Guía de iniciación al diseño y desarrollo de videojuegos en

SCRATCH

GENERALITAT VALENCIANA
Conselleria de Participación, Transparencia, Cooperación y Calidad Democrática

MUSOL
Municipalistas por la Solidaridad y el Fortalecimiento Institucional

Esta publicación ha sido realizada con el apoyo financiero de la Generalitat Valenciana. El contenido de la publicación es responsabilidad exclusiva de MUSOL y no refleja necesariamente la opinión de la Generalitat Valenciana

Sobre la guia

A través de esta guía, se explican los pasos a seguir para crear el ***prototipo*** de un video juego sencillo con el motor de juego Scratch.

En el marco del proyecto “Integración de la educación para el desarrollo en la educación de personas adultas – fase 3. Género, Migraciones y Cambio Climático” – ejecutado en consorcio con Psicólogas Sin Fronteras gracias al apoyo financiero de la Generalitat Valenciana – se han diseñado actividades dirigidas a dotar al profesorado y alumnado de competencias cognitivas, actitudinales, procedimentales y pedagógicas para abordar los objetivos de desarrollo sostenible relacionadas con el cambio climático. Entre estas actividades se encuentra la elaboración de la presente “Guía de iniciación al diseño y desarrollo de videojuegos en Scratch”.

Esta guía pretende ser un instrumento pedagógico que permita al profesorado abordar en sus aulas desafíos medioambientales y sociales a través de la programación de videojuegos. A través de esta guía, se detallan los pasos a seguir para crear el prototipo de un video juego sencillo con el motor de juego Scratch. Se explica el funcionamiento del motor de juego Scratch y se explican las etapas a seguir para elaborar, en seis sesiones, un videojuego básico. Diseñada para ser utilizada por un público sin experiencia previa en programación de videojuegos, esa guía cuenta con un lenguaje sencillo y con el apoyo de materiales visuales. Se podrá usar en las aulas para elaborar de forma participativa y con una herramienta atractiva e innovadora -Scratch- videojuegos que aborden desafíos relacionados con los ODS, a la vez que se fomenten las competencias tecnológicas y digitales del alumnado."

Índice

¿Qué es Scratch?

Scratch es un motor de juego que se centra principalmente en la creación de videojuegos, animaciones e historias interactivas en dos dimensiones. Esta herramienta permite que sus usuarios programen a través de un lenguaje de programación visual basado en bloques muy intuitivo, ideal para aquellas personas que no tienen experiencia previa en el mundo de la programación.

Scratch se puede usar tanto de forma online a través de un navegador o de forma offline descargándolo. Se recomienda usar la versión online, ya que así nos aseguramos de que estamos usando la versión más actualizada del programa. Una de las ventajas de esta herramienta es que podemos compartir nuestros proyectos con la comunidad de Scratch y podemos recibir comentarios por parte de sus usuarios y usuarias. Por otro lado, podemos acceder a los proyectos de otras personas y ver cómo están creados o como han sido programados.

La página oficial de Scratch está repleta de tutoriales y material complementario tanto como para el alumnado como para el profesorado que nos puede ayudar a aprender a usar esta herramienta y que nos puede inspirar a crear nuestros propios proyectos.

Al final de la presentación adjuntamos un glosario donde explicamos algunas palabras que puedan resultar complejas.

Ir al glosario

Clase 1 - Introducción

En la primera clase, realizaremos una dinámica colectiva con el alumnado para poder identificar y debatir sobre los desafíos existentes en el barrio. A continuación, haremos una selección de las temáticas medioambientales o sociales a abordar en los videojuegos del alumnado**.**

Iniciaremos al alumnado en la narrativa de forma breve y concisa, para que puedan desarrollar la historia que quieran contar en sus casas.
Les propondremos crear ellos mismos el diseño narrativo del juego ¿Que quieren contar?:

Los videojuegos avanzan mediante Quest (misiones) nosotros crearemos una Quest principal con una estructura muy básica:

- Un personaje nos pide ayuda y nos otorga una misión importante, para ello deben crear narrativamente este personaje, decidir quien es, sus necesidades, como se expresa, sí lo conocemos previamente y redactar sus palabras.
- El protagonista debe cumplir dicha misión (no habrá aporte narrativo ya que es la parte jugable)
- El protagonista cumplirá la misión y volverá a hablar con el personaje inicial para cerrar la historia. Los alumnos deberán crear este cierre de historia por su cuenta.

Con estos tres sencillos pasos los alumnos podrán dar color a su videojuego mediante la narrativa, dotando de distinta personalidad a cada uno de los proyectos. Como se trata de un proyecto centrado al desarrollo y la programación, son los propios alumnos los que deben desarrollar en paralelo a las tareas los diálogos, teniendo en mente la historia que quieren contar y redactando al final del proyecto. Se trata de un trabajo de fondo que se realizará a lo largo del curso donde se irán escuchando las ideas de los alumnos y resolviendo dudas.

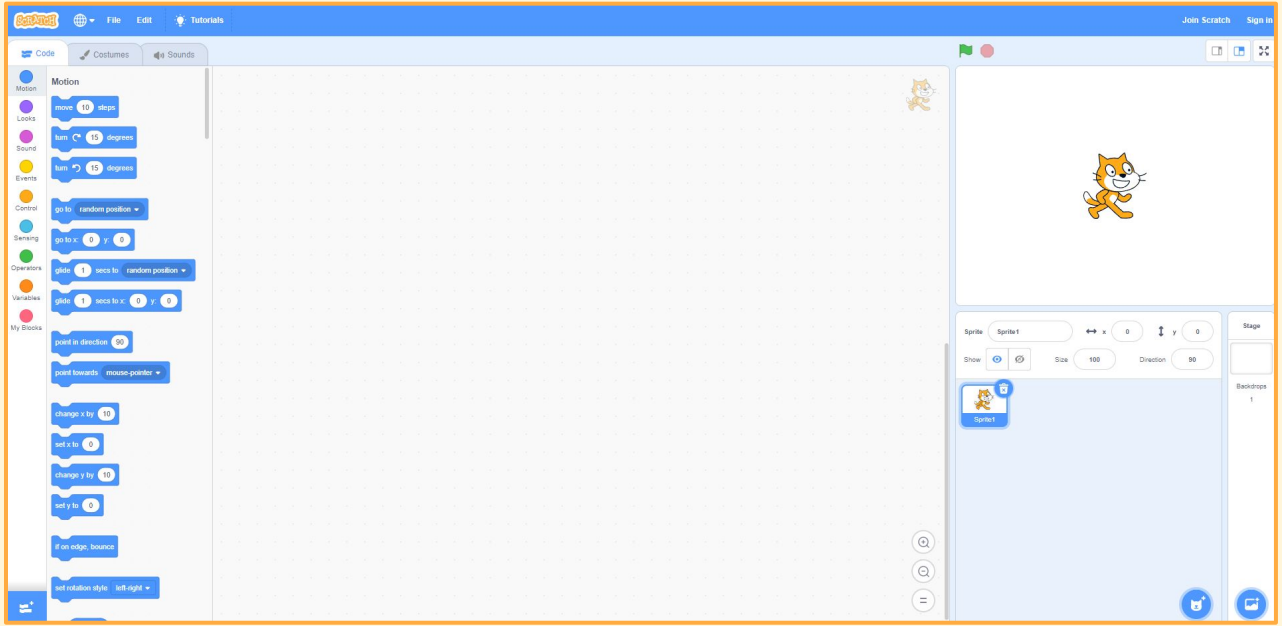
Clase 1 - Extra

Para aquellos que tengan interés se les explicará brevemente el camino del héroe y se les recomendará la lectura del libro *El viaje del héroe de Joseph Campbell,* así como buscar información relacionada en internet para poder dar más brillo a sus historias. La guionización, creación de personajes y de historias es una parte importante en la industria del videojuego que no necesita de medios para desarrollarse.

Este apartado no formará parte del curso activamente pero es un dato relevante que podría motivar a alguno de los alumnos.

Apartado 1 - El editor de código

Para poder empezar a trabajar en nuestro proyecto primero tenemos que comprender cómo funciona Scratch y qué elementos tiene. Para ello iremos a la página oficial de Scratch donde en la parte superior junto al logo le daremos al botón “crear” para crear un nuevo proyecto y nos familiarizaremos con el interfaz lo mantendremos en inglés ya que si quieren seguir explorando utilizarán el lenguaje estándar.

