Adrián Jiménez Villarreal

71904816P | UO227602@uniovi.es

Juego Plataformas

Ampliaciones

Contenido

[Ampliaciones 2](#_Toc465613524)

[Nuevos tipos de enemigos 3](#_Toc465613525)

[Enemigo que salta o vuela 3](#_Toc465613526)

[Enemigo que te persigue 3](#_Toc465613527)

[Scroll en el eje Y 3](#_Toc465613528)

[Items recolectables 3](#_Toc465613529)

[Descripción 3](#_Toc465613530)

[Creación de la clase recolectable 3](#_Toc465613531)

[Creación de la clase contador 5](#_Toc465613532)

[Modificación del nivel 5](#_Toc465613533)

[Plataformas móviles 6](#_Toc465613534)

[Caja que se puedan arrastrar 6](#_Toc465613535)

[Tiles destruibles 6](#_Toc465613536)

[Tiles escalera 6](#_Toc465613537)

[Tiles solidos con inclinación 6](#_Toc465613538)

[Enemigos más inteligentes 6](#_Toc465613539)

[Punto de salvado 6](#_Toc465613540)

[Puertas 6](#_Toc465613541)

[Disparo con gravedad 6](#_Toc465613542)

[Disparo direccional 6](#_Toc465613543)

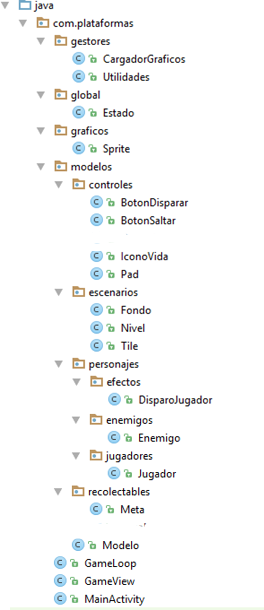
[Completar la interfaz del juego 6](#_Toc465613544)

[Multijugador con teclado 6](#_Toc465613545)

[Otras ampliaciones propuestas por el alumno 6](#_Toc465613546)

# Ampliaciones

Antes de realizar cualquiera de las ampliaciones se ha reorganizado la estructura de paquetes, añadido nuevos y creado una clase con los estados que puede estar los objetos.



## Nuevos tipos de enemigos

### Descripción

Lo más adecuado sería crear una clase base para enemigo o un interface Java y que los diferentes tipos de enemigos heredasen de la clase base o implementaran el interface.

Al menos se debe crear un tipo de enemigo nuevo con diferente imagen, velocidad y capaz de disparar

### Creación de la clase base Enemigo y sus derivados

Para empezar, crearemos una clase base para los enemigos, en este caso usaremos la clase Enemigo como base que sea ella quien extienda de modelo

|  |
| --- |
| **public class** Enemigo **extends** Modelo {  **public static final** String ***CAMINANDO\_DERECHA*** = **"Caminando\_derecha"**;  **public static final** String ***CAMINANDO\_IZQUIERDA*** = **"caminando\_izquierda"**;   **public static final** String ***MUERTE\_DERECHA*** = **"muerte\_derecha"**;  **public static final** String ***MUERTE\_IZQUIERDA*** = **"muerte\_izquierda"**;    **protected** Sprite **sprite**;  **protected** HashMap<String,Sprite> **sprites** = **new** HashMap<String,Sprite>();   **public double velocidadX** = 1.2;  **protected int orientacion**;  **public int estado**;   **public static final int *DERECHA*** = 1;  **public static final int *IZQUIERDA*** = -1;    **public** Enemigo(Context context, **double** xInicial, **double** yInicial) {  **super**(context, 0, 0, 40, 40);   **this**.**x** = xInicial;  **this**.**y** = yInicial - **altura**/2;   }   **public void** inicializar (){   Sprite caminandoDerecha = **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***enemyrunright***),  **ancho**, **altura**,  4, 4, **true**);  **sprites**.put(***CAMINANDO\_DERECHA***, caminandoDerecha);   Sprite caminandoIzquierda = **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***enemyrun***),  **ancho**, **altura**,  4, 4, **true**);  **sprites**.put(***CAMINANDO\_IZQUIERDA***, caminandoIzquierda);    Sprite muerteDerecha = **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***enemydieright***),  **ancho**, **altura**,  4, 8, **false**);  **sprites**.put(***MUERTE\_DERECHA***, muerteDerecha);   Sprite muerteIzquierda = **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***enemydie***),  **ancho**, **altura**,  4, 8, **false**);  **sprites**.put(***MUERTE\_IZQUIERDA***, muerteIzquierda);     **sprite** = caminandoDerecha;  }    **public void** actualizar (**long** tiempo) {  **boolean** finSprite = **sprite**.actualizar(tiempo);   **if** ( **estado** == Estado.***INACTIVO*** && finSprite){  **estado** = Estado.***ELIMINAR***;  }  **if** (**estado** == Estado.***INACTIVO***){  **if** (**velocidadX** > 0) {  **sprite** = **sprites**.get(***MUERTE\_DERECHA***);  **orientacion** = ***DERECHA***;  }  **else**{  **sprite** = **sprites**.get(***MUERTE\_IZQUIERDA***);  **orientacion** = ***IZQUIERDA***;  }  } **else** {   **if** (**velocidadX** > 0) {  **sprite** = **sprites**.get(***CAMINANDO\_DERECHA***);  **orientacion** = ***DERECHA***;  }  **if** (**velocidadX** < 0) {  **sprite** = **sprites**.get(***CAMINANDO\_IZQUIERDA***);  **orientacion** = ***IZQUIERDA***;  }   }  }    **public void** girar(){  **velocidadX** = **velocidadX**\*-1;  }   **public void** dibujar(Canvas canvas){  **sprite**.dibujarSprite(canvas, (**int**) **x** - Nivel.*scrollEjeX*, (**int**) **y** - Nivel.*scrollEjeY*);  }  **public void** destruir (){  **velocidadX** = 0;  **estado** = Estado.***INACTIVO***;  }  } |

Ahora pasaremos a crear las clases hijas, cada una con sus diferencias.

Clase **EnemigoBásico**.

|  |
| --- |
| **public class** EnemigoBasico **extends** Enemigo {  **public** EnemigoBasico(Context context, **double** xInicial, **double** yInicial) {  **super**(context, xInicial, yInicial);  **cDerecha** = 15;  **cIzquierda** = 15;  **cArriba** = 20;  **cAbajo** = 20;   **estado** = Estado.***ACTIVO***;   inicializar();  } } |

### Creación de los nuevos tipos de disparo

Como ahora los enemigos de la ampliación necesitarán disparar crearemos una clase base **Disparo** con lo común y sus hijas serán **DisparoEnemigo** y **DisparoJugador.**

Clase **Disparo.**

|  |
| --- |
| **public class** Disparo **extends** Modelo {   **protected** Sprite **sprite**;  **public double velocidadX** = 10;   **public** Disparo(Context context, **double** xInicial, **double** yInicial) {  **super**(context, xInicial, yInicial, 35, 35);   }   **public void** inicializar (){  **sprite**= **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**,  R.drawable.***animacion\_disparo3***),  **ancho**, **altura**,  24, 5, **true**);  }  **public void** actualizar (**long** tiempo) {  **sprite**.actualizar(tiempo);  }   **public void** dibujar(Canvas canvas){  **sprite**.dibujarSprite(canvas, (**int**) **x** - Nivel.*scrollEjeX*, (**int**) **y** - Nivel.*scrollEjeY*);  } } |

Clase **DisparoEnemigo**.

|  |
| --- |
| **public class** DisparoEnemigo **extends** Disparo{  **public** DisparoEnemigo(Context context, **double** xInicial, **double** yInicial, **int** orientacion) {  **super**(context, xInicial, yInicial);   **if** (orientacion == Enemigo.***IZQUIERDA***)  **velocidadX** = **velocidadX**\*-1;   **cDerecha** = 6;  **cIzquierda** = 6;  **cArriba** = 6;  **cAbajo** = 6;   inicializar();  } } |

Clase **DisparoJugador**.

|  |
| --- |
| **public class** DisparoJugador **extends** Disparo {   **public** DisparoJugador(Context context, **double** xInicial, **double** yInicial,**int** orientacion) {  **super**(context, xInicial, yInicial);   **if** (orientacion == Jugador.***IZQUIERDA***)  **velocidadX** = **velocidadX**\*-1;   **cDerecha** = 6;  **cIzquierda** = 6;  **cArriba** = 6;  **cAbajo** = 6;   inicializar();  }   **public void** inicializar (){  **sprite**= **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**,  R.drawable.***animacion\_disparo3***),  **ancho**, **altura**,  24, 5, **true**);  }   **public void** actualizar (**long** tiempo) {  **sprite**.actualizar(tiempo);  }   **public void** dibujar(Canvas canvas){  **sprite**.dibujarSprite(canvas, (**int**) **x** - Nivel.*scrollEjeX*, (**int**) **y** - Nivel.*scrollEjeY*);  } } |

Con todo esto creado podremos pasar a crear la clase **EnemigoAmpliacion.**

### Creación del nuevo tipo de enemigo

Clase **EnemigoAmpliacion**.

|  |
| --- |
| **public class** EnemigoAmpliacion **extends** Enemigo {   **protected int cadenciaDisparo** = 3000;  **protected long milisegundosDisparo** = 0;   **public** EnemigoAmpliacion(Context context, **double** xInicial, **double** yInicial) {  **super**(context, xInicial, yInicial);   **velocidadX** = 1.8;  **cDerecha** = 15;  **cIzquierda** = 15;  **cArriba** = 20;  **cAbajo** = 20;   **estado** = Estado.***ACTIVO***;   inicializar();  }   @Override  **public void** inicializar() {  Sprite caminandoDerecha = **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***enemyrunrightampliacion***),  **ancho**, **altura**,  4, 4, **true**);  **sprites**.put(***CAMINANDO\_DERECHA***, caminandoDerecha);   Sprite caminandoIzquierda = **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***enemyrunampliacion***),  **ancho**, **altura**,  4, 4, **true**);  **sprites**.put(***CAMINANDO\_IZQUIERDA***, caminandoIzquierda);    Sprite muerteDerecha = **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***enemydierightampliacion***),  **ancho**, **altura**,  3, 6, **false**);  **sprites**.put(***MUERTE\_DERECHA***, muerteDerecha);   Sprite muerteIzquierda = **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***enemydieampliacion***),  **ancho**, **altura**,  3, 6, **false**);  **sprites**.put(***MUERTE\_IZQUIERDA***, muerteIzquierda);     **sprite** = caminandoDerecha;  }   **public** Disparo disparar(**long** milisegundos) {  **if** (milisegundos - **milisegundosDisparo**> **cadenciaDisparo** + Math.*random*()\* **cadenciaDisparo**) {  **milisegundosDisparo** = milisegundos;  **return new** DisparoEnemigo(**context**, **x**, **y**,**orientacion**);  }  **return null**;  } } |

Ya tenemos todo lo necesario para incluirlo en el juego ahora solo habrá que modificar la clase **Nivel.**

### Inclusión en el Nivel

En esta clase habrá que modificar el método aplicarReglasDeMovimiento, ya que este es el que se encargara de gestionar tanto los nuevos enemigos, como los disparos que estos realizaran. Además debemos gestionar como incluir los enemigos en el mapa.

