1 LIBGDX Movendo os graficos

UNIDADE 2: Movendo os gráficos. A lóxica do noso xogo. O parámetro delta

1.1 Sumario

- 1 Creando as clases dos personaxes
- 2 A clase Mundo
- 3 Clase Array: Engadindo moitas personaxes ó noso mundo
    - 3.1 Clase Array: Definición
    - 3.2 Clase Array: Engadir novos obxectos
    - 3.3 Clase Array: Recorrendo o array
    - 3.4 Clase Array: Borrando un elemento do array
    - 3.5 Clase Array: obtendo o número de elementos do array
    - 3.6 Aplicando a clase Array ó noso xogo
- 4 Movendo os gráficos. O parámetro delta
- 5 Xestionando os diferentes tipos de elementos móviles
- 6 Tarefas avanzadas

1.1.1 Creando as clases dos personaxes

Como xa imaxinaredes, para mover os gráficos imos necesitar gardar a posición sobre cada un dos participantes do noso xogo. Desta forma imos poder movelos en función dunha velocidade.

O proceso de gardar información sobre cada personaxe do noso xogo é o seguinte:

- Definimos a clase que vai servir para gardar dita información.
- Na clase Mundo do paquete 'com.plategaxogo2d.modelo' creamos os obxectos que van facer uso da información gardada.
- Na clase RendererXogo utilizamos o obxecto definido na clase Mundo para debuxar os nosos personaxes.

Esta forma de facelo non é obrigatoria, pero serve para separar os datos (modelo) do renderizado (vista).

Ó longo deste curso se dará a opción de desenvolver o xogo seguindo 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...
- 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. O veremos máis adiante.

Seguimos có noso xogo...

Como todos os personaxes van ter unha información parecida, imos crear unha clase abstracta de nome **Personaxes** no paquete 'com.plategaxogo2d.modelo'.

Nota: Quen estea máis cómodo creando unha clase para cada personaxe é libre de facelo.

Dita clase terá as seguintes propiedades e métodos:

- Propiedades tamaño, velocidade, posición. De tipo Vector2.
- Métodos get e set de cada propiedade.
- Método update: modifica a posición en función da velocidade. Usará delta. Explicado posteriormente.

Código da clase Personaxe:
Clase que vai servir como base para os personaxes do noso mundo.

```
package com.plategaxogo2d.modelo;

import com.badlogic.gdx.math.Vector2;

public abstract class Personaxe {

/**
         * Constructor por defecto
         */
public Personaxe(){

}
/**
         * Instancia unha personaxe
         *
         * @param posicion
         * @param tamaño
         * @param velocidade_max
         */
public Personaxe(Vector2 posicion, Vector2 tamano, float velocidade_max) {
this.posicion = posicion;
this.tamano = tamano;
this.velocidade_max = velocidade_max;
}

/**
         * Velocidade que toma cando se move.
         */
public float velocidade_max;
/**
         * Velocidade actual
         */
protected float velocidade = 0;
/**
         * Posición
         */
protected Vector2 posicion;
/**
         * Tamaño
         */
protected Vector2 tamano;

/**
         * 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;
}

/**
        * Devolve o tamaño
        * @return tamano
        */
public Vector2 getTamano() {
return tamano;
}

/**
        * Modifica o tamano
        *
        * @param width: novo tamano de ancho
        * @param height: novo tamano de alto
        */
public void setTamano(float width, float height) {
this.tamano.set(width,height);
}

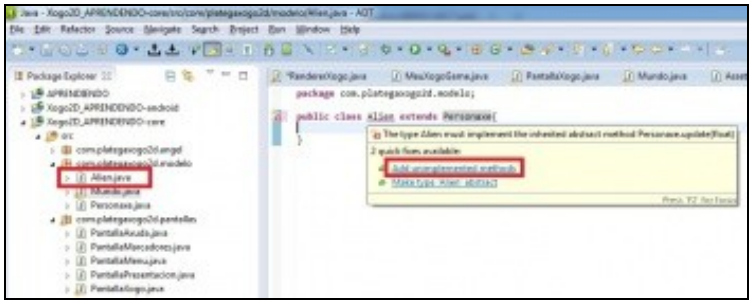
/**
        * Actualiza a posición en función da velocidade
        * @param delta: tempo entre unha chamada e a seguinte
        */
public abstract void update(float delta);
}
```

Agora engadir ó noso mundo os demais elementos móbiles.

