TUTORIAL PRÁCTICAS

PARA PROFESORES
Y PRINCIPIANTES

Javier Fernández Panadero

versión 22 marzo 2018

Índice

¿Qué tenemos aquí?

Este es un manual de prácticas de Scratch que voy a usar con mis alumnos de secundaria y bachillerato.

Las ventajas que le veo a Scratch son:

- Podemos olvidarnos de la sintaxis que da muchos quebraderos de cabeza a los principiantes, y centrarnos en el flujo del programa
- Curva de aprendizaje muy rápida
- Posibilidad de hacer programas de cierta complejidad y aparente sofisticación desde el principio
- Posibilidad de usar multihilo y trabajar con funciones separadas que en otros lenguajes hay que ingeniarse para ir intercalando en un mismo hilo de difícil seguimiento a principiantes
- Es un estándar muy extendido que ha generado interesantes “hijos” como Mblock. También es muy parecido a Blockly y otros sistemas de bloques populares.

Desventajas

- No deja de ser un sistema de bloques y en la progresión de los estudiantes habrá que acabar pasando a código en algún momento

Las prácticas empezarán **sin suponer conocimiento previo**. En ese camino se irán introduciendo conceptos de programación y formas de resolver problemas.

No voy a eternizarme en cosas llenas de posibilidades, como por ejemplo, modificar la apariencia de los objetos o importar imágenes, algo que podéis explorar vosotros lo que queráis, me centraré más en abrir nuevos caminos y dejar que los desarrolléis por vuestra cuenta.

De igual forma, me centraré más en los aspectos de programación e interacción. En lugar de repetir varios ejemplos donde se use el mismo concepto. La unión de estas cosas y la mezcla en infinitas posibilidades será vuestra.

Para el principiante puede ser interesante empezar paso a paso, para los más avanzados o profes que estéis buscando material, quizá os sea útil ir leyendo en diagonal a ver si encontráis algo que os sirva.

A veces simplemente sugiero modificaciones a programas que acabamos de hacer.

El objetivo no es hacer un programa muy complejo sino daros herramientas elementales para que os podáis embarcar en esa tarea vosotros.

Si alguien tiene los recursos y las ganas de hacer cualquier aportación, por pequeña que os parezca, por este material, podéis hacerlo aquí:
https://www.paypal.me/javierfpanadero

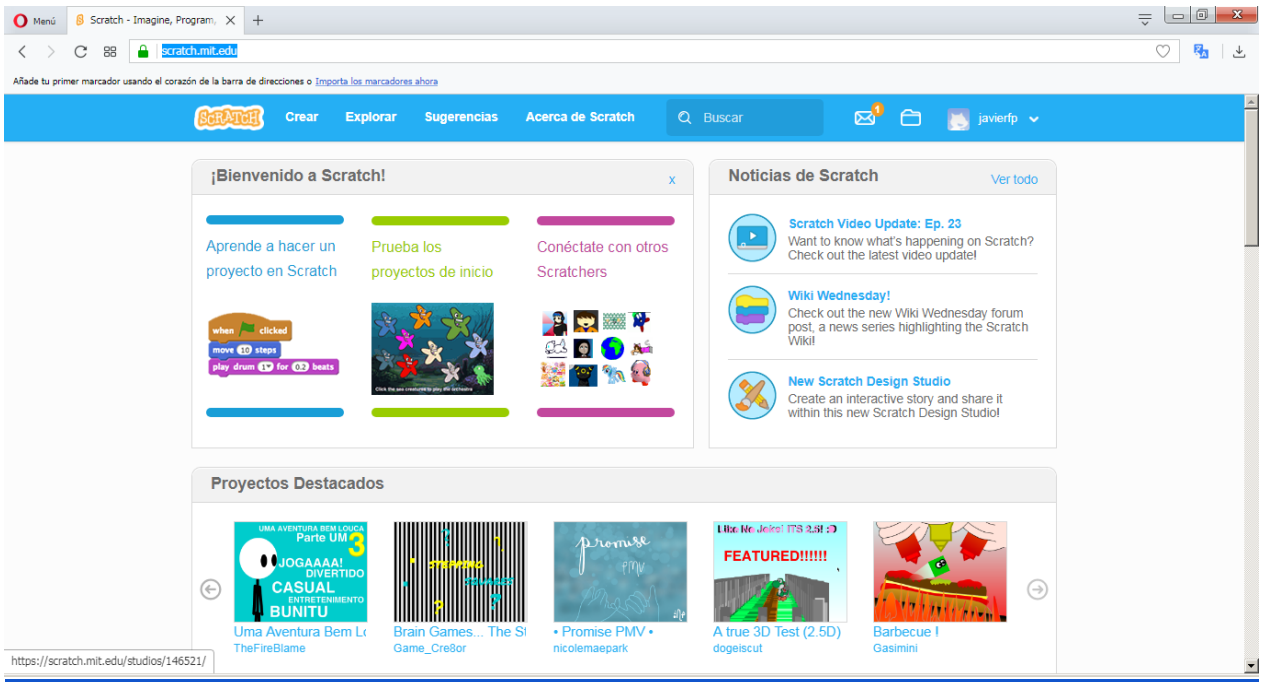
Juntos somos más.