1º Se crea una lista con los disparos de los enemigos.

2º Se inicializa en el método inicializar.

3\* Se gestiona la inclusión de nuevos tipos de enemigo.

|  |
| --- |
| **case 'N'**:  *// Enemigo  // Posición centro abajo* **int** xCentroAbajoTileN = x \* Tile.*ancho* + Tile.*ancho* / 2;  **int** yCentroAbajoTileN = y \* Tile.*altura* + Tile.*altura*;  **enemigos**.add(**new** EnemigoAmpliacion(**context**, xCentroAbajoTileN, yCentroAbajoTileN));   **return new** Tile(**null**, Tile.***PASABLE***); |

4º Se incluye en el método aplicarReglasDeMovimiento la creación de los disparos de los enemigos dentro del iterador de enemigos, justo cuando estos están a un rango del jugador.

|  |
| --- |
| **if** (tileXJugadorIzquierda - rango < tileXEnemigoIzquierda &&  tileXJugadorIzquierda + rango > tileXEnemigoIzquierda){  **…**  **long** tiempo = System.*currentTimeMillis*(); **if** (enemigo.**estado** == Estado.***ACTIVO***){  **if**(enemigo **instanceof** EnemigoAmpliacion) {  DisparoEnemigo disparo = (DisparoEnemigo) ((EnemigoAmpliacion)enemigo).disparar(tiempo);  **if** (disparo != **null**) {  **disparosEnemigos**.add(disparo);  }  }  } |

5º Se añade la colisión con este tipo de disparos para el jugador.

|  |
| --- |
| **for** (Iterator<Disparo> iterator = **disparosEnemigos**.iterator(); iterator.hasNext();) {  DisparoEnemigo disparoEnemigo = (DisparoEnemigo) iterator.next();  **Mismo código para recorrer los disparos del jugador, solo cambia el if final.**  **…**  **if** (disparoEnemigo.colisiona(**jugador**)){  **if**(**jugador**.golpeado() <= 0){  **nivelPausado** = **true**;  **mensaje** = CargadorGraficos.*cargarBitmap*(**context**, R.drawable.***you\_lose***);  **jugador**.restablecerPosicionInicial();  inicializar();  **return**;  }  } } |

Con esto los nuevos enemigos ya dispararían si están en un rango cercano al jugador, tendrían un sprite diferente y se moverían más rápido. Con lo que la ampliación estaría completa.

## Enemigo que salta o vuela

### Descripción

Incluir un nuevo tipo de enemigo capaz de saltar o volar (no puede volar siempre en línea recta, ha de moverse en el eje X e Y).

### Creación del nuevo tipo de enemigo

Seguiremos la misma manera de creación de un nuevo tipo de enemigo vista en Nuevos tipos de enemigos. Pero en este caso contendrá unos ligeros la clase **EnemigoVolador**.

|  |
| --- |
| **public class** EnemigoVolador **extends** Enemigo {   **public double velocidadY** = 1.0;  **public double yInicial**;  **public double yDestino**;  **public boolean direccion**;   **public** EnemigoVolador(Context context, **double** xInicial, **double** yInicial) {  **super**(context, xInicial, yInicial);   **this**.**yInicial** = **y**;  **velocidadX** = 1.8;  **cDerecha** = 15;  **cIzquierda** = 15;  **cArriba** = 20;  **cAbajo** = 20;   **estado** = Estado.***ACTIVO***;   inicializar();  }   @Override  **public void** inicializar() {  Sprite caminandoDerecha = **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***enemyrightvolador***),  **ancho**, **altura**,  3, 6, **true**);  **sprites**.put(***CAMINANDO\_DERECHA***, caminandoDerecha);   Sprite caminandoIzquierda = **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***enemyvolador***),  **ancho**, **altura**,  3, 6, **true**);  **sprites**.put(***CAMINANDO\_IZQUIERDA***, caminandoIzquierda);    Sprite muerteDerecha = **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***enemydievolador***),  **ancho**, **altura**,  3, 6, **false**);  **sprites**.put(***MUERTE\_DERECHA***, muerteDerecha);   Sprite muerteIzquierda = **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***enemydievolador***),  **ancho**, **altura**,  3, 6, **false**);  **sprites**.put(***MUERTE\_IZQUIERDA***, muerteIzquierda);     **sprite** = caminandoDerecha;  }  **public void** volar() {  **if** (**direccion** && **y** >= **yDestino**) {  **y** -= **velocidadY**;  **if** (**y** <= **yDestino**) {  **direccion** = **false**;  **yDestino** = **y** + calcularMovimiento();  **if** (**yDestino** > **yInicial** + 33)  **yDestino** = **yInicial** + 33;  }  }  **if** (!**direccion** && **y** <= **yDestino**) {  **y** += **velocidadY**;  **if** (**y** >= **yDestino**) {  **direccion** = **true**;  **yDestino** = **y** - calcularMovimiento();  **if** (**yDestino** < **yInicial** - 33)  **yDestino** = **yInicial** - 33;  }  }  **if** (**yDestino** == 0 && **y** == **yInicial**) {  **int** movimiento = calcularMovimiento();  **int** posicion = (**int**) Math.*floor*(Math.*random*() \* (2 - 0 + 1) + (0));  **if** (posicion == 0) {  **yDestino** = **y** - movimiento;  **direccion** = **true**;  } **else** {  **yDestino** = movimiento + **y**;  **direccion** = **false**;  }  }  }   **private int** calcularMovimiento(){  **return** (**int**)Math.*floor*(Math.*random*()\*(25-1+1)+(1));  } |

### Inclusión del tipo de enemigo en el nivel

Al igual que con los anteriores enemigos que hemos añadido se debe añadir los nuevos tipos de enemigo.

|  |
| --- |
| **case 'V'**:  *// Enemigo  // Posición centro abajo* **int** xCentroAbajoTileV = x \* Tile.*ancho* + Tile.*ancho* / 2;  **int** yCentroAbajoTileV = y \* Tile.*altura* + Tile.*altura*;  **enemigos**.add(**new** EnemigoVolador(**context**, xCentroAbajoTileV, yCentroAbajoTileV));   **return new** Tile(**null**, Tile.***PASABLE***); |

Posteriormente deberemos añadir las nuevas reglas de movimiento para este tipo de enemigo.

Cuando la velocidad de X del enemigo sea > 0

|  |
| --- |
| **…**  **}else if**(tileXEnemigoDerecha + 1 <= anchoMapaTiles() - 1 &&  **mapaTiles**[tileXEnemigoDerecha + 1][tileYEnemigoInferior].**tipoDeColision** ==  Tile.***PASABLE*** &&  **mapaTiles**[tileXEnemigoDerecha + 1][tileYEnemigoCentro].**tipoDeColision** ==  Tile.***PASABLE*** &&  **mapaTiles**[tileXEnemigoDerecha + 1][tileYEnemigoSuperior].**tipoDeColision** ==  Tile.***PASABLE*** &&  **mapaTiles**[tileXEnemigoDerecha + 1][tileYEnemigoInferior + 1].**tipoDeColision** ==  Tile.***PASABLE*** && enemigo **instanceof** EnemigoVolador){   enemigo.**x** += enemigo.**velocidadX**;  ((EnemigoVolador) enemigo).volar();   *// Sino, me acerco al borde del que estoy*  }  … |

Cuando la velocidad de X del enemigo sea < 0

|  |
| --- |
| **…**  **}else if**(tileXEnemigoIzquierda - 1 >= 0 &&  **mapaTiles**[tileXEnemigoIzquierda - 1][tileYEnemigoInferior].**tipoDeColision** ==  Tile.***PASABLE*** &&  **mapaTiles**[tileXEnemigoIzquierda - 1][tileYEnemigoCentro].**tipoDeColision** ==  Tile.***PASABLE*** &&  **mapaTiles**[tileXEnemigoIzquierda - 1][tileYEnemigoSuperior].**tipoDeColision** ==  Tile.***PASABLE*** &&  **mapaTiles**[tileXEnemigoIzquierda - 1][tileYEnemigoInferior + 1].**tipoDeColision** == Tile.***PASABLE*** && enemigo **instanceof** EnemigoVolador){   enemigo.**x** += enemigo.**velocidadX**;  ((EnemigoVolador) enemigo).volar();  }  .... |

Con esto ya tendríamos al nuevo enemigo volador.

## Enemigo que te persigue

### Descripción

Incluir un nuevo enemigo especial, que se coloca detrás del jugador desde el inicio del nivel y persigue al jugador de forma constante (replica el mismo camino que hizo el jugador). Este enemigo no podrá ser eliminado.

## Scroll en el eje Y

### Descripción

Permitir que los mapas tengan más altura que la propia pantalla, se debe aplicar un scroll en el eje Y similar al que se aplicó en el eje X.

### Modificación de la clase Nivel

Esta ampliación será similar al scroll del eje X.

