LIBGDX Preparando o esqueleto

UNDIDADE 2: Preparando o 'esqueleto' do noso xogo

Preparando o 'esqueleto' do noso xogo

Como comentamos anteriormente, a clase que usan os diferentes proxectos debe ser unha subclase da clase *ApplicationAdapter*, pero tamén comentamos que podía ser unha subclase da clase *Game*.

Que diferenza hai entre unha opción e outra ?

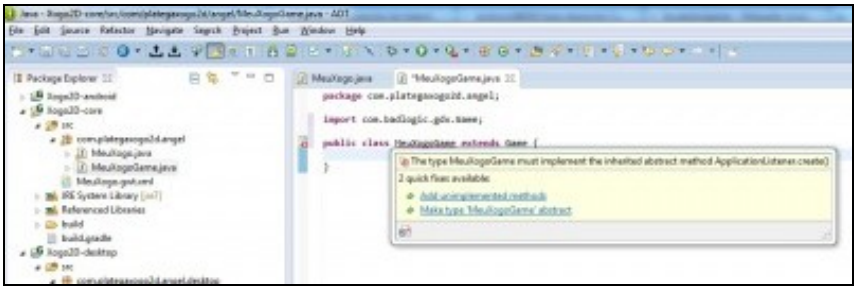
A diferenza se atopa en que se usamos a clase Game imos poder ter (a nivel de programación) unha clase por cada pantalla para o noso xogo e podemos xestionalas-programalas independentemente. Moito máis práctico, claro e doado de manter. En caso de usar a ApplicationAdapter, todo o código de noso xogo debería ir dentro do render de dita clase (que é o método que se chama de forma continuada) e se o xogo tivera varias pantallas teríamos que engadir algunha lóxica de programación para que amosara unha ou outra segundo o caso.

Polo tanto imos preparar o noso proxecto para utilizar a clase Game.

- Primeiro imos crear unha nova clase no proxecto Xogo2D-Core, e dentro deste no paquete com.plategaxogo2d.o_voso_nome, que é o paquete onde se atopa a clase MeuXogo. Dámoslle de nome MeuXogoGame e facemos que derive da clase Game.

Nota: Ó facelo deberemos de importar dita clase coa combinación de teclas Control+Shift+O ou ben situarnos enriba da clase e escoller a opción *Import .....*

Despois de facelo veremos que aparece un erro no nome da clase. Se nos situamos enriba dela aparecerá unha ventá para engadir os métodos que deben estar definidos na clase.



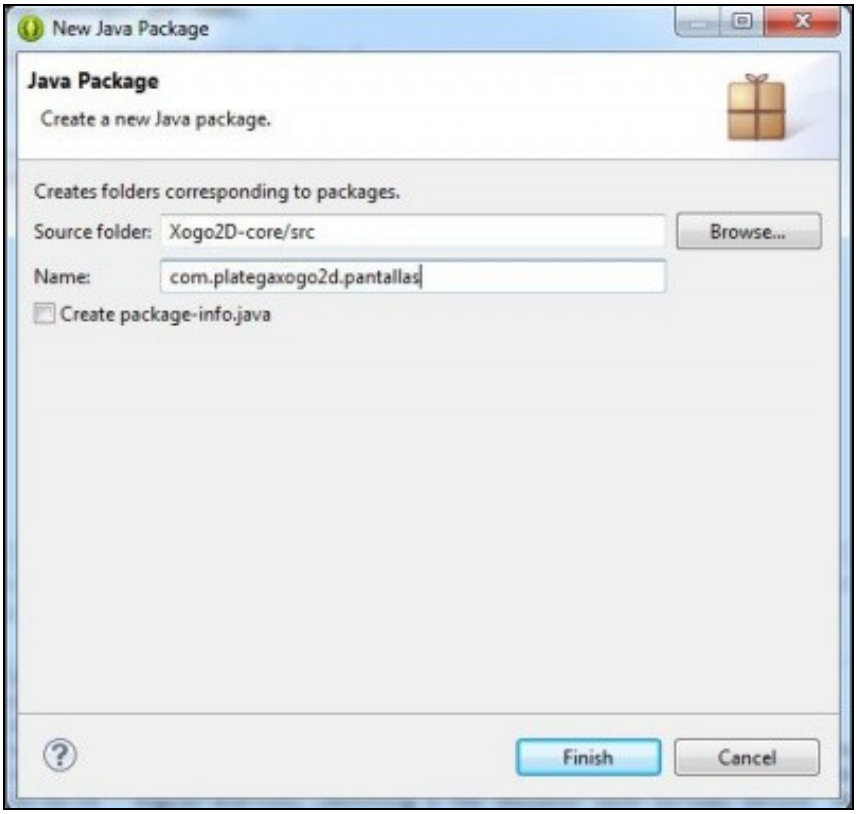
Aparecerá o método create...

