|  |
| --- |
| Software de entretenimiento y Videojuegos Jordán Pascual : pascualjordan@uniovi.es |

|  |  |
| --- | --- |
| 2 | Naves y Plataformas - Parte 1 |

# Introducción

En esta práctica vamos a finalizar el **juego de Naves**. Una vez lo tengamos listo comenzaremos lo tomaremos como base para crear un juego de tipo **plataformas** algo más complejo.

**Marcador de puntos – Icono de puntos**

Dentro de **GameLayer.js** vamos a incluir un icono/fondo de una estrella (lo colocaremos junto al texto que indica cuantos puntos tenemos).

|  |
| --- |
| **class** GameLayer **extends** Layer {   constructor() {  **super**();  **this**.iniciar();  }   iniciar() {  **this**.**fondoPuntos** =  **new** Fondo(***imagenes***.**icono\_puntos**, 480\*0.85,320\*0.05);   **this**.**jugador** = **new** Jugador(50, 50); |

Recordamos que el **fondoPuntos** es un elemento que también se debe **dibujar** (los elementos de interfaz se suelen dibujar el último lugar).

|  |
| --- |
| dibujar (){  **this**.**fondo**.dibujar();  **for** (**var** i=0; i < **this**.**disparosJugador**.length; i++) {  **this**.**disparosJugador**[i].dibujar();  }  **this**.**jugador**.dibujar();  **for** (**var** i=0; i < **this**.**enemigos**.length; i++){  **this**.**enemigos**[i].dibujar();  }  **this**.**fondoPuntos**.dibujar(); } |

Abrimos la aplicación y comprobamos que el icono aparece sin errores.



**Marcador de puntos – Texto**

Creamos un nuevo fichero en **modelos/Texto.js** , realmente este elemento no es un “modelo” de tipo imagen, su representación en la pantalla es una forma primitiva Texto.

|  |
| --- |
| **class** Texto {   constructor(valor, x, y) {  **this**.**valor** = valor;  **this**.**x** = x;  **this**.**y** = y;  }   dibujar (){  ***contexto***.**font** = **"20px Arial"**;  ***contexto***.**fillStyle** = **"white"**;  ***contexto***.**textAlign** = **"left"**;  ***contexto***.fillText(**this**.**valor**,**this**.**x**,**this**.**y**);  }  } |

Agregamos el Script **modelos/Texto.js** en el I**ndex.html**

|  |
| --- |
| <**script src="src/modelos/Texto.js"**></**script**> <**script src="src/modelos/Modelo.js"**></**script**> |

En el **GameLayer** vamos a crear un objeto de tipo Texto con el nombre **puntos**, mostrará la cantidad de enemigos eliminados. El valor inicial será 0.

|  |
| --- |
| iniciar() {  **this**.**fondoPuntos** =  **new** Fondo(***imagenes***.**icono\_puntos**, 480\*0.85,320\*0.05);   **this**.**puntos** = **new** Texto(0,480\*0.9,320\*0.07 ); |

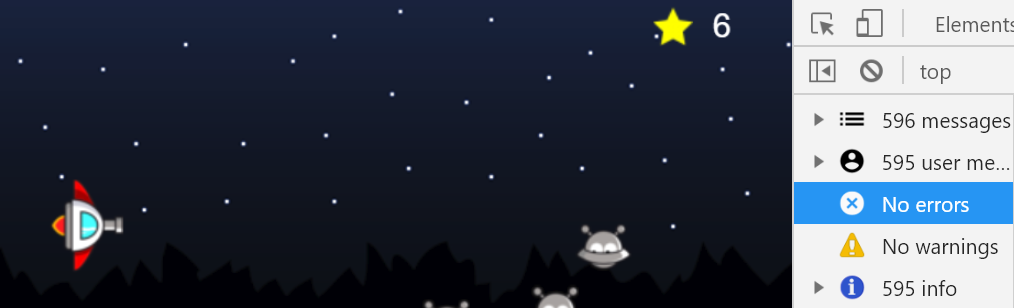
Dibujamos el elemento **puntos**.

|  |
| --- |
| dibujar (){  **this**.**fondo**.dibujar();  **for** (**var** i=0; i < **this**.**disparosJugador**.length; i++) {  **this**.**disparosJugador**[i].dibujar();  }  **this**.**jugador**.dibujar();  **for** (**var** i=0; i < **this**.**enemigos**.length; i++){  **this**.**enemigos**[i].dibujar();  }  **this**.**fondoPuntos**.dibujar();  **this**.**puntos**.dibujar(); } |

Entramos en el método **actualizar()** y aumentamos un punto al destruir un enemigo

|  |
| --- |
| *// colisiones , disparoJugador - Enemigo* **for** (**var** i=0; i < **this**.**disparosJugador**.length; i++){  **for** (**var** j=0; j < **this**.**enemigos**.length; j++){  **if** (**this**.**disparosJugador**[i] != **null** &&  **this**.**enemigos**[j] != **null** &&  **this**.**disparosJugador**[i].colisiona(**this**.**enemigos**[j])) {   **this**.**disparosJugador**.splice(i, 1);  i = i-1;  **this**.**enemigos**.splice(j, 1);  j = j-1;  **this**.**puntos**.**valor**++;  }  } } |

Abrimos la aplicación y comprobamos que los enemigos eliminados se contabilizan sin errores.

****

**Animación básica – Infinita**

Una de las técnicas básicas de animación 2D se basa en el cambio rápido entre imágenes para crear una sensación de animación.

Para evitar tener que cargar cientos de ficheros en un proyecto, se suelen incluir varios “fotogramas” en un mismo fichero, son los conocidos como **Sprite Sheet** (Hoja de sprites, probar a buscar en Google Sprite Sheet). El uso de hojas de sprites mejora significativamente el rendimiento respecto a cargar imágenes una a una.



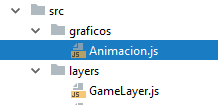
Hoja de sprites de megaman

Aunque podemos manejar **Animaciones** sin **hojas de** **Sprites** en ocasiones se asocia una animación a una **hoja de Sprites** concreta (o a una parte de una).



jugador\_impactado.png -> Animación Impactado

Creamos el directorio **src/graficos** y dentro el fichero **src/graficos/Animacion.js**



La clase **Animación** tiene los siguientes parámetros que el usuario debe especificar:

* **imagenSrc** : URL del fichero png donde está la hoja de Sprites
* **Alto y ancho del modelo:** las dimensiones del elemento en el juego, por ejemplo, el **Jugador** mide 50x50
* **velocidadRefresco:** el Sprite tiene varios fotogramas, ¿Cada cuantas iteraciones del loop se cambia de uno frame a otro?
* **framesTotales:** cuantos fotogramas hay en la hoja de Sprites

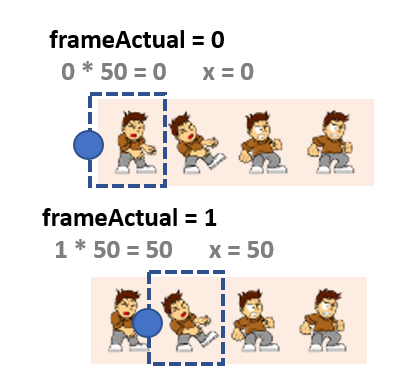
Además, requiere otras 2 variables para su funcionamiento interno:

* **rectanguloDibujo:** este rectángulo indica que frame/fotograma de toda la hoja de Sprites se tiene que dibujar
* **ultimActualización :** guarda de forma temporal cuantas iteraciones hace que se cambió de frame/fotograma

Tiene los siguientes métodos:

* **actualizar().** Mira cuantas iteraciones del juego han pasado desde la última vez que se movió el rectángulo de dibujo. Sí han pasado más de las que indica la **velocidadRefresco** pasa al siguiente Frame de la animación (si llega al último vuelve al primero).

