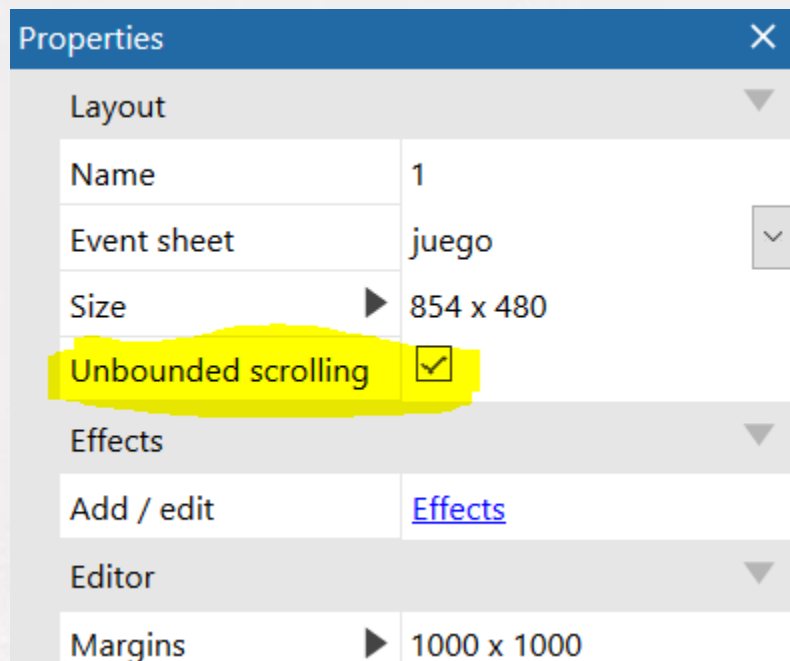



# Puliendo el juego



[Introducción al “Game Feel”](#)



Añadimos un nuevo objeto “sprite” a nuestro proyecto para que nos ayude a controlar la cámara del juego. Es importante que hagamos click sobre el “layout” (no encima de un objeto) y que en el inspector activemos la opción de “Unbounded scrolling”.

System	Every tick	Add action	Add...
player	Is mover_derecha	camera	Set position to $(\text{lerp}(\text{camera.X}, \text{player.X} + \text{distancia\_camara}, \text{camara\_velocidad}), \text{player.Y})$
		Add action	Add...
player	 Is mover_derecha	camera	Set position to $(\text{lerp}(\text{camera.X}, \text{player.X} - \text{distancia\_camara}, \text{camara\_velocidad}), \text{player.Y})$
		Add action	Add...

 Global number **distancia\_camara** = 128

 Global number **camara\_velocidad** = 0.2

Creamos un nuevo evento que se ejecuta constantemente (“on every tick”) para controlar la posición de la cámara.

Además, haciendo click derecho sobre la “event sheet”, hemos añadido las variables globales **distancia\_camara** y **camara\_velocidad**, que nos permiten cambiar valores más rápido.

`ceil(x)` Round up x e.g. `ceil(5.1) = 6`

`cosp(a, b, x)` Cosine interpolation of *a* to *b* by *x*. Calculates  $(a + b + (a - b) * \cos(x * 180^\circ)) / 2$ .

`cubic(a, b, c, d, x)` Cubic interpolation through *a*, *b*, *c* and *d* by *x*. Calculates `lerp(qarp(a, b, c, x), qarp(b, c, d, x), x)`.

`distance(x1, y1, x2, y2)` Calculate distance between two points

`exp(x)` Calculate  $e^x$

`floor(x)` Round down x e.g. `floor(5.9) = 5`

`infinity` A floating point number value representing infinity.

**`lerp(a, b, x)` Linear interpolation of *a* to *b* by *x*. Calculates  $a + x * (b - a)$ .**

`unlerp(a, b, y)` Reverse linear interpolation: if `lerp(a, b, x) = y`, then `unlerp(a, b, y) = x`. Calculates  $(y - a) / (b - a)$ .

`ln(x)` Log to base *e* of *x*.

`log10(x)` Log to base 10 of *x*.

`max(a, b [, c...])`, `min(a, b [, c...])` Calculate maximum or minimum of the given numbers. Any number of parameters can be used as long as there are at least two.

`pi` The mathematical constant  $\pi$  (3.14159...)

`qarp(a, b, c, x)` Quadratic interpolation through *a*, *b* and *c* by *x*. Calculates `lerp(lerp(a, b, x), lerp(b, c, x), x)`.

