

1 LIBGDX Mapas. Uso de tiled.

UNDIDADE 2: Mapas. Uso de Tiled

1.1 Sumario

- 1 Introducción
- 2 Instalación do editor de mapas
- 3 Creando un novo mapa
- 4 Creando o mapa
- 5 Utilizando o mapa
- 6 Exemplo de código
- 7 TAREFA OPTATIVA A FACER

1.2 Introducción

Información na wiki: <https://github.com/libgdx/libgdx/wiki/Tile-maps>

Nota: Esta sección non está relacionada directamente con ningunha das partes vistas no xogo. Esta indicado [neste enlace](#).

Neste punto imos explicar algúns aspectos básicos no desenvolvemento de mapas para xogos, tendo moitas máis posibilidades que as explicadas aquí.

Cando facemos o deseño dun xogo, nos atopamos coa dificultade de deseñar todo o escenario do mesmo.

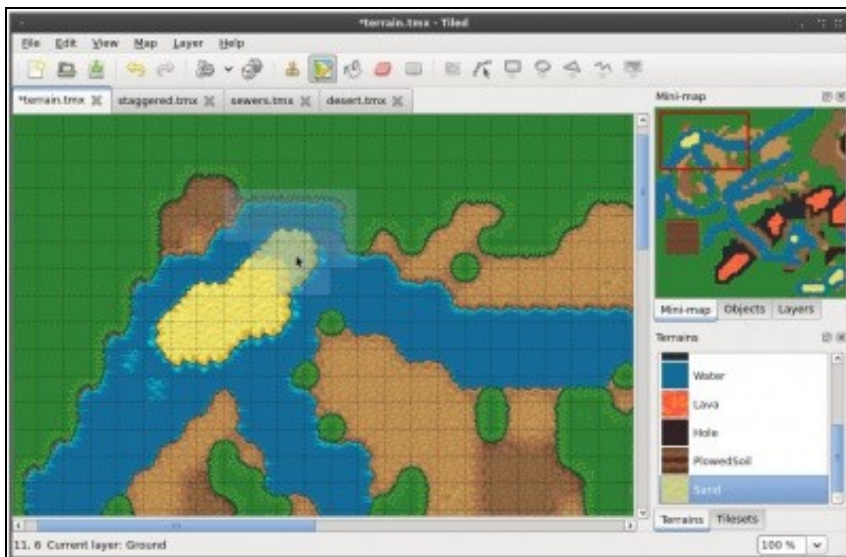
Se é pequeno ou ben o xogo transcorre nunha única pantalla non teríamos problema, pero se o xogo é de tipo 'scroll' cun escenario grande facer o deseño do mesmo 'a man' pode ser moi difícil.

Para solucionalo podemos facer uso dun editor de mapas. Existen varias ferramentas gratuítas para facelos.

- Nos imos usar o [MapEditor](#).

A idea deste tipo de editores é a de deseñar un mapa utilizando unha estrutura cuadrículada na que en cada cuadrícula imos situar un gráfico que pode ter unha serie de atributos (coma tagged) que imos poder ler dende o libgdx.

Nesta imaxe podemos ver un exemplo de deseño dun xogo utilizando dito editor:



Imaxen obtida de <http://www.mapeditor.org>.

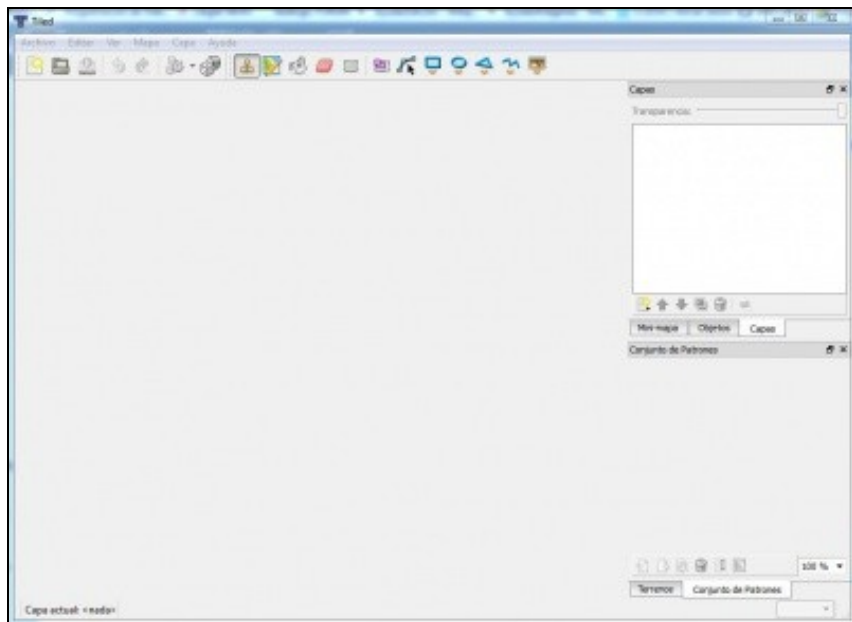
Como podemos observar na parte da dereita temos nun rectángulo vermello a parte do mapa que estamos a visualizar na parte central. Como podedes ver o mapa está dividido en cadrados e cada un deles é un gráfico posto por nos para conformar o mapa completo.