Marca con un“**rectángulo**” de la hoja de sprites que debe dibujarse. El rectángulo siempre tiene las mismas dimensiones, solo cambia su X, donde empieza, que depende de FrameActual. Ejemplo:



* **dibujar().** Pinta en el Canvas la parte de la hoja de sprites marcada por el **rectangulo**

|  |
| --- |
| **class** Animacion {   constructor(imagenSrc, modeloAncho, modeloAlto, velocidadRefresco, framesTotales) {  **this**.**imagen** = **new *Image***();  **this**.**imagen**.src = imagenSrc;   **this**.**modeloAncho** = modeloAncho;  **this**.**modeloAlto** = modeloAlto;  **this**.**velocidadRefresco** = velocidadRefresco;  **this**.**framesTotales** = framesTotales;   **this**.**frameActual** = 0;  **this**.**frameAncho** = **this**.**imagen**.width / **this**.**framesTotales**;  **this**.**frameAlto** = **this**.**imagen**.height;   **this**.**rectanguloDibujo** = {};  **this**.**rectanguloDibujo**.**x** = 0;  **this**.**rectanguloDibujo**.**y** = 0;  **this**.**rectanguloDibujo**.**ancho** = **this**.**frameAncho**;  **this**.**rectanguloDibujo**.**alto** = **this**.**frameAlto**;   **this**.**ultimaActualizacion** = 0;  }   actualizar (){  **this**.**ultimaActualizacion**++;   **if** (**this**.**ultimaActualizacion** > **this**.**velocidadRefresco**) {  **this**.**ultimaActualizacion** = 0;  *// actualizar el frame* **this**.**frameActual**++;  *// Si llega al último frame evuelve al primero* **if** (**this**.**frameActual** >= **this**.**framesTotales**) {  *// reiniciar, es infinita* **this**.**frameActual** = 0;  }  }  *// actualizar el rectangulo (siguiente frame)* **this**.**rectanguloDibujo**.**x** = **this**.**frameActual** \* **this**.**frameAncho**;  }   dibujar (x, y){  ***contexto***.drawImage(  **this**.**imagen**,  **this**.**rectanguloDibujo**.**x**,  **this**.**rectanguloDibujo**.**y**,  **this**.**rectanguloDibujo**.**ancho**,  **this**.**rectanguloDibujo**.**alto**,  x - **this**.**modeloAncho**/2 ,  y - **this**.**modeloAlto**/2,  **this**.**modeloAncho**,  **this**.**modeloAlto**);  }  } |

Agregamos el Script **Animacion.js** al **index.html**.

|  |
| --- |
| <**script src="src/graficos/Animacion.js"**></**script**> <**script src="Main.js"**></**script**> |

|  |
| --- |
| **IMPORTANTE:** la implementación actual de **animación** procesa solo hojas de Sprites:   * De una línea * Donde todos los frames tienen el mismo ancho   Por eso le sirve solo con saber cuántos frame hay para recortarlos  **Tamaño de la imagen / número de frames** : ya sabe cuánto mide cada uno de ancho.    Como veremos en el bloque 2, los motores complejos (frames de diferentes tamaños y varias líneas) llevan un fichero de coordenadas asociado al png. |

Incluimos una variable **animación** en el **Enemigo.js**

La variable **animación** señala la animación que se está ejecutando actualmente

En este caso solo tiene una **aMover**, pero si tuviera varias habría que ir cambiando la referencia, por ejemplo: **animación = aSaltar**

La animación se debe **actualizar()** para que cambie de frame/fotograma cada X iteraciones.

Hay que redefinir el método **dibujar()** del **modelo**, ya no queremos pintar la **imagen**, sino la animación actual a la que apunta la variable **animación.**

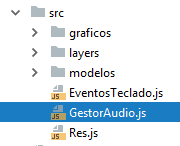
|  |
| --- |
| **class** Enemigo **extends** Modelo {   constructor(x, y) {  **super**(***imagenes***.**enemigo**, x, y)  **this**.**aMover** = **new** Animacion(***imagenes***.**enemigo\_movimiento**,   **this**.**ancho**, **this**.**alto**, 6, 3);  *// Ref a la animación actual* **this**.**animacion** = **this**.**aMover**;   **this**.**vy** = 0;  **this**.**vx** = 1;  }   actualizar (){  *// Actualizar animación* **this**.**animacion**.actualizar();   **this**.**vx** = -1;  **this**.**x** = **this**.**x** + **this**.**vx**;  }   dibujar (){  **this**.**animacion**.dibujar(**this**.**x**, **this**.**y**);  }  } |

Ejecutamos la aplicación, comprobamos que la animación se visualiza y que no hay errores.



**\*Audio (para hacer en casa)**

Creamos el fichero **src/GestorAudio.js** será un fichero que contendrá únicamente funciones, sin clases



Agregamos el Script **fondo.js** al **index.html**.

|  |
| --- |
| <**script src="src/GestorAudio.js"**></**script**> <**script src="src/modelos/Texto.js"**></**script**> <**script src="src/modelos/Modelo.js"**></**script**> |

Creamos una variable para la música de ambiente/fondo llamada **musicaAmbiente**, la música de fondo suele reproducirse en forma de bucle (usamos la propiedad **loop=true**).