Javier Fernández Panadero
Marzo 2018

¿Qué es Scratch?

Scratch es un entorno de programación gráfica para acercar la programación a jóvenes y principiantes.

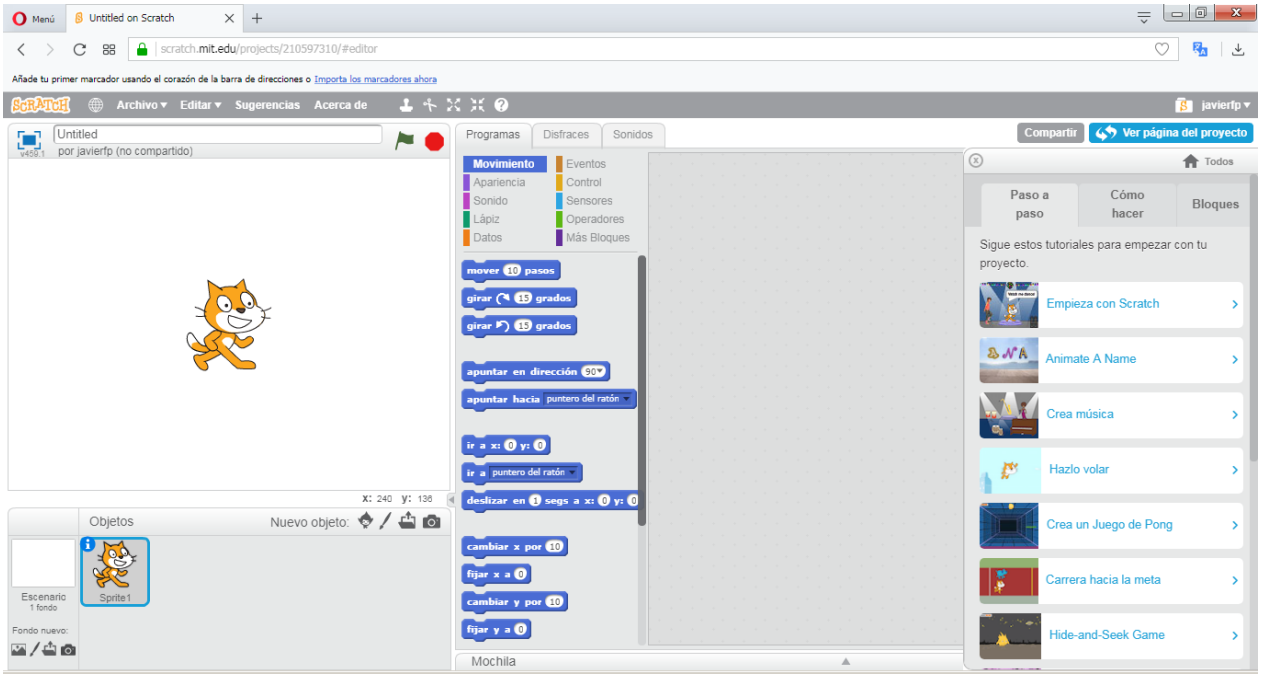
Es una iniciativa del MIT, aquí tenéis su web https://scratch.mit.edu



Tenéis una versión online (que os recomendaría, porque desde ahí podéis acceder a montones de proyectos compartidos por una gran comunidad, ayuda en línea y demás. **OJO: Necesita FLASH para funcionar.**

También os podéis descargar una versión offline.