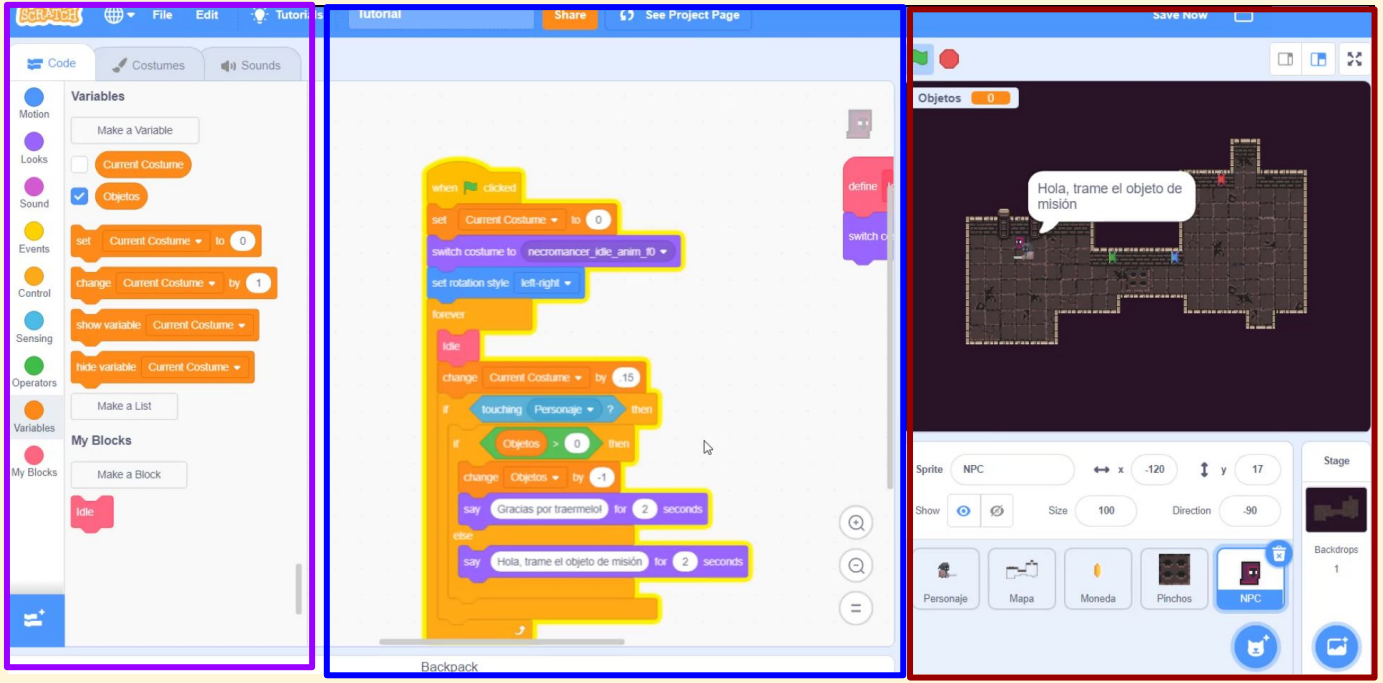


Este será el aspecto inicial de nuestro proyecto

Ir a Scratch

Clase 2 - Introducción a Scratch

Esta sería la imagen del proyecto “final” la aprovechamos para enunciar los tres apartados principales del interfaz, los dividiremos por: izquierdo, central y derecho, a continuación explicamos su funcionamiento



El **panel izquierdo** contiene todos los bloques que usaremos para programar el código del proyecto. Estos bloques son instrucciones sencillas que iremos combinando para hacer que nuestro juego funcione. Las explicaremos en detalle más adelante.

El **panel derecho** sirve para gestionar la parte gráfica del proyecto. La **parte superior** es la ventana donde podremos ver el juego mientras que la **parte inferior** es el listado de todas las Sprites (imágenes 2D) del proyecto, donde podemos renombrarlas o cambiarles la posición, rotación y escala. Las Sprites son cualquier tipo de actor (elementos interactuables) que podemos incluir en el juego, cada uno con su propio código de programación. Estas pueden ser desde los personajes hasta cajas o cualquier tipo de elemento jugable del proyecto. Por último, junto al listado de Sprites está el listado de Backdrops. Un Backdrop es el fondo sobre el que colocamos los elementos del juego, que también puede tener su propio código de programación, aunque nosotros lo utilizaremos únicamente con fines estéticos para crear el fondo del escenario.

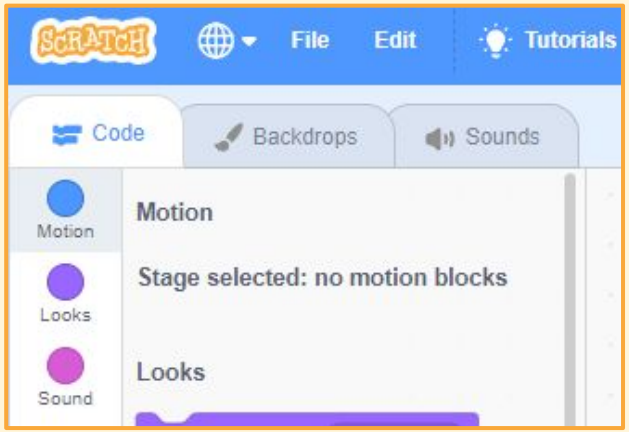
Por último el **panel central** es el espacio de trabajo donde colocaremos los bloques (nombrados anteriormente) para programar nuestro juego. La programación que aquí se ve es la del Sprite o el Backdrop que tengamos seleccionado en ese momento.

Editor de Sprites

Además de estos tres paneles, en la esquina superior izquierda podemos ver que también hay tres pestañas. Todo lo que hemos visto hasta ahora está dentro de la pestaña "**Code**", que es la que más vamos a utilizar.

La siguiente, "**Costumes**" o "**Backdrop**", sirve para cambiar la apariencia de la Sprite o Backdrop que tengamos seleccionada. Desde aquí podemos modificar el aspecto visual de todos los elementos del proyecto. Principalmente la utilizaremos para animar o para decorar el fondo. Veremos cómo trabajar en profundidad con este editor más adelante.

La última pestaña es "**Sounds**", que sirve para editar los sonidos del proyecto. Como vamos a crear un prototipo no necesitaremos trabajar con ella.



Tipos de Bloques:

En la pestaña Code tenemos varias bolitas de colores, cada bolita representa un tipo de bloques, procedemos a explicarlos:

Motion (azul oscuro): Esta categoría gestiona el movimiento del personaje, pudiendo modificar o conocer cosas como su rotación o su desplazamiento.

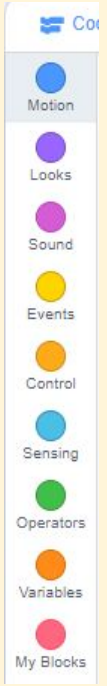
Looks (morado): Nos permite gestionar el aspecto del personaje. Aquí podemos incluir diálogos, cambiar el tamaño del personaje, añadir algunos efectos e incluso animarlo.

Sounds (malva): Estos bloques sirven para ejecutar y modificar sonidos, aunque no lo utilizaremos para este proyecto.

Events (amarillo): Esta categoría define cuándo pasan las cosas, como por ejemplo qué sucede cuando pulsas una tecla, cuando empieza el juego, etc. Es uno de los que más vamos a utilizar, aunque intentaremos mantenerlo lo más sencillo posible.

Control (naranja): Estos bloques ya son más cercanos a la programación típica, aunque también intentaremos utilizarlos de la forma más sencilla posible. Con esto podemos gestionar todo lo que sucede en nuestro código, haciendo que repita varias veces otras instrucciones, decidiendo si ejecutar unas instrucciones u otras, etc.

Sensing (azul clarito): Con esto podemos saber qué está sucediendo dentro del juego, como si el personaje está tocando otro elemento de la escena, o qué teclas estamos tocando en ese momento. Puede sonar parecido a “Events”, pero veremos las diferencias a medida que lo vayamos aplicando.



Operators (Verde): Esta parte también está muy relacionada con la programación típica, y puede resultar un poco más complicada de utilizar que otras. Permite hacer comparaciones lógicas y operaciones matemáticas.

Variables (Naranja Oscuro): Aquí podemos crear y editar las variables de nuestro proyecto. Las variables son toda la información que queremos guardar, como por ejemplo cuántas monedas ha recogido nuestro personaje, cuanta salud le queda, etc.

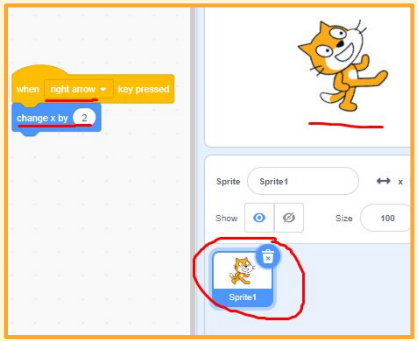
My Blocks (rosa): En “Mis Bloques” podemos crear nuestros propios bloques para reutilizar código. Si hay trozos de código que vayamos a repetir muchas veces, podemos convertirlo en un solo bloque rosa para que nos sea más fácil trabajar. En programación a esto se lo conoce como “funciones”. Es parecido a “Events”, aunque iremos viendo las diferencias al trabajar con ambos.

Con estos conocimientos en mente comenzamos a diseñar nuestro prototipo.

Programar el Movimiento Básico

Antes de avanzar es conveniente ver un ejemplo práctico de cómo programar en Scratch. Para ello haremos que la mascota de Scratch se mueva de forma muy básica, afrontaremos algunos problemas que puedan surgir y entenderemos por qué utilizamos algunos bloques y no otros.