Los juegos suelen tener dos tipos de audio

* La música de ambiente que siempre está sonando
* Pequeños efectos, disparar, explosiones, etc. Para estos efectos declaramos un array **efectos** (en el cual los vamos declarando todos), y con la función **reproducirEfecto(efecto)** los reproducimos en un Audio diferente al de la música ambiente.

|  |
| --- |
| **var *musicaAmbiente*** = **new *Audio***(**"res/musica\_ambiente.mp3"**); ***musicaAmbiente***.loop = **true**;  **var *efectos*** = {  **disparo** : **"res/efecto\_disparo.mp3"**,  **explosion** : **"res/efecto\_explosion.mp3"**, }  **function** *reproducirMusica*() {  ***musicaAmbiente***.play(); }  **function** *pararMusica*() {  ***musicaAmbiente***.stop(); }  **function** *reproducirEfecto*( srcEfecto ) {  **var** efecto = **new *Audio***( srcEfecto );  efecto.play(); } |

Incluimos una llamada a **reproducirMusica()** en **iniciar()** del **GameLayer.**

|  |
| --- |
| iniciar() {  *reproducirMusica*();  **this**.**fondoPuntos** =  **new** Fondo(***imagenes***.**icono\_puntos**, 480\*0.85,320\*0.05); |

Abrimos la clase **Jugador** e incluimos un reproducir efecto de disparo en la función **disparar().**

|  |
| --- |
| disparar(){  **if** ( **this**.**tiempoDisparo** == 0) {  *// reiniciar Cadencia* **this**.**tiempoDisparo** = **this**.**cadenciaDisparo**;  *reproducirEfecto*(***efectos***.**disparo**);  **return new** DisparoJugador(**this**.**x**, **this**.**y**); |

Comprobamos que los efectos funcionan de forma correcta. En caso de estar utilizando el **Chrome** abrimos la consola F12 y pulsamos en actualizar la web.

Aparece un error, se debe a la política del Chrome frente los Autoplay antes de que el usuario interactúe con el DOM <https://developers.google.com/web/updates/2017/09/autoplay-policy-changes> Esto se solucionará cuando metamos un menú inicial al juego.

En el resto de los navegadores no aparecerá esta particularidad. Comprobamos que el efecto del disparo sí que suena porque se realiza después de una interacción con la página.

**Disparos fuera de la pantalla**

En la versión actual los disparos del jugador pueden eliminar enemigos incluso cuando estos están fuera de la pantalla. (El disparo sigue avanzando de forma infinita fuera de la zona visible hasta que choque con un enemigo)

Para eliminar este comportamiento podríamos incluir un método **estaEnPantalla()** en la clase **Modelo** (lo incluimos aquí en lugar de en el disparo porque en el futuro será útil en otros casos)

Este método nos permitirá saber si un modelo está en la zona visible del canvas **estaEnPantalla** comprueba básicamente si sus 4 esquinas están dentro de los 4 límites.

|  |
| --- |
| estaEnPantalla (){  **if** ( **this**.**x** - **this**.**ancho**/2 <= 480 &&  **this**.**x** + **this**.**ancho**/2 >= 0 &&  **this**.**y** - **this**.**alto**/2 <= 320 &&  **this**.**y** + **this**.**alto**/2 >= 0 ){  **return true**;  }  **return false**; } |

En el método **actualizar()** del **GameLayer** recorremos la lista de disparos y eliminamos los que estén fuera de la pantalla.

|  |
| --- |
| actualizar (){  ***console***.log(**"disparosJugador: "**+**this**.**disparosJugador**.length);  *// Eliminar disparos fuera de pantalla* **for** (**var** i=0; i < **this**.**disparosJugador**.length; i++){  **if** ( **this**.**disparosJugador**[i] != **null** &&  !**this**.**disparosJugador**[i].estaEnPantalla()){   **this**.**disparosJugador**.splice(i, 1);  i=i-1;  }  } |

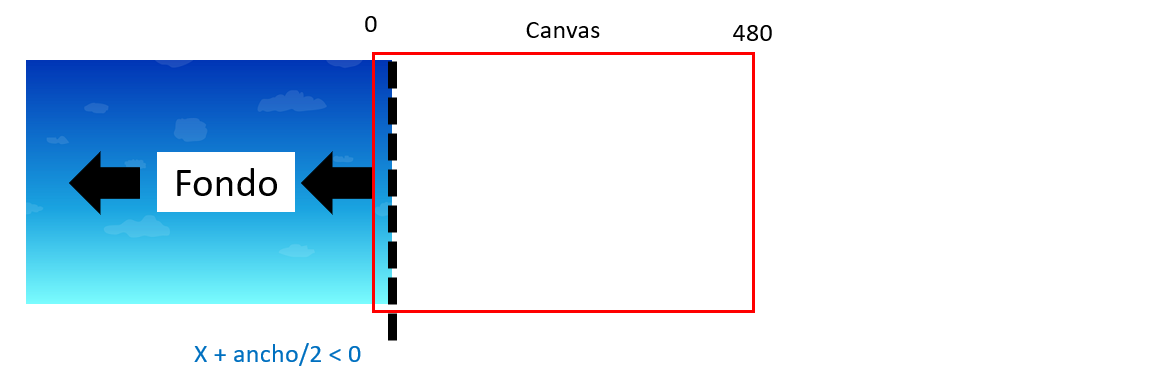
|  |
| --- |
| **Tiempo de vida.** Muchas plataformas utilizan una variable llamada tiempo de vida que se va reduciendo en una unidad en cada iteración del **actualizar(),** en cada iteración se comprueba si el **tiempo de vida == 0,** y en ese caso se elimina.  Con el tiempo de vida estaríamos en este caso controlando la “distancia efectiva” del disparo. |

**\*Fondo Móvil**  
Ahora vamos a modificar la clase **Fondo.js** para que admita la posibilidad de mover el fondo. Los fondos móviles se desplazan hacia un lado de la pantalla y utilizan una imagen auxiliar (igual al fondo original) para completar la pantalla, generan una sensación de movimiento infinito.

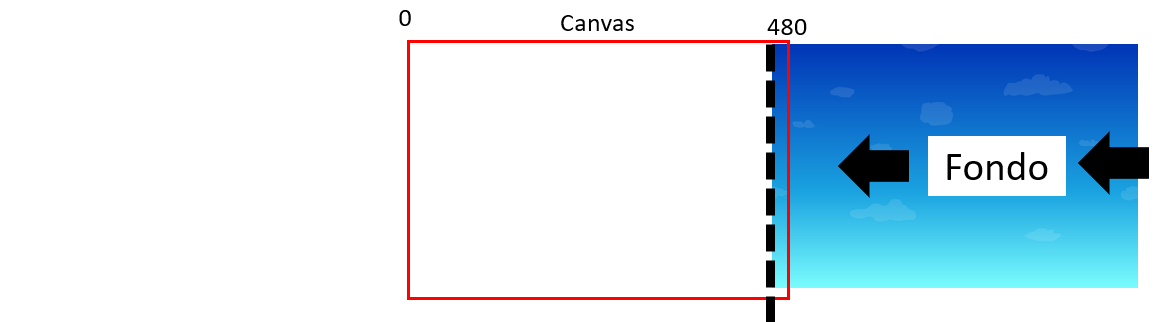
Como el fondo se va a poder mover incluir una velocidad **vx** .

**actualizar(),** si el fondo tiene velocidad **vx != 0** significa que va a ser “movil” necesitamos crear un **fondoAuxilar** (fondoAux). El movimiento es muy sencillo el fondo se mueve hacia la derecha o la izquierda dependiendo de la velocidad vx pero cuando se sale del Canvas debe aparecer por el lado contrario.

