LIBGDX As interfaces para capturar eventos

UNIDADE 2: Interface de xestión de eventos

As interfaces

Información na wiki: https://github.com/libgdx/libgdx/wiki/Event-handling

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

- InputProcessor: Xestiona os eventos de pulsación de teclas, pulsación sobre a pantalla (móbil), e arrastre do dedo pola pantalla (móbil)...
- 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)....Esta interface está explicada na Unidade 3: Xestión de Eventos: GestureListener.

Para engadir unha interface temos que utilizar a palabra implements na definición da clase igual que fixemos coa interface Screen. Ó facelo vos dará un erro sobre o nome da clase. Se situades o rato enriba do erro aparecerá unha opción de **Add unimplimented methods** que debemos escoller.

Código da clase PantallaXogo
Obxectivo: engadir unha interface de xestión de eventos.

```
public class PantallaXogo implements Screen,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;
}

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

Os eventos que controlamos con esta interface son:

- keyDown(): Tecla premida.
- keyUp(): Tecla liberada.
- keyTyped(): Tecla premida e xera un carácter unidade.
- touchDown(): O dedo prema a pantalla (móbil) ou o botón do rato é premido.
- touchUp(): O dedo deixa de premer a pantalla ou o botón do rato deixa de ser premido.
- touchDragged(): Dedo ou rato é arrastrado pola pantalla. No caso do rato o botón debe estar presionado.
- mouseMoved(): O rato se move pola pantalla e non se pulsa ningún botón.
- scrolled(): Cando se preme a roda de scroll do rato.

Como vemos todos os método devolven un boolean. Este é usado no caso de que teñamos diferentes interfaces de xestión de eventos no mesmo xogo e non queremos que o evento se traslade á seguinte interface (poñeríase return true). Normalmente o deixaremos por defecto.

Unha vez temos a interface temos que *informar* á nosa clase que os eventos teñen que ir a dita interface.

Isto o logramos chamando ó método Gdx.input.setInputProcessor(obxecto) indicando como obxecto o obxecto da clase que ten implementada a interface. No noso caso this:

- Gdx.input.setInputProcessor(this)

Onde temos que poñer dita orde ?

- O poñeremos nos métodos show e resume.

Cando acabamos de xestionar os eventos chamaremos ó mesmo método pero enviando null como obxecto:

- Gdx.input.setInputProcessor(null)

Onde temos que poñer dita orde ?

- O poñeremos nos métodos pause, hide e dispose (pode ser que non sexa necesario poñelo en hide se programamos que cando cambiemos de pantalla chamemos ó método dispose).

Código da clase PantallaXogo
Obxectivo: Informar á clase onde se atopa a xestión de eventos.

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

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

@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);
rendererxogo.dispose();
}
```

Agora xa temos todo preparado para xestionar os eventos. Empecemos coa parte máis sinxela (a versión PC) e faremos que cando prememos as teclas do cursor movamos algo.

Como estamos programando seguindo o MVC, teremos que informar á clase controladora que se pulsou unha tecla e actuar en consecuencia.

Como o facemos ?

Existen moitas alternativas. Eu vos vou expoñer a que aprendín.

Definimos na clase ControladorXogo un tipo de datos Enum cos diferentes controis que podemos ter sobre o noso xogo. Neste caso teremos as catro teclas dos cursores:

```
public enum Keys {
ESQUERDA,DEREITA,ARRIBA,ABAIXO
}
```

Despois definimos un hashmap e o inicializamos a false (lembrar importar a clase coa combinación de teclas Control+Shift+O).

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

E por último, para poder acceder a esta variable dende a clase PantallaXogo, faremos os método pulsarTecla e liberarTecla dentro desta clase ControladorXogo:

```
/**
         * 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 só queda chamar a estes métodos dende a clase PantallaXogo...

Código da clase PantallaXogo
Obxectivo: Informar á clase controladora da tecla pulsada (lembrar importar o paquete Input coa combinación de teclas Contro+Shift+O).