Lo primero será extraer de los bloques divididos por bolitas de colores que hemos mencionado antes, el material necesario y arrastrarlo hasta la zona central del proyecto. Empezaremos por la **bolita amarilla "events"**, añadiremos la **caja amarilla** (estas cajas reciben el nombre de **bloques**) que pone "When **[-----]** key pressed" en el espacio donde ahora hay guiones, hay un menú desplegable, despliégalo y selecciona la opción "When [right arrow] key pressed" a continuación con el sprite del personaje seleccionado añadimos el **bloque motion** "Change X by [2]" y la ponemos justo bajo el anterior bloque. Ahora si pulsas la tecla derecha, el gato se mueve en esa dirección. Se puede cambiar el número para editar la velocidad a la que se mueve. Parece que podríamos usar el bloque "Move [] Steps", pero este hará que nuestro personaje avance siempre en la dirección en la que mira, lo que no nos conviene para este proyecto. El problema de usar "When [right arrow] key pressed" es que si pulsas esa tecla siempre se moverá, quieras o no. Nos es más conveniente separar cuando iniciamos el juego y cuándo acaba, para que toda la programación, como la de los controles, funcione sólo mientras se juega.

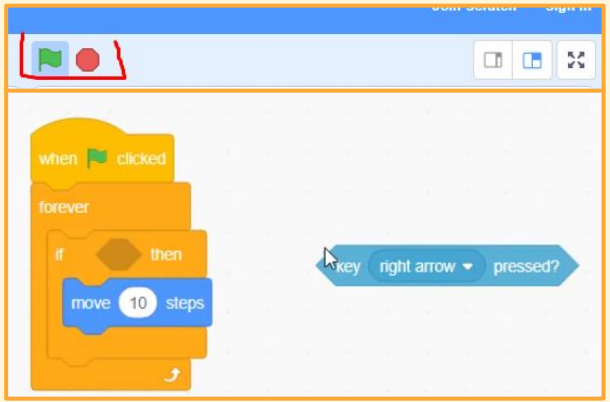


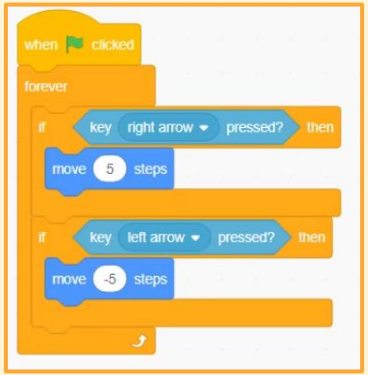
Clase 2 - Introducción a Scratch

Cambiamos de tarea, borramos los bloques anteriores y colocamos el **evento** de “When (Green Flag) Clicked”. Este evento se activa cuando pulsamos la bandera verde que hay sobre la pantalla del juego, marcando el inicio de la partida. Si pulsamos la señal roja detenemos el juego, dejando de ejecutar el código.

A continuación colocamos el **bloque control** “Forever”. Es una caja un poco distinta, lo que introduzcamos en ella será ejecutado continuamente, muchas veces por segundo.

Dentro de “Forever”, colocamos el **bloque control** “If [ ] then”, dentro de este nuevo nodo añadiremos como antes el **bloque motion** “Change X by [2]”. Esto comprueba una condición, y si se cumple entonces el personaje se mueve. Como podemos apreciar las cajas tienen formas que conectan entre sí, para añadir la condición que acabamos de mencionar debemos añadir el **bloque sensings** “key [ ] pressed?”, que tiene forma hexagonal, y debemos introducirlo en el hexágono que hay en el interior de la caja “If [ ] then” quedando “if [right arrow] pressed”.





Realmente la programación por bloques es como escribir órdenes. Viendo esta podemos leer que al iniciar el juego comprueba continuamente si se pulsa la tecla derecha, y si es así mueve el personaje en esa dirección.

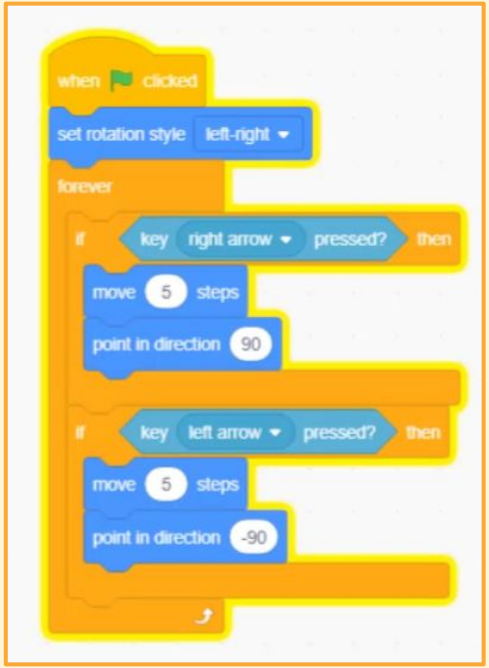
Podemos repetir este proceso duplicando los bloques dentro de “Forever”, poniendo esta vez la flecha izquierda, y cambiando la velocidad por el mismo número en negativo, para que se mueva a la izquierda.

Solo que en este caso, al caminar hacia la izquierda el gato no se gira, sino que se desplaza de espaldas. En Scratch el personaje mira hacia la derecha si su rotación es de 90º, y mira hacia la izquierda si su rotación es de -90º.

Para arreglarlo vamos a colocar el **bloque motion** "point in direction [90]" cuando queramos que el personaje se mueva a la derecha, y "point in direction [-90]" para la izquierda. Finalmente añadimos el **bloque motion** "set rotation style [left-right]" justo bajo el evento de iniciar el juego, para que nuestro personaje pueda rotar correctamente.

Para hacer que se mueva verticalmente nos basta con añadir dentro del "Forever" otro **bloque control** "If [ ] then", con la condición "is [up arrow] pressed", y dentro el **bloque motion** "Change Y by [2]". Si repetimos lo mismo con "is [down arrow] pressed" y "Change Y by [-2]" el personaje podrá moverse hacia abajo.

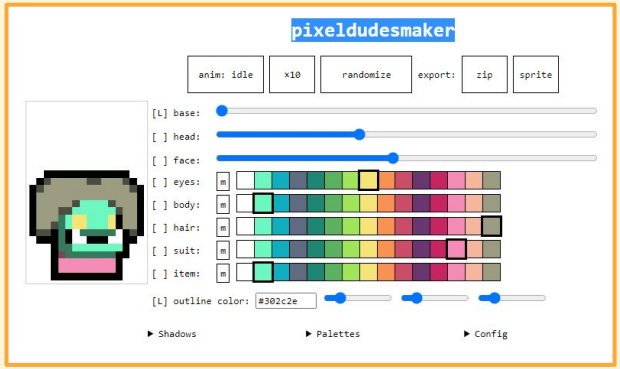
Con todo esto hemos logrado crear el movimiento del personaje de nuestro proyecto. Lo renombramos como "Character" para poder localizarlo fácilmente, y le ponemos la escala a 50. En la parte inferior del panel derecho podemos ver el nombre y la información de las sprites si clicamos encima del nombre podemos editarlo, y lo mismo sucede con la escala.



Crear el Personaje

Tenemos dos herramientas diferentes para diseñar nuestro personaje, ambas igualmente válidas: Pixeldudesmaker y pixel art rpg character creator. Cada una tiene su estilo y parámetros para editar el personaje, e incluyen una opción para crearlo al azar.

Ahora podemos divertirnos un rato probando las distintas opciones y creando el personaje que más nos guste. Una vez tengamos el personaje, lo exportamos (“export: sprite” para pixeldudesmaker y “export / save” para Pixel Art Character Creator). Esto nos creará una sprite sheet, que es una imagen con todas las animaciones del personaje.

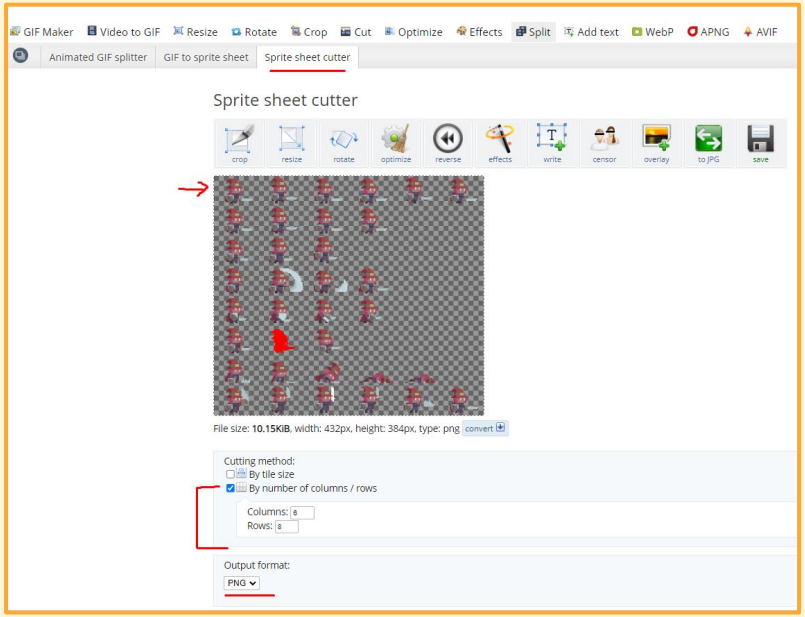


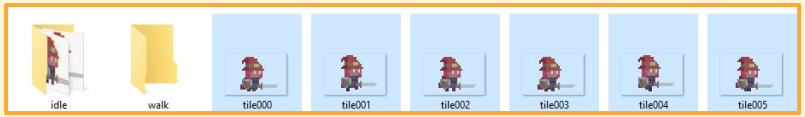
Para importar el personaje y sus animaciones en Scratch tendremos que trocear esa sprite sheet, para tener cada imagen de la animación por separado. Ezgif.com nos permite hacerlo fácilmente. Dentro del apartado “Split” está la pestaña “Sprite Sheet Cutter”, donde podemos arrastrar nuestra sprite sheet para trocearla.