Por ejemplo, sí se sale por el lado izquierdo



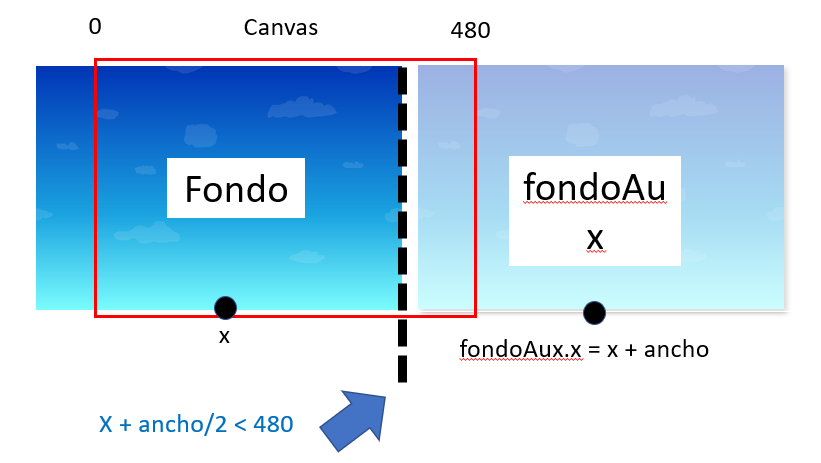
Tiene que reaparecer por la derecha



|  |
| --- |
| **class** Fondo **extends** Modelo {   constructor(rutaImagen, x, y) {  **super**(rutaImagen, x, y)  **this**.**vx** = 0;  }   actualizar(){  **if** ( **this**.**vx** != 0) {  **if** ( **this**.**fondoAux** == **null**){  **this**.**fondoAux** =  **new** Fondo(**this**.**imagen**.**src**, **this**.**x**, **this**.**y**);  }   **this**.**x** = **this**.**x** + **this**.**vx**;   *// margen derecho se sale por la izquierda* **if** (**this**.**x** + **this**.**ancho** / 2 < 0) {  *// vuelve a aparecer por la parte derecha* **this**.**x** = 480 + **this**.**ancho** / 2;  }  *// margen izquierdo se sale por la derecha* **if** (**this**.**x** - **this**.**ancho** / 2 > 480 ) {  *// vuelve a la parte izquierda* **this**.**x** = 0 - **this**.**ancho** / 2;  }  }  }  } |

Pero no basta con que el fondo se mueva, hay que dibujar algo en los “espacios en blanco”, para ello usamos el **fondoAux**, moviéndolo donde el fondo original deja espacios en blanco.

Por ejemplo, si está dejando hueco por la derecha, el lado derecho del fondo es < 480 (derecha de la pantalla)



|  |
| --- |
| dibujar(){  **super**.dibujar();   **if** ( **this**.**fondoAux** != **null** ) {  *// hueco por la izquierda* **if** ( **this**.**x** - **this**.**ancho**/2 > 0){  *// pintar auxiliar por la izquierda* **this**.**fondoAux**.**x** = **this**.**x** - **this**.**ancho**;  }  *// hueco por la derecha* **if** (**this**.**x** + **this**.**ancho**/2 < 480){  *// pintar auxiliar por la derecha* **this**.**fondoAux**.**x** =**this**.**x** + **this**.**ancho**;  }  **this**.**fondoAux**.dibujar();  }  } |

Volvemos al **GameLayer.js,** cuando ejecutamos el método **actualizar()** del el resto de elementos actualizamos también el **fondo**, antes debemos asignarle una velocidad para que se mueva (Aunque en este caso siempre le asignamos velocidad vx=-1 en algunos juegos se varia la velocidad del fondo en función de la velocidad **vx** del **Jugador**)

|  |
| --- |
| actualizar (){  **this**.**fondo**.**vx** = -1;  **this**.**fondo**.actualizar(); |

**Entrenamiento**

**Bomba**

Crear un nuevo tipo de **Modelo** que cuando colisiona con el Jugador elimina a todos los **Enemigos** que haya en pantalla.

En algoritmo de generación de enemigos se deben generar también “bombas” que permitan cambiar a disparo especial cada N iteraciones del juego.

**Enemigos que disparan**

Un posible enfoque para que los **Enemigos** puedan disparar seria:

* Crear una clase **DisparoEnemigo** , puede ser casi igual al disparo del jugador, pero con la velocidad de los disparos invertidos y preferiblemente otra imagen.
* Incluir un array de **disparosEnemigo** en el **GameLayer** (Como hicimos con los disparos del jugador estos también deben ser actualizados y dibujados).
* Implementar una función **disparar()** en el enemigo que retorne un objeto **DisparoEnemigo** en la misma posición en la que se encuentra el enemigo. Llamaremos a la función **disparar()** de los enemigos en cada iteración del GameLayer (método **actualizar()** del GameLayer)
* Para evitar que el enemigo dispare en cada iteración del juego debemos hacer que **disparar()** no siempre devuelva disparo (podría devolver null) Usamos una variable **cadenciaDisparoEnemigo = 50** como “cuenta atrás”, la vamos reduciendo uno a uno y cuando llega a 0 retornamos SÍ que retornamos un disparo y la volvemos a poner a 0.
* Por último, podríamos incluir una condición para llamar al **disparar()** del enemigo[i], por ejemplo que el enemigo este en pantalla. (**enemigo[i]. estaEnPantalla () == true**) Así evitaremos que dispare cuando ni siquiera está en el juego.
* En lugar o adicionalmente al uso de **estaEnPantalla()** podría ser comprobar que el **Jugador** y el **enemigo[i]** están relativamente cerca.

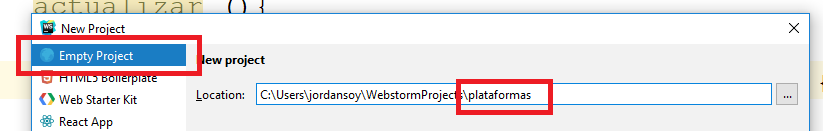
**Enemigos más aleatorios**

Podemos hacer que la velocidad **vx** y la **cadenciaDisparo** (si es que disparan) de cada enemigo fuera ligeramente diferente incluyendo pequeños márgenes de aleatoriedad en sus constructores, para generar un aleatorio en JS utilizamos la siguiente instrucción.