- Comencemos polo **alien**.

Creamos polo tanto unha clase no paquete 'com.plategaxogo2d.modelo' de nome Alien e que derive da clase Personaxe creada anteriormente.

Ó facelo daranos un erro, xa que ó igual que cando usamos unha interface, será necesario desenvolver os métodos abstractos. Temos que situarnos enriba do nome da clase e escoller a opción **Add unimplemented methods** como indica a seguintes imaxe:

- Creamos a clase Alien
- 

Prememos sobre a opción Add unimplemented methods.

Código da clase Alien

```
public class Alien extends Personaxe{

public Alien(Vector2 posicion, Vector2 tamano, float velocidade_max) {
super(posicion, tamano, velocidade_max);
}

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

}

}
```

- **Nota**: A velocidade será explicada posteriormente cando expliquemos delta.

1.1.2 A clase Mundo

Todos os protagonistas do noso xogo teñen que estar instanciados na clase Mundo.

No noso caso imos crear o protagonista do noso xogo, o alien.

Código da clase Mundo:
Obxectivo: engadir un personaxe ó noso mundo.

```
public class Mundo {

 public static final int TAMANO_MUNDO_ANCHO=300;
 public static final int TAMANO_MUNDO_ALTO=500;

 private Alien alien;

 public Mundo(){
 alien = new Alien(new Vector2(100,20), new Vector2(15,15),100);
 }

 public Alien getAlien() {
         return alien;
 }
}
```

Sempre faremos estes tres pasos:

- Definir o obxecto: liña 6.
- Instanciar o obxecto no constructor da clase Mundo: liñas 8-10.
- Definir un método get xa que o imos necesitar para debuxar (clase RendererXogo) e controlar (clase ControladorXogo): liñas 12-14.

Agora é necesario que debuxemos o alien e aquí é onde entra outra vez o concepto modelo-vista-controlador.

O obxecto que vai gardar ó noso mundo con todos os seus personaxes vai estar *definido na clase PantallaXogo*. Polo tanto é necesario pasarlle dito obxecto á clase RendererXogo para que visualice as personaxes definidas na clase Mundo.

A forma de facelo será a seguinte:

Código da clase PantallaXogo :
Obxectivo: pasarlle á clase RendererXogo o obxecto da clase Mundo.

```
public class PantallaXogo implements Screen {

private Mundo meuMundo;
        .............

public PantallaXogo(MeuXogoGame meuXogoGame) {
```

```
meuMundo = new Mundo();

this.meuXogoGame = meuXogoGame;
rendererxogo = new RendererXogo(meuMundo);
}
        ............
```

Vos dará un error xa que aínda non está modificada a clase RendererXogo:

Código da clase RendererXogo :
Obxectivo: pasarlle á clase RendererXogo o obxecto da clase Mundo.

```
public class RendererXogo {

    private OrthographicCamera camara2d;
    private SpriteBatch spritebatch;

    private ShapeRenderer shaperender;
    private boolean debugger=true;

    private Mundo meuMundo;

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

        camara2d = new OrthographicCamera();
        spritebatch = new SpriteBatch();
        shaperender = new ShapeRenderer();
    }
    ................
}
```

Agora só queda debuxar o alien. **Código da clase RendererXogo :**
Obxectivo: Debuxar o alien instanciado na clase Mundo.

```
public class RendererXogo {
        ................
private void debuxarAlien(){
Alien alien = meuMundo.getAlien();
spritebatch.draw(AssetsXogo.textureAlien,
                        alien.getPosicion().x,alien.getPosicion().y,alien.getTamano().x,alien.getTamano().y);
}
        public void render(float delta) {
Gdx.gl.glClearColor(0, 0, 0, 1);
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

spritebatch.begin();

debuxarAlien();

spritebatch.end();

if (debugger) {
debugger();
}
    }
    ................
```