1º Se crea la nueva variable scrollEjeY inicializada a 0

2º Después de iniciar el mapa de tiles se le da el valor inicial al scroll del eje Y, pero antes de esto debe valer 0.

|  |
| --- |
| scrollEjeY = 0; **mensaje** = CargadorGraficos.*cargarBitmap*(**context**, R.drawable.***description***);  ……  inicializarMapaTiles();  scrollEjeY = (**int**) (altoMapaTiles() - tilesEnDistanciaY(GameView.*pantallaAlto*))\*Tile.*altura*; |

3º El método dibujar tiles deberá tener las condiciones del scroll del eje y.

|  |
| --- |
| …  **int** tileYJugador = (**int**) **jugador**.**y** / Tile.*altura*;  **int** arriba = (**int**) (tileYJugador - tilesEnDistanciaY(**jugador**.**y** - *scrollEjeY*)); arriba = Math.*max*(0,arriba);  **if** ( **jugador** .**y** <  altoMapaTiles()\* Tile.*altura* - GameView.*pantallaAlto*\*0.3 )  **if**( **jugador** .**y**- *scrollEjeY* > GameView.*pantallaAlto* \* 0.7 )  *scrollEjeY* += (**int**) ((**jugador** .**y** - *scrollEjeY*) - GameView.*pantallaAlto*\* 0.7);   **if** ( **jugador** .**y** > GameView.*pantallaAlto*\*0.3 )  **if**( **jugador** .**y** - *scrollEjeY* < GameView.*pantallaAlto* \*0.3 )  *scrollEjeY* -= (**int**) (GameView.*pantallaAlto*\*0.3 -(**jugador** .**y** - *scrollEjeY*));   **int** abajo = arriba +  GameView.*pantallaAlto* / Tile.*altura* + 1;  abajo = Math.*min*(abajo, altoMapaTiles() - 1);   **for** (**int** y = arriba; y <= abajo; ++y) {  **for** (**int** x = izquierda; x <= derecha; ++x) {  **if** (**mapaTiles**[x][y].**imagen** != **null**) {  *// Calcular la posición en pantalla correspondiente  // izquierda, arriba, derecha , abajo* **mapaTiles**[x][y].**imagen**.setBounds(  (x \* Tile.*ancho*) - *scrollEjeX*,  (y \* Tile.*altura*) - *scrollEjeY*,  (x \* Tile.*ancho*) + Tile.*ancho* - *scrollEjeX*,  (y \* Tile.*altura* + Tile.*altura*) - *scrollEjeY*);   **mapaTiles**[x][y].**imagen**.draw(canvas);  }  } } |

Con todo esto ya estaría acabado de añadir el scroll del eje Y en el nivel, ahora habría que añadirlo a cada clase que este dibujada en el nivel, como el jugador, los enemigos, etc… como se hizo con el scroll en el eje X.

## Items recolectables

### Descripción

Incluir ítems recolectable inmovibles que puedan ser recolectados por el Jugador al colisionar con ellos. Debemos incluir un contador de ítems que muestre el número en pantalla. El Ítem debe ser animado, utilizando el siguiente sprite:



El número y posición de los ítems se deberán poder especificar en el mapa del Nivel.

### Creación de la clase recolectable

Dentro del paquete recolectables crear la clase **Recolectable.**

|  |
| --- |
| **package** com.plataformas.modelos.recolectables;  **import** android.content.Context; **import** android.graphics.Canvas;  **import** com.plataformas.R; **import** com.plataformas.gestores.CargadorGraficos; **import** com.plataformas.global.Estado; **import** com.plataformas.graficos.Sprite; **import** com.plataformas.modelos.Modelo; **import** com.plataformas.modelos.escenarios.Nivel;  **import** java.util.HashMap;  */\*\*  \* Created by uo227602 on 05/10/2016.  \*/* **public class** Recolectable **extends** Modelo {   **public static final** String ***GEMA\_GIRANDO*** = **"Gema\_Girando"**;  **public static final** String ***GEMA\_DESAPARECIENDO*** = **"Gema\_desapareciendo"**;   *//Puntero sprite actual* **private** Sprite **sprite**;   **private** HashMap<String,Sprite> **sprites** = **new** HashMap<String,Sprite> ();   **double xInicial**;  **double yInicial**;   **int estado**;    **public** Recolectable(Context context, **double** xInicial, **double** yInicial) {  **super**(context, xInicial,yInicial, 40, 40);   **this**.**xInicial**=xInicial;  **this**.**yInicial**=yInicial;   inicializar();  }   **private void** inicializar() {   Sprite gemaGirando = **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***gem***),  **ancho**, **altura**,  4, 8, **true**);  **sprites**.put(***GEMA\_GIRANDO***, gemaGirando);   Sprite gemaDesapareciendo = **new** Sprite(  CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***item\_on\_collected***),  **ancho**, **altura**,  5, 10, **false**);  **sprites**.put(***GEMA\_DESAPARECIENDO***, gemaDesapareciendo);  **estado** = Estado.***ACTIVO***;  **sprite** = gemaGirando;  }   **public void** dibujar(Canvas canvas){  **sprite**.dibujarSprite(canvas, (**int**)**x** - Nivel.*scrollEjeX* , (**int**)**y** );  }   **public void** actualizar(**long** tiempo){  **boolean** finSprite = **sprite**.actualizar(tiempo);   **if** (**estado** == Estado.***INACTIVO***) {  **sprite** = **sprites**.get(***GEMA\_DESAPARECIENDO***);  }  **else**{  **sprite** = **sprites**.get(***GEMA\_GIRANDO***);  }  **if** ( **estado** == Estado.***INACTIVO*** && finSprite){  **estado** = Estado.***ELIMINAR***;  }  }  **public void** destruir(){  **estado** = Estado.***INACTIVO***;  }  **public int** getEstado() {  **return estado**;  } } |

### Creación de la clase contador

Dentro del paquete controles crear la clase **Contador.**

|  |
| --- |
| **package** com.plataformas.modelos.controles;  **import** android.content.Context; **import** android.graphics.Canvas; **import** android.graphics.Color; **import** android.graphics.Paint;  **import** com.plataformas.GameView; **import** com.plataformas.R; **import** com.plataformas.gestores.CargadorGraficos; **import** com.plataformas.modelos.Modelo;  */\*\*  \* Created by uo227602 on 05/10/2016.  \*/* **public class** Contador **extends** Modelo {   **int puntuacion**=0;   **public** Contador(Context context) {  **super**(context, GameView.*pantallaAncho*\*0.90 , GameView.*pantallaAlto*\*0.1,  GameView.*pantallaAlto*, GameView.*pantallaAncho*);   **altura** = 40;  **ancho** = 40;  **imagen** = CargadorGraficos.*cargarDrawable*(context, R.drawable.***score***);  }   **public void** actualizarPuntuacion(**int** puntuacion){  **this**.**puntuacion**+=puntuacion;  }  **public void** reiniciarPuntuacion(){  **this**.**puntuacion** = 0;  }  @Override  **public void** dibujar(Canvas canvas){  **int** yArriva = (**int**) **y** - **altura** / 2;  **int** xIzquierda = (**int**) **x** - **ancho** / 2;  Paint paint = **new** Paint();  paint.setColor(Color.***WHITE***);  paint.setAntiAlias(**true**);  paint.setTextSize(20);  canvas.drawText(String.*valueOf*(**puntuacion**), xIzquierda - **ancho**, yArriva + **altura**/2, paint);  **imagen**.setBounds(xIzquierda, yArriva, xIzquierda  + **ancho**, yArriva + **altura**);  **imagen**.draw(canvas);  }  } |

### Modificación del nivel

En el nivel habrá que añadir la lista de recolectables, cargar los recolectables en esta lista y gestionar las colisiones con estos en el método aplicar reglas de movimiento.

|  |
| --- |
| **private** List<Recolectable> **recolectables** = **new** ArrayList<>(); |

|  |
| --- |
| **case 'a'**:  **int** xCentroAbajoTileR = x \* Tile.*ancho* + Tile.*ancho*/2;  **int** yCentroAbajoTileR = y \* Tile.*altura* + Tile.*altura*;  **recolectables**.add(**new** Recolectable(**context**, xCentroAbajoTileR, yCentroAbajoTileR));   **return new** Tile(**null**, Tile.***PASABLE***); |

|  |
| --- |
| **for**(Iterator<Recolectable> iterator = **recolectables**.iterator();iterator.hasNext();){  Recolectable recolectable = iterator.next();  **if**(**jugador**.colisiona(recolectable) && recolectable.getEstado()==Estado.***ACTIVO***){  recolectable.destruir();  **gameView**.*contador*.actualizarPuntuacion(1);  **break**;  }  **if** (recolectable.getEstado() == Estado.***ELIMINAR***){  iterator.remove();  **continue**;  } } |

## Plataformas móviles

### Descripción

Incluir plataformas móviles con movilidad en el eje X.

Deben moverse con una velocidad constante preestablecida hasta que colisionen con algún elemento. El jugador y los enemigos deben poder caminar sobre estas plataformas.

Las plataformas se representarán en el mapa de nivel con el carácter 'X'.

### Creación de la clase Plataforma

Dentro del paquete escenarios.

|  |
| --- |
| **public class** Plataforma **extends** Modelo {   **double velocidadX** = 3;   **public** Plataforma(Context context, **double** x, **double** y) {  **super**(context, x, y, 32, 40);   **this**.**y** = y -**altura**/2;    **imagen** = CargadorGraficos.*cargarDrawable*(context, R.drawable.***platform***);  }   **public boolean** girar(){  **velocidadX** = **velocidadX**\*-1;  **return true**;  }   **public void** dibujar(Canvas canvas){  **int** yArriva = (**int**) **y** - **altura**/2 - Nivel.*scrollEjeY*;  **int** xIzquierda = (**int**) **x** - **ancho**/2 - Nivel.*scrollEjeX*;   **imagen**.setBounds(xIzquierda, yArriva, xIzquierda  + **ancho**, yArriva + **altura**);  **imagen**.draw(canvas);  } } |

### Modificación de la clase Nivel

Primero se incluirá una lista de plataformas, esta lista se inicializará con lo leído de los ficheros del nivel, además se deberá dibujar cada elemento en el nivel.

Posterior a esto se debe crear una variable global booleana denominada giradoPlataforma. La usaremos para saber cuándo gira la plataforma.

