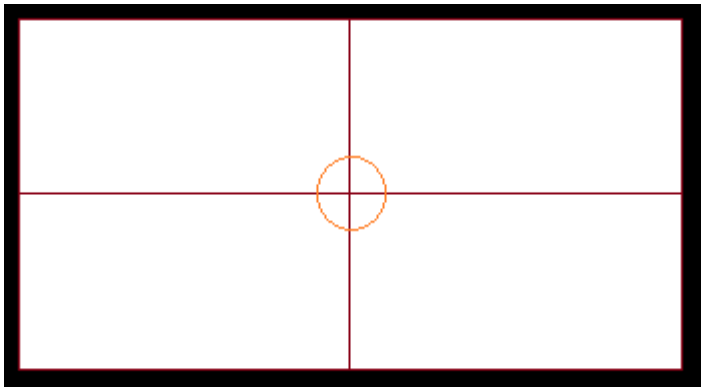
```
public class PrincipalXogo implements ApplicationListener{

	private OrthographicCamera camaraortho;

	@Override
	public void create() {
		// TODO Auto-generated method stub
		camaraortho = new OrthographicCamera();
	}
	.................
```

Agora temos que darlle un tamaño á cámara do que vai visualizar. Dito tamaño é o que se coñece como **VIEWPORT**.



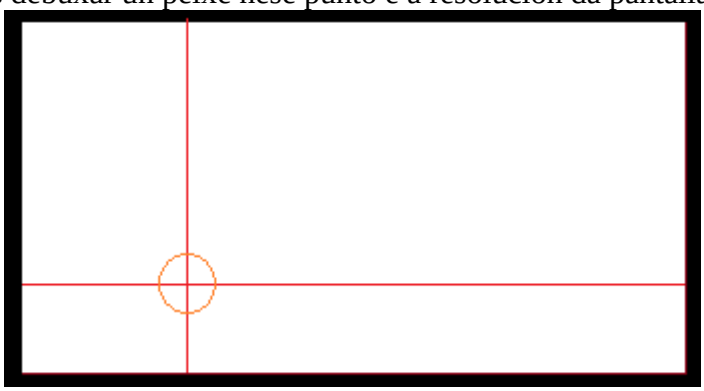
O que se atopa entre o plano near e o plano far é o que se vai visualizar, e o tamaño do que se ve (o viewport) está definido polo rectángulo verde que é igual o rectángulo rosa do fondo. A área que se vai a visualizar é o que se coñece como ViewFrustrum.

O tamaño da pantalla pode variar xa que varía a resolución da mesma. Se aplicamos como tamaño do noso xogo dita resolución non podemos posicionar de forma absoluta os nosos protagonistas, xa que se por exemplo:



Nesta pantalla cunha resolución de 400x200 o punto central estaría na coordenada 200x100.

Pero se nos mandamos debuxar un peixe nese punto e a resolución da pantalla cambia a 800x400:



Como vemos a posición non é a mesma.
Para evitar dito problema podemos 'invetarnos' unhas unidades de medida para o noso xogo.
Por exemplo, segundo vimos antes unha das resolucións máis usadas é a de 800x480 isto da unha relación de aspecto de 800/480 = 1,66. Imos facer que o noso mundo teña un tamaño de 332x200 e así mantemos a nosa relación de aspecto (332/200=1,66).

Polo tanto imos crear unha nova clase (para que quede máis claro) de nome Mundo, onde definimos o tamaño do noso mundo usando unha clase de libgdx denominada Vector2.

Arquivo Mundo.java

```
public class Mundo {

	/**
	 * Indica o tamaño do noso mundo dentro do videoxogo. Inicialmente 332x200
	 */
	public final static Vector2 TAMAÑO_MUNDO = new Vector2(332,200);

}
```

Agora modificamos o método resize da clase PrincipalXogo para darlle o tamaño á cámara ortográfica.

¿ Por que facelo nese método ?
Porque máis adiante veremos como podemos facer cando a resolución da pantalla onde se vai a executar o xogo non ten esta relación de aspecto. Sería dentro deste método onde teríamos que modificar a relación de aspecto da cámara para adaptala á da pantalla ou ben utilizar dito factor para debuxar todo en función do mesmo.

Arquivo PrincipalXogo.java
Modificamos o método resize.

```
@Override
public void resize(int width, int height) {
        // TODO Auto-generated method stub
        camaraortho.setToOrtho(false,Mundo.TAMAÑO_MUNDO.x,Mundo.TAMAÑO_MUNDO.y);
        camaraortho.update();
}
```

Nota: información sobre a relación de aspecto: http://es.wikipedia.org/wiki/Relaci%C3%B3n_de_aspecto

O método para modificar o tamaño da cámara é setToOrtho e leva tres parámetros. O primeiro indica se queremos que a coordenada 'y = 0' sexa abaixo (valor false) ou empeza arriba (valor true).

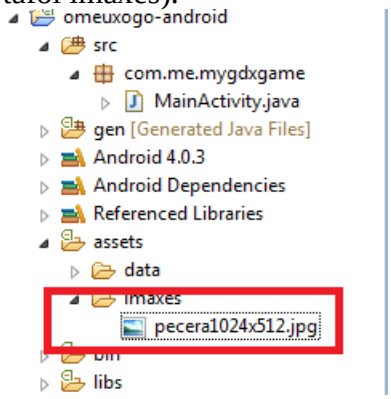
Nota importante: calquera modificación que se faga sobre a cámara debe chamarse o método update.

Que estamos a facer cando dicimos que a nosa cámara vai ter unhas coordenadas de 332x200 ?

Imaxinemos que temos un dispositivo móbil cunha resolución en pantalla de 800x480. Cando tocamos sobre a pantalla daranos unha coordenadas. Así o centro da pantalla será (400,240). O que imos facer nos será 'proxectar' ditas coordenadas reais a coordenadas da nosa cámara (xa o veremos máis adiante) dándonos unhas coordenadas de (166,100) que é o centro da pantalla da nosa cámara.

Agora temos que debuxar o fondo do noso xogo. Ó gráfico utilizado imos chamarlle **TEXTURA**.
A textura co fondo se atopa no cartafol do DVD /Proxectos Eclipse/Proxecto 2D peixes/1-Version sen gancho/omeuxogo-android/assets/imaxes/pecera1024x512.jpg

Temos que copialo ó cartafol 'assets' do proxecto Android. Podemos crear unha estrutura de cartafois dentro de assets (crear cartafol imaxes).



Nota: Ó non utilizar opengl es 2.0 as texturas teñen que ser potencia de dous (2,4,8,16,32,64,...) no seu ancho e alto. Veremos máis adiante como podemos utilizar unha textura (TextureAtlas) no que estean debuxadas todas as texturas que necesitemos e soamente este arquivo que engloba a todas terá que ser potencia de dous.

Para poder referenciar as texturas dende calquera parte e debido a que cando saímos da aplicación (no caso da versión Android) o contexto gráfico se perde (é dicir, as referencias a todos os recursos gráficos que teñamos xa non valen cando volvemos á aplicación) imos cargar ditas texturas dende unha clase separada na que todas as texturas son de clase (Static).

Arquivo AssetsXogo.java
Creamos a propiedade que carga a textura de fondo e dous métodos para inicialas e liberalas.

```
public class AssetsXogo {

	/**
	 * Textura do fondo
	 */
	public static Texture fondopecera;

	/**
	 * Método encargado de cargar todas as texturas
	 */
	public static void cargarTexturas(){
		FileHandle imageFileHandle = Gdx.files.internal("imaxes/pecera1024x512.jpg");

		fondopecera = new Texture(imageFileHandle);

	}

	/**
	 * Método encargado de liberar todas as texturas
	 */
	public static void liberarTexturas(){
		if (fondopecera!=null)
			fondopecera.dispose();
	}
}
```

Nota: Gdx.files.internal fai referencia o cartafol 'assets' do proxecto Android.

Nota: En Android, cando cambiamos de aplicación (por exemplo se atendemos a unha chamada) pérdese o que se denomina 'contexto de opengl'. Isto significa que todas as texturas cargadas deixan de ter validez cando se volta a aplicación, polo que é necesario cargalas de novo, pero o telas definidas como static isto non é necesario.

Imos facer que cando se inicie o xogo cargue as texturas e cando remate que as libere.

Arquivo PrincipalXogo.java
Modifcamos os métodos create,

```
	@Override
	public void create() {
		// TODO Auto-generated method stub
		AssetsXogo.cargarTexturas();

		camaraortho = new OrthographicCamera();
	}

	@Override
```

```
	public void pause() {
		// TODO Auto-generated method stub

	}

	@Override
	public void resume() {
		// TODO Auto-generated method stub


	}

	@Override
	public void dispose() {
		// TODO Auto-generated method stub
		AssetsXogo.liberarTexturas();

	}
```

Agora necesitamos debuxar dita textura. Para facelo definimos o un obxecto que pertence á clase SpriteBatch.
Establecemos que dito obxecto vai debuxar de acordo ó sistema de coordenadas definido na cámara (332x200). Isto o facemos no método resize.
Definimos o método render e preparamos o obxecto spritebatch para debuxar.

Arquivo PrincipalXogo.java

Definimos o obxecto SpriteBatch, o instanciamos no método resize e preparamos para debuxar no método render.

```
public class PrincipalXogo implements ApplicationListener{

	private OrthographicCamera camaraortho;
	private SpriteBatch spritebatch;


	@Override
	public void create() {
		// TODO Auto-generated method stub
		camaraortho = new OrthographicCamera();
		spritebatch = new SpriteBatch();
	}

	@Override
	public void resize(int width, int height) {
		// TODO Auto-generated method stub
		camaraortho.setToOrtho(false,Mundo.TAMAÑO_MUNDO.x,Mundo.TAMAÑO_MUNDO.y);
		camaraortho.update();

		spritebatch.setProjectionMatrix(camaraortho.combined);
	}

	@Override
	public void render() {
		// TODO Auto-generated method stub
		Gdx.gl.glClearColor(0, 0, 0, 1);		// Borra a pantalla Red-Green-Blue-Alfa
		Gdx.gl.glClear(GL11.GL_COLOR_BUFFER_BIT);

		spritebatch.begin();

		spritebatch.end();


	}
```

Imos analizar pouco a pouco todo o que estamos a facer.

1) Definimos un obxecto da clase SpriteBatch e o instanciamos no método create.
2) No método resize indicámoslle que o seu sistema de coordenadas sexa o definido na cámara. O que temos que ter claro neste momento e que estamos a debuxar puntos nos que se lles aplican operacións con matrices para 'colocalos' no seu sitio. Cando definimos a cámara, indicando un tamaño de viewport, estamos creando unha matriz (matriz de proxección) de tal forma que se aplicamos dita matriz a os puntos este se 'colocan' na posición adecuada.

```
spritebatch.setProjectionMatrix(camaraortho.combined);
```

   Con esta orde estamos a indicar o spritebatch que lle aplique a todo o que debuxe as transformacións gardadas na cámara ortográfica.

   Nota: isto temos que facelo sempre despois de chamar ó método update da cámara

3) No método render() borramos a pantalla en cada fotograma e facemos que todo o que se atope entre o begin e o end sexa enviado á vez a unidade gráfica para a súa visualización. Isto aumenta o rendemento xa que é moito mellor enviar un só gráfico que moitos á GPU (Graphics Process Unit).
   Nota: máis adiante explicaremos o borrado da pantalla.

Agora imos chamar a un método que debuxe o fondo. Dito método chamarase dibuxarFondo e será chamado dende o begin – end do obxecto spritebatch. Para debuxar unha textura só temos que usar o método draw do obxecto spritebatch, mandándolle a textura, posición e o tamaño do que se quere debuxar.

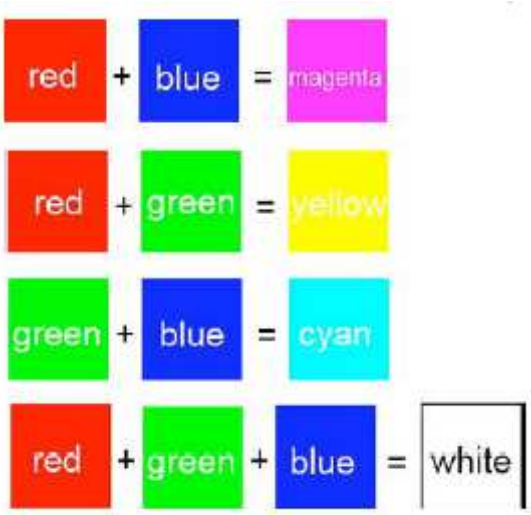
Arquivo PrincipalXogo.java
Creamos o método dibuxarFondo.

```
/** Dibuxa o fondo do xogo
 *
 */
private void dibuxarFondo(){
        spritebatch.draw(AssetsXogo.fondopecera,0,0,Mundo.TAMAÑO_MUNDO.x,Mundo.TAMAÑO_MUNDO.y);
}
@Override
public void render() {
        // TODO Auto-generated method stub
        Gdx.gl.glClearColor(0, 0, 0, 1);        // Borra a pantalla Red-Green-Blue-Alfa
        Gdx.gl.glClear(GL11.GL_COLOR_BUFFER_BIT);

        spritebatch.begin();

        dibuxarFondo();

        spritebatch.end();
}
```

Aumentando o rendimento: O nivel de transperencia.
As cores.

Cando debuxamos un gráfico, a textura ten información de que cor ten que ter cada pixel que se vai debuxar.
Esta información segue un modelo de cores. Nos imos a centrarnos no modelo RGB (Red Green Blue). Existen outros modelos coma YUV ou CMYK.

Con estas tres cores básicas (vermello, verde e azul) podemos ter calquera cor, mesturándoos na proporción correcta.
Por exemplo:



Información sacada do libro Beginning Android 4 Games

Ditos cores se especifican coa combinación de tres: RED – GREEN – BLUE.
Cada cor básico pode expresarse cun número flotante entre 0.0 – 1.0 (sendo 1 a intensidade máis grande e 0 a máis pequena).
Tamén podemos codificar cada cor cun byte con valor entre 0 a 255.

Dependendo das diferentes opcións podemos ter:

- Codificación RGB de 32 bits: ten 12 bytes por cada pixel (32 bits por cor (ó ser flotante) = 4 bytes x 3 cores=12 bytes). As intensidades poden varias entre 0.0 e 1.0
- Codificación RGB de 24 bits: ten 3 ou 4 bytes (se ten alfa, visto despois) por cada pixel (8 bits por cor = 1 bytes x 3 cores=3 bytes). As intensidades poden varias entre 0 e 255. A orde dos compoñentes pode ser RGB ou BGR. Se coñece coma RGB888 ou BGR888.
- Codificación RGB de 16 bits: ten 2 bytes por cada pixel. Red e blue teñen unha intensidade que varía entre 0 e 31 (5 bits x 2 = 10 bits) e o verde unha intensidade entre 0 e 63 (6 bits). A orde pode ser RGB ou BGR e se coñece coma RGB565 ou BGR565.

O normal e que traballemos co formato RGB888.

Formato de imaxes.

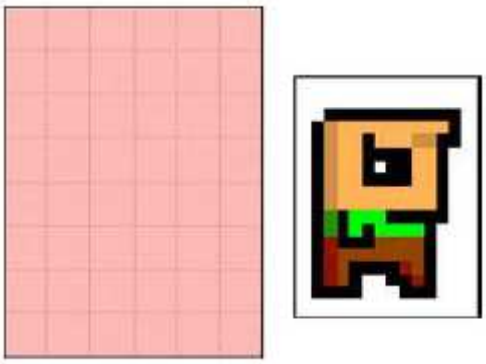
Á hora de gardar as imaxes temos multitude de formatos.
Dous dos máis coñecidos son: **JPEG e PNG**.

JPEG é un formato que perde calidade con respecto a imaxe orixinal.
PNG non perde calidade e é idéntica a imaxe orixinal.

Sempre que utilicemos unha imaxe que non requira transparencias (visto a continuación) nin unha calidade excesiva deberemos elixir jpg xa que ten un gran nivel de compresión.

Compoñente ALFA e transparencias.

A compoñente alfa e un valor que indica o nivel de transparencia dunha imaxe. Normalmente ó superpoñer unha imaxe sobre outra a imaxe superior 'tapa' a inferior. Pero no caso de gráficos como personaxes en movemento, queremos que só se superpoña a imaxe do personaxe. O problema ven que cando definimos un personaxe este se define cun rectángulo:



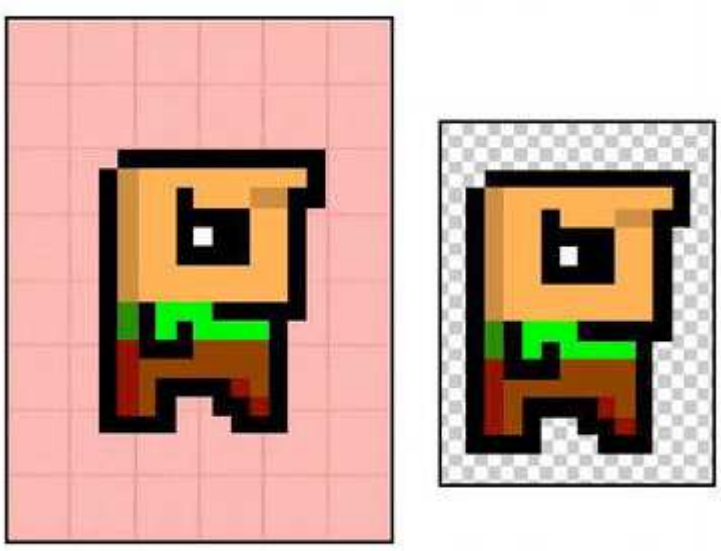
Información obtida do libro 'Beginning Android 4 Games'

O poñer enriba do fondo a imaxe do personaxe pasaría isto:

Grazas o compoñente alfa podemos establecer que unha determinada cor sexa transparente (o que se coñece como alfa).

Así, no caso anterior de formato gráfico, podemos falar de RGBA8888 (e as súas variantes). Con isto estamos a dicir que a compoñente alfa está composto por un valor que vai dende 0 (totalmente transparente) a 255 (totalmente opaco).

Se facemos que a cor branca sexa transparente teremos isto:

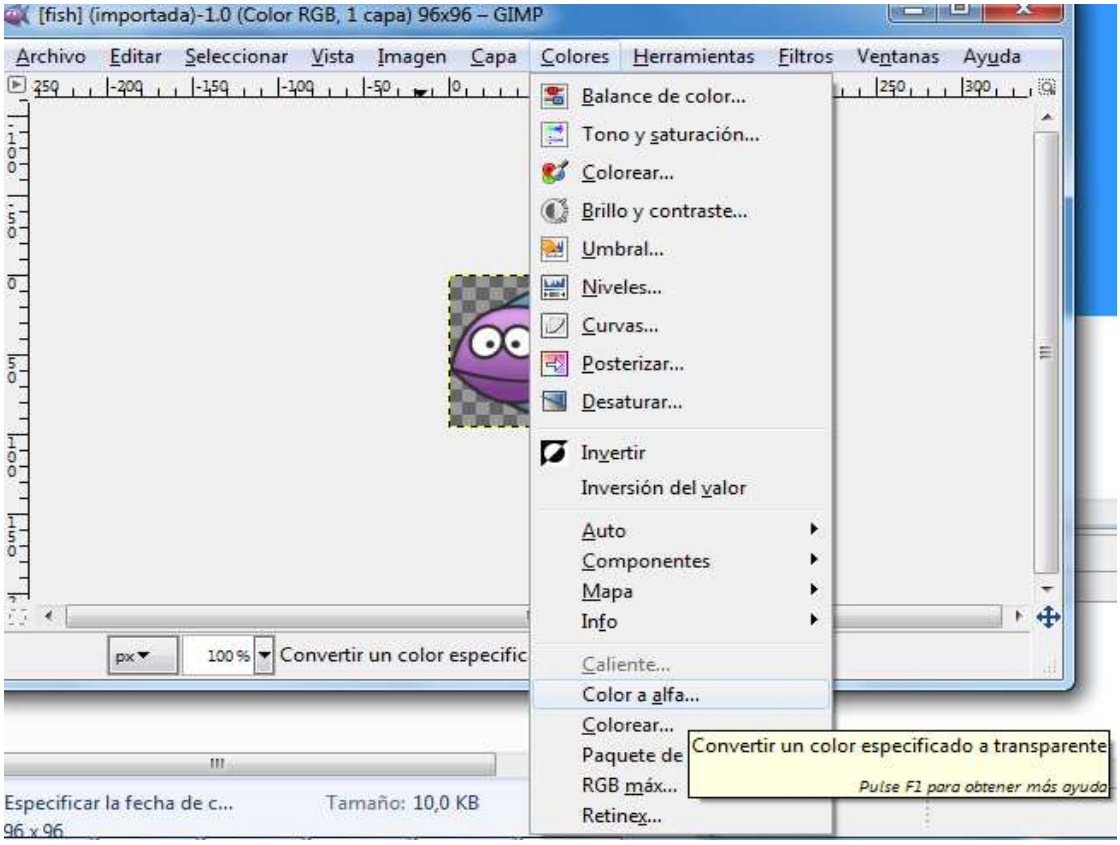


Nos imos utilizar o programa GIMP para facer este tipo de efectos.
Dito programa o podemos descargar dende http://www.gimp.org.es/ ou ben instalalo dende o DVD no cartafol /SOFTWARE/Graficos/gimp-2.8.2-setup-1.exe.

O que temos que ter claro é que **se a imaxe necesita transparencias non podemos gardalaen formato JPG xa que non o soporta**. Utilizaremos o formato PNG.

Vídeo de cómo hacer un fondo transparente en GIMP: https://www.youtube.com/watch?v=m-7j6bP94fg

Outra forma de facelo:



Exemplo de como cambiar unha cor e que sexa transparente.

En vez de suprimir cambiamos unha cor.

Voltando ó noso xogo.

Cando debuxamos o fondo este non vai ter transparencia ningunha, e para aumentar o rendemento podemos informarlle ó obxecto spritebach que desactive a mesma:

```
/** Dibuxa o fondo do xogo
 *
 */
private void dibuxarFondo(){
        spritebatch.disableBlending();
        spritebatch.draw(AssetsXogo.fondopecera,0,0,Mundo.TAMAÑO_MUNDO.x,Mundo.TAMAÑO_MUNDO.y);
        spritebatch.enableBlending();
}
```

Nota: a clase SpriteBatch ten varios métodos que podemos usar, coma por exemplo setColor(r,g,b,a) que debuxa un tinte na textura coa cor indicada e co nivel de transparencia que indiquemos (a = alfa; se é igual a 1 é totalmente opaco e con 0 totalmente transparente).

Agora que temos o fondo imos debuxar o pescador.
Para isto facemos o mesmo que co fondo, copiamos a textura do pescador (se atopa na solución do proxecto no DVD no cartafol /Proxectos Eclipse/Proxecto 2D peixes/1-Version sen

gancho/omeuxogo-android/assets/imaxes/pescador.png) ó cartafol assets/imaxes do noso proxecto.

Cargamos dita textura no método cargarTexturas e a liberamos.

Arquivo AssetsXogo.java
Cargamos a textura do pescador e a liberamos.

```
	public static void cargarTexturas(){
		FileHandle imageFileHandle = Gdx.files.internal("imaxes/pecera1024x512.jpg");
		fondopecera = new Texture(imageFileHandle);

		imageFileHandle = Gdx.files.internal("imaxes/pescador.png");
		pescador = new Texture(imageFileHandle);


	}
	public static void liberarTexturas(){
		if (fondopecera!=null)
			fondopecera.dispose();
		if (pescador!=null)
			pescador.dispose();
	}
```

Agora temos que saber onde debuxar dita textura. Para facelo máis fácil de manter imos crear unha clase Pescador que teña coma atributos a posición. Dita posición estará gardada nunha propiedade que pertenza á clase Vector2 que proporciona o framework e que permite gardar dous valores (x,y). Tamén imos gardar o tamaño da barca (con respecto ó noso sistema de coordenadas da cámara, é dicir, 332x200).

Arquivo Pescador.java
Definimos a posición e tamaño con Vector2. O tamaño será unha constante de clase.
Inicializamos no constructor a posición.
Creamos os método set e get para cada un deles.

Nota: Os métodos get e set pódense xerar de forma automática premendo o botón dereito sobre a pantalla de código e escollendo a opción 'Source' => Generate Getters and Setters.

```
/**
 * Representa o pescador coa barca
 * @author angel
 *
 */
public class Pescador {

	private Vector2 posicion;
	/**
	 * Tamaño do pescador. Inicialmente
	 */
	public static final Vector2 TAMAÑO = new Vector2(50,50);

	public Pescador(){
		posicion = new Vector2(50,128);
	}

	/**
	 * Devolve a posicion do pescador
	 * @return posicion
	 */
	public Vector2 getPosicion() {
		return posicion;
	}
```

```
	public void setPosicion(Vector2 posicion) {
		this.posicion = posicion;
	}

}
```

Imos a instanciar a clase Pescador dende o clase Mundo e crearemos en dita clase un método que devolva o pescador.

Arquivo Mundo.java
Definimos a propiedade Pescador e a instanciamos no constructor.
Creamos un método get para dita propiedade.

```
	/**
	 * O pescador do noso mundo
	 */
	private Pescador pescador;

	public Mundo(){
		pescador = new Pescador();
	}

	/**
	 * Devolve o pescador do noso mundo
	 * @return pescador
	 */
	public Pescador getPescador(){
		return pescador;
	}
```

Agora só temos que dibuxalo como fixemos co fondo pero utilizando os datos do pescador.
Debemos obter o pescador a través do método getPescador da clase Mundo.

Arquivo PrincipalXogo.java
Definimos unha propiedade mundo que instanciamos no create.
Definimos unha propiedade pescador que asinamos no create.
Creamos un método dibuxarPescador que dibuxe o pescador.

```
	private Mundo mundo;
	private Pescador pescador;
	@Override
	public void create() {
		// TODO Auto-generated method stub
		AssetsXogo.cargarTexturas();

		camaraortho = new OrthographicCamera();
		spritebatch = new SpriteBatch();

		mundo = new Mundo();
		pescador = mundo.getPescador();
	}
	/**
	 * Dibuxa o pescador
	 */
	private void dibuxarPescador(){

		spritebatch.draw(AssetsXogo.pescador,pescador.getPosicion().x,pescador.getPosicion().y,
				Pescador.TAMAÑO.x,Pescador.TAMAÑO.y);
	}
	@Override
```

```
public void render() {
        // TODO Auto-generated method stub
        Gdx.gl.glClearColor(0, 0, 0, 1);        // Borra a pantalla Red-Green-Blue-Alfa
        Gdx.gl.glClear(GL11.GL_COLOR_BUFFER_BIT);

        spritebatch.begin();

        dibuxarFondo();
        dibuxarPescador();

        spritebatch.end();
}
```

Agora temos que mover o pescador. Para isto imos a utilizar o concepto de velocidade. Para mover o soldado imos darlle unha velocidade. Inicialmente dita velocidade estará a 0. Cando premamos unha tecla ou movamos o móbil, aumentaremos a velocidade do pescador.

Creamos por tanto un método update no que lle pasaremos un parámetro de nome delta e tipo float. (despois veremos para que sirve).
Dito método update vai mover a posición X do pescador en función da velocidade.
A velocidade vai indicar cantas 'unidades por segundo' vai moverse o noso pescador.
Lembrar que o noso mundo mide 332x200.

Por tanto definiremos unha propiedade velocidade inicialmente a 0, co método set asociado.

Arquivo Pescador.java
Creamos o método update.
Definimos a velocidade cos métodos get e set.

```
/**
 * Velocidade do pescador
 */
private float velocidade=0;
.............
/**
 * Modifica a velocidade do pescador
 * @param velocidade: a nova velocidade
 */
public void setVelocidade(float velocidade) {
        this.velocidade = velocidade;
}
/**
 * Devolve a velocidade do pescador
 * @return velocidade
 */
public float getVelocidade(){
        return velocidade;
}
/**
 * Actualiza a posición do pescador en función da velocidade
 * @param delta: tempo entre unha chamada e a seguinte
 */
public void update(float delta){
        posicion.x +=velocidade*delta;
}
.................
```

Aquí entra en xogo outro concepto: **o tempo delta.**
Delta é o tempo que pasa entre unha chamada ó método e a seguinte chamada.

Nota: dende calquera sitio podemos facer uso do seguinte método para obter delta:

```
float delta = Gdx.graphics.getDeltaTime();
```

E para que sirve dito valor ?

Imaxinemos que temos dous dispositivos Android, un funcionado a 60fps (fps: fotogramas por segundo) e outro a 30fps.

Isto quere dicir que vai chamar ó método update ese número de veces por segundo. Se temos a seguinte instrución para mover o noso pescador:

```
mundo.getPescador().getPosicion().x-=1;
```

Como temos no noso xogo un ancho de 332 unidades.

Vai suceder que:

Máquina1:

60fps => móvese 60 unidades por segundo => Tarda 332/60 en percorrer toda a pantalla => 5,5 segundos.

Máquina2:

30fps => móvese 30 unidades por segundo => Tarda 332/30 en percorrer toda a pantalla => 11 segundos.

Polo tanto o pescador moveríase máis rápido na primeira máquina e non parece moi xusto :). Como podemos facer para que o movemento do pescador sexa independente do hardware da máquina ?

Pois multiplicando a velocidade do soldado por delta, sendo delta o tempo que tarda en chamar ó método de actualización da posición.

Por exemplo, seguindo o caso anterior:

Máquina1:
Delta = 1/60
Pescador.x -= delta*1

Canto tarda o soldado en percorrer 332 unidades ?

Nun segundo percorre 1 unidade => 60*1/60*1
É delta
60 son as veces que chama por segundo
Entón en **332 segundos** percorre 332 unidades.

Máquina2:
Delta = 1/30
Pesador.x -=delta*1
Canto tarda o pesador en percorrer 332 unidades ?

Nun segundo percorre 1 unidade => 30*1/30*1

É delta

30 son as veces que chama por segundo

Entón en **332 segundos** percorre 332 unidades.

Como vemos tarda o mesmo.
O que está facendo é axustar diminuíndo a velocidade do pescador en función de delta. Se delta é máis grande (chama moitas veces por segundo) fai que se mova máis lentamente.

O valor 1 é velocidade que ten o pescador. Podemos aumentala para que tarde menos en percorrer toda a pantalla (332 unidades) pero o tempo será igual en calquera dispositivo.
Exercicio:

Imos facer que no noso caso o pescador tarde 4 segundos en percorrer toda a pantalla.
Qué velocidade teríamos que darlle ?

Solución:
Velocidade = espazo / tempo = 332 / 4 = 83

Definimos a velocidade coma unha constante de clase na clase Pescador.
Arquivo Pescador.java
Engadimos unha constante coa velocidade que pode ter o pescador.

```
/**
 * Velocidade que toma o pescador cando se move. Inicialmente 83
 */
public final static float VELOCIDADE_MAX=83;
```

Agora imos chamar ó procedemento update do pescador dende o render da clase PrincipalXogo.

Arquivo PrincipalXogo.java
Definimos unha propiedade delta de tipo float.
Asinamos deltatime a dita propiedade no método render.
Chamamos dende o método render ó método update do pescador.

```
private float delta;
@Override
public void render() {
        // TODO Auto-generated method stub
        Gdx.gl.glClearColor(0, 0, 0, 1);        // Borra a pantalla Red-Green-Blue-Alfa
        Gdx.gl.glClear(GL11.GL_COLOR_BUFFER_BIT);

        spritebatch.begin();

        dibuxarFondo();
        dibuxarPescador();

        spritebatch.end();

        delta = Gdx.graphics.getDeltaTime();
        pescador.update(delta);
}
```

Se executamos agora o xogo o pescador non se move xa que será necesario modificar a velocidade.

Para modificar a velocidade temos que 'capturar' os eventos de entrada.
No noso xogo temos a opción de teclado (para o caso da versión Desktop) e o acelerómetro para a versión Android.

Para ver isto primeiro temos que falar das INTERFACES DE XESTIÓN DE EVENTOS.

O motor libgdx ten dúas interfaces para xestionar todos os eventos:

- InputProcessor: Xestiona os eventos de pulsación sobre a pantalla e arrastre do dedo...
- GestureListener: Xestiona outro tipo de eventos coma son os de dobre pulsación (evento tap), o clásico movemento con dous dedos para facer un zoom da pantalla (evento zoom)....

No noso xogo imos a utilizar só a primeira interface.