Después de subir nuestra imagen y pulsar “Upload”, accederemos al editor de imagen. Seleccionamos el método “by columns/rows” y ponemos la cantidad de “columnas y filas” que tiene nuestra imagen para que la recorte correctamente.

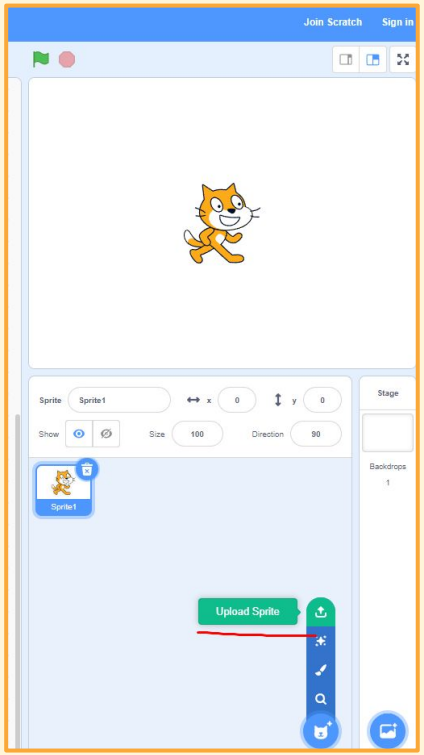
Nos aseguramos de que el formato elegido es PNG, pulsamos en “Cut!” y lo guardamos como un ZIP. Al descomprimir las imágenes (con 7zip o similar) es recomendable hacer carpetas y guardar en ellas las dos animaciones que vamos a utilizar.

- Idle (imágenes de la 5 a la 8 para pixeldudesmaker, y de la 7 a la 10 para Pixel Art Character Creator): Animación para cuando el personaje no se mueve.
- Walk (imágenes de la 9 a la 12 para pixeldudesmaker, y de la 1 a la 6 para Pixel Art Character Creator): Animación para cuando el personaje se mueve.





Clase 3 - Animaciones en Scratch



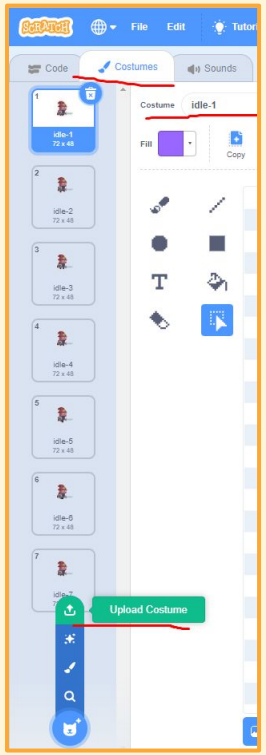
Importar el Personaje

Dentro de Scratch, en la parte inferior derecha donde está la lista de Sprites y Backdrops, hay un desplegable que nos permite importar nuestros propios sprites.

Importamos la primera imagen de la animación de idle, lo que nos creará la sprite de nuestro personaje. Ahora, en la pestaña “costumes” podemos encontrar el mismo desplegable para importar abajo a la izquierda, e importamos el resto de imágenes del idle.
Para que sea más fácil trabajar con ello, renombramos la primera imagen del idle como “Idle-1”, y así sucesivamente con todas.

Aprovechamos ahora para importar las imágenes de la animación de walk, y las renombramos del mismo modo.

Con el personaje ya importado, **arrastramos el código** que creamos en la clase anterior para el personaje de Scratch **sobre el sprite de nuestro personaje** para copiar el código. Una vez hecho esto, ya podemos borrar el gato de Scratch para trabajar sólo con los elementos de nuestro proyecto.



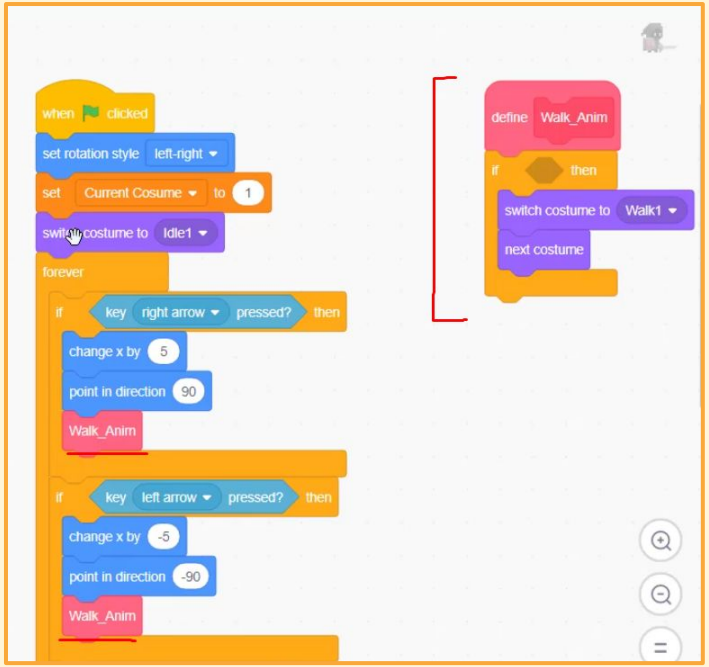
Animar el Personaje

Como tenemos cuatro entradas de movimiento, una por cada dirección, pero queremos que el personaje haga la animación de caminar (walk) en todas ellas, vamos a crear un bloque propio para no repetir el mismo código todo el rato.

Creamos un bloque en el apartado **"My Blocks"**. Vamos a llamarlo Walk_Anim, y vamos a habilitar la pestaña inferior para que se ejecute sin el refresco de la pantalla (esto crea automáticamente un bloque llamado "define [Walk_Anim]" que utilizaremos más adelante). Añadimos este nuevo bloque dentro de cada uno de los **bloques control** "If [] then" al final del código .

Pero para poder gestionar la animación tendremos que saber si el personaje ya estaba andando o no, así que dentro del apartado **"Variables"** creamos una, sólo para este sprite, que llamaremos "Current Costume". Puedes quitar el tick que hay en el listado de variables para no verla en la pantalla de juego.

Esta variable representa qué imagen tiene ahora el personaje, de todas las que hemos importado. Queremos que al iniciar el juego siempre empiece estando quieto, así que bajo el **bloque motion** "set rotation style [left-right]" colocamos otro de **variable**, "set [Current Costume] to [0]", y otro de **looks**, "switch costume to [idle 1]"



Clase 3 - Animaciones en Scratch

La forma más intuitiva de abordar las animaciones sería con un “if [ ] then”, comprobando si el personaje estaba ya haciendo la animación de andar, para que si es así continúe, y si no la inicie desde el principio.

Pero nosotros vamos a usar una fórmula matemática para calcularlo automáticamente, mejorando el rendimiento del proyecto. Necesitaremos del **bloque looks** “switch costume to [ ]”, la **variable** “Current Costume”, y **los bloques operator** “[ ] + [ ]”, “floor of [ ]” (puede poner ABS en lugar de floor o similar ya que es un menú desplegable) y “[ ] mod [ ]”. El conjunto de los bloques sería algo tal que:

“Switch Costume to [[5] + [[floor of [Current Costume]] mod [6]]]”

5 es la imagen en la que comienza la animación de andar, y 6 es la cantidad de imágenes que tiene esa animación, ya que esta va del 5 al 10. Todo esto se puede ver en la pestaña de “Costumes”.

“Floor” lo que hace es eliminar los decimales, mientras que “mod” se asegura que el primer número que introducimos nunca supere el segundo, y finalmente vamos a usar “Current Costume” para calcular el avance del tiempo.

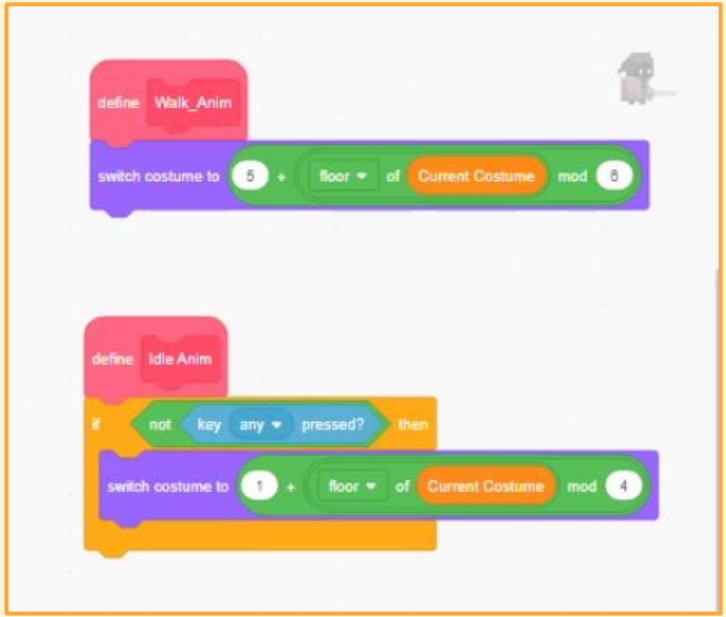
Colocamos estos bloques bajo el de “define [Walk_Anim]”(que se creó automáticamente antes), a continuación introducimos al final del código que hemos creado dentro de **“forever”** el **bloque de variable** “Change [Current Costume] by [0.5]”. Con este último número podemos elegir la velocidad a la que queremos que vayan las animaciones.