Esta sería la pantalla de inicio de la versión online



A la izquierda podéis ver a un objeto, el gato, en un escenario donde podremos cargar más objetos, fondos y hacer que evolucionen.

En la parte central tenéis bloques disponibles para empezar a programar, están agrupados por categorías. Podéis pasar de una categoría a otra en el cuadro de arriba y veréis que el color de los bloques tiene que ver con la categoría a la que pertenecen.

En la parte gris de la derecha es donde soltaremos los bloques para crear los programas que queramos hacer.

A la derecha del todo os he dejado abierta la ayuda que tenéis en línea, que incluye ejemplos y pequeños tutoriales, por si queréis echar un ojo. Particularmente interesante por si tenéis dudas en el funcionamiento de algún bloque.

Primeros pasos… del gato.

Arrastrad este bloque a la parte gris y soltarlo. Está en el grupo MOVIMIENTO.

Como hemos dicho esta zona es donde ponemos el código (en bloques) de nuestros programas.

Haced click sobre él y observad al gato. Se mueve una pequeña distancia.

Ese número de pasos es editable, haced click en donde está el número y poned los pasos que queráis. Después click en el bloque y veréis que anda.

Estupendo.

Los dos siguientes bloques hacen girar al gato un número de grados en sentido horario y antihorario.

Sácalos y pruébalos. Fácil, ¿no?

Fíjate que los bloques tienen unos pequeños salientes arriba y abajo, eso es porque se pueden poner unos encima de otros y hacer un **hilo de programación.**

Habrás visto que se “iluminan” por la parte que pueden encajar.

Como trabajaremos esto mucho, te daré unas indicaciones para ir más rápido.

COMO USAR BLOQUES

- Si quieres mover todo el hilo, click y arrastra el primero
- Si haces click en uno de enmedio y arrastras, te llevarás desde ese hacia abajo
- Con botón derecho puedes duplicar o borrar bloques o hilos completos
- Para borrar bloques o hilos también puedes arrastrarlos y soltarlos en la parte central

No te preocupes cuando te falte espacio en la zona de programación, porque se extiende sola y aparecerán barras de desplazamiento para moverte.

Nota para profes: A veces cuando proyectamos esto no se ve muy bien desde los puestos de los alumnos, verás que abajo a la derecha tienes un zoom, pero si agrandas mucho no te caben bien los hilos, vaya un lío. Lo que suelo hacer yo es "esconder el escenario" (sólo offline) o "hacerlo más pequeño". En el menú EDITAR.

No me alargo más con esto… haced todas las gansadas que os apetezca con el gato y luego seguimos.

EVENTOS

Hasta ahora hemos hecho que el código se active cuando hacemos click sobre él, pero podemos hacer que el código se ejecute cuando ocurra algún **evento**, esto quiere decir que el programa está siempre atento, comprobando si algo ocurre (si has pulsado una tecla, por ejemplo), sin que tú tengas que escribir nada.

Sacad el grupo de bloques EVENTOS haciendo click aquí

Fíjate que estos bloques no tienen la posibilidad de que algo se les ponga encima, son un bloque inicial, pero debajo puedes poner los bloques que consideres.

Vamos a fijarnos de momento en los tres primeros:

1. Al presionar la bandera se refiere a la bandera verde que hay sobre el escenario, es el bloque de principio de programa que usaremos, típicamente.

