

# Stencyl

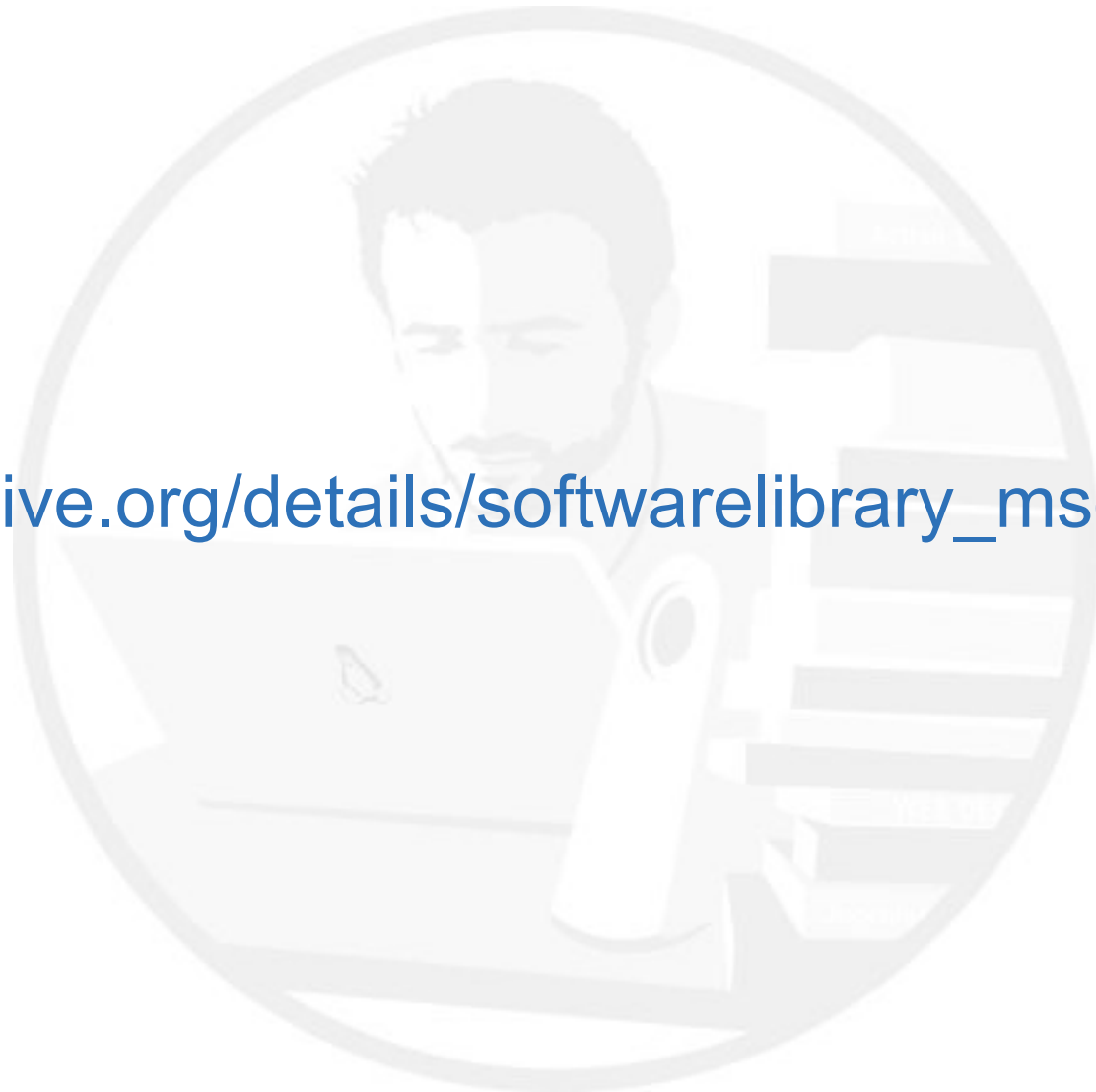
Stencyl es una plataforma de creación de videojuegos 2D. Permite crear videojuegos desde Linux, OS X y Windows para varias plataformas como iPhone, iPad, Android, Flash, Windows, Mac y Linux.

## **Tiene tres versiones:**

- Básica gratuita que permite exportar vía web con Adobe Flash Player.
- Pago básica que exporta también a ordenadores por un valor de \$99 USD.
- Estudio, que permite publicar para iOS y Android por un valor de \$199.

# Juegos MSDOS

[https://archive.org/details/softwarelibrary\\_msdos\\_games](https://archive.org/details/softwarelibrary_msdos_games)



# Instalación

## Descarga

Tener en cuenta que en linux habrá que instalar unas cuantas librerías. El proceso de instalación de dichas librerías se detalla en el link junto al link de descarga para linux.

# Iniciar la aplicación

Cuando lo hacemos nos sale la opción de loguearnos. Esto será necesario si somos usuarios de pago, en caso contrario, no hace falta.

# Interfaz

Arranca el juego



Stencil Forge: permite descargar recursos de internet.

Cuando entremos en un juego y queremos regresar a la interfaz principal, pulsaremos File → close game

Pestañas que me permiten navegar entre lo que tenemos abierto y volver al escritorio.

# Glosario

**Actores**: Los personajes del juego (malos, buenos, etc. ) así como las entidades con las que podemos interactuar.

**Escenas**: normalmente, los niveles del juego; aunque puede haber un nivel con varias escenas.

# Primer juego

Pulsamos en el recuadro que se indica en el escritorio de la interfaz para crear un nuevo juego.

Seleccionamos blank game.

# Crear una escena con un fondo

- 1 – Dashboard → Resources → Scenes → new
- 2 – Dashboard → Resources → Backgrounds → new → le pongo una imagen → Botón verde en la esquina superior derecha (Attach to scene).

No es muy recomendable usar una imagen de fondo para un juego en el que la cámara sigue al personaje, porque a menudo son demasiado pesadas y serán difíciles de mover por nuestra aplicación. En su lugar, será mejor usar tiles, como veremos más adelante.

Si usamos tiles, tener en cuenta que su tamaño debe coincidir con el tamaño de los tiles asociados a la escena que definimos al crearla, u ocurrirán cosas raras.