Clase 3 - Animaciones en Scratch

Crearemos también la animación de Idle. Para eso crearemos desde **"My Blocks"** un bloque llamado Idle_Anim, nuevamente haremos que se ejecute sin refresco de imagen. Al definirlo tenemos que comprobar si el personaje no se está moviendo, para ejecutar la animación. Para eso le pondremos un **bloque control** "If [ ] then", y en el comprobador (el rombo entre if y then) ponemos un **bloque operador** "not [ ]", y dentro del espació del "not [ ]", un **bloque sensing** "key [any] pressed?" (en any puede poner space u otra palabra ya que es un menú desplegable) . Los juntamos de modo que el bloque quede como:

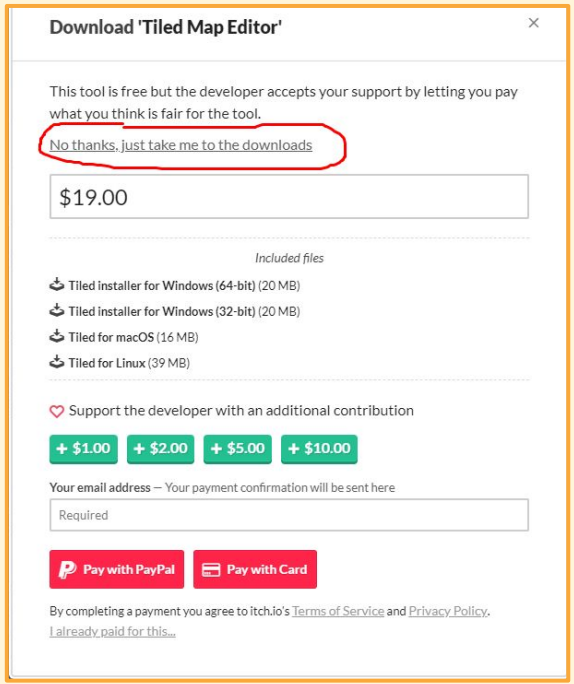
"If [not[key [any] pressed] then"

Dentro de este "If [] then" ponemos la misma operación que hicimos para la animación de andar, cambiando los valores de la imagen de inicio y la cantidad de imágenes que tiene la animación de idle, quedando tal que así:
"Switch Costume to [[1] + [[floor of [Current Costume]] mod [4]]]"



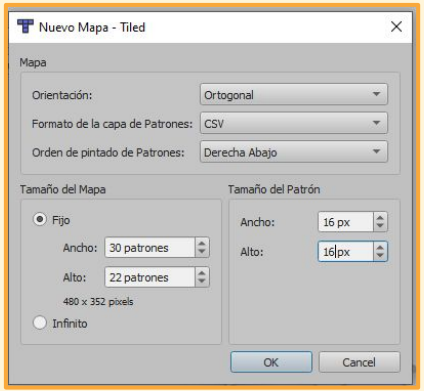
Creación de Entornos en Tiled

Para la creación del entorno va a ser necesaria la sprite sheet del entorno que aportamos para el ejemplo, y la descarga del Tiled Map Editor.



Al darle al botón “Download” aparecerá este cartel, es necesario darle a “No thanks, just take me to the downloads” para no abonar nada ya que el programa es gratuito aunque podemos donar si lo deseamos.

Al iniciar el programa seleccionamos la opción “Nuevo Mapa”. Lo importante es que la orientación sea Ortogonal, que tenga 30 patrones de ancho por 23 patrones de alto, y que el tamaño del patrón sea de 16 x 16 píxeles.

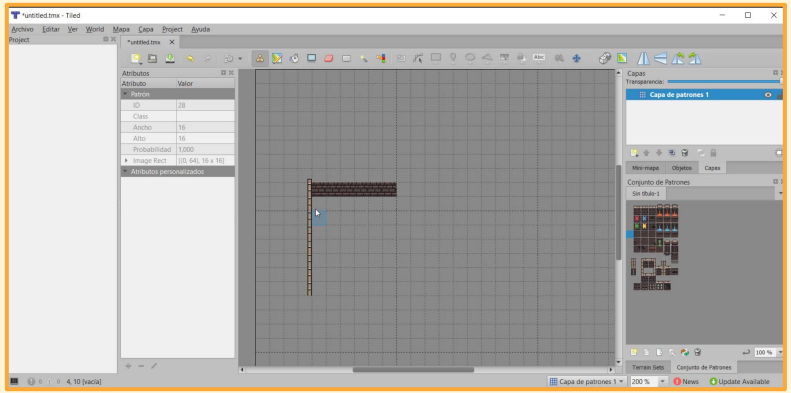


El siguiente paso es clickar en “importar un nuevo conjunto de patrones”, botón que está en el menú de abajo a la derecha. Donde pone “explorar” seleccionamos la imagen que hemos preparado, y la importamos.

Ahora hay que dibujar las paredes del nivel en el que vamos a jugar. Hay que usar las imágenes de bordes y paredes de forma que encajen entre ellas, siendo creativo a la hora de diseñar el nivel. Lo importante a tener en cuenta es que todo lo que coloquemos aquí ahora tendrá colisión con el personaje, no se podrá atravesar, y que no podremos modificarlo después en Scratch.

Por lo tanto, evitaremos utilizar las piezas del suelo, que se aplicarán más adelante. Lo que sí pueden usarse son piezas diferentes de pared e incluso columnas, para ir decorando el entorno.

Una vez tengamos el entorno como nos guste, vamos a “Archivo”, y seleccionamos “Exportar Como Imagen”. Lo nombramos como “Map” y lo exportamos.



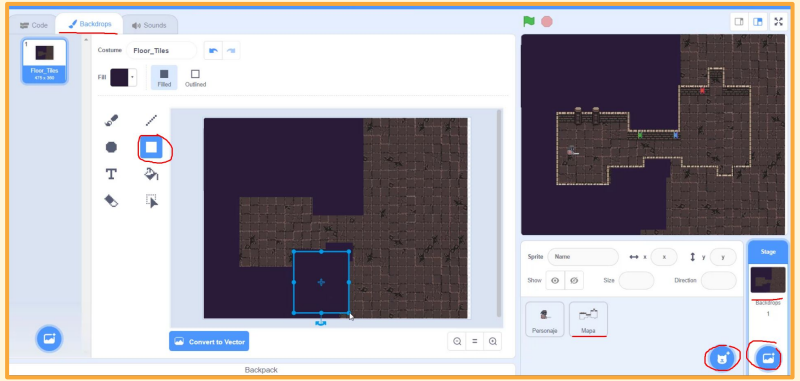
Importar y Ajustar el Entorno

Con esto podemos cerrar el Tiled, y volvemos a Scratch. Ahí importamos este mapa como si fuera otra Sprite. Para ajustarlo correctamente, vamos a darle al mapa un tamaño de 200.

También vamos a facilitar una imagen llamada Floor_Tiles para las losas que representan el suelo de nuestro escenario. Lo importamos como Backdrop, y movemos el mapa que habíamos creado para que encaje con las losas del suelo.

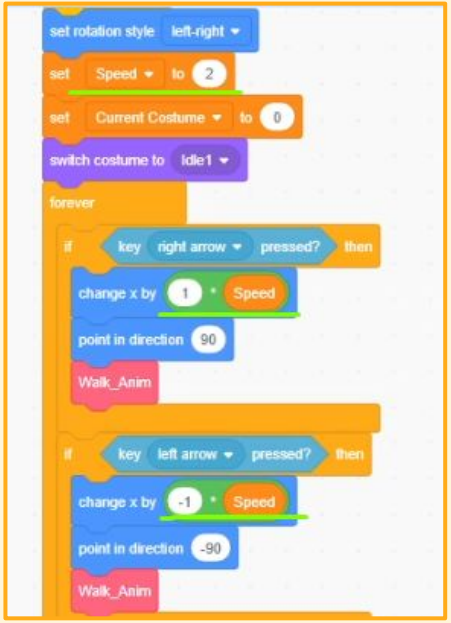
Con el backdrop seleccionado, vamos a lo que era la pestaña de “Costumes”, que esta vez también se llamará “Backdrop”.

Seleccionamos el pincel cuadrado y elegimos un color discreto para el exterior de nuestro mapa. Dibujamos un rectángulo sobre el fondo que queda fuera del mapa que habíamos creado, podemos ajustarlo estirando de los bordes para que se acople a una de las paredes de nuestro mapa. Repetimos el proceso todas las veces que sea necesario para tapar el exterior del mapa, viéndose sólo el suelo de la parte que hemos delimitado con paredes.