Posteriormente a esto se modifica el método aplicarReglasMovimiento.

|  |
| --- |
| **……**  **for** (Iterator<Plataforma> iterator = **plataformas**.iterator(); iterator.hasNext(); ) {  Plataforma plataforma = iterator.next();   **int** tileXPlataformaIzquierda =  (**int**) (plataforma.**x** - (plataforma.**ancho** / 2 - 1)) / Tile.*ancho*;  **int** tileXPlataformaDerecha =  (**int**) (plataforma.**x** + (plataforma.**ancho** / 2 - 1)) / Tile.*ancho*;  **int** tileYPlataformaInferior =  (**int**) (plataforma.**y** + (plataforma.**altura** / 2 - 1)) / Tile.*altura*;  **int** tileYPlataformaCentro =  (**int**) plataforma.**y** / Tile.*altura*;  **if** (tileXPlataformaDerecha == tileXPlataformaIzquierda && **giradoPlataforma**) {  **giradoPlataforma** = **false**;  plataforma.**x** += plataforma.**velocidadX**;  } **else** {  **if** (plataforma.**velocidadX** > 0) {  **if** (tileXPlataformaDerecha + 1 <= anchoMapaTiles() - 1 &&  **mapaTiles**[tileXPlataformaDerecha + 1][tileYPlataformaInferior].**tipoDeColision** ==  Tile.***PASABLE*** &&  **mapaTiles**[tileXPlataformaDerecha + 1][tileYPlataformaCentro].**tipoDeColision** ==  Tile.***PASABLE***){   plataforma.**x** += plataforma.**velocidadX**;  **mapaTiles**[tileXPlataformaDerecha][tileYPlataformaCentro] = **new** Tile(**null**, Tile.***SOLIDO***);  **if** (!**giradoPlataforma**)  **mapaTiles**[tileXPlataformaDerecha - 1][tileYPlataformaCentro] = **new** Tile(**null**, Tile.***PASABLE***);  } **else if** (tileXPlataformaDerecha + 1 <= anchoMapaTiles() - 1) {  **int** TileEnemigoDerecho = tileXPlataformaDerecha \* Tile.*ancho* + Tile.*ancho*;  **double** distanciaX = TileEnemigoDerecho - (plataforma.**x** + plataforma.**ancho** / 2);   **if** (distanciaX > 0) {  **double** velocidadNecesaria = Math.*min*(distanciaX, plataforma.**velocidadX**);  plataforma.**x** += velocidadNecesaria;  } **else** {  **mapaTiles**[tileXPlataformaDerecha - 1][tileYPlataformaCentro] = **new** Tile(**null**, Tile.***PASABLE***);  **giradoPlataforma** = plataforma.girar();  }  } **else** {  plataforma.girar();  }  }  }  **if** (tileXPlataformaDerecha == tileXPlataformaIzquierda && **giradoPlataforma**) {  **giradoPlataforma** = **false**;  plataforma.**x** += plataforma.**velocidadX**;  } **else** {  **if** (plataforma.**velocidadX** < 0) {  **if** (tileXPlataformaIzquierda - 1 >= 0 &&  **mapaTiles**[tileXPlataformaIzquierda - 1][tileYPlataformaInferior].**tipoDeColision** ==  Tile.***PASABLE*** &&  **mapaTiles**[tileXPlataformaIzquierda - 1][tileYPlataformaCentro].**tipoDeColision** ==  Tile.***PASABLE***){   plataforma.**x** += plataforma.**velocidadX**;  **mapaTiles**[tileXPlataformaIzquierda][tileYPlataformaCentro] = **new** Tile(**null**, Tile.***SOLIDO***);  **if** (!**giradoPlataforma**)  **mapaTiles**[tileXPlataformaIzquierda + 1][tileYPlataformaCentro] = **new** Tile(**null**, Tile.***PASABLE***);  } **else if** (tileXPlataformaIzquierda - 1 >= 0) {  **giradoPlataforma** = **false**;  **int** TileEnemigoIzquierdo = tileXPlataformaIzquierda \* Tile.*ancho*;  **double** distanciaX = (plataforma.**x** - plataforma.**ancho** / 2) - TileEnemigoIzquierdo;  **if** (distanciaX > 0) {  **double** velocidadNecesaria =  Utilidades.*proximoACero*(-distanciaX, plataforma.**velocidadX**);  plataforma.**x** += velocidadNecesaria;  } **else** {  **mapaTiles**[tileXPlataformaIzquierda + 1][tileYPlataformaCentro] = **new** Tile(**null**, Tile.***PASABLE***);  **giradoPlataforma** = plataforma.girar();  }  } **else** {  plataforma.girar();  }  }  }  **if**(**jugador**.colisiona(plataforma)){  **jugador**.**x**+=plataforma.**velocidadX**;  }  **for**(Enemigo enemigo:**enemigos**){  **if**(enemigo.colisiona(plataforma)){  enemigo.**x**+=plataforma.**velocidadX**;  }  } }  …… |

## Caja que se puedan arrastrar

### Descripción

Incluir cajas que puedan ser arrastradas por el nivel cuando el jugador colisiona con ellos en el eje X.

Este tipo de elemento se representará en el mapa de nivel con la letra Q.

### Creación de la caja

|  |
| --- |
| **public class** Caja **extends** Modelo {   **double velocidadX** = 3;   **public** Caja(Context context, **double** x, **double** y) {  **super**(context, x, y, 32, 40);   **this**.**y** = y -**altura**/2;    **imagen** = CargadorGraficos.*cargarDrawable*(context, R.drawable.***platform\_dos***);  }    **public void** dibujar(Canvas canvas){  **int** yArriva = (**int**) **y** - **altura**/2 - Nivel.*scrollEjeY*;  **int** xIzquierda = (**int**) **x** - **ancho**/2 - Nivel.*scrollEjeX*;   **imagen**.setBounds(xIzquierda, yArriva, xIzquierda  + **ancho**, yArriva + **altura**);  **imagen**.draw(canvas);  } } |

### Modificación del nivel

Primero se creará una serie de cajas, se inicializarán, se asignarán al mapa de tiles y se dibujarán. Además de una variable booleana para saber cuándo esta encima de las cajas.

Posteriormente se modificará el método aplicarReglasMovimiento.

|  |
| --- |
| **…**  **for** (Iterator<Caja> iterator = **cajas**.iterator(); iterator.hasNext(); ) {   Caja caja = iterator.next();   **int** tileXCajaIzquierda =  (**int**) (caja.**x** - (caja.**ancho** / 2 - 1)) / Tile.*ancho*;  **int** tileXCajaDerecha =  (**int**) (caja.**x** + (caja.**ancho** / 2 - 1)) / Tile.*ancho*;  **int** tileYCajaInferior =  (**int**) (caja.**y** + (caja.**altura** / 2 - 1)) / Tile.*altura*;  **int** tileYCajaCentro =  (**int**) caja.**y** / Tile.*altura*;  **int** tileYCajaSuperior = (**int**) (caja.**y** - (caja.**altura** / 2 - 1)) / Tile.*altura*;  **int** tileJugador = (**int**) **jugador**.**x** / Tile.*ancho*;  **if** (**jugador**.getVelocidadX() > 0) {  **if** (**jugador**.colisiona(caja) && tileJugador == tileXCajaIzquierda && **mapaTiles**[tileXCajaDerecha][tileYCajaCentro].**tipoDeColision** == Tile.***PASABLE*** && **mapaTiles**[tileXCajaDerecha][tileYCajaInferior + 1].**tipoDeColision** == Tile.***SOLIDO*** && tileXCajaDerecha + 1 <= anchoMapaTiles() - 1) {  caja.**x** += **jugador**.getVelocidadX();  **encimaCaja** = **false**;  } **else if** (**jugador**.colisiona(caja) && tileYJugadorInferior == tileYCajaCentro && !**encimaCaja** && **jugador**.**enElAire**) {  **jugador**.**y** = (caja.**y** - (caja.**altura**));  **jugador**.**enElAire** = **false**;  **encimaCaja** = **true**;  } **else if** (**jugador**.colisiona(caja) && **encimaCaja**) {  **jugador**.**enElAire** = **false**;  **encimaCaja** = **true**;  } **else if** (!**jugador**.colisiona(caja) && (tileJugador >= tileXCajaDerecha || tileYJugadorCentro < tileYCajaSuperior+1)) {  **jugador**.**enElAire** = **true**;  **encimaCaja** = **false**;  }  } **else if** (**jugador**.getVelocidadX() < 0) {  **if** (**jugador**.colisiona(caja) && tileJugador == tileXCajaDerecha && **mapaTiles**[tileXCajaIzquierda][tileYCajaCentro].**tipoDeColision** == Tile.***PASABLE*** && **mapaTiles**[tileXCajaIzquierda][tileYCajaInferior + 1].**tipoDeColision** == Tile.***SOLIDO*** && tileXCajaIzquierda - 1 >= 0) {  caja.**x** += **jugador**.getVelocidadX();  **encimaCaja** = **false**;  } **else if** (**jugador**.colisiona(caja) && tileYJugadorInferior == tileYCajaCentro && !**encimaCaja** && **jugador**.**enElAire**) {  **jugador**.**y** = (caja.**y** - (caja.**altura**));  **jugador**.**enElAire** = **false**;  **encimaCaja** = **true**;  } **else if** (**jugador**.colisiona(caja) && **encimaCaja**) {  **jugador**.**enElAire** = **false**;  **encimaCaja** = **true**;  } **else if** (!**jugador**.colisiona(caja) && (tileJugador <= tileXCajaIzquierda || tileYJugadorCentro < tileYCajaSuperior+1)) {  **jugador**.**enElAire** = **true**;  **encimaCaja** = **false**;  }  } **else** {  **if** (**jugador**.colisiona(caja) && tileYJugadorInferior == tileYCajaCentro && !**encimaCaja** && **jugador**.**enElAire**) {  **jugador**.**y** = (caja.**y** - (caja.**altura**));  **jugador**.**enElAire** = **false**;  **encimaCaja** = **true**;  } **else if** (**jugador**.colisiona(caja) && **encimaCaja**) {  **jugador**.**enElAire** = **false**;  **encimaCaja** = **true**;  } **else if** (tileJugador==tileYCajaCentro && !**jugador**.colisiona(caja) && tileYJugadorCentro < tileYCajaSuperior+1) {  **jugador**.**enElAire** = **true**;  **encimaCaja** = **false**;  }  }  **for** (Enemigo enemigo : **enemigos**) {  **if** (enemigo.colisiona(caja)) {  enemigo.girar();  }  }  **for** (Plataforma plataforma : **plataformas**) {  **if** (plataforma.colisiona(caja)) {  plataforma.girar();  }  } }  …  **if** (jugador.getVelocidadY() >= 0 && !**encimaCaja**  ) {  … |

## Tiles destruibles

### Descripción

Incluir tiles que desaparezcan a los pocos segundos de que el Jugador entre en contacto con ellos (este colocado encima).