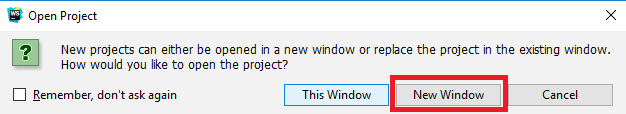
|  |
| --- |
| **var** valor = ***Math***.random() \* (maximo - minimo) + minimo; |

# Recursos - Plataformas

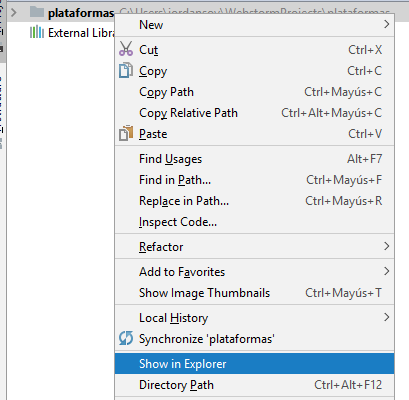
Creamos un nuevo proyecto de tipo **Empty Project**, al cual llamamos **plataformas**.



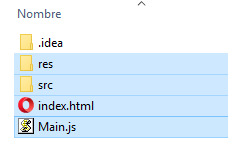
El jetbrains nos permite tener varios proyectos abiertos a (cada uno en una instancia diferente del IDE). Cuando nos pregunte donde queremos abrirlo pulsamos en **“New Window”**



Para ver rápidamente la carpeta en la que está un proyecto o fichero podemos hacer click derecho encima de él y pulsar en **Show in Explorer**



Copiamos todos los ficheros menos **.idea** del juego de naves dentro del **plataformas**



Descargamos del campus virtual la carpeta **2-res.zip** , descomprimes el contenido en la carpeta **/res** del **plataformas**, al menos dos archivos se van a sobrescribir, el **jugador** y el **disparo\_jugador.**

|  |  |
| --- | --- |
| jugador.png | disparo\_jugador.png |

Añadimos también todos los nuevos recursos que vamos a utilizar al **Res.js**

|  |
| --- |
| **var *imagenes*** = {  **jugador** : **"res/jugador.png"**,  **fondo** : **"res/fondo.png"**,  **enemigo** : **"res/enemigo.png"**,  **enemigo\_movimiento** : **"res/enemigo\_movimiento.png"**,  **disparo\_jugador** : **"res/disparo\_jugador2.png"**,  **disparo\_enemigo** : **"res/disparo\_enemigo.png"**,  **icono\_puntos** : **"res/icono\_puntos.png"**,  **icono\_vidas** : **"res/icono\_vidas.png"**,  **icono\_recolectable** : **"res/icono\_recolectable.png"**,  **fondo\_2** : **"res/fondo\_2.png"**,  **jugador\_idle\_derecha** : **"res/jugador\_idle\_derecha.png"**,  **jugador\_idle\_izquierda** : **"res/jugador\_idle\_izquierda.png"**,  **jugador\_corriendo\_derecha** : **"res/jugador\_corriendo\_derecha.png"**,  **jugador\_corriendo\_izquierda** : **"res/jugador\_corriendo\_izquierda.png"**,  **jugador\_disparando\_derecha** : **"res/jugador\_disparando\_derecha.png"**,  **jugador\_disparando\_izquierda** : **"res/jugador\_disparando\_izquierda.png"**,  **jugador\_saltando\_derecha** : **"res/jugador\_saltando\_derecha.png"**,  **jugador\_saltando\_izquierda** : **"res/jugador\_saltando\_izquierda.png"**,  **enemigo\_morir** : **"res/enemigo\_morir.png"**,  **bloque\_tierra** : **"res/bloque\_tierra.png"**,  **bloque\_metal** : **"res/bloque\_metal.png"**,  **bloque\_fondo\_muro** : **"res/bloque\_fondo\_muro.png"**,  **copa** : **"res/copa.png"**,  **pad** :**"res/pad.png"**,  **boton\_disparo** : **"res/boton\_disparo.png"**,  **boton\_salto** : **"res/boton\_salto.png"**,  **boton\_pausa** : **"res/boton\_pausa.png"**,  **menu\_fondo** : **"res/menu\_fondo.png"**,  **boton\_jugar** : **"res/boton\_jugar.png"**,  **mensaje\_como\_jugar** : **"res/mensaje\_como\_jugar.png"**,  **mensaje\_ganar** : **"res/mensaje\_ganar.png"**,  **mensaje\_perder** : **"res/mensaje\_perder.png"**, }; |

# Jugador – Animaciones básicas

**Idle parado**

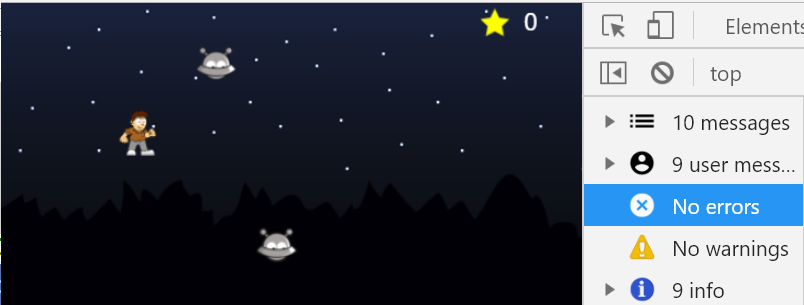
Abrimos la clase **Jugador** y declaramos una nueva animación de forma casi idéntica a como lo hicimos en Enemigo. Primero declaramos las variables **aIdleDerecha** y **animación** (puntero a la animación que queremos ver actualmente)

|  |
| --- |
| constructor(x, y) {  **super**(***imagenes***.**jugador** , x, y)  **this**.**vx** = 0; *// velocidadX* **this**.**vy** = 0; *// velocidadY   // Animaciones* **this**.**aIdleDerecha** = **new** Animacion(***imagenes***.**jugador\_idle\_derecha**,  **this**.**ancho**, **this**.**alto**, 6, 8);  **this**.**animacion** = **this**.**aIdleDerecha**; |

Actualizamos la animación ( **actualizar()** para que cambie de frame). También sobrescribimos el método **dibujar()** del Jugador para que use la nueva animación.

|  |
| --- |
| actualizar(){  **this**.**animacion**.actualizar();  *// Tiempo Disparo* **if** ( **this**.**tiempoDisparo** > 0 ) {  **this**.**tiempoDisparo**--;  }   **this**.**x** = **this**.**x** + **this**.**vx**;  **this**.**y** = **this**.**y** + **this**.**vy**; }   dibujar (){  **this**.**animacion**.dibujar(**this**.**x**, **this**.**y**); } |

Abrimos la aplicación y comprobamos que funciona sin errores.

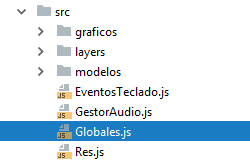


# Estados y orientaciones

Cuando un elemento tiene más de una animación solemos necesitar gestionar estados (saltando, disparando, moviéndose, etc.) y orientaciones (derecha, izquierda)

En este caso los estados se utilizarán sobre todo para determinar que animación ejecutar, (aunque los estados también pueden ser tenidos en cuenta para lógica de negocio / comportamientos).