Iniciar Correctamente el Personaje

Para hacer que sea más fácil de modificar la velocidad del personaje, vamos a crear sólo para esta sprite una **variable** que llamaremos “Speed”. Tras el **evento** de inicializar ponemos el **bloque variable** “set [speed] to [2]”, y en cada **bloque** “change x by[ ]” y “change y by [ ]” que pusimos para el movimiento, en esos huecos “[]” ponemos esta **variable** speed, junto al **bloque operator** de multiplicación, “[ ] * [ ]”. En el movimiento para arriba y derecha multiplicaremos Speed por 1, y en el de izquierda y abajo multiplicaremos Speed por -1.



Además, es conveniente poner tras el **evento** de iniciar el juego los **bloques looks** “show”, “switch backdrop to [floor_tiles]” y “go to [front] layer”. De este modo nos aseguramos de que empezamos en el mapa en el que toca y que siempre será visible el personaje.

También es recomendable poner tras estos bloques al iniciar el juego el **bloque motion** “go to x [ ] y [ ]”, donde pondremos las coordenadas en las que queremos que empiece siempre nuestro personaje.

Clase 5 - Colisiones y objetos

Rehacer el Movimiento del Personaje

Ahora mismo nuestro personaje puede atravesar las paredes del escenario. Para evitarlo tendremos que crearle una colisión para que pueda chocar contra ellas.

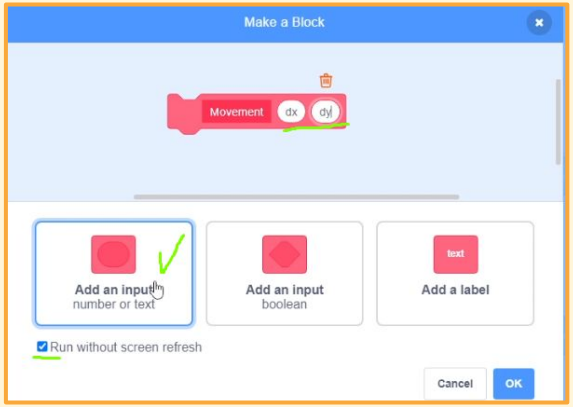
Empezaremos creando otro bloque desde **"My Blocks",** también sin refresco de pantalla, al que llamaremos Movement, pero en esta ocasión añadiremos una de las opciones que nos ofrece Scratch "add and input, number or text". Estas dos entradas las llamaremos "dx" y "dy", y representan la velocidad del personaje en cada uno de los ejes.

Vamos a sustituir los **bloques** "Change X by [ ]" que utilizamos para el movimiento por este nuevo **bloque** "Movement" que hemos creado. El valor que utilizábamos, los **bloques** "[Speed] * [1]" para el movimiento a derecha y "[Speed] * [-1]" para izquierda, los colocamos en la primera entrada del **bloque** Movement, y en la segunda ponemos un 0. Por ejemplo, para el movimiento a la derecha nos quedaría tal que así:

"Movement [[Speed]*[1]] [0]"

Esto es porque haremos que el primer valor sea la velocidad con la que nos movemos en el eje horizontal, y la segunda con la que nos movemos en el vertical. Ahora repetimos el proceso al sustituir los **bloques** "Change Y by [ ]" que hemos usado para el movimiento vertical, poniendo su valor en la segunda entrada del **bloque** "Movement", dejando la primera como 0. Por ejemplo, esta vez el movimiento hacia arriba nos quedaría de este modo:

"Movement [0] [[Speed]*[1]]"



Clase 5 - Colisiones y objetos

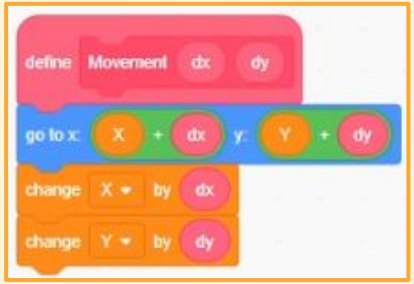
Con esto estamos agrupando el movimiento del personaje en una sola caja. En principio nos obliga a calcular el movimiento de una forma un poco más complicada, pero nos ayudará a gestionar las colisiones.

A continuación creamos dos **variables** sólo para esta Sprite, que llamaremos “X” e “Y”. Con ellas marcaremos la posición del personaje en todo momento. Bajo el **evento** de inicializar pondremos los **bloques de variables** “Set X to [ ]” y “Set Y to [ ]”, y les asignaremos los valores X (-145) e Y (-10) que utilizamos para poner al inicio nuestro personaje.

Ahora volviendo al bloque **Define “Movement [dx] [dy]”**. Podemos arrastrar las entradas “dx” y “dy” para poder trabajar con ellas, que representan los valores que aplicamos cada vez que hemos usado este bloque.

Vamos a definirlo colocando debajo el **bloque de movement** “go to x [ ] y [ ]”. Lo que queremos hacer es sumar el movimiento a la posición actual del personaje, y actualizarla. Para ello dentro del nuevo bloque añadiremos **el “[]+[]” de operations** en ambos huecos y usaremos nuestras **variables** “X” e “Y” (respectivamente) para ocupar el primer hueco de la nueva caja de operations, y les sumaremos las entradas de “dx” y “dy (respectivamente) que ocuparon el segundo hueco de dicha caja. El resultado sería algo tal como:

“go to x[[X]+[dx]] y[[Y]]+[dy]]”



A continuación de esto ponemos del **bloque de variable** “Change [X] by[ ]”, y como parámetro en el hueco añadimos “dx”, justo debajo ponemos del **bloque de variable** “Change [Y] by[ ]”, y como parámetro “dy”.

Con todo esto podemos ver que el movimiento del personaje se siente exactamente igual que lo hacía antes, con la diferencia de que lo hemos unificado dentro de un único bloque que podremos editar fácilmente.

La Colisión del Personaje

Ahora que hemos unificado el movimiento, vamos a hacer que el personaje se detenga al chocar contra las paredes de nuestro mapa. Entre los bloques “go to x [ ] y [ ]” y “Change [X] by[ ]” del movimiento, colocamos del **bloque de control** “If [ ] then”,como parámetro en el hueco introducimos del **bloque sensing** “Touching [Map]?”, para ejecutar el interior del if si el personaje toca nuestro sprite de mapa.

Dentro del **if** vamos a introducir del **bloque motion** “go to x [ ] y [ ]”, y le asignamos a la x nuestra **variable** “X” y a la y la **variable** “Y”, y a continuación del **bloque de control** “Stop [this Script]”. Con esto hacemos que al chocar con las paredes nuestro personaje no avance, y deje de moverse.

Pero si nuestro personaje lleva una espada o cualquier otro elemento, este chocará. Para evitarlo vamos a crear un Collider, que será la única parte del personaje que choque con todo. Dentro de la pestaña Costumes, duplicamos la primera imagen del Idle y la arrastramos al final de la lista de imágenes. Creamos un rectángulo por encima que ocupe el tamaño que queremos que choque contra las cosas, y borramos con la goma todas las partes del personaje que sobren alrededor. A esta imagen la llamaremos Collider.

Para aplicarlo, en el código de **define “Movement [dx] [dy]”,** ponemos antes que el resto un **bloque looks** “Switch Costume to [Collider]”. Esto lo que hace es poner la imagen de la colisión para calcular cuando choca contra el entorno, y automáticamente cambiarla por la que queremos que se vea, tan rápido que ni nos damos cuenta.

Clase 5 - Colisiones y objetos

Objetos

Ahora que nuestro personaje tiene colisión, podrá interactuar con otros elementos del entorno. Para el entorno vamos a coger elementos del Dungeon Tileset II.

Buscamos entre sus archivos la coin_anim, una moneda, y la importamos, junto con todas las imágenes para su animación, como ya hicimos con el personaje. Le damos una escala de 120 para ajustarla al tamaño del juego.

La moneda también empezará a funcionar al iniciar el juego, así que le añadimos un **bloque de evento** "When Green Flag Clicked". Creamos una **variable** solo para esta sprite llamada "Current Costume", y bajo el evento de iniciar el juego ponemos del **bloque variable** "set [Current Costume] to [0]", y del **bloque looks** "switch costume to [coin_anim_f1]" y "Show". A continuación ponemos un **bloque de control** "Forever".

Ahora en **"My Blocks"** creamos un bloque llamado "Idle Anim", sin refresco de pantalla, y lo colocamos dentro del forever. A continuación del "Idle Anim" ponemos del **bloque de variable** "change [Current Costume] by [0.2]" para decir a qué velocidad va la animación.

Ahora dentro del **bloque de define** "Idle Anim" ponemos la misma serie de bloques que ya hicimos con el personaje, con los valores adecuados de imagen de inicio y cantidad de imágenes de la animación, siendo algo como esto:

"Switch Costume to [[1] + [[floor of [Current Costume]] mod [4]]]"

Como podemos ver, lo mismo que hicimos para el personaje sirve para cualquier elemento animado. Para gestionar animaciones va a ser repetir siempre el mismo proceso.

Sólo nos queda implementar qué pasa cuando la recoge el personaje. Creamos una **variable** para todas las sprites llamada “Coin Amount”. Nos conviene desactivar el tick del resto de variables, también de nuestro personaje, y dejar sólo esta activa, para poder ver en todo momento la cantidad de monedas que tenemos.