Arquivo PrincipalXogo.java
Engadimos a interface e os métodos da mesma.

```
public class PrincipalXogo implements ApplicationListener,InputProcessor{
	@Override
	public boolean keyDown(int keycode) {
		// TODO Auto-generated method stub
		return false;
	}

	@Override
	public boolean keyUp(int keycode) {
		// TODO Auto-generated method stub
		return false;
	}

	@Override
	public boolean keyTyped(char character) {
		// TODO Auto-generated method stub
		return false;
	}

	@Override
	public boolean touchDown(int screenX, int screenY, int pointer, int button) {
		// TODO Auto-generated method stub
		return false;
	}

	@Override
	public boolean touchUp(int screenX, int screenY, int pointer, int button) {
		// TODO Auto-generated method stub
		return false;
	}

	@Override
	public boolean touchDragged(int screenX, int screenY, int pointer) {
		// TODO Auto-generated method stub
		return false;
	}

	@Override
	public boolean mouseMoved(int screenX, int screenY) {
		// TODO Auto-generated method stub
		return false;
```

```
	}

	@Override
	public boolean scrolled(int amount) {
		// TODO Auto-generated method stub
		return false;
	}
```

Podedes consultar o que fai cada un dos métodos en:
http://code.google.com/p/libgdx/wiki/InputEvent

Os da interface GestureListener os podedes atopar en:
http://code.google.com/p/libgdx/wiki/InputGestureDetection

No noso caso imos a capturar a tecla esquerda - dereita e en función de súa pulsación modificamos a velocidade do pescador (Nota: a clase Keys ten a relación de todas as teclas).

Exercicio: realizalo anterior.

Solución:

```
	@Override
	public boolean keyDown(int keycode) {
		// TODO Auto-generated method stub
		if (keycode==Keys.LEFT){
			pescador.setVelocidade(-Pescador.VELOCIDADE_MAX);
		}
		if (keycode==Keys.RIGHT){
			pescador.setVelocidade(Pescador.VELOCIDADE_MAX);
		}
		return false;
	}
```

Nota: o return true / false de cada método indica se o S.O. debe de seguir controlando o evento (mandaría a 'sinal' de tecla pulsada ó resto de aplicacións do S.O.) ou non.

Parace que xa está verdade ?
POIS NON!!!

Se facedes a proba podedes observar coma o pescador non se move.
Isto é debido a que temos que informarlle ó motor de xogos de que a nosa clase (PrincipalXogo) vai ser a que xestione os eventos (ata o de agora só estamos modificando métodos dunha interface que incorporamos).

O facemos coa orde Gdx.input. setInputProcessor(clase que ten a interface InputProcessor).
Cando saímos do xogo teríamos que liberar a xestión de eventos enviándolle null coma clase.

Arquivo PrincipalXogo.java
Asinar a clase que vai a xestionar os eventos no método create e liberala no dispose.

```
	@Override
	public void create() {
		// TODO Auto-generated method stub
		AssetsXogo.cargarTexturas();

		camaraortho = new OrthographicCamera();
		spritebatch = new SpriteBatch();

		mundo = new Mundo();
		pescador = mundo.getPescador();
```

```
        Gdx.input.setInputProcessor(this);
}
@Override
public void pause() {
        // TODO Auto-generated method stub

        Gdx.input.setInputProcessor(this);
}

@Override
public void resume() {
        // TODO Auto-generated method stub

        Gdx.input.setInputProcessor(null);
}

@Override
public void dispose() {
        // TODO Auto-generated method stub
        AssetsXogo.liberarTexturas();
        Gdx.input.setInputProcessor(null);
}
```

Podedes probar agora o xogo e podedes observar que temos un problema. O pescador non para. Temos que capturar o evento de liberar a tecla para poñer a velocidade a 0.

Exercicio: poñer a 0 a velocidade
Solución:

Arquivo PrincipalXogo.java

```
@Override
public boolean keyUp(int keycode) {
        // TODO Auto-generated method stub
        if ((keycode==Keys.LEFT) || (keycode==Keys.RIGHT)){
                pescador.setVelocidade(0);
        }

        return false;
}
```

Agora imos poñerlle uns limites o pescador para cando chegue ó final da pantalla ou o principio. O que temos que facer é controlar a posición, mirar se chega os límites (os temos na clase Mundo) e asinarlle a posición onde se produce o choque (os límites).

Arquivo PrincipalXogo.java
Creamos un método controlarPescador que leve control sobre a posición do pescador para os límites.
Chamamos a dito método dende o método render.

```
@Override
public void render() {
        // TODO Auto-generated method stub
        Gdx.gl.glClearColor(0, 0, 0, 1);        // Borra a pantalla Red-Green-Blue-Alfa
        Gdx.gl.glClear(GL11.GL_COLOR_BUFFER_BIT);

        spritebatch.begin();

        dibuxarFondo();
        dibuxarPescador();

        spritebatch.end();
```

```
            delta = Gdx.graphics.getDeltaTime();
            pescador.update(delta);
            controlarPescador();


    }
    /**
     * Controla o pescador: límites
     */
    private void controlarPescador(){
            if (pescador.getPosicion().x >= Mundo.TAMAÑO_MUNDO.x-Pescador.TAMAÑO.x){
                    pescador.setPosicion(new Vector2(Mundo.TAMAÑO_MUNDO.x-Pescador.TAMAÑO.x,
                                    pescador.getPosicion().y));
            }
            if (pescador.getPosicion().x <= 0){
                    pescador.setPosicion(new Vector2(0,pescador.getPosicion().y));
            }
    }
```

Nota: podemos aumentar o rendemento se en vez de crear un Vector2 xa temos unha propiedade creada temporal (xa instanciada) e así non temos que crear un novo vector cada vez, ou ben crear un novo método en Pescador para asinarlle a posición pero enviándolle dous valores (x,y) de tipo float.

Agora imos controlalo co acelerómetro do dispositivo Android.
Podemos ver como controlar o acelerómetro en libgdx:
http://code.google.com/p/libgdx/wiki/InputAccelerometer

Modificamos o AndroidManifest.xml na versión Android (visto anteriormente):

```
<uses-feature android:required="true" android:name="android.hardware.sensor.accelerometer"/>
```

Modificamos a configuración no método onCreate da MainActivity:

```
cfg.useAccelerometer=true;
```

Nota: como podemos ver no enlace anterior poderíamos comprobar se o dispositivo ten acelerómetro e en caso contrario ofrecer outro tipo de control.

No noso caso imos controlar o acelerómetro do eixe 'y' de tal forma que se o valor do mesmo é maior que 2, asinamos o pescador a velocidade máxima en positivo e se é menor que -2, a velocidade máxima en negativo.
Non nos podemos esquecer de que cando o acelerómetro teña un valor entre -2 e +2 debemos asinar a velocidade 0 ó pescador.
Tamén temos que ter en conta que non debe facer nada en caso de non dispoñer de acelerómetro (por exemplo se executamos a versión desktop).

Exercicio: facer o dito anteriormente.

Arquivo PrincipalXogo.java
Creamos un método novo de nome controlarAcelerómetro que controle o acelerómetro e sexa chamado dende render.

```
    public void render() {
            // TODO Auto-generated method stub
            Gdx.gl.glClearColor(0, 0, 0, 1);        // Borra a pantalla Red-Green-Blue-Alfa
```

```
            Gdx.gl.glClear(GL11.GL_COLOR_BUFFER_BIT);

            spritebatch.begin();

            dibuxarFondo();
            dibuxarPescador();

            spritebatch.end();

            delta = Gdx.graphics.getDeltaTime();
            pescador.update(delta);
            controlarPescador();
            controlarAcelerometro();
    }
    /**
     * Controla o acelerómetro para mover o pescador
     */
    public void controlarAcelerometro(){

            if (!Gdx.input.isPeripheralAvailable(Peripheral.Accelerometer))
                    return;
            int inclinacion = (int) Gdx.input.getAccelerometerY();

            if (inclinacion > 2)
                    pescador.setVelocidade(Pescador.VELOCIDADE_MAX);
            if (inclinacion < -2)
                    pescador.setVelocidade(-Pescador.VELOCIDADE_MAX);
            if ((inclinacion>-2) && (inclinacion<2))
                    pescador.setVelocidade(0);
    }
```

Agora imos facer que cando se pulse a pantalla (ou ben se pulse a tecla ABAIXO) o gancho baixe e cando o solte ou chegue ata o chan, suba ata arriba.

Teríamos varias opcións para facer dita solución. Nos imos a usar as **TEXTURE REGION** que veñen a ser partes dunha texture.
O que imos facer será coller a imaxe do pescador e facer dúas 'texture region', unha que será o pescador e outra o gancho que se moverá.

Todas a información de cada unha destas partes vai ir gardadas nun Texture Atlas.

Un atlas ven a ser un arquivo onde están gardadas as coordenadas (posición e tamaño) para facer referencia a partes dun arquivo de imaxe png no que se atopan todas as texturas necesarias do noso xogo (texturas 2D).

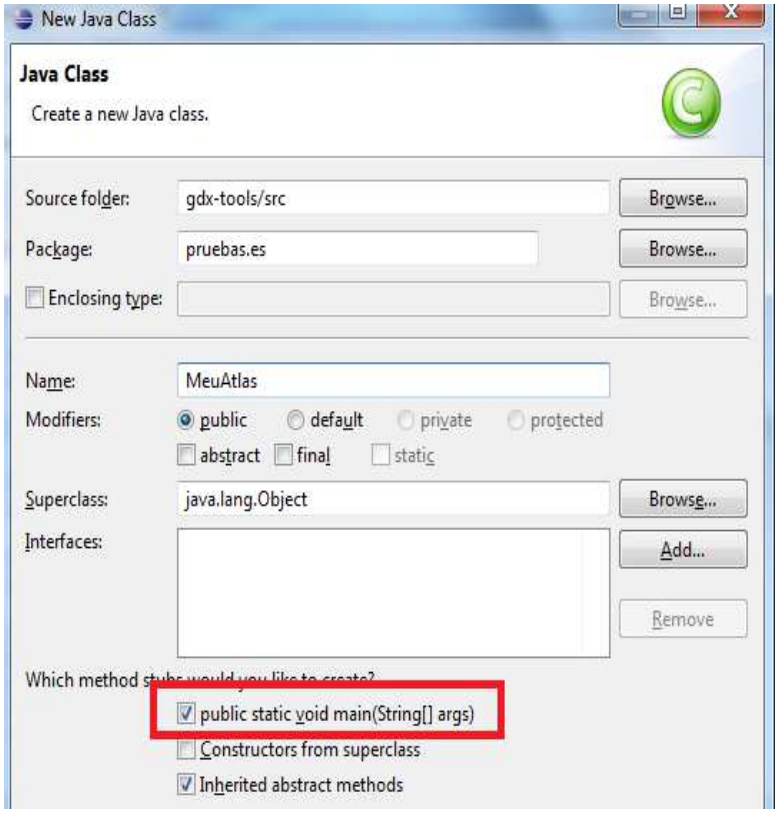
Para crear dito atlas temos que ter nun cartafol todos os arquivos que conforman todas as texturas do noso xogo (se son moitas se poderían facer varias, agrupándoas por categorías ou como queira o programador).

As vantaxes que teñen o uso dun Atlas temos:

- As dimensións dos gráficos xa non teñen que ser potencias de dous aínda que utilicemos o Open gl 1.0. Só ten que ser potencia de dous o arquivo con todos os gráficos.
- Temos un mellor rendemento xa que só temos que acceder unha vez o disco para cargar as texturas.

O proceso:

- Primeiro temos que engadir a eclipse un proxecto que podemos atopar no repositorio do Libgdx de nome Gdx-tools (con este proxecto veñen moitas máis utilidades das descritas aquí). https://github.com/libgdx/libgdx/tree/master/extensions/gdx-tools
  Nota: existen contornos gráficos para facer o mesmo que imos explicar.
- Despois podemos crear un paquete no que crearemos unha clase co método main:



- Dentro do método main poñeremos o seguinte:

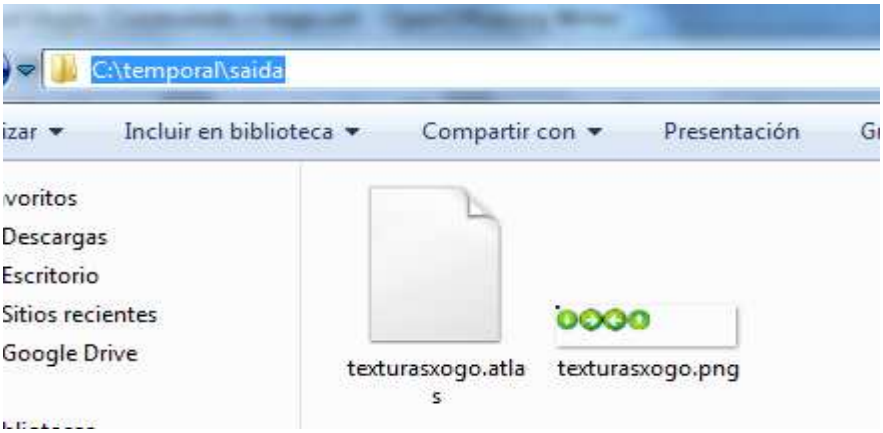
```
TexturePacker2.process("C:\\temporal\\texturas", "C:\\temporal\\saida", "texturasxogo");
```

  Nota: haberá que importar a clase con Ctrl+Shift+O

Sendo o primeiro parámetro o cartafol onde están todas as texturas:

O segundo parámetro é o cartafol de saída, onde se van crear os dous arquivo necesarios.

O terceiro parámetro indica o nome que van levar ou dous arquivos que van crearse cando executemos a clase. No caso de exemplo teremos:



Unha vez temos os arquivos os levamos ó cartafol Assets do proxecto Android.
Para cargar cada unha das texturas, xa dentro do noso proxecto de xogo co motor Libgdx poñeremos:

```
        public static TextureAtlas texturas_todas = new TextureAtlas("texturas/texturasxogo.atlas");
```

// Exemplo de texturas cargadas a partires do exemplo:

```
        public static TextureRegion textura_fondo_negro;
        public static TextureRegion textura_control[] = new TextureRegion[4];

        textura_fondo_negro = texturas_todas.findRegion("fondonegro");
        textura_control[0] = texturas_todas.findRegion("buttonnexticon32");
        textura_control[1] = texturas_todas.findRegion("buttonpreviousicon32");
        textura_control[2] = texturas_todas.findRegion("buttonupicon32");
        textura_control[3] = texturas_todas.findRegion("buttondownicon32");
```

Como vemos, **referenciamos cada textura polo nome físico** que tiña cada unha delas.
Nota: indicar que en vez de usar un obxecto da clase Texture utilizamos un da clase TextureRegion, pero o seu funcionamento é o mesmo.

Se o facemos utilizando a **GUI**:

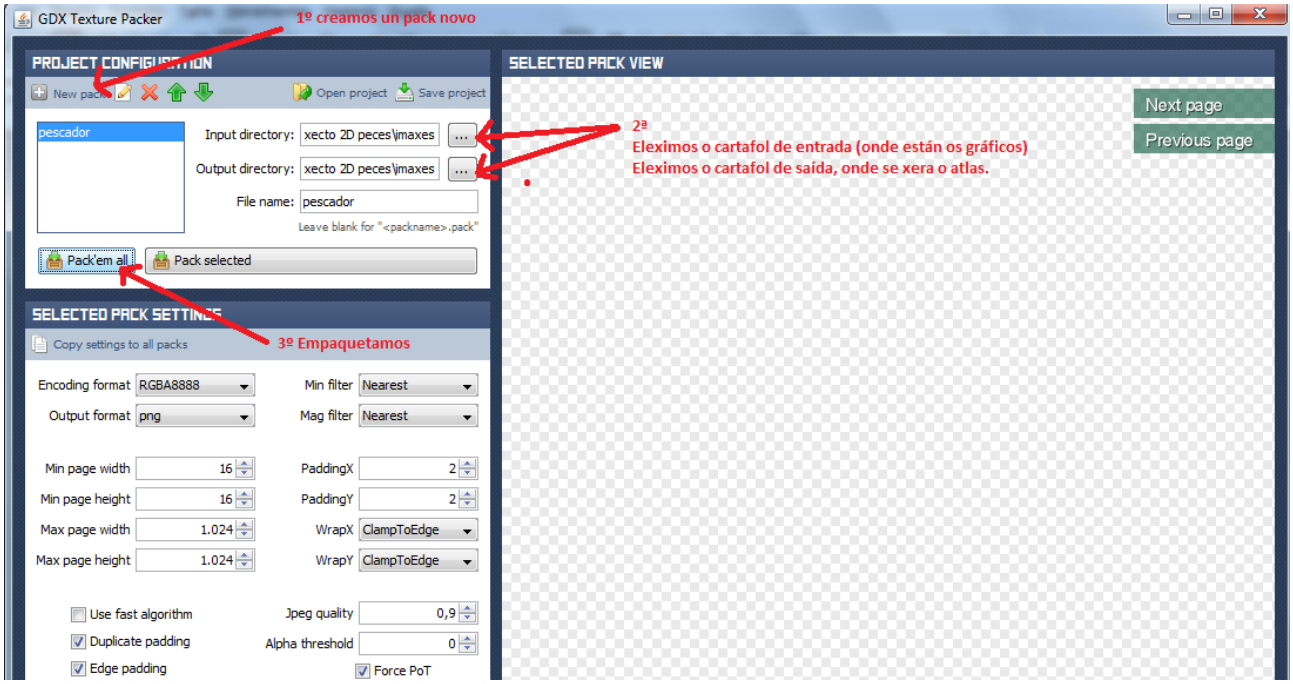
Nota: Antes disto, imos dividir a imaxe do pescador en dúas, o pescador e o gancho. Unha vez feito

as gardamos nun cartafol e executamos a aplicación.

Para crear un Texture Atlas podemos usar programas comerciais como o texture packer: http://www.codeandweb.com/texturepacker

Ou ben usar unha aplicación feita para o framework, a cal se atopa no DVD /Software/Graficos/gdx-texturepacker-3.2.0/iniciar.bat

Podedes consultar como funciona en: http://code.google.com/p/libgdx/wiki/TexturePacker

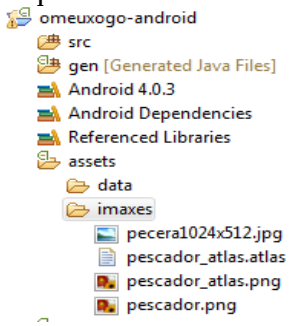


As imaxes orixinais as temos en:

/DVD/Proxectos Eclipse/Proxecto 2D peixes/imaxes/entrada

As imaxes a copiar nun cartafol serán: pescador_sengancho.png - gancho.png

Unha vez xerado a textura e o atlas os debemos gardar no cartafol assets da versión Android e xa poderíamos borrar a imaxe do pescador que tiñamos antes.



Agora temos que modificar a clase que carga os Assets, para indicarlle que ten que cargar o atlas e modificar o pescador para que sexa un TextureRegion:

Arquivo AssetsXogo.java
Se cargar un atlas coas texturas e se modifica a propiedade pescador de Texture => TextureRegion.

```
/**
 * Textura Atlas con todos os gráficos.
 */
static TextureAtlas texturas_todas;

/**
 * Textura do pescador
 */
public static TextureRegion pescador;

public static void cargarTexturas(){

        texturas_todas = new TextureAtlas("imaxes/pescador_atlas.atlas");
        FileHandle imageFileHandle = Gdx.files.internal("imaxes/pecera1024x512.jpg");
        fondopecera = new Texture(imageFileHandle);

        pescador = texturas_todas.findRegion("pescador_sengancho");
}
public static void liberarTexturas(){
        if (fondopecera!=null)
                fondopecera.dispose();
        if (texturas_todas!=null)
                texturas_todas.dispose();
}
```

NOTA: existe unha clase AssetManager que pode empregarse para cargar as texturas de forma asíncrona, de tal forma que a aplicación non quede 'colgada' por tempo de carga moi alto.
O lóxico sería ter unha pantalla específica (cunha barra de progreso) que cargue os recursos gráficos e de son.

Máis información en: http://code.google.com/p/libgdx/wiki/AssetManager

Agora tamén temos que cargar o gancho nunha textura e posicionalo. Como o gancho vai ter unha posición e velocidade diferente ó pescador podemos facer unha clase para gardar dita información. Tamén terá un método update no que se modificará a posición en función da velocidade (só se modificará a posición 'y' xa que a 'x' será igual á da barca) e que será chamado dende o método render da clase PrincipalXogo.

Arquivo AssetsXogo.java
Cargamos a textura do gancho.

```
/**
 * Textura do gancho
 */
public static TextureRegion gancho;

public static void cargarTexturas(){
        texturas_todas = new TextureAtlas("imaxes/pescador_atlas.atlas");

        FileHandle imageFileHandle = Gdx.files.internal("imaxes/pecera1024x512.jpg");
        fondopecera = new Texture(imageFileHandle);

        pescador = texturas_todas.findRegion("pescador_sengancho");
        gancho = texturas_todas.findRegion("gancho");
}
```

Creamos a clase Gancho.

Arquivo Gancho.java
Propiedades tamaño, velocidade, posición.
Métodos get e set.
Método update: modifica a posición y en función da velocidade.

PERO PARADE UN POUCO !!!!!

Resulta que moitos dos métodos e propiedades son os mesmos que no caso do pescador.
Podemos facer unha clase abstracta que incorpore ditas propiedades e métodos e despois darlles os valores concretos para cada clase.

Arquivo Personaxes.java
Propiedades e métodos comúns. Clase Abstracta.
Tamaño deixe de ser static final para ser public xa que se non teríamos que definilo en cada subclase e o facemos public para acceder directamente sen usar get ou set xa que o tamaño non vai variar nunca. O mesmo que a velocidade máxima.
Creamos un método setPosicion(float,float) para non ter que facer new´s cando queiramos modificar a posición.

```
public abstract class Personaxe {

	/**
	 * Instancia unha personaxe
	 * @param posicion
	 * @param tamaño
	 * @param velocidade_max
	 */
	public Personaxe(Vector2 posicion, Vector2 tamaño, float velocidade_max){
		this.posicion=posicion;
		this.tamaño=tamaño;
		this.velocidade_max=velocidade_max;
	}
	/**
	 * Velocidade que toma cando se move.
	 */
	public int velocidade_max;
	/**
	 * Velocidade actual
	 */
	protected int velocidade=0;
	/**
	 * Posición
	 */
	protected Vector2 posicion;
	/**
	 * Tamaño
	 */
	public Vector2 tamaño;


	/**
	 * Devolve a posicion
	 * @return posicion
	 */
	public Vector2 getPosicion() {
		return posicion;
	}
```

```
        /**
         * Modifica a posición
         * @param posicion: a nova posicion
         */
        public void setPosicion(Vector2 posicion) {
                this.posicion = posicion;
        }

        /**
         * Modifica a posición
         * @param x: nova posición x
         * @param y: nova posición y
         */
        public void setPosicion(float x,float y){
                posicion.x = x;
                posicion.y = y;
        }

        /**
         * Modifica a velocidade
         * @param velocidade: a nova velocidade
         */
        public void setVelocidade(float velocidade) {
                this.velocidade = velocidade;
        }
        /**
         * Devolve a velocidade
         * @return velocidade
         */
        public float getVelocidade(){
                return velocidade;
        }

        /**
         * Actualiza a posición en función da velocidade
         * @param delta: tempo entre unha chamada e a seguinte
         */
        abstract void update(float delta);

}
```

Arquivo Gancho.java

Deriva de Personaxes.java

```
public class Gancho extends Personaxe{

        public Gancho(){
                super(new Vector2(0,158),new Vector2(10,10), 30);
        }
        @Override
        public void update(float delta) {
                // TODO Auto-generated method stub
                posicion.y+=velocidade*delta;
        }

}
```

Exercicio: Modificar a clase Pescador para que derive de Personaxes.

Solución:

```
public class Pescador extends Personaxe {

        public Pescador(){
                super(new Vector2(50,128),new Vector2(50,50),83);
        }


        /**
         * Actualiza a posición do pescador en función da velocidade
```

```
         * @param delta: tempo entre unha chamada e a seguinte
         */
        public void update(float delta){
                posicion.x +=velocidade*delta;
        }
}
```

Agora imos debuxar o gancho.
O gancho sempre vai a acompañar o pescador no eixe x e variará a posición 'y' en función da velocidade.

Imos a implementar isto.

Arquivo PrincipalXogo.java
Creamos o método dibuxarGancho que debuxe o gancho en función da súa posición.
Creamos o método actualizarGancho para que modifique a posición en función da posición do pescador e chame ó método update do gancho. Dito método será chamado dende o render.

```
        /**
         * Debuxa o gancho do pescador
         */
        private void dibuxarGancho(){
                spritebatch.draw(AssetsXogo.gancho,gancho.getPosicion().x,gancho.getPosicion().y,
                        gancho.tamaño.x,gancho.tamaño.y);
        }

        @Override
        public void render() {

                ...................
                spritebatch.begin();

                dibuxarFondo();
                dibuxarPescador();
                dibuxarGancho();

                ..............

                spritebatch.end();


                pescador.update(delta);
                actualizarGancho(delta);
                actualizarPeixes(delta);
                ......................
        }

        /**
         * Actualiza a posición en función da posición do pescador
         * Chama a update do gancho
         * @param delta
         */
        private void actualizarGancho(float delta){

                gancho.setPosicion(pescador.getPosicion().x+42,gancho.getPosicion().y);
                gancho.update(delta);

        }
```

O seguinte que temos que solucionar é baixar o gancho do pescador ó premer a pantalla (ou pulsar a tecla abaixo na versión Desktop).

En principio temos que darlle unha velocidade ó gancho ó premer e poñer a velocidade en negativo

cando soltemos. Pero como máis adiante imos ter que saber se o gancho está en estado de 'pescando' para recoller un peixe, imos crear unha propiedade de tipo booleana en Gancho.

Arquivo Gancho.java

Creamos unha propiedade booleana pescando=false e os métodos get e set.

```
private boolean pescando;

/**
 * Devolve true ou false se está pescando
 * @return pescando
 */
public boolean getPescando() {
	return pescando;
}

/**
 * Cambia o estado de pescando
 * @param pescando
 */
public void setPescando(boolean pescando) {
	this.pescando = pescando;
}
```

Arquivo PrincipalXogo.java

O premer sobre a pantalla ou tecla abaixo modifica a velocidade 'y' do gancho á '-velocidade máxima'.

Ó levantar o dedo da pantalla ou deixar de premer a tecla abaixo modifica a velocidade 'y' do gancho á '+velocidade máxima'.

```
@Override
public boolean keyDown(int keycode) {
	// TODO Auto-generated method stub
	if (keycode==Keys.LEFT){
		pescador.setVelocidade(-pescador.velocidade_max);
	}
	if (keycode==Keys.RIGHT){
		pescador.setVelocidade(pescador.velocidade_max);
	}
	if (keycode==Keys.DOWN){
		gancho.setVelocidade(-gancho.velocidade_max);
	}
	return false;
}

@Override
public boolean keyUp(int keycode) {
	// TODO Auto-generated method stub
	if ((keycode==Keys.LEFT) || (keycode==Keys.RIGHT)){
		pescador.setVelocidade(0);
	}
	if (keycode==Keys.DOWN){
		gancho.setVelocidade(+gancho.velocidade_max);
	}

	return false;
}
```

Agora temos que facer que cando o gancho baixe e deixemos de premer a tecla de baixar, este non poida volver baixar ata que chegue arriba. Para isto imos utilizar a variable booleana 'pescando' da clase Gancho.

Imos facer que cando o pescador pulse a pantalla (ou tecla abaixo) cambie o estado de dita variable e que ás veces que prema a pantalla estando nese estado non faga nada.
Tamén teremos que controlar cando o gancho chega arriba de todo para que a velocidade y sexa igual a 0 e cando chegue abaixo de todo para que a velocidade sexa positiva (suba o gancho).

Definimos na clase Gancho o límite superior do gancho cunha constante de clase.

Arquivo Gancho.java
Definimos na clase Gancho o límite superior do gancho cunha constante de clase.

```
	public final static int LIMITE_Y_SUPERIOR = 158;
```

Arquivo PrincipalXogo.java
Cambiamos a velocidade do gancho e a variable pescando en función da posición do gancho.
Definimos un método controlarGancho que controle os límites do gancho para que cambie de sentido (cando chega abaixo de todo) e pare (cando chegue arriba).
Chamamos a dito método dende o render.

```
	/**
	 * Controla os límites do gancho
	 */
	private void controlarGancho(){
		if (gancho.getPosicion().y>Gancho.LIMITE_Y_SUPERIOR){ // Poñer sempre '>' nunca '==' xa que se move con valores float´s
			gancho.setVelocidade(0);
			gancho.setPescando(false);
			gancho.getPosicion().y=Gancho.LIMITE_Y_SUPERIOR;
		}
		else {
			if (gancho.getPosicion().y<=0){  // Limite inferior
				gancho.setVelocidade(+gancho.velocidade_max);
			}
		}
	}

	@Override
	public void render() {
		// TODO Auto-generated method stub
//		Gdx.gl.glClearColor(0, 0, 0, 1);		// Borra a pantalla Red-Green-Blue-Alfa
//		Gdx.gl.glClear(GL11.GL_COLOR_BUFFER_BIT);	// Xa non é necesario xa que borra co fondo.

		spritebatch.begin();

		dibuxarFondo();
		dibuxarPescador();
		dibuxarGancho();

		spritebatch.end();

		delta = Gdx.graphics.getDeltaTime();
		pescador.update(delta);
		gancho.update(delta);
		controlarPescador();
		controlarAcelerometro();
		controlarGancho();
	}
	@Override
	public boolean keyDown(int keycode) {
		// TODO Auto-generated method stub
		if (keycode==Keys.LEFT){
			pescador.setVelocidade(-pescador.velocidade_max);
		}
```

```
		if (keycode==Keys.RIGHT){
			pescador.setVelocidade(pescador.velocidade_max);
		}
		if (keycode==Keys.DOWN){
			if (!gancho.getPescando()){
				gancho.setVelocidade(-gancho.velocidade_max);
				gancho.setPescando(true);
			}
		}
		return false;
	}

	@Override
	public boolean keyUp(int keycode) {
		// TODO Auto-generated method stub
		if ((keycode==Keys.LEFT) || (keycode==Keys.RIGHT)){
			pescador.setVelocidade(0);
		}
		if (keycode==Keys.DOWN){
			gancho.setVelocidade(+gancho.velocidade_max);
		}

		return false;
	}
	@Override
	public boolean touchDown(int screenX, int screenY, int pointer, int button) {
		// TODO Auto-generated method stub
		if (!gancho.getPescando()){
			gancho.setVelocidade(-gancho.velocidade_max);
			gancho.setPescando(true);
		}
		return false;
	}

	@Override
	public boolean touchUp(int screenX, int screenY, int pointer, int button) {
		// TODO Auto-generated method stub
		gancho.setVelocidade(+gancho.velocidade_max);
		return false;
	}
```

Agora imos facer que o sedal da caña acompañe ó gancho. Isto non ten moita complicación xa que o único que temos que facer é crear un textura que sexa un punto negro e que se debuxe dende onde está o pescador ata onde está o gancho.

Creamos polo tanto a textura de nome punto.png (se atopa en /DVD/Proxectos Eclipse/Proxecto 2D peixes/imaxes/entrada) , xeramos novamente o atlas e cargamos a textura na clase AssetsXogo.

Arquivo AssetsXogo.java

Cargamos a textura punto do atlas.

```
	/**
	 * Textura da corda da caña
	 */
	public static TextureRegion sedal;

	public static void cargarTexturas(){
		texturas_todas = new TextureAtlas("imaxes/pescador_atlas.atlas");

		FileHandle imageFileHandle = Gdx.files.internal("imaxes/pecera1024x512.jpg");
		fondopecera = new Texture(imageFileHandle);

		pescador = texturas_todas.findRegion("pescador_sengancho");
		gancho = texturas_todas.findRegion("gancho");
		sedal = texturas_todas.findRegion("punto");
	}
```

Arquivo PrincipalXogo.java

Creamos un método de nome dibuxarXedal que debuxe o punto dende o pescador ata o gancho.

Nota: cando debuxamos co spritebatch, se poñemos un width e/ou height negativo ten o efecto de debuxar o inverso.

```
        /**
         * Dibuxa o sedal
         */
        private void dibuxarSedal(){
                spritebatch.draw(AssetsXogo.sedal,pescador.getPosicion().x+41+
(gancho.tamaño.x/2),Gancho.LIMITE_Y_SUPERIOR+gancho.tamaño.y,0.75f,-(Gancho.LIMITE_Y_SUPERIOR-
gancho.getPosicion().y));
        }
        @Override
        public void render() {

                spritebatch.begin();

                dibuxarFondo();
                dibuxarPescador();
                dibuxarGancho();
                dibuxarSedal();

                spritebatch.end();

                delta = Gdx.graphics.getDeltaTime();
                pescador.update(delta);
                gancho.update(delta);
                controlarPescador();
                controlarAcelerometro();
                controlarGancho();
        }
```

Os peixes. ANIMACIÓNS.