[Descargar fondo](#)



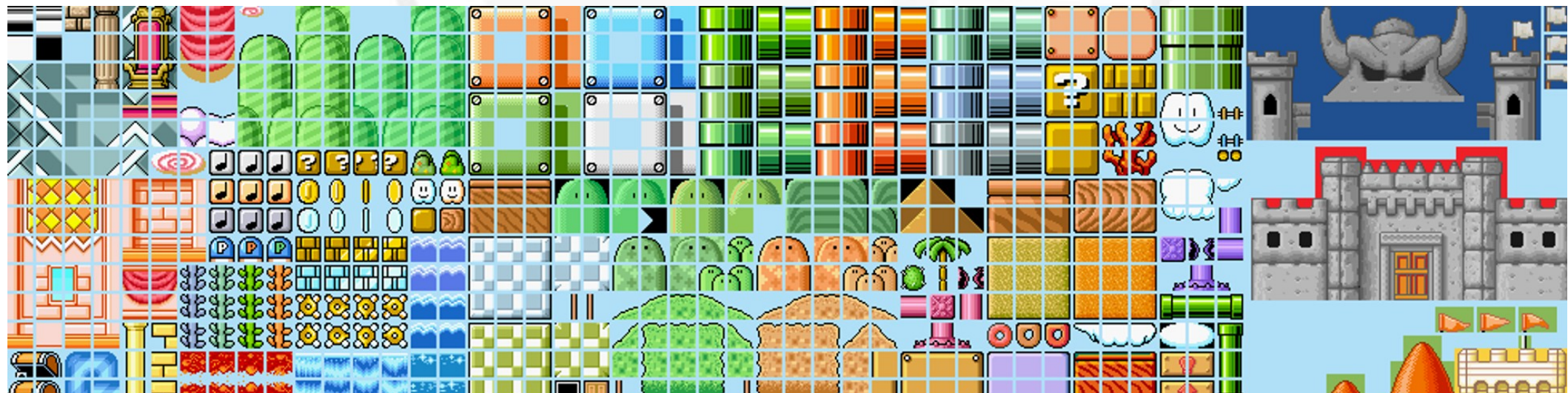
# Tiles

Un tile es una porción gráfica.  
Un tileset es un conjunto de tiles.

Por muchos tiles que tengamos en la librería del videojuego, stencyl sólo utilizará los que necesite.

Resources → tileset → dejamos el tamaño más estándar (32). X Spacing, Y Spacing, da separación entre tiles. X Border, Y Border, coloca un borde alrededor de los tiles.

<http://www.spritters-resource.com>  
<http://opengameart.org>



# Insertar tiles de uno en uno

- 1 – Omito la importación de la imagen.
- 2 – Dibujo la rejilla de tiles.
- 3 – Selecciono la herramienta “Select tiles” en la barra de herramienta de tiles.
- 4 – Voy importando los tiles uno a uno.

Los cuadrados amarillos que aparecen a la derecha son los áreas de colisiones, que puedo ir asignando a cada tile. Por defecto, para cada tile la colisión es de tipo square.

En la parte inferior de la pantalla podemos importar frames para animar los tiles.

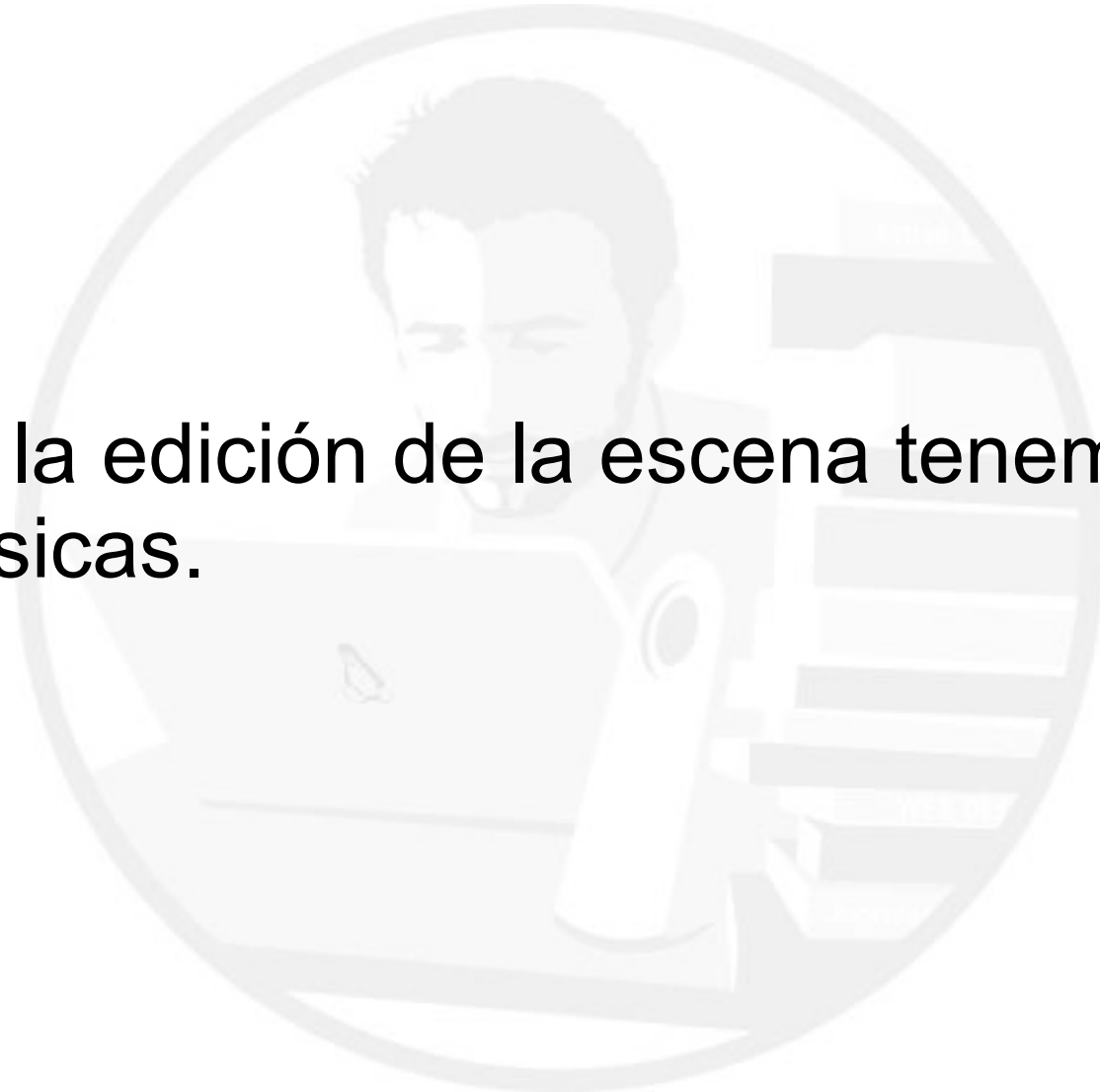
# Ejercicio

Importar correctamente el siguiente fichero de tiles, e ir pintándolos en una nueva escena.