`round(x)` Round *x* to the nearest whole number e.g. `round(5.6) = 6`

Home

 Getting started >

Get Construct 3

Using an account

Construct 3 on mobile

System requirements

 Overview >

 Interface >

 Project primitives >

 Tips & guides >

 Behavior reference >

 Plugin reference >




 System reference >

Scripting >

La función `lerp`, así como todas las acciones, expresiones y condiciones disponibles en Construct, se pueden encontrar en el [manual online](#). Leerlo por encima para hacernos una idea de todas las características del motor es muy recomendable.

▼ **Disparar**

▼ →  Keyboard On **Space** pressed

 System	Create object  <b>bullet</b> on layer " <b>imagen_suelo</b> " at ( <i>player_img.X</i> , <i>player_img.Y</i> + 16)
 camera	Shake  ScrollTo with magnitude 15 for 0.4 seconds (Reducing magnitude)

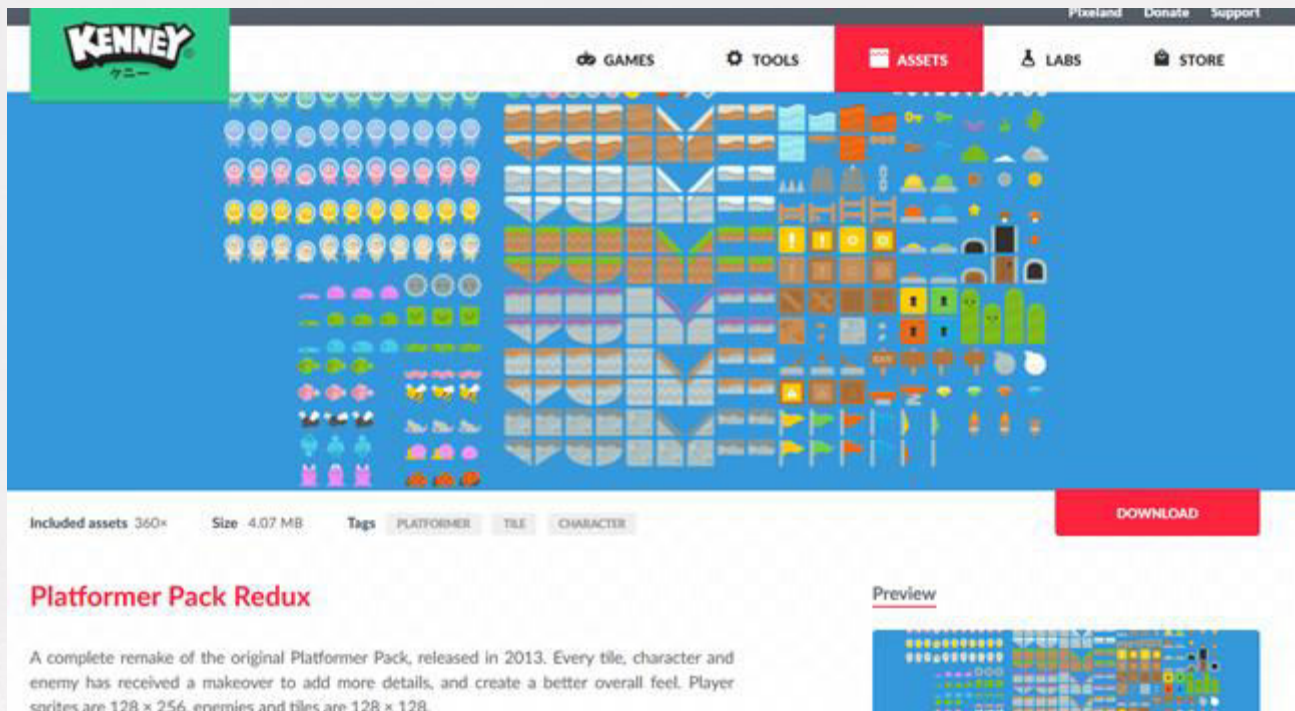
Add action Add...

También podemos usar este nuevo “sprite” que hemos creado para agitar ligeramente la cámara cada vez que disparamos.