```
@Override
public boolean keyDown(int keycode) {
// TODO Auto-generated method stub
// Liberamos as teclas para que se arrastramos o dedo a outro control sen soltar o anterior non xunte o efecto
controladorXogo.liberarTecla(ControladorXogo.Keys.ABAIXO);
controladorXogo.liberarTecla(ControladorXogo.Keys.ARRIBA);
controladorXogo.liberarTecla(ControladorXogo.Keys.ESQUERDA);
controladorXogo.liberarTecla(ControladorXogo.Keys.DEREITA);

switch(keycode){
case Input.Keys.UP:
controladorXogo.pulsarTecla(ControladorXogo.Keys.ARRIBA);
break;
case Input.Keys.DOWN:
controladorXogo.pulsarTecla(ControladorXogo.Keys.ABAIXO);
break;
case Input.Keys.LEFT:
controladorXogo.pulsarTecla(ControladorXogo.Keys.ESQUERDA);
break;
case Input.Keys.RIGHT:
controladorXogo.pulsarTecla(ControladorXogo.Keys.DEREITA);
break;
}


return false;
}
```

Como vedes non ten moita complicación. Simplemente comprobamos que tecla se preme e informamos á clase controladora.

TAREFA 2.7 A FACER: Esta parte está asociada á realización dunha tarefa.

Exercicio proposto: Agora é necesario mover o alien. Isto o facemos na clase ControladorXogo modificando a súa velocidade en función da tecla pulsada. Vos atrevedes a facelo ?

Pistas:

- O alien ten unha mesma velocidade (non varía, non ten aceleración) pero pode moverse nos dous eixes (x/y). Teredes que crear na clase Alien os métodos necesarios para modificar dita velocidade en cada un dous eixes e obter dita velocidade.
- O método update terá que ter en conta a velocidade nos dous eixes. Neste xogo non permitimos que se mova en diagonal xa que cando prememos unha tecla desactivamos as anteriores.
- Lembrar que na clase ControladorXogo debemos modificar a velocidade cando se preme a tecla correspondente e para de moverse cando soltamos a tecla.
- Lembrar chamar ó método update do alien dende a clase ControladorXogo.

Posible solución:

Código da clase Alien
Obxectivo: Definimos os métodos para xestionar a velocidade nos eixes x/y e modificamos o método update.

```
public class Alien extends Personaxe{

private Vector2 velocidade;

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

velocidade = new Vector2(0,0);

}

public float getVelocidadeX(){
return velocidade.x;
}
```

```
public float getVelocidadeY(){
return velocidade.y;
}

public void setVelocidadeX(float x){
velocidade.x = x;

}
public void setVelocidadeY(float y){
velocidade.y = y;
}

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

setPosicion(getPosicion().x+velocidade.x*delta, getPosicion().y+velocidade.y*delta);

}

}
```

Como vemos usamos un Vector2 para gardar as dúas velocidades. Cando prememos unha tecla faremos que dita velocidade valga a velocidade máxima (a que lle mandamos no constructor cando instanciamos o obxecto alien)

Código da clase ControladorXogo
Obxectivo: Modificamos a velocidade do alien en función da tecla pulsada.

```
public class ControladorXogo {

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

private void controlarAlien(float delta){

// Actualiza Alien
alien.update(delta);
}

/**
         * Vai chamar a todos os métodos para mover e controlar os personaxes Tamén
         * xestionará os eventos producidos polo usuario e que veñen dende a clase
         * PantallaXogo
         *
         * @param delta
         */
public void update(float delta) {

controlarCoches(delta);
controlarRochas(delta);
controlarTroncos(delta);

controlarNave(delta);
controlarAlien(delta);

procesarEntradas();

}
private void procesarEntradas(){

if (keys.get(Keys.DEREITA))
alien.setVelocidadeX(alien.velocidade_max);
if (keys.get(Keys.ESQUERDA))
alien.setVelocidadeX(-alien.velocidade_max);
if (!(keys.get(Keys.ESQUERDA)) && (!(keys.get(Keys.DEREITA))))
alien.setVelocidadeX(0);
```

```
if (keys.get(Keys.ARRIBA))
alien.setVelocidadeY(alien.velocidade_max);
if (keys.get(Keys.ABAIXO))
alien.setVelocidadeY(-alien.velocidade_max);
if (!(keys.get(Keys.ARRIBA)) && (!(keys.get(Keys.ABAIXO))))
alien.setVelocidadeY(0);

}
```

Nota: Modificar a orde de chamada do método debuxarAlien na clase RendererXogo para que sexa a última. Desta forma o alien vaise debuxar por enriba de todos os demais elementos.

Agora imos facer unha pequena modificación no xogo para que a parte baixa da pantalla conte cunha banda negra onde van ir as vidas salvadas e o crono do xogo.

Isto xa deberiades ser capaces de facelo. Só temos que debuxar a textura baseada no gráfico LIBGDX_puntonegro.jpg que temos feito da tarefa 2.3. Como imos ter que gardar o tamaño desta banda negra para poder debuxalo dende a clase RendererXogo e facer que o Alien non poida baixar dende a clase ControladorXogo, imos definir o tamaño nunha clase de nome Controis que a definiremos no paquete principal.

Código da clase Controis
Obxectivo: Gardamos o tamaño dos controis utilizados no xogo. Neste caso o tamaño da banda negra inferior.

```
public class Controis {

public final static Rectangle FONDO_NEGRO = new Rectangle(0,0,Mundo.TAMANO_MUNDO_ANCHO,12);
}
```

Agora debuxamos dito control na clase RendererXogo:

Código da clase RendererXogo
Obxectivo: Debuxar a banda inferior negra.

```
..............
private void debuxarControis(){

// Fondo negro
spritebatch.draw(AssetsXogo.texturePuntoNegro, Controis.FONDO_NEGRO.x,Controis.FONDO_NEGRO.y,Controis.FONDO_NEGRO.width,Controis.FONI

}

    public void render(float delta) {
Gdx.gl.glClearColor(0, 0, 0, 1);
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

spritebatch.begin();

debuxarFondo();

debuxarNave();

debuxarCoches();
debuxarRochas();
debuxarTroncos();

debuxarAlien();

debuxarControis();
spritebatch.end();

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

Exercicio proposto: Facer que o alien non saia da pantalla, é dicir, que se mova nos límites do noso mundo e que non poda ir máis abaixo da banda inferior negra.

Posible solución:
Código da clase ControladorXogo
Obxectivo: Facer que o Alien se mova dentro dos límites da pantalla.

```
.....................
private void controlarAlien(float delta){

// Actualiza Alien
alien.update(delta);

// Impide que se mova fora dos límites da pantalla
if (alien.getPosicion().x <=0){
alien.setPosicion(0, alien.getPosicion().y);
}
else {
if (alien.getPosicion().x >= Mundo.TAMANO_MUNDO_ANCHO-alien.getTamano().x){
alien.setPosicion(Mundo.TAMANO_MUNDO_ANCHO-alien.getTamano().x, alien.getPosicion().y);
}

}

if (alien.getPosicion().y <=Controis.FONDO_NEGRO.height){
alien.setPosicion(alien.getPosicion().x,Controis.FONDO_NEGRO.height);
}
else {
if (alien.getPosicion().y >= Mundo.TAMANO_MUNDO_ALTO-alien.getTamano().y){
alien.setPosicion(alien.getPosicion().x, Mundo.TAMANO_MUNDO_ALTO-alien.getTamano().y);
}

}


}
```

Como temos definido o noso xogo (MVC) agora é moi sinxelo engadir novas formas de control (acelerómetro, gráfico, touchpad,...) xa que o único que temos que facer é controlar na clase que ten a interface cando se preme o control e chamar ó método presionarTecla coa 'tecla' pulsada.

TAREFA 2.8.A A FACER: Esta parte está asociada á realización dunha tarefa.

Tarefas avanzadas

TAREFA AVANZADA OPTATIVA: Esta parte está asociada á realización dunha tarefa avanzada: Acelerómetro/Compás.

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

TAREFA AVANZADA OPTATIVA: Esta parte está asociada á realización dunha tarefa avanzada: TochPad (joystick).

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