[Descargar tiles](#)

# Físicas

Dentro de la edición de la escena tenemos la pestaña físicas.



# Ejercicio - Actores

1 – Dashboard → Actor types → new

2 – Creamos una animación.

3 – Añadimos las animaciones de andar hacia ambos lados y de estar parado hacia ambos lados.

Deberemos definir el número de columnas y filas que contiene el sprite.

Ahora puedo ir a la escena y tendré disponible el actor para ser insertado.

Puedo recortar el área de colisión de Tux dentro de la pestaña “Collision”. Será necesario en este caso.

[Descargar a tux](#)

# Colisiones

## **Redefinir el área de colisiones.**

Dentro del actor → pestaña colisiones. Desde el menú lateral podemos cambiar el ancho y el alto de este área. También podemos definir un área poligonal. Para borrar, seleccionamos el área de colisiones → supr.

## **Nota:**

Tener cuidado a la hora de definir un área de colisiones para una animación de varios frames. Quizás el área sea apropiado para el primer frame pero no para los siguientes.

# Juego de plataformas



Verlo funcionando

# Definir un comportamiento para andar

Entramos en un proyecto → Dashboard → LOGIC → Actor Behaviors

Podremos crear comportamientos que serán intercambiables entre juegos.

Podemos elegir entre:

- Design Mode (modo gráfico).
- Code Mode (escribiendo código fuente).
- Comportamientos de actor.
- Comportamientos de escena



- 1 - Escogemos Design mode y comportamientos de actor.
- 2 - El nombre del comportamiento será caminar.
- 3 - + Add Event → Basics → When updating
- 4 - Pestaña Palette → Botón Flow → Pestaña conditions → Conditionals → If
- 5 - Botón User Input → Pestaña Keyboard & Mouse → Keyboard → Control is down
- 6 - Botón Actor → Pestaña Motion → Speed → set x-speed → le asignamos el atributo velocidad (valor 20) que debemos haber creado previamente.
- 7 - Botón Actor → Pestaña Draw → Pieza Switch animation to → Pieza as animation → copio y pego el nombre de la animación de tux corriendo
- 8 - Attach Actor type

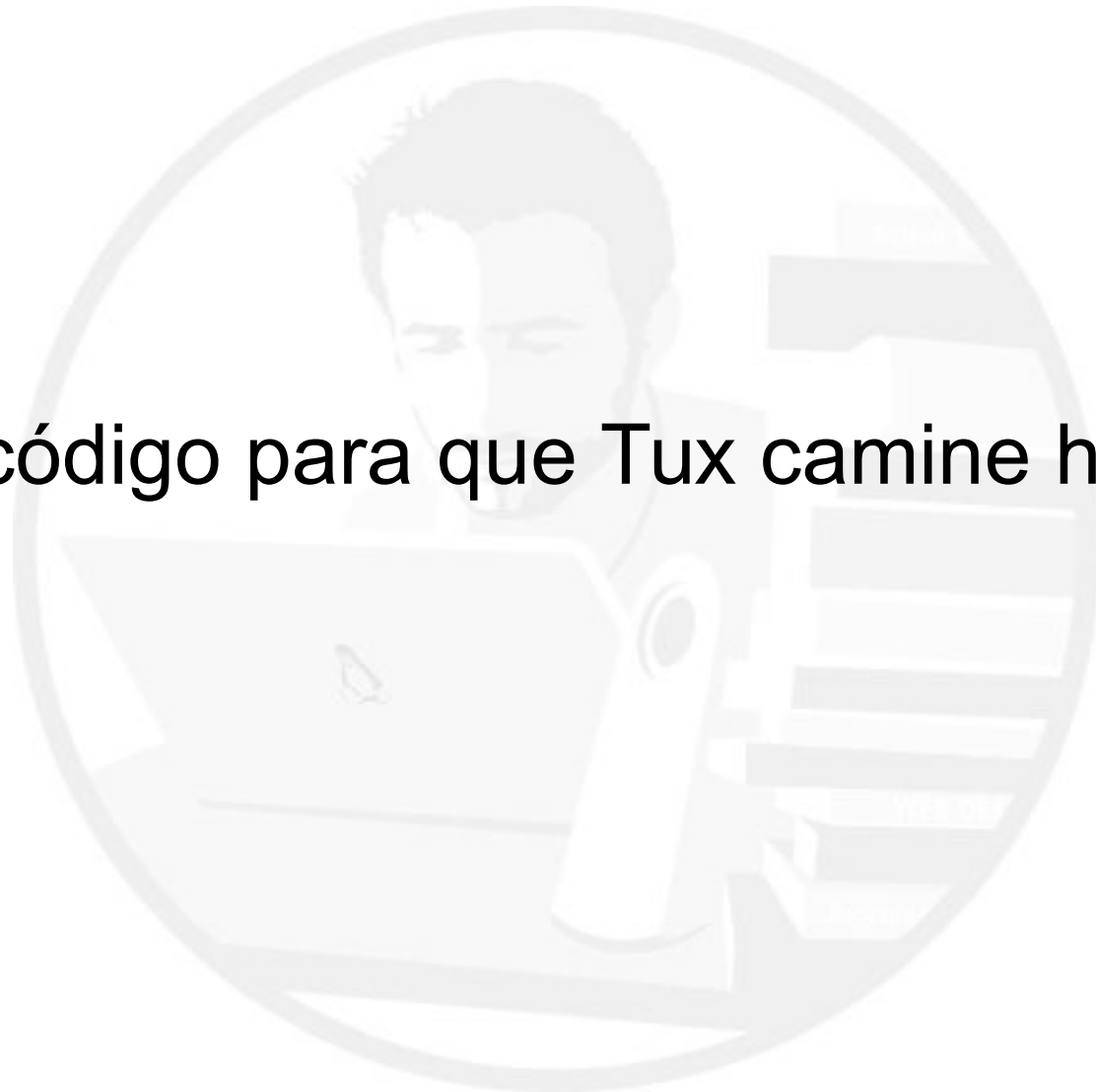


# Ejercicio

Añadir el “flow otherwise” en el que haré que el actor detenga su desplazamiento y se quede con el frame congelado. De momento tux seguirá teniendo la animación de desplazamiento aunque realmente no se mueva.