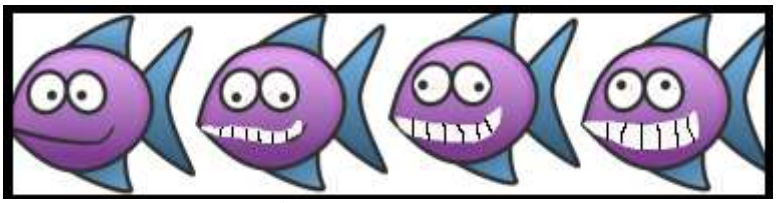
Agora imos afrontar o problema dos peixes.
Igual que antes imos manexar texturas, pero agora imos facer que o peixe teña unha animación.

Podedes consultar como se fan animacións en
http://code.google.com/p/libgdx/wiki/SpriteAnimation

O idea é moi sinxela. Consiste en debuxar de forma continuada un conxunto de TextureRegion que conforman a animación.

O proceso é o seguinte:

- Cargamos nun obxecto Texture ou TextureRegion as imaxes que conforman a nosa animación:



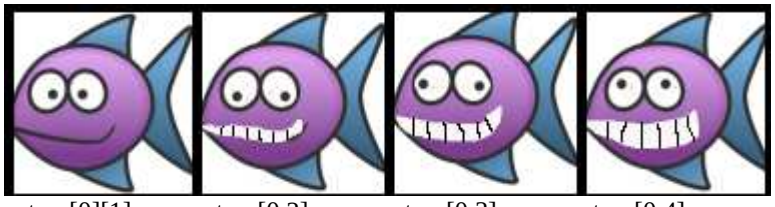
Nome fish1_animacion no texture atlas

No noso caso imos a engadir estas imaxes no atlas xerado anteriormente.
Temos a imaxe no DVD (/DVD/Proxectos Eclipse/Proxecto 2D peixes/imaxes/entrada).
Unha vez cargado novo atlas teremos unha TextureRegion que fará referencia a ditas imaxes:

```
TextureRegion trPeixe = texturas_todas.findRegion("fish1_animacion");
```

Agora imos dividir dita imaxe en anacos máis pequenos. Cada peixe ten un tamaño de 96x96 polo tanto imos dividir dita imaxe en 4 anacos. Isto o facemos có método split que devolve un array bidimensional de TextureRegion, no noso caso devolverá un array de 1 fila e 4 columnas:

```
TextureRegion[][] tmp = trPeixe.split(96,96);
```



tmp[0][1] tmp[0,2] tmp[0,3] tmp[0,4]

Nota: poderíamos ter máis imaxes (máis filas) e o método split ó chegar ó final pasaría a seguinte fila deixando unha altura de 96 píxeles (o indicado no método).

A continuación temos que crear unha texture region dunha dimensión que sexa igual ó número de frames a amosar. No noso caso son 1x4 = 4.

Podemos calcula o número de filas e columnas así:

NUM_FILAS = HEIGHT_TEXTUREGION_ fish1_animacion / 96 = 96 / 96 = 1
NUM_COLUMNA = WIDTH_TEXTUREGION_ fish1_animacion / 96 = 384 / 96 = 4

```
int num_filas = trPeixe.getRegionHeight() / 96;
int num_columnas = trPeixe.getRegionWidth() / 96;

TextureRegion[] framesanimacion = new TextureRegion[num_filas*num_columnas];
```

O seguinte será traspasar as texturas gardadas no array bidimensional:

```
int index = 0;
for (int i = 0; i < num_filas; i++) {
        for (int j = 0; j < num_columnas; j++) {
             framesanimacion[index++] = tmp[i][j];
        }
}
```

E xa como paso final teremos que crear un obxecto da clase Animation no que lle pasamos o array de TextureRegion e o tempo que debe pasar ata que se cambie de frame:

```
peixe1 = new Animation(0.15f,framesanimacion);
```

O código completo:

Arquivo AssetsXogo.java
Creando unha animación.

```
/**
 * Animación do peixe
 */
public static Animation peixe_lila;

/**
 * Carga as animacions dos peixes
 */
private static void cargandoAnimacions(){

        TextureRegion trPeixe = texturas_todas.findRegion("fish1_animacion");
        TextureRegion[][] tmp = trPeixe.split(96,96);
        int num_filas = trPeixe.getRegionHeight() / 96;
        int num_columnas = trPeixe.getRegionWidth() / 96;

        TextureRegion[] framesanimacion = new TextureRegion[num_filas*num_columnas];
int index = 0;
for (int i = 0; i < num_filas; i++) {
        for (int j = 0; j < num_columnas; j++) {
        framesanimacion[index++] = tmp[i][j];
        }
}

        peixe_lila = new Animation(0.15f,framesanimacion);

}

public static void cargarTexturas(){
        texturas_todas = new TextureAtlas("imaxes/pescador_atlas.atlas");

        FileHandle imageFileHandle = Gdx.files.internal("imaxes/pecera1024x512.jpg");
        fondopecera = new Texture(imageFileHandle);

        pescador = texturas_todas.findRegion("pescador_sengancho");
```

```
                gancho = texturas_todas.findRegion("gancho");
                sedal = texturas_todas.findRegion("punto");

                cargandoAnimacions();
        }
```

A segunda parte é crear unha clase peixe que derive de Personaxes. Ademais creamos unha propiedade 'tipopeixe' que vai indicar o tipo de peixe (hai 3 tipos).

Arquivo Peixe.java
Definimos propiedade tipopeixe que vai indicar o tipo de peixe (hai 3 tipos).
Definimos posición, tamaño e velocidade.

```
public class Peixe extends Personaxe{

        /**
         * Indica o tipo de peixe para dibuxalo correctamente.
         */
        public enum Tipos_peixe {
                AZUL,LILA,AMARELO
        }

        private Tipos_peixe tipopeixe;
        public Peixe() {
                super(new Vector2(Mundo.TAMAÑO_MUNDO.x,90), new Vector2(20,20), 40);
                tipopeixe = Tipos_peixe.LILA;
                velocidade = velocidade_max;
                // TODO Auto-generated constructor stub
        }
        /**
         * Devolve o tipo de peixe
         * @return tipopeixe
         */
        public Tipos_peixe getTipopeixe() {
                return tipopeixe;
        }

        @Override
        public void update(float delta) {
                // TODO Auto-generated method stub
                posicion.x -= velocidade*delta;
        }

}
```

Igual que fixemos co pescador imos engadir o peixe ó noso mundo e crear o método get.

Arquivo Mundo.java
Engadimos un peixe ó noso mundo.

```
        /**
         * Os peixes do mundo
         */
        private Peixe peixe;

        public Mundo(){
                pescador = new Pescador();
                gancho = new Gancho();
                peixe = new Peixe();
        }
        /**
         * Devolve os peixes
         * @return peixe
         */
        public Peixe getPeixe() {
```

```
        return peixe;
}
```

Agora temos que debuxalo.
Para facelo temos que facer uso dunha propiedade que leve o tempo para que cada 0.15 segundos (o indicado cando creamos a animación) cambie de frame.
Dita variable só é un acumulador de delta.

Arquivo PrincipalXogo.java
Creamos un cronómetro.

```
private float stateTime;// Leva o tempo transcurrido para cambiar de frame.

public void create() {
        // TODO Auto-generated method stub
        stateTime=0;
        ……….

@Override
public void render() {

        delta = Gdx.graphics.getDeltaTime();
        stateTime += delta;

        spritebatch.begin();

        dibuxarFondo();
        dibuxarPescador();
        dibuxarGancho();
        dibuxarSedal();
```

Agora só nos queda informa ó spritebatch que debuxe o frame adecuado en función do tempo transcorrido.

Como peixe1 é un obxecto da clase Animation, ten un método (`getKeyFrame`) que devolve o frame actual. O segundo parámetro do método indica se queremos que se repitan os frames ou pare a animación ó chegar ó final.

Arquivo PrincipalXogo.java
Definimos unha propiedade da clase Peixe que será o peixe do noso mundo.
Creamos unha propiedade da clase TextureRegion que vai gardar o frame actual
Creamos un método dibuxarPeixes(delta) que será chamado dende o render e dibuxará o frame actual.

```
private Peixe peixe;
private TextureRegion currentFrame;     // Frame actual do peixe a dibuxar

public void create() {

        ..............
        pescador = mundo.getPescador();
        gancho = mundo.getGancho();
        peixe = mundo.getPeixe();
        ............

/**
 * Dibuxa os peixes
 */
private void dibuxarPeixes(float delta){
        if (peixe.getTipopeixe()==Peixe.Tipos_peixe.LILA){
                currentFrame = AssetsXogo.peixe_lila.getKeyFrame(stateTime, true);
```

```
                spritebatch.draw(currentFrame,peixe.getPosicion().x,peixe.getPosicion().y,
                                peixe.tamaño.x,peixe.tamaño.y);
        }
}

@Override
public void render() {
        ..............
        spritebatch.begin();

        dibuxarFondo();
        dibuxarPescador();
        dibuxarGancho();
        dibuxarSedal();
        dibuxarPeixes(delta);

        spritebatch.end();
        dibuxarFondo();
        dibuxarPescador();
        dibuxarGancho();
        dibuxarSedal();
        dibuxarPeixes(delta);

        spritebatch.end();


        pescador.update(delta);
        gancho.update(delta);
        peixe.update(delta);

        controlarPescador();
        controlarAcelerometro();
        controlarGancho();
}
```

Exercicio: Intentade engadir un peixe animado de cor amarelo coa súa propia animación ó voso mundo.

ESPERADE!!!!!!!!!!!!!!!!!!!

Isto non ten senso. No noso mundo imos ter moitos peixes. Que imos facer o mesmo con cada un deles....?

NON!

O que temos que facer é usar un array de peixes e percorrer cada peixe do array debuxándoo.

Entón:

- Engadimos ó Atlas as diferentes animacións e texturas e os cargamos na clase AssetsXogo. Temos 3 tipos de peixes no DVD /DVD/Proxectos Eclipse/Proxecto 2D peixes/imaxes/entrada:

  fish1_animacion.png
  fish2_animacion.png
  fish3.png

  Nota: Un dos peixes non ten animación.

```
/**
 * Animación do peixe amarelo
 */
public static Animation peixe_amarelo;

/**
 * Textura do peixe azul
 */
public static TextureRegion peixe_azul;

private static void cargandoAnimacions(){

        TextureRegion trPeixe = texturas_todas.findRegion("fish1_animacion");
        TextureRegion[][] tmp = trPeixe.split(96,96);
        int num_filas = trPeixe.getRegionHeight() / 96;
        int num_columnas = trPeixe.getRegionWidth() / 96;

        TextureRegion[] framesanimacion = new TextureRegion[num_filas*num_columnas];
int index = 0;
for (int i = 0; i < num_filas; i++) {
        for (int j = 0; j < num_columnas; j++) {
        framesanimacion[index++] = tmp[i][j];
        }
}

        peixe_lila = new Animation(0.15f,framesanimacion);

        trPeixe = texturas_todas.findRegion("fish2_animacion");
        tmp = trPeixe.split(96,96);
        num_filas = trPeixe.getRegionHeight() / 96;
        num_columnas = trPeixe.getRegionWidth() / 96;

        framesanimacion = new TextureRegion[num_filas*num_columnas];
index = 0;
for (int i = 0; i < num_filas; i++) {
        for (int j = 0; j < num_columnas; j++) {
        framesanimacion[index++] = tmp[i][j];
        }
}

        peixe_amarelo = new Animation(0.15f,framesanimacion);

}
public static void cargarTexturas(){
        texturas_todas = new TextureAtlas("imaxes/pescador_atlas.atlas");

        FileHandle imageFileHandle = Gdx.files.internal("imaxes/pecera1024x512.jpg");
        fondopecera = new Texture(imageFileHandle);

        pescador = texturas_todas.findRegion("pescador_sengancho");
        gancho = texturas_todas.findRegion("gancho");
        sedal = texturas_todas.findRegion("punto");

        peixe_azul = texturas_todas.findRegion("fish3");

        cargandoAnimacions();
}
```

- Agora o noso mundo vai estar formado por un array de peixes.

Arquivo Mundo.java

Pasamos de:

```
        private Peixe peixe;
```

A:

```
        private Array<Peixe> peixes;

Modificamos constructor:
        public Mundo(){
                pescador = new Pescador();
                gancho = new Gancho();
                peixes = new Array<Peixe>();
        }

Pasamos de:
        public Peixe getPeixe() {
                return peixe;
        }
A:
        public Array<Peixe> getPeixes() {
                return peixes;
        }
```

Agora na clase Mundo imos crear un método que engada un novo peixe ó noso array e imos modificar a clase Peixe para que no constructor cre un peixe de forma aleatoria nos seus parámetros de velocidade, tipo , tamaño e posición na coordenada y.

Para xerar eses números aleatorios imos a definir un método na clase Mundo de tipo public static que devolva un número aleatorio entre dous números:

Arquivo Mundo.java
Creamos método que devolva un número aleatorio.

```
        /**
         * Xera un nº aleatorio entre min e max
         * @param min
         * @param max
         * @return o número aleatorio
         */
        public static int xerarAleatorio(int min,int max) {
                Random aleat = new Random();
                return (aleat.nextInt(max-min)+min);
        }
```

Arquivo Peixe.java
Xera de forma aleatoria un peixe.

```
        public Peixe() {
                super(new Vector2(Mundo.TAMAÑO_MUNDO.x,Mundo.xerarAleatorio(1,100)),
                      new Vector2(Mundo.xerarAleatorio(8,30),0), Mundo.xerarAleatorio(30, 60));
                tamaño.y = tamaño.x;

                int tipo = Mundo.xerarAleatorio(0, 3);
                switch(tipo){
                        case 0:
                                tipopeixe = Tipos_peixe.LILA;
                                break;
                        case 1:
                                tipopeixe = Tipos_peixe.AMARELO;
                                break;
                        case 2:
                                tipopeixe = Tipos_peixe.AZUL;
                                break;

                }
                velocidade = velocidade_max;
        }
```

Arquivo Mundo.java

Creamos o método crearPeixe que engada un peixe ó array de peixes.

```
/**
 * Engade un peixe ó array de Peixes
 */
public void engadirPeixe(){
        peixes.add(new Peixe());
}
```

Modificamos a clase PrincipalXogo xa que agora temos que recoller do noso mundo un array de peixes. Percorrer o array e actualizar e debuxar cada un deles.

Arquivo PrincipalXogo.java

Pasamos de:

```
private Peixe peixe;
```

A:

```
private Array<Peixe> peixes;
```

Pasamos de:

```
public void create() {
        ................
        peixe = mundo.getPeixe();
        ..............
}
```

A:

```
        peixes = mundo.getPeixes();
```

Exercicio:

Modifica o método dibuxarPeixes para que debuxe o peixe en función do tipo.
Para percorrer un array podemos usar:

- a clase Iterator
  https://code.google.com/p/libgdx/wiki/SimpleApp (buscar por Iterator)

```
Iterator <Peixe> iter = mundo.getPeixes().iterator();
while(iter.hasNext()){
        Peixe peixe = iter.next();
}
```

- un for da forma:

```
for (Peixe peixe : mundo.getPeixes()){

}
```

Solución:

```
/**
 * Dibuxa os peixes
 */
private void dibuxarPeixes(float delta){

	for (Peixe peixe : mundo.getPeixes()){
		if (peixe.getTipopeixe()==Peixe.Tipos_peixe.LILA){
			currentFrame = AssetsXogo.peixe_lila.getKeyFrame(stateTime, true);

			spritebatch.draw(currentFrame,peixe.getPosicion().x,peixe.getPosicion().y,
					peixe.tamaño.x,peixe.tamaño.y);
		}
		else if (peixe.getTipopeixe()==Peixe.Tipos_peixe.AMARELO){
			currentFrame = AssetsXogo.peixe_amarelo.getKeyFrame(stateTime, true);

			spritebatch.draw(currentFrame,peixe.getPosicion().x,peixe.getPosicion().y,
					peixe.tamaño.x,peixe.tamaño.y);
		}
		else if (peixe.getTipopeixe()==Peixe.Tipos_peixe.AZUL){

			spritebatch.draw(AssetsXogo.peixe_azul,peixe.getPosicion().x,
				peixe.getPosicion().y, peixe.tamaño.x,peixe.tamaño.y);
		}
	}
}
```

- Creamos un método actualizarPeixes(delta) para que chame ó método update de todos os peixes do array.

Pasamos de:

```
@Override
public void render() {
.....................
peixe.update(delta);
...........
```

A:

```
@Override
public void render() {
.....................
actualizarPeixes(delta);
...........
```

Creamos o método:

```
/**
 * Actualiza a posición dos peixes
 */
private void actualizarPeixes(float delta){
	for(Peixe peixe: mundo.getPeixes()){
		peixe.update(delta);
	}
}
```

Se executamos agora o xogo vemos que non aparecen peixes. Isto é debido a que non estamos a engadir ningún peixe o array.

Imos facer que cada certo tempo se xere un peixe.
Agora mesmo temos un contador (stateTime) que sempre se está incrementando.
Podemos xerar un número aleatorio entre 1-5, de tal forma que cando stateTime chege a stateTime+nº aleatorio, engada un novo peixe.

Exercicio:
Intentade facelo. A propiedade que leva o tempo dádelle de nome `tempoParaCrearPeixe.`

Solución
Temos que modificar a clase PrincipalXogo e engadir unha propiedade nova que gardará o tempo no que ten que crearse o peixe (o tempo actual máis o número aleatorio).

Arquivo PrincipalXogo.java
Engadimos o método crearPeixes.

```
public void render() {
        ..................
        pescador.update(delta);
        gancho.update(delta);
        actualizarPeixes(delta);
        crearPeixes();
        ..............
/**
 * Crea un peixe de forma aleatoria no tempo
 */
private void crearPeixes(){
        if (tempoParaCrearPeixe <= stateTime){
                mundo.engadirPeixe();
                tempoParaCrearPeixe = stateTime+Mundo.xerarAleatorio(1, 5);
        }

}
```

Se probamos agora xa podemos ver como aparecen os peixes.

NOTA IMPORTANTE: Agora mesmo os peixes só desaparecen da pantalla pero seguen existindo, o que sucede é que están fora do que vemos.

Temos por tanto que eliminar os peixes que cheguen ó final. Para forrar recursos ó ideal sería non

eliminalos, senón reubicalos e que comezaran dende a dereita. Desta forma nos aforraríamos novos new de peixes e o garbage collection non tería que facer nada.

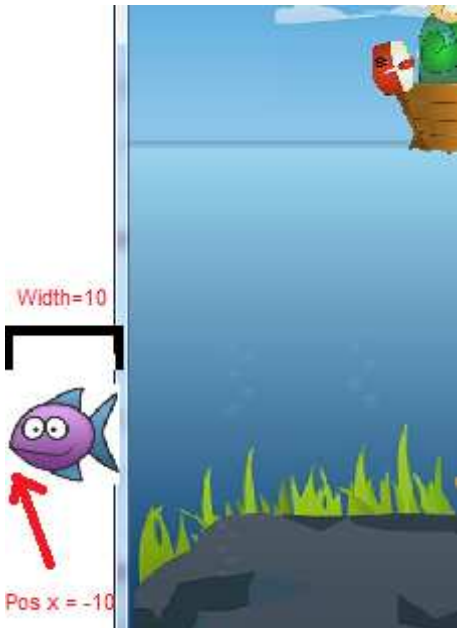
NOTA: sempre que se poida é mellor facer un 'new' no create a modificalo no render (os seus valores) que estar facendo new´s dentro do render. Lembrade que o render chámase 30-65 veces por segundo.

Como temos feito o noso xogo (xeramos novos peixes aleatoriamente e o número nunca vai ser moi alto) imos optar por eliminalos do array cando cheguen ó final.

Arquivo PrincipalXogo.java
Creamos o método controlarPeixe chamado dende render que elimine os peixes do array.

```
	/**
	 * Elimina o peixe do array ó chegar ó final
	 */
	private void controlarPeixes(){
		for(Peixe peixe: mundo.getPeixes()){
			if (peixe.getPosicion().x<=-peixe.tamaño.x){
				mundo.getPeixes().removeValue(peixe, true);
			}
		}
	}
```

Fixarse coma a posición dos peixes no eixe x é o negativo do seu ancho.



Agora imos facer unha mellora, que vai ser que cando o pescador se poña a pescar (o gancho estea baixando ou subindo) non se poda mover.

Exercicio: intentádeo facelo. Lembrade a propiedade boolena do gancho.
Solución:

Arquivo PrincipalXogo.java

```
@Override
public boolean keyDown(int keycode) {
	// TODO Auto-generated method stub
	if ((keycode==Keys.LEFT) && (!gancho.getPescando())){
		pescador.setVelocidade(-pescador.velocidade_max);
	}
	if ((keycode==Keys.RIGHT) && (!gancho.getPescando())){
		pescador.setVelocidade(pescador.velocidade_max);
	}
	if (keycode==Keys.DOWN){
		if (!gancho.getPescando()){
			gancho.setVelocidade(-gancho.velocidade_max);
			gancho.setPescando(true);
		}
	}
	return false;
}
public void controlarAcelerometro(){

	if (!Gdx.input.isPeripheralAvailable(Peripheral.Accelerometer))
			return;
	if (gancho.getPescando()) return;

	int inclinacion = (int) Gdx.input.getAccelerometerY();

	if (inclinacion > 2)
		pescador.setVelocidade(pescador.velocidade_max);
	if (inclinacion < -2)
		pescador.setVelocidade(-pescador.velocidade_max);
	if ((inclinacion>-2) && (inclinacion<2))
		pescador.setVelocidade(0);
}
```

Outra mellora que podemos ter é limitar o número de peixes á vez que poden estar na pantalla. Creamos unha constante de clase na clase Mundo que sexa o número máximo de peixes e modificamos o procedemento crearPeixes para que non cre ningún máis se acadamos o máximo valor.

Arquivo Mundo.java

Creamos constante de clase.

```
/**
 * Indica o número de peixes que poden estar á vez na pantalla
 */
public final static int NUM_PEIXES_MAXIMO=4;
```

Arquivo PrincipalXogo.java

Modificamos o método crearPeixes.

```
private void crearPeixes(){
	if (mundo.getPeixes().size==Mundo.NUM_PEIXES_MAXIMO) return;

	if (tempoParaCrearPeixe <= stateTime){
		mundo.engadirPeixe();
		tempoParaCrearPeixe = stateTime+Mundo.xerarAleatorio(1, 5);
	}
}
```

Ben, parece que o xogo avanza, pero antes de dicirvos que todo isto está mal feito (me refiro á forma) imos a implantar a parte de pescar os peixes.

A idea é que cando o gancho se atope cun peixe, este suba xunto co gancho e na parte superior esquerda aparezan os peixes recollidos.

O primeiro é saber cando o gancho se atopa co peixe.

DETECCIÓN DE COLISIÓNS.

Para detectar o choque imos utilizar a clase **Intersector**.
Dita clase ten moitos métodos para detectar cando dúas formas interseccionan.
Como o peixe e o gancho teñen unha forma rectangular-cadrada, nos usaremos:

```
Intersector.overlapRectangles(r1, r2)
```

Dito método espera recibir dous rectángulos coma parámetros. A clase Rectángulo ten catro datos: pos_x,pos_y,tam_x,tam_y.

Imos modificar a clase Personaxes para engadir unha propiedade rectángulo co seu método get e un método actualizarRectangulo que modifique a posición do mesmo en función da posición actual e modificamos o método update para que chame a dito método.

Arquivo Personaxe.java
Creamos propiedade da clase Rectangle.
Métodos get e actualizarRectangulo.

```
/**
 * Rectángulo para os choques co gancho
 */
protected Rectangle rectangulo;

public Personaxe(Vector2 posicion, Vector2 tamaño, int velocidade_max){
        this.posicion=posicion;
        this.tamaño=tamaño;
        this.velocidade_max=velocidade_max;
        rectangulo = new Rectangle(posicion.x, posicion.y, tamaño.x, tamaño.y);
}
/**
 * Devolve o rectángulo asociado
 * @return rectangulo
 */
public Rectangle getRectangulo() {
        return rectangulo;
}

/**
 * Modifica a posición do rectángulo coa posición actual
 * Non necesitamos modificar o tamaño xa que non cambia
 */
public void actualizarRectangulo() {
        this.rectangulo.x = posicion.x;
        this.rectangulo.y = posicion.y;

}
```

Agora temos que modificar os métodos update de Gancho e Peixe para que chamen ó método actualizarRectangulo.

Arquivo Peixe.java

Modificamos o método update.

```
@Override
public void update(float delta) {
	// TODO Auto-generated method stub
	posicion.x -= velocidade*delta;
	actualizarRectangulo();
}
```

Arquivo Gancho.java

Modificamos o método update.

```
@Override
public void update(float delta) {
	// TODO Auto-generated method stub
	posicion.y+=velocidade*delta;
	actualizarRectangulo();
}
```

Como o debuxo do gancho inclúe a bola de flotación e só queremos que teña en conta o gancho, podemos modificar o tamaño 'height' do rectángulo para que o seu alto sexa a mitade:

```
public Gancho(){
	super(new Vector2(0,158),new Vector2(10,10), 30);
	rectangulo.setHeight(tamaño.x/2);	// A altura do gancho é a mitade, xa que engloba a bola amarela
```

Xa temos os rectángulo só queda controlar cando interseccionan...Isto o imos facer no método controlarGancho. No momento en que sabemos que o peixe foi collido polo gancho imos a modificar o estado dunha propiedade nova na clase Peixe que vainos a servir para facer que o peixe teña a mesma posición que o gancho ata que chegue arriba de todo.

Arquivo Peixe.java

Creamos a propiedade boolean pescado cos método set e get. Se no método set enviamos o valor true imos poñer a 0 a súa velocidade xa que non ten que moverse (empeza a subir co gancho).

```
/**
 * Indica se foi pescado polo gancho
 */
private boolean pescado;
/**
 * Indica se foi pescado o peixe
 */

public boolean getPescado() {
	return pescado;
}
/**
 * Cambia o estado do peixe a pescado (igual a true) e segue o gancho
 * @param pescado
 */
public void setPescado(boolean pescado) {
	if (pescado) velocidade = 0;
	this.pescado = pescado;
}
```

Arquivo PrincipalXogos.java

Modificamos o método controlarGancho para que controle cando o peixe se atopa có gancho.
Modificamos o método controlarPeixes para que os peixes que estean pescados teñan a mesma posición que o gancho (está subindo) e sexan eliminados cando cheguen arriba de todo.

```
	private void controlarGancho(){
		if (gancho.getPosicion().y>Gancho.LIMITE_Y_SUPERIOR){ // Poñer sempre '>' nunca '==' xa que se move con valores float´s
			gancho.setVelocidade(0);
			gancho.setPescando(false);
			gancho.getPosicion().y=Gancho.LIMITE_Y_SUPERIOR;
		}
		else {
			if (gancho.getPosicion().y<=0){  // Limite inferior
				gancho.setVelocidade(+gancho.velocidade_max);
			}
		}

		// Comprobamos se o gancho colle a un peixe
		for (Peixe peixe : mundo.getPeixes()){
			if (Intersector.overlapRectangles(peixe.getRectangulo(), gancho.getRectangulo())){
				peixe.setPescado(true);
			}
		}
	}

	private void controlarPeixes(){
		for(Peixe peixe: mundo.getPeixes()){
			if (peixe.getPescado()) {	// Está pescado polo gancho
				peixe.setPosicion(gancho.getPosicion().x,gancho.getPosicion().y);
			}

			if (peixe.getPosicion().y==Gancho.LIMITE_Y_SUPERIOR){   // Chegou arriba
				mundo.getPeixes().removeValue(peixe, true);
			}

			if (peixe.getPosicion().x<=-peixe.tamaño.x){    // Chega ó final da pantalla
				mundo.getPeixes().removeValue(peixe, true);
			}
		}

	}
```

NOTA:

Cando usamos intersecciónss pode ser necesario saber onde quedan os rectángulos que conforman os nosos gráficos.
Podemos facer uso da clase ShapeRenderer que permite debuxar figuras xeométricas.

Por exemplo:

```
		shaperenderer.begin(ShapeType.Filled);
		shaperenderer.setColor(Color.RED);
		for (Peixe peixe : mundo.getPeixes()){

			shaperenderer.rect(peixe.getRectangulo().x,peixe.getRectangulo().y,
					peixe.getRectangulo().width,peixe.getRectangulo().height);
		}
		shaperenderer.end();
```

Sendo shaperenderer un obxecto da clase ShapeRenderer instanciado no método create e asinándolle as coordenadas da cámara no método RESIZE da forma:

```
		shaperenderer.setProjectionMatrix(camaraortho.combined);
```

Exercicio:
Facer que os peixes poidan vir da parte esquerda aleatoriamente.

Pista: lembrade que se poñemos un número negativo no valor width do método draw do spritebatch debuxa a textura invertida, PERO as súas coordenadas seguen sendo como se estivera debuxado 'normalmente'.

Necesitamos un campo boolean que indique de onde ven o peixe.

Para ser un pouco máis difícil (e real) podemos facer que o rectángulo que conforma o peixe sexa máis pequeno (que se corresponda coa súa boca máis ou menos). Para isto modificamos o tamaño do rectángulo a súa metade no eixe Y e a un terzo no eixe X.

Solución:

O primeiro que temos que pensar é que imos necesitar un campo que vai indicar se o peixe ven da esquerda ou dereita.
Cando veña da esquerda a velocidade vai ter que se negativa (mirade o método update) e a posición inicial x ten que ser 0 ou mellor o seu ancho en negativo.

Facemos todo isto:

Arquivo Peixe.java

```
        /**
         * Indica se sae da esquerda da pantalla
         */
        private boolean saedaesquerda;
        public Peixe() {
                super(new Vector2(Mundo.TAMAÑO_MUNDO.x,Mundo.xerarAleatorio(1,100)), new
Vector2(Mundo.xerarAleatorio(8,30),0), Mundo.xerarAleatorio(30, 60));
                tamaño.y = tamaño.x;            // O peixe ten que ter o mesmo ancho que alto. Como o ancho o
xeramos aleatoriamente, unha vez o sabemos llo asinamos o y
                getRectangulo().setHeight(tamaño.y/2);  // O rectángulo se crea a partires do tamaño.
                getRectangulo().setWidth(tamaño.x/3);   // O rectángulo se crea a partires do tamaño

                pescado=false;

                int tipo = Mundo.xerarAleatorio(0, 3);
                switch(tipo){
```

```
                        case 0:
                                tipopeixe = Tipos_peixe.LILA;
                                break;
                        case 1:
                                tipopeixe = Tipos_peixe.AMARELO;
                                break;
                        case 2:
                                tipopeixe = Tipos_peixe.AZUL;
                                break;

                }
                velocidade = velocidade_max;
                if(Mundo.xerarAleatorio(0, 2)==1) {
                        saedaesquerda=true;
                        velocidade = -velocidade;
                        posicion.x = -tamaño.x;
                }
                else saedaesquerda=false;

        }
        /**
         * Sobre escribimos o método para que o rectángulo estea na parte da boca cando o peixe ven da esquerda
         */
        public void actualizarRectangulo() {
                super.actualizarRectangulo();
                if (saedaesquerda){
                        rectangulo.x = posicion.x+tamaño.x-rectangulo.width;
                }
        }

        /**
         * Indica se o peixe sae da esquerda ou non. Necesario para controlar os límites
         * @return saedaesquerda
         */
        public boolean getSaedaesquerda() {
                return saedaesquerda;
        }
```

Agora temos que pensar en varias cousas na clase PrincipalXogo.