Para facilitar el uso de los estados y orientaciones vamos a crear dos objetos globales donde declararemos todos los posibles, estados y orientaciones que van a ser contemplados en el juego. Creamos un nuevo fichero en **src/Globales.js**



|  |
| --- |
| **var *estados*** = {}; ***estados***.**moviendo**= 2; *// Incluye parado, derecha , izquierda* ***estados***.**saltando** = 3; ***estados***.**muriendo** = 4; ***estados***.**muerto** = 5; ***estados***.**disparando** = 6; ***estados***.**impactado** = 7;  **var *orientaciones*** = {}; ***orientaciones***.**derecha** = 2; ***orientaciones***.**izquierda** = 3; |

Agregamos el script **Globales.js** al **index.html**, podemos hacerlo en primer lugar.

|  |
| --- |
| <**canvas id="canvas" width="480" height="320"**></**canvas**>  <**script src="src/Globales.js"**></**script**> <**script src="src/GestorAudio.js"**></**script**> |

Vamos a agregar tres animaciones nuevas al **Jugador**: **aIdleIzquierda** , **aCorriendoDerecha** y **aCorriendoIzquierda**.

|  |
| --- |
| **super**(***imagenes***.**jugador** , x, y) **this**.**vx** = 0; *// velocidadX* **this**.**vy** = 0; *// velocidadY  // Animaciones* **this**.**aIdleDerecha** = **new** Animacion(***imagenes***.**jugador\_idle\_derecha**,  **this**.**ancho**, **this**.**alto**, 6, 8); **this**.**aIdleIzquierda** = **new** Animacion(***imagenes***.**jugador\_idle\_izquierda**,  **this**.**ancho**, **this**.**alto**, 6, 8); **this**.**aCorriendoDerecha** =   **new** Animacion(***imagenes***.**jugador\_corriendo\_derecha**,   **this**.**ancho**, **this**.**alto**, 6, 8); **this**.**aCorriendoIzquierda** = **new** Animacion(***imagenes***.**jugador\_corriendo\_izquierda**,   **this**.**ancho**, **this**.**alto**, 6, 8, **null**);  **this**.**animacion** = **this**.**aIdleDerecha**; |

El método **actualizar()** del **Jugador** va a ser el encargado de determinar a qué animación apuntará la variable **animación**

En la primera versión seleccionamos una lógica muy sencilla, en función de la velocidad **this.xv** del **jugador** se seleccionará la animación.

**Únicamente tenemos en cuenta el eje de las X (va a ser un juego con perspectiva frontal, las Y serán para saltos)**

|  |
| --- |
| actualizar(){  **this**.**animacion**.actualizar();   *// Selección de animación* **if** ( **this**.**vx** > 0 ){  **this**.**animacion** = **this**.**aCorriendoDerecha**;  }  **if** ( **this**.**vx** < 0 ){  **this**.**animacion** = **this**.**aCorriendoIzquierda**;  }  **if** ( **this**.**vx** == 0){  **this**.**animacion** = **this**.**aIdleDerecha**;  *// ¿Siempre derecha?* } |

Abrimos el juego vemos que siempre que el jugador se para (**this.vx == 0)** lo hace orientado hacia la derecha.

|  |  |
| --- | --- |
|  |  |

La animación de **Idle (parado)** requiere saber la **orientación** anterior, en el futuro puede haber otras animaciones que seguramente necesitarán esta orientación como disparar o saltar.

Creamos una variable **orientación** en el **Jugador**.

|  |
| --- |
| constructor(x, y) {  **super**(***imagenes***.**jugador** , x, y)  **this**.**orientacion** = ***orientaciones***.**derecha**; |

En la función **actualizar()** cambiamos en esa **orientación** en función de la velocidad **this.vx.** Primero determinamos la **orientación**, luego la **animación**.

**Borrar toda la selección de animación anterior.**

|  |
| --- |
| actualizar(){  **this**.**animacion**.actualizar();   *// Establecer orientación* **if** ( **this**.**vx** > 0 ){  **this**.**orientacion** = ***orientaciones***.**derecha**;  }  **if** ( **this**.**vx** < 0 ){  **this**.**orientacion** = ***orientaciones***.**izquierda**;  }   *// Selección de animación* **~~if~~** ~~(~~ **~~this~~**~~.~~**~~vx~~** ~~> 0 ){~~ **~~this~~**~~.~~**~~animacion~~** ~~=~~ **~~this~~**~~.~~**~~aCorriendoDerecha~~**~~;  }~~ **~~if~~** ~~(~~ **~~this~~**~~.~~**~~vx~~** ~~< 0 ){~~ **~~this~~**~~.~~**~~animacion~~** ~~=~~ **~~this~~**~~.~~**~~aCorriendoIzquierda~~**~~;  }~~ **~~if~~** ~~(~~ **~~this~~**~~.~~**~~vx~~** ~~== 0){~~ **~~this~~**~~.~~**~~animacion~~** ~~=~~ **~~this~~**~~.~~**~~aIdleDerecha~~**~~;~~ *~~// ¿Siempre derecha?~~* ~~}~~   **if** ( **this**.**vx** != 0 ) {  **if** (**this**.**orientacion** == ***orientaciones***.**derecha**) {  **this**.**animacion** = **this**.**aCorriendoDerecha**;  }  **if** (**this**.**orientacion** == ***orientaciones***.**izquierda**) {  **this**.**animacion** = **this**.**aCorriendoIzquierda**;  }  }  **if** ( **this**.**vx** == 0){  **if** (**this**.**orientacion** == ***orientaciones***.**derecha**) {  **this**.**animacion** = **this**.**aIdleDerecha**;  }  **if** (**this**.**orientacion** == ***orientaciones***.**izquierda**) {  **this**.**animacion** = **this**.**aIdleIzquierda**;  }  } |

Comprobamos que el juego funciona de forma correcta y sin errores.



**\*EXTRA: Cambiar fondo del juego.** Accedemos constructor del **GamLayer.js** indicando que la imagen del fondo es **imagen.fondo\_2**

|  |
| --- |
| iniciar() {  *reproducirMusica*();  **this**.**fondoPuntos** =  **new** Fondo(***imagenes***.**icono\_puntos**, 480\*0.85,320\*0.05);  **this**.**puntos** = **new** Texto(0,480\*0.9,320\*0.07 );  **this**.**jugador** = **new** Jugador(50, 50);  **this**.**fondo** = **new** Fondo(***imagenes***.**fondo\_2**,480\*0.5,320\*0.5); |



# Animaciones finitas

Hasta el momento todas las animaciones utilizadas eran infinitas se ejecutaban en forma de bucle.

Muchos juegos utilizan animaciones finitas, el código tiene que controlar cuando se llega al último frame de la animación para quitarlas. La animación de **disparar** del **Jugador** será finita.



En primer lugar, vamos a modificar **Animacion.js** para que permita animaciones finitas. Basta con añadir un parámetro adicional **callback** en el constructor, será una referencia/puntero a la función que queremos ejecutar cuando la animación finita acabe.