Agora debemos de cambiar a clase que usan as diferentes plataformas pola nova clase creada.

```
package com.plategaxogo2d.angel.desktop;

import com.badlogic.gdx.backends.lwjgl.LwjglApplication;
import com.badlogic.gdx.backends.lwjgl.LwjglApplicationConfiguration;
import com.plategaxogo2d.angel.MeuXogoGame;

public class DesktopLauncher {
public static void main (String[] arg) {
LwjglApplicationConfiguration config = new LwjglApplicationConfiguration();
new LwjglApplication(new MeuXogoGame(), config);
}
}
```

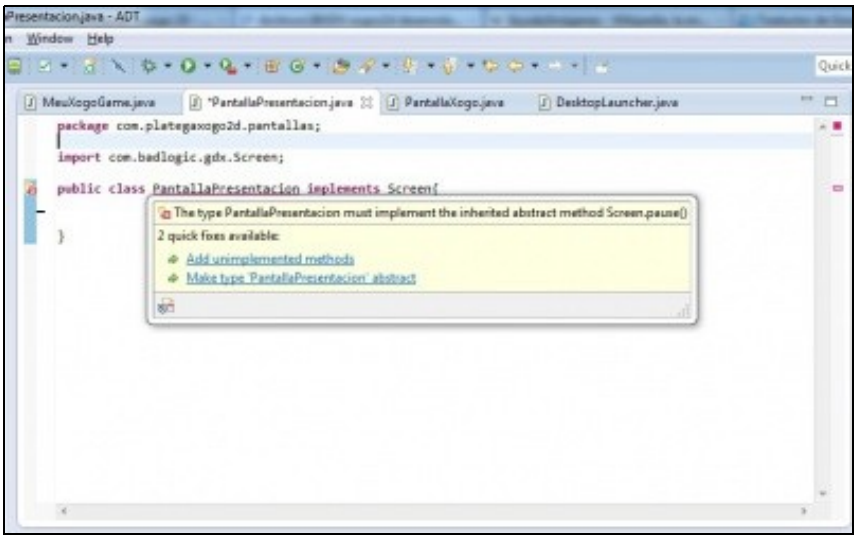
Nota: Exemplo feito sobre a versión Desktop do xogo. Deberemos facelo en todos os proxectos (html-android).

Agora imos crear un paquete onde van estar as pantallas do noso xogo. Dito paquete terá de nome *com.plategaxogo2d.pantallas*. Para crear o paquete prememos o botón dereito sobre o cartafol 'src' do proxecto Xogo2D-Core e escollemos a opción *New => Package*.



Dentro de dito paquete creamos unha clase de nome *PantallaPresentacion* que implemente a interface *Screen* (despois da definición da clase deberemos poñer *implements Screen*).

Ó facelo e despois de facer o import da clase (control+shift+O) aparecerá un erro enriba da clase. Igual que fixemos anteriormente, nos situamos enriba do nome da clase e escollemos a opción de Add Unimplemented Methods.



TAREFA A REALIZAR: Faremos as seguintes pantallas (repetiremos o proceso anterior):

- A da HighScores. Chamarémoslle *PantallaMarcadores*.
- A de pausa. Chamarémoslle *PantallaPause*
- A principal coas opcións de menú. Chamarémoslle *PantallaPresentacion*
- A do xogo. Chamarémoslle *PantallaXogo*.

Como facemos agora para pasar o control a cada unha das pantallas ? Faise a través do método setScreen da clase Game.