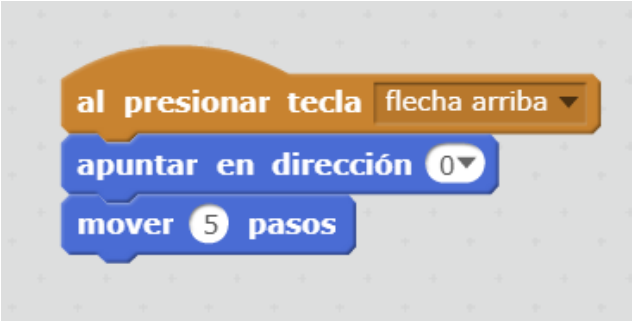
2. Al presionar la tecla espacio permite detectar la pulsación de una tecla en particular, o de cualquiera si quieres. Despliega el menú y verás.
3. “Al hacer click en este objeto” está pensado para programas donde actuéis con el ratón y asociar código a que hagas click en una puerta, por ejemplo.

CONTROLEMOS AL GATO

Con lo que hemos visto os propongo el primer ejercicio más serio…

Práctica 1. Mover gato con cursores ver.1

Quiero mover al gato usando las teclas de las flechas de los cursores.



Esto es un hilo de programación.

Scratch está mirando constantemente a ver si pulso la tecla “flecha arriba” y en cuanto lo haga pondrá al gato mirando hacia arriba y andará 5 pasos. Probadlo.

Duplicad este hilo y retocad para hacer los otros tres hilos que nos faltan para movernos en las cuatro direcciones.

Muy bien, facilito, ¿no? De hecho es tan fácil que os lo dejo sin poner ;)

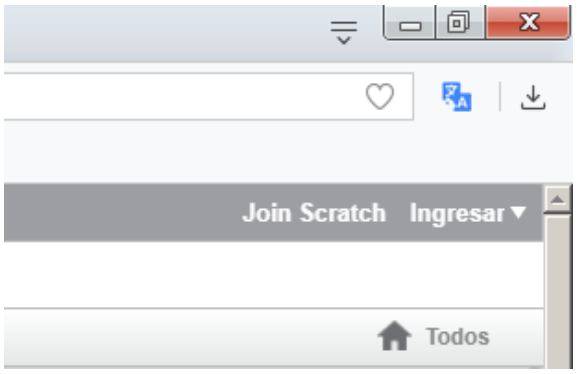
Esto que hemos hecho es programación **multihilo**, varios hilos que se ejecutan a la vez y que podrían incluso interactuar entre ellos.

VAMOS A REGISTRARNOS

Seguro que os apetece guardar vuestro programa, que os ha quedado estupendo.

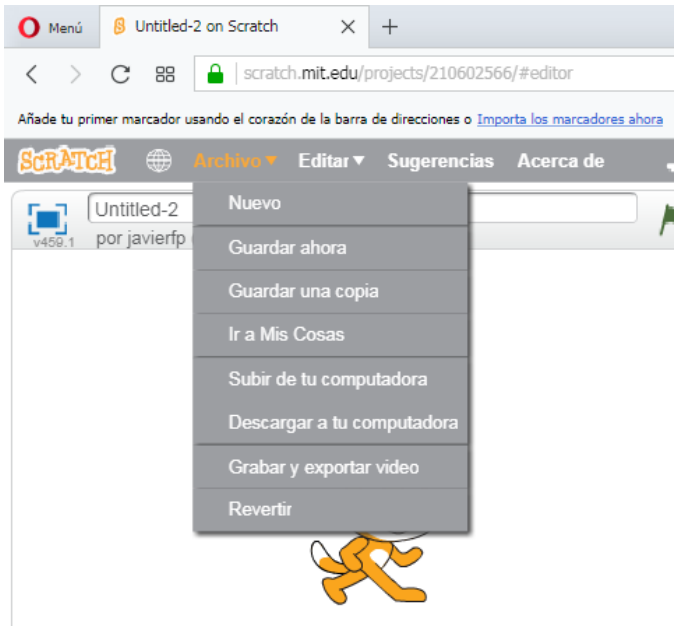
Bien, es el momento de registrarnos.

Arriba a la derecha



De esta forma podréis ir guardando vuestro proyectos, además de compartirlos con la comunidad.