Nos enteramos que la animación ha acabado porque se invoca la función **callback**.

|  |
| --- |
| constructor(imagenSrc, modeloAncho, modeloAlto, velocidadRefresco, framesTotales, callback) {  *// Nuevo para animaciones finitas* **this**.**callback** = callback;   **this**.**imagen** = **new *Image***();  **this**.**imagen**.src = imagenSrc; |

En **actualizar()** de **Animación.js** comprobamos si estamos en el último frame, si estamos en el último frame y tenemos un **callback** != **null** lo ejecutamos (llamamos a la función)

|  |
| --- |
| actualizar (){  **this**.**ultimaActualizacion**++;   **if** (**this**.**ultimaActualizacion** > **this**.**velocidadRefresco**) {  **this**.**ultimaActualizacion** = 0;  *// actualizar el frame* **this**.**frameActual**++;  *// Si llega al último frame evuelve al primero* **if** (**this**.**frameActual** >= **this**.**framesTotales**) { *~~// reiniciar, es infinita~~* **~~this~~**~~.~~**~~frameActual~~** ~~= 0;~~  **if** ( **this**.**callback** != **null**){  *// avisar de que acabo* **this**.**frameActual** = 0;  **this**.**callback**();  } **else** {  *// reiniciar, es infinita* **this**.**frameActual** = 0;  }  }  }  *// actualizar el rectangulo (siguiente frame)* **this**.**rectanguloDibujo**.**x** = **this**.**frameActual** \* **this**.**frameAncho**; } |

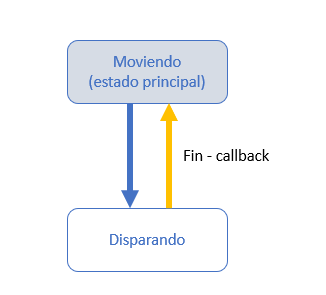
Para crear animaciones finitas bastará con agregar un parámetro adicional **callback** al constructor de **Animación** . (Sí al crear una instancia de **Animación** no le enviamos este último parámetro **callback** es lo mismo que enviar un null).

# Jugador – Animación de disparar y Orientación de disparo

Para manejar múltiples animaciones que dependen de acciones vamos a necesitar manejar **Estados** (Ej: disparar, curarse, recargar, etc) . Vamos a crear una variable **estado** en el **Jugador**.

|  |
| --- |
| **class** Jugador **extends** Modelo {   constructor(x, y) {  **super**(***imagenes***.**jugador** , x, y)  **this**.**estado** = ***estados***.**moviendo**; |

* El **estado** cambiará a **estados.disparando** cuando se crea un disparo,
* Volverá al estado anterior **estado = estados.moviendo** cuando termine la animación de disparar.



|  |
| --- |
| disparar(){  **if** ( **this**.**tiempoDisparo** == 0) {  *// reiniciar Cadencia* **this**.**estado** = ***estados***.**disparando**;  **this**.**tiempoDisparo** = **this**.**cadenciaDisparo**;  *reproducirEfecto*(***efectos***.**disparo**);  **return new** DisparoJugador(**this**.**x**, **this**.**y**);  } **else** {  **return null**;  } }  finAnimacionDisparar(){  **this**.**estado** = ***estados***.**moviendo**; } |

Declaramos las dos animaciones de disparo en el **constructor** del jugador, al terminar estas animaciones finitas se debe llamar a **finAnimacionDisparar().**

|  |
| --- |
| *// Animaciones* **this**.**aDispararDerecha** = **new** Animacion(***imagenes***.**jugador\_disparando\_derecha**,  **this**.**ancho**, **this**.**alto**,6,4,**this**.finAnimacionDisparar.bind(**this**) ); *// No pasar funciones del DIRECTAMNTE COMO callback // El objeto que ejecute la función no sabrá interpretar el "this."* **this**.**aDispararIzquierda** = **new** Animacion(***imagenes***.**jugador\_disparando\_izquierda**,  **this**.**ancho**,**this**.**alto**,6,4,**this**.finAnimacionDisparar.bind(**this**));  **this**.**aIdleDerecha** = **new** Animacion(***imagenes***.**jugador\_idle\_derecha**,  **this**.**ancho**, **this**.**alto**, 6, 8); |

|  |
| --- |
| **IMPORTANTE, Javascript y bind(this).** NO podemos pasar como parámetro solo el nombre del método, ya que el método está dentro de un objeto y usa referencias **this.variableDelObjeto** dentro de su implementación  Si este método se ejecuta en otro contexto **this** no referenciará al objeto **Jugador**. **bind(this)** asigna el operador **this** de forma correcta cuando entrega el método. |

El **Estado** del **Jugador** va a variar de forma correcta entre **moviendo** y **disparando**. Solo queda hacer que el método **actualizar()** asigne la **this.animación** en función del estado.

Una opción podría ser eliminar lo anterior y usar un **switch** por Estados.

|  |
| --- |
| *// Selección de animación*  **~~if~~** ~~(~~ **~~this~~**~~.~~**~~vx~~** ~~!= 0 ) {~~ **~~if~~** ~~(~~**~~this~~**~~.~~**~~orientacion~~** ~~==~~ ***~~orientaciones~~***~~.~~**~~derecha~~**~~) {~~ **~~this~~**~~.~~**~~animacion~~** ~~=~~ **~~this~~**~~.~~**~~aCorriendoDerecha~~**~~;  }~~ **~~if~~** ~~(~~**~~this~~**~~.~~**~~orientacion~~** ~~==~~ ***~~orientaciones~~***~~.~~**~~izquierda~~**~~) {~~ **~~this~~**~~.~~**~~animacion~~** ~~=~~ **~~this~~**~~.~~**~~aCorriendoIzquierda~~**~~;  } }~~ **~~if~~** ~~(~~ **~~this~~**~~.~~**~~vx~~** ~~== 0){~~ **~~if~~** ~~(~~**~~this~~**~~.~~**~~orientacion~~** ~~==~~ ***~~orientaciones~~***~~.~~**~~derecha~~**~~) {~~ **~~this~~**~~.~~**~~animacion~~** ~~=~~ **~~this~~**~~.~~**~~aIdleDerecha~~**~~;  }~~ **~~if~~** ~~(~~**~~this~~**~~.~~**~~orientacion~~** ~~==~~ ***~~orientaciones~~***~~.~~**~~izquierda~~**~~) {~~ **~~this~~**~~.~~**~~animacion~~** ~~=~~ **~~this~~**~~.~~**~~aIdleIzquierda~~**~~;  } }~~**switch** (**this**.**estado**){  **case *estados***.**disparando**:  **if** (**this**.**orientacion** == ***orientaciones***.**derecha**) {  **this**.**animacion** = **this**.**aDispararDerecha**;  }  **if** (**this**.**orientacion** == ***orientaciones***.**izquierda**) {  **this**.**animacion** = **this**.**aDispararIzquierda**;  }  **break**;  **case *estados***.**moviendo**:  **if** ( **this**.**vx** != 0 ) {  **if** (**this**.**orientacion** == ***orientaciones***.**derecha**) {  **this**.**animacion** = **this**.**aCorriendoDerecha**;  }  **if** (**this**.**orientacion** == ***orientaciones***.**izquierda**) {  **this**.**animacion** = **this**.**aCorriendoIzquierda**;  }  }  **if** ( **this**.**vx** == 0){  **if** (**this**.**orientacion** == ***orientaciones***.**derecha**) {  **this**.**animacion** = **this**.**aIdleDerecha**;  }  **if** (**this**.**orientacion** == ***orientaciones***.**izquierda**) {  **this**.**animacion** = **this**.**aIdleIzquierda**;  }  }  **break**; } |