Mellora: Se imos a referenciar a un obxecto moitas veces pode ser mellor definilo no constructor e así non ter que facer uso da función getXXXX continuamente. Por exemplo:

```
public class RendererXogo {
    private Alien alien;
    .........
    public RendererXogo(Mundo mundo) {
this.meuMundo = mundo;
        alien = mundo.getAlien();
        .......
    }
```

```
}
```

Se probamos a executar o xogo podemos ver como o alien aparece debuxado.

TAREFA 2.4 A FACER: Esta parte está asociada á realización dunha tarefa.

1.1.3 Clase Array: Engadindo moitas personaxes ó noso mundo

Pasos previos: Antes de explicar como manexar moitos personaxes no xogo imos crear una nova clase que vai representar os elementos móbiles do xogo: pedras-troncos-coches. Creala no paquete Modelo.

```
public class ElementoMobil extends Personaxe{

public ElementoMobil(Vector2 posicion, Vector2 tamano, float velocidade_max) {
super(posicion, tamano, velocidade_max);
}
@Override
public void update(float delta) {
// TODO Auto-generated method stub

}

}
```

Nos xogos imos necesitar engadir un número indeterminado de elementos. No caso que estamos a desenvolver temos o caso dos elementos móbiles (troncos, coches,... pero podemos pensar en outros exemplos como os disparos, inimigos...

Para poder manexar estes elementos faremos uso dos arrays, pero xa implementados coma unha clase específica de LIBGDX.

Clase Array.

Métodos máis importantes:

- **Método public void add(T value)**: engade un novo elemento ó array.
- **Método contains(T value, boolean identity)**: informa se o elemento está no array.
- **public T get(int index)**: devolve o elemento do array indicado por index.
- **Método removeValue(T value, boolean identity)**: elimina un elemento do array.
- **items**: accede ós elementos do array

1.1.3.1 Clase Array: Definición

Esta clase sempre que teñamos un número non fixo de personaxes, será a elixida. O proceso para utilizala será o seguinte:

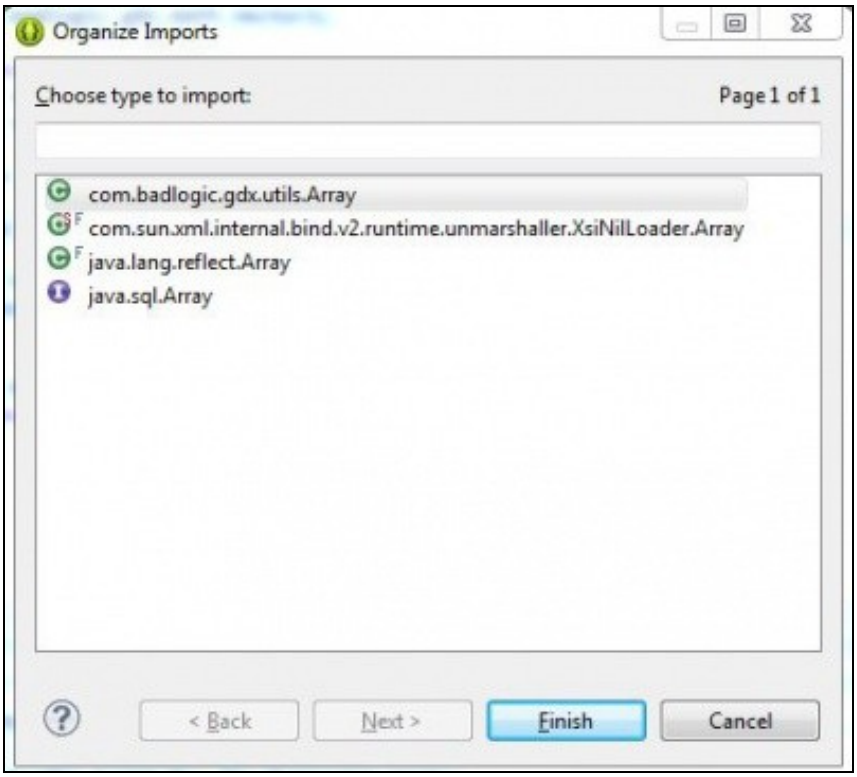
- Definir un obxecto desta clase indicando o tipo de clase que vai albergar (isto se indica cos caracteres < e >) da seguinte maneira:

```
private Array<ElementoMobil> vehiculos;
```