Arquivo PrincipalXogo.java

- Cando sae da esquerda, para que desapareza o valor de x xa non pode ser igual ó seu ancho en negativo, se non que ten que ser o ancho do mundo.

```
        private void controlarPeixes(){
                for(Peixe peixe: mundo.getPeixes()){
                        if (peixe.getPescado()) {       // Está pescado polo gancho
                                peixe.setPosicion(gancho.getPosicion().x,gancho.getPosicion().y);
                        }

                        if (peixe.getPosicion().y==Gancho.LIMITE_Y_SUPERIOR){   // Chegou arriba
                                mundo.getPeixes().removeValue(peixe, true);
                        }

                        if ((peixe.getPosicion().x<=-peixe.tamaño.x) && !peixe.getSaedaesquerda()){     // Chega ó final da pantalla pola esquerda
                                mundo.getPeixes().removeValue(peixe, true);
                        }
                        if ((peixe.getPosicion().x>=Mundo.TAMAÑO_MUNDO.x) && peixe.getSaedaesquerda()){ // Chega ó final da pantalla pola esquerda
                                mundo.getPeixes().removeValue(peixe, true);
                        }
                }

        }
```

- Temos que debuxar o peixe ó revés cando ven da esquerda, pero axustando a posición X para que o debuxe desprazado o seu ancho:


```
/**
 * Dibuxa os peixes
 */
private void dibuxarPeixes(float delta){

        for (Peixe peixe : mundo.getPeixes()){
                float tam,posx;
                if (peixe.getSaedaesquerda()) {
                        tam = -peixe.tamaño.x;
                        posx = peixe.getPosicion().x + peixe.tamaño.x;
                }
                else  {
                        tam = peixe.tamaño.x;
                        posx = peixe.getPosicion().x;
                }

                if (peixe.getTipopeixe()==Peixe.Tipos_peixe.LILA){
                        currentFrame = AssetsXogo.peixe_lila.getKeyFrame(stateTime, true);
                        spritebatch.draw(currentFrame,posx,peixe.getPosicion().y,tam,peixe.tamaño.y);
                }
                else if (peixe.getTipopeixe()==Peixe.Tipos_peixe.AMARELO){
                        currentFrame = AssetsXogo.peixe_amarelo.getKeyFrame(stateTime, true);
                        spritebatch.draw(currentFrame,posx,peixe.getPosicion().y,tam,peixe.tamaño.y);
                }
                else if (peixe.getTipopeixe()==Peixe.Tipos_peixe.AZUL){
                        spritebatch.draw(AssetsXogo.peixe_azul,posx,peixe.getPosicion().y,
                                        tam,peixe.tamaño.y);
                }
        }
}
```

Agora imos facer que cando o peixe está enganchado e chega arriba teremos que gardar nalgún sitio cantos peixes leva collidos o pescador e debuxalos. O mellor sitio para gardar dita información será na clase Pescador.

Arquivo Pescador.java
Engadimos unha propiedade num_peixes_collidos co método get.
Engadimos un método aumentarNumPeixesCollidos que incremente en un o número de peixes collidos.

Engadimos o método inicializaPeixesCollidos para poñer a 0. Necesario para cando remate o xogo e empece un novo.

```
/**
 * Número de peixes pescados
 */
private int num_peixes_collidos;

public Pescador(){
        super(new Vector2(50,128),new Vector2(50,50),83);
        num_peixes_collidos=0;
}

/**
 * Devolve o número de peixes collidos
 * @return num_peixes_collidos
 */
public int getNum_peixes_collidos() {
        return num_peixes_collidos;
}

/**
 * Pon a 0 o número de peixes collidos
 */
public void inicializaPeixesCollidos() {
        this.num_peixes_collidos = 0;
}

/**
 * Incrementa en un o número de peixes collidos
 */
public void aumentarNumPeixesCollidos(){
        num_peixes_collidos++;
}
```

Exercicio: Facer que se incremente o número de peixes cando se collen e debuxar na parte superior esquerda en forma de fila tantos peixes como teña collido o pescador.

Solución:

Arquivo PrincipalXogo.java

```
private void controlarPeixes(){
        for(Peixe peixe: mundo.getPeixes()){
                if (peixe.getPescado()) {       // Está pescado polo gancho
                        peixe.setPosicion(gancho.getPosicion().x,gancho.getPosicion().y);
                }

                if (peixe.getPosicion().y==Gancho.LIMITE_Y_SUPERIOR){   // Chegou arriba
                        pescador.aumentarNumPeixesCollidos();
                        mundo.getPeixes().removeValue(peixe, true);
                ……..

private void dibuxarNumPeixesCollidos(){
        int posx;
        for (int cont=0;cont<pescador.getNum_peixes_collidos();cont++){
                posx = 5+10*cont;
                spritebatch.draw(AssetsXogo.peixe_azul, posx, Mundo.TAMAÑO_MUNDO.y-10, 9,9);
        }
}

@Override
public void render() {

        ……..
```

```
                dibuxarSedal();
                dibuxarPeixes(delta);
                dibuxarNumPeixesCollidos();

                spritebatch.end();
                ..................
```

Outra cousa que temos que arranxar é que cando o gancho ten un peixe non poida coller outro peixe.

Exercicio: intentade facer o anterior.
Pista: usar unha propiedade nova na clase Gancho.

Solución:
Engadimos unha propiedade nova ó gancho que indique se ten un peixe e modificamos o método controlarGancho para que no caso que teña un peixe que non comprobe por outros peixes.

Arquivo Gancho.java

```
        private boolean tenunpeixe;     // O gancho ten un peixe para que non poida coller outro.
        public Gancho(){
                super(new Vector2(0,158),new Vector2(10,10), 30);
                rectangulo.setHeight(tamaño.x/2); // A altura do gancho é a mitade, xa que engloba a bola
amarela

                pescando=false;
                tenunpeixe=false;
        }
        /**
         * Devolve true se o gancho ten un peixe.
         * @return tenunpeixe
         */
        public boolean getTenunpeixe() {
                return tenunpeixe;
        }

        /**
         * Modifica o valor que indica se o gancho ten un peixe
         * @param tenunpeixe: o novo valor
         */
        public void setTenunpeixe(boolean tenunpeixe) {
                this.tenunpeixe = tenunpeixe;
        }
```

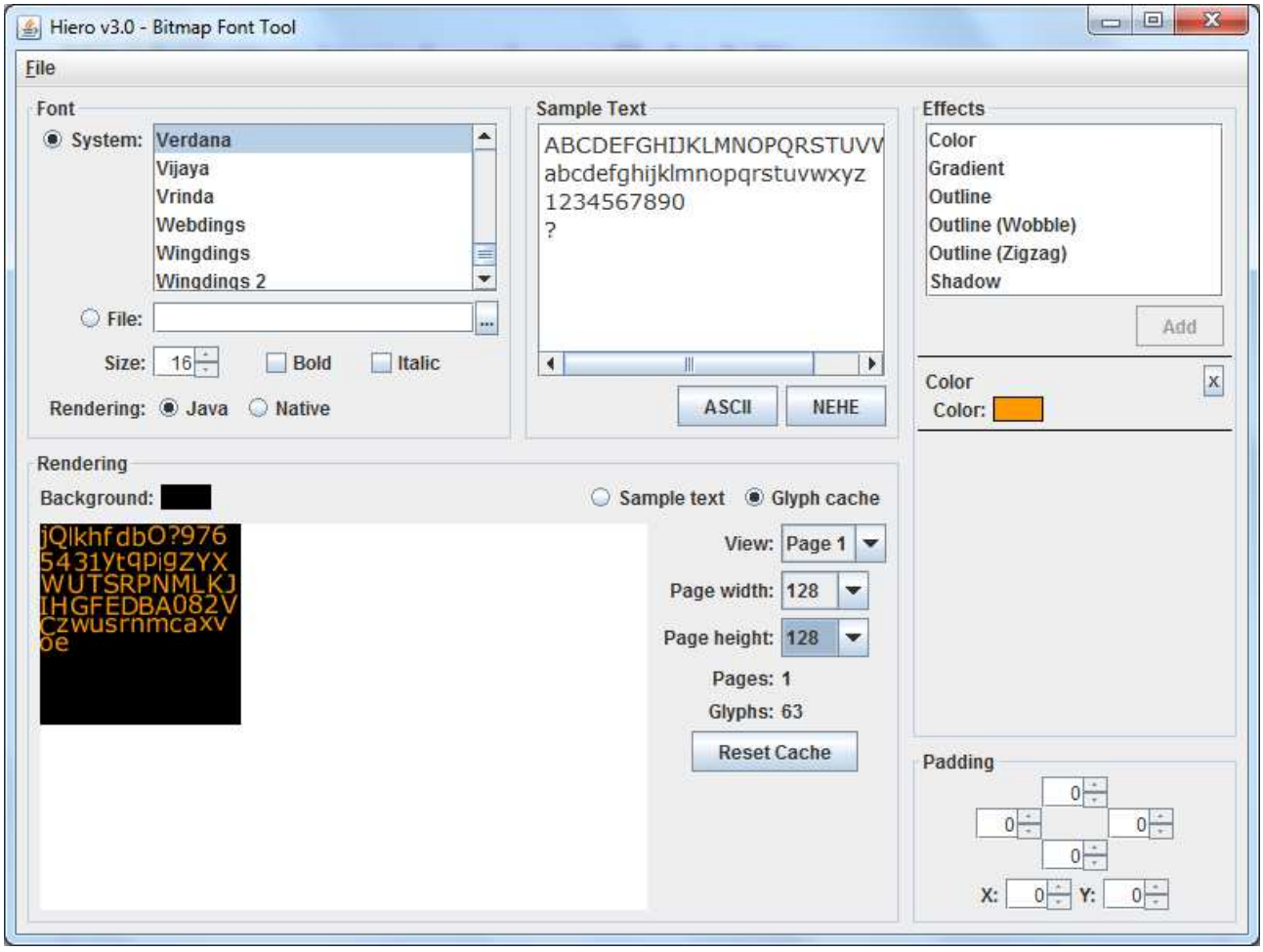
Arquivo PrincipalXogo.java

```
        private void controlarGancho(){
                if (gancho.getPosicion().y>Gancho.LIMITE_Y_SUPERIOR){ // Poñer sempre '>' nunca '==' xa que
se move con valores float´s
                        gancho.setVelocidade(0);
                        gancho.setPescando(false);
                        gancho.getPosicion().y=Gancho.LIMITE_Y_SUPERIOR;
                }
                else {
                        if (gancho.getPosicion().y<=0){  // Limite inferior
                                gancho.setVelocidade(+gancho.velocidade_max);
                        }
                }

                if (!gancho.getTenunpeixe()){
                        // Comprobamos se o gancho colle a un peixe
                        for (Peixe peixe : mundo.getPeixes()){
                                if (Intersector.overlapRectangles(peixe.getRectangulo(),
                                    gancho.getRectangulo())){
```

```
                                peixe.setPescado(true);
                                gancho.setTenunpeixe(true);
                            }
                        }
                    }

        }

        private void controlarPeixes(){

                        ........................
                        if (peixe.getPosicion().y==Gancho.LIMITE_Y_SUPERIOR){    // Chegou arriba
                                pescador.aumentarNumPeixesCollidos();
                                mundo.getPeixes().removeValue(peixe, true);
                                gancho.setTenunpeixe(false);
                        }
                        ..................
```

Agora (lembrade que polo de agora estamos a aprender como se fan as cousas, pero non da forma correcta ???) imos facer máis entretido o xogo e imos poñer un contador de tempo, de tal forma que o noso protagonista ten que recoller 10 peixes en dous minutos se quere volver a casa.

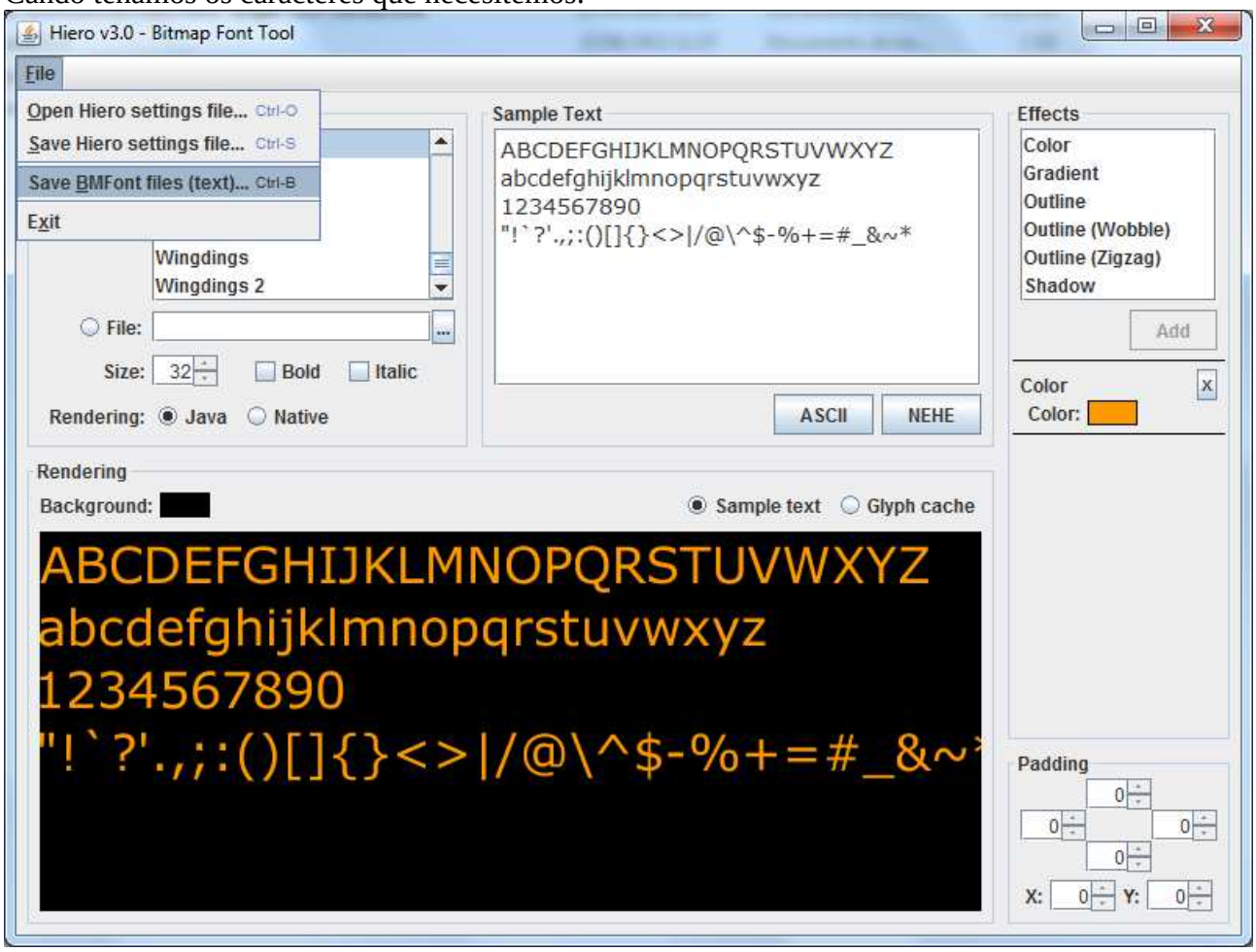
O cronómetro xa o temos (stateTime) pero necesitamos visualizar texto na pantalla.
O texto necesita uns arquivos especias coa extensión fnt. Para crealos imos necesitar outra ferramenta denominada hiero a cal a tedes no DVD (/SOFTWARE/Fontes/hiero.jnlp)



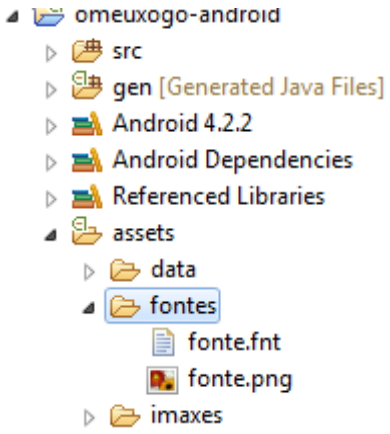
Máis información en: http://code.google.com/p/libgdx/wiki/Hiero
Para executala: https://wiki.libgdx.googlecode.com/git/jws/hiero.jnlp
Un paso a paso: http://singletechgames.com/2012/04/20/crear-archivos-fnt-con-hiero/

A idea é crear un png con todas as letras que se necesiten para o noso xogo (lembrade poñer as letras acentuadas).

Cando teñamos os caracteres que necesitemos:



Isto crea dous arquivos: fonte.fnt e fonte.png
Imos copialos a un cartafol 'fontes' dentro de 'assets' no proxecto Android dende o DVD (/DVD/Proxectos Eclipse/Proxecto 2D peixes/fontes).



Agora temos que cargar ditas fontes nun bitmap.

Exemplo de uso de BitmapFont: http://code.google.com/p/libgdx-users/wiki/addingText2D

Imos a clase AssetsXogo, definimos unha propiedade bitmapfont da clase BitmapFont e creamos un novo método de nome cargarFontes que sexa chamado dende cargarTexturas no que se cargue a fonte creada previamente.

Arquivo AssetsXogo.java
Creamos un novo método de nome cargarFontes.
Chamamos a dito método dende cargarTexturas.
Liberamos o bitmabfont dende o método liberarTexturas.

```
/**
 * Fonte a usar polo xogo
 */
public static BitmapFont bitmapfont;

/**
 * Carga o arquivo png coas fontes de texto
 */
public static void cargarFontes(){
	bitmapfont = new BitmapFont(Gdx.files.internal("fontes/fonte.fnt"), false);
}
public static void cargarTexturas(){
	.....................
	cargandoAnimacions();
	cargarFontes();
}
public static void liberarTexturas(){
	if (fondopecera!=null)
		fondopecera.dispose();
	if (texturas_todas!=null)
		texturas_todas.dispose();
	if (bitmapfont!=null)
		bitmapfont.dispose();
}
```

Despois imos a declarar unha constante de clase na clase Mundo que sexa os segundos que teñen que pasar ata que remate o xogo. Como temos o contador de tempo stateTime, o cronómetro será a constante menos o tempo transcorrido.

Arquivo Mundo.java

```
        /**
         * Número de segundos ata que remata o xogo
         */
        public static final int NUM_SEGUNDOS_XOGO=120;
```

Agora temos que debuxar o texto na pantalla.
Isto se fai có método draw da clase BitmapFont.

NOTA IMPORTANTE: cando utilizamos cadeas, cada vez que facemos un + entre dúas cadeas se crea un novo obxecto String que despois ten que eliminar o garbage collection.
Cando manexemos cadeas é mellor utilizar a clase StringBuilder.

Podemos ver as diferenzas en:
http://www.java-tips.org/java-se-tips/java.lang/difference-between-string-stringbuffer-and-stringbu.html

Arquivo PrincipalXogo.java

```
        private StringBuilder stringbuilder;    // Para traballar con cadeas que van ser visualizadas

        @Override
        public void create() {
                // TODO Auto-generated method stub
                stateTime=0;
                tempoParaCrearPeixe = 0;
                stringbuilder = new StringBuilder(3);   // O crono ten 3 números
                ……….

        /**
         * Dibuxa o cronometro
         */
        private void dibuxarCronometro(){
                stringbuilder.setLength(0);     // Borramos o strinbuilder
                AssetsXogo.bitmapfont.draw(spritebatch, stringbuilder.append((int)(Mundo.NUM_SEGUNDOS_XOGO-
stateTime)), Mundo.TAMAÑO_MUNDO.x-30, Mundo.TAMAÑO_MUNDO.y);

        }

        @Override
        public void render() {

                ……..................

                spritebatch.begin();

                dibuxarFondo();
                dibuxarPescador();
                dibuxarGancho();
                dibuxarSedal();
                dibuxarPeixes(delta);
                dibuxarNumPeixesCollidos();

                dibuxarCronometro();

                spritebatch.end();
                ……..........................
```

Agora teríamos que controlar que o cronómetro chegue a 0 e nese caso cambiar de pantalla.
Para facer isto imos refacer todo o xogo para seguir o patrón de programación **Modelo – Vista – Controlador (MVC)**.

O concepto é moi sinxelo.
Dividimos o xogo en tres partes:

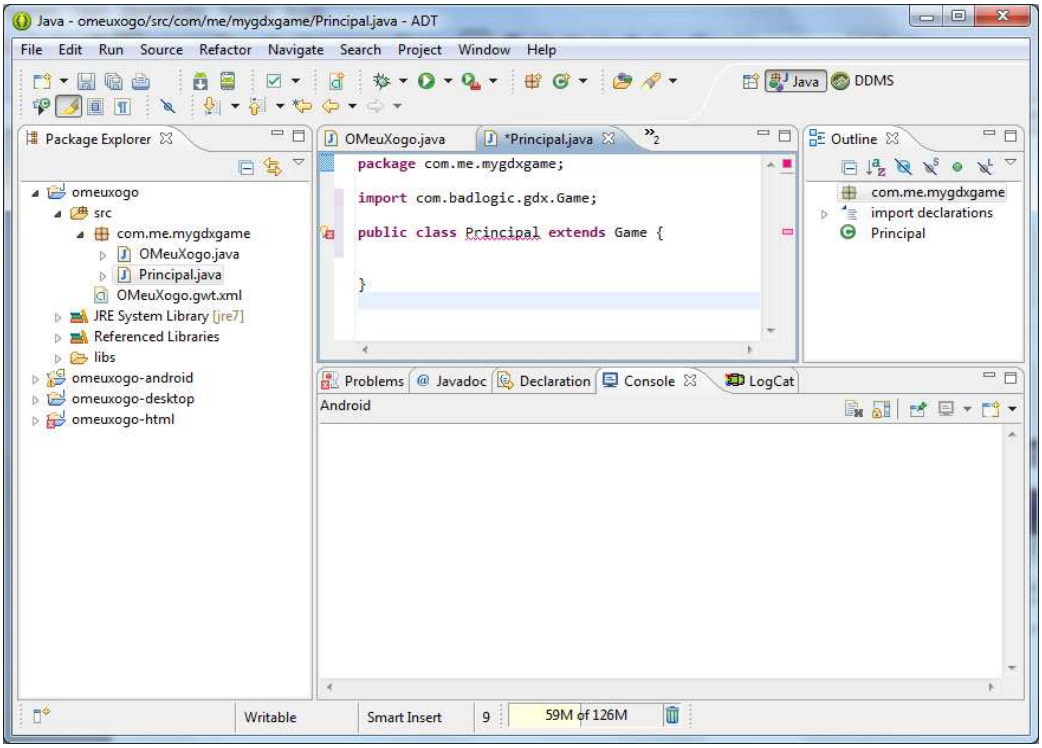
- Modelo: onde están definidas as clases que nos serven de base para o noso mundo. Isto xa o estamos a facer agora. Serían as clases Mundo, Personaxe, Piexe, Gancho, Pescador.
- Vista: aquí só se debuxan os obxectos que conforman o noso mundo e a cámara en base as súas propiedades de posición e tamaño.
- Controlador: controla todo o que sucede no noso mundo e modifica as propiedades dos obxectos que pertencen ó Modelo.

  A maiores imos ter unha zona de nome Pantallas que serán cada unha das pantallas do noso xogo e que terán as interfaces que permiten xestionar os eventos e que informarán á clase controladora de que evento se trata.

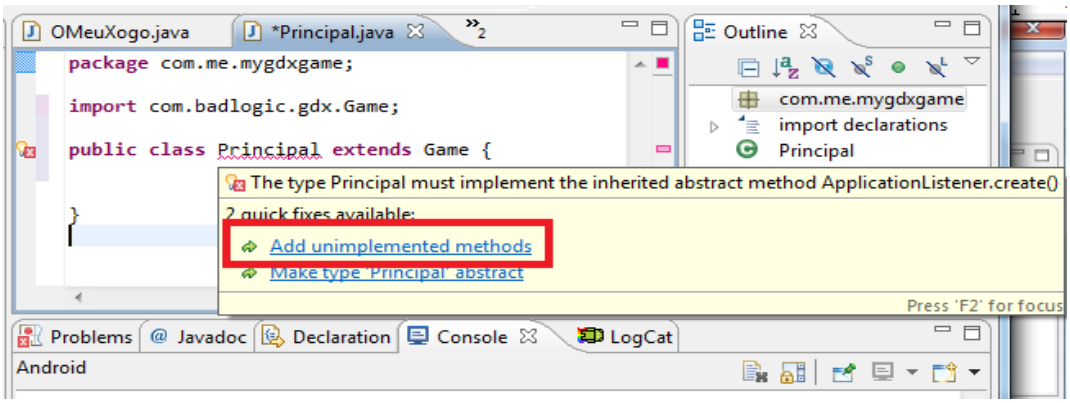
Agora mesmo estamos a mesturar a parte de Vista-Controlador na clase PrincipalXogo.

Imos crear a pantalla onde se vai desenrolar o xogo.
Para iso primeiro temos que facer as seguintes modificacións:

- Definimos unha clase de nome Principal que derive da clase Game. A clase Game implementa a interface ApplicationListener e vainos permitir cambiar de pantalla dunha forma moi sinxela.



Como vemos temos un erro na clase (liña suliñada en vermello debaixo do nome da clase). Isto é debido a que ó derivar da clase Game está obrigando a implementar aqueles métodos que non están sobrescritos na clase Game (que implementa a interface). Neste caso é o método onCreate.
En Eclipse non temos que escribir dito método, se pode xerar automática situándonos sobre a clase Principal e escollendo a opción 'Add unimplemented Methods':



Podedes ver as propiedades e métodos da clase Game en
http://libgdx.badlogicgames.com/nightlies/docs/api/com/badlogic/gdx/Game.html

Esta vai ser a clase que van chamar as diferentes pantallas do noso xogo, xa que como comentamos antes, a clase Game tamén implementa a interface ApplicationListener.

Facemos os cambios nas diferentes versións:

Versión Android:

```
public class MainActivity extends AndroidApplication {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        AndroidApplicationConfiguration cfg = new AndroidApplicationConfiguration();
        cfg.useAccelerometer=true;
        cfg.useCompass=false;
        cfg.useWakelock=true;
        cfg.useGL20 = false;

        initialize(new Principal(), cfg);
    }
}
```

Versión Desktop:

```
public class Main {
        public static void main(String[] args) {
                LwjglApplicationConfiguration cfg = new LwjglApplicationConfiguration();
                cfg.title = "O Meu Xogo";
                cfg.useGL20 = false;
                cfg.width = 800;
                cfg.height = 480;

                new LwjglApplication(new Principal(), cfg);
        }
}
```

Versión HTML:

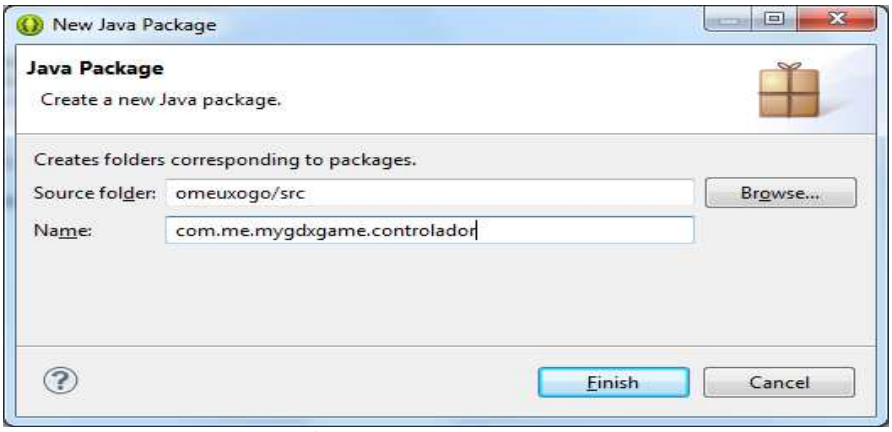
```
public class GwtLauncher extends GwtApplication {
        @Override
```

```
        public GwtApplicationConfiguration getConfig () {
                GwtApplicationConfiguration cfg = new GwtApplicationConfiguration(480, 320);
                return cfg;
        }

        @Override
        public ApplicationListener getApplicationListener () {
                return new Principal();
        }
}
```

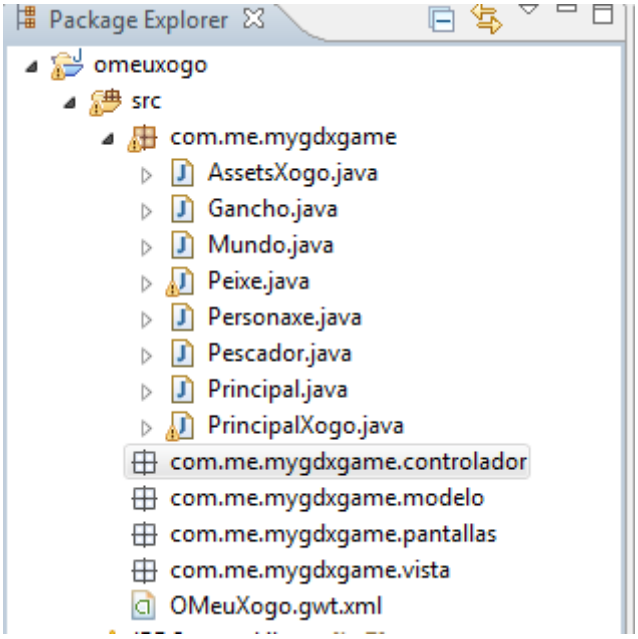
Dende a clase Game podemos indicar que Screen (pantalla) queremos cargar chamando ó método setScreen.

Agora seguindo o patrón de deseño Modelo – Vista – Controlador (MVC) imos crear un paquete para:

- As pantallas
- O modelo
- As vistas
- Os Controladores (cada pantalla pode ter o seu controlador)



Exemplo de paquete onde irán as clases 'controladoras' do noso xogo



Agora creamos no paquete 'pantallas' unha clase de nome 'PantallaXogo' que implemente a interface Screen.

Nota: ó facelo lembrade premer as teclas Ctrol+Shift+O para importar a clase e despois implementar os métodos da interface como fixemos antes (situándonos enriba do nome da clase).

O código é bastante perecido o da interface ApplicationListener pero incorpora dous novos métodos: show e hide. Show se executa cando se carga a pantalla e aparece (se chama unha vez) e hide cando saímos da pantalla (se executa despois do evento pause).

```
public class PantallaXogo implements Screen {

	@Override
	public void render(float delta) {
		// TODO Auto-generated method stub

	}

	@Override
	public void resize(int width, int height) {
		// TODO Auto-generated method stub

	}

	@Override
	public void show() {
		// TODO Auto-generated method stub

	}

	@Override
	public void hide() {
		// TODO Auto-generated method stub

	}

	@Override
	public void pause() {
```

```
                // TODO Auto-generated method stub

        }

        @Override
        public void resume() {
                // TODO Auto-generated method stub

        }

        @Override
        public void dispose() {
                // TODO Auto-generated method stub

        }

}
```

Podedes ver a descrición da clase en
http://libgdx.badlogicgames.com/nightlies/docs/api/com/badlogic/gdx/Screen.html

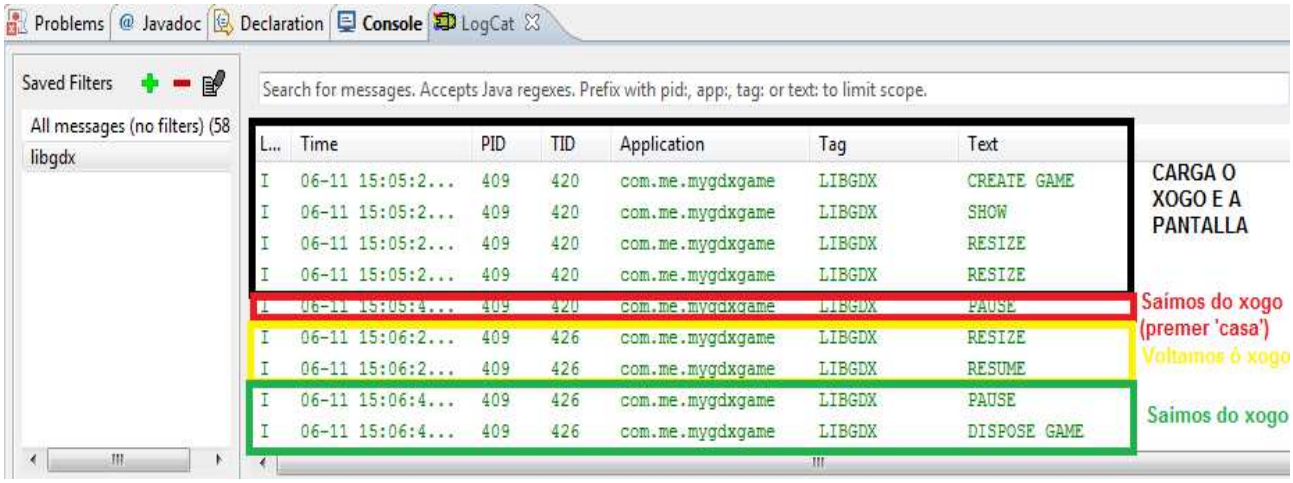
Como se pode ver na documentación o método 'dispose' non é chamado automáticamente polo que teremos que chamalo dende o método 'dispose' da clase Game.

Agora que temos a clase Screen imos crear un obxecto de dita clase na clase Game e asinarlle dita pantalla para que a cargue.

Arquivo Principal.java
Cargamos a pantalla do xogo.

```
public class Principal extends  Game {

        PantallaXogo pantallaxogo;

        @Override
        public void create() {
                // TODO Auto-generated method stub

                Gdx.app.log("LIBGDX", "CREATE GAME");
                pantallaxogo = new PantallaXogo();
                setScreen(pantallaxogo);
        }

        @Override
        public void dispose(){
                Gdx.app.log("LIBGDX", "DISPOSE GAME");
                pantallaxogo.dispose();
        }
}
```

Se sacades por consola unha mensaxe en cada un dos eventos podedes observar o seguinte:



Como a idea é ter varias pantallas e ir dunha a outra, necesitamos o obxeto da clase Game (é o que ten o método setScreen) na clase PantallaXogo para cando queiramos cambiar de pantalla.