Dentro del **bloque** “Forever”, sobre los otros dos, ponemos un **bloque** “If [ ] then”, y como comprobación ponemos un **bloque sensing** “is touching [character]?”. Dentro del if ponemos el **bloque** “Change [Coin Amount] by [1]”, del **bloque looks** “Hide”, y del **de control** “Stop [this Script]”. Con esto hacemos que si el personaje la toca, aumenta su cantidad de monedas, ocultamos esta moneda para que no se vea, y hacemos que deje de poder ser recogida de nuevo.

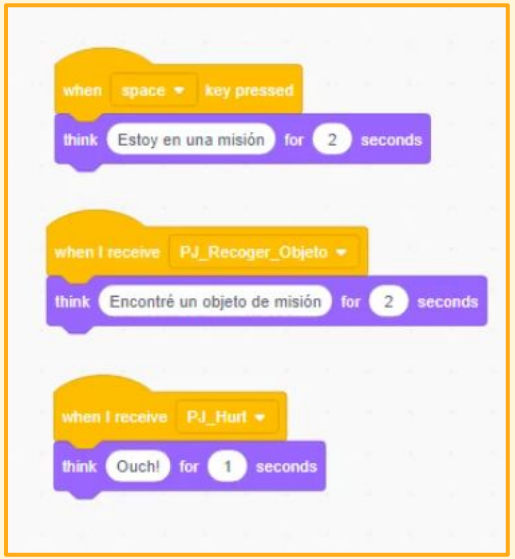
Finalmente para asegurarnos de que nuestro personaje siempre empieza sin monedas, volvemos a su código y bajo el evento de iniciar el juego le ponemos el **bloque variable** “Set [Coin Amount] to [0]”.

Clase 6 - Diálogos

Usar Bloques de Diálogo

Scratch tiene muy integrado el sistema de diálogos, haciendo que sea muy fácil de aplicar. Un ejemplo directo y sencillo es, volviendo al código de nuestro personaje, poner otro **bloque** “If [ ] then” dentro del “Forever”, al que esta vez le añadimos la **condición sensing** “is [spacebar] pressed?”. Dentro de ese “if” ponemos el **bloque looks** “think [ ] for [2] seconds”, al que le podemos poner el texto que queramos que piense el personaje al pulsar la barra espaciadora.

También podemos hacer que el personaje diga algo al recoger un objeto. Para eso añadimos al código del personaje un **bloque de evento** “When I Recieve [message]”. En lugar de message creamos un nuevo evento, que lo llamaremos “PC_Pick_Item”. Ahora, debajo de este evento volvemos a colocar el **bloque** “think [ ] for [2] seconds”, con el texto que queremos que piense el personaje al recoger el objeto.
Ahora desde el código de la moneda, añadimos el **bloque event** “Broadcast [PC_Pick_Item]” justo encima de donde colocamos el **bloque** “Stop [this Script]”.



Personajes No Jugadores

Ahora que sabemos cómo van los bloques básicos de diálogos, podemos crear otros personajes que interactúen con el nuestro. Importamos un nuevo personaje con su animación de idle, al que llamaremos NPC. Este puede ser creado con el pixeldudesmaker o el Pixel Art Character Creator, o también puede escogerse uno de los que hay en el Dungeon Tileset II.

Al igual que con nuestro personaje principal, a este le creamos la **variable** "Current Costume" sólo para esta sprite, y añadimos el **evento** "When Green Flag Clicked", seguido del **bloque** "Set [Current Costume] to [0]", y del **bloque looks** "Switch Costume to [idle_0]"y "Show", del **bloque motion**"Set Rotation Style [left-right]" y el "Forever".

Creamos en **my block** "Idle_Anim" sin refresco de pantalla. En la **definición de este bloque** pondríamos la fórmula de las animaciones, cuyos valores dependerán de nuestra animación de idle:

"Switch Costume to [[1] + [[floor of [Current Costume]] mod [4]]]"

Ahora ponemos dentro del "Forever" nuestro **bloque** "Idle_Anim", y el **bloque variable** "Change [Current Costume] by [0.15]".

Con esto tenemos al NPC con su animación, ahora queremos hacer que interactúe con nuestro personaje. Para eso dentro del "Forever", tras los bloques que ya tenemos, ponemos un **bloque** "If [ ] then", y como **condición** le ponemos "Touching [Character] ?".

De este modo reaccionará cuando nos acerquemos a él. Lo siguiente que queremos es que si llevamos objetos encima los reciba y nos de un mensaje, y si no los llevamos nos los solicite con un mensaje diferente.

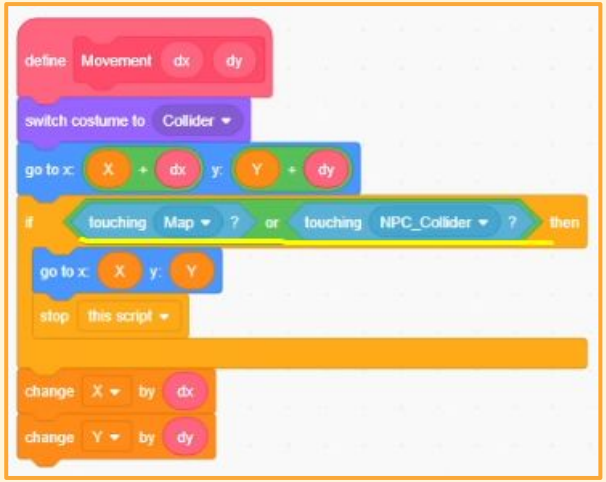
Para eso dentro de ese “If [ ] then” ponemos otro **bloque de control**, “If [ ] then else”. Este hace que si se da la condición, ejecute un trozo de código, y si no ejecute otra diferente. Como **condición** le ponemos “[Coin Amount] > [0]”, usando nuestra **variable** “Coin Amount”.

Dentro de la primera parte de este “If [ ] then else” ponemos del **bloque variable** “Change [Coin Amount] by [-1]”, y de **looks** “Say [ ] for [4] Seconds”, donde el NPC agradecerá que le hayamos dado el objeto.

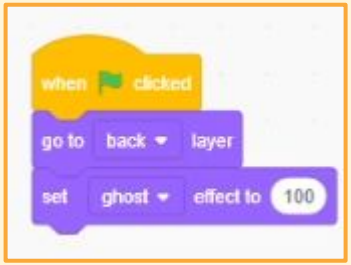
En la segunda parte de este “If [ ] then else”, que sucede cuando no tenemos ningún objeto, ponemos sólo del **bloque looks** “Say [ ] for [2] Seconds”, donde el NPC nos pedirá que vayamos a buscar el objeto.

Ya tenemos al NPC funcionando, su único problema es que podemos pasar por encima de él. Vamos a intentar hacer su colisión de una forma más rápida y sencilla, ya que este NPC no se va a mover del sitio.

Dónde están los personajes en el marco inferior derecho de la pantalla hacemos clic en el icono con forma de gato y seleccionamos paint. Creamos una nueva sprite que sea un rectángulo ligeramente más pequeño que NPC (veremos a tiempo real como queda en el mapa de juego, debemos hacer que cuadre sobre el NPC), a la que llamaremos NPC_Collider. Idealmente esta colisión tendría que estar centrada en el editor de imagen para que nos sea más fácil trabajar con ella.



La colocamos en el mapa encima del NPC para ver que se ajusta correctamente. Ahora, crearemos el código del NPC_Collider añadiendo el **bloque** “When Green Flag Clicked”, y a continuación los **bloques looks** “Go to [Back] Layer”, y “Set [Ghost] Effect to [100]”. De este modo estará ahí, pero no será visible.



Ahora para que esta colisión afecte a nuestro personaje, volvemos al código de Character. Donde **definíamos el bloque** “Movement [dx] [dy]” volvemos a ver la **comprobación** del “If [ ] then” que era “touching [Map] ?”. Vamos a añadirle el **operador** “[ ] or [ ]” para comprobar si toca la colisión del NPC también, de modo que quedaría:

“[Touching [Map] ?] or [Touching [NPC_Collider] ?]”

Contenido extra

Contenido extra - trampas

Trampas

Como ejemplo para montar una trampa en nuestro escenario vamos a coger elementos también del Dungeon Tileset II. En este caso importaremos como Sprite la floor_spikes_anim_f0, y añadiremos como costumes el resto de imágenes de este asset.

Como ya hicimos con el personaje y la moneda, creamos su **variable** para esta sprite solo “Current Costume”. Añadimos el **bloque de evento** “When Green Flag Clicked”, y a continuación ponemos de nuevo el **bloque variable** “set [Current Costume] to [0]”, y los **bloques looks** “switch costume to [floor_spikes_anim_f0]” y “Show”.

Como ya hicimos anteriormente, vamos a gestionar su animación. Creamos un **bloque** llamado “Anim”, sin refresco de pantalla. Bajo el bloque donde se define volvemos a poner la fórmula para las animaciones:

“Switch Costume to [[1] + [[floor of [Current Costume]] mod [4]]]”

La diferencia entre esta trampa y otras cosas que hemos creado antes, es que queremos que su animación se pause un momento cuando los pinchos están fuera o dentro del suelo, además de que no queremos que haga daño siempre al jugador, sólo mientras se vean los pinchos.

Para eso, creamos un nuevo **evento**, “When I Recieve [message]”, que nombraremos “Spikes Loop”. Bajo el **bloque** “Show” que colocamos anteriormente, ponemos el **bloque** “Broadcast [Spikes Loop]”.