### ▼ Disparar

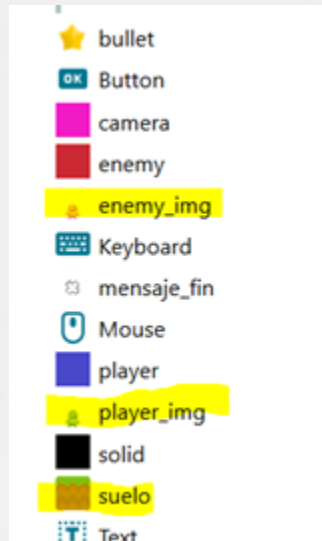
▼ → Keyboard	On <b>Space</b> pressed	System	Create object 🌟 <b>bullet</b> on layer "imagen_suelo" at ( <i>player_img.X</i> , <i>player_img.Y + 16</i> )
		camera	Shake 🌀 ScrollTo with magnitude 15 for 0.4 seconds (Reducing magnitude)
		Add action	Add...
player	Is <b>mover_derecha</b>	bullet	Set 🏹 Bullet angle of motion to $-10 + \text{random}(15)$ degrees
		Add action	Add...
player	✗ Is <b>mover_derecha</b>	bullet	Set 🏹 Bullet angle of motion to $170 + \text{random}(15)$ degrees
		Add action	Add...

Con una pequeña modificación sobre el ángulo de dirección de las balas podemos añadir un pequeño factor de aleatoriedad que le dará más variedad.

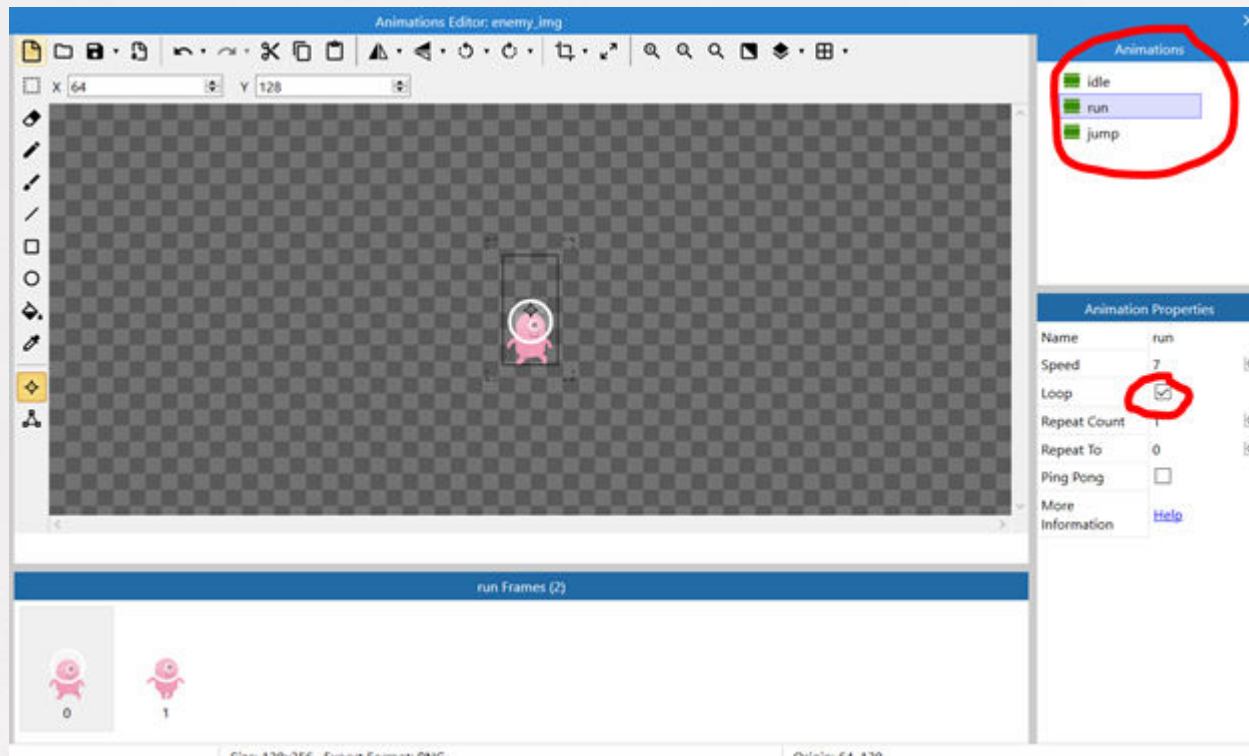


Durante el curso vamos a usar uno de [los packs de imágenes gratuitos de Kenney](#). Van bien para prototipar, pero no se deberían usar en un juego que vamos a lanzar ya que lo haría muy genérico.