# Ejercicio

Añadir el código para que Tux camine hacia la izquierda.



# Crear un atributo local

Dentro del actor → pestaña events → pestaña attributes  
→ add attribute

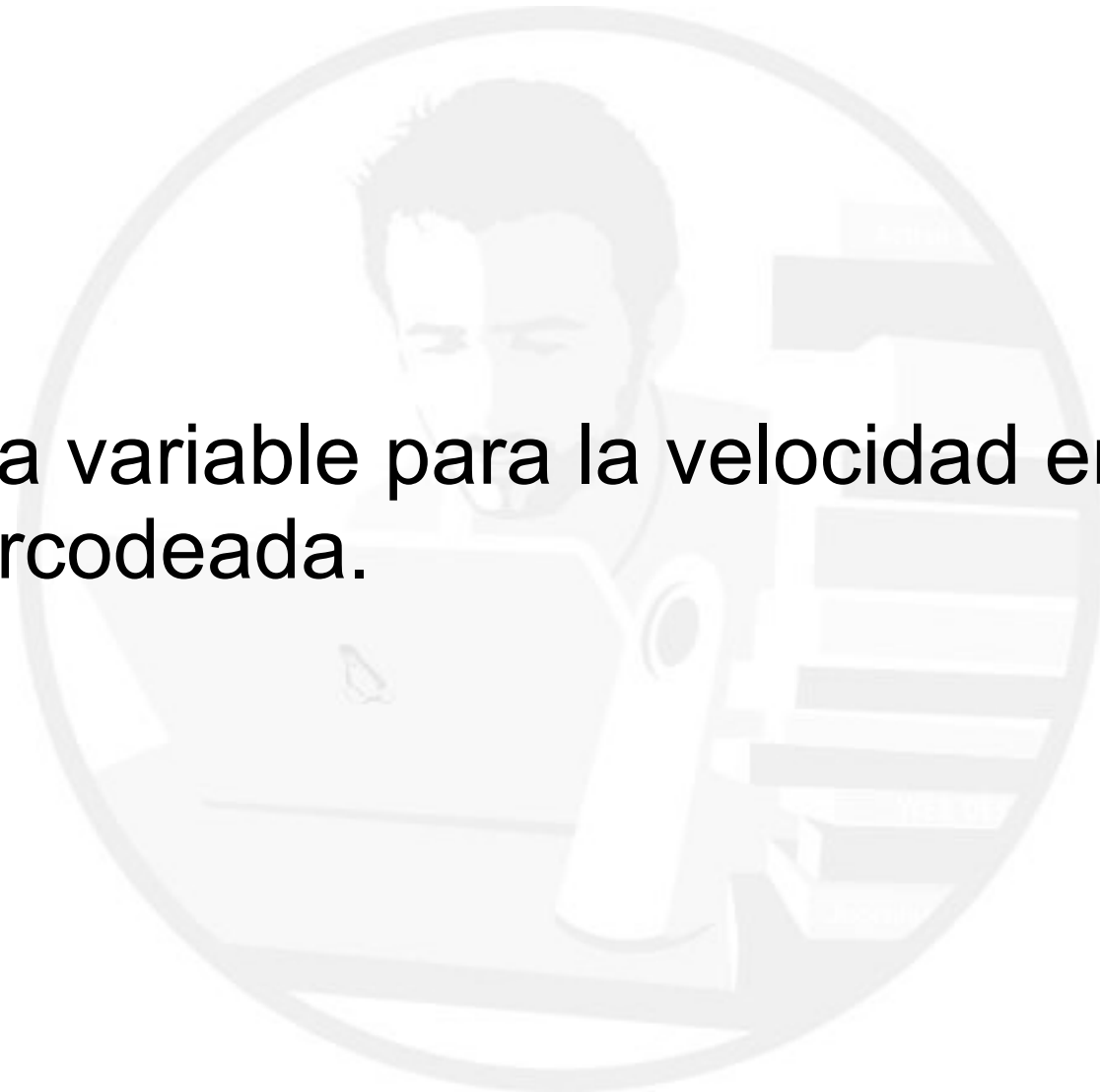
**Tipo actor, actor group, actor type, scene:** vamos a intentar evitarlos porque consumen bastantes recursos.

**Effect:** vamos a intentar evitarlo porque sólo funciona en flash.

Para obtener la velocidad negativa usaremos:  
Numbers & test → negative

# Ejercicio

Utilizar una variable para la velocidad en lugar de tenerla harcodeada.



# Ejercicio

Hacer que Tux mire hacia el lado correcto cuando pare de moverse.

Para ello crearemos un atributo numérico global llamado “orientation” que puede valer 1 o 0.

Lo haremos numérico en lugar de booleano, por sí más adelante queremos el personaje tenga más de dos orientaciones (derecha, izquierda, arriba, abajo, diagonales, etc).

Lo haremos global porque más adelante lo utilizaremos para disparar en la dirección correcta y querremos accederlo.

Lo modificaré (attributes → setters) cuando el actor se esté desplazando y recuperaré (attributes → getters) cuando deje de hacerlo para dejar puesta la animación que me interesa.

# Ejercicio

Crearemos un nuevo comportamiento llamado “saltar”.

Para ello utilizaremos el evento “was pressed”.

# Ejercicio

Haremos que Tux no salga volando para arriba si pulsamos muchas veces el botón de saltar.

Para ello crearemos un atributo numérico que setearemos a 1 cuando saltemos y que volveremos a setear a 0 al cabo de un tiempo utilizando Flow → Time → do after.

Sólo podremos saltar si dicho atributo vale 0.

# Ejercicio - movimiento enemigo

Añadir un enemigo con un comportamiento que hará que se mueva aleatoriamente a cada segundo. Para ello usaremos:

- **When creating**
- Flow → time → do every x seconds
- Numbers & Text → random between 0 and 1

[Descargar enemigo](#)



# Apunte sobre la muerte

Cada actor debe matarse a sí mismo. Esta es la mejor práctica y ofrece el mejor rendimiento.

Además cuando disparamos contra un actor, la lógica del daño que recibe este debe estar escrita en el actor, no en la bala. De lo contrario, el método “do after x seconds” no funcionará, por ejemplo.