Imos facer no noso xogo que o control pase á pantalla de xogo. Podemos facelo de dúas formas:

Opción a): Neste caso creamos un obxecto da clase PantallaXogo e seremos nos dende a clase MeuXogoGame quen chamaremos ó método dispose de dito obxecto. Só instanciamos unha vez cada pantalla de xogo.

```
package com.plategaxogo2d.angel;

import com.badlogic.gdx.Game;
import com.plategaxogo2d.pantallas.PantallaXogo;

public class MeuXogoGame extends Game {

private PantallaXogo pantallaxogo;

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

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

}
```

Opción b): Creamos implicitamente un obxecto de clase PantallaXogo pero non gardamos referencia ningunha a el. Teremos que chamar dende a propia pantalla ó método dispose cando rematemos (cambiemos de pantalla). **USAREMOS ESTA OPCIÓN**.

```
package com.plategaxogo2d.angel;

import com.badlogic.gdx.Game;
import com.plategaxogo2d.pantallas.PantallaXogo;

public class MeuXogoGame extends Game {

@Override
public void create() {

setScreen(new PantallaXogo());
}

}
```

A partir deste momento, o framework páselle o control o método render da clase PantallaXogo e se queda nese método ata que decidamos cambiar de pantalla...

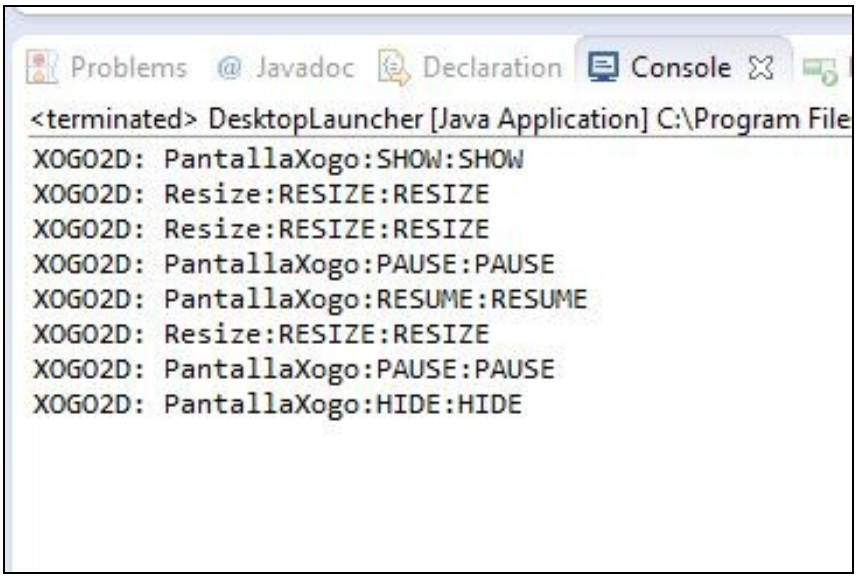
Analizamos agora os métodos implementados na clase *PantallaXogo*.

```
package com.plategaxogo2d.pantallas;

import com.badlogic.gdx.Screen;
import com.plategaxogo2d.angel.Utiles;

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
Utiles.imprimirLog("Resize", "RESIZE", "RESIZE");

}

@Override
public void show() {
// TODO Auto-generated method stub
Utiles.imprimirLog("PantallaXogo", "SHOW", "SHOW");
}

@Override
public void hide() {
// TODO Auto-generated method stub
Utiles.imprimirLog("PantallaXogo", "HIDE", "HIDE");

}

@Override
public void pause() {
// TODO Auto-generated method stub
Utiles.imprimirLog("PantallaXogo", "PAUSE", "PAUSE");

}

@Override
public void resume() {
// TODO Auto-generated method stub
Utiles.imprimirLog("PantallaXogo", "RESUME", "RESUME");

}

@Override
public void dispose() {
// TODO Auto-generated method stub
Utiles.imprimirLog("PantallaXogo", "DISPOSE", "DISPOSE");

}

}
```