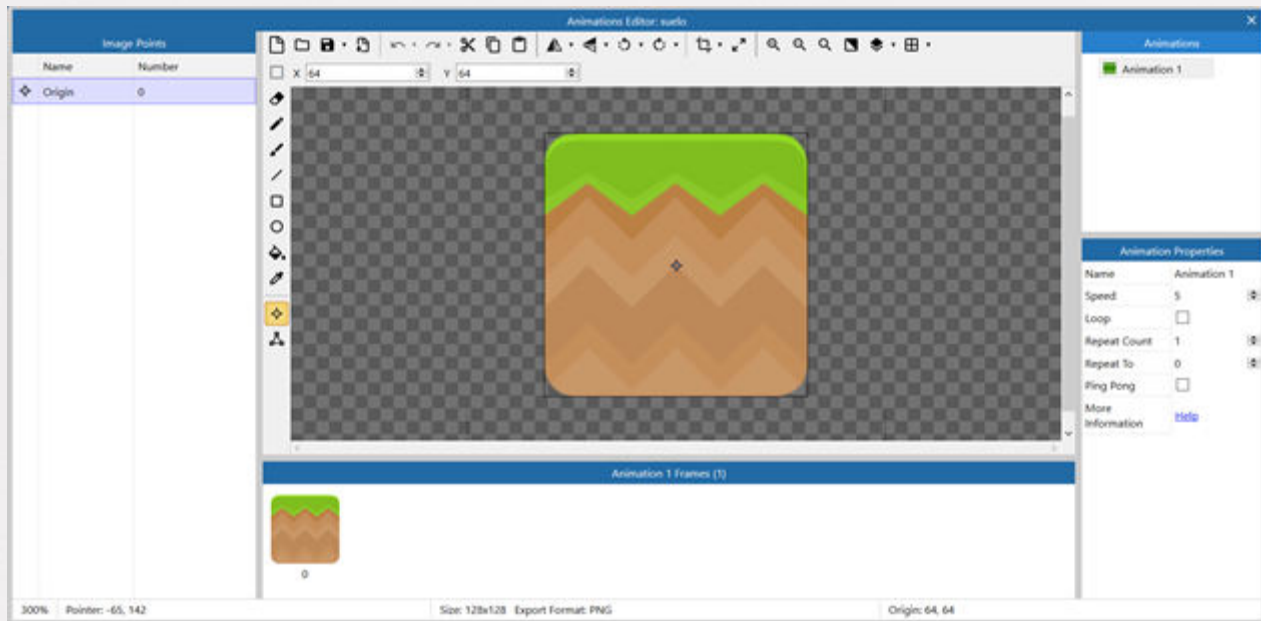




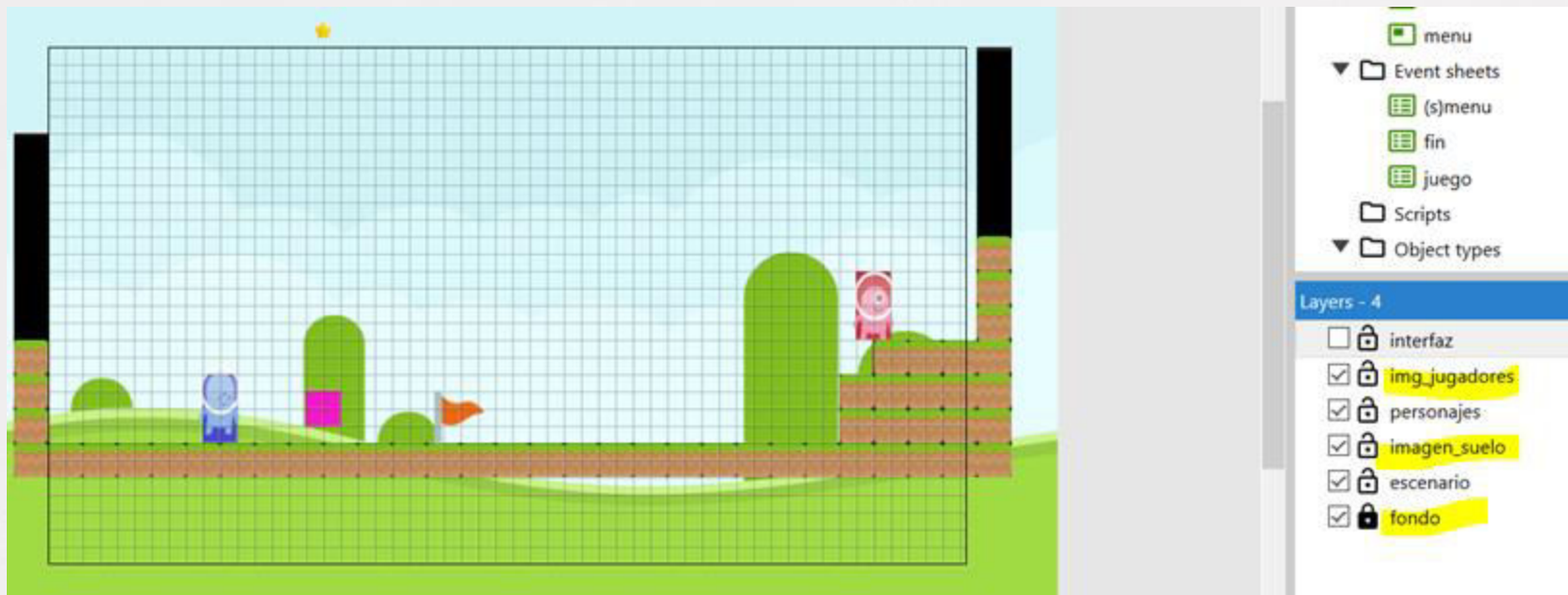
A continuación creamos tres nuevos sprites para colocar sobre nuestros “player”, “enemy” y “solid”. Podríamos modificar directamente los “sprites” anteriores, pero es algo que es desaconsejable ya que puede dar problemas de muchos tipos a largo plazo.