O faremos así:

Arquivo PantallaXogo.java
Ter unha referencia á clase Game.

```
	private Principal game;
	public PantallaXogo(Principal principal){
		game=principal;
	}
```

Arquivo Principal.java
Enviar unha referencia de game á pantalla.

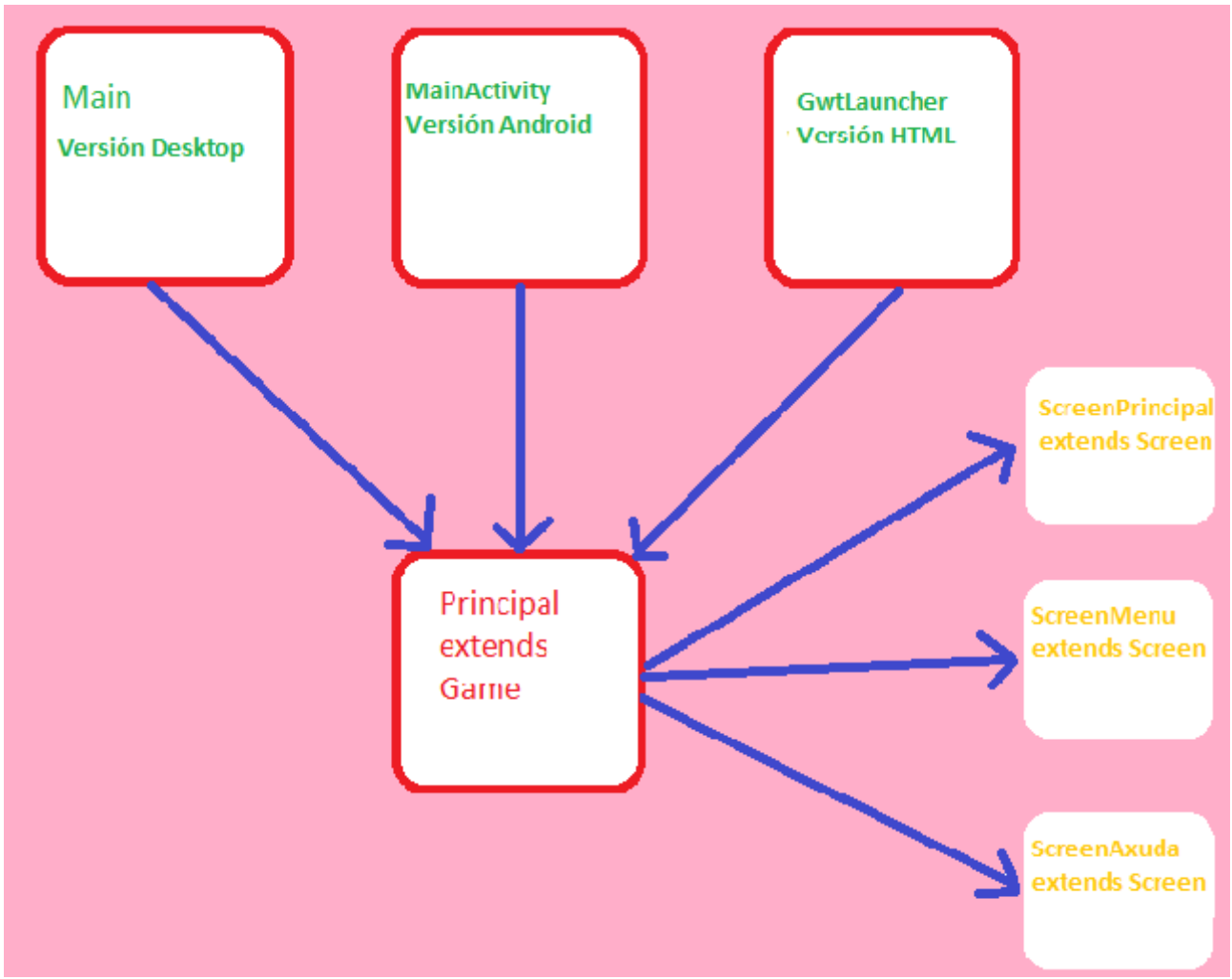
```
public class Principal extends  Game {

	PantallaXogo pantallaxogo;

	@Override
	public void create() {
		// TODO Auto-generated method stub

		pantallaxogo = new PantallaXogo(this);
		setScreen(pantallaxogo);
	}

	@Override
	public void dispose(){
		super.dispose();
		pantallaxogo.dispose();
	}
}
```

A función que vai ter a pantalla (Screen) é de informar á clase Controladora dos eventos que se produzan e chamar á clase Renderer (no paquete Vista) para que debuxe o mundo.

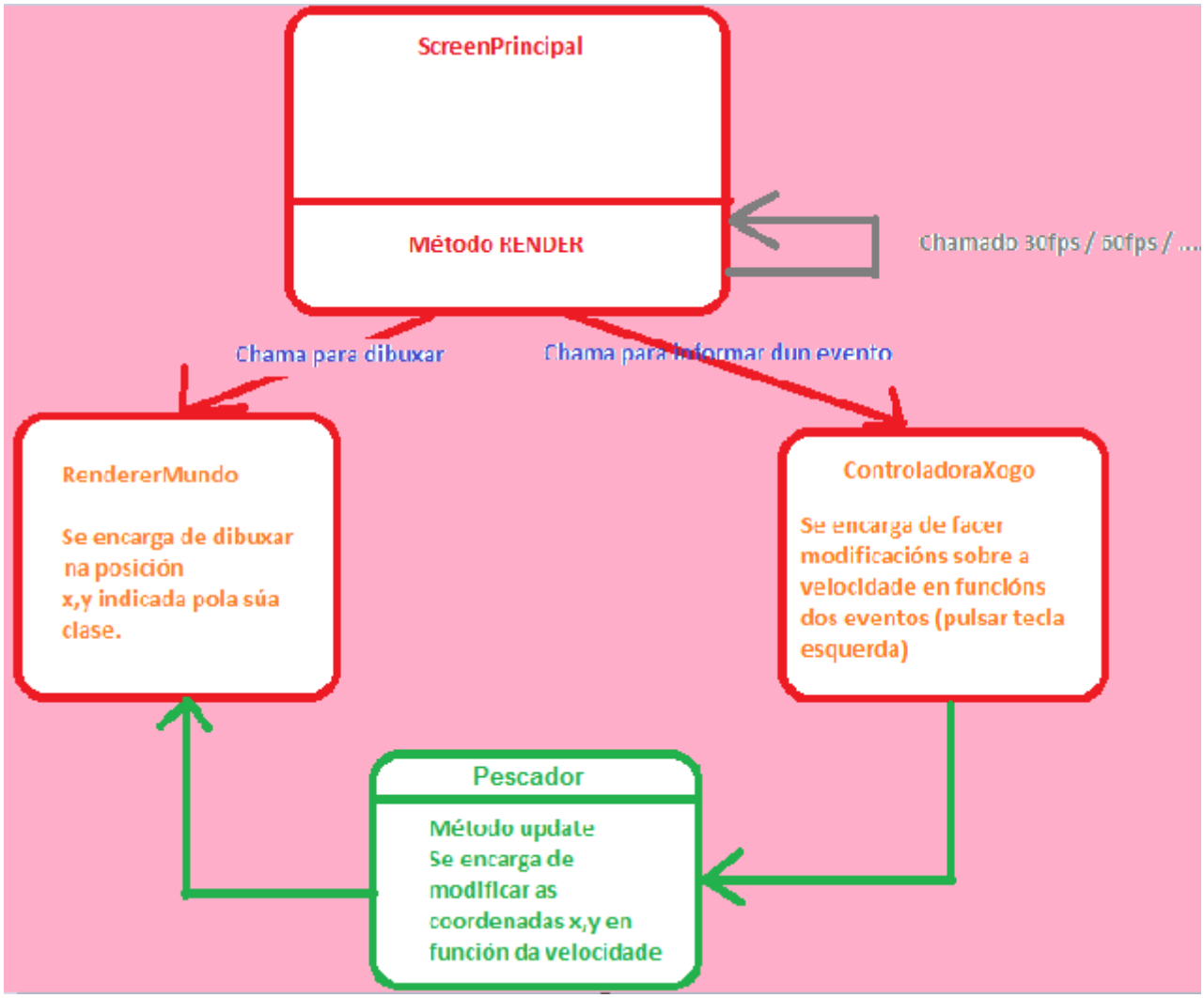
Parece un pouco lioso, pero imos facer un resumo do que estamos a facer:

Os nosos 3 proxectos chaman a mesma clase a cal ten que, ou ben implementar a interface ApplicationListener ou ben derivar da clase Game.

Neste segundo caso, establecemos a clase Screen que vai visualizarse co método setScreen dende a clase Principal.

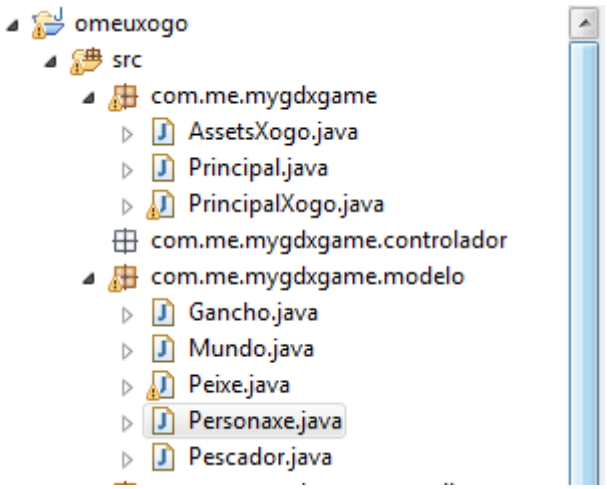


Unha vez situado na clase Screen correspondente, automaticamente o motor de xogos vai a chamar de forma continua (dependendo da potencia da máquina) ó método render da clase Screen. O que facemos nese método é chamar a clase que vai debuxar todo o da pantalla e a clase que vai controlar todo o que pasa. As dúas van facer uso das clase definidas no paquete de modelado:



Fixarse como o que estamos a facer é o 'esqueleto' do noso xogo.

Agora imos mover as clase que conforman o noso mundo para o paquete Modelo.
Así movemos as clases: Mundo – Pescador – Gancho – Peixe – Personaxe.



A continuación imos crear a clase que vai debuxar todo. Esta debe estar no paquete Vista e imos darlle de nome RendererXogo.
Dita clase é a que ten a cámara e o SpriteBatch.

Arquivo RendererXogo.java
Definimos un constructor no que lle pasamos unha referencia ó noso mundo (clase Mundo).
Definimos un método resize que será chamado dende o método resize da clase PantallaXogo.

```
public class RendererXogo {

	private Mundo mundo;

	public RendererXogo(Mundo mundo){
		this.mundo = mundo;
	}

	public void resize(int width, int height) {

	}
	public void dispose(){

	}
}
```

Despois volveremos a dita clase.

Agora imos definir a clase controladora. Imos chamarlle ControladorXogo e estará no paquete Controlador. Encargarase de controlar todo o que pasa no noso mundo (choques, movementos,..) e de responder as accións (input´s) dos eventos producidos polo dispositivo.

Como dende dita clase imos ter que modificar o estado do xogo (cando remate) e cambiar de pantalla, imos a enviarlle a propia clase Screen como referencia.

Arquivo ControladorXogo.java

```
public class ControladorXogo {

	private PantallaXogo pantallaxogo;
	private Mundo mundo;

	public ControladorXogo(Mundo mundo,PantallaXogo pantalla){
		pantallaxogo=pantalla;
		this.mundo = mundo;
	}
	public void dispose(){

	}

}
```

Imos a integrar estas dúas clases (ControladorXogo e RendererXogo) na clase PantallaXogo.
Por outra banda tamén cargaremos dende esta clase as texturas e as liberaremos chegado o caso.
Tamén xestionaremos os eventos polo que implementaremos as interfaces e informaremos o motor que esta clase xestiona os eventos.

Arquivo PantallaXogo.java

```
public class PantallaXogo implements Screen,InputProcessor {

	private Principal game;
	private Mundo mundo;
	private ControladorXogo controladorxogo;
	private RendererXogo rendererxogo;


	public PantallaXogo(Principal principal){
		game=principal;

	}

	@Override
	public void render(float delta) {
		// TODO Auto-generated method stub

	}

	@Override
	public void resize(int width, int height) {
		// TODO Auto-generated method stub
		rendererxogo.resize(width, height);
	}

	@Override
	public void show() {
		// TODO Auto-generated method stub
		AssetsXogo.cargarTexturas();

		mundo = new Mundo();
		controladorxogo = new ControladorXogo(mundo, this);
		rendererxogo = new RendererXogo(mundo);

		Gdx.input.setInputProcessor(this);
	}

	@Override
	public void hide() {
		// TODO Auto-generated method stub
	}
```

```
@Override
public void pause() {
	// TODO Auto-generated method stub
	Gdx.input.setInputProcessor(null);
}

@Override
public void resume() {
	// TODO Auto-generated method stub
	Gdx.input.setInputProcessor(this);
}

@Override
public void dispose() {
	// TODO Auto-generated method stub
	Gdx.input.setInputProcessor(null);
	if (rendererxogo!=null)
		rendererxogo.dispose();
	if (controladorxogo!=null)
		controladorxogo.dispose();
	AssetsXogo.liberarTexturas();
}

@Override
public boolean keyDown(int keycode) {
	// TODO Auto-generated method stub
	return false;
}

@Override
public boolean keyUp(int keycode) {
	// TODO Auto-generated method stub
	return false;
}

@Override
public boolean keyTyped(char character) {
	// TODO Auto-generated method stub
	return false;
}

@Override
public boolean touchDown(int screenX, int screenY, int pointer, int button) {
	// TODO Auto-generated method stub
	return false;
}

@Override
public boolean touchUp(int screenX, int screenY, int pointer, int button) {
	// TODO Auto-generated method stub
	return false;
}

@Override
public boolean touchDragged(int screenX, int screenY, int pointer) {
	// TODO Auto-generated method stub
	return false;
}

@Override
public boolean mouseMoved(int screenX, int screenY) {
	// TODO Auto-generated method stub
	return false;
}

@Override
public boolean scrolled(int amount) {
	// TODO Auto-generated method stub
	return false;
}
```

```
}
```

Agora temos que 'desprazar' o código que tiñamos na clase PrincipalXogo á clase RendererXogo, pero só aquelo que serve para dibuxar.

Propiedades:

```
private OrthographicCamera camaraortho;
private SpriteBatch spritebatch;
private Mundo mundo;
private Pescador pescador;
private Gancho gancho;
private Array<Peixe> peixes;

private float delta;
private float stateTime;// Leva o tempo transcurrido para cambiar de frame.
private TextureRegion currentFrame;      // Frame actual do peixe a dibuxar
private StringBuilder stringbuilder;     // Para traballar con cadeas que van ser visualizadas
```

Non temos método create polo que temos que levar as ordes necesarias dende Create ó Constructor:

```
public RendererXogo(Mundo mundo){
	this.mundo = mundo;

	stateTime=0;
	stringbuilder = new StringBuilder(2);

	camaraortho = new OrthographicCamera();
	spritebatch = new SpriteBatch();

	pescador = mundo.getPescador();
	gancho = mundo.getGancho();
	peixes = mundo.getPeixes();

}
```

Modificamos o método resize:

```
public void resize(int width, int height) {
	camaraortho.setToOrtho(false,Mundo.TAMAÑO_MUNDO.x,Mundo.TAMAÑO_MUNDO.y);
	camaraortho.update();

	spritebatch.setProjectionMatrix(camaraortho.combined);
}
```

Método dispose:

```
public void dispose(){
	spritebatch.dispose();
}
```

Levamos todos os método de dibuxo que tiñamos:

```
/** Dibuxa o fondo do xogo
 *
 */
private void dibuxarFondo(){
	spritebatch.disableBlending();
	spritebatch.draw(AssetsXogo.fondopecera,0,0,Mundo.TAMAÑO_MUNDO.x,Mundo.TAMAÑO_MUNDO.y);
	spritebatch.enableBlending();
}

/**
 * Dibuxa o pescador
```

```
         */
        private void dibuxarPescador(){


spritebatch.draw(AssetsXogo.pescador,pescador.getPosicion().x,pescador.getPosicion().y,pescador.tamaño.x,pesc
ador.tamaño.y);
        }

        /**
         * Dibuxa o gancho do pescador
         */
        private void dibuxarGancho(){


spritebatch.draw(AssetsXogo.gancho,gancho.getPosicion().x,gancho.getPosicion().y,gancho.tamaño.x,gancho.tamañ
o.y);
        }

        /**
         * Dibuxa o sedal
         */
        private void dibuxarSedal(){
                spritebatch.draw(AssetsXogo.sedal,pescador.getPosicion().x+41+
(gancho.tamaño.x/2),Gancho.LIMITE_Y_SUPERIOR+gancho.tamaño.y,0.75f,-(Gancho.LIMITE_Y_SUPERIOR-
gancho.getPosicion().y));
        }

        /**
         * Dibuxa os peixes
         */
        private void dibuxarPeixes(float delta){

                for (Peixe peixe : mundo.getPeixes()){
                        float tam,posx;
                        if (peixe.getSaedaesquerda()) {
                                tam = -peixe.tamaño.x;
                                posx = peixe.getPosicion().x + peixe.tamaño.x;
                        }
                        else  {
                                tam = peixe.tamaño.x;
                                posx = peixe.getPosicion().x;
                        }

                        if (peixe.getTipopeixe()==Peixe.Tipos_peixe.LILA){
                                currentFrame = AssetsXogo.peixe_lila.getKeyFrame(stateTime, true);
                                spritebatch.draw(currentFrame,posx,peixe.getPosicion().y,tam,peixe.tamaño.y);
                        }
                        else if (peixe.getTipopeixe()==Peixe.Tipos_peixe.AMARELO){
                                currentFrame = AssetsXogo.peixe_amarelo.getKeyFrame(stateTime, true);
                                spritebatch.draw(currentFrame,posx,peixe.getPosicion().y,tam,peixe.tamaño.y);
                        }
                        else if (peixe.getTipopeixe()==Peixe.Tipos_peixe.AZUL){

        spritebatch.draw(AssetsXogo.peixe_azul,posx,peixe.getPosicion().y,tam,peixe.tamaño.y);
                        }
                }
        }

        private void dibuxarNumPeixesCollidos(){
                int posx;
                for (int cont=0;cont<pescador.getNum_peixes_collidos();cont++){
                        posx = 5+10*cont;
                        spritebatch.draw(AssetsXogo.peixe_azul, posx, Mundo.TAMAÑO_MUNDO.y-10, 9,9);
                }
        }

        /**
         * Dibuxa o cronometro
         */
        private void dibuxarCronometro(){
                stringbuilder.setLength(0);     // Borramos o strinbuilder
```

```
                AssetsXogo.bitmapfont.draw(spritebatch, stringbuilder.append((int)(Mundo.NUM_SEGUNDOS_XOGO-
stateTime)), Mundo.TAMAÑO_MUNDO.x-30, Mundo.TAMAÑO_MUNDO.y);

        }
```

Creamos un método de nome render(float delta) que vai chamar a todos estes métodos (era o método render na clase PrincipalXogo).

```
        /**
         * Método que dibuxa todos os elementos do noso mundo.
         * Chamado dende o método render da clase PantallaXogo (Screen)
         * @param delta: tempo
         */
        public void render(float delta){
                delta = Gdx.graphics.getDeltaTime();
                stateTime += delta;

                spritebatch.begin();

                dibuxarFondo();
                dibuxarPescador();
                dibuxarGancho();
                dibuxarSedal();
                dibuxarPeixes(delta);
                dibuxarNumPeixesCollidos();

                dibuxarCronometro();

                spritebatch.end();


        }
```

Dende a clase PantallaXogo (método render) temos que chamar ó método render da clase RendererXogo.

Arquivo PantallaXogo.java

```
        @Override
        public void render(float delta) {
                // TODO Auto-generated method stub
                rendererxogo.render(delta);
        }
```

Se executades agora ides ver como o gancho non está onde debe.
Isto é debido a que non estamos chamando o método actualizarGancho que colocaba o gancho na posición do pescador (o faremos máis adiante).

Agora falta facer como dende a clase ScreenPrincipal se pode 'informar' á clase controladora de que se pulsou unha tecla.

Para isto imos definir uns estados na clase controladora que van a indicar que se pulsou.
No noso xogo imos ter tres 'estados' posibles: ESQUERDA, DEREITA,ACTIVARGANCHO.

Imos implementar ditos estados con variables booleanas que nos indicarán se se recibiu dende o dispositivo algunha entrada que faga modificar ditos estados (poder ser unha tecla, pulsar sobre unha parte da pantalla co dedo ou usar o acelerómetro nun dispositivo móbil).

Arquivo ControladoraXogo.java

Creamos un array de estados utilizando unha clase HashMap.

```
public enum Keys {
        ESQUERDA,DEREITA,ACTIVARGANCHO
}
HashMap<Keys, Boolean> keys = new HashMap<ControladorXogo.Keys, Boolean>();
{
        keys.put(Keys.ESQUERDA, false);
        keys.put(Keys.DEREITA, false);
        keys.put(Keys.ACTIVARGANCHO, false);
};
```

O que estamos a facer é:

- Crear unha clase enumeración cos seguintes 'estados':
```
public enum Keys {
        ESQUERDA,DEREITA,ACTIVARGANCHO
}
```
- Creamos un hasmap que ven ser un array no que gardamos unha variable boolena para cada estado posible:
```
HashMap<Keys, Boolean> keys = new HashMap<ControladorXogo.Keys, Boolean>();
```
- Inicilizamos dito mapa:
```
{
        keys.put(Keys.ESQUERDA, false);
        keys.put(Keys.DEREITA, false);
        keys.put(Keys.ACTIVARGANCHO, false);
};
```

Agora creamos un método para pulsar unha tecla e un método para liberar unha tecla. Ditos métodos serán chamados dende a clase ScreenPrincipal cando se preme a tecla. Un dos métodos fará true (se pulsa a tecla) e o outro fará false (se libera) o estado.

Arquivo ControladoraXogo.java

Creamos os métodos necesarios para modificar o estado do xogo.

```
/**
 * Modifica o estado do mapa de teclas e pon a true
 * @param tecla: tecla pulsada
 */
public void pulsarTecla(Keys tecla){
        keys.put(tecla, true);
}
/**
 * Modifica o estado do mapa de teclas e pon a false
 * @param tecla: tecla liberada
 */
public void liberarTecla(Keys tecla){
        keys.put(tecla, false);
}
```

Agora temos que relacionar a clase ControladorXogo coa clase PantallaXogo. Para isto imos crear un método update(float delta) na clase ControladorXogo que será chamado dende a clase PantallaXogo.

Arquivo ControladorXogo.java

Creamos método update para 'enlazar' a clase PantallaXogo coa clase ControladorXogo.

```
/**
 * Chamado dende a clase PantallaXogo (Screen)
 * @param delta: tempo transcurrido
 */
```

```
public void update(float delta){

}
```

Arquivo PantallaXogo.java
Chamamos o método update da clase ControladorXogo.

```
@Override
public void render(float delta) {
        // TODO Auto-generated method stub
        rendererxogo.render(delta);
        controladorxogo.update(delta);
}
```

Podemos definir as seguintes propiedades que imos necesitar nesta clase:

Arquivo ControladorXogo.java:

```
private Pescador pescador;
private Gancho gancho;
private Array<Peixe> peixes;
private float tempoParaCrearPeixe;
```

e dámoslles uns valores no Construtor:

```
public ControladorXogo(Mundo mundo,PantallaXogo pantalla){
        pantallaxogo=pantalla;
        this.mundo = mundo;

        tempoParaCrearPeixe = 0;
        pescador = mundo.getPescador();
        gancho = mundo.getGancho();
        peixes = mundo.getPeixes();

}
```

Agora podemos copiar os métodos que 'controlaban' todo, pero un deles dará un erro (crearPeixe) xa que utiliza unha propiedade que queda na clase RendererXogo (stateTime).

Aquí teríamos a opción de crear unha propiedade nova en Mundo que fora o cronómetro, pero para acortar o tempo imos facer que dita propiedade sexa de tipo public static e así poderemos acceder dende a clase ControladorXogo:

Arquivo RendererXogo.java
Modificamos a propiedade stateTime.

```
public static float stateTime;  // Leva o tempo transcurrido para cambiar de frame.
```

Podemos copiar os seguintes métodos dende a clase PrincipalXogo á clase ControladorXogo:

Arquivo ControladorXogo.java

```
/**
 * Actualiza a posición en función da posición do pescador
 * Chama a update do gancho
 * @param delta
```

```
         */
        private void actualizarGancho(float delta){

                gancho.setPosicion(pescador.getPosicion().x+42,gancho.getPosicion().y);
                gancho.update(delta);

        }

        /**
         * Elimina o peixe do array ó chegar ó final
         */
        private void controlarPeixes(){
                for(Peixe peixe: mundo.getPeixes()){
                        if (peixe.getPescado()) {       // Está pescado polo gancho
                                peixe.setPosicion(gancho.getPosicion().x,gancho.getPosicion().y);
                        }

                        if (peixe.getPosicion().y==Gancho.LIMITE_Y_SUPERIOR){    // Chegou arriba
                                pescador.aumentarNumPeixesCollidos();
                                mundo.getPeixes().removeValue(peixe, true);
                                gancho.setTenunpeixe(false);
                        }

                        if ((peixe.getPosicion().x<=-peixe.tamaño.x) && !peixe.getSaedaesquerda()){     //
Chega ó final da pantalla pola esquerda
                                mundo.getPeixes().removeValue(peixe, true);
                        }
                        if ((peixe.getPosicion().x>=Mundo.TAMAÑO_MUNDO.x) && peixe.getSaedaesquerda()){ //
Chega ó final da pantalla pola esquerda
                                mundo.getPeixes().removeValue(peixe, true);
                        }
                }

        }

        /**
         * Crea un peixe de forma aleatoria no tempo
         */
        private void crearPeixes(){
                if (mundo.getPeixes().size==Mundo.NUM_PEIXES_MAXIMO) return;
                if (tempoParaCrearPeixe <= stateTime){
                        mundo.engadirPeixe();
                        tempoParaCrearPeixe = stateTime+Mundo.xerarAleatorio(1, 5);
                }

        }

        /**
         * Actualiza a posición dos peixes
         */
        private void actualizarPeixes(float delta){
                for(Peixe peixe: mundo.getPeixes()){
                        peixe.update(delta);
                }
        }

        /**
         * Controla os límites do gancho
         * Controla cando se atopa cun peixe
         */
        private void controlarGancho(){
                if (gancho.getPosicion().y>Gancho.LIMITE_Y_SUPERIOR){ // Poñer sempre '>' nunca '==' xa que
se move con valores float´s
                        gancho.setVelocidade(0);
                        gancho.setPescando(false);
                        gancho.getPosicion().y=Gancho.LIMITE_Y_SUPERIOR;
                }
                else {
                        if (gancho.getPosicion().y<=0){  // Limite inferior
                                gancho.setVelocidade(+gancho.velocidade_max);
                        }
```

```
                }

                if (!gancho.getTenunpeixe()){
                        // Comprobamos se o gancho colle a un peixe
                        for (Peixe peixe : mundo.getPeixes()){
                                if (Intersector.overlapRectangles(peixe.getRectangulo(),
gancho.getRectangulo())){
                                        peixe.setPescado(true);
                                        gancho.setTenunpeixe(true);
                                }
                        }
                }

        }

        /**
         * Controla o pescador: límites
         */
        private void controlarPescador(){
                if (pescador.getPosicion().x >= Mundo.TAMAÑO_MUNDO.x-pescador.tamaño.x){
                        pescador.setPosicion(new Vector2(Mundo.TAMAÑO_MUNDO.x-
pescador.tamaño.x,pescador.getPosicion().y));
                }
                if (pescador.getPosicion().x <= 0){
                        pescador.setPosicion(new Vector2(0,pescador.getPosicion().y));
                }
        }
```

Chamaremos a ditos métodos dende o método update da clase ControladorXogo:

Arquivo ControladorXogo.java
Chamamos os métodos controladores.

```
        public void update(float delta){

                pescador.update(delta);
                actualizarGancho(delta);
                actualizarPeixes(delta);
                crearPeixes();

                controlarPescador();
                controlarGancho();
                controlarPeixes();
        }
```

Agora imos modificar a velocidade do pescador en función de se cambiou o estado do xogo (podedes ver como isto o facíamos dende a clase PrincipalXogo => Método keyDown)

Para isto creamos un novo método que vai controlar todos os cambios de estado e modificará a velocidade do pescador. Imos chamarlle a dito método procesarEntrada e será chamado dende o método update da clase ControladorXogo. Teremos que copiar o código da clase PrincipalXogo cando premíamos unha tecla (método keydown) e cando a liberábamos.

Arquivo ControladorXogo.java
Creamos o método procesarEntrada().
Trasladamos as ordes postas no evento touchdown e touchup da clase PrincipalXogo.
Chamamos a dito método dende o método update() da mesma clase.

```
        /**
         * Procesa as entradas de eventos que se produzan na clase PantallaXogo.
```

```
	 */
	private void procesarEntrada(){

		if ((!keys.get(Keys.ESQUERDA)) && (!keys.get(Keys.DEREITA))){
			pescador.setVelocidade(0);
		}
		if ((!gancho.getPescando())){
			if (keys.get(Keys.ESQUERDA)){
				pescador.setVelocidade(pescador.velocidade_max);
			}
			if (keys.get(Keys.DEREITA)){
				pescador.setVelocidade(-pescador.velocidade_max);
			}
			if (keys.get(Keys.ACTIVARGANCHO)){
				gancho.setVelocidade(-gancho.velocidade_max);
				gancho.setPescando(true);
			}
		}
		if ((gancho.getPescando()) && !(keys.get(Keys.ACTIVARGANCHO))){
			gancho.setVelocidade(+gancho.velocidade_max);
		}


	}
	public void update(float delta){

		pescador.update(delta);
		actualizarGancho(delta);
		actualizarPeixes(delta);
		crearPeixes();

		controlarPescador();
		controlarGancho();
		controlarPeixes();

		procesarEntrada();
	}
```

Agora modificamos os métodos keydown e keyup da clase PantallaXogo para poñer a true/false o estado da clase ControladorXogo.

Arquivo PantallaXogo.java

Modificamos os métodos touchdown e touchup.

```
	@Override
	public boolean keyDown(int keycode) {
		// TODO Auto-generated method stub
		if (keycode == Keys.LEFT)
			controladorxogo.pulsarTecla(ControladorXogo.Keys.ESQUERDA);
		if (keycode == Keys.RIGHT)
			controladorxogo.pulsarTecla(ControladorXogo.Keys.DEREITA);
		if (keycode == Keys.DOWN)
			controladorxogo.pulsarTecla(ControladorXogo.Keys.ACTIVARGANCHO);
		return false;
	}

	@Override
	public boolean keyUp(int keycode) {
		// TODO Auto-generated method stub
		if (keycode == Keys.LEFT)
			controladorxogo.liberarTecla(ControladorXogo.Keys.ESQUERDA);
		if (keycode == Keys.RIGHT)
			controladorxogo.liberarTecla(ControladorXogo.Keys.DEREITA);
		if (keycode == Keys.DOWN)
			controladorxogo.liberarTecla(ControladorXogo.Keys.ACTIVARGANCHO);
		return false;
	}
```

Exercicio: Agora teremos que facer o mesmo no caso do acelerómetro.

```
/**
 * Controla o acelerómetro para mover o pescador
 */
public void controlarAcelerometro(){

	if (!Gdx.input.isPeripheralAvailable(Peripheral.Accelerometer))
			return;

	int inclinacion = (int) Gdx.input.getAccelerometerY();

	if (inclinacion > 2)
		controladorxogo.pulsarTecla(ControladorXogo.Keys.DEREITA);
	if (inclinacion < -2)
		controladorxogo.pulsarTecla(ControladorXogo.Keys.ESQUERDA);
	if ((inclinacion>-2) && (inclinacion<2)){
		controladorxogo.liberarTecla(ControladorXogo.Keys.ESQUERDA);
		controladorxogo.liberarTecla(ControladorXogo.Keys.DEREITA);
	}

}

@Override
public void render(float delta) {
	// TODO Auto-generated method stub
	rendererxogo.render(delta);
	controladorxogo.update(delta);
	controlarAcelerometro();
}
```

Exercicio: Controlar agora que cando se toque á pantalla o gancho baixe.