Este tipo de tiles se representará en el mapa de nivel con la letra W.

### Creacion de Destructible

|  |
| --- |
| **public class** Destructible **extends** Modelo {   **long tiempoPiso**;   **public** Destructible(Context context, **double** x, **double** y) {  **super**(context, x, y, 32, 40);   **this**.**y** = y -**altura**/2;    **imagen** = CargadorGraficos.*cargarDrawable*(context, R.drawable.***platform\_tres***);  }   **public void** dibujar(Canvas canvas){  **int** yArriva = (**int**) **y** - **altura**/2 - Nivel.*scrollEjeY*;  **int** xIzquierda = (**int**) **x** - **ancho**/2 - Nivel.*scrollEjeX*;   **imagen**.setBounds(xIzquierda, yArriva, xIzquierda  + **ancho**, yArriva + **altura**);  **imagen**.draw(canvas);  }  } |

### Modificacion del Nivel

Se añadirá una lista de destructibles, se inicializarán, se añadirán para que se puedan reconocer al cargar el mapa de tiles y por ultimo se dibujaran.

Posteriormente a esto se modificará el aplicarReglasMovimiento

|  |
| --- |
| **for**(Iterator<Destructible> iterator = **destructibles**.iterator(); iterator.hasNext();){  Destructible destructible = iterator.next();   **int** tileXDestruc = (**int**) destructible.**x** / Tile.*ancho*;  **int** tileYDestruc = (**int**) destructible.**y** / Tile.*altura*;  **if**(**jugador**.colisiona(destructible) && destructible.**tiempoPiso**==0){  destructible.**tiempoPiso** = System.*currentTimeMillis*();  }  **if**(destructible.**tiempoPiso**>0 && System.*currentTimeMillis*()-destructible.**tiempoPiso**>=1000){  **mapaTiles**[tileXDestruc][tileYDestruc] = **new** Tile(**null**,Tile.***PASABLE***);  iterator.remove();  **continue**;  } } |

## Tiles escalera

### Descripción

Incluir tiles de tipo escalera que permitan al jugador subir en el eje Y en línea recta.

Este tipo de tiles se representará en el mapa de nivel con la letra R

### Creacion de la Escalera

|  |
| --- |
| **public class** Escalera **extends** Modelo {    **public** Escalera(Context context, **double** x, **double** y) {  **super**(context, x, y, 32, 40);   **this**.**y** = y -**altura**/2;    **imagen** = CargadorGraficos.*cargarDrawable*(context, R.drawable.***escalera***);  }    **public void** dibujar(Canvas canvas){  **int** yArriva = (**int**) **y** - **altura**/2 - Nivel.*scrollEjeY*;  **int** xIzquierda = (**int**) **x** - **ancho**/2 - Nivel.*scrollEjeX*;   **imagen**.setBounds(xIzquierda, yArriva, xIzquierda  + **ancho**, yArriva + **altura**);  **imagen**.draw(canvas);  } } |

### Modificacion del Nivel

Se incluirá una lista de escaleras, se inicializara, se incluirá el código para poder ser reconocida y se crearan las variables booleanas padAbajoPulsado,padArribaPulsado

Posteriormente se modificará el método aplicarReglasMovimiento

|  |
| --- |
| **…**  **for**(Escalera escalera:**escaleras**){  **if**(**jugador**.colisiona(escalera) && **orientacionPadY**>0 && **padArribaPulsado** && **mapaTiles**[tileXJugador][tileYJugadorSuperior].**tipoDeColision**==Tile.***PASABLE***){  **jugador**.setVelocidadY(0);  **jugador**.**y** -= 3;  **jugador**.setVelocidadY(0);  **padArribaPulsado**=**false**;  }  **else if**(**jugador**.colisiona(escalera) && **orientacionPadY**<0 && **padAbajoPulsado**){  **jugador**.setVelocidadY(0);  **jugador**.**y** += 3;  **jugador**.setVelocidadY(0);  **padAbajoPulsado**=**false**;  }  **else if**(**jugador**.colisiona(escalera) && **orientacionPad**!=0){  **jugador**.setVelocidadY(0);  }  **else if**(**jugador**.colisiona(escalera)){  **jugador**.setVelocidadY(0);  }  }  … |

### Modificacion del GameView

Habra que definir cuando esta pulsado arriba o abajo, además de modificar como se reconocen las teclas por teclado cambiando el salto de la W a la E.

Cuando esta pulsado

|  |
| --- |
| **if** (**accion**[i] != **ACTION\_UP**) {  **if**(orientacion!=0) {  pulsacionPadMover = **true**;  *nivel*.**orientacionPad** = orientacion;  }  **else if**(orientacionY<0){  pulsacionPadMover=**false**;  *nivel*.**orientacionPadY** = orientacionY;  *nivel*.**entraPuerta**=**true**;  *nivel*.**padAbajoPulsado**=**true**;  }  **else if**(orientacionY>0){  pulsacionPadMover=**false**;  *nivel*.**orientacionPadY** = orientacionY;  *nivel*.**entraPuerta**=**true**;  *nivel*.**padArribaPulsado**=**true**;  }  **else if**(orientacionY==0){  pulsacionPadMover=**false**;  *nivel*.**orientacionPadY** = orientacionY;  *nivel*.**entraPuerta**=**false**;  *nivel*.**padArribaPulsado**=**false**;  } } |

Redefinir teclado

|  |
| --- |
| @Override **public boolean** onKeyDown(**int** keyCode, KeyEvent event) {  Log.*v*(**"Tecla"**,**"Tecla pulsada: "**+keyCode);   **if**( keyCode == 32) {  *nivel*.**orientacionPad** = -0.5f;  }  **if**( keyCode == 29) {  *nivel*.**orientacionPad** = 0.5f;  }  **if**( keyCode == 47) {  *nivel*.**orientacionPadY** = -0.5f;  *nivel*.**padAbajoPulsado** = **true**;  }  **if**( keyCode == 51) {  *nivel*.**orientacionPadY** = 0.5f;  *nivel*.**padArribaPulsado** = **true**;  }  **if**( keyCode == 33) {  *nivel*.**botonSaltarPulsado** = **true**;  }  **if**( keyCode == 62) {  *nivel*.**botonDispararPulsado** = **true**;  }  **return super**.onKeyDown(keyCode, event); } |

## Tiles solidos con inclinación

### Descripción

Incluir un nuevo tipo de tiles que pueda ser utilizado para crear cuestas o pendientes. Ejemplo: un tile con inclinación de 45 grados.

Este tipo de tiles se representará en el mapa de nivel con la letra L.

## Enemigos más inteligentes

### Descripción

Modificar el comportamiento de los enemigos para que actúen de una manera más inteligente, por ejemplo: que se muevan directamente hacia el jugador, traten de esquivar sus disparos, etc.

## Punto de salvado

### Descripción

Incluir un elemento que al colisionar con el permita al jugador salvar esa posición, en caso de perder el jugador debería comenzar desde ese punto del mapa (en lugar de desde el punto inicial del nivel). Este tipo de elemento se representará en el mapa de nivel con la letra A.

### Creación del objeto SavePoint

Dentro del paquete recolectables creamos la clase SavePoint.

|  |
| --- |
| **public class** SavePoint **extends** Modelo {    **double xSalvada**;  **double ySalvada**;  **boolean salvado**;   **public** SavePoint(Context context, **double** x, **double** y) {  **super**(context, x, y, 32,40);   **this**.**y** = y - **altura**/2;;   **xSalvada** = x;  **ySalvada** = y-Tile.*altura*;   **imagen** = CargadorGraficos.*cargarDrawable*(context, R.drawable.***flagyellow\_down***);  }    **public void** dibujar(Canvas canvas){  **int** yArriba = (**int**) **y** - **altura** / 2 - Nivel.*scrollEjeY*;  **int** xIzquierda = (**int**) **x** - **ancho** / 2 - Nivel.*scrollEjeX*;   **imagen**.setBounds(xIzquierda, yArriba, xIzquierda  + **ancho**, yArriba + **altura**);  **imagen**.draw(canvas);   }   **public double** getxSalvada(){  **return xSalvada**;  }  **public double** getySalvada(){  **return ySalvada**;  }  **public boolean** getSalvado(){  **return salvado**;  }  **public void** setSalvado(**boolean** salvado){  **this**.**salvado**=salvado;  }   **public void** changeImage() {  **imagen** = CargadorGraficos.*cargarDrawable*(**context**, R.drawable.***flagyellow***);  } } |

### Modificación de la clase Nivel

Se creará una lista de SavePoints ya que podrá haber más de uno por nivel y se rellenará con los SavePoints determinados en el mapa de tiles con una A. Se añadirá también una variable booleana para saber que ha sido pisada al menos un SavePoint.

Se modificará el método inicializar.

|  |
| --- |
| **if**(!**checkPoint**) {  **recolectables** = **new** ArrayList<>();  **savePoints** = **new** ArrayList<>();  *scrollEjeX* = 0;  *scrollEjeY* = 0;  **mensaje** = CargadorGraficos.*cargarBitmap*(**context**, R.drawable.***description***);  **nivelPausado** = **true**;  GameView.*contador*.reiniciarPuntuacion();  **enemigos** = **new** LinkedList<Enemigo>();  **disparosJugador** = **new** LinkedList<Disparo>();  **disparosEnemigos** = **new** LinkedList<Disparo>();  **fondos** = **new** Fondo[2];  **fondos**[0] = **new** Fondo(**context**, CargadorGraficos.*cargarBitmap*(**context**,  R.drawable.***capa1***), 0);  **fondos**[1] = **new** Fondo(**context**, CargadorGraficos.*cargarBitmap*(**context**,  R.drawable.***capa2***), 1f);  **iconosVida** = **new** IconoVida[3];   **iconosVida**[0] = **new** IconoVida(**context**, GameView.*pantallaAncho* \* 0.05,  GameView.*pantallaAlto* \* 0.1);  **iconosVida**[1] = **new** IconoVida(**context**, GameView.*pantallaAncho* \* 0.15,  GameView.*pantallaAlto* \* 0.1);  **iconosVida**[2] = **new** IconoVida(**context**, GameView.*pantallaAncho* \* 0.25,  GameView.*pantallaAlto* \* 0.1);  inicializarMapaTiles();  *scrollEjeY* = (**int**) (altoMapaTiles() - tilesEnDistanciaY(GameView.*pantallaAlto*)) \* Tile.*altura*; } |

Se modificará el método aplicarReglasDeMovimiento.

|  |
| --- |
| **...**  **for** (Iterator<SavePoint> iterator = **savePoints**.iterator(); iterator.hasNext();) {  SavePoint savePoint = iterator.next();  **if**(**jugador**.colisiona(savePoint) && !savePoint.getSalvado()){  **jugador**.setPosicionInicial(savePoint.getxSalvada(),savePoint.getySalvada());  savePoint.setSalvado(**true**);  savePoint.changeImage();  **checkPoint**=**true**;  } }  … |

Con esto ya tendríamos los puntos de guardado insertados en el videojuego.