Ahora, bajo el bloque “When I Recieve [Spikes Loop]”, ponemos un bloque de control “Repeat [40]”, y dentro del mismo ponemos nuestro bloque “Anim”, y el bloque variable “Change [Current Costume] by [0.1]”. Duplicamos este bloque “repeat” con su contenido y lo ponemos a continuación, solo que esta vez el valor del “Current Costume” será de -0.1. Después de este último “repeat”, ponemos el bloque control “Wait [1] Seconds” y el de evento “Broadcast [Spikes Loop]”.

Con esto hacemos que los pinchos salgan del suelo, y la repetimos en sentido inverso para que vuelvan a esconderse, que pase 1 segundo, y que entonces se repita el proceso. Como la animación dura 4 imágenes, y hemos hecho que vaya 10 veces más lenta, tenemos que hacer que la animación se actualice 40 veces, por eso hemos utilizado estos valores.

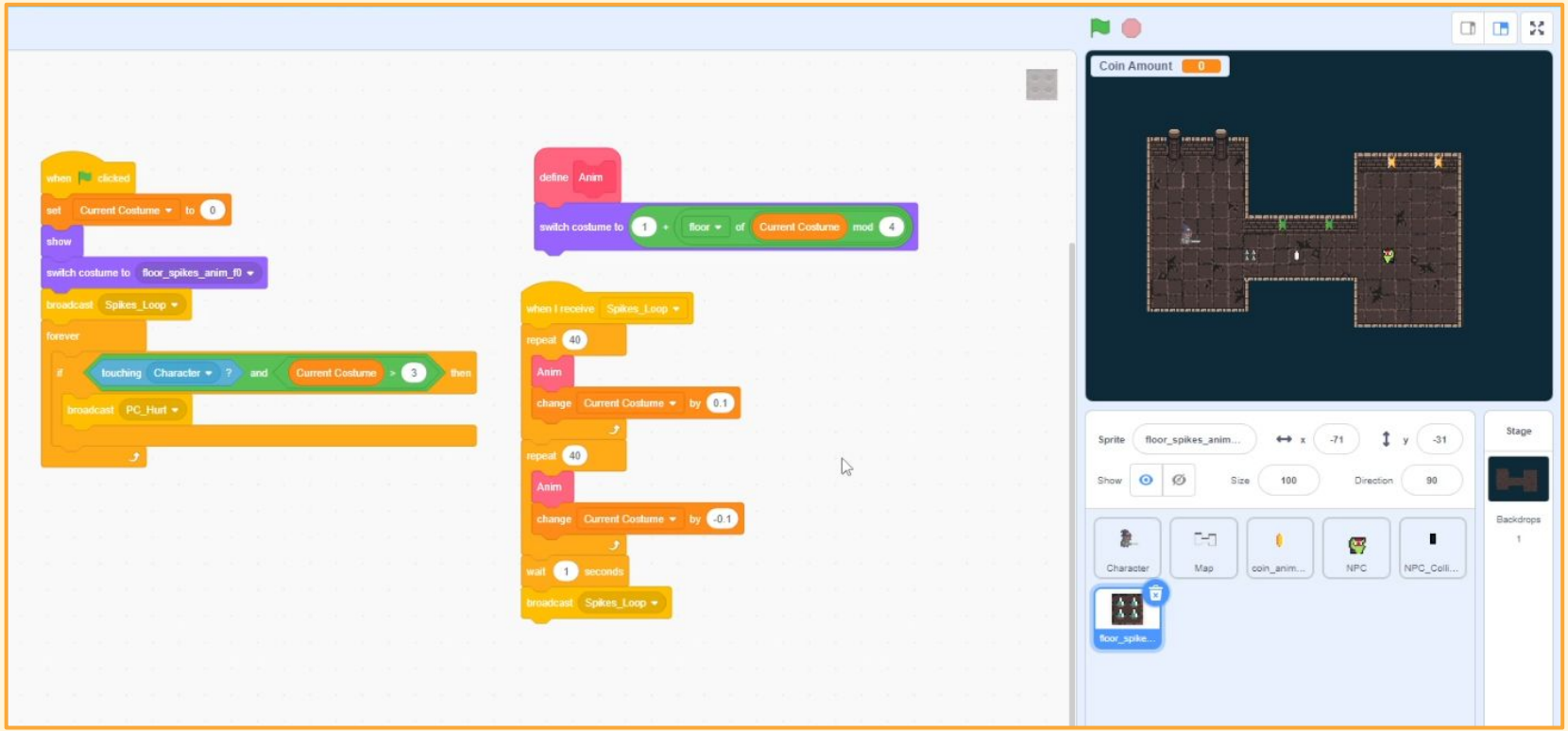
Ahora dentro del personaje creamos y nombramos un evento, “When I Recieve [PC_Hurt]”, y bajo él ponemos el bloque “Say [Ouch!] for [1] seconds”.

De nuevo en el código de los pinchos, al final del código que tenemos para inicializarlos, el de la banderita verde, ponemos un bloque “Forever”, y dentro un bloque “If [ ] then”. Para su condición vamos a usar el bloque “Touching [Character] ?”, la variable “Current Costume”, y los bloques “[ ] and [ ]” y “[ ] > [ ]”. Los colocaríamos de modo que quedasen así:

“[Touching [Character] ?] and [[Current Costume] > [3]]”

Dentro de este “if” colocamos ya el bloque de evento “Broadcast [PC_Hurt]”. Todo esto sirve para comprobar si el personaje está encima de la trampa mientras se ven los pinchos, y entonces hacer que el personaje se queje.

Contenido extra - trampas



Contenido extra - escaleras

Escaleras

Por último vamos a hacer que el personaje pueda moverse rápidamente de un lado a otro del mapa añadiendo un atajo. Importamos la imagen floor_ladder del Dungeon Tileset II como una sprite, la colocamos en el mapa, y la renombramos como Stairs_1.

Dentro del código de nuestro Character, creamos y nombramos un nuevo **evento**, “When I Recieve [Stairs_1]”, y le añadimos del **bloque motion** “Go to x [ ] y [ ]” con la posición a la que queremos que le lleven las escaleras. Justo después añadimos las **variables** “Set [X] to [ ]” y “Set [Y] to [ ]”, poniendo las coordenadas a las que llevamos el personaje.

Ahora en el código de las escaleras ponemos el **evento** “Whe Green Flag Clicked”, seguido de los **bloques de control** “Forever” e “If [ ] then”, cuya **comprobación** es “Touching [Character] ?”. Finalmente dentro pondremos del **bloque events** “Broadcast [Stairs_1]”.

Con esto estamos comprobando continuamente si las escaleras están tocando al personaje, y de hacerlo lo llevan a la posición que queremos.

Si queremos duplicarlas para hacer los dos extremos del atajo, podemos hacer click derecho sobre Stairs_1 en nuestra lista de sprites, y duplicarlas. En este caso tendríamos que crear otro bloque en nuestro Character, “When I Recieve [Stairs_2]”, añadiendo lo mismo que en el anterior, pero con una nueva posición.

Por supuesto, hemos de asegurarnos que estas nuevas escaleras hacen “Broadcast [Stairs_2]” para que funcionen correctamente. También es importante asegurarse de que el personaje no es teletransportado donde están las escaleras, sino a un sitio cercano, porque de ser así se quedaría en bucle entre las dos escaleras.

Con todo esto ya tenemos nuestro personaje totalmente funcional, en una mazmorra, con objetos, trampas y otros personajes con los que puede interactuar.

Glosario

Motor de juego: Es un programa formado por un conjunto de herramientas que nos permite diseñar, crear y representar un videojuego.

Lenguaje de programación: Es un tipo de lenguaje que permite que las personas nos comuniquemos con sistemas informáticos para que estos realicen las tareas que deseemos.

Sprite: Imágenes que aparecen por pantalla dentro de un videojuego que pueden ser creados a mano o computacionalmente. En Scratch las Sprites no son sólo el aspecto visual, sino que también contienen su propia programación.

Sprite sheet: Una imagen formada por distintas imágenes (sprites) representando una o varias animaciones. Se suele utilizar este formato para guardar las animaciones porque es mucho más eficiente para trabajar con ellas.

Assets: Todos aquellos recursos u objetos que se utilizan en un videojuego. Por ejemplo sprite, animaciones, música, sonido…

Fotograma: Se trata de una imagen fija. Entonces cuando hablamos de fotogramas en un vídeo o en un videojuego es porque estos están formados a partir de una secuencia de fotogramas que se reproducen a gran velocidad.

Eventos: Acciones que ocurren durante el juego, y que ejecutan instrucciones concretas de la programación.

Condiciones: Programación que permite comparar información o variables, y en función del resultado elegir si ejecutar unas instrucciones u otras.

Variables: Representación de aquello que varía o que está sujeto al cambio, cuya información queremos poder consultar en cualquier momento. La *x* de la posición del personaje del juego es un ejemplo de variable.

Colisiones: Parte de un asset que gestiona lo que ocurre cuando varios objetos ocupan una misma área de espacio o de píxeles (total o parcial).

Animación idle: Hace referencia al tipo de animaciones en las que el personaje no está realizando ningún tipo de acción, animación en reposo o standby.