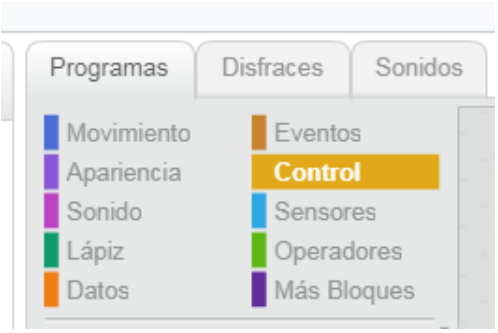
Una vez que estéis dentro, ya podréis guardar vuestros proyectos, compartirlos, editarlos, etc.



BUCLES Y CONDICIONALES

Son las estructuras básicas de control que usaremos mil veces. Son muy intuitivas, ya verás.

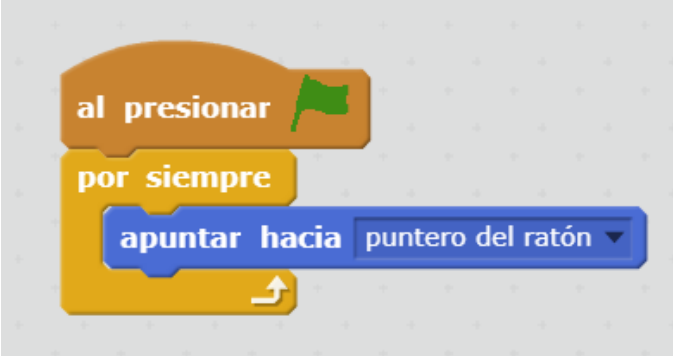
Lo primero es abrir el conjunto de bloques de control



Fijémonos en el bucle infinito

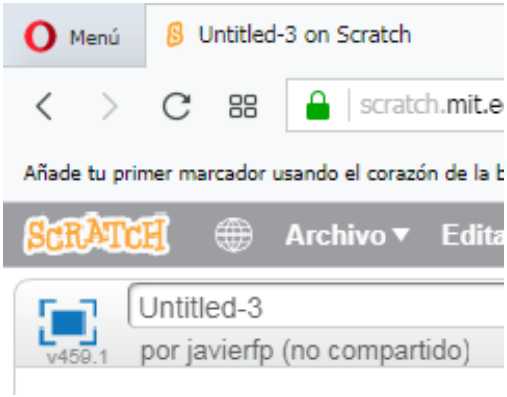
Este bucle pasa a repetir una y otra vez el hilo de bloques que pongamos dentro. No te preocupe el tamaño del hueco, que según quieras soltar cosas dentro se abrirá, o bien cuando muevas este bloque para “envolver” al hilo que quieras.

Práctica 2. El gato nos mira constantemente



Si te fijas, aunque podría haber activado el bucle infinito simplemente haciendo click sobre él, tenemos que hacer las cosas limpitas, por eso he puesto el disparador “al presionar la bandera”.

Además, así aprovecho para enseñarte otra cosa: **Cómo ejecutar los programas a pantalla completa**



Mira arriba a la izquierda de tu pantalla ese cuadro azul (en esta captura abajo a la izquierda).

Así puedes entrar (y luego con un icono similar salir) de pantalla completa.

Usa la bandera verde para comenzar el programa y la señal roja de stop que está al lado para pararlo. Terminará todos los hilos.

Cabe un pregunta importante, ¿por qué **NO VALE** hacer un programa sólo con esto?

Bueno, no hace falta que me creas.

Haz este código, ejecútalo y mira cómo funciona, ¿qué conclusiones sacas?

MUY IMPORTANTE:

> ***Lo mejor de la programación es que hace lo que dices.***
> ***Lo peor de la programación es que hace lo que dices.***

Recordad esta cita… y usadla mucho si sois profesores.

¿Por qué es lo mejor?
Porque los programas no se vuelven locos y hacen lo que les parece, independientemente de lo que yo les haya ordenado (saludos, Skynet)
¿Por qué es lo peor?
Porque no va a ayudarme adivinando qué es lo que quiero hacer, se lo tengo que decir explícitamente.

Una manera de aprender programación, es imaginarse que uno es el ordenador o el robot y seguir todas las instrucciones una a una, a ver cómo funcionaría. Para las clases es bueno hacer esto antes de ejecutar para ver si luego hace lo que esperábamos o no y por qué.

Este código dice: Cuando le dé a la bandera, que el gato apunte al ratón y ya.
UNA SOLA VEZ.