Solución:

Arquivo PantallaXogo.java
Control do toque da pantalla.

```
@Override
public boolean touchDown(int screenX, int screenY, int pointer, int button) {
	// TODO Auto-generated method stub
	controladorxogo.pulsarTecla(ControladorXogo.Keys.ACTIVARGANCHO);
	return false;
}

@Override
public boolean touchUp(int screenX, int screenY, int pointer, int button) {
	// TODO Auto-generated method stub
	controladorxogo.liberarTecla(ControladorXogo.Keys.ACTIVARGANCHO);

	return false;
}
```

FACENDO O XOGO UN POUCO MÁIS COMPLICADO.

Como somos un pouco malvados e non queremos que o xogo sexa moi fácil de acabar imos a introducir a aceleración nos nosos peixes.

Podemos facer que a velocidade do peixe se incremente cunha aceleración.
Nota: a definimos na clase personaxe e o pescador imos darlle unha aceleración de 0.

Para isto definimos unha propiedade nova na clase Personaxe.
Arquivo Personaxe.java
Creamos a propiedade aceleracion.

```
	/**
	 * Aceleración dos personaxes
	 */
	protected float aceleracion;

	/**
	 * Instancia unha personaxe
	 * @param posicion
	 * @param tamaño
	 * @param velocidade_max
	 */
	public Personaxe(Vector2 posicion, Vector2 tamaño, int velocidade_max){
		this.posicion=posicion;
		this.tamaño=tamaño;
		this.aceleracion=0;
		this.velocidade_max=velocidade_max;
		rectangulo = new Rectangle(posicion.x, posicion.y, tamaño.x, tamaño.y);
	}

	/**
	 * Instancia unha personaxe
	 * @param posicion
	 * @param tamaño
	 * @param velocidade_max
	 * @param aceleracion
	 */
	public Personaxe(Vector2 posicion, Vector2 tamaño, int velocidade_max,float aceleracion){
		this(posicion,tamaño,velocidade_max);
		this.aceleracion = aceleracion;
	}
```

Agora temos que modificar o constructor da clase Peixe.

Arquivo Peixe.java
Modificamos o constructor para engadir a aceleración.
Modificamos o método update.

```
	public Peixe() {
		super(new Vector2(Mundo.TAMAÑO_MUNDO.x,Mundo.xerarAleatorio(1,100)), new
Vector2(Mundo.xerarAleatorio(8,30),0), Mundo.xerarAleatorio(30, 60));
		tamaño.y = tamaño.x;		// O peixe ten que ter o mesmo ancho que alto. Como o ancho o
xeramos aleatoriamente, unha vez o sabemos llo asinamos o y
		getRectangulo().setHeight(tamaño.y/2);  // O rectángulo se crea a partires do tamaño.
		getRectangulo().setWidth(tamaño.x/3);   // O rectángulo se crea a partires do tamaño

		pescado=false;

		int tipo = Mundo.xerarAleatorio(0, 3);
		switch(tipo){
			case 0:
```

```
                        tipopeixe = Tipos_peixe.LILA;
                        break;
                case 1:
                        tipopeixe = Tipos_peixe.AMARELO;
                        break;
                case 2:
                        tipopeixe = Tipos_peixe.AZUL;
                        break;

        }
        velocidade = velocidade_max;
        aceleracion = Mundo.xerarAleatorio(5, 10);
        if(Mundo.xerarAleatorio(0, 2)==1) {
                saedaesquerda=true;
                velocidade = -velocidade;
                aceleracion=-aceleracion;
                posicion.x = -tamaño.x;
        }
        else saedaesquerda=false;

}

@Override
public void update(float delta) {
        // TODO Auto-generated method stub
        velocidade += aceleracion*delta;
        posicion.x -= velocidade*delta;
        actualizarRectangulo();
}
```

Facendo pause e saír.

Normalmente (sobre todo nun dispositivo móbil) imos querer poder 'pausalo' para atender a unha chamada ou cambiar de aplicación.

O que imos facer primeiro é poñer unha icona de pausa na parte superior-dereita da pantalla.

Engadimos a icona de 'pause' e a da 'saír' ó noso xogo e xeramos o atlas modificado (os gráficos os tedes na solución do proxecto DVD (/Proxectos Eclipse/Proxecto 2D peixes/imaxes/entrada/). Copiamos o atlas modificado ó cartafol Assets da versión Android.

Agora na clase AssetsXogo imos ter que definir as textureRegion de cada un deles.

Exercicio: definir as propiedades 'exit' e 'pause' na clase assetsxogo e cargalas.
Solución:

```
/**
 * Textura de pause
 */
public static TextureRegion pause;
/**
 * Textura de sair
 */
public static TextureRegion sair;

public static void cargarTexturas(){
        texturas_todas = new TextureAtlas("imaxes/pescador_atlas.atlas");

        FileHandle imageFileHandle = Gdx.files.internal("imaxes/pecera1024x512.jpg");
        fondopecera = new Texture(imageFileHandle);

        pescador = texturas_todas.findRegion("pescador_sengancho");
        gancho = texturas_todas.findRegion("gancho");
        sedal = texturas_todas.findRegion("punto");

        sair = texturas_todas.findRegion("sair");
        pause = texturas_todas.findRegion("pause");
        ......................
```

Agora definiremos a posición e o tamaño de ditos gráficos por medio da clase Rectangle e o faremos na clase PantallaXogo.

Arquivo PantallaXogo.java
Definir dúas propiedades que indiquen a posición e tamaño das iconas Pause e Saír.

```
/**
 * Controlo PAUSE da pantalla de xogo
 */
public static final Rectangle CONTROL_PAUSE = new Rectangle(Mundo.TAMAÑO_MUNDO.x-80,
                                                            Mundo.TAMAÑO_MUNDO.y-20,20,20);
/**
 * Controlo SAIR da pantalla de xogo
 */
public static final Rectangle CONTROL_SAIR = new Rectangle(Mundo.TAMAÑO_MUNDO.x-55,
                                                           Mundo.TAMAÑO_MUNDO.y-20,20,20);
```

Agora debuxamos ditas texturas na clase RendererXogo.

Arquivo RendererXogo.java
Debuxamos os controis de pause e saír.

```
/**
 * Dibuxa as controis do xogo coma Pause e Sair
 */
private void dibuxarControis(){
        spritebatch.draw(AssetsXogo.pause, PantallaXogo.CONTROL_PAUSE.x,PantallaXogo.CONTROL_PAUSE.y,
                        PantallaXogo.CONTROL_PAUSE.width,PantallaXogo.CONTROL_PAUSE.height);
        spritebatch.draw(AssetsXogo.sair, PantallaXogo.CONTROL_SAIR.x,PantallaXogo.CONTROL_SAIR.y,
                        PantallaXogo.CONTROL_SAIR.width,PantallaXogo.CONTROL_SAIR.height);
}

public void render(float delta){
        delta = Gdx.graphics.getDeltaTime();
        stateTime += delta;

        spritebatch.begin();

        dibuxarFondo();
        dibuxarPescador();
        dibuxarGancho();
        dibuxarSedal();
        dibuxarPeixes(delta);
        dibuxarNumPeixesCollidos();

        dibuxarCronometro();

        dibuxarControis();

        spritebatch.end();
}
```

Agora temos que implementar o código necesario.

No caso de saír vai ser sinxelo, xa que só temos que controlar cando o dedo preme a icona e chamar o método Gdx.app.exit.

O método que temos que modificar é o touchDown da clase PantallaXogo.
Se vos fixades dito método ten entre os seus parámetros x,y que se corresponde coas coordenadas x,y da nosa resolución de pantalla.

É dicir, cando lanzamos o xogo na versión Desktop vemos que estamos executando a nosa aplicación unha resolución de 800x480:

```
        cfg.width = 800;
        cfg.height = 480;
```

Cando eu preme a pantalla darame as coordenadas dentro desta resolución.

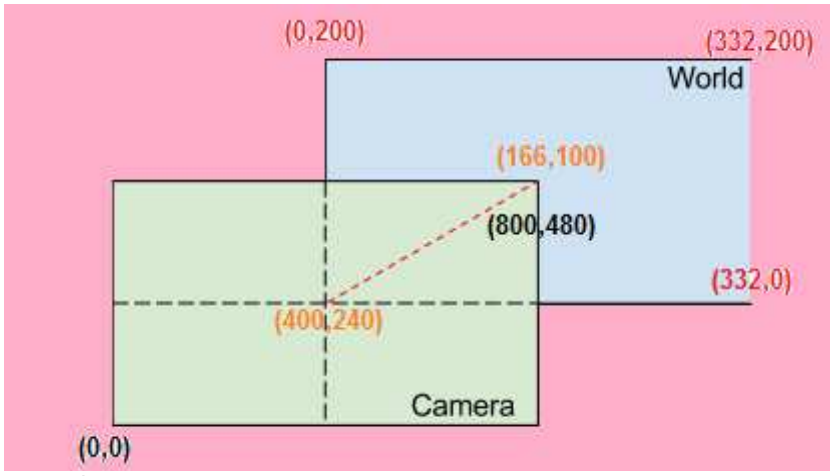
Pero por outra banda, cando definimos o noso mundo establecemos unha resolución de 332xs200:

```
public final static Vector2 TAMAÑO_MUNDO = new Vector2(332,200);
```

E para facer que a cámara debuxase todo neste sistema de coordenadas fixemos isto na clase RendererXogo:

```
camaraortho.setToOrtho(false,Mundo.TAMAÑO_MUNDO.x,Mundo.TAMAÑO_MUNDO.y);
camaraortho.update();

spritebatch.setProjectionMatrix(camaraortho.combined);
```

Agora temos que pasar das coordenadas reais as coordenadas do noso Mundo para así poder controlar se prememos sobre a icona de saír ou pause (que están definidos no sistema de coordenadas do noso mundo).



Nota: en negro as coordenadas da cámara; en vermello as coordenadas do mundo e en laranxa o centro da pantalla nos dous sistemas de coordenadas.

Para obter as coordenadas do noso mundo a partires da coordenadas da pantalla temos que facer uso do método 'unproyect' do obxecto cámara.

Dentro da clase OrthographicCamera dispoñemos dos métodos:

- unproyect: pasa do sistema de coordenadas reais ó sistema de coordenadas da nosa cámara.
- proyect: pasa do sistema de coordenadas da nosa cámara ó sistema de coordenadas reais.

Polo tanto necesitamos acceder ó obxecto cámara dende a clase PantallaXogo.
Cambiamos a propiedade camaraortho da clase RendererXogo.

Arquivo RendererXogo.java
Modificamos a propiedade camaraortho a static que sexa accedida dende a clase PantallaXogo.

Pasamos de:

```
private OrthographicCamera camaraortho;
```

A:

```
public static OrthographicCamera camaraortho;
```

Agora temos que facer o proceso de paso de sistemas de coordenadas.
Tanto en 3D como 2D o proceso é o mesmo. O método unproyect espera recibir 3 datos (x,y,z) polo que temos que facer uso dun Vector3 poñendo como valor 0 a coordenada Z.

```
Vector3 coord = new Vector3(screenX,screenY,0);
RendererXogo.camaraortho.unproject(coord);      // Pasa a coordenadas do noso mundo
```

A propiedade coord se modifica ó chamar ó método unproyect.

Podemos facer un círculo que representará o punto onde se premeu co dedo.

```
Circle pulsa = new Circle(coord.x,coord.y,5f);
```

Agora só temos que usar a clase Intersector para ver se 'chocan' o círculo e o rectángulo que conforman os controis.

```
if (Intersector.overlapCircleRectangle(pulsa, CONTROL_SAIR)){
        Gdx.app.exit();
}
```

Todo xunto:

Arquivo PantallaXogo.java
Implementamos saír no xogo.

```
@Override
public boolean touchDown(int screenX, int screenY, int pointer, int button) {
        // TODO Auto-generated method stub
        Vector3 coord = new Vector3(screenX,screenY,0);
        RendererXogo.camaraortho.unproject(coord); // Pasa a coordenadas do noso mundo e as garda en
coord

        Circle pulsa = new Circle(coord.x,coord.y,5f);  // Poñemos a mitade do tamaño dos controis
        if (Intersector.overlapCircleRectangle(pulsa, CONTROL_SAIR)){
                Gdx.app.exit();
        }


        controladorxogo.pulsarTecla(ControladorXogo.Keys.ACTIVARGANCHO);
        return false;
}
```

Agora imos implementar a pausa. Imos facer que cando o usuario preme o botón de pause apareza no centro da pantalla o mesmo botón (máis grande) para saír do estado de pause e seguir xogando. O que temos que facer é non chamar á clase ControladorXogo cando estea no estado de pause. Como ademais imos ter que debuxar a icona de pause en grande imos definir o seu rectángulo e imos crear unha propiedade de clase (static) que informe cando o xogo está en pause para facer que a clase RendererXogo dibuxe a icona.

Arquivo PantallaXogo.java
Implementamos pause no xogo.

```
/**
 * Indica se o xogo está en pause
 */
public static boolean pause;

/**
 * Controlo PAUSE da pantalla de xogo no centro en grande
 */
public static final Rectangle CONTROL_PAUSE_GRANDE = new Rectangle(Mundo.TAMAÑO_MUNDO.x/2-15,
                                                                   Mundo.TAMAÑO_MUNDO.y/2,30,30);

public PantallaXogo(Principal principal){
        game=principal;
```

```
                pause = false;
        }

        @Override
        public void render(float delta) {
                // TODO Auto-generated method stub
                rendererxogo.render(delta);
                if (!pause) {
                        controladorxogo.update(delta);
                        controlarAcelerometro();
                }
        }

        @Override
        public boolean touchDown(int screenX, int screenY, int pointer, int button) {
                // TODO Auto-generated method stub
                Vector3 coord = new Vector3(screenX,screenY,0);
                RendererXogo.camaraortho.unproject(coord);      // Pasa a coordenadas do noso mundo e as
garda en coord

                Circle pulsa = new Circle(coord.x,coord.y,5f);  // Poñemos a mitade do tamaño dos controis
                if (Intersector.overlapCircleRectangle(pulsa, CONTROL_SAIR)){
                        Gdx.app.exit();
                }

                if (Intersector.overlapCircleRectangle(pulsa, CONTROL_PAUSE)){
                        pause = true;
                        return false;   // Para que non baixe o gancho
                }
                if (Intersector.overlapCircleRectangle(pulsa, CONTROL_PAUSE_GRANDE)){
                        pause = false;
                        return false;   // Para que non baixe o gancho
                }

                controladorxogo.pulsarTecla(ControladorXogo.Keys.ACTIVARGANCHO);
                return false;
        }
```

Arquivo RendererXogo.java

Modificamos para que debuxe a icona de Pause en grande.

```
        /**
         * Dibuxa as controis do xogo coma Pause e Sair
         */
        private void dibuxarControis(){
                spritebatch.draw(AssetsXogo.pause,
PantallaXogo.CONTROL_PAUSE.x,PantallaXogo.CONTROL_PAUSE.y,PantallaXogo.CONTROL_PAUSE.width,PantallaXogo.CONTR
OL_PAUSE.height);
                spritebatch.draw(AssetsXogo.sair,
PantallaXogo.CONTROL_SAIR.x,PantallaXogo.CONTROL_SAIR.y,PantallaXogo.CONTROL_SAIR.width,PantallaXogo.CONTROL_
SAIR.height);

                if (PantallaXogo.pause){
                        spritebatch.draw(AssetsXogo.pause, PantallaXogo.CONTROL_PAUSE_GRANDE.x,
                                PantallaXogo.CONTROL_PAUSE_GRANDE.y,PantallaXogo.CONTROL_PAUSE_GRANDE.width,
                                PantallaXogo.CONTROL_PAUSE_GRANDE.height);
                }
        }
```

NOTA: poderíamos facer unha pantalla de Pause e que fora chamada cando prememos a icona de pause do xogo, pero o manexo de varias pantallas o imos ver a continuación.

Imos facer unha modificación máis xa que se vos fixades o cronómetro non para cando estamos en Pause.
Isto é debido a que o cronómetro o estamos 'calculando' no render que seguimos chamando nese

estado.

Para solucionalo imos crear unha nova propiedade na clase Mundo de tipo float que leverá o tempo transcurrido do noso xogo. A faremos static e será actualizada na clase controladora (método update) e debuxada na clase renderer.

Exercicio: facer o anterior.
Solución:

Arquivo Mundo.java
Definimos e inicializamos a propiedade 'cronometro'

```
	/**
	 * Leva o tempo transcurrido do xogo. Usada como cronómetro
	 */
	public static float cronometro;
	public Mundo(){
		pescador = new Pescador();
		gancho = new Gancho();
		peixes = new Array<Peixe>();
		cronometro=0;
	}
```

Arquivo ControladorXogo.java
Actualizamos o cronómetro.

```
	public void update(float delta){

		pescador.update(delta);
		actualizarGancho(delta);
		actualizarPeixes(delta);
		crearPeixes();

		Mundo.cronometro += delta;

		controlarPescador();
		controlarGancho();
		.....
```

Arquivo RendererXogo.java
Dibuxamos o cronómetro.

```
	private void dibuxarCronometro(){
		stringbuilder.setLength(0);	// Borramos o strinbuilder
		AssetsXogo.bitmapfont.draw(spritebatch, stringbuilder.append((int)(Mundo.NUM_SEGUNDOS_XOGO-
Mundo.cronometro)), Mundo.TAMAÑO_MUNDO.x-30, Mundo.TAMAÑO_MUNDO.y);

	}
```

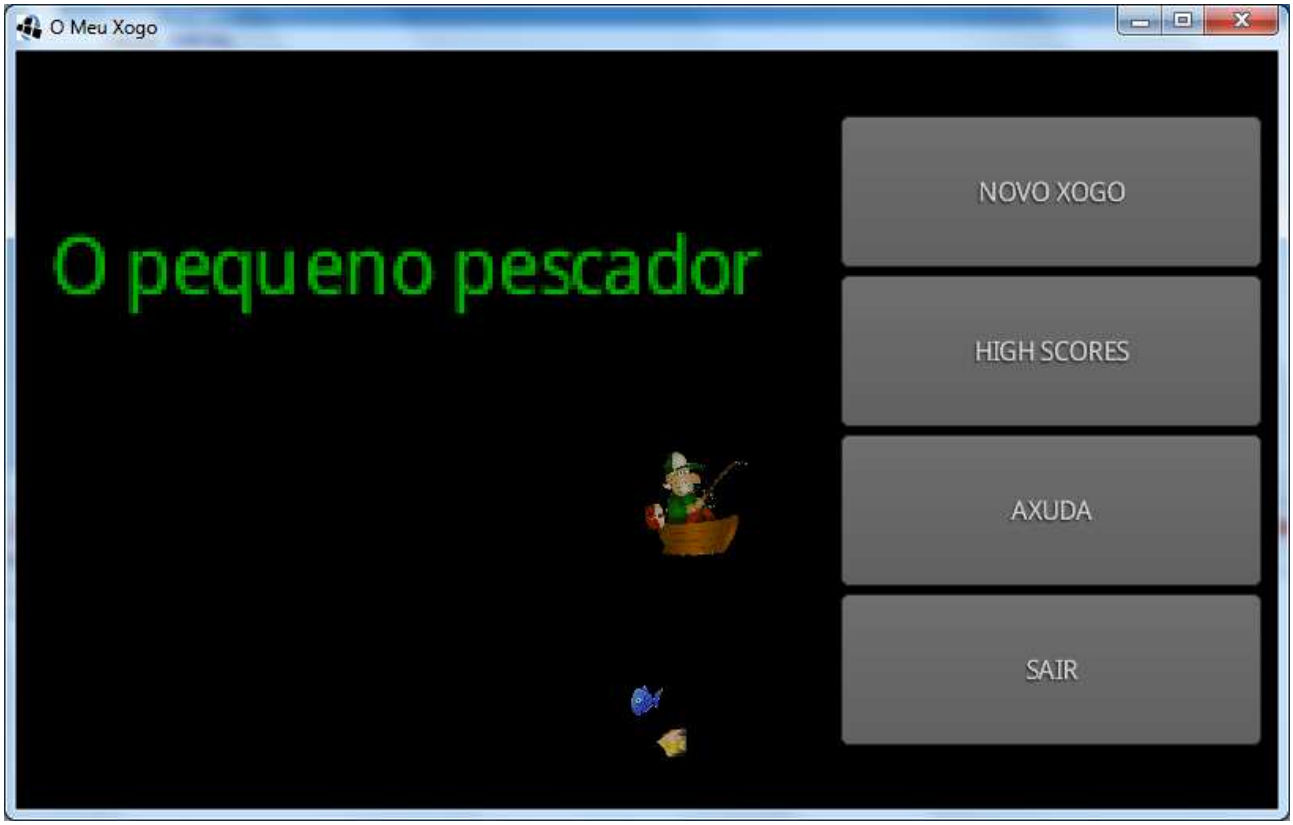
Ademais imos facer que cando saiamos da pantalla do xogo en Android premendo o botón de 'casa' este quede en estado de pause.

Arquivo PantallaXogo.java
Facemos que cando deixemos a pantalla do xogo este quede en estado de pause.

```
	@Override
	public void pause() {
		// TODO Auto-generated method stub
		Gdx.input.setInputProcessor(null);
		pause = true;
	}
```

Creando novas pantallas.

- Pantalla Inicial.



Nesta pantalla imos facer uso doutro recurso do motor de xogos que é o Scene2d-ui (scene 2d user interface).

Este paquete incorpora moitas clases típicas dunha interface (botóns, cadros de texto,....).
Para poder andar con estes elementos é necesario ter un estilo definido para cada un deles. O estilo o podemos xerar por programación, pero o máis normal é telo nun arquivo externo coa extensión json.

Podedes consultar acerca do scene2d ui neste enlace:
http://code.google.com/p/libgdx/wiki/scene2dui

Como punto de partido imos a utilizar o arquivo json xunto cos bmp, atlas e fonte necesarios que veñen nos exemplos do motor de xogos:
https://github.com/libgdx/libgdx/tree/master/tests/gdx-tests-android/assets/data
=> Arquivo uiskin.atlas, uiskin.png,uiskin.json e default.fnt.

O que imos facer con estes arquivos é crear un estilo (parecido o CSS de HTML ou os arquivos skin de ASP.NET).

Se editamos o arquivo json podemos ver entre outras cousas:

```
com.badlogic.gdx.graphics.g2d.BitmapFont: { default-font: { file: default.fnt } },
com.badlogic.gdx.graphics.Color: {
        green: { a: 1, b: 0, g: 1, r: 0 },
        white: { a: 1, b: 1, g: 1, r: 1 },
        red: { a: 1, b: 0, g: 0, r: 1 },
        black: { a: 1, b: 0, g: 0, r: 0 }
},
com.badlogic.gdx.scenes.scene2d.ui.Skin$TintedDrawable: {
        dialogDim: { name: white, color: { r: 0, g: 0, b: 0, a: 0.45 } }
},
com.badlogic.gdx.scenes.scene2d.ui.Button$ButtonStyle: {
        default: { down: default-round-down, up: default-round },
        toggle: { down: default-round-down, checked: default-round-down, up: default-round }
},
.....................
```

Nesta liña:

```
com.badlogic.gdx.graphics.g2d.BitmapFont: { default-font: { file: default.fnt } },
```

indicamos o tipo de fonte de texto que imos a usar (default.fnt) nos elementos gráficos (label´s, botóns con texto,...)

Aquí definimos constantes de cores:

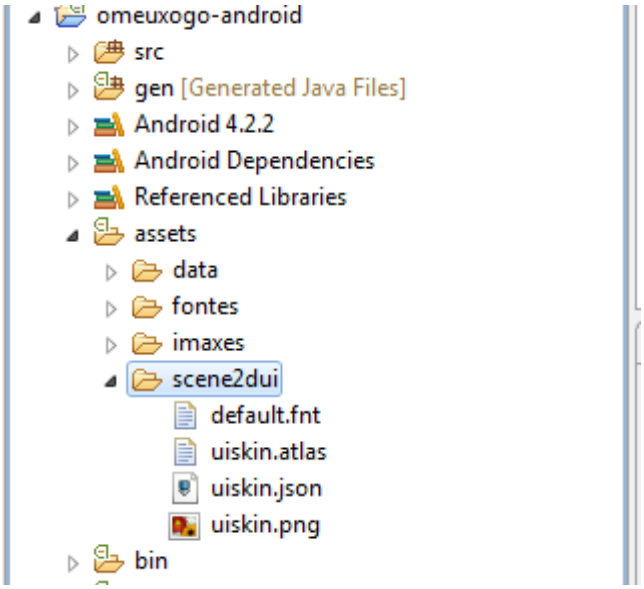
```
com.badlogic.gdx.graphics.Color: {
        green: { a: 1, b: 0, g: 1, r: 0 },
        white: { a: 1, b: 1, g: 1, r: 1 },
        red: { a: 1, b: 0, g: 0, r: 1 },
        black: { a: 1, b: 0, g: 0, r: 0 }
},
```

Aquí definimos o estilo dos botóns:

```
com.badlogic.gdx.scenes.scene2d.ui.Button$ButtonStyle: {
        default: { down: default-round-down, up: default-round },
        toggle: { down: default-round-down, checked: default-round-down, up: default-round }
},
```

Por defecto cando prememos sobre un, poñerá o gráfico 'default-round-down' e o soltalo 'default-round'. Ditos gráficos os podedes atopar no atlas.

Imos a copiar todos estes arquivos (están no proxecto solución do DVD /Proxectos Eclipse/Proxecto 2D peixes/stage2dui/) a unha cartafol nova dentro da versión Android do noso xogo, de nome stage2dui:



Agora imos crear unha nova pantalla que implemente a interface Screen no paquete 'pantallas' de nome MenuXogo.java.

Arquivo MenuXogo.java

Creamos o arquivo.

Como dende esta pantalla imos ter que cambiar de pantalla (ó premer novo xogo por exemplo) temos que ter unha referencia á clase Principal para chamar o método setScreen.

```
public class MenuXogo  implements Screen {

	private Principal game;

	public MenuXogo(Principal game){
		this.game=game;
	}
	@Override
	public void render(float delta) {
		// TODO Auto-generated method stub

	}

	@Override
	public void resize(int width, int height) {
		// TODO Auto-generated method stub

	}

	@Override
	public void show() {
		// TODO Auto-generated method stub

	}

	@Override
	public void hide() {
		// TODO Auto-generated method stub
```

```
	}

	@Override
	public void pause() {
		// TODO Auto-generated method stub

	}

	@Override
	public void resume() {
		// TODO Auto-generated method stub

	}

	@Override
	public void dispose() {
		// TODO Auto-generated method stub

	}

}
```

Non imos programar seguindo o MVC xa que co scene2d xa veremos que non é posible (mezcla modelo e vista), e ademais tampouco vai ter unha complexidade tan grande a pantalla.

Primeiro imos cargar os arquivos necesarios para esta pantalla.
Creamos un método cargarAssets que o faga. Imos utilizar a clase AssetManager (xa comentada anteriormente) e a clase Skin que lerá os datos do arquivo json e será o estilo a usar polos compoñentes gráficos.

Arquivo MenuXogo.java

```
	AssetManager amanager;
	Skin skin;

	/** Carga os assets necesarios para esta pantalla.
	 *
	 */
	private void cargarAssets() {
		Gdx.app.log("LIBGDX","CARGA TEXTURAS");
		amanager = new AssetManager();
		amanager.load("scene2dui/uiskin.atlas",TextureAtlas.class);
		amanager.finishLoading();
		TextureAtlas atlas = amanager.get("scene2dui/uiskin.atlas", TextureAtlas.class);
		skin = new Skin(Gdx.files.internal("scene2dui/uiskin.json"), atlas);  // Cargamos os estilos

		AssetsXogo.cargarTexturas();


	}
```

Como vemos cargamos os assets do xogo (`AssetsXogo.cargarTexturas();`). Xa que os cargamos en varias pantallas (MenuXogo e PrincipalXogo) podemos levar a carga das texturas no método create da clase Game e a liberación no método dispose da mesma clase.

Arquivo Principal.java

```
	@Override
	public void create() {
		// TODO Auto-generated method stub
		AssetsXogo.cargarTexturas();

		pantallaxogo = new PantallaXogo(this);
		menuxogo=new MenuXogo(this);
```

```
                setScreen(menuxogo);
        }

        @Override
        public void dispose(){
                super.dispose();
                menuxogo.dispose();
                pantallaxogo.dispose();
                AssetsXogo.liberarTexturas();
        }
```

Arquivo PantallaXogo.java

Comentamos as liñas que cargan as texturas e as liberan.

```
        @Override
        public void show() {
                // TODO Auto-generated method stub
//              AssetsXogo.cargarTexturas();

                mundo = new Mundo();
                controladorxogo = new ControladorXogo(mundo, this);
                rendererxogo = new RendererXogo(mundo);

                Gdx.input.setInputProcessor(this);
        }
        @Override
        public void dispose() {
                // TODO Auto-generated method stub
                Gdx.input.setInputProcessor(null);
                if (rendererxogo!=null)
                        rendererxogo.dispose();
                if (controladorxogo!=null)
                        controladorxogo.dispose();
//              AssetsXogo.liberarTexturas();
        }
```

Volvendo á pantalla de menú....comentamos ou quitamos a liña que carga os AssetsXogo.

Agora no método show cargamos os assets específicos desta pantalla.
Modificamos o método dispose que libere todas as texturas cargadas.

Arquivo MenuXogo.java

```
        @Override
        public void show() {
                // TODO Auto-generated method stub
                cargarAssets();

        }
        @Override
        public void dispose() {
                // TODO Auto-generated method stub
                if (skin!=null) skin.dispose();
                if (amanager!=null)     amanager.dispose();
        }
```

Unha vez cargados os elementos gráficos entramos no Scene2D.
Neste paquete a clase que necesitamos para debuxar é **Stage**. Dentro do stage existen '**Actors**' que son os elementos que se debuxan. Cada actor ten unha textura, posición, tamaño....

Nota: máis conceptos en: http://code.google.com/p/libgdx/wiki/scene2d

Por exemplo, imos facer que o stage estea composto por un label cun texto.

Arquivo MenuXogo.java
Definimos propiedade da clase Stage.
Engadimos un actor Label o stage.
Definimos un método presentarPantalla() que cargue o label no stage e que sexa chamado dende o método show.

Arquivo MenuXogo.java

```
Stage stage;

public MenuXogo(Principal game){
        this.game=game;
        stage = new Stage();

        cargarAssets();
        presentarPantalla();
}
/**
 * Engade todo o necesario á pantalla de menú
 */
private void presentarPantalla(){

        // TITULO
        Label titulo = new Label("O pescador",skin);
        titulo.setColor(Color.GREEN);
        titulo.setFontScale(2);
        titulo.setBounds(20, 0, stage.getWidth()/2,stage.getHeight()-100);
        titulo.setAlignment(Align.top | Align.left);

        stage.addActor(titulo);
}
```

Agora para facer que o stage debuxe todos os actors que o compoñen hai que modificar o método render.

Arquivo MenuXogo.java

```
@Override
public void render(float delta) {
        // TODO Auto-generated method stub
        Gdx.gl.glClear(GL11.GL_COLOR_BUFFER_BIT);

      stage.act(Gdx.graphics.getDeltaTime());
      stage.draw();

}
```

e tamén modificamos o método resize para que o stage teña sempre a resolución do dispositivo como tamaño para debuxar mantendo a relación de aspecto.

```
@Override
public void resize(int width, int height) {
        // TODO Auto-generated method stub
        stage.setViewport(width, height, true);
}
```

Nota: podedes probar despois como poñendo a false e diminuíndo o ancho da pantalla na versión desktop como se estreita o texto.