# Agregar evento de aplastamiento

- 1 – Add event → Collisions → Actor of type
- 2 – If
- 3 – Collision → the top of actor 1
- 4 – Cambiamos la animación a aplastado

## Notas:

- Para que esto funcione correctamente es importante que en las físicas del prota hayamos establecido que no puede rotar.
- Después de la animación puedo llamar al comando kill para que desaparezca el malo.

# La cámara sigue al actor

- 
- 1 – Dentro de los comportamientos del actor
  - 2 - Scene → View → move camera to center actor

# Atributos globales

Son aquellos accesibles por todas las entidades del juego.

Creación de un atributo global:

Menú settings → Attributes →

Nombre: vidas

Tipo: número

Valor: 4

# Colisiones – Tux pierde una vida

- 1 – En la pestaña evento del enemigo completamos el if con que ya teníamos con: `otherwise (set vida to (vida – 1))`, dónde vida es un atributo global
- 2 – Añadimos un nuevo comportamiento a Tux: `Always(if (vida = 0){kill self})`
- 3 – Qué error nos dá? Cómo solucionarlo?

# Problema: Tux pierde todas las vidas de golpe

Podemos solucionarlo dándole un tiempo de invencibilidad.

Creamos un atributo global llamado invencible que vale 0.

Cuando el prota colisiona con un enemigo lo ponemos a 1 y esperamos un segundo para volver a ponerlo a 0 (tiempo durante el que el prota será invencible).

# Ejercicio

Añadir el enemigo de la tortuga.

Cuando el prota la pise, esta debe quedar dentro de su caparazón.

Crear una variable local para la tortuga que inicialmente valga 0. Cuando la tortuga está dentro del caparazón, la variable valdrá 1. Si la variable vale 1, la tortuga podrá morir.

[Descargar tortuga](#)

# Ejercicio – Tux dispara

Creamos un actor y le vinculamos el sprite de la bala.

Creamos el comportamiento disparar.

```
always(if(pulso action 1){
```

```
scene → create actor type (posición x del actor, posición y del actor) at ...
```

**Importante:** si el objeto creado no se vé en la posición “front”, probaremos con “middle”, y si no, finalmente probaremos con “back”.

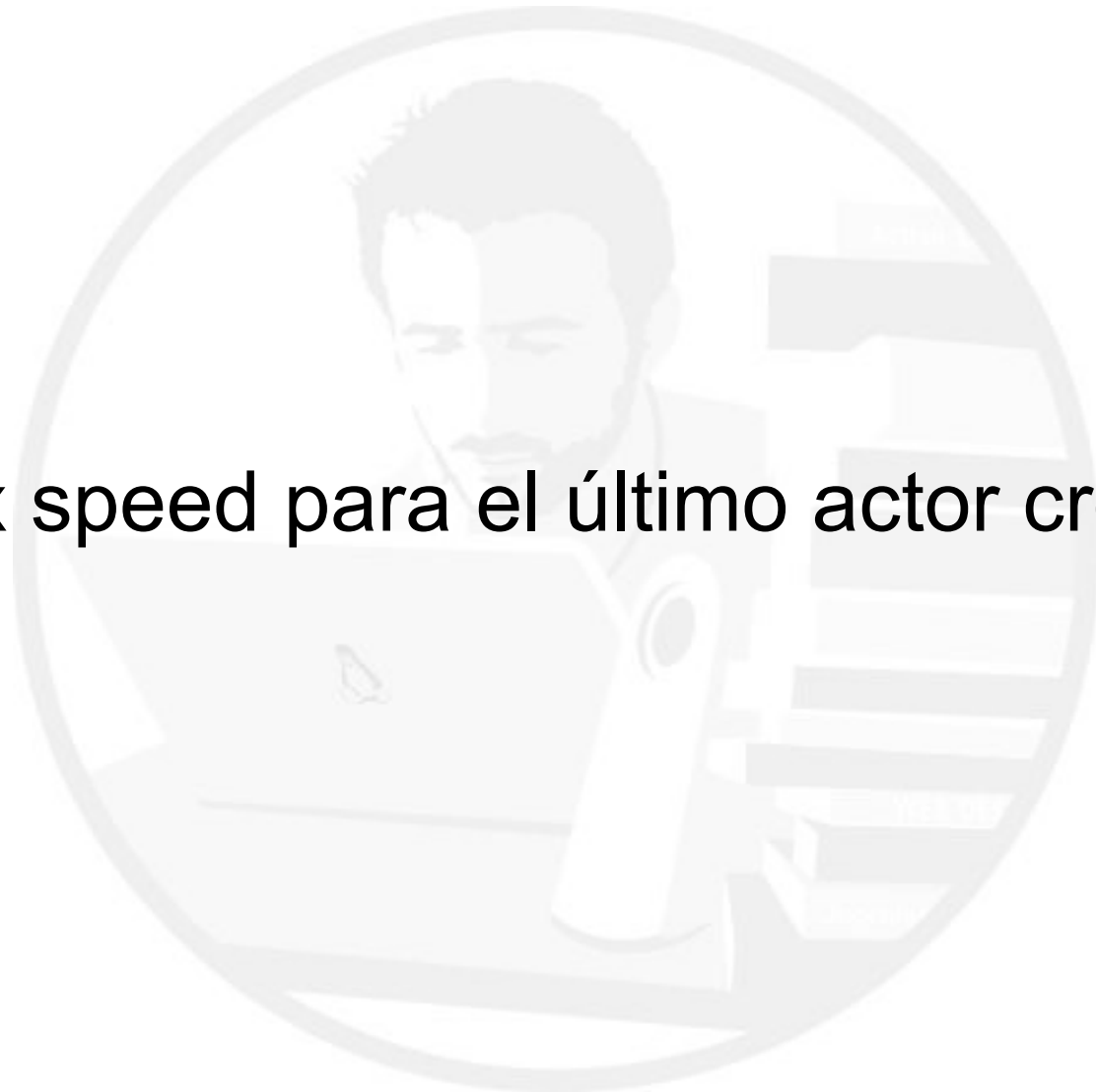
```
})
```

[Descargar bala de cañón](#)



# Asignar movimiento a la bala

set x speed para el último actor creado



# Hacer que el disparo no desplace a nuestro personaje

Para ello modificaremos el comportamiento disparar añadiendo unos pixeles a la posición de creación de la bala en función de la dirección a la que estamos apuntando y del tamaño del personaje.