Y eso es justo lo que hace. Si yo quiero que lo haga una y otra vez, por siempre, se lo tengo que decir, con el bucle infinito.

Así que tranquilos, el programa hará lo que dices… pero, ¡díselo, hombre!

Si quiero que lo haga todo el rato, por siempre… como dijimos

Práctica 3. Mover al gato con cursores ver.2

Os propongo ahora que movamos al gato con los cursores pero usando bucles y condicionales.

Empecemos con el **condicional**… verás que también es fácil

Quiero que: Si pulso la tecla “flecha arriba” que el gato se mueva hacia arriba

Mirad el bloque que tenemos en el apartado CONTROL

Tengo dos huecos donde poner cosas, uno con forma de hexágono y el típico hueco que ya conocemos para poner código.

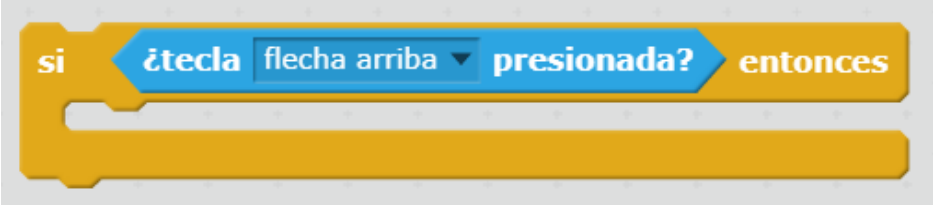
En el hueco de arriba voy a poner la condición que quiero valorar (a ver si se cumple o no). Intentad poner ahí un bloque de los de movimiento que conocemos y veréis que no entra, que no os deja. Esto es fantástico, nos ahorrará muchos errores. Scratch no nos deja hacer cosas que no funcionan.

NOTA: Hay una diferencia entre que Scratch no me deje poner código incorrecto (poner bloques donde no es) a que mis programas funcionen como yo quiera.
Por ejemplo. Si yo quiero la sal y digo la frase “Pásame el azúcar”, la frase es correcta desde el punto de vista del lenguaje (no he dicho “Pásame cantar”, que no tiene sentido), pero no conseguiré el objetivo que quería.

Hay varios tipos de bloques hexagonales y todos serán del mismo tipo **booleanos**. No os asustéis, eso quiere decir que el bloque, como conjunto, puede tomar sólo dos valores: verdadero o falso, será una pregunta que se contesta con sí o no.

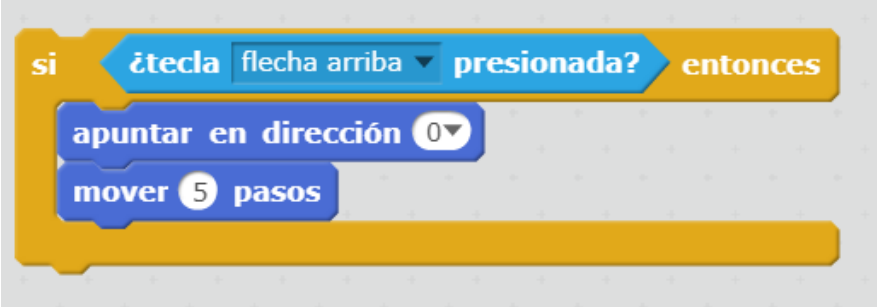
Id a apartado SENSORES y veréis que hay varios con esa forma hexagonal y todos hacen preguntas que se contestan con sí o no. A mí me interesa ahora que saquéis este

Metedlo en el condicional. Lo acercáis hasta que se "ilumine" el agujerito hexagonal y ahí lo soltáis, veréis que se adapta al tamaño. Tened cuidado que si lo dejáis sólo encima no funcionará. Elegid también la tecla que queráis que sea evaluada.



Casi está. Lo que hace el bloque es preguntarse: Si es verdad que la tecla "flecha arriba" está presionada, entonces, ¿qué quiero hacer?

En nuestro caso, movernos hacia arriba. Pues ponemos el código dentro y listo.



Probad el código y ved si funciona.

Perdonadme la maldad… pero no funciona, ¿¿verdad??