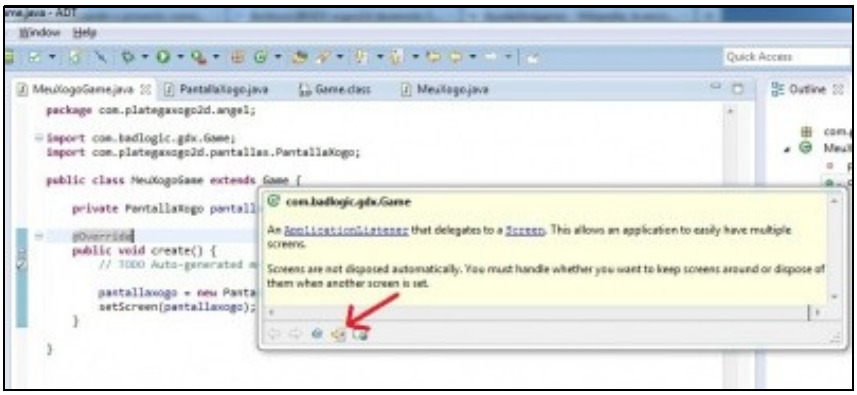
Como vemos son moi parecidos os vistos anteriormente na clase *ApplicationAdapter* pero con algunha pequena diferenza...

Se executados o proxecto Desktop, por exemplo, aparecerá unha ventá negra (é normal). Se minimizades a ventá e despois pechades o xogo veredes a secuencia de eventos.

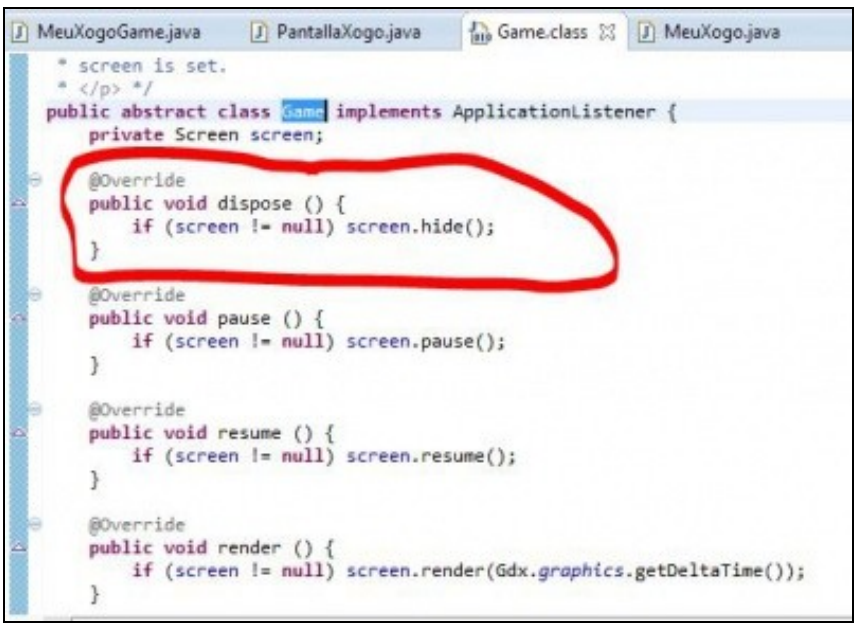


Como vemos, cando pechamos o xogo non executa o método dispose, se non o método hide. Por qué é así ? Podemos ver que é o que fai a clase Game cando chama o método dispose.

Situar o cursor sobre o nome Game na clase MeuXogoGame, situarvos enriba da parte amarela na ventá que aparece, e premede sobre o botón de *Open Declaration*...



Abrirase unha nova ventá có código de dita clase. Podemos observar o que fai cando se chama ó método dispose...



Polo tanto, temos que ser nos o que chamemos a dito método, ou ben dende a clase MeuXogoGame cando se produza o evento de dispose ou ben no método hide, dependendo como chamaramos dende MeuXogoGame a esta pantalla.

Opción a): Para facelo só temos que sobrescribir dito método.

```
package com.plategaxogo2d.angel;

import com.badlogic.gdx.Game;
import com.plategaxogo2d.pantallas.PantallaXogo;

public class MeuXogoGame extends Game {

private PantallaXogo pantallaxogo;

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

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

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

É IMPORTANTE LEMBRAR QUE SEMPRE TEREMOS QUE SER NOS O QUE DEBEMOS CHAMAR O MÉTODO DISPOSE DA CLASE QUE IMPLEMENTE A INTERFACE SCREEN.

Opción b): A que seguiremos nos.

Clase MeuXogoGame:

```
package com.plategaxogo2d.angel;

import com.badlogic.gdx.Game;
import com.plategaxogo2d.pantallas.PantallaXogo;

public class MeuXogoGame extends Game {

@Override
public void create() {

setScreen(new PantallaXogo());
}

}
```

Nesta forma, imos ter que distinguir cando cambiamos de pantalla por ir á pantalla de **PAUSA** do xogo e cando cambiamos de pantalla por **SAÍR** do xogo ou por **REMATAR** o xogo. É dicir. imos ter estas posibilidades:

- ◊ O xogador remata o xogo. Cambiamos á pantalla de HighScores e debemos chamar ó método dispose.
- ◊ O xogador preme o control de saír do xogo. Cambiamos á pantalla PRINCIPAL e debemos chamar ó método dispose.
- ◊ O xogador preme pausa. Cambiamos á pantalla de PAUSE mantendo o estado completo do xogo. Non debemos chamar ó método dispose.
- ◊ EN ANDROID:
  - O xogador preme o botón de cambio de aplicación ou atende a unha chamada. O estado se mantén. Non facemos nada (non imos cambiar de pantalla).
  - O xogador preme o botón de BACK para saír. Cambiamos á pantalla PRINCIPAL e debemos chamar ó método dispose.

Para atender a todas estas posibilidades imos facer uso de tres variables, **pausa**, **sair** e **finXogo** de tipo booleano. A propiedade finXogo será true cando remate o tempo do xogo. Pause e Sair serán utilizadas cando prememos os controis.

Clase PantallaXogo:

```
public class PantallaXogo implements Screen {

private boolean pause;
private boolean finxogo;
private boolean sair;


        ...........
```

- Vos estaredes preguntando, e por que é tan importante o método dispose...? Porque escribiremos nese método as ordes necesarias para liberar da memoria os recursos que teña dita pantalla, como poden ser os gráficos...

Como comentamos anteriormente, imos usar ó método setScreen da clase Game para poder cambiar de pantalla. Agora mesmo, o control está na clase PantallaXogo, no método render. Ó cabo dun tempo quereremos cambiar de pantalla e acceder á pantalla principal, ou a de HighScores ou outra calquera. Como vimos, deberemos chamar ó método setScreen da clase Game, pero isto agora mesmo só o podemos facer dende a clase MeuXogoGame que é a que deriva de Game. Como podemos chamar ó método setScreen dende a clase PantallaXogo ?

Unha forma de facelo é pasando ó constructor da clase PantallaXogo o obxecto que deriva da clase Game da seguinte forma.

Código da clase PantallaXogo:

```
public class PantallaXogo implements Screen {

        private boolean pause;
private boolean finxogo;
private boolean sair;

private MeuXogoGame meuxogogame;

public PantallaXogo(MeuXogoGame meuxogogame){
this.meuxogogame=meuxogogame;
}

        ..............
}
```

Código da clase MeuXogoGame:

```
public class MeuXogoGame extends Game {

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

setScreen(new PantallaXogo(this));
}

@Override
public void dispose(){
super.dispose();
Utiles.imprimirLog("MeuXogoGame", "DISPOSE", "DISPOSE");
}
}
```

Agora podemos dende a clase PantallaXogo facer uso do obxecto *meuxogogame* e chamar ó método setScreen para cambiar de pantalla...

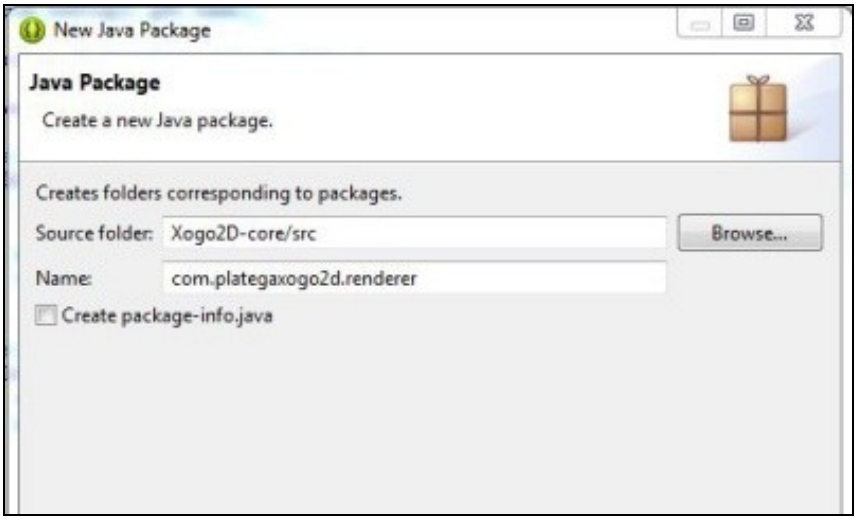
TAREFA A REALIZAR: Facer o mesmo que na PantallaXogo e crear o contructor para pasarlle un obxecto da clase MeuXogoGame ás pantallas:

PantallaPresentacion.
PantallaMarcadores.
PantallaPause.

Agora imos crear un novo paquete de nome **com.plategaxogo2d.renderer** onde van ir as clases necesarias para debuxar cada unha das pantallas. Poderíamos facer unha clase Renderer para cada pantalla pero normalmente a complexidade das pantallas (menos a do xogo) é mínima polo que normalmente só o faremos sobre á do xogo (RendererXogo).

Nota: Loxicamente poderíamos aproveitar as posibilidades da programación orientada a obxectos e facer unha clase con todo o común e herdar dela, pero como estamos a facer unha primeira aproximación a como funciona o framework isto xa quedaría para máis adiante unha vez que teñades soltura no manexo do framework.

Igual que fixemos antes, crearemos un paquete que terá de nome com.plategaxogo2d.renderer. Para crear o paquete prememos o botón dereito sobre o cartafol 'src' do proxecto Xogo2D-Core e escollemos a opción New => Package.



- Dentro de dito paquete creamos unha clase de nome **RendererXogo**.

Esta clase é a que vai a debuxar todos os elementos gráficos do xogo. Poderíamos facelo na mesma clase PantallaXogo ? Si. Por que non o facemos ? Por facilidade á hora de manter o xogo. Desta forma imos separar a parte de 'Control' da parte de 'Visualización' ou 'Render'.

Lembrar que ata o visto ata aquí, agora mesmo o control do programa se atopa no método render da clase PantallaXogo. É esta clase a que vai recibir o control do programa, a que ten o método resize que se chama de forma automática cando se cambia o tamaño,o método dispose que o temos que chamar nos... o que imos facer será chamar ós métodos da clase RendererXogo que imos usar: dispose, render e resize.

- Agora temos que chamar a un método da clase RendererXogo de forma continua. Creamos por tanto un método en dita clase. Imos chamarlle render e vai levar un parámetro de tipo float de nome delta (xa falaremos para que serve).

- Imos definir un método de nome dispose no que poñeremos o código necesario para liberar á memoria dos recursos utilizados pola clase RendererXogo.
- Imos definir un método resize que levará dous parámetros (width e height de tipo int) no que modificaremos o tamaño da cámara se é necesario(o veremos posteriormente).

Código da clase RendererXogo:

```
package com.plategaxogo2d.renderer;

public class RendererXogo {

/**
         * Debuxa todos os elementos gráficos da pantalla
         * @param delta: tempo que pasa entre un frame e o seguinte.
         */
public void render(float delta){
```

```
}
public void resize(int width, int height) {


}
public void dispose(){

}
}
```

Agora dende a PantallaXogo crearemos un obxecto de dita clase e chamaremos ós métodos *render*, *dispose* e *resize*.

Código da clase PantallaXogo:

```
package com.plategaxogo2d.pantallas;

import com.badlogic.gdx.Screen;
import com.plategaxogo2d.angel.MeuXogoGame;
import com.plategaxogo2d.renderer.RendererXogo;

public class PantallaXogo implements Screen {
private boolean pause;
private boolean finXogo;
private boolean sair;

private MeuXogoGame meuxogogame;
private RendererXogo rendererxogo;

public PantallaXogo(MeuXogoGame meuxogogame){
this.meuxogogame=meuxogogame;
rendererxogo=new RendererXogo();
}

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

rendererxogo.render(delta);
}

@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
}

@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

rendererxogo.dispose();
}

}
```

Neste punto podedes facer unha copia de todos os proxectos e así tedes unha copia para restaurar ademais de ter unha base para empezar outro xogo. Isto xa o explicamos neste enlace.

-- Ángel D. Fernández González -- (2014).