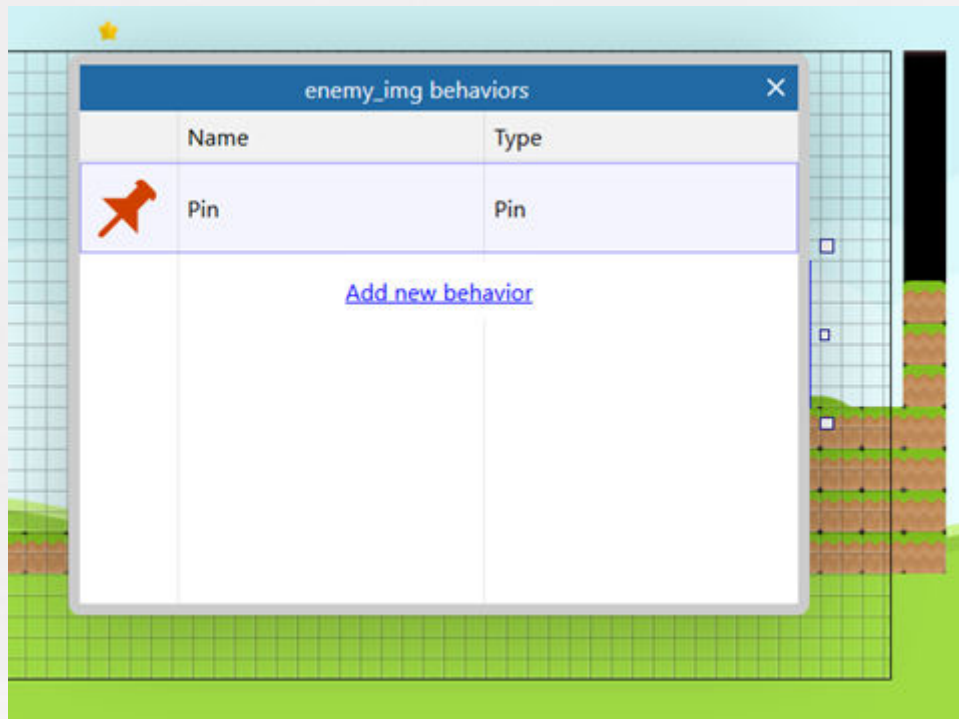
Hacemos doble click sobre cada imagen para abrir el editor. Después vamos haciendo click derecho en “Animations” para añadir nuevas animaciones. En el panel de abajo podemos añadir varios “frames” o imágenes a cada animación. Es importante marcar la casilla de “loop” cuando queremos que una animación se repita (como es el caso de correr).



Para el suelo, y para avanzar rápido, vamos a usar una sola imagen. El pack de Kenney tiene varias baldosas (o "tiles") disponibles que se pueden usar para crear escenarios más consistentes.



Para que sea más fácil organizar todo, se recomienda crear nuevas capas donde colocar todos los “assets”. Para los objetos que están en movimiento (como “player” o “enemy”) nos falta añadir un poco de lógica si queremos que la imagen que hemos añadido se vaya moviendo por pantalla con el objeto.







Primero añadimos a estos nuevos sprites el “behavior pin”, que nos permitirá pegarlos a otro objeto. Queda activar esta lógica e indicar a qué objeto queremos que se peguen.