Vaya, sí que funciona, si le ponéis la bandera y tenéis la tecla apretada puede que notéis algo, pero… no es lo que queríais.

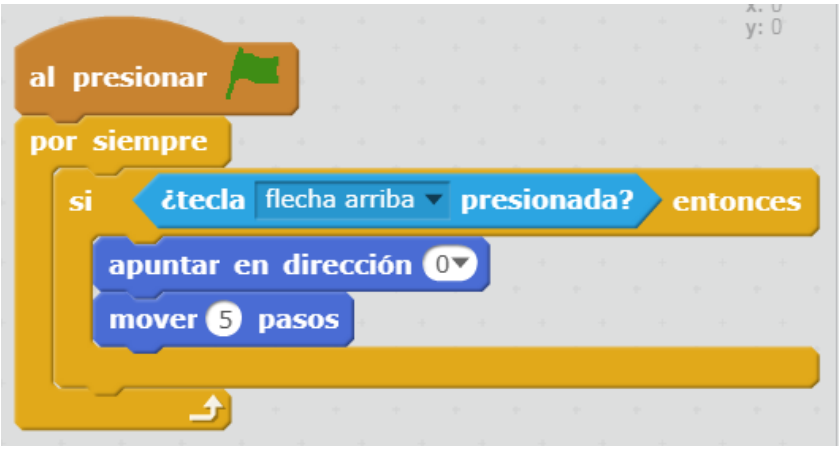
Recordad que lo hemos hablado. El código no es incorrecto, pero no hace lo que tú deseas… y no lo hace, porque NO SE LO HAS DICHO.

¿Quieres que esté SIEMPRE mirando si estás pulsando la tecla?

¿SE LO HAS DICHO?

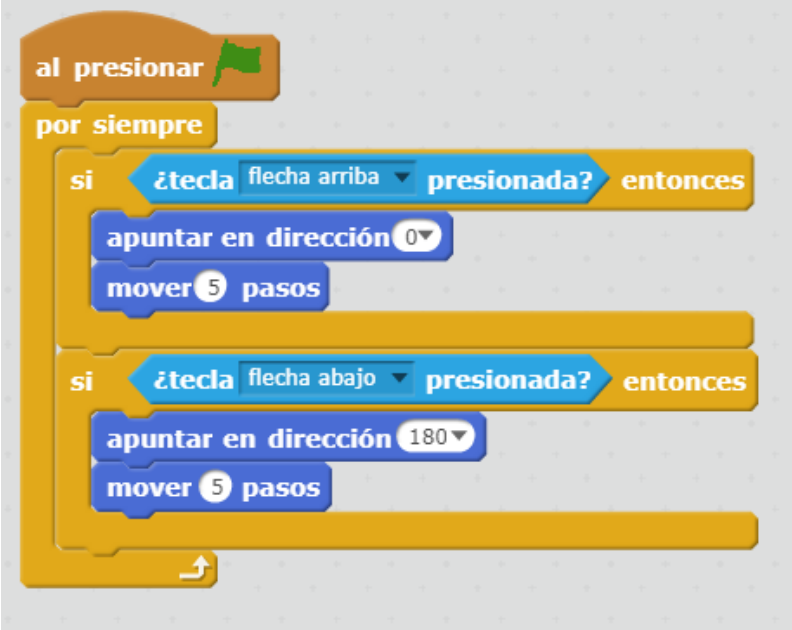
Recuerda: *EL CÓDIGO SÓLO HACE LO QUE LE DICES*

A ver así



Ahora sí, le has dicho que mire una y otra vez y que si está pulsada la tecla que se ponga apuntando hacia arriba y que ande un poquito. Y eso es justo lo que hará.
Es ULTRAOBEDIENTE.
Y esa es la única manera de tener el control.

Así que bien puesto quedaría así:



Te pongo sólo dos, pon tú los otros dos condicionales para que también pueda moverse a izquierda y derecha.

Este programa es un solo hilo. Una y otra vez mira los cuatro condicionales y en el caso de que se cumpla alguno, ejecuta el código que le has puesto dentro a cada uno.

Cada una de esas vueltas del bucle se llaman **