## Puertas

### Descripción

Tipo de elemento especifico que trasladan al jugador de un punto a otro del nivel.

Las puertas se definirán en pareja, siempre estarán conectadas de dos en dos. Este tipo de elemento se representará en el mapa de nivel con los números 9,8,7,5,4.

Un número para cada par de puertas conectadas, por lo tanto podría haber un máximo de 10 puertas en el nivel.

### Creación del objeto Puerta

|  |
| --- |
| **public class** Puerta **extends** Modelo {   **public int numero**;   **public** Puerta(Context context, **double** x, **double** y, **int** numero) {  **super**(context, x, y, 64, 40);  **this**.**numero** = numero;  **imagen** = CargadorGraficos.*cargarDrawable*(context, R.drawable.***puerta***);  }   **public double** getX(){  **return x**;  }  **public double** getY(){  **return y**;  }   **public void** dibujar(Canvas canvas){  **int** yArriva = (**int**) **y** - **altura** - Nivel.*scrollEjeY*;  **int** xIzquierda = (**int**) **x** - **ancho**/2 - Nivel.*scrollEjeX*;   **imagen**.setBounds(xIzquierda, yArriva, xIzquierda  + **ancho**, yArriva + **altura**);  **imagen**.draw(canvas);  } } |

### Modificación del Nivel

Primero se añadirán un conjunto de puertas, se inicializará vacío, posteriormente se dibujaran antes siempre que el personaje, se añadirán también a la inicialización del mapa de tiles, una variable booleana para saber cuándo entra por la puerta y posteriormente se modificara el método aplicarReglasMovimiento.

|  |
| --- |
| **for**(Puerta puerta:**puertas**){  **for**(Puerta puerta2:**puertas**){  **if**(puerta!=puerta2){  **if**(puerta.**numero**==puerta2.**numero**){  **if**(**jugador**.colisiona(puerta) && **entraPuerta** && System.*currentTimeMillis*()-**tiempoEspera** >=2000){  **jugador**.**x** = puerta2.getX();  **jugador**.**y** = puerta2.getY()-Tile.*altura*;  **entraPuerta**=**false**;  **tiempoEspera** = System.*currentTimeMillis*();  }  **else if**(**jugador**.colisiona(puerta2) && **entraPuerta** && System.*currentTimeMillis*()-**tiempoEspera** >=2000){  **jugador**.**x** = puerta.getX();  **jugador**.**y** = puerta.getY()-Tile.*altura*;  **entraPuerta**=**false**;  **tiempoEspera** = System.*currentTimeMillis*();  }  }  }  } } |

### Modificación del Pad

Habrá que crear una pequeña área para poder pulsar solo en el eje y para poder entrar en la puerta.

|  |
| --- |
| **public int** getOrientacionX(**float** cliclX) {  **int** xPulsado = (**int**) (**x** - cliclX);  **return** xPulsado>15 || xPulsado<-15 ? xPulsado : 0; } **public int** getOrientacionY(**float** clicY){  **int** yPulsado = (**int**) (**y** - clicY);  **return** yPulsado>15 || yPulsado<-15 ? yPulsado : 0; } |

### Modificación del GameView

Por ultimo cuando se pulsa el pad habrá que controlar donde se pulsa para saber si debe entrar por la puerta, en caso de que así sea se cambiara la variable de entrar puerta a true.

|  |
| --- |
| **float** orientacion = **pad**.getOrientacionX(**x**[i]); **float** orientacionY = **pad**.getOrientacionY(**y**[i]); *// Si almenosuna pulsacion está en el pad* **if** (**accion**[i] != **ACTION\_UP**) {  **if**(orientacion!=0) {  pulsacionPadMover = **true**;  *nivel*.**orientacionPad** = orientacion;  }  **else if**(orientacionY!=0){  pulsacionPadMover=**false**;  *nivel*.**entraPuerta**=**true**;  } } |

## Disparo con gravedad

### Descripción

Aplicar la fuerza de gravedad al disparo del Jugador, debe "rebotar" contra el suelo e ir perdiendo la velocidad progresivamente, cuando pierde toda la velocidad desaparece.

### Modificación del disparoJugador

Se añadirá una variable rebotes.

|  |
| --- |
| **public int rebotes** = 5; |

### Modificación del Nivel

Se modifica el método aplicarReglasMovimiento a la altura de los disparos del jugador.

|  |
| --- |
| **for** (Iterator<Disparo> iterator = **disparosJugador**.iterator(); iterator.hasNext(); ) {   DisparoJugador disparoJugador = (DisparoJugador) iterator.next();   **int** tileXDisparo = (**int**) disparoJugador.**x** / Tile.*ancho*;  **int** tileYDisparoInferior =  (**int**) (disparoJugador.**y** + disparoJugador.**cAbajo**) / Tile.*altura*;   **int** tileYDisparoSuperior =  (**int**) (disparoJugador.**y** - disparoJugador.**cArriba**) / Tile.*altura*;   **if** (disparoJugador.**velocidadX** > 0) {  *// Tiene delante un tile pasable, puede avanzar.* **if** (tileXDisparo + 1 <= anchoMapaTiles() - 1 &&  **mapaTiles**[tileXDisparo + 1][tileYDisparoInferior].**tipoDeColision** == Tile.***PASABLE*** &&  **mapaTiles**[tileXDisparo + 1][tileYDisparoSuperior].**tipoDeColision** == Tile.***PASABLE***) {   disparoJugador.**x** += disparoJugador.**velocidadX**;  disparoJugador.**y** += **velocidadGravedad**;   }**else if** (tileXDisparo + 1 <= anchoMapaTiles() - 1 &&  **mapaTiles**[tileXDisparo + 1][tileYDisparoInferior].**tipoDeColision** == Tile.***SOLIDO*** &&  **mapaTiles**[tileXDisparo + 1][tileYDisparoSuperior].**tipoDeColision** == Tile.***PASABLE*** && disparoJugador.**rebotes**>0 ) {   disparoJugador.**x** += disparoJugador.**velocidadX**/2;  disparoJugador.**y** -= **velocidadGravedad**;  disparoJugador.**rebotes**--;   } **else if** (tileXDisparo <= anchoMapaTiles() - 1) {   **int** TileDisparoBordeDerecho = tileXDisparo \* Tile.*ancho* + Tile.*ancho*;  **double** distanciaX =  TileDisparoBordeDerecho - (disparoJugador.**x** + disparoJugador.**cDerecha**);   **if** (distanciaX > 0) {  **double** velocidadNecesaria =  Math.*min*(distanciaX, disparoJugador.**velocidadX**);  disparoJugador.**x** += velocidadNecesaria;  } **else** {  iterator.remove();  **continue**;  }  }  }   *// izquierda* **if** (disparoJugador.**velocidadX** <= 0) {  **if** (tileXDisparo - 1 >= 0 &&  tileYDisparoSuperior < altoMapaTiles() - 1 &&  **mapaTiles**[tileXDisparo - 1][tileYDisparoSuperior].**tipoDeColision** ==  Tile.***PASABLE*** &&  **mapaTiles**[tileXDisparo - 1][tileYDisparoInferior].**tipoDeColision** ==  Tile.***PASABLE***) {   disparoJugador.**x** += disparoJugador.**velocidadX**;  disparoJugador.**y** += **velocidadGravedad**;  *// No tengo un tile PASABLE detras  // o es el INICIO del nivel o es uno SOLIDO* }**else if**(tileXDisparo - 1 >= 0 &&  tileYDisparoSuperior < altoMapaTiles() - 1 &&  **mapaTiles**[tileXDisparo - 1][tileYDisparoSuperior].**tipoDeColision** ==  Tile.***PASABLE*** &&  **mapaTiles**[tileXDisparo - 1][tileYDisparoInferior].**tipoDeColision** ==  Tile.***SOLIDO*** && disparoJugador.**rebotes**>0){   disparoJugador.**x** += disparoJugador.**velocidadX** / 2;  disparoJugador.**y** -= **velocidadGravedad**;  disparoJugador.**rebotes**--;   }**else if** (tileXDisparo >= 0) {  *// Si en el propio tile del jugador queda espacio para  // avanzar más, avanzo* **int** TileDisparoBordeIzquierdo = tileXDisparo \* Tile.*ancho*;  **double** distanciaX =  (disparoJugador.**x** - disparoJugador.**cIzquierda**) - TileDisparoBordeIzquierdo;   **if** (distanciaX > 0) {  **double** velocidadNecesaria =  Utilidades.*proximoACero*(-distanciaX, disparoJugador.**velocidadX**);  disparoJugador.**x** += velocidadNecesaria;  } **else** {  iterator.remove();  **continue**;  }  }  }  **for** (Enemigo enemigo : **enemigos**) {  **if** (disparoJugador.colisiona(enemigo)) {  **gameView**.**gestorAudio**.reproducirSonido(GestorAudio.***SONIDO\_ENEMIGO\_GOLPEADO***);  enemigo.destruir();  iterator.remove();  **break**;  }  }  **for**(Plataforma plataforma:**plataformas**){  **if**(disparoJugador.colisiona(plataforma)){  iterator.remove();  **continue**;  }  }  **for**(Caja caja:**cajas**){  **if**(disparoJugador.colisiona(caja)){  iterator.remove();  **continue**;  }  }  **for**(Puerta puerta:**puertas**){  **if**(disparoJugador.colisiona(puerta)){  iterator.remove();  **continue**;  }  } } |

## Disparo direccional

### Descripción

Cambiar el botón de disparar por un joystick direccional que permita especificar el ángulo del disparo en el eje X e Y.