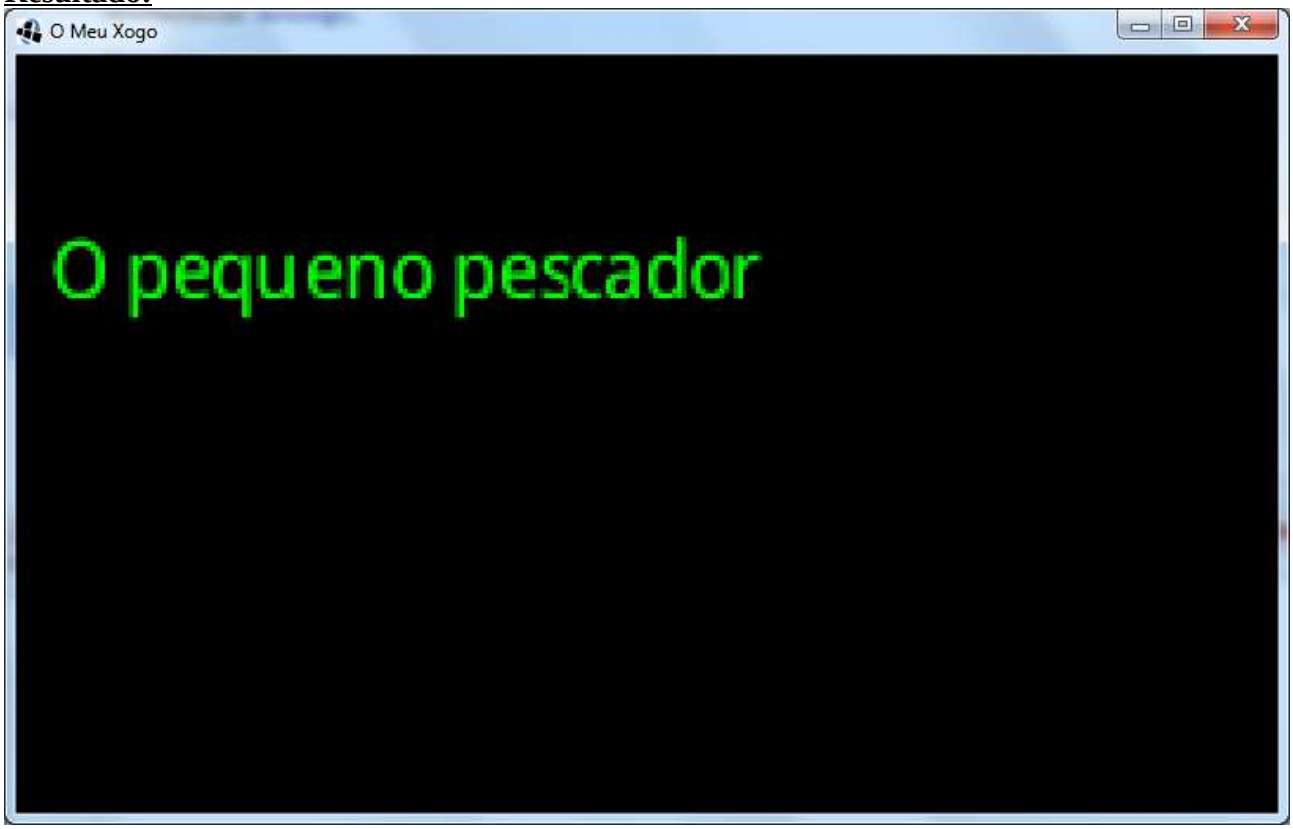
Para poder facer unha proba imos modificar a clase Principal.

Arquivo Principal.java

Creamos unha propiedade MenuXogo, a instanciamos e facemos que sexa o screen inicial. Chamamos o método dispose cando se peche o xogo.

```
public class Principal extends  Game {

	PantallaXogo pantallaxogo;
	MenuXogo menuxogo;

	@Override
	public void create() {
		// TODO Auto-generated method stub
		AssetsXogo.cargarTexturas();

		pantallaxogo = new PantallaXogo(this);
		menuxogo=new MenuXogo(this);
		setScreen(menuxogo);
	}

	public MenuXogo getMenuxogo() {
		return menuxogo;
	}

	public PantallaXogo getPantallaxogo() {
		return pantallaxogo;
	}

	@Override
	public void dispose(){
		super.dispose();
		menuxogo.dispose();
		pantallaxogo.dispose();
		AssetsXogo.liberarTexturas();
	}
}
```

Resultado:



Accións:

O uso da clase Stage nos vai permitir facer diferentes accións sobre os actors que o compoñen.

Así temos:

3 tipos de accións:

1. Accións animadas.
2. Accións compostas.
3. Outras accións.

As accións animadas modifican varias propiedades dos actor´s, coma a posición, rotación, escala e factor alfa. Son as seguintes.

- FadeIn – modifica o factor alfa do actor do valor que teña ó valor 1.
- FadeOut - modifica o factor alfa do actor do valor que teña ó valor 0.
- FadeTo - modifica o factor alfa do actor do valor que teña ó valor especificado.
- MoveBy – move o actor unha distancia especificada.
- MoveTo – move o actor a unha posición específica.
- RotateBy – rota o actor engadindo ó ángulo especificado.
- RotateTo – rota o actor ata un ángulo especificado.
- ScaleTo – escala o actor ó valor indicado.

Accións compostas combinan múltiple accións en unha:

- Parallel – executa todas as accións en paralelo. Todas á vez.
- Sequence – executa todas as accións en secuencia. Unha detrás de outra.

Outras accións:

- Repeat – repite a acción n-veces.
- Forever – repite a acción indefinidamente.
- Delay – retrasa a execución dunha acción o tempo especificado.
- Remove – elimina o Actor especificado do Stage.

Máis información en: https://code.google.com/p/libgdx/wiki/scene2d#Actions

No noso caso imos facer que a pantalla principal apareza cun efecto de fadeIn que dure 4 segundos. Inicialmente poderíamos facer isto no final do procedemento presentarPantalla():

```
stage.addAction(Actions.fadeIn(4));
```

pero se probamos podemos ver que non pasa nada. Isto é debido a que por defecto o texto se ve. Necesitamos que o texto apareza oculto e despois faga o efecto de aparecer.
O poderíamos arranxar así:

```
        stage.addAction(Actions.fadeOut(0));
        stage.addAction(Actions.fadeIn(4));
```

Pero podemos facelo doutra forma máis elegante, xa que a acción pode estar composta por moitas accións as cales se poden executar en paralelo, secuencial, ... como vimos antes.

No noso caso quedaría así:

Arquivo MenuXogo.java

```
    private void presentarPantalla(){

        // TITULO
        Label titulo = new Label("O pescador",skin);
        titulo.setColor(Color.GREEN);
        titulo.setFontScale(2);
        titulo.setBounds(20, 0, stage.getWidth()/2,stage.getHeight()-100);
        titulo.setAlignment(Align.top | Align.left);

        stage.addActor(titulo);

        stage.addAction(Actions.sequence(Actions.fadeOut(0), Actions.fadeIn(4)));
    }
```

Agora imos colocar a 4 botóns na parte dereita da pantalla. Para facelo imos utilizar outro actor que é un obxecto da clase Table á cal imos engadirlle os catro botóns.

Máis información:

http://code.google.com/p/libgdx/wiki/scene2dui
https://code.google.com/p/table-layout/

```
    private void presentarPantalla(){

        // TITULO
        Label titulo = new Label("O pescador",skin);
        titulo.setColor(Color.GREEN);
        titulo.setFontScale(2);
        titulo.setBounds(20, 0, stage.getWidth()/2,stage.getHeight()-100);
        titulo.setAlignment(Align.top | Align.left);

        stage.addActor(titulo);


        // TABOA E BOTONS
        Table tabla = new Table(skin);
        tabla.setFillParent(true);
        tabla.defaults().space(5);      // Espazo por defecto arredor dos compoñenetes
        tabla.defaults().fillX();       // o elemento (label,botón,..) ocupa todo o ancho => centrar

        tabla.align(Align.right);
        tabla.padRight(10);

        float ancho = stage.getWidth()/3;       // Repartimos o ancho e alto da pantalla
        float alto = stage.getHeight()/5;


        TextButton novoxogo = new TextButton("NOVO XOGO", skin);
        tabla.add(novoxogo).height(alto).width(ancho);
        tabla.row();
        TextButton highscores = new TextButton("HIGH SCORES", skin);
        tabla.add(highscores).height(alto).width(ancho);
        tabla.row();
        TextButton axuda = new TextButton("AXUDA", skin);
```

```
        tabla.add(axuda).height(alto).width(ancho);
        tabla.row();
        TextButton sair = new TextButton("SAIR", skin);
        tabla.add(sair).height(alto).width(ancho);

        stage.addActor(tabla);

        stage.addAction(Actions.sequence(Actions.fadeOut(0), Actions.fadeIn(4)));

    }
```

Nota: ós propios actores poden ter accións asociadas, chamando o método addAction de cada un deles.

Nota: para non ter que estar escribindo continuamente Actions.acción podemos facer este import:

```
import static com.badlogic.gdx.scenes.scene2d.actions.Actions.*;
```

Para darlle un pouco máis de vidilla a pantalla inicial imos facer que un pescador e uns peixes se movan na parte inferior esquerda:

```
    private void presentarPantalla(){

        // TITULO
        Label titulo = new Label("O pescador",skin);
        titulo.setColor(Color.GREEN);
        titulo.setFontScale(2);
        titulo.setBounds(20, 0, stage.getWidth()/2,stage.getHeight()-100);
        titulo.setAlignment(Align.top | Align.left);

        stage.addActor(titulo);


        // TABOA E BOTONS
        Table tabla = new Table(skin);
        tabla.setFillParent(true);
        tabla.defaults().space(5);
        tabla.defaults().fillX();

        tabla.align(Align.right);
        tabla.padRight(10);

//      tabla.debugTable();             // Se queremos ver as li?as que conforman a t?boa

        float ancho = stage.getWidth()/3;       // Repartimos o ancho e alto da pantalla
        float alto = stage.getHeight()/5;


        TextButton novoxogo = new TextButton("NOVO XOGO", skin);
        tabla.add(novoxogo).height(alto).width(ancho);
        tabla.row();
        TextButton highscores = new TextButton("HIGH SCORES", skin);
        tabla.add(highscores).height(alto).width(ancho);
        tabla.row();
        TextButton axuda = new TextButton("AXUDA", skin);
        tabla.add(axuda).height(alto).width(ancho);
        tabla.row();
        TextButton sair = new TextButton("SAIR", skin);
        tabla.add(sair).height(alto).width(ancho);

        stage.addActor(tabla);

        // PESCADOR E PEIXES MOVENDOSE
        Image pescador = new Image(AssetsXogo.pescador);
```

```
                float desplazamentoX=stage.getWidth()-ancho-80;
                pescador.setBounds(10, stage.getHeight()/3,70,70);
                Action accionPescador = forever(Actions.sequence(moveBy(desplazamentoX,0,4),moveBy(-
desplazamentoX,0,4)));
                pescador.addAction(accionPescador);
                stage.addActor(pescador);

                Image peixe1 = new Image(AssetsXogo.peixe_azul);
                peixe1.setBounds(0, stage.getHeight()/5,20,20);
                Image peixe2 = new Image(AssetsXogo.peixe_amarelo.getKeyFrame(0));
                peixe2.setBounds(0, 30,20,20);

                float altura1 = Mundo.xerarAleatorio((int)20, (int)stage.getHeight()/5);
                float altura2 = Mundo.xerarAleatorio((int)20, (int)stage.getHeight()/5);

                Action accionPeixe1 =
forever(Actions.sequence(rotateBy(180),moveTo(desplazamentoX,altura1,3),rotateBy(180),moveTo(-
20,altura2,3)));
                Action accionPeixe2 =
forever(Actions.sequence(rotateBy(180),moveTo(desplazamentoX,altura2,3),rotateBy(180),moveTo(-
20,altura1,4)));

                peixe1.addAction(accionPeixe1);
                peixe2.addAction(accionPeixe2);


                stage.addActor(peixe1);
                stage.addActor(peixe2);

                stage.addAction(Actions.sequence(Actions.fadeOut(0), Actions.fadeIn(4)));

        }
```

Agora necesitamos facer que cando se preme o botón faga algo.
Primeiro temos que informar ó motor que os eventos os vai xestionar o Stage.

Arquivo MenuXogo.java

```
        @Override
        public void show() {
                // TODO Auto-generated method stub
                Gdx.input.setInputProcessor(stage);
        }
        @Override
        public void pause() {
                // TODO Auto-generated method stub
                Gdx.input.setInputProcessor(null);
        }

        @Override
        public void resume() {
                // TODO Auto-generated method stub
                Gdx.input.setInputProcessor(stage);
        }
        @Override
        public void dispose() {
                // TODO Auto-generated method stub
                Gdx.input.setInputProcessor(null);
                if (skin!=null) skin.dispose();
                if (amanager!=null)     amanager.dispose();
        }
```

E agora temos que implementar o código necesario cando se faga click sobre o botón:

Método presentarPantalla():

```
.....
        TextButton novoxogo = new TextButton("NOVO XOGO", skin);
        tabla.add(novoxogo).height(alto).width(ancho);
        tabla.row();
        TextButton highscores = new TextButton("HIGH SCORES", skin);
        tabla.add(highscores).height(alto).width(ancho);
        tabla.row();
        TextButton axuda = new TextButton("AXUDA", skin);
        tabla.add(axuda).height(alto).width(ancho);
        tabla.row();
        TextButton sair = new TextButton("SAIR", skin);
        tabla.add(sair).height(alto).width(ancho);

        sair.addListener(new ClickListener(){
                public void clicked (InputEvent event, float x, float y) {
                        Gdx.app.exit();
                }
        });
        ..................................
```

Exercicio: facer que cando se prema 'Novo xogo' se inicie o xogo.

Solución:

```
        novoxogo.addListener(new ClickListener(){
                public void clicked (InputEvent event, float x, float y) {
                        game.setScreen(game.getPantallaxogo());
                }
        });
```

Mellora:
Imos facer que cando saímos pida confirmación nunha caixa de diálogo:

Arquivo MenuXogo.java
Modificamos o método presentarPantalla:

```
        sair.addListener(new InputListener(){
                public boolean touchDown (InputEvent event, float x, float y, int pointer, int button) {
                        preguntarSair();
                        return true;
                }
        });
```

Creamos o método preguntarSair():

```
        /**
         * Lanza o cadro de diálogo de saír.
         */
        private void preguntarSair(){

                new Dialog("Sair do xogo", skin, "dialog") {    // As tildes non aparecen xa que non están no
png das letras
                        protected void result (Object object) {
                                if ((Boolean) object)
                                        Gdx.app.exit();
                        }
                        }.text("Estás seguro de querer sair ?").button("Yes", true).button("No",
false).key(Keys.ENTER, true).key(Keys.ESCAPE, false).show(stage);
        }
```

Mellora:
Imos facer que cando o usuario preme na 'frecha atrás' de Android, pregunte se quere saír do xogo.

```
public MenuXogo(Principal game){
        Gdx.input.setCatchBackKey(true);
        this.game=game;
        stage = new Stage() {
                @Override
                public boolean keyDown(int keyCode) {
                    if (keyCode == Keys.BACK) {
                        preguntarSair();
                    }
                    return super.keyDown(keyCode);
                }
        };
        cargarAssets();
        presentarPantalla();
}
```

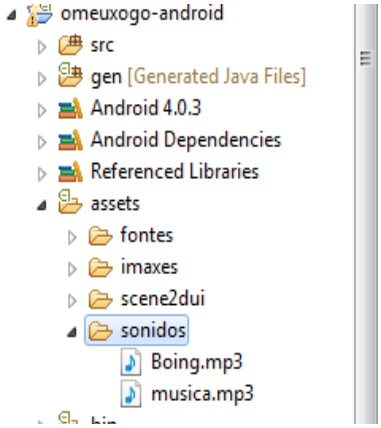
MUSICA:

Agora imos facer que o xogo comece cunha música de fondo e que se poda quitar premendo nunha icona.

Podemos descargar a música dende: http://dig.ccmixter.org/free_music
pero podedes atopar outros sitios web de música gratuíta.
Os efectos de son podemos atopalos en http://www.soundboard.com/category/Sound-Effects

No noso xogo descargamos esta música: *The Long Goodbye* by John Pazdan e un efecto de son (Boing.mp3).

Primeiro deberemos copiar o arquivo de música e o efecto de son (o podedes atopar no DVD /Proxectos Eclipse/Proxecto 2D peixes/sonidos) ó cartafol Assets da versión Android. Imos crear un cartafol de nome 'sonidos'.



Nota: debedes controlar o tamaño do arquivo de música xa que moitas veces podemos baixar a calidade de audio para que ocupe menos.
Se queredes editar un son/música podedes usar o programa AUDICITY:

http://audacity.sourceforge.net/?lang=es

Agora debemos de 'cargar' a música e o son, pero lembrade que no caso de cargar moitos elementos gráficos e de son será conveniente utilizar a clase AssetManager como fixemos na pantalla de menú.

Creamos por tanto unha clase Sonidos que estea no paquete principal. No constructor cargaremos todos os efectos de son e música.

Para isto teremos que utilizar a clase Music para a música e Sound para os efectos de son.
Definiremos todas as propiedades como public static para poder acceder dende o resto de clases.
Para cargalas só temos que facer uso dos métodos newSound e newMusic do paquete Gdx.audio.

Arquivo Sonidos.java
Carga os efectos de son e música do xogo.
Creamos os método para pausar e reanudar a música de fondo.
Creamos o método para liberar os recursos.

```
public class Sonidos {

	public static Sound collepeixe;
	private static Music musicafondo;

	public static void inicializarSonidos(){
		collepeixe = Gdx.audio.newSound(Gdx.files.internal("sonidos/Boing.mp3"));
		musicafondo = Gdx.audio.newMusic(Gdx.files.internal("sonidos/musica.mp3"));
		musicafondo.setVolume(0.5f);
		musicafondo.setLooping(true);
	}
	public static void playMusica(){
		musicafondo.play();
	}

	public static void stopMusica(){
		musicafondo.pause();
	}

	public static void dispose(){
		collepeixe.dispose();
		musicafondo.dispose();
	}
}
```

Agora temos que facer que cando o xogo comece, a música empece a soar e cando remate liberar os recursos.

Arquivo Principal.java
Cargamos a música, liberamos e comezamos a tocar.

```
	@Override
	public void create() {
		// TODO Auto-generated method stub
		AssetsXogo.cargarTexturas();
		Sonidos.inicializarSonidos();
		Sonidos.playMusica();

		pantallaxogo = new PantallaXogo(this);
```

```
                menuxogo=new MenuXogo(this);
                setScreen(menuxogo);

        }
        @Override
        public void dispose(){
                super.dispose();
                menuxogo.dispose();
                pantallaxogo.dispose();
                AssetsXogo.liberarTexturas();
                Sonidos.dispose();
        }
```

Exercicio: facede que cando o pescador colla un peixe soe o efecto de son (a clase Sound ten un método play).

Solución:
Arquivo ControladorXogo.java

```
        private void controlarGancho(){
                if (gancho.getPosicion().y>Gancho.LIMITE_Y_SUPERIOR){ // Poñer sempre '>' nunca '==' xa que
se move con valores float´s
                        gancho.setVelocidade(0);
                        gancho.setPescando(false);
                        gancho.getPosicion().y=Gancho.LIMITE_Y_SUPERIOR;
                }
                else {
                        if (gancho.getPosicion().y<=0){  // Limite inferior
                                gancho.setVelocidade(+gancho.velocidade_max);
                        }
                }

                if (!gancho.getTenunpeixe()){
                        // Comprobamos se o gancho colle a un peixe
                        for (Peixe peixe : mundo.getPeixes()){
                                if (Intersector.overlapRectangles(peixe.getRectangulo(),
gancho.getRectangulo())){
                                        Sonidos.collepeixe.play(1); // 1 é o volumen
                                        peixe.setPescado(true);
                                        gancho.setTenunpeixe(true);
                                }
                        }
                }

        }
```

Exercicio: facede que cando o xogo entra en pause a música se poña en pause.

Solución: poderíamos ir os métodos de pause das dúas pantallas, pero como van facer o mesmo en todas as que haia podemos sobreescribir o método pause da pantalla que deriva de Game.

Arquivo Principal.java

```
        @Override
        public void pause() {
                super.pause();
                Sonidos.stopMusica();
        }
        @Override
        public void resume() {
                super.resume();
                Sonidos.playMusica();
        }
```

Exercicio: Agora temos que engadir un check (on/off) para activar / desactivar a música (usar clase CheckBox).
Pista: usar a interface ClickListener.

Solución:

Arquivo MenuXogo.java
Engadimos un checkbox para activar ou desactivar a música.
Método presentarPantalla():

```
private void presentarPantalla(){

        // CHECKBOX MUSICA
        final CheckBox chkMusica = new CheckBox("MUSICA",skin);
        chkMusica.setChecked(true);
        chkMusica.setPosition(10, stage.getHeight()-50);
        chkMusica.addListener(new ClickListener(){
            @Override
            public void clicked(InputEvent event, float x, float y)
            {
                        if(chkMusica.isChecked())
                                Sonidos.playMusica();
                        else
                                Sonidos.stopMusica();
            }
        });

        stage.addActor(chkMusica);
.....
```

NOTA: podemos gardar as preferencias do xogo utilizando a clase Preferences => http://www.badlogicgames.com/wordpress/?p=1585

Pantalla HIGH SCORES

Dita pantalla debe amosar os 3 mellores pescadores do xogo (os que conseguiron pescar máis peixes nos dous minutos).

Por un lado debemos gardar ditas puntuacións xunto cos nomes dos gañadores. Para facelo podemos utilizar as preferencias (nomeadas antes) ou ben acceder a un arquivo.

Imos a usar esta segunda opción.

A idea é a seguinte: gardaremos nun arquivo os seguinte datos:

nome
puntuación
nome
puntuación
nome
puntuación

Estas puntuacións as cargaremos nun array bi-dimensional de tipo String.
Crearemos un método que cargue ditas puntuacións dun arquivo e outro que as garde.
Ademais crearemos un método que engada unha puntuación (xunto co nome) na posición correcta dentro deste array de puntuacións así como outro método que nos devolva, dada unha puntuación, a posición dentro do array de puntuacións.

Arquivo Settings.java. Dentro do paquete principal
Facemos o dito anteriormente.

```
public class Settings {


	public final static String[][] highscores = {{"------","0"},
	                                             {"------","0"},
	                                             {"------","0"}};

	public final static String file = ".opescador";

	public static void load () {
		BufferedReader in = null;
		try {
			in = new BufferedReader(new InputStreamReader(Gdx.files.external(file).read()));
			String valor="";
			for (int i = 0; i < 3; i++) {
				valor = String.valueOf(in.readLine());
				highscores[i][0] = (valor.length()==0) ? "------":valor;// Nome
				valor = String.valueOf(in.readLine());
				highscores[i][1] = (valor.length()==0) ? "0":valor;     // Puntuación
			}
		} catch (Throwable e) {
			// :( It's ok we have defaults
		} finally {
			try {
				if (in != null) in.close();
			} catch (IOException e) {
				Gdx.app.log("LIBGDX","Erro ó cargar");
			}
		}
	}
```

```
        public static void save () {
                BufferedWriter out = null;
                try {
                        out = new BufferedWriter(new
OutputStreamWriter(Gdx.files.external(file).write(false)));
                        for (int i = 0; i < 3; i++) {
                                out.write(highscores[i][0]);    // Nome
                                out.newLine();
                                out.write(highscores[i][1]);    // Puntuación
                                out.newLine();
                        }

                } catch (Throwable e) {
                } finally {
                        try {
                                if (out != null) out.close();
                        } catch (IOException e) {
                                Gdx.app.log("LIBGDX","Erro ó gardar");
                        }
                }
        }

        // Engade unha nova puntuación o array e devolve o posto. Se devolve -1 non foi engadido
        public static int addGañador (String nome, int numpeixes) {
                for (int i = 0; i < 3; i++) {
                        if (Integer.parseInt(highscores[i][1]) < numpeixes) {
                                for (int j = 2; j > i; j--){    // Movemos valores
                                        highscores[j][0] = highscores[j - 1][0];
                                        highscores[j][1] = highscores[j - 1][1];
                                }
                                highscores[i][0] = nome;
                                highscores[i][1] = String.valueOf(numpeixes);
                                return i+1;
                        }
                }
                return -1;
        }
        // Devolve o posto dentro dos 3 tempos máis rápidos e -1 se non foi rápido de abondo
        public static int comprobarPosto (int numpeixes) {
                for (int i = 0; i < 3; i++) {
                        if (Integer.parseInt(highscores[i][1]) < numpeixes) {
                                return i+1;
                        }
                }
                return -1;
        }

}
```

Neste caso estamos a falar de programación (traballo cun array bidimensional).
O único que nos interesa e como o framework traballa cos arquivos.
Como vemos temos no paquete **Gdx.files** todo o necesario para traballar con arquivos.

Podedes consultar todo o referente a o manexo dos arquivos en
http://code.google.com/p/libgdx/wiki/FileHandling

Cabe sinalar que temos a opción de manexar 'localizacións' de arquivos diferentes. No caso de utilizar un dispositivo Android:

- Gdx.files.internal: fai referencia a arquivos gardados na memoria interna do dispositivo Android. Só son accesibles pola aplicación. Só se ten permiso de LECTURA.
- Gdx.files.external: fai referencia a arquivos gardados na tarxeta SD do dispositivo Android. Son accesibles por tódolas aplicacións. É NECESARIO TER PERMISO DE ESCRITURA sobre a tarxeta SD no AndroidManifiest (*android.permission.WRITE_EXTERNAL_STORAGE*).

Nota: Se poñemos como nome de arquivo un que teña un punto o comenzo non se visualizará no administrador de arquivos de Android.

Na versión desktop o arquivo o garda en: C:\Documents And Settings\Ususario.

Agora imos crear a pantalla de HIGH SCORES, pero antes imos facer unhas modificacións.

Como dixemos anteriormente pode ser necesario o uso da clase AssetManager para cargar os recursos do noso xogo.
Imos cambiar a forma de cargar os nosos recursos gráficos utilizando dita clase.

Agora mesmo na clase MenuXogo estamos a cargar os recursos utilizando a clase AssetManager. Imos 'levar' a carga de ditos recursos á clase AssetsXogo e usaremos a clase AssetManager para cargar todo.

Nota: Se podería facer unha pantalla de carga que indicara por onde vai a carga dos recursos gráficos.

Arquivo AssetsXogo.java

```
public class AssetsXogo {

	/**
	 * Textura do fondo
	 */
	public static Texture fondopecera;
	/**
	 * Textura do pescador
	 */
	public static TextureRegion pescador;
	/**
	 * Textura do gancho
	 */
	public static TextureRegion gancho;
	/**
	 * Textura da corda da caña
	 */
	public static TextureRegion sedal;

	/**
	 * Textura Atlas con todos os gráficos.
	 */
	static TextureAtlas texturas_todas;

	/**
	 * Animación do peixe lila
	 */
	public static Animation peixe_lila;

	/**
	 * Animación do peixe amarelo
	 */
	public static Animation peixe_amarelo;

	/**
	 * Textura do peixe azul
	 */
	public static TextureRegion peixe_azul;

	/**
	 * Textura de pause
```

```
         */
        public static TextureRegion pause;
        /**
         * Textura de sair
         */
        public static TextureRegion sair;

        /**
         * Fonte a usar polo xogo na pantalla de xogo
         */
        public static BitmapFont bitmapfont;

        /**
         * Usado para cargar todos os assets.
         */
        private static AssetManager amanager;

        /**
         * Estilo dos elementos gráficos. Usado na principal e no high scores
         */
        public static Skin skin;

        /**
         * Carga os estilos para amosar texto
         */
        public static void cargarAssets() {
                amanager = new AssetManager();
                // Estilos para amosar no texto na pantalla principal e highscores
                amanager.load("scene2dui/uiskin.atlas",TextureAtlas.class);
                amanager.load("scene2dui/uiskin.json", Skin.class, new
SkinLoader.SkinParameter("scene2dui/uiskin.atlas"));


                // Fontes para utilizar na pantalla de xogo
                BitmapFontParameter bmfparameter = new BitmapFontParameter();
                bmfparameter.flip=false;
                amanager.load("fontes/fonte.fnt",BitmapFont.class,bmfparameter);

                // Carga imaxes do xogo
                amanager.load("imaxes/pescador_atlas.atlas",TextureAtlas.class);

                // Imaxe de fondo
                amanager.load("imaxes/pecera1024x512.jpg",Texture.class);

                amanager.finishLoading();

                // Recolle as fontes
                bitmapfont = amanager.get("fontes/fonte.fnt", BitmapFont.class);

                // Recolle o mapa do skin
                skin = amanager.get("scene2dui/uiskin.json", Skin.class);

                texturas_todas = amanager.get("imaxes/pescador_atlas.atlas", TextureAtlas.class);
                fondopecera = amanager.get("imaxes/pecera1024x512.jpg", Texture.class);

                pescador = texturas_todas.findRegion("pescador_sengancho");
                gancho = texturas_todas.findRegion("gancho");
                sedal = texturas_todas.findRegion("punto");

                sair = texturas_todas.findRegion("sair");
                pause = texturas_todas.findRegion("pause");

                peixe_azul = texturas_todas.findRegion("fish3");

                cargandoAnimacions();
```

```
	}

	/**
	 * Carga as animacions dos peixes
	 */
	private static void cargandoAnimacions(){

		TextureRegion trPeixe = texturas_todas.findRegion("fish1_animacion");
		TextureRegion[][] tmp = trPeixe.split(96,96);
		int num_filas = trPeixe.getRegionHeight() / 96;
		int num_columnas = trPeixe.getRegionWidth() / 96;

		TextureRegion[] framesanimacion = new TextureRegion[num_filas*num_columnas];
	int index = 0;
	for (int i = 0; i < num_filas; i++) {
		for (int j = 0; j < num_columnas; j++) {
		framesanimacion[index++] = tmp[i][j];
		}
	}

		peixe_lila = new Animation(0.15f,framesanimacion);

		trPeixe = texturas_todas.findRegion("fish2_animacion");
		tmp = trPeixe.split(96,96);
		num_filas = trPeixe.getRegionHeight() / 96;
		num_columnas = trPeixe.getRegionWidth() / 96;

		framesanimacion = new TextureRegion[num_filas*num_columnas];
	index = 0;
	for (int i = 0; i < num_filas; i++) {
		for (int j = 0; j < num_columnas; j++) {
		framesanimacion[index++] = tmp[i][j];
		}
	}
		peixe_amarelo = new Animation(0.15f,framesanimacion);
	}

	/**
	 * Método encargado de liberar todas as texturas
	 */
	public static void liberarTexturas(){

		amanager.dispose();

	}
}
```

Agora modificamos a clase Principal para que utilice os recursos cargados có AssetManager.

Arquivo Principal.java

```
	@Override
	public void create() {
		// TODO Auto-generated method stub
		AssetsXogo.cargarAssets();
		Sonidos.inicializarSonidos();
		Sonidos.playMusica();
		Settings.load();

		pantallaxogo = new PantallaXogo(this);
		menuxogo=new MenuXogo(this);

		setScreen(menuxogo);

	}
	@Override
	public void dispose(){
		super.dispose();
		menuxogo.dispose();
		pantallaxogo.dispose();
```

```
		AssetsXogo.liberarTexturas();
		Sonidos.dispose();
		Settings.save();
	}
```

Agora modificamos a clase MenuXogo para que fago uso do skin que se cargou na clase AssetsXogo.

Arquivo MenuXogo.java

```
	AssetManager amanager;
	Skin skin;

	private void cargarAssets() {
	.....

	public MenuXogo(Principal game){
		this.game=game;
		stage = new Stage();
		skin = AssetsXogo.skin;

		cargarAssets();
		presentarPantalla();
	}
	@Override
	public void dispose() {
		// TODO Auto-generated method stub
		if (skin!=null)	skin.dispose();
		if (amanager!=null)amanager.dispose();

	}
```

Creando a pantalla HighScores:
Xa podemos crear o arquivo HighScores.java no paquete de pantallas e implementar a interface screen e facer un constructor que reciba unha referencia da clase Principal.

Solución:

```
public class HighScores implements Screen {

	private Principal game;

	public HighScores(Principal game){
		this.game=game;
	}

	.................
}
```

Na clase Principal:
Podemos crear unha propiedade que sexa referencia a dita pantalla.
Liberar a pantalla no método dispose.
Crear un método getHighScores que devolva unha referencia a dita pantalla.

Na clase MenuXogo:
Facer que cando se preme o botón 'High Scores' amose a nova pantalla.

Exercicio: facer o anterior.

Solución:

Arquivo Principal.java

```
HighScores highscores;

@Override
public void create() {
        // TODO Auto-generated method stub
        AssetsXogo.cargarAssets();
        Sonidos.inicializarSonidos();
        Sonidos.playMusica();
        Settings.load();

        pantallaxogo = new PantallaXogo(this);
        menuxogo=new MenuXogo(this);
        highscores = new HighScores(this);
        ……..
}
public HighScores getHighscores() {
        return highscores;
}
@Override
public void dispose(){
        super.dispose();
        menuxogo.dispose();
        pantallaxogo.dispose();
        highscores.dispose();
        AssetsXogo.liberarTexturas();
        Sonidos.dispose();
}
```

Arquivo MenuXogo.java
Método presentarPantalla():

```
        highscores.addListener(new ClickListener(){
                public void clicked (InputEvent event, float x, float y) {
                        game.setScreen(game.getHighscores());
                }
        });
```

Agora xa podemos traballar coa clase HighScores.
Ó igual que na clase MenuXogo imos facer uso da clase Stage para debuxar en forma de táboa os tres mellores tempos cun botón que volva ó menú principal.