Creamos un nuevo evento que se ejecuta al cargarse el “layout”. Aquí fijamos la imagen del protagonista al objeto “player”. Repetimos el proceso para “enemy”, pero aquí hemos añadido un “for each enemy\_img” (por si tenemos varias instancias) y le indicamos a construct que debería coger el “enemy” más cercano a cada “enemy\_img” para fijarlo.

Hay muchas formas de conseguir este resultado. Otra sería indicándole a Construct que cree una instancia de de “enemy\_img” sobre cada “enemy” y que lo fije a él (en lugar de colocarlos nosotros manualmente).

player	 Platform is moving	player_i... Set animation to <b>"run"</b> (play from beginning)
		Add action
player	 Platform is moving	player_i... Set animation to <b>"idle"</b> (play from beginning)
		Add action
player	 Platform is on floor	
player	 Platform is on floor	player_i... Set animation to <b>"jump"</b> (play from beginning)
		Add action

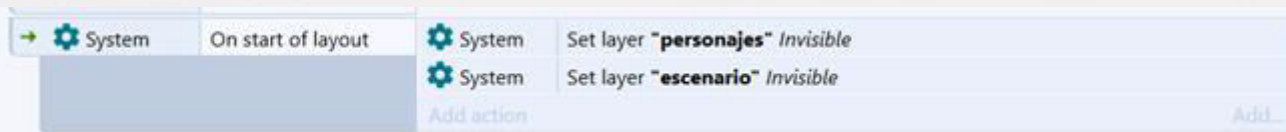
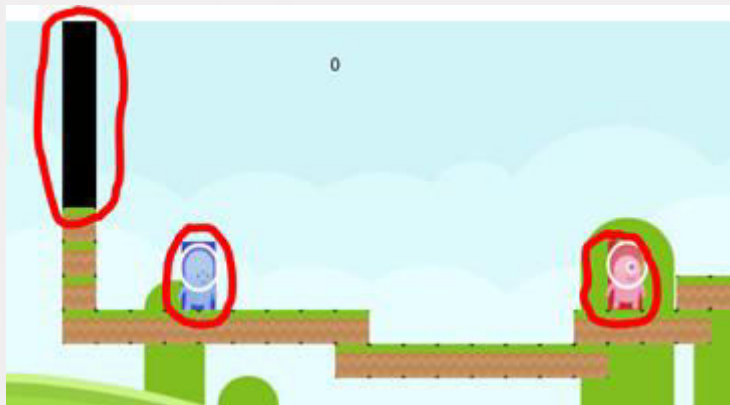
Añadimos unos eventos adicionales para indicar al objeto "player\_img" qué animación debe mostrar en función de lo que esté haciendo el objeto "player" (al que controla el jugador con las teclas W,A y D).



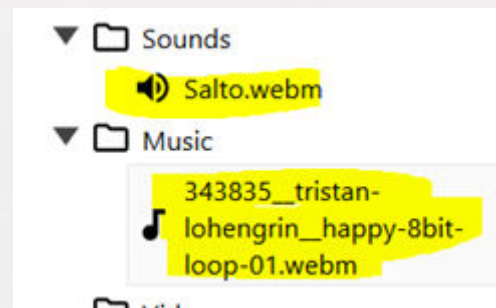
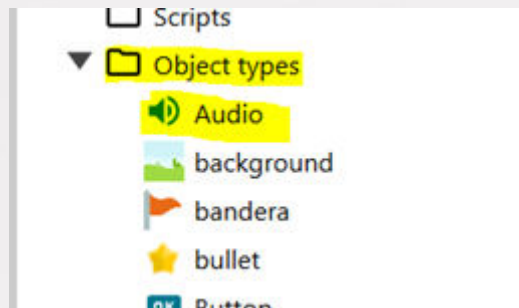
 System  enemy  enemy	For each  <b>enemy</b>	 enemy_i... Set animation to <b>"run"</b> (play from beginning)
	 Platform is moving	Add action <span>Add...</span>
	Is <b>perseguir</b>	
 System  enemy  enemy	For each  <b>enemy</b>	 enemy_i... Set animation to <b>"jump"</b> (play from beginning)
	Is <b>perseguir</b>	Add action <span>Add...</span>
	 Platform is on floor	

Creamos unos eventos similares para los enemigos. Con cuidado de añadir un "for each" y de asegurarnos de que la variable perseguir está activada.