- Instanciamos o obxecto no constructor da clase:

```
vehiculos = new Array<ElementoMobil>();
```

Nota: Ó utilizar a clase Array vai dar un erro xa que non está importada. Cando a importedes (control +shift+O) vos dará varias opcións:



Deberedes de escoller **com.badlogic.gdx.utils.Array**.

1.1.3.2 Clase Array: Engadir novos obxectos

Debemos de usar o método add.

```
vehiculos.add(new ElementoMobil(new Vector2(10,400),new Vector2(20,15),65));
```

Neste exemplo, definimos o array vehiculos como un array de obxectos que van pertencer á clase ElementoMobil. Polo tanto será necesario crear un obxecto de dita clase. Os valores que lle enviamos xa os vimos anteriormente e son: posición (2 valores), tamaño (2 valores) e velocidade.

Podemos facer tantos add como necesitemos.

1.1.3.3 Clase Array: Recorrendo o array

Para percorrer un array podemos usar:

- A través do método Iterator

Exemplo da wiki: https://github.com/libgdx/libgdx/wiki/A-simple-game (buscar por Iterator)

Exemplo aplicado ó noso xogo:

```
Iterator <ElementoMobil> iter = vehiculos.iterator();
while(iter.hasNext()){
  ElementoMobil vehiculo = iter.next();
}
```

- A través dun for da forma:

```
for (ElementoMobil vehiculo : vehiculos){
  // vehiculo é cada un dos obxectos que están no array vehiculos.
}
```

1.1.3.4 Clase Array: Borrando un elemento do array

- Método removeValue(T value, boolean identity)

  ◊ Parámetro T value: é o obxecto a borrar.
  ◊ Parámetro boolean identity: se é true só compara os valores das propiedades do obxecto cos do array. Se é false compara que sexa o mesmo obxecto. Normalmente poñeremos true.

Exemplo de uso:

```
vehiculos.removeValue(vehiculo1, true);
```

Este método está sobrecargado polo que tamén podemos borrar por un índice,....

Por exemplo:

- Método public T removeIndex(int index)

Devolve o obxecto eliminado do array polo índice indicado.

1.1.3.5 Clase Array: obtendo o número de elementos do array

- **items**: accede ós elementos do array.

A través de dito array podemos obter o número de obxectos que temos no array da seguinte forma:

```
int num = vehiculos.items.length;
```

- **Método size**: devolve o número de elementos do array.

1.1.3.6 Aplicando a clase Array ó noso xogo

Exercicio Proposto:

Na clase Mundo:

- Definimos un array de coches que van ser todos os coches. Engadimos dous coches de proba ó array.
- Creamos un método getCoches que nos devolva dito array.

Na clase RendererXogo:

- Creamos un método debuxarCoches que percorra o array e debuxe as coches. Chamamos a dito método dende o método render.

Solución: Clase Mundo:

```
public class Mundo {
         ..............
 private Array<ElementoMobil>coches;

 public Mundo(){
 alien = new Alien(new Vector2(100,20), new Vector2(15,15),100);
 nave = new Nave(new Vector2(0,480),new Vector2(40,20),60f);

 coches = new Array<ElementoMobil>();
 coches.add(new ElementoMobil(new Vector2(10,400),new Vector2(20,15),65));
 coches.add(new ElementoMobil(new Vector2(40,400),new Vector2(20,15),65));
 }

public Array<ElementoMobil>getCoches(){
return coches;
}
```