Arquivo HighScores.java

```
public class HighScores implements Screen {
        Stage stage;
        Skin skin;

        /**
         * Recolle a puntuación do pescador
         */
        public static int highscore=-1;


        private Principal game;

        public HighScores(Principal game){
```

```
                skin=AssetsXogo.skin;
                this.game=game;
                stage = new Stage();
        }

        /**
         * Engade todo o necesario á pantalla de menú
         */
        private void presentarPantalla(){


        // retrieve the default table actor
        Table table = new Table(skin);
        table.setFillParent(true);
        table.defaults().spaceBottom( 30 );
        Label titulo = new Label("High scores",skin);
        titulo.setColor(Color.GREEN);
        titulo.scale(30f);
        table.add(titulo).colspan( 2 ).center();
        table.row();

        // Podería facerse cun bucle
        table.row();
        table.add(Settings.highscores[0][0]);
        table.add(Settings.highscores[0][1]);

        table.row();
        table.add(Settings.highscores[1][0]);
        table.add(Settings.highscores[1][1]);

        table.row();
        table.add(Settings.highscores[2][0]);
        table.add(Settings.highscores[2][1]);

        TextButton backButton = new TextButton( "Voltar o menu", skin);
        backButton.addListener(new InputListener(){
                        public boolean touchDown (InputEvent event, float x, float y, int pointer, int
button) {
                                game.setScreen(game.getMenuxogo());
                                return true;
                        }
        });

        table.row();
        table.add( backButton ).size( 250, 60 ).colspan( 2 );

        stage.addActor(table);

    }
        @Override
        public void render(float delta) {
                // TODO Auto-generated method stub
                Gdx.gl.glClear(GL11.GL_COLOR_BUFFER_BIT);

                stage.act(Gdx.graphics.getDeltaTime());
                stage.draw();

        }

        @Override
        public void resize(int width, int height) {
                // TODO Auto-generated method stub
                stage.setViewport(width, height, true);
        }
        @Override
        public void show() {
                // TODO Auto-generated method stub
                stage.clear();  // Chamamos varias veces a presentarPantalla, dende o Menu principal e xogo.
                Gdx.input.setInputProcessor(stage);

        }
```

```
	@Override
	public void pause() {
		// TODO Auto-generated method stub
		Gdx.input.setInputProcessor(null);
	}

	@Override
	public void resume() {
		// TODO Auto-generated method stub
		Gdx.input.setInputProcessor(stage);
	}

	@Override
	public void dispose() {
		// TODO Auto-generated method stub
		Gdx.input.setInputProcessor(null);
	}
}
```

Como a esta pantalla podemos vir dende a do xogo (cando remate o tempo) e dende a de menú, imos facer que:

- Se ven dende a pantalla de menú, simplemente chamamos a presentarPantalla.
- Se ven dende o xogo, necesitamos acceder ós peixes que pescou o pescador. Isto o veremos máis adiante pero por agora chega con saber que esta información a imos ter na variable de clase highscore.

  Agora só queda implementar a lóxica. O facemos no método show da clase HighScores:

```
		@Override
		public void show() {
			// TODO Auto-generated method stub
			Gdx.input.setInputProcessor(stage);
			stage.clear();  // Chamamos varias veces a presentarPantalla, dende o Menu principal
e xogo.

			if (Settings.comprobarPosto(highscore)==-1)
				presentarPantalla();			// Non pregunta nome
			else
				preguntarGañador();   // Pregunta nome e presenta pantalla

		}
```

O método preguntarGañador amosa unha caixa de diálogo no que se pide introducir o nome:

```
	/**
	 * Lanza o cadro de preguntar nome
	 */
	private void preguntarGañador(){
		Dialog dialogo = new Dialog("O teu nome", skin, "dialog");

		dialogo.setSize(200, 200);

		final TextField nome = new TextField("",skin);
		Button ok = new Button(skin);
		ok.add("ACEPTAR");
		ok.addListener(new ClickListener(){
			public void clicked (InputEvent event, float x, float y) {
				Settings.addGañador(nome.getText(),highscore);
				Settings.save();
				highscore=-1;
				presentarPantalla();
			}
		});
```

```
        dialogo.button(ok);
        dialogo.getContentTable().row();
        dialogo.getContentTable().add(nome).width(135);

        dialogo.show(stage);
        stage.setKeyboardFocus(nome);
    }
```

Cando aceptamos, o nome xunto coa puntuación é enviado o método addGañador de Settings que engade os datos ó array. Despois gardamos dito array no disco e presentamos a pantalla. Inicializamos a propiedade *highscore* para que se volvemos dende a pantalla de menú non volva a preguntar o nome.

Nota: outra forma de pedir información ó usuario a podedes atopar en: https://github.com/libgdx/libgdx/blob/master/tests/gdx-tests/src/com/badlogic/gdx/tests/TextInputDialogTest.java

NA VERSIÓN ANDROID:

Arquivo AndroidManifiest.xml:

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

Agora temos que facer que cando o cronómetro chegue a 0, amose a pantalla de highscores. O cambio o temos que facer na clase PantallaXogo pero o control na clase ControladorXogo.

Exercicio: intentade facer o anterior.

Arquivo PantallaXogo.java

```
/**
 * Indica se o xogo acabou
 */
public static boolean acabou;
public PantallaXogo(Game principal){
        game=principal;
        pause = false;
        acabou = false;
}
/**
 * Inicializa o mundo e o xogo
 */
public void inicializarXogo(){
        acabou=false;
        Mundo.cronometro = Mundo.NUM_SEGUNDOS_XOGO;
        mundo.getPeixes().clear();
        mundo.getPescador().inicializaPeixesCollidos();
}

@Override
public void render(float delta) {
        // TODO Auto-generated method stub
        rendererxogo.render(delta);
        if (!pause) {
                controladorxogo.update(delta);
                controlarAcelerometro();
        }
        if (acabou){
```

```
                HighScores.highscore = mundo.getPescador().getNum_peixes_collidos();
                inicializarXogo();
                game.setScreen(game.getHighscores());
        }
}
```

Arquivo ControladorXogo.java

```
public void update(float delta){

        pescador.update(delta);
        actualizarGancho(delta);
        actualizarPeixes(delta);
        crearPeixes();

        Mundo.cronometro += delta;

        controlarPescador();
        controlarGancho();
        controlarPeixes();
        controlarCronometro();

        procesarEntrada();
}

/**
 * Controla que o cronómetro chegue a 0 para acabar o xogo
 */
private void controlarCronometro(){
        if (Mundo.cronometro>= Mundo.NUM_SEGUNDOS_XOGO){
                PantallaXogo.acabou=true;
        }
}
```

Agora só queda a pantalla de axuda.
Podemos implementala de dúas maneiras:

- Cun stage como fixemos no highscores e pantalla de menú.
- Cunha imaxe na que estaría a axuda.

Imos facelo da segunda forma, tendo xa a imaxe feita no DVD (/Proxectos Eclipse/Proxecto 2D peixes/imaxes/axuda/axuda.jpg).

Exercicio: Cargamos dita imaxe no AssetsXogo cunha textura de clase de nome pantallaaxuda.

Solución:

Arquivo AssetsXogo.java

```
/**
 * Pantalla de axuda
 */
public static Texture pantallaaxuda;

public static void cargarAssets() {
        ..................
        // Imaxe da axuda
        amanager.load("imaxes/axuda.jpg",Texture.class);

        amanager.finishLoading();
        ..................
```

```
		fondopecera = amanager.get("imaxes/pecera1024x512.jpg", Texture.class);

		pantallaaxuda = amanager.get("imaxes/axuda.jpg", Texture.class);

		pescador = texturas_todas.findRegion("pescador_sengancho");
		.....................
```

Arquivo Axuda.java

```
public class Axuda implements Screen {
	private OrthographicCamera camaraortho;
	private SpriteBatch spritebatch;
	private Principal game;

	public Axuda(Principal game){
		camaraortho = new OrthographicCamera();
		spritebatch = new SpriteBatch();
		spritebatch.disableBlending();

		this.game=game;
	}

	@Override
	public void render(float delta) {
		// TODO Auto-generated method stub
		spritebatch.begin();
		spritebatch.draw(AssetsXogo.pantallaaxuda,0,0,Mundo.TAMAÑO_MUNDO.x,Mundo.TAMAÑO_MUNDO.y);
		spritebatch.end();

		if (Gdx.input.isTouched()){
			game.setScreen(game.getMenuxogo());
		}
	}

	@Override
	public void resize(int width, int height) {
		// TODO Auto-generated method stub
		camaraortho.setToOrtho(false,Mundo.TAMAÑO_MUNDO.x,Mundo.TAMAÑO_MUNDO.y);
		camaraortho.update();

		spritebatch.setProjectionMatrix(camaraortho.combined);

	}

	@Override
	public void show() {
		Gdx.input.setInputProcessor(null); // Se non o poñemos, ó pulsar sobre a pantalla Android
'leva' o evento de click á pantalla anterior e poderíamos premer un botón da pantalla de menú.

	}


	@Override
	public void hide() {
		// TODO Auto-generated method stub

	}

	@Override
	public void pause() {
		// TODO Auto-generated method stub

	}

	@Override
	public void resume() {
		// TODO Auto-generated method stub
```

```
        }

        @Override
        public void dispose() {
                // TODO Auto-generated method stub

                spritebatch.dispose();

        }

}
```

NOTA: como podemos observar as pantallas do xogo teñen todas a mesma estructura (implementar a interface screen, teñen unha cámara en perspectiva,....). Poderíamos facer unha clase abstracta que xa implementara os métodos comúns....

Agora debemos modificar Principal e MenuXogo para que chame a dita pantalla cando preme o botón de axuda.

Exercicio: facer o anterior.
Solución:

Arquivo Principal.java

```
        Axuda pantallaaxuda;

        public void create() {
                // TODO Auto-generated method stub

                AssetsXogo.cargarAssets();
                Sonidos.inicializarSonidos();
                Sonidos.playMusica();
                Settings.load();

                pantallaxogo = new PantallaXogo(this);
                menuxogo=new MenuXogo(this);
                highscores = new HighScores(this);
                pantallaaxuda = new Axuda(this);
                ..............


        public Axuda getPantallaaxuda() {
                return pantallaaxuda;
        }

        @Override
        public void dispose(){
                super.dispose();

                menuxogo.dispose();
                pantallaxogo.dispose();
                highscores.dispose();
                pantallaaxuda.dispose();
                AssetsXogo.liberarTexturas();
                Sonidos.dispose();;
        }
```

Arquivo MenuXogo.java
Método presentarPantalla():

```
        private void presentarPantalla(){
                ......................

                axuda.addListener(new ClickListener(){
                        public void clicked (InputEvent event, float x, float y) {
                                game.setScreen(game.getPantallaaxuda());
                        }

                });
                ......................
```

Como mellora imos facer que cando volvamos do xogo se o pescador colleu 5 peixes, aparece unha mensaxe de felicitación e outra en caso contrario.

Primeiro definimos en forma de constante na clase Pescador o número de peixes que ten que coller para gañar.

Arquivo Pescador.java

```
        /**
         * Número de peixes necesrios para que gañe
         */
        public final static int NUM_PEIXES_PARA_GAÑAR = 5;
```

Agora modificamos a clase HighScores no método show para que amose unha caixa de diálogo para informar como o fixo.

Arquivo HighScores.java

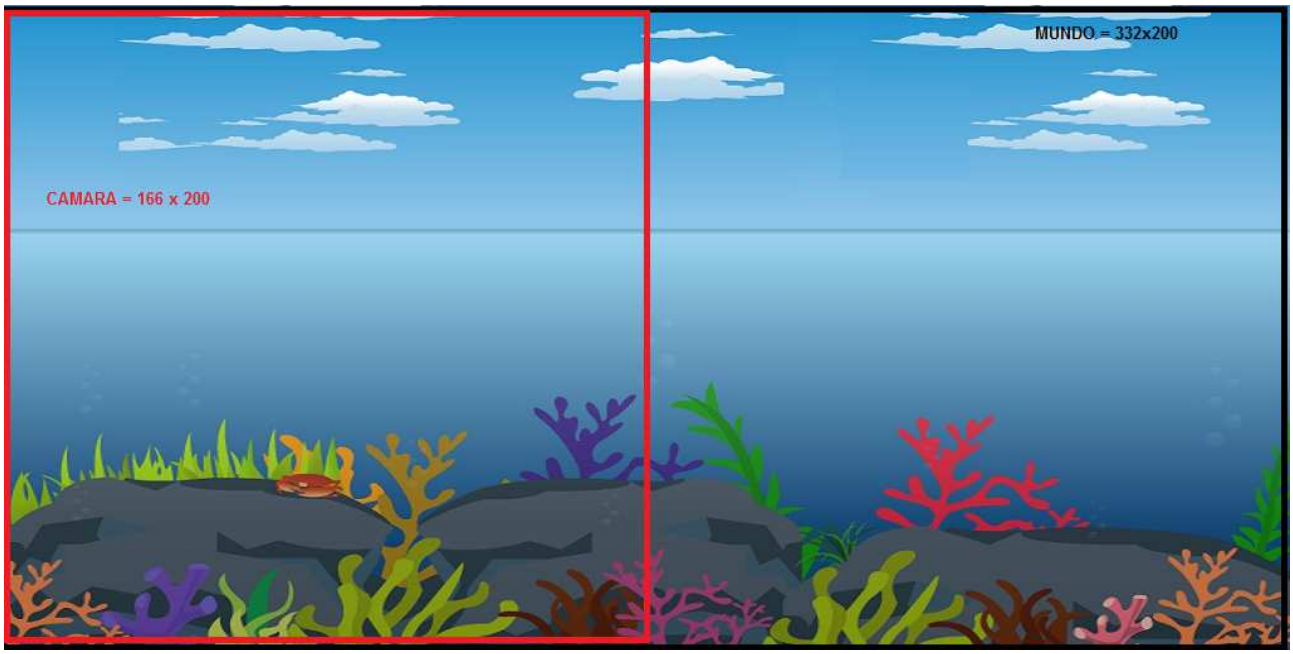
```
@Override
public void show() {
        // TODO Auto-generated method stub
        Gdx.input.setInputProcessor(stage);

        stage.clear();  // Chamamos varias veces a presentarPantalla, dende o Menu principal e xogo.

        if (highscore!=-1){
             String texto="";
             if (highscore>=Pescador.NUM_PEIXES_PARA_GAÑA)    // FELICIDADES, pescaches o necesario.
                     texto = "Felicidades. \nA familia pudo comer...";
             else
                     texto = "Hoxe foi un dia triste...";

             Dialog dialogo = new Dialog("Acabou o xogo", skin, "dialog"){
                        public void result(Object object){
                        if (Settings.comprobarPosto(highscore)==-1)
                                        presentarPantalla();            // Non pregunta nome
                                else
                                        preguntarGañador();    // Pregunta nome e presenta pantalla

                        }
             };
             dialogo.button("Aceptar", true).key(Keys.ENTER, true);
             dialogo.text(texto);
             dialogo.setModal(true);
             dialogo.show(stage);
        }
        else
                presentarPantalla();            // Non pregunta nome

}
```

XOGO CON SCROLL

Imos facer unha pequena modificación para que o xogo teña un scroll horizontal.
Existen varias aproximaciones para facer isto.
A idea é ter unha textura de fondo que se repita de forma indefinida.
No repositorio do framework podedes ver unha forma de facelo:
https://github.com/libgdx/libgdx/blob/master/tests/gdx-tests/src/com/badlogic/gdx/tests/ParallaxTest.java

Outra forma:
http://code.google.com/p/libgdx-users/wiki/ScrollingTexture

Nos imos a utilizar un pequeno truco que vai consistir en facer que a cámara teña un tamaño máis pequeno no eixe X.



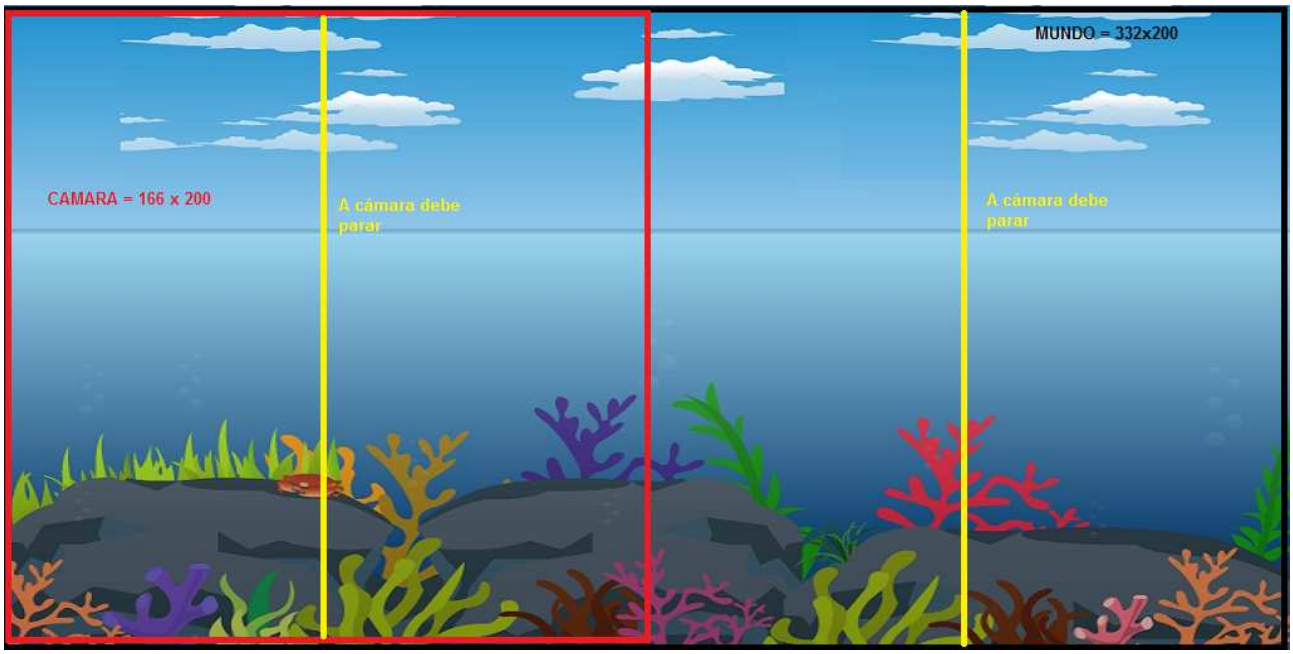
Loxicamente os gráficos sairán un pouco deformados xa que os peixes que miden 30x30 (por exemplo) vanse debuxar nun ancho de 166 en vez de nun de 332.

As modificacións son as seguintes:

Arquivo RendererXogo.java

```
public void resize(int width, int height) {
        camaraortho.setToOrtho(false,Mundo.TAMAÑO_MUNDO.x/2,Mundo.TAMAÑO_MUNDO.y);
        camaraortho.update();
```

Agora temos que facer que a cámara se mova seguindo ó pescador, pero tendo en conta os límites esquerdo e dereito, xa que cando o pescador chegue a posición 1/4 do mundo a cámara debe parar e o mesmo cando chegue a posición 3/4 do mundo.



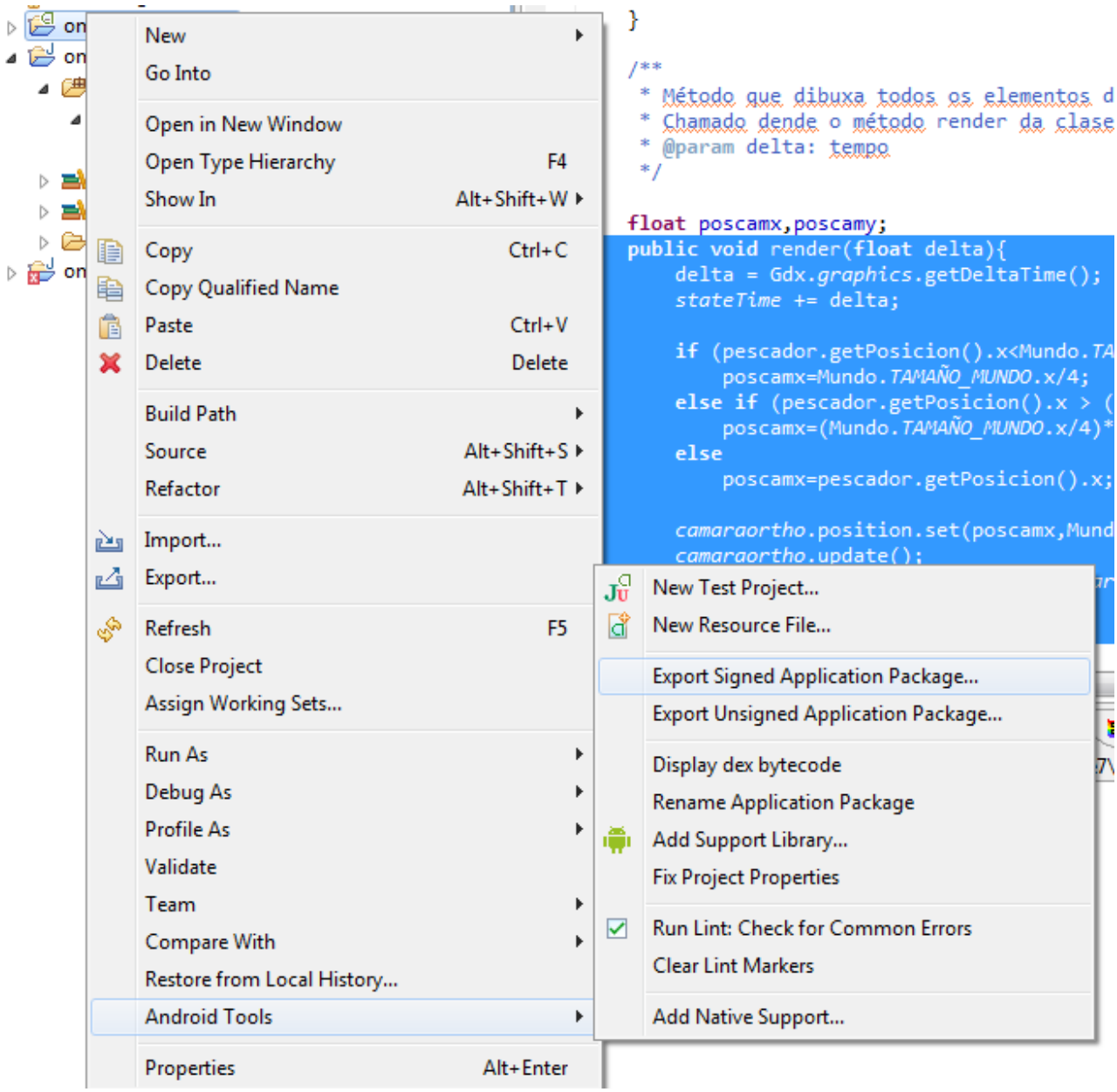
Arquivo RendererXogo.java

```
        float poscamx,poscamy;

        public void render(float delta){
                delta = Gdx.graphics.getDeltaTime();
                stateTime += delta;

                if (pescador.getPosicion().x<Mundo.TAMAÑO_MUNDO.x/4)
                        poscamx=Mundo.TAMAÑO_MUNDO.x/4;  // A mitade do ancho do mundo = x/2 (mirar
resize)
                else if (pescador.getPosicion().x > (Mundo.TAMAÑO_MUNDO.x/4)*3)
                        poscamx=(Mundo.TAMAÑO_MUNDO.x/4)*3;
                else
                        poscamx=pescador.getPosicion().x;

                camaraortho.position.set(poscamx,Mundo.TAMAÑO_MUNDO.y/2, 0);
                camaraortho.update();
                spritebatch.setProjectionMatrix(camaraortho.combined);

                spritebatch.begin();
```

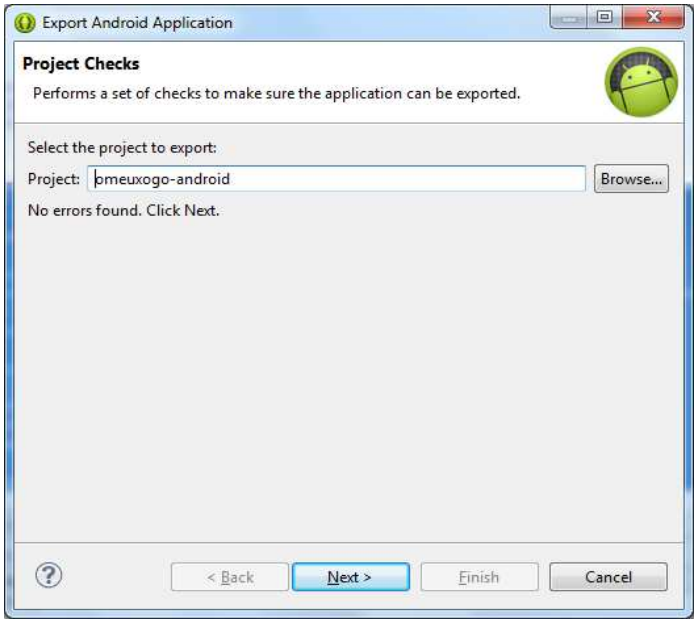
INSTALANDO O XOGO NUN DISPOSITIVO ANDROID.

Temos que xerar o arquivo apk da aplicación pero coa aplicación 'firmada' cun keystore que ven ser un almacén de claves.

O podemos facer dende o entorno do eclipse. Simplemente nos situamos sobre a versión de Android, prememos botón dereito e escollemos a opción Android Tools => 'Export Signed Application Package':



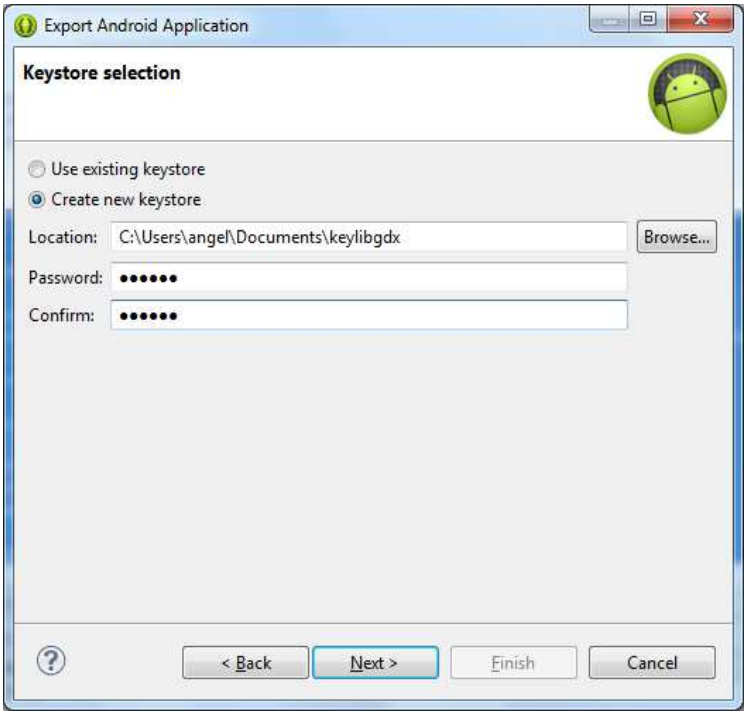
Aparece un asistente:



Prememos next.

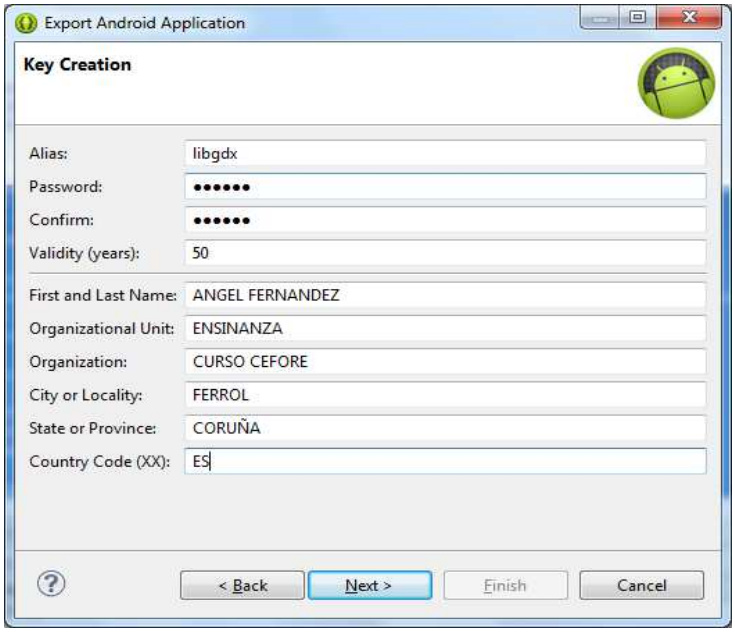
Na seguinte pantalla indica se xa temos un keystore. Se o tivésemos deberíamos seleccionalo.

Como imos crear un novo escollemos a segunda opción, indicamos onde se vai gardar o keystore e a contrasinal do almacén de claves.



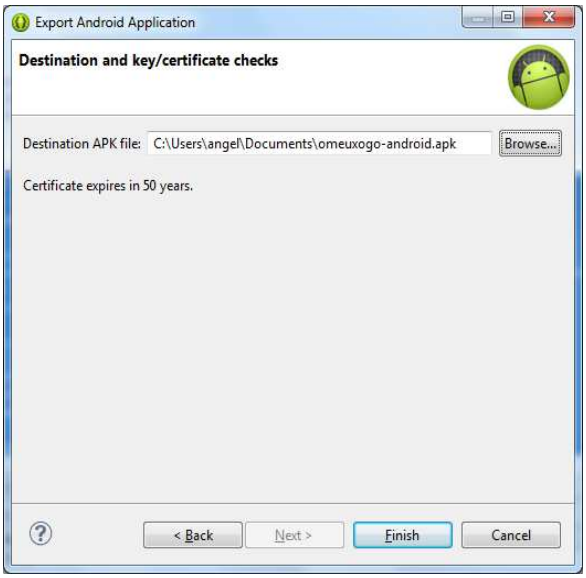
Prememos next.

- **Alias**: Un alias para o keystore. Pode ser o mesmo que o nome ou unha abreviación do mesmo.
- **Password**: Novamente asinámoslle unha contrasinal. Ten que ter o lo menos 6 caracteres. Esta vai ser a contrasinal da key,
- **Validity (years)**: Aquí indicamos o tempo que vai ser válida a nosa keystore en anos.
- Os seguintes campos fan referencia a información persoal e da organización. O campo de **Country Code**, se pode consultar no listado da ISO 3166-1.



Prememos Next.

Indicamos o lugar onde se vai xerar o apk xa coa firma:



Agora xa podemos instalar o apk en calquera dispositivo con Android.

FUNCIÓNS NON VISTAS.

Neste apartado nomearei algunhas das funcións e actividades non vistas neste curso.

- **Descarga e instalación do repositorio.**
  https://github.com/libgdx/libgd => Repositorio
  https://code.google.com/p/libgdx/wiki/SourceBuildin => Instruccións

- **Motor de físicas para 2D box2d**:
  http://code.google.com/p/libgdx/wiki/PhysicsBox2D
  https://code.google.com/p/libgdx-users/wiki/box2d

- **Motor de físicas para 2D box2d con luces 2D**: https://code.google.com/p/box2dlights/

- **Mapas en Libgdx:** http://code.google.com/p/libgdx/wiki/GraphicsTileMaps

- **Integrando Google Play Services:** http://helios.hud.ac.uk/u1070589/blog/?p=202

- **Editor de partículas:** http://code.google.com/p/libgdx/wiki/ParticleEditor

- **Perspectiva isométrica:** http://www.badlogicgames.com/wordpress/?p=2032

ENLACES E BIBLIOGRAFIA.

- **Libro 'Beginning Android Games': http://www.apress.com/9781430246770**
- **Titorial:** http://obviam.net/index.php/getting-started-in-android-game-development-with-libgdx-create-a-working-prototype-in-a-day-tutorial-part-1/
- **Explicación de Skin e arquivos json:** http://code.google.com/p/libgdx/wiki/Skin
- **Como crear un arquivo json para facer un skin en Scene2D:** http://pimentoso.blogspot.com.es/2013/04/libgdx-scene2d-skin-quick-tutorial.html
- **Animacións en SCENE2D:** http://libgdx.l33tlabs.org/docs/api/com/badlogic/gdx/scenes/scene2d/AnimationAction.html
- **Engade módulos para soporte de xogos multi-xogador:** http://cebrianbrothers.blogspot.com.es/2012/11/libgdx-net.html#!/2012/11/libgdx-net.html
- **Desenrolo dun xogo con Stage2d:** http://www.bigbrainmaster.com/blog/
- **Librería JAVA para facer comunicacións vía NETWORK:** https://code.google.com/p/kryonet/
- **Motor de física 2D: http://dpk.net/2011/05/01/libgdx-box2d-tiled-maps-full-working-example-part-1**
- **Titorial:** http://rotatingcanvas.com/tutorials-index/
- **Physics Body Editor:** http://www.aurelienribon.com/blog/projects/physics-body-editor/
- **Crear unha animación 2D:** http://siondream.com/blog/general/2d-character-creation-and-animation-for-dummies/