Si pulsamos “play” para probar el juego veremos en pantalla los objetos “solid”, “player” y “enemy”. No queremos que estos “assets” se vean, pero es conveniente dejarlos visibles mientras estamos desarrollando el juego ya que nos pueden ayudar a detectar errores. Cuando queramos ocultarlos podemos añadir un evento que haga invisibles las capas donde estén.



Podemos agregar efectos de sonido y pistas de música al proyecto, pero si no añadimos el objeto “audio” no podremos usarlos en las “event sheets” del juego.

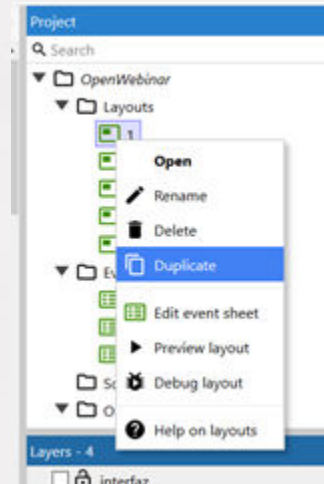
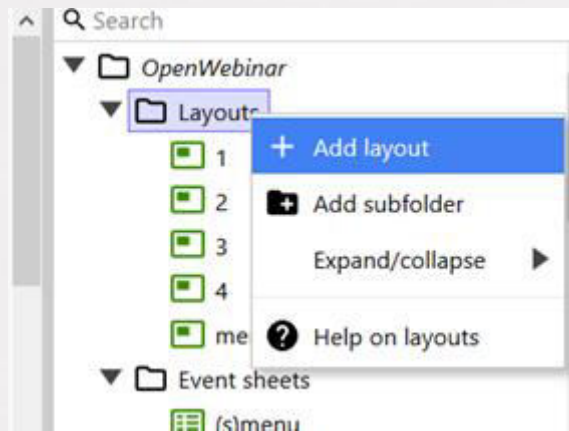
▼ audio

→  System	On start of layout	 Audio	Play <b>343835_tristan-lohengrin_happy-8bit-loop-01</b> looping at volume 0 dB (tag "music")
 Audio	✗ Tag " <b>music</b> " is playing		
		Add action	Add...

→  player	 Platform On jump	 Audio	Play <b>Salto</b> not looping at volume 0 dB (tag "salto")
		Add action	Add...

Con el evento de arriba hacemos que al empezar un "layout" arranque la música siempre y cuando no esté sonando ya.

También podemos usar estas acciones de audio para añadir efectos de sonido a cada acción como en el evento de abajo.



Para añadir nuevos niveles (o “layouts”) al proyecto podemos hacer click derecho en la carpeta de “layouts” y seleccionar “Add layout”.

En este ejemplo es recomendable elegir el “layout” en el que hemos montado la escena del juego y duplicarlo. Así el nuevo “layout” tendrá todas las propiedades que habíamos definido ya.

Global number **distancia\_camara** = 128

Global number **camara\_velocidad** = 0.2

Global number **nivel** = 1

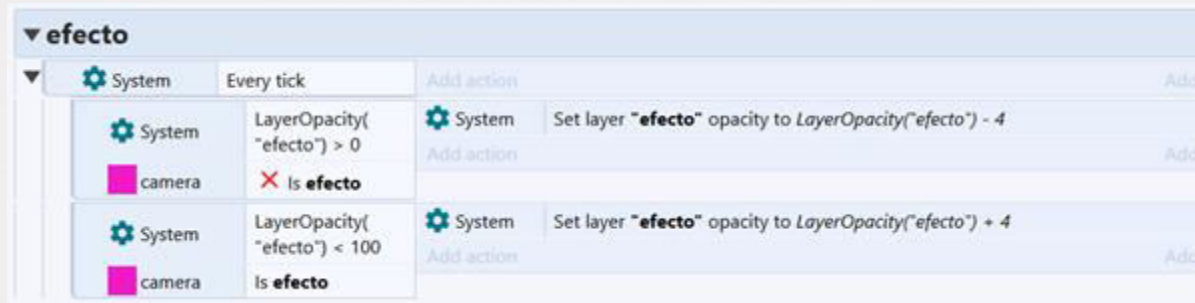
Global number **puntuacion\_total** = 0

Para unir los diferentes niveles que hemos hecho vamos a crear dos nuevas variables globales numéricas: nivel (que tiene 1 como valor inicial) y puntuacion\_total.

En el ejemplo tenemos 3 niveles y un cuarto que actúa como pantalla de fin. La lógica de transición entre niveles del ejemplo funciona porque los nombres de las “layouts” son: 1, 2, 3 y 4. Si se quiere poner otro nombre a los “layouts” habría que cambiar la lógica un poco.