# Regiones

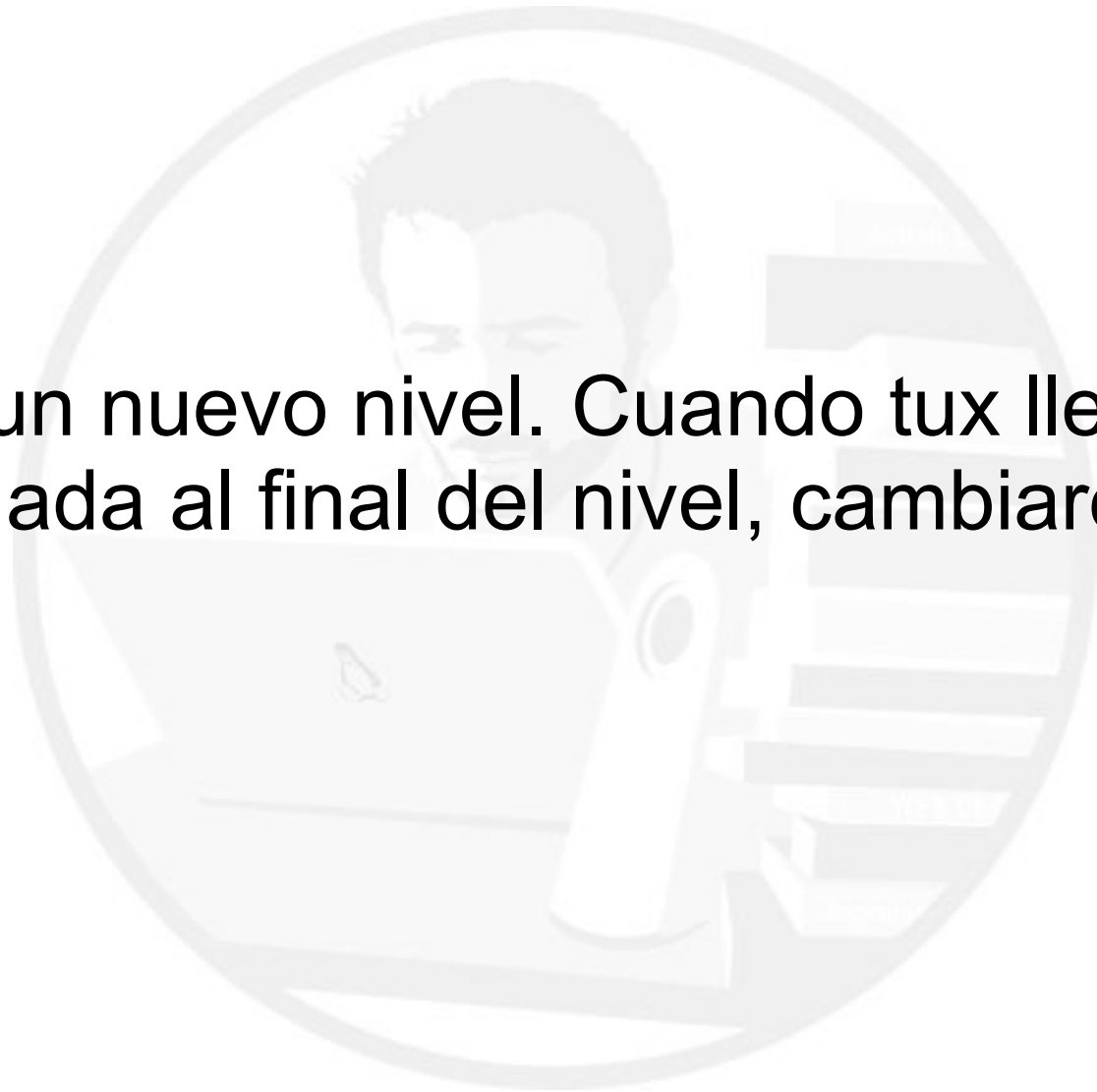
Las podemos utilizar para detonar ciertos eventos. Así, por ejemplo si un personaje pasa por ella podríamos hacer que apareciesen enemigos del techo, etc.

Entramos en un escenario.


- 1 – Pestaña Scene → creamos una región.
- 2 – Pestaña Events → Add events → Actors → Enter or leaves a region → Specific Actor → marcamos la región creada.
- 3 – Esto detonará el evento:  
Scene → Game Flow → Switch to scene (se recarga la escena)

# Cambio de nivel

Creamos un nuevo nivel. Cuando tux llegue a una región situada al final del nivel, cambiaremos al siguiente.



# Añadir nuevos controles de teclado



Settings → Controls

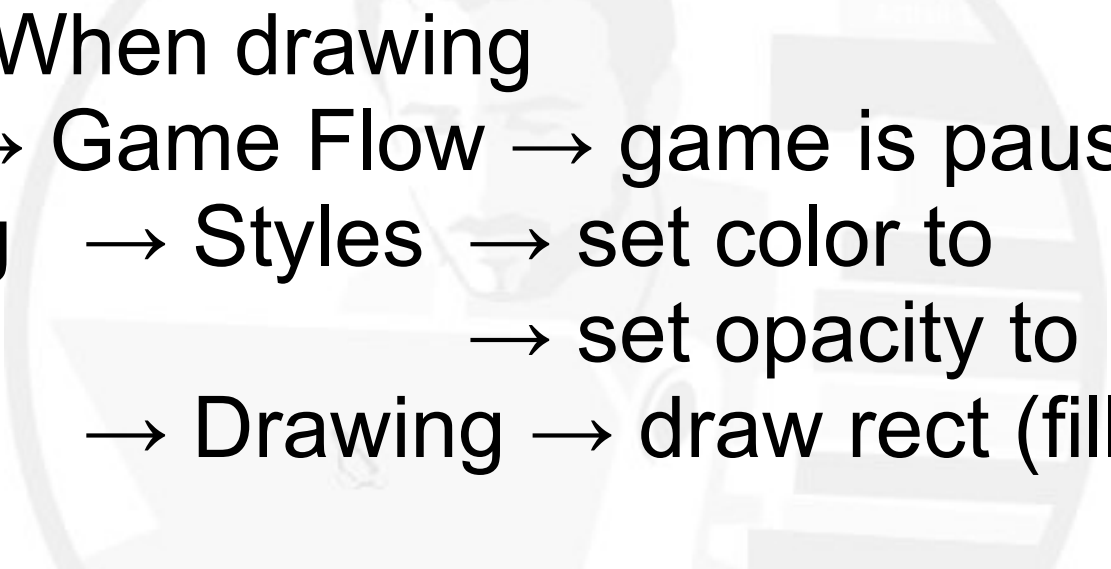
# Pausar el juego

Todos los actores tienen que tener habilitado “Puede ser pausado” dentro de las físicas del actor.

