LIBGDX Scene

UNIDADE 3: Scene2D

Sumario

Introdución

Información na wiki: https://github.com/libgdx/libgdx/wiki/Scene2d

Clase utilizadas:

- Clase Stage
- Clase Actor

Xa vimos nun punto anterior o uso do Scene2D pero aplicado ó deseño de interface para o usuario.

Nota: É importante ter lido o punto anterior xa que imos poñer código xa explicado no mesmo.

Neste parte imos ver como aplicar o Scene2D pero aplicado o desenvolvemento dun xogo (sobre as personaxes que integran o xogo).

O Scene é outra forma de xestionar os elementos gráficos nun xogo. Basease no uso de Actor´s. Cada Actor ven ser un elemento gráfico do xogo sobre o cal queremos ter algún control. Permite debuxar e xestionar o control do elemento dentro da propia clase Actor (mestura Modelo-Vista-Controlador). Fai que o seu control sexa moi sinxelo e como característica que a min máis me gosta, ten a posibilidade de asociar a cada Actor unha serie de accións. As accións poden afectar ó seu aspecto (deformalo, cambiarlle a cor,....) como a súa posición,... Ditas accións pódense programar para que vaian unha detrás doutra, en paralelo, que se executen despois dun tempo, que se repitan....

Scene e o viewport

Información adicional: https://github.com/libgdx/libgdx/wiki/Viewports

Ó igual que sucedía cando deseñamos un xogo, imos necesitar o uso dunha cámara.
O Stage incorpora a súa propia cámara na que podemos definir o viewport.
Dende hai pouco, podemos determinar o aspecto que vai manter o viewport cando hai un cambio de resolución ou se visualizamos o xogo nun dispositivo cunha resolución diferente á deseñada.

Nota: Lembrar que xa vimos os problemas de manter a relación de aspecto neste punto.

No enlace anterior ven perfectamente explicado os diferentes viewports e o efecto que podemos conseguir con cada un.

O proceso é o seguinte:

- Creamos o tipo de viewport que queremos e o asinamos o Stage.

```
private Stage stage;
```

```
@Override
public void create () {

FitViewport fitViewPort = new FitViewport(480,800);
stage = new Stage();
stage.setViewport(fitViewPort);
```

Nota: Có new FitViewport estamos a crear unha nova cámara ortográfica.

- Agora actualizamos o viewport de acordo á resolución do dispositivo, pero como estamos a utilizar un Viewport que mantén a proporción de aspecto, aparecerán barras no lateral en caso de aumentar de tamaño a pantalla.

```
@Override
public void resize(int width, int height) {
// TODO Auto-generated method stub
stage.getViewport().update(width, height, true);
}
```

Nota: Dependendo do xogo, o terceiro parámetro será false, xa que poñendo true facemos que a cámara se sitúe no centro da pantalla.

Un exemplo completo:

- Deberedes descargar este arquivo e descomprimir o arquivo no cartafol assets da versión Android: Media:LIBGDX_stage2dui.zip

Cȩdigo da clase Stage2D
Obxectivo: Amosar o uso dun viewport no stage.

```
import com.badlogic.gdx.ApplicationAdapter;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.assets.AssetManager;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.g2d.TextureAtlas;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.ui.Label;
import com.badlogic.gdx.scenes.scene2d.ui.Skin;
import com.badlogic.gdx.scenes.scene2d.ui.TextButton;
import com.badlogic.gdx.utils.viewport.FillViewport;

public class Stage2D extends ApplicationAdapter {

private AssetManager assetManager;
private Skin skin;
private Stage stage;


@Override
public void create () {

assetManager = new AssetManager();
assetManager.load("uiskin.atlas",TextureAtlas.class);
assetManager.finishLoading();
TextureAtlas atlas = assetManager.get("uiskin.atlas", TextureAtlas.class);
skin = new Skin(Gdx.files.internal("uiskin.json"), atlas); // Cargamos os estilos

FitViewport fitViewPort = new FitViewport(480,800);
stage = new Stage();
stage.setViewport(fitViewPort);

Gdx.input.setInputProcessor(stage);

cargarElementosGraficos();


}

private void cargarElementosGraficos(){

Label titulo = new Label("Texto de exemplo",skin);
```

```
titulo.setColor(Color.RED);
titulo.setFontScale(2);
titulo.setBounds(0, 50, Gdx.graphics.getWidth(), 10);

TextButton boton = new TextButton("Premer aquí", skin);
boton.setPosition(0,100);
stage.addActor(boton);
stage.addActor(titulo);

}


@Override
public void render() {
Gdx.gl.glClearColor(1, 1, 1, 1);
Gdx.gl.glClear(GL20.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.getViewport().update(width, height, true);
}

@Override
public void dispose() {

Gdx.input.setInputProcessor(null);
assetManager.dispose();
skin.dispose();

stage.dispose();
}
}
```

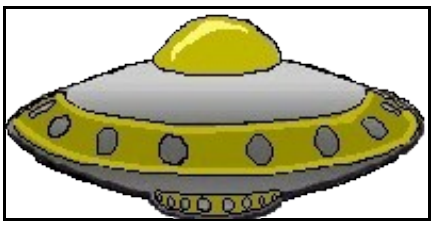
Actores

Unha vez temos decidido o tipo de cámara/relación de aspecto que queremos ter no xogo necesitamos engadir as personaxes ó Stage.

Cada personaxe en Stage denomínase Actor.

Un Actor terá información sobre a súa posición (getPosition/setPosition), tamaño (setSize/getSize) e textura asociada entre outra moita información. Ven ser a clase Modelo que estabamos a usar no desenvolvemento do xogo.

- Imos traballar con esta textura do xogo.



Descargádea e levala ó cartafol assets da versión Android.

Para engadir un Actor ó Stage primeiro temos que definilo.

```
public class Personaxe extends Actor{

private Texture textura;

Personaxe(){
```

```
textura = new Texture(Gdx.files.internal("LIBGDX_itin1_nave.png"));
setBounds(480, 0, 40,40);
}

     @Override
     public void draw(Batch batch, float alpha){
         batch.draw(textura,getX(),getY(),getWidth(),getHeight());
     }

     @Override
     public void act(float delta){
             super.act(delta);
     }
}
```

- Liña 1: Derivamos da clase Actor.
- Liña 6: Neste caso estamos a cargar a textura que vai debuxar.
- Liña 7: Indicamos a súa posición (480,0) e tamaño (60,60).

Ni que dicir ten que toda esta información poderíamosla pasar ó constructor cando instanciamos o obxecto.

Agora ben algo moi importante.

- Liñas 10-13: Este é o método que vai chamar o Stage de forma automática por cada Actor que teña. Lembrar que o facemos chamando ó método draw do Stage.

O mínimo que temos que poñer é:

```
batch.draw(textura,getX(),getY());
```

Pero se o facemos así, o tamaño da textura será o tamaño que teña ela orixinalmente. Non fará caso do width e height asinados na instrución setBounds. Se queremos que faga caso ó tamaño:

```
batch.draw(textura,getX(),getY(),getWidth(),getHeight());
```

Se queremos que faga caso á rotacións ou outro tipo de operacións sobre o debuxo da textura, teremos que facer uso da versión completa do método draw. public void draw (Texture texture, float x, float y, float originX, float originY, float width, float height, float scaleX, float scaleY, float rotation, int srcX, int srcY, int srcWidth, int srcHeight, boolean flipX, boolean flipY);

Como esta forma é bastante complexa, podemos facer uso dunha variante que consiste en utilizar como primeiro parámetro unha TextureRegion en vez dunha Texture. Lembrar que as TextureRegion están explicadas no punto TextureAtlas.

Se decidimos facelo desta forma, debemos de ter unha TextureRegion:

```
public class Personaxe extends Actor{

private TextureRegion tr;

Personaxe(){
Texture textura = new Texture(Gdx.files.internal("LIBGDX_itin1_nave.png"));
tr = new TextureRegion(textura);
setBounds(480, 0, 40,40);
}
 @Override
     public void draw(Batch batch, float alpha){
         batch.draw(tr, getX(), getY(), getWidth()/2, getHeight()/2, getWidth(), getHeight(),
               1, 1, getRotation());
     }
```

Os parámetro do método draw serán neste caso: textureregion, posX,posY,OriginX,OriginY, TamX, TamY, escalaX, escalaY, ángulo de rotación.

Porque teríamos que elixir este método ? Veremos posteriormente (e xa vimos no punto do Stage2D UI) que ós actores imos poder asociarlles accións. Se eu asocio unha acción que só mova a Texture/TextureRegion chegaría con poñer:

```
batch.draw(textura,getX(),getY(),getWidth(),getHeight());
```

Pero se eu asocio unha acción que leve consigo unha rotación da textura, necesitará implementar este último caso.

O veremos cun exemplo despois.

- Liñas 15-18: Aquí poñeríamos as instrucións para mover os nosos actores (o que fixemos no xogo no método update de cada personaxe).

Se imos asociar accións a estes actores **teremos que chamar ó método super.act**.

Exemplo de código

Imos ver un exemplo no que visualizaremos a nave.

Preparación:

- Descargade o seguinte arquivo e levalo ó cartafol assets da versión Android.

- Crear unha nova clase e cambiar os diferentes proxectos para que carguen dita clase.

Código da clase Stage2D_Actor
Obxectivo: Visualizar unha textura utilizando o Stage.

```
public class Stage2D_Actor extends ApplicationAdapter {

private Stage stage;

@Override
public void create () {

StretchViewport viewPort = new StretchViewport(480,800);
stage = new Stage();
stage.setViewport(viewPort);

Personaxe nave = new Personaxe();
stage.addActor(nave);

Gdx.input.setInputProcessor(stage);

}


@Override
public void render() {
Gdx.gl.glClearColor(0, 0, 0, 1);
Gdx.gl.glClear(GL20.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.getViewport().update(width, height, true);
}

@Override
public void dispose() {

Gdx.input.setInputProcessor(null);

stage.dispose();
}
}
```

```
public class Personaxe extends Actor{

private Texture textura;

Personaxe(){
textura = new Texture(Gdx.files.internal("LIBGDX_itin1_nave.png"));
setBounds(480, 0, 40,40);
}

 @Override
     public void draw(Batch batch, float alpha){
        batch.draw(textura,getX(),getY(),getWidth(),getHeight());
     }

     @Override
     public void act(float delta){
             super.act(delta);
     }
}
```

Accións nos Actores

Información na wiki: https://github.com/libgdx/libgdx/wiki/Scene2d#actions

É un dos elementos máis espectaculares que ten este tipo de programación.

O uso da clase Stage vainos permitir facer diferentes accións sobre os actors que o compoñen.

Dispoñemos de 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.

As accións poden aplicarse sobre cada Actor ou sobre o Stage completo.

Por exemplo, imos facer que a pantalla apareza toda negra e vaian aparecendo os compoñentes do Stage.

Para iso temos que utilizar as Accións: Actions.fadeOut e Actions.fadeIn.

```
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.

```
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.*;
```

Agora é cando entra en xogo o método draw utilizado na clase Personaxe.

Imaxinemos que queremos que a nave se mova de forma indefinida de esquerda a dereita (nun tamaño de pantalla fixo de 480x800 unidades).

A acción a engadir á nave será:

```
nave.addAction(Actions.forever(Actions.sequence(Actions.moveTo(0, nave.getY(),5),Actions.moveTo(480-nave.getWidth(),
nave.getY(),5))));
```

Como vemos temos:

- Actions.forever: Repite de forma indefinida a acción que ven a continuación.
- Actions.sequence: Executa na orde indicada o conxunto de accións que veñen a continuación.
- Actions.moveTo(0, nave.getY(), 5),Actions.moveTo(0, nave.getY(), 5): Move cara a coordenada (0,posición Y da nave) a nave cunha duración de 5 segundos e move cara posición (480-nave.getWidth(),posición Y da nave) cunha duración de 5 segundos.

Isto o vai poder facer xa que no método draw temos:

```
batch.draw(textura,getX(),getY(),getWidth(),getHeight());
```

```
Personaxe nave = new Personaxe();
nave.addAction(Actions.forever(Actions.sequence(Actions.moveTo(0, nave.getY(),5),Actions.moveTo(480-nave.getWidth(), nave.getY(),5)))
stage.addActor(nave);
```

Nota: Probádeo.

Agora imaxinemos que o mesmo tempo queremos rotar a nave 360 grados en 5 segundos.

A acción a engadir:

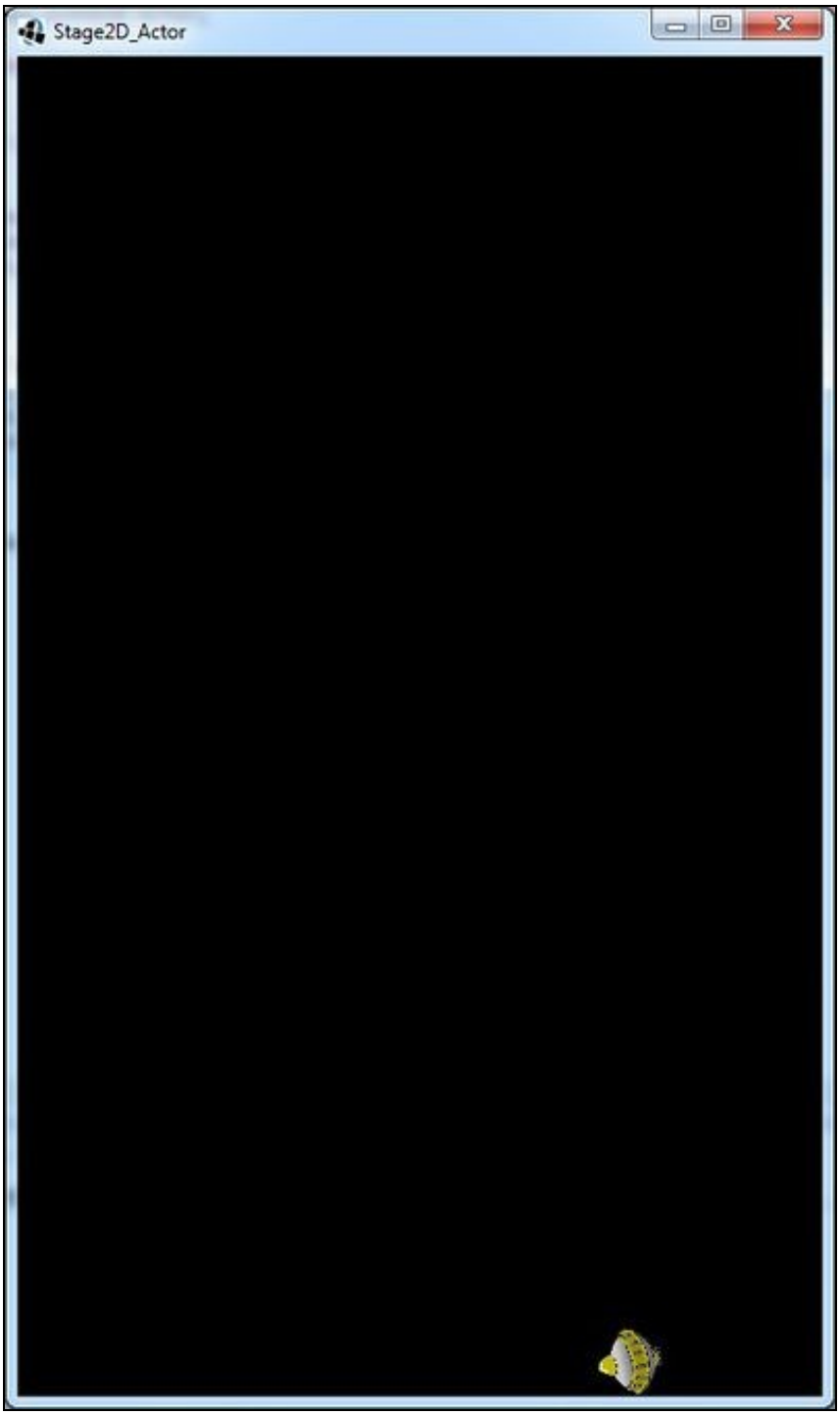
```
Personaxe nave = new Personaxe();
nave.addAction(Actions.forever(Actions.sequence(Actions.moveTo(0, nave.getY(),5),Actions.moveTo(480-nave.getWidth(), nave.getY(),5)))
nave.addAction(Actions.rotateBy(360, 5));
stage.addActor(nave);
```

Pero que vai pasar ? Que o método draw non ten o rotate polo que non fará nada (de feito da un null pointer exception)

Teremos que cambiar o método draw pola versión completa utilizando un TextureRegion:

```
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.scenes.scene2d.Actor;

public class Personaxe extends Actor{

private TextureRegion tr;

Personaxe(){
Texture textura = new Texture(Gdx.files.internal("LIBGDX_itin1_nave.png"));
tr = new TextureRegion(textura);
setBounds(480, 0, 40,40);
}

 @Override
     public void draw(Batch batch, float alpha){
         batch.draw(tr, getX(), getY(), getWidth()/2, getHeight()/2, getWidth(), getHeight(),
               1, 1, getRotation());
     }

     @Override
     public void act(float delta){
             super.act(delta);
     }
}
```

O resultado:


Stage2D_Actor

Xestión de Eventos

Introdución

O Stage xestiona os eventos que se producen entre os actores que o conforman.

No que se refire á xestión dos choques entre elementos, poderíamos xestionar os eventos como fixemos no desenvolvemento do xogo, facendo uso da clase Intersector como vimos anteriormente. Teríamos que percorrer todos os actores que conforman o Stage e comprobar se chocan entre eles ou con outros personaxes que non sexan Actor´s.

Se o facemos desta última forma, teremos que crear un obxecto da clase Rectangle dentro do Actor e actualizalo cando se mova no método act:

C¢digo da clase Personaxe
Obxectivo: Crear un obxecto da clase Rectangle para xestionar os choques.

```
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.scenes.scene2d.Actor;

public class Personaxe extends Actor{

private TextureRegion tr;
private Rectangle bounds;

Personaxe(){
Texture textura = new Texture(Gdx.files.internal("LIBGDX_itin1_nave.png"));
tr = new TextureRegion(textura);
setBounds(480, 0, 40,40);

bounds = new Rectangle();
}

 @Override
     public void draw(Batch batch, float alpha){
         batch.draw(tr, getX(), getY(), getWidth()/2, getHeight()/2, getWidth(), getHeight(),
               1, 1, getRotation());
     }

     @Override
     public void act(float delta){
             super.act(delta);
             bounds.set(getX(),getY(),getWidth(),getHeight());

     }

     public Rectangle getBounds(){
              return bounds;
     }


}
```

Como comentamos antes, o Stage xestiona os eventos que se producen nos actores.

```
@Override
public void create () {
                ..........
Gdx.input.setInputProcessor(stage);

}
```

- Se necesitamos xestionar os eventos da forma tradicional (cunha interface InputProcessor ou GestureListener) teremos que engadir as dúas formas como vimos neste apartado.

Á hora de xestionar os eventos veremos que os Actor´s levan consigo a Interface que vai xestionar o seu propio evento de Click.

Dita interface vai facer uso do método hit.

- Método hit: Dito método se chama de forma automática e controla cando un Actor é 'tocado' polo usuario premendo sobre a pantalla.

O código que ven por defecto neste método e para controlar unha forma rectangular. Se temos que controlar unha forma diferente (non rectangular), deberemos sobreescribilo e devolver o Actor en función de que se colisione ou non.

O método leva dous coordenadas que son as coordenadas no que o usuario preme a pantalla. Se o personaxe (actor) se toca, debemos facer un return do mesmo. En caso contrario debemos de devolver un null. Debemos comprobar se o actor en Touchable (se pode tocar) e se é visible.

Por exemplo:

```
// This hit() instead of checking against a bounding box, checks a bounding circle.
        public Actor hit(float x, float y, boolean touchable){
            // If this Actor is hidden or untouchable, it cant be hit
            if(!this.isVisible() || this.getTouchable() == Touchable.disabled)
                return null;

            // Get centerpoint of bounding circle, also known as the center of the rect
            float centerX = getWidth()/2;
            float centerY = getHeight()/2;

            // Calculate radius of circle
            float radius = (float) Math.sqrt(centerX * centerX +
                    centerY * centerY);

            // And distance of point from the center of the circle
            float distance = (float) Math.sqrt(((centerX - x) * (centerX - x))
                    + ((centerY - y) * (centerY - y)));

            // If the distance is less than the circle radius, it's a hit
            if(distance <= radius) return this;

            // Otherwise, it isnt
            return null;
        }
    }
```

Nota: Exemplo de código obtido de http://www.gamefromscratch.com/post/2013/12/11/LibGDX-Tutorial-3C-Scene-management.aspx

Un Actor só terá en conta os eventos se é clickable e é visible.

Podemos cambiar o estado dun Actor chamando ós métodos:

- setTouchable
- setVisible

Por exemplo:

```
nave.setTouchable(Touchable.disabled);
nave.setVisible(true);
```

Sendo nave un obxecto pertencente á clase Actor.

Interface

Unha vez temos un Actor engadido ó Stage, é clickeable e está visible podemos xestionar os eventos que se producen sobre el facendo uso das interfaces:

- Interface InputListener:

Nese enlace tedes todos os métodos / eventos que podemos xestionar.

Exemplos de eventos serían:

- Facer click sobre o actor.
- Arrastrar o rato (drag).
- Levantar o rato.

.....

Un exemplo:

```
public class Personaxe extends Actor{

        ..........
Personaxe(){
             ..........
xestionarEventos();
}
private void xestionarEventos(){
addListener(new InputListener(){
 public boolean touchDown(InputEvent event, float x, float y, int pointer, int buttons){
                    return true;
                }
                public void touchUp(InputEvent event, float x, float y, int pointer, int buttons){
                }
                public void touchDragged (InputEvent event, float x, float y, int pointer) {
                }
});

}
}
```

- Interface ActorGestureListener:

Exemplo de eventos que podemos capturar con dita interface temos:

- Pam.
- Zoom.
- Evento de premer durante un tempo longo.

Un exemplo:

```
public class Personaxe extends Actor{

        ..........
Personaxe(){
             ..........
xestionarEventos();
}
private void xestionarEventos(){

addListener(new ActorGestureListener(){
public boolean longPress(Actor actor, float x, float y){
                                Gdx.app.log("FIRE!!!", "PREMEMOS DURANTE UN TEMPO LONGO");
return true;
}
```

```
});

}
}
```

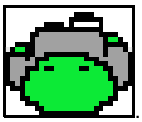
Exemplo de código

Neste exemplo imos ver como a nave se move de esquerda a dereita na parte superior e aparecerá no centro un alien.

Ó premer sobre o alien, este irá cara arriba. Se toca a nave esta se fará máis pequena.

Preparación:

- Descargade o seguinte arquivo e levalo ó cartafol assets da versión Android.

- Crear unha nova clase de nome Stage2D_Actor_2 e cambiar os diferentes proxectos para que carguen dita clase.

Código da clase Personaxe
Obxectivo: Modeliza a nave e o alien.

```
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;

public class Personaxe extends Actor{

private TextureRegion tr;
private Rectangle bounds;

Personaxe(Vector2 pos, String grafico){
Texture textura = new Texture(Gdx.files.internal(grafico));
tr = new TextureRegion(textura);


setBounds(pos.x,pos.y, 40,40);

bounds = new Rectangle();
}

 @Override
     public void draw(Batch batch, float alpha){
         batch.draw(tr, getX(), getY(), getWidth()/2, getHeight()/2, getWidth(), getHeight(),
               1, 1, getRotation());
     }

     @Override
     public void act(float delta){
             super.act(delta);
             bounds.set(getX(),getY(),getWidth(),getHeight());

     }


     public Rectangle getBounds(){
              return bounds;
```

```
    }


}
```

Desta clase non hai nada que explicar xa que é a usada nos exercicios anteriores. O único diferente é o constructor no que lle enviamos a posición nun Vector2 e o nome do gráfico a cargar.

Cȩdigo da clase AlienStage
Obxectivo: Representa o alien. É necesario crear unha clase xa que imos xestionar o evento de click sobre ela.

```
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.InputListener;
import com.badlogic.gdx.scenes.scene2d.actions.Actions;
import com.badlogic.gdx.scenes.scene2d.actions.MoveToAction;

public class AlienStage extends Personaxe{

private MoveToAction action;

AlienStage(Vector2 pos, String grafico){
super(pos,grafico);

xestionarEventos();
}

private void xestionarEventos(){
addListener(new InputListener(){
 public boolean touchDown(InputEvent event, float x, float y, int pointer, int buttons){
 action = Actions.moveTo(getX(), AlienStage.this.getStage().getViewport().getViewportHeight(), 3);
 addAction(action);
             return true;
            }
});

}

    @Override
    public void act(float delta){
              super.act(delta);

              if (getY()>=getStage().getViewport().getViewportHeight())
              setY(100);

              for (Actor actor : getStage().getActors()){
              if (actor.getName()=="nave") {
              Personaxe nave = (Personaxe)actor;
              if (nave.getBounds().overlaps(getBounds())){
              nave.setSize(nave.getWidth()-5, nave.getHeight()-5);
              removeAction(action);
              setPosition(240, 100);
              return;
              }
              }
              }
    }


}
```

- Liña 10: Imos gardar a acción do Alien de subir cando prememos sobre el. Isto vai ser necesario xa que cando choque coa nave debemos de eliminar dita acción.
- Liñas 19-25: Xestionamos o click sobre o alien. Nese caso engadimos unha acción que consiste en mover o alien cara arriba durante 3 segundos.
- Liñas 33-34: Se o alien chega á parte de arriba (viewportHeight) volve a colocarse na parte baixa.
- Liña 36: Percorremos o stage obtendo cada un dos actores do mesmo.

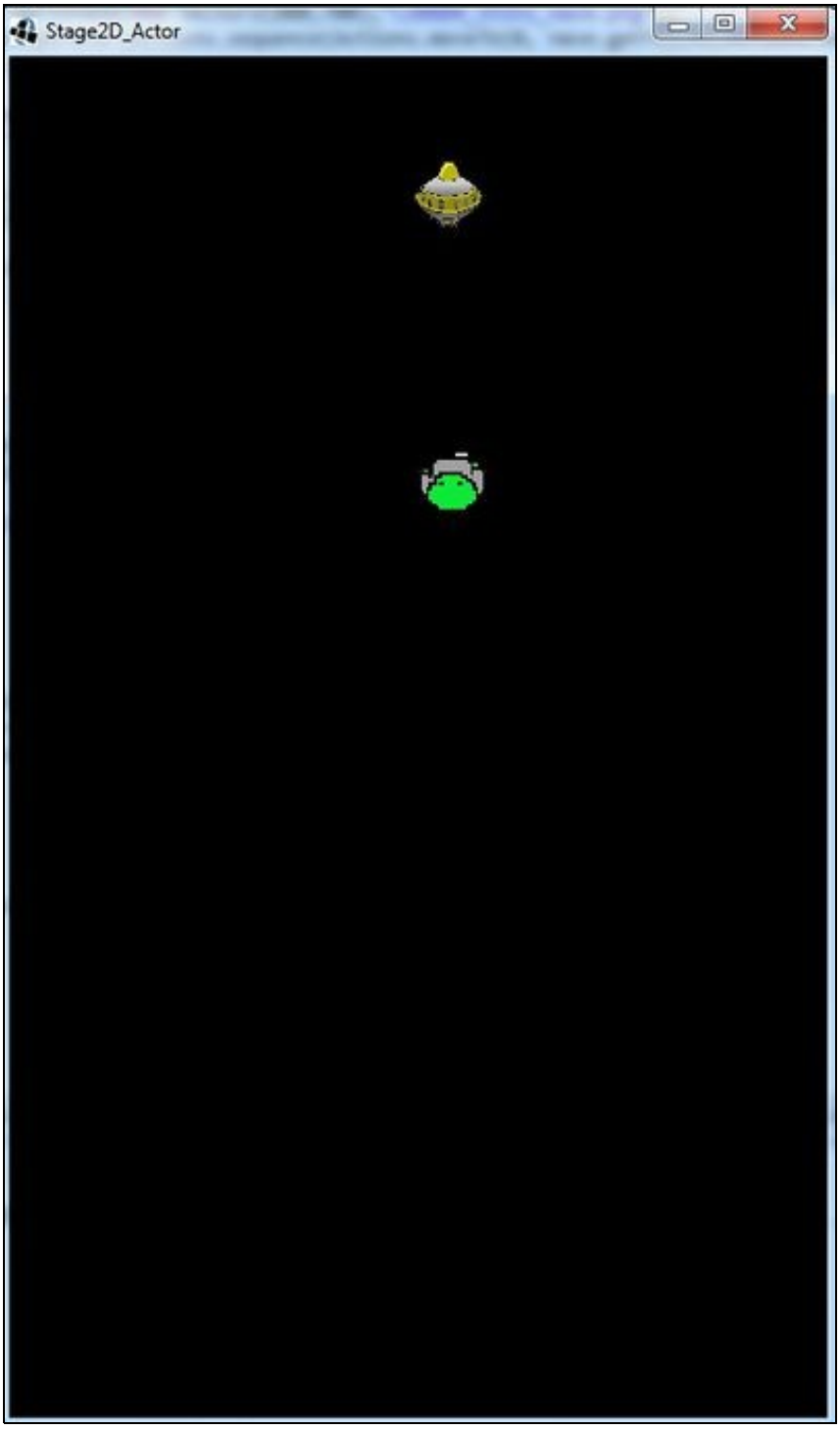
- Liña 37: Veremos despois que antes de engadir ó stage o actor, damos un nome para identificalo. Nesta liña comprobamos se o actor é a nave.
- Liña 38: Convertemos o actor e un obxecto da clase Personaxe.
- Liña 39: Comprobamos se chocan o Alien e a nave.
- Liñas 40-43: En caso de chocar modificamos o tamaño da nave e movemos ó alien a súa posición inicial eliminando a acción de mover cara arriba.

Código da clase Stage2D_Actor_2
Obxectivo: Xestiona todo o stage.

```
import com.badlogic.gdx.ApplicationAdapter;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.actions.Actions;
import com.badlogic.gdx.utils.viewport.StretchViewport;

public class Stage2D_Actor_2 extends ApplicationAdapter {

private Stage stage;


@Override
public void create () {

StretchViewport viewPort = new StretchViewport(480,800);
stage = new Stage();
stage.setViewport(viewPort);

Personaxe nave = new Personaxe(new Vector2(800,700),"LIBGDX_itin1_nave.png");
nave.addAction(Actions.forever(Actions.sequence(Actions.moveTo(0, nave.getY(),5),Actions.moveTo(480-nave.getWidth(), nave.getY(),5)))
nave.setName("nave");

AlienStage alien = new AlienStage(new Vector2(240,100),"LIBGDX_itin1_alien.png");
alien.setName("alien");

stage.addActor(nave);
stage.addActor(alien);

Gdx.input.setInputProcessor(stage);

}


@Override
public void render() {
Gdx.gl.glClearColor(0, 0, 0, 1);
Gdx.gl.glClear(GL20.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.getViewport().update(width, height, true);
}

@Override
public void dispose() {

Gdx.input.setInputProcessor(null);

stage.dispose();
}
```

```
}
```

O executar teremos como resultado isto:



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