Además, para hacer una transición más fluida entre escenas vamos a añadir un pequeño efectos de oscurecimiento cada vez que termina un nivel. Para ello creamos una nueva capa debajo de interfaz. Debe tener la casilla de “transparent” sin marcar y el fondo negro.



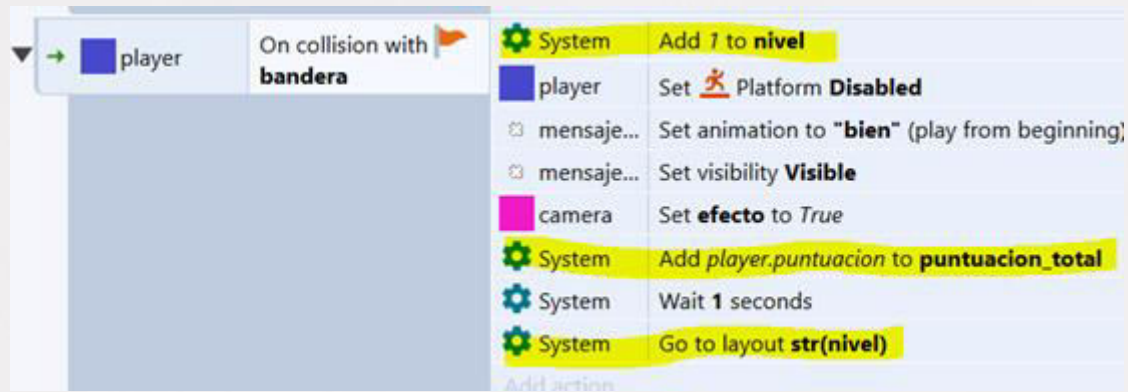
Ahora creamos una variable booleana en el objeto cámara para marcar cuando queremos ocultar o mostrar esta nueva capa. Añadiremos un evento que se ejecuta constantemente ("Every tick") y que cambiará la opacidad de la capa en función de esta variable.

▼ Fin

→ enemy	On collision with <b>player</b>	player	Destroy
		mensaje...	Set visibility <b>Visible</b>
		camera	Set <b>efecto</b> to <b>True</b>
		System	Wait <b>1</b> seconds
		System	Restart layout
		Add action	
▼ → player	On collision with <b>bandera</b>	System	Add <b>1</b> to <b>nivel</b>
		player	Set  Platform <b>Disabled</b>
		mensaje...	Set animation to " <b>bien</b> " (play from beginning)
		mensaje...	Set visibility <b>Visible</b>
		camera	Set <b>efecto</b> to <b>True</b>
		System	Add <i>player.puntuacion</i> to <b>puntuacion_total</b>
		System	Wait <b>1</b> seconds
		System	Go to layout <b>str(nivel)</b>
		Add action	

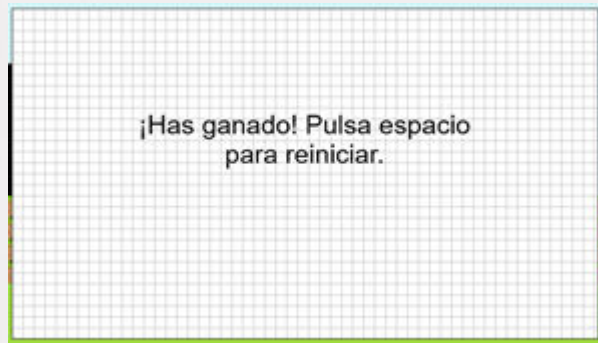
Es importante cambiar el booleano a "true" cuando queramos oscurecer la capa y hacer la transición.





Cada vez que el jugador llega a la bandera le sumamos 1 a la variable nivel y le sumamos player.puntuacion a puntuacion\_total.

Para terminar usamos “Go to layout str(nivel)” para ir al siguiente nivel. Nuestra variable nivel es un número, la acción “Go to layout” carga un nuevo “layout” con el nombre que le indiquemos, pero sólo admite cadenas de texto, por eso usamos la función “str()” para transformar nuestra variable nivel de número a texto.



El último nivel es un “layout” al que le hemos asignado un “event sheet” con sólo dos eventos. Uno que se ejecuta al cargarse y que cambia el texto en pantalla para que ponga cuántos enemigos hemos destruido y un segundo evento, que se activa al pulsar espacio, que reinicia nuestro progreso y nos lleva al menú.