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

Clase RendererXogo:

```
..............
private void debuxarCoches(){
for (ElementoMobil coche : meuMundo.getCoches()){
spritebatch.draw(AssetsXogo.textureCoche,coche.getPosicion().x,coche.getPosicion().y,coche.getTamano().x,coche.getTamano().y);
}
}

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

    /**
     * Debuxa todos os elementos graficos da pantalla
     *
     * @param delta
     *            : tempo que pasa entre un frame e o seguinte.
     */
    public void render(float delta) {
Gdx.gl.glClearColor(0, 0, 0, 1);
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

spritebatch.begin();

debuxarAlien();
debuxarNave();

debuxarCoches();

spritebatch.end();

if (debugger) {
debugger();
}
    }
    ..............
```

1.1.4 Movendo os gráficos. O parámetro delta

Chegou o momento de mover os gráficos. Como vos daredes conta mover os gráficos vai consistir en modificar a propiedade de posición.

Preparación: Agora ides facer unha copia da clase RendererXogo, xa que imos modificala para amosarvos como se poden mover os gráficos. Premede co rato sobre a clase, botón dereito e escollede a opción Copy. Despois repetides a operación pero escolledes a opción Paste. Vos preguntará un nome para a clase. Indicade **UD2_4_RendererXogo**.

Modificade a pantalla PantallaXogo para que chame a esta clase.

Pensaredes, pois moi fácil, poñemos este código no método render:

Código da clase UD2_4_RendererXogo:
Obxectivo:Mover os gráficos na pantalla.

```
private void debuxarCoches(){
for (ElementoMobil coche : meuMundo.getCoches()){
coche.setPosicion(coche.getPosicion().x+1, coche.getPosicion().y);

                        spritebatch.draw(AssetsXogo.textureCoche,
                                 coche.getPosicion().x,coche.getPosicion().y,
                                 coche.getTamano().x,coche.getTamano().y);
}
}
```

Pois estades equivocados. Atendede á seguinte explicación.

Primeiro temos que entender que representa **delta**: é o tempo que pasa entre unha chamada ó método e a seguinte chamada.

Lembrar que delta o temos como parámetro na clase PantallaXogo, no método render. Dito valor é o que estamos a pasar ó método render da clase RendererXogo.

De todas formas podemos obter delta en calquera parte do código chamando ó método **getDeltaTime()**:

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

Nota: A nivel de rendemento mellor pasalo como parámetro como facemos nos que estar a chamar á función continuamente.

Agora que xa temos claro que é imos ver como o podemos utilizar.

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 a un personaxe:

```
personaxe.setPosicion(personaxe.getPosicion().x+1, personaxe.getPosicion().y);
```

Como temos no noso xogo un ancho de 300 unidades.

Vai suceder que:

- **Máquina1:**

  60fps => móvese 60 unidades por segundo => Tarda 300/60 en percorrer toda a pantalla => 5 segundos.

- **Máquina2:**

  30fps => móvese 30 unidades por segundo => Tarda 300/30 en percorrer toda a pantalla => 10 segundos.

Polo tanto moveríase máis rápido na primeira máquina e non parece moi xusto :).

Como podemos facer para que o movemento sexa independente do hardware da máquina ? Pois multiplicando a velocidade por delta, sendo delta o tempo que tarda en chamar ó método de actualización da posición.

Por exemplo, seguindo o caso anterior:

- **Máquina1: a 60 fps. Quere dicir que chama 60 veces nun segundo ó método update.**

  Delta = 1/60
  posición.x = posición.x + (1 * delta) :multiplicamos a velocidade por delta. Estamos a poñer unha velocidade de 1 unidade por segundo.
  Lembrar que o noso mundo mide 500 unidades de ancho.

  Canto tempo tarde en percorrer o noso mundo ?

  Nun segundo percorre:

  60 (fps = veces por segundo) * 1/60 (é delta) * 1 (a velocidade) = 1.
  Polo tanto vai sumar unha unidade á posición x cada segundo.

  O noso mundo mide 300 unidades = 300 segundos.