Creamos la pausa como un evento de la escena:

```
always(if(key is pressed){  
    Scene → Game Flow → Pause Game  
})
```

# Oscurecer la escena en la pausa



```
Basics → When drawing
if(Scene → Game Flow → game is paused){
    Drawing → Styles → set color to
                        → set opacity to
    → Drawing → draw rect (fill)
}
```

# Ejercicio

Si vuelvo a pulsar el botón de pause el juego debe reanudarse.

Para ello utilizaré el método Scene → Game Flow → game is paused.



# Música

Utilizaremos única y exclusivamente ficheros mp3 a 44.1 Khz y 16 bits (el mismo formato con otras calidades podría dar problemas).

Para importar, en el editor de bloques:  
Sounds and images → Play sound

## **Ejercicio:**

Poner sonido a la bala en el juego de la vista cenital.

[Descargar audio.](#)

# Sacar monedas

Creamos dos nuevos actores, coin\_box y coin.

Añadimos eventos a la coin\_box:

Add event → Collisions → Actor Type

Cuando el actor colisione con la coin\_box por abajo, debemos crear el nuevo actor moneda.

Evitaremos que la coin\_box sea afectada por la gravedad en físicas → general del actor.

La detección de colisiones es muy sensible. Implementar la lógica necesaria para que una vez que sale una moneda ya no salgan más.

Finalmente hacer que la moneda salga perfectamente vertical hacia arriba y desaparezca al cabo de un segundo.

# Textos – crear un marcador de monedas

Creamos una fuente: Dashboard → Fonts → Create new font

En el escenario:

When Drawing → Drawing → set current font to  
→ Drawing → draw text (x2 veces para concatenar)

# Convertir mario en un run and jump

En una nueva capa oculta colocamos un punto que avanza a velocidad constante y que será el que sigue la cámara.

Daremos velocidad a ese punto en el when created.

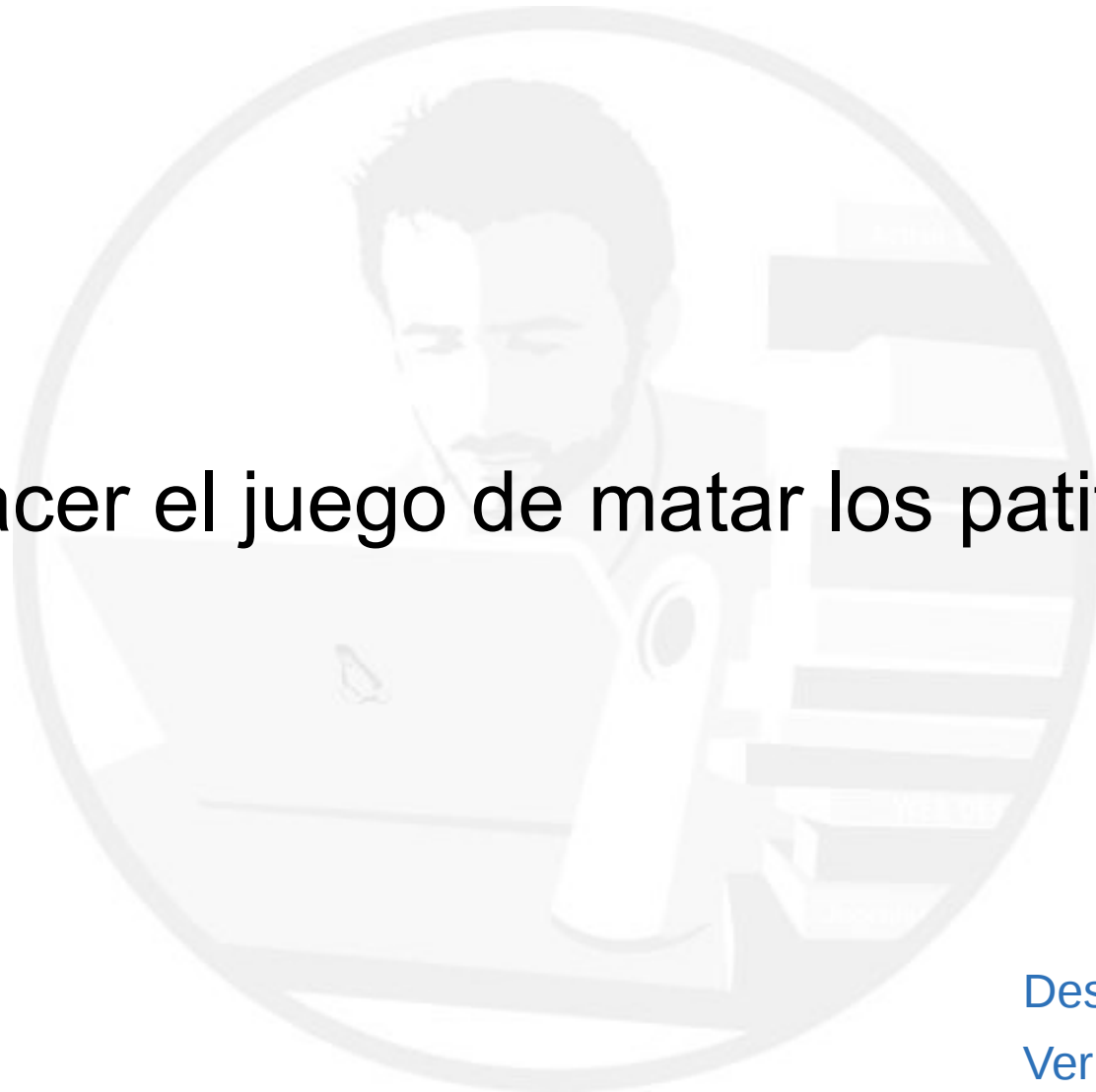
# Ejercicio - cenital

- 1 - Crear un juego nuevo.
- 2 - Crear una escena.
- 3 - Rellenarla con los siguientes tiles (vista cenital).
- 4 - Crear un actor.
- 5 – En su pestaña físicas, especificaremos deshabilitar colisiones continuas.
- 6 - Crearle 4 animaciones a partir de estos recursos.
- 7 – Le creamos el comportamiento de caminar.
- 8 – El protagonista puede disparar.