### Modificaciones en Disparo

|  |
| --- |
| **public double** velocidadY = 5; **public int angulo**; **public int impulsos**=15; |

### Modificaciones BotonDisparo

Habra que añadir 3 metodos, dos para controlar donde se pulsan en x e y, un tercero para controlar el angulo con el que saldrá.

|  |
| --- |
| **public int** getOrientacionX(**float** cliclX) {  **return** (**int**) (**x** - cliclX);  } **public int** getOrientacionY(**float** clicY){  **return** (**int**) (**y** - clicY); }  **public int** calcularAngulo(**float** orientacionX,**float** orientacionY){  **if**(orientacionX<15 && orientacionX>-15){  **if**(orientacionY>25){  **return** 90;  }  **if**(orientacionY<-20){  **return** -90;  }  }  **else if**(orientacionX>15 || orientacionX<-15){  **if**(orientacionY>15){  **return** 45;  }  **if**(orientacionY>5 && orientacionY<15){  **return** 0;  }  **if**(orientacionY<-15){  **return** -45;  }  }  **return** 0; } |

### Modificaciones Nivel

Habra que modificar como se comportarn los disparos al moverse

|  |
| --- |
| **for** (Iterator<Disparo> iterator = **disparosJugador**.iterator(); iterator.hasNext(); ) {   DisparoJugador disparoJugador = (DisparoJugador) iterator.next();   **int** tileXDisparo = (**int**) disparoJugador.**x** / Tile.*ancho*;  **int** tileYDisparoInferior =  (**int**) (disparoJugador.**y** + disparoJugador.**cAbajo**) / Tile.*altura*;   **int** tileYDisparoSuperior =  (**int**) (disparoJugador.**y** - disparoJugador.**cArriba**) / Tile.*altura*;   **if** (disparoJugador.**velocidadX** > 0) {  *// Tiene delante un tile pasable, puede avanzar.* **if** (tileXDisparo + 1 <= anchoMapaTiles() - 1 &&  **mapaTiles**[tileXDisparo + 1][tileYDisparoInferior].**tipoDeColision** == Tile.***PASABLE*** &&  **mapaTiles**[tileXDisparo + 1][tileYDisparoSuperior].**tipoDeColision** == Tile.***PASABLE***) {   **if**(disparoJugador.**angulo**==0) {  disparoJugador.**x** += disparoJugador.**velocidadX**;  }  **else if**(disparoJugador.**angulo**==45 && disparoJugador.**y**-3>0 && disparoJugador.**impulsos**>0) {  disparoJugador.**x** += disparoJugador.**velocidadX**;  disparoJugador.**y** -= disparoJugador.**velocidadY**;  disparoJugador.**impulsos**--;  }  **else if**(disparoJugador.**angulo**==90 && disparoJugador.**y**-3>0 && disparoJugador.**impulsos**>0) {  disparoJugador.**y** -= disparoJugador.**velocidadY**;  disparoJugador.**impulsos**--;  }  **else if**(disparoJugador.**angulo**==-45) {  disparoJugador.**x** += disparoJugador.**velocidadX**;  disparoJugador.**y** += disparoJugador.**velocidadY**;  }  **else if**(disparoJugador.**angulo**==-90) {  disparoJugador.**y** += disparoJugador.**velocidadY**-**velocidadGravedad**;  }  **else if**((disparoJugador.**angulo**==45 || disparoJugador.**angulo**==90) && disparoJugador.**y**-3<0){  iterator.remove();  **continue**;  }  **else if**(disparoJugador.**impulsos**<=0 && disparoJugador.**angulo**!=90 && disparoJugador.**angulo**!=-90){  disparoJugador.**x** += disparoJugador.**velocidadX**;  }  disparoJugador.**y** += **velocidadGravedad**;   }**else if** (tileXDisparo + 1 <= anchoMapaTiles() - 1 &&  **mapaTiles**[tileXDisparo + 1][tileYDisparoInferior].**tipoDeColision** == Tile.***SOLIDO*** &&  **mapaTiles**[tileXDisparo + 1][tileYDisparoSuperior].**tipoDeColision** == Tile.***PASABLE*** && disparoJugador.**rebotes**>0 ) {   disparoJugador.**x** += disparoJugador.**velocidadX**/2;  disparoJugador.**y** -= **velocidadGravedad**;  disparoJugador.**rebotes**--;   } **else if** (tileXDisparo <= anchoMapaTiles() - 1) {   **int** TileDisparoBordeDerecho = tileXDisparo \* Tile.*ancho* + Tile.*ancho*;  **double** distanciaX =  TileDisparoBordeDerecho - (disparoJugador.**x** + disparoJugador.**cDerecha**);   **if** (distanciaX > 0) {  **double** velocidadNecesaria =  Math.*min*(distanciaX, disparoJugador.**velocidadX**);  disparoJugador.**x** += velocidadNecesaria;  } **else** {  iterator.remove();  **continue**;  }  }  }   *// izquierda* **if** (disparoJugador.**velocidadX** <= 0) {  **if** (tileXDisparo - 1 >= 0 &&  tileYDisparoSuperior < altoMapaTiles() - 1 &&  **mapaTiles**[tileXDisparo - 1][tileYDisparoSuperior].**tipoDeColision** ==  Tile.***PASABLE*** &&  **mapaTiles**[tileXDisparo - 1][tileYDisparoInferior].**tipoDeColision** ==  Tile.***PASABLE***) {   **if**(disparoJugador.**angulo**==0) {  disparoJugador.**x** += disparoJugador.**velocidadX**;  }  **else if**(disparoJugador.**angulo**==45 && disparoJugador.**y**-3>0 && disparoJugador.**impulsos**>0) {  disparoJugador.**x** += disparoJugador.**velocidadX**;  disparoJugador.**y** -= disparoJugador.**velocidadY**;  disparoJugador.**impulsos**--;  }  **else if**(disparoJugador.**angulo**==90 && disparoJugador.**y**-3>0 && disparoJugador.**impulsos**>0) {  disparoJugador.**y** -= 4;  disparoJugador.**impulsos**--;  }  **else if**(disparoJugador.**angulo**==-45) {  disparoJugador.**x** += disparoJugador.**velocidadX**;  disparoJugador.**y** += disparoJugador.**velocidadY**;  }  **else if**(disparoJugador.**angulo**==-90) {  disparoJugador.**y** += disparoJugador.**velocidadY**-**velocidadGravedad**;  }  **else if**((disparoJugador.**angulo**==45 || disparoJugador.**angulo**==90) && disparoJugador.**y**-3<0){  iterator.remove();  **continue**;  }  **else if**(disparoJugador.**impulsos**<=0 && disparoJugador.**angulo**!=90 && disparoJugador.**angulo**!=-90){  disparoJugador.**x** += disparoJugador.**velocidadX**;  }  disparoJugador.**y** += **velocidadGravedad**;  *// No tengo un tile PASABLE detras  // o es el INICIO del nivel o es uno SOLIDO* }**else if**(tileXDisparo - 1 >= 0 &&  tileYDisparoSuperior < altoMapaTiles() - 1 &&  **mapaTiles**[tileXDisparo - 1][tileYDisparoSuperior].**tipoDeColision** ==  Tile.***PASABLE*** &&  **mapaTiles**[tileXDisparo - 1][tileYDisparoInferior].**tipoDeColision** ==  Tile.***SOLIDO*** && disparoJugador.**rebotes**>0){   disparoJugador.**x** += disparoJugador.**velocidadX** / 2;  disparoJugador.**y** -= **velocidadGravedad**;  disparoJugador.**rebotes**--;   }**else if** (tileXDisparo >= 0) {  *// Si en el propio tile del jugador queda espacio para  // avanzar más, avanzo* **int** TileDisparoBordeIzquierdo = tileXDisparo \* Tile.*ancho*;  **double** distanciaX =  (disparoJugador.**x** - disparoJugador.**cIzquierda**) - TileDisparoBordeIzquierdo;   **if** (distanciaX > 0) {  **double** velocidadNecesaria =  Utilidades.*proximoACero*(-distanciaX, disparoJugador.**velocidadX**);  disparoJugador.**x** += velocidadNecesaria;  } **else** {  iterator.remove();  **continue**;  }  }  } |

### Modificaciones GameView

Por ultimo habrá que modificar como se disparara ya que se calculará un angulo aproximado según donde se pulse en el pad.

|  |
| --- |
| **int** anguloDisparo = **botonDisparar**.calcularAngulo(orientacion,orientacionY);  **if** (**accion**[i] == **ACTION\_DOWN**) {  **if**(orientacion!=0 && orientacionY!=0) {  *nivel*.**botonDispararPulsado** = **true**;  *nivel*.**orientacionDisparoY** = orientacionY;  *nivel*.**anguloDisparo**=anguloDisparo;  }  **else if**(orientacion!=0 && orientacionY==0) {  *nivel*.**botonDispararPulsado** = **true**;  *nivel*.**orientacionDisparoY** = orientacionY;  *nivel*.**anguloDisparo**=anguloDisparo;  }  **else**{  *nivel*.**botonDispararPulsado**=**true**;  }  } |

## Completar la interfaz del juego

### Descripción

Incluir un menú de inicio, selección de nivel y efectos de sonido durante el juego.