Probamos la aplicación, nos aseguramos de que las animaciones funcionen y que no haya errores de ejecución.

|  |  |
| --- | --- |
|  |  |

El **jugador** siempre dispara hacia la derecha. Podemos usar la propiedad **this.orientacion** para determinar que el DisparoJugador que se crea en **disparar()** se mueva hacia un lado u otro.

|  |
| --- |
| disparar(){  **if** ( **this**.**tiempoDisparo** == 0) {  *// reiniciar Cadencia* **this**.**estado** = ***estados***.**disparando**;  **this**.**tiempoDisparo** = **this**.**cadenciaDisparo**;    *reproducirEfecto*(***efectos***.**disparo**);  **~~return new~~** ~~DisparoJugador(~~**~~this~~**~~.~~**~~x~~**~~,~~ **~~this~~**~~.~~**~~y~~**~~);~~  **var** disparo = **new** DisparoJugador(**this**.**x**, **this**.**y**);  **if** ( **this**.**orientacion** == ***orientaciones***.**izquierda** ){  disparo.**vx** = disparo.**vx**\*-1; *//invertir* }  **return** disparo;  } **else** {  **return null**;  } } |

# Enemigo – Animación y estados parar morir

Vamos a agregar una **animación finita** más, la cual es relativa a un estado ligeramente diferente al resto de estados que nos podemos encontrar en un juego, **“morir”**. Para ello necesitamos dos estados en lugar de uno, que pueden ser:

* **muriendo** : el enemigo ya no participa en el juego pero no podemos sacarlo ya que se debe seguir dibujando y animando hasta que termine su animación de muriendo.
* **muerto**: el enemigo ha acabado de morir y ya se puede sacar del juego.

Agregamos la variable **estado** al **Enemigo** (igual que hicimos con el Jugador ya que va a manejar diferentes animaciones).

Declaramos la animación de muerte **aMorir** (en el enemigo no consideramos **orientaciones**, hemos usado gráficos que siempre miran al frente).

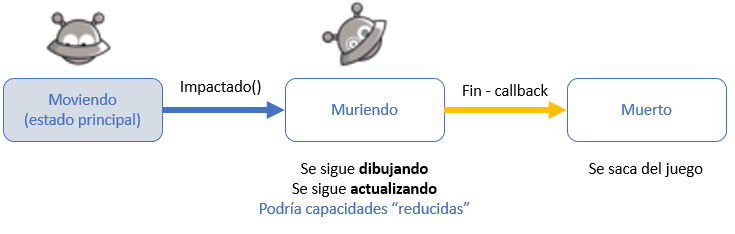
|  |
| --- |
| constructor(x, y) {  **super**(***imagenes***.**enemigo**, x, y)  **this**.**estado** = ***estados***.**moviendo**;   **this**.**aMover** = **new** Animacion(***imagenes***.**enemigo\_movimiento**,  **this**.**ancho**, **this**.**alto**, 6, 3);   **this**.**aMorir** = **new** Animacion(***imagenes***.**enemigo\_morir**,  **this**.**ancho**,**this**.**alto**,6,8, **this**.finAnimacionMorir.bind(**this**));  *// Ref a la animación actual* **this**.**animacion** = **this**.**aMover**;   **this**.**vy** = 0;  **this**.**vx** = 1; }  finAnimacionMorir(){  **this**.**estado** = ***estados***.**muerto**; } |

Modificamos su función **actualizar()** para que seleccione la animación en función del estado. También podría estar bien que no se moviese mientras esta muriendo (o sí depende del juego).

|  |
| --- |
| actualizar (){  *// Actualizar animación* **this**.**animacion**.actualizar();    **switch** (**this**.**estado**){  **case *estados***.**moviendo**:  **this**.**animacion** = **this**.**aMover**;  **break**;  **case *estados***.**muriendo**:  **this**.**animacion** = **this**.**aMorir**;  **break**;  }   **if** ( **this**.**estado** != ***estados***.**muriendo**) {  **this**.**vx** = -1;  **this**.**x** = **this**.**x** + **this**.**vx**;  } } |

Creamos la función **impactado()** que directamente va a poner al enemigo en el **estado =** **muriendo**. (Si el enemigo tuviera varias vidas, habría que ir reduciéndosela antes de matarlo).

|  |
| --- |
| impactado(){  **if** ( **this**.**estado** != ***estados***.**muriendo** ){  **this**.**estado** = ***estados***.**muriendo**;  } } |



Ahora solo falta modificar el **GameLayer.js** para que los **enemigos** no se eliminen directamente del array **enemigos**.

Nos dijimos al método **actualizar()** que es el que controla la lógica de las colisiones.

Sí un **DisparoJugador** colisiona con un **enemigo** ejecutamos el método **impactado()** del enemigo ( le pondrá estado **muriendo** pero no lo sacará aun del juego ).

Al final de la función **actualizar()** sacamos del juego a todos los **enemigos** que tengan el **estado.muerto** (esos ya han terminado de morir)

|  |
| --- |
| *// colisiones , disparoJugador - Enemigo* **for** (**var** i=0; i < **this**.**disparosJugador**.length; i++){  **for** (**var** j=0; j < **this**.**enemigos**.length; j++){  **if** (**this**.**disparosJugador**[i] != **null** &&  **this**.**enemigos**[j] != **null** &&  **this**.**disparosJugador**[i].colisiona(**this**.**enemigos**[j])) {   **this**.**disparosJugador**.splice(i, 1);  i = i-1;  **~~this~~**~~.~~**~~enemigos~~**~~.splice(j, 1);  j = j-1;~~  **this**.**enemigos**[j].**impactado**();  **this**.**puntos**.**valor**++;  }  } }  *// Enemigos muertos fuera del juego* **for** (**var** j=0; j < **this**.**enemigos**.length; j++){  **if** ( **this**.**enemigos**[j] != **null** &&  **this**.**enemigos**[j].**estado** == ***estados***.**muerto** ) {    **this**.**enemigos**.splice(j, 1);  j = j-1;  } } |