[Descargar prota](#)  
[Verlo funcionando](#)

# Ejercicio

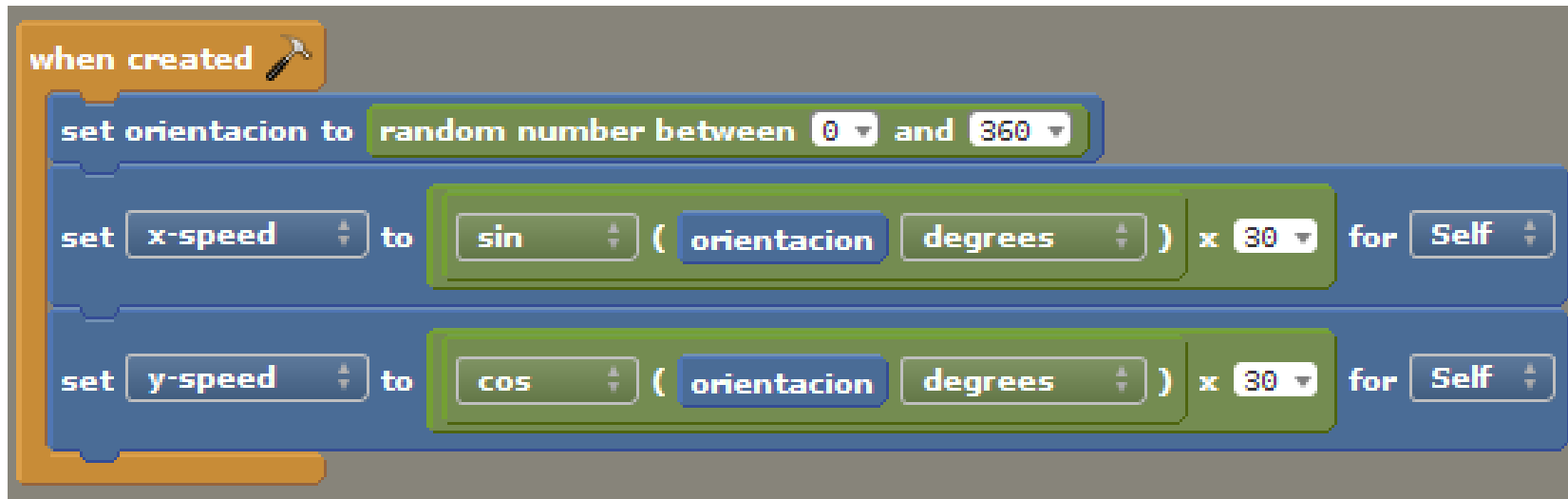
Hacer el juego de matar los patitos



[Descargar recursos](#)

[Verlo funcionando](#)

# Ejercicio - Pong



Definiremos un único comportamiento para ambos ladrillos. Para ello definiremos dos globales globales de tipo action, que configuraremos con las teclas correctos para cada ladrillo.

Para que la pelota rebote con los límites de la pantalla, podemos colocar unos tiles, y en la sección físicas → material de la pelota definiremos friction=0, bounciness=1.

Para que la pelota rebote contra el ladrillo puedo setearle physics → General → normal. Sin embargo, esto hará que los ladrillos se superpongan sobre los tiles. Para evitarlo, dar a los ladrillos un peso exagerado hará que colisiones con los tiles y que la pelota no los pueda desplazar. También es aconsejable indicar que la pelota tenga su peso mínimo, es decir 0.01 Kg.

[Descargar ladrillo](#)

[Verlo funcionando](#)

# Ejercicio - Pistoleros

Verlo funcionando

Descargar foto pistolero





# Ejercicio – ovejas borrachas



Descargar ovejas

Verlo funcionando

# Ejercicio - darts

A partir del código del método Updated, completar el código para el método Drawing, para completar correctamente el juego de la diana.



[Verlo funcionando](#)

[Descargar diana](#)

# Aviso Legal

Los derechos de propiedad intelectual sobre el presente documento son titularidad de D. Pablo Monteserín Fernández Administrador, propietario y responsable de pablomonteserin.com. El ejercicio exclusivo de los derechos de reproducción, distribución, comunicación pública y transformación pertenecen a la citada persona.

Queda totalmente prohibida la reproducción total o parcial de las presentes diapositivas fuera del ámbito privado (impresora doméstica, uso individual, sin ánimo de lucro).

La ley que ampara los derechos de autor establece que: “La introducción de una obra en una base de datos o en una página web accesible a través de Internet constituye un acto de comunicación pública y precisa la autorización expresa del autor”.

El contenido de esta obra está protegido por la Ley, que establece penas de prisión y/o multa, además de las correspondientes indemnizaciones por daños y perjuicios, para quienes reprodujesen, plagiaran, distribuyeren o comunicaren públicamente, en todo o en parte, o su transformación, interpretación o ejecución fijada en cualquier tipo de soporte o comunicada a través de cualquier medio.

El usuario que acceda a este documento no puede copiar, modificar, distribuir, transmitir, reproducir, publicar, ceder, vender los elementos anteriormente mencionados o un extracto de los mismos o crear nuevos productos o servicios derivados de la información que contiene.

Cualquier reproducción, transmisión, adaptación, traducción, modificación, comunicación al público, o cualquier otra explotación de todo o parte del contenido de este documento, efectuada de cualquier forma o por cualquier medio, electrónico, mecánico u otro, están estrictamente prohibidos salvo autorización previa por escrito de Pablo Monteserín.

El autor de la presente obra podría autorizar a que se reproduzcan sus contenidos en otro sitio web u otro soporte (libro, revista, e-book, etc.) siempre y cuando se produzcan dos condiciones:

1. Se solicite previamente por escrito mediante email al correo pablomonteserin@pablomonteserin.com.
2. En caso de aceptación, no se modifiquen los textos y se cite la fuente con absoluta claridad.

Una parte de las imágenes utilizadas en este documento no son propiedad de Pablo Monteserín, por lo que, si alguna de estas imágenes estuviera sujeta a derechos de autor, o a algún otro tipo de derecho que impida su publicación en este documento, una vez que el autor, Pablo Monteserín, tenga conocimiento del hecho, procederá a la retirada inmediata de la imagen protegida por los derechos pertinentes.