1.3 Instalación do editor de mapas

Descargamos o programa en función do S.O. premendo o botón *DOWNLOAD* da paxina principal.

Neste momento (Xullo do 2014) a versión descargada é a *0.9.1*.

Unha vez descargado e instalado o programa ó executalos aparecerá unha pantalla coma a seguinte:



1.4 Creando un novo mapa

- O primeiro é crear un novo mapa. Para iso prememos a opción de menú **Archivo** e escollemos a opción de **Novo Mapa**.



Nesta pantalla temos que cubrir:

- ◇ O tipo de perspectiva que queremos usar: No noso caso será Ortogonal. Poderíamos escoller isométrica. Tedes máis información sobre este tipo de vista [neste enlace](#).
- ◇ Formato de capa: Ven ser o algoritmo de compresión que utiliza. Deixamos o que ven por defecto.
- ◇ Tamaño de Mapa: Cada un dos cadrados que conforman o mapa denomínase patrón. O número de patróns polo tamaño de cada un danos o tamaño total do mapa en píxeles.

No noso caso imos a escoller **25 patróns de ancho e 30 patróns de alto**.
◊ Tamaño do patrón: O número de píxeles por cada patrón. Deixamos o valor por defecto.

Nota: Na parte de abaixo á esquerda aparece o tamaño do mapa en píxeles.

- Unha vez aceptado aparecerá unha pantalla dividida en cadrados (cada un dos patróns de 32x32 píxeles). Agora necesitamos cargar o gráfico con todas as formas que imos usar no desenvolvemento do mapa.

O gráfico a cargar é o seguinte:



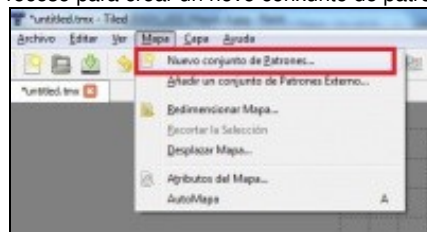
Imaxe obtida de <http://opengameart.org/content/consolidated-hard-vacuum-terrain-tilesets>

Autor "Hard Vacuum" art by Daniel Cook (Lostgarden.com)

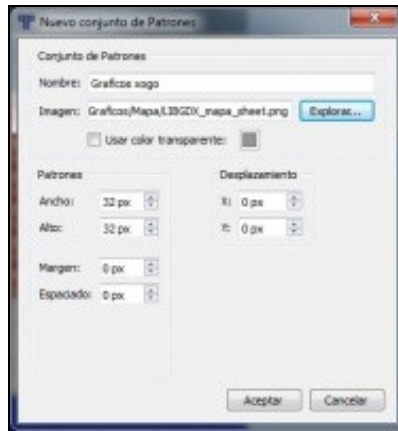
O proceso é o seguinte:

-
-

Proceso para crear un novo conxunto de patróns



Debemos escoller a opción de Menú Mapa e a opción **Novo conxunto de patróns**.



Dámoslle un nome, indicamos a imaxe descargada con todos os gráficos. Os demais valores os deixamos como están.



Agora na parte da abaixo á dereita aparece o gráfico cargado.

Explicacións adicionais:

- O que facemos neste paso é dividir o gráfico en anacos formando unha cadrícula. Os datos que nos piden son os referidos a cada anaco do gráfico. Antes o fixemos có Mapa.
- Pode ser que os gráficos que importedes para facer o mapa teñan unha cor transparente para certas partes. Por exemplo neste gráfico:



Imaxe obtida de <http://opengameart.org/content/golgotha-textures-tunnelroadjpg>

O que teríamos que facer é descubrir o valor desa cor nas compoñentes Red-Green-Blue utilizando o Gimp e escoller a opción 'Usar color transparente'. Premendo sobre a cor aparecerá unha nova ventá onde podemos asinar os valores da cor que queiramos que sexa transparente.

- O tamaño de cada patrón debe corresponderse co tamaño que indicamos cando creamos o mapa. Poden ter un valor diferente pero non é o normal.
- Se o gráfico a engadir tivera separacións entre as partes, teríamos que axustar o margin e espaciado para que se corresponda có que ven no gráfico.

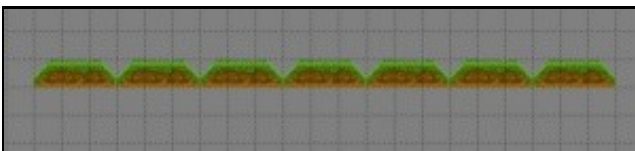
1.5 Creando o mapa

Agora chega o momento de arrastrar os patróns ó mapa.

- Podemos seleccionar varios patróns á vez premendo có botón dereito do rato e arrastrando o mesmo:








- Podemos duplicar ó seleccionado no mapa. Para facelo temos que, cun patrón seleccionado, ir ó mapa, premer a tecla SHIFT (maiusculas) e premer o botón esquerdo do rato sobre o mapa. Nese intre debuxarase ó seleccionado no conxunto de patróns. Agora, soltando o botón do rato PERO SEN SOLTAR A TECLA SHIFT, moveremos o rato e veremos como se duplica ó seleccionado.



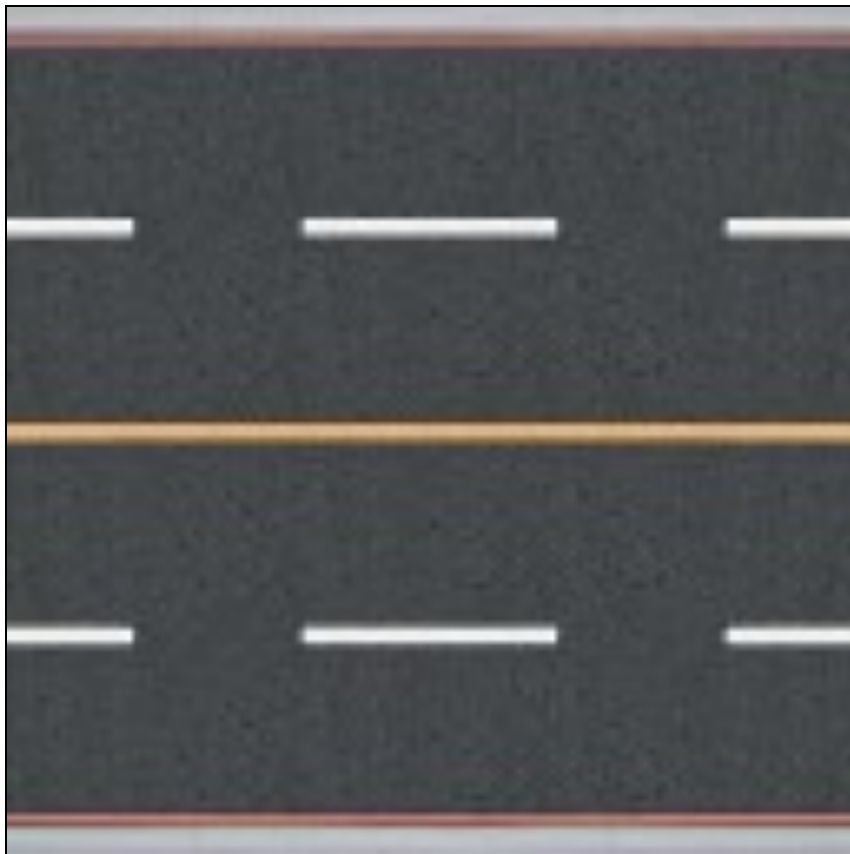
Exemplo de duplicación de patrón

- Podemos ter un fondo no noso mapa. Para encher todo o mapa dun patrón, debemos escoller a ferramenta de recheo. Na parte superior temos as seguintes:

Herramientas para las capas de patrones.	
	Brocha de estampar: A que utilizamos para facer o mapa.
	Brocha de terreo: Permite crear previamente certo conxunto de bloques para ser utilizados no terreo.
	Ferramente de recheo: Para encher o mapa cun patrón.
	Goma: Para borrar unha área específica.
	Selección: Permite seleccionar varios patróns no mapa de forma rectangular

Escollemos un patrón (por exemplo un que sexa todo verde) e coa ferramenta de recheo escollida prememos sobre o mapa:

No caso do xogo que estamos a desenvolver, imos engadir outro gráfico, que son as estradas, ó mapa:



Agora xa podemos desenvolver o mapa do xogo.

Estamos a ver só unha pequena parte do que se pode facer có editor de mapas.

- Así, podemos ter diferentes capas de patróns. Cada capa estará sobre a outra, de tal forma que teremos que engadir aqueles patróns que vaian diante máis arriba na xerarquía de capas.

Nota: Para mover unha capa temos que marcala e premer a frecha arriba-abaixo da ventá de capas.

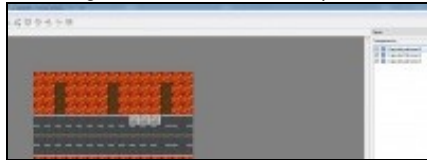


Neste exemplo temos 3 capas de patróns. O capa 2 é a capa que contén as 3 caras do gráfico. Están arriba da capa 1 que é a capa que contén as estradas, lavas,... Na capa de patrón 3 hai uns gráficos pero non se ven porque están despois da capa 2.

- Cada patrón pode ter uns atributos. Despois con programación podemos comprobar os atributos de cada cela e obrar en consecuencia.

-
-

Engadindo un atributo a un patrón



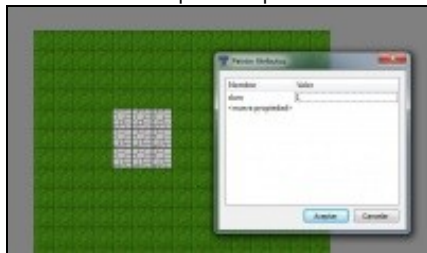
No exemplo imos engadir un novo atributo á pedra.

-



Escollemos o elemento do patrón e prememos o botón dereito.

-



Engadimos o atributo duro = 1.

- Tamén podemos engadir obxectos e darlles atributos, controlando despois por programación.

Para isto temos que engadir unha capa de obxectos.

1.6 Utilizando o mapa

- Paso 1)

Unha vez temos o mapa deseñado o gardaremos.

Isto vai crear un arquivo coa extensión tmx que se pode editar cun editor de textos. Temos que levar dito arquivo ó **cartafol assets xunto có gráfico/gráficos usados para facer o mapa.**

- **Paso 2)** Unha vez temos o mapa no cartafol assets da versión android necesitamos cargalo.

Para cargar o mapa debemos usar a **clase TmxMapLoader**.

```
private TiledMap map;
    .....
@Override
public void create() {
    map = new TmxMapLoader().load("mapa2.tmx");
}
```

Chamamos ó método load de dita clase mandando como argumento o nome do mapa que previamente gardamos no cartafol assets. A chamada a dito método devolve un obxecto da clase TiledMap que representa todo o mapa con todas as súas capas.

- **Paso 2)**

Unha vez temos o mapa necesitamos 'renderizalo'. Para facelo necesitamos utilizar a **clase OrthogonalTiledMapRenderrer**.

```
private OrthogonalTiledMapRenderrer rendererMapa;
private TiledMap map;
    .....
@Override
public void create() {
    map = new TmxMapLoader().load("mapa2.tmx");
    rendererMapa = new OrthogonalTiledMapRenderrer(map);
}
```

- **Paso 3)**

Necesitamos utilizar unha cámara para indicarlle á clase anterior que utilice as matrices da cámara para debuxar o mapa (matriz de proxección).

A diferenza do xogo desenvolto por nos, no que inventábamnos unhas unidades ficticias e facíamos que os gráficos se debuxaran dentro desas coordenadas ficticias, neste caso se o mapa mide 480 pixeles de ancho, por exemplo, teremos que indicar que a cámara teña 480 de viewportwidth para que visualice todo o ancho do mapa. É dicir, a cámara vai seguir traballando con unidades, pero ten que pasar que o ancho da cámara debe coincidir co ancho do mapa se queremos que se visualice todo o ancho do mesmo. En caso contrario (se facemos que a cámara teña un ancho menor) visualizaremos só unha parte do mapa, tanto de ancho como de alto, facendo un efecto zoom.

```
private OrthographicCamera camara;
    .....
@Override
public void resize(int width, int height) {
    // TODO Auto-generated method stub
    camara.setToOrtho(false, 480, 800);
    camara.position.set(240, 400, 0);
    camara.update();
    rendererMapa.setView(camara);
}
```

- Liña 9: Chamamos ó método setView da clase OrthogonalTiledMapRenderrer para que renderice o mapa de acordo ós parámetros da cámara.

Nota: Pódese poñer width e height do método resize como parámetros da cámara, pero se a resolución é mais ancha ou máis alta que o tamaño do mapa visualizaríamos de máis (espazo en branco).

- **Paso 4)**

Agora necesitamos chamar dentro do método render da nosa clase, ó método render da clase OrthogonalTiledMapRenderer.

```
@Override
public void render() {
    Gdx.gl.glClearColor(1, 1, 1, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

    rendererMapa.setView(camara);
    rendererMapa.render();

}
```

• Paso 5)

Liberamos a memoria.

```
@Override
public void dispose() {
    rendererMapa.dispose();
    mapa.dispose();

}
```

• Paso 6)

Se queremos acceder ás propiedades do mapa, como por exemplo os atributos da celas, primeiro debemos acceder á capa do mapa (lembrar que podemos ter moitas capas diferentes). Para iso debemos usar a [clase TiledMapTileLayer](#):

Cargamos a capa do mapa:

```
private TiledMapTileLayer capaXogo;
    .....
@Override
public void create() {
    .....

    capaXogo = (TiledMapTileLayer)mapa.getLayers().get(0);
}
}
```

Agora xa podemos acceder as celas individuais chamando ó método `getCell` que devolve unha cela e despois ó método `getTile` que devolve o `Tile`:

```
capaXogo.getCell(x,y).getTile().getProperties()
```

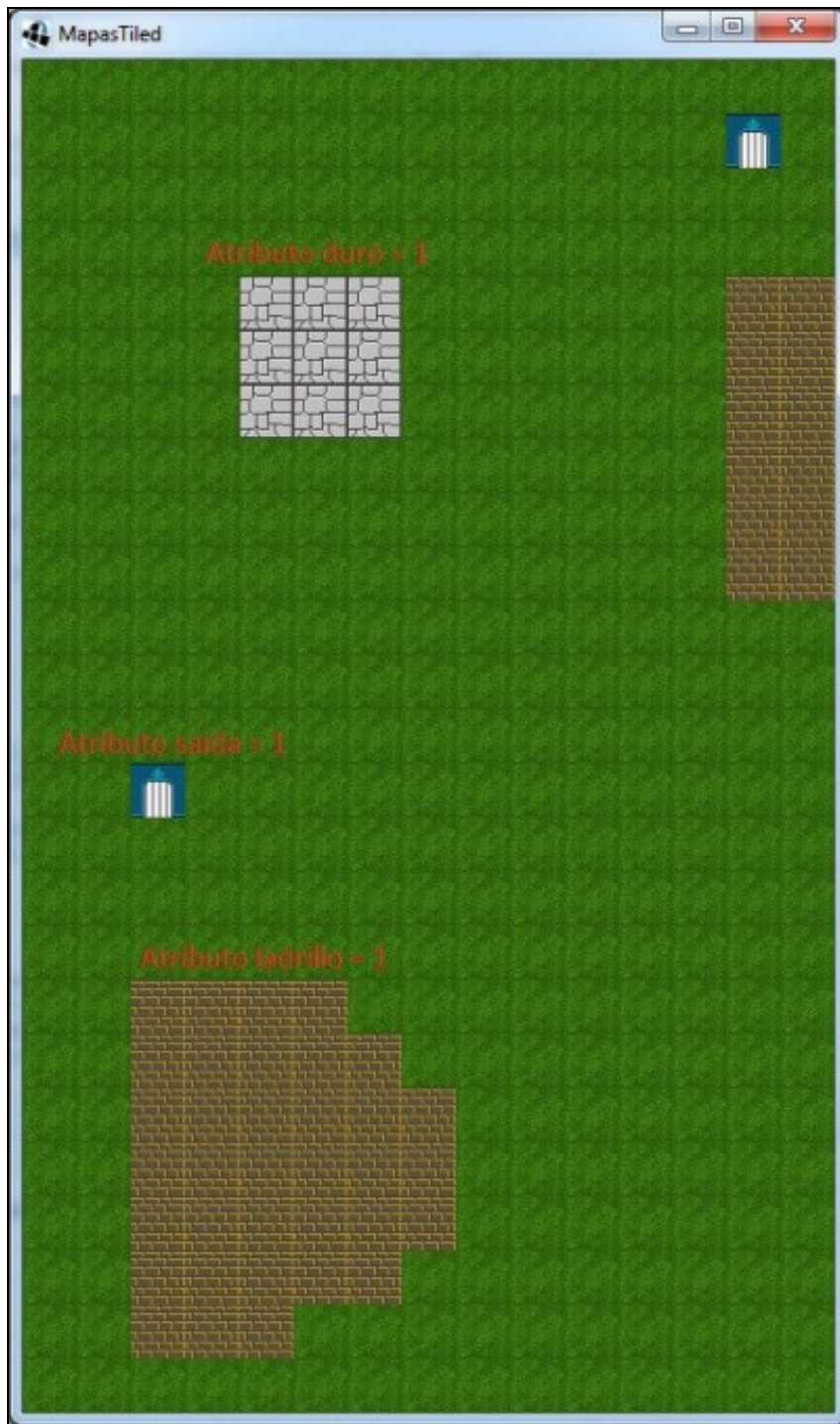
O máis complicado é pasar das coordenadas de pantalla ás coordenadas do mapa que son o número de patróns definidos no mapa:



A idea é a seguinte: Se estou na posición 320,160 en píxeles e sei que cada patrón ocupa 32 bits => $320/32,160/32$ => 10,5. Estou na cela 10,5 do mapa.

1.7 Exemplo de código

Neste exemplo vaise cargar un mapa que ten elementos con atributos. Iremos ler ditos atributos e indicar cando se 'tocan'.



O mapa mide 15x30 patróns e cada patrón ten un tamaño de 32 píxeles. Por tanto temos un tamaño para o xogo de 480x960 píxeles.

Como comentamos antes, o ancho da cámara vai ser de 480 para que non teñamos que facer scroll no ancho e terá un alto de 800. Así facemos coincidir cun dos tamaños habituais dos dispositivos Android.

Isto vai levar consigo que poidamos ter un scroll hacia arriba de 160 píxeles ($800 + 160 = 960$).

No exemplo imos incorporar a interface `InputProcessor` para que cando prememos sobre o mapa indique se tocamos algúns dos elementos con atributos. En caso contrario intentará facer scroll ata o máximo posible para que só se vexa o mapa.

Preparación:

- Descargade o seguinte arquivo e descomprimídeo no cartafol assets da versión Android. [Media:LIBGDX Mapa.zip](#).
- Crear unha nova clase e cambiar os diferentes proxectos para que carguen dita clase.

Código da classe MapasTiled

Objetivo: Amosar como funciona un TiledMap.

```
import com.badlogic.gdx.ApplicationAdapter;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputProcessor;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.maps.tiled.TiledMap;
import com.badlogic.gdx.maps.tiled.TiledMapTileLayer;
import com.badlogic.gdx.maps.tiled.TmxMapLoader;
import com.badlogic.gdx.maps.tiled.renderers.OrthogonalTiledMapRenderer;
import com.badlogic.gdx.math.Vector3;

public class MapasTiled extends ApplicationAdapter implements InputProcessor{

    private TiledMap map;
    private OrthogonalTiledMapRenderer renderMap;
    private OrthographicCamera camara;

    private TiledMapTileLayer capaXogo;

    @Override
    public void create() {

        camara = new OrthographicCamera();

        map = new TmxMapLoader().load("mapa2.tmx");
        renderMap = new OrthogonalTiledMapRenderer(map);

        capaXogo = (TiledMapTileLayer)map.getLayers().get(0);

        Gdx.input.setInputProcessor(this);

    }

    @Override
    public void render() {
        Gdx.gl.glClearColor(1, 1, 1, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        renderMap.setView(camara);
        renderMap.render();

    }

    @Override
    public void resize(int width, int height) {
        // TODO Auto-generated method stub
        camara.setToOrtho(false, 480, 800);
        camara.position.set(240, 400, 0);
        camara.update();

        renderMap.setView(camara);
    }

    @Override
    public void dispose() {
        renderMap.dispose();
        map.dispose();

        Gdx.input.setInputProcessor(null);

    }

    @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
Vector3 pos = new Vector3(screenX,screenY,0);

camara.unproject(pos);

Gdx.app.log("Punto unproyect:", String.valueOf(pos));
Vector3 postiled = new Vector3(pos.x / capaXogo.getTileWidth(),pos.y / capaXogo.getTileHeight(),0);
Gdx.app.log("TILED:", String.valueOf(postiled));

if (capaXogo.getCell(((int)postiled.x), ((int)postiled.y)).getTile().getProperties().containsKey("duro")){
Gdx.app.log("TILED","CHOCA");
return false;
}
if (capaXogo.getCell(((int)postiled.x), ((int)postiled.y)).getTile().getProperties().containsKey("saida")){
Gdx.app.log("TILED","SAIDA");
return false;
}
if (capaXogo.getCell(((int)postiled.x), ((int)postiled.y)).getTile().getProperties().containsKey("ladrillo")){
Gdx.app.log("TILED","LADRILLO");
return false;
}

if (pos.y < camara.viewportHeight/2)
camara.position.set(240,camara.viewportHeight/2,0);
else
if (pos.y > (560))
camara.position.set(240,560,0);
else
camara.position.set(240,pos.y,0);
camara.update();

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;
}

```

}

}

- Liña 91: Nesta liña pasamos das coordenadas da pantalla á coordenadas do xogo.

Isto necesita unha explicación. Se a cámara non se move, por defecto, o tamaño do viewport é de 480x800. A cámara está situada no centro (posición 240,400). Se cambiamos de resolución, axustará a nova a estes tamaños. Se facemos o unProyect dun punto calquera daranos un punto entre 480 e 800 unidades.

Se movemos a cámara cara arriba (lembrar que temos 160 pixeles/unidades de diferenza) a cámara estará na posición 240,560, pero a área de visualización da cámara (o viewport) non se modifica e polo tanto cando premamos daranos en coordenadas de pantalla un punto entre 480,800. O que fai o unProyect é darnos o punto con respecto o seu tamaño total (960 de altura).

- Liña 93: Unha vez temos o punto debemos dividilo polo tamaño do patrón para darnos a coordenada x,y do mapa (dividimos polo tamaño de cada cela para obter a cela no que nos atopamos).
- Liñas 97,101 e 105: Accedemos a cela para obter o atributo que ten asociado e amosar unha mensaxe en función do tipo de atributo.
- Liñas 110-117: Movemos a cámara cara arriba / abaixo ata os límites de 560 unidades (parte superior) e 400 unidades (parte inferior).

1.8 TAREFA OPTATIVA A FACER

- Modifícase este mapa do xogo dándolle un alto máis grande (agora mesmo ten 20 patróns = 800 pixeles): [Media:Libgdx_Mapaxogo.zip](#)

Para modificar o tamaño do mapa tedes que ir có MapEditor e escoller a opción do menú Mapa => Redimensionar Mapa.

Levade a parte das plataformas da nave espacial arriba de todo e engadide novas partes ó mapa.

Modifícase o código do xogo para engadir novas seccións (camiños de terra, novos ríos,... que cubran o novo tamaño.

Como tarefa optativa se pode engadir obxectos con atributos e facer que se se tocan o alien morra ou choque e non poida pasar por eles.

Lembrar que agora teremos un scroll. O máis sinxelo é facer que a cámara estea centrada sobre o alien que se irá movendo cara arriba, tendo en conta os límites claro.