- **Máquina2: a 30 fps. Quere dicir que chama 30 veces nun segundo ó método update.**

  Delta = 1/30
  posición.x = posición.x + (1 * delta) :multiplicamos a velocidade por delta. Estamos a poñer unha velocidade de 1 unidade por segundo.
  Lembrar que o noso mundo mide 300 unidades de ancho.

Canto tempo tarde en percorrer o noso mundo ?

Nun segundo percorre:

30 (fps = veces por segundo) * 1/30 (é delta) * 1 (a velocidade) = 1.
Polo tanto vai sumar unha unidade á posición x cada segundo.

O noso mundo mide 300 unidades = 300 segundos.

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

TAREFA 2.5 A FACER: Esta parte está asociada á realización dunha tarefa.

Cando implementamos un xogo a velocidade dos personaxes a imos obter da clase correspondente, pero tendo en conta que:

- ◊ O personaxe ten unha velocidade inicial que non cambia (caso coches)
- ◊ O personaxe ten unha velocidade dependendo da iteración do usuario co xogo (caso do Alien que só se moverá se o usuario preme sobre a pantalla).

Poderíamos introducir a velocidade na clase RendererXogo, pero se queredes ter unha forma de desenvolver o xogo 'limpa' e fácil de manter eu vos aconsello o seguinte punto. Modelo - Vista - Controlador.

1.1.5 Xestionando os diferentes tipos de elementos móviles

Como elementos móviles van ter diferentes texturas (gráficos), definimos na clase ElementoMobil unha propiedade de **tipo enum** que vai a gardar o tipo de elemento.

Modificaremos o constructor para pasarlle dita información cando creamos un elemento móvil.

Nota: Poderiamos utilizar unha propiedade de tipo numérica e definir: 1 (inimiga 1), 2 (inimiga 2),...pero desta forma queda máis claro:

Código da clase ElementoMobil:
Obxectivo: Engadimos unha propiedade de tipo enum para gardar o tipo de elemento (coche,autobus,tronco e pedra) e facemos o método get.

```
public class ElementoMobil extends Personaxe{

public static enum TIPOS_ELEMENTOS {COCHE, AUTOBUS, TRONCO, ROCA};
private TIPOS_ELEMENTOS tipo;

public ElementoMobil(Vector2 posicion, Vector2 tamano, float velocidade_max,TIPOS_ELEMENTOS tipo) {
super(posicion, tamano, velocidade_max);
setVelocidade(velocidade_max);
this.tipo=tipo;
}

public TIPOS_ELEMENTOS getTipo() {
return tipo;
}

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

setPosicion(posicion.add((velocidade*delta),0));
}
```

```
}
```

Tamén teremos que modificar o método render da clase RendererXogo para que teña en conta o tipo (coche-autobus) e escolla a textura adecuada.

Código da clase RendererXogo:
Obxectivo: Modificamos o método debuxarCoches

```
private void debuxarCoches(){
Texture textura=null;
for (ElementoMobil coche : meuMundo.getCoches()){
switch(coche.getTipo()){
case COCHE:
textura = AssetsXogo.textureCoche;
break;
default:
textura = AssetsXogo.textureAutobus;
break;
}
spritebatch.draw(textura,
                                        coche.getPosicion().x,coche.getPosicion().y,
                                        coche.getTamano().x,coche.getTamano().y);
}
}
```

Nota:: Poderíamos usar un único método para debuxar todos os elementos móbiles pero para iso teríamos que ter un único array con todos os elementos móbiles. Por motivos de simplificación teremos os elementos móbiles por tipo en arrays diferentes.

Posteriormente, no punto Creando o Mundo poñeremos unha solución máis escalable pero tamén máis complexa.

1.1.6 Tarefas avanzadas

TAREFA AVANZADA OPTATIVA: Esta parte está asociada á realización dunha tarefa avanzada. Animacións..

TAREFA AVANZADA OPTATIVA: Esta parte está asociada á realización dunha tarefa avanzada. Vector dirección..

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