### Creación de las pantallas

MenuActivity

|  |
| --- |
| **public class** MenuActivity **extends** Activity **implements** View.OnClickListener {   ImageButton **nuevoJuego**;   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  *// Establecer el control de sonido multimedia como predefinido* **this**.setVolumeControlStream(AudioManager.***STREAM\_MUSIC***);   *// Pantalla completa, sin titulo* requestWindowFeature(Window.***FEATURE\_NO\_TITLE***);  getWindow().setFlags(LayoutParams.***FLAG\_FULLSCREEN***,  LayoutParams.***FLAG\_FULLSCREEN***);  setContentView(R.layout.***activity\_menu***);   **nuevoJuego** = (ImageButton) findViewById(R.id.***imageButton***);  **nuevoJuego**.setOnClickListener(**this**);   }    @Override  **public void** onClick(View v) {  **if**(v==**nuevoJuego**){  Intent actividadJuego = **new** Intent(MenuActivity.**this**,  SeleccionModo.**class**);  startActivity(actividadJuego);  finish();  }  } } |

Layout

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  xmlns:tools="http://schemas.android.com/tools"  android:id="@+id/activity\_menu"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:paddingBottom="@dimen/activity\_vertical\_margin"  android:paddingLeft="@dimen/activity\_horizontal\_margin"  android:paddingRight="@dimen/activity\_horizontal\_margin"  android:paddingTop="@dimen/activity\_vertical\_margin"  tools:context="com.plataformas.MenuActivity"  android:background="@drawable/fondo\_menu\_principal"**>   <**ImageButton  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  app:srcCompat="@drawable/boton\_play"  android:id="@+id/imageButton"  android:layout\_alignParentBottom="true"  android:layout\_centerHorizontal="true"  android:background="@drawable/boton\_play"** /> </**RelativeLayout**> |

SeleccionModoActivity

|  |
| --- |
| **public class** SeleccionModo **extends** Activity **implements** View.OnClickListener {   ImageButton **unJugador**;  ImageButton **dosJugadores**;   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  *// Establecer el control de sonido multimedia como predefinido* **this**.setVolumeControlStream(AudioManager.***STREAM\_MUSIC***);   *// Pantalla completa, sin titulo* requestWindowFeature(Window.***FEATURE\_NO\_TITLE***);  getWindow().setFlags(WindowManager.LayoutParams.***FLAG\_FULLSCREEN***,  WindowManager.LayoutParams.***FLAG\_FULLSCREEN***);  setContentView(R.layout.***activity\_seleccion\_modo***);   **unJugador** = (ImageButton) findViewById(R.id.***unJugadorButton***);  **dosJugadores** = (ImageButton) findViewById(R.id.***dosJugadoresButton***);  **unJugador**.setOnClickListener(**this**);   }    @Override  **public void** onClick(View v) {  **if**(v==**unJugador**){  Intent actividadJuego = **new** Intent(SeleccionModo.**this**,  MainActivity.**class**);  actividadJuego.putExtra(**"Modo"**,**"Un jugador"**);  startActivity(actividadJuego);  finish();  }  **if**(v==**dosJugadores**){  Intent actividadJuego = **new** Intent(SeleccionModo.**this**,  MainActivity.**class**);  actividadJuego.putExtra(**"Modo"**,**"Dos jugadores"**);  startActivity(actividadJuego);  finish();  }  } } |

Layout

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  xmlns:tools="http://schemas.android.com/tools"  android:id="@+id/activity\_seleccion\_modo"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:paddingBottom="@dimen/activity\_vertical\_margin"  android:paddingLeft="@dimen/activity\_horizontal\_margin"  android:paddingRight="@dimen/activity\_horizontal\_margin"  android:paddingTop="@dimen/activity\_vertical\_margin"  tools:context="com.plataformas.SeleccionModo"  android:background="@drawable/fondo\_seleccionar\_nivel"**>   <**ImageButton  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  app:srcCompat="@drawable/boton\_one"  android:layout\_alignParentTop="true"  android:layout\_centerHorizontal="true"  android:layout\_marginTop="54dp"  android:id="@+id/unJugadorButton"  android:background="@drawable/boton\_one"** />   <**ImageButton  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  app:srcCompat="@drawable/boton\_two"  android:layout\_marginTop="17dp"  android:id="@+id/dosJugadoresButton"  android:layout\_below="@+id/unJugadorButton"  android:layout\_alignLeft="@+id/unJugadorButton"  android:layout\_alignStart="@+id/unJugadorButton"  android:background="@drawable/boton\_two"** /> </**RelativeLayout**> |

### Creación del Gestor de audio

Dentro del paquete gestores.

|  |
| --- |
| **public class** GestorAudio **implements** MediaPlayer.OnPreparedListener {  **public static final int *SONIDO\_DISPARO\_JUGADOR*** = 1;  **public static final int *SONIDO\_SALTO\_JUGADOR*** = 2;  **public static final int *SONIDO\_ENEMIGO\_GOLPEADO*** = 3;  **public static final int *SONIDO\_JUGADOR\_GOLPEADO*** = 4;  *// Pool de sonidos, para efectos de sonido.  // Suele fallar el utilizar ficheros de sonido demasiado grandes* **private** SoundPool **poolSonidos**;  **private** HashMap<Integer, Integer> **mapSonidos**;  **private** Context **contexto**;  *// Media Player para bucle de sonido de fondo1.* **private** MediaPlayer **sonidoAmbiente**;  **private** AudioManager **gestorAudio**;   **private static** GestorAudio *instancia* = **null**;   **public static** GestorAudio getInstancia(Context contexto,  **int** idMusicaAmbiente) {  **synchronized** (GestorAudio.**class**) {  **if** (*instancia* == **null**) {  *instancia* = **new** GestorAudio();  *instancia*.initSounds(contexto, idMusicaAmbiente);  }  **return** *instancia*;  }  }    **public static** GestorAudio getInstancia() {  **return** *instancia*;  }   **private** GestorAudio() {  }   **public void** initSounds(Context contexto, **int** idMusicaAmbiente) {  **this**.**contexto** = contexto;  **poolSonidos** = **new** SoundPool(4, AudioManager.***STREAM\_MUSIC***, 100);  **mapSonidos** = **new** HashMap<Integer, Integer>();  **gestorAudio** = (AudioManager) contexto  .getSystemService(Context.***AUDIO\_SERVICE***);  **sonidoAmbiente** = MediaPlayer.*create*(contexto, idMusicaAmbiente);  **sonidoAmbiente**.setLooping(**true**);  **sonidoAmbiente**.setVolume(1, 1);  }   **public void** reproducirMusicaAmbiente() {  **try** {  **if** (!**sonidoAmbiente**.isPlaying()) {  **try** {  **sonidoAmbiente**.setOnPreparedListener(**this**);  **sonidoAmbiente**.prepareAsync();  } **catch** (Exception e) {  }  }  } **catch** (Exception e) {  }  }   **public void** pararMusicaAmbiente() {  **if** (**sonidoAmbiente**.isPlaying()) {  **sonidoAmbiente**.stop();  }  }   @Override  **public void** onPrepared(MediaPlayer mp) {  mp.start();  }   **public void** registrarSonido(**int** index, **int** SoundID) {  **mapSonidos**.put(index, **poolSonidos**.load(**contexto**, SoundID, 1));  }   **public void** reproducirSonido(**int** index) {  **float** volumen =  **gestorAudio**.getStreamVolume(AudioManager.***STREAM\_MUSIC***);   volumen =  volumen / **gestorAudio**.getStreamMaxVolume(AudioManager.***STREAM\_MUSIC***);   **poolSonidos**.play(  (Integer) **mapSonidos**.get(index),  volumen, volumen, 1, 0, 1f);  }  } |

### Modificación del GameView

Creación de la variable publica gestorAudio y su inicialización en el constructor llamando al siguiente método.

|  |
| --- |
| **public void** inicializarGestorAudio(Context context) {  **gestorAudio** = GestorAudio.*getInstancia*(context, R.raw.***musica\_fondo***);  **gestorAudio**.reproducirMusicaAmbiente();  **gestorAudio**.registrarSonido(GestorAudio.***SONIDO\_DISPARO\_JUGADOR***,  R.raw.***lanzar\_objeto***);  **gestorAudio**.registrarSonido(GestorAudio.***SONIDO\_SALTO\_JUGADOR***,R.raw.***salto\_jugador***);  **gestorAudio**.registrarSonido(GestorAudio.***SONIDO\_ENEMIGO\_GOLPEADO***,R.raw.***disparo\_golpea***);  **gestorAudio**.registrarSonido(GestorAudio.***SONIDO\_JUGADOR\_GOLPEADO***,R.raw.***jugador\_golpeado***); } |

### Modificación del Nivel

Llamar a los diferentes sonidos donde se produzcan las interacciones

Saltar

|  |
| --- |
| **if** (**botonSaltarPulsado**) {  **gameView**.**gestorAudio**.reproducirSonido(GestorAudio.***SONIDO\_SALTO\_JUGADOR***);  **botonSaltarPulsado** = **false**; } |

Disparar

|  |
| --- |
| **if** (**botonDispararPulsado**) {  **gameView**.**gestorAudio**.reproducirSonido(GestorAudio.***SONIDO\_DISPARO\_JUGADOR***);  **disparosJugador**.add(**new** DisparoJugador(**context**, **jugador**.**x**, **jugador**.**y**, **jugador**.**orientacion**));  **botonDispararPulsado** = **false**; } |

Ser golpeado por disparo enemigo

|  |
| --- |
| **if** (disparoEnemigo.colisiona(**jugador**)) {  **gameView**.**gestorAudio**.reproducirSonido(GestorAudio.***SONIDO\_JUGADOR\_GOLPEADO***);  **if** (**jugador**.golpeado() <= 0) {  **nivelPausado** = **true**;  **mensaje** = CargadorGraficos.*cargarBitmap*(**context**, R.drawable.***you\_lose***);  inicializar();  **jugador**.restablecerPosicionInicial();  **return**;  } } |

Ser golpeado por enemigo

|  |
| --- |
| **if** (**jugador**.colisiona(enemigo)) {  **gameView**.**gestorAudio**.reproducirSonido(GestorAudio.***SONIDO\_JUGADOR\_GOLPEADO***);  **if** (**jugador**.golpeado() <= 0) {  **nivelPausado** = **true**;  **mensaje** = CargadorGraficos.*cargarBitmap*(**context**, R.drawable.***you\_lose***);  inicializar();  **jugador**.restablecerPosicionInicial();  **return**;  } } |

Golpear enemigo con disparo

|  |
| --- |
| **if** (disparoJugador.colisiona(enemigo)) {  **gameView**.**gestorAudio**.reproducirSonido(GestorAudio.***SONIDO\_ENEMIGO\_GOLPEADO***);  enemigo.destruir();  iterator.remove();  **break**; } |

## Multijugador con teclado

### Descripción

Permitir que haya dos jugadores en pantalla de forma simultánea, cada jugador se manejará con diferentes teclas del teclado.

El modo de juego será cooperativo, y un jugador no podrá seguir avanzando si el otro no avanza también.

## Uso de una librería de físicas externa

### Descripción

<http://www.jbox2d.org>

Para la iteración del jugador con el mapa de tiles.

## Otras ampliaciones propuestas por el alumno

### Descripción

# Video de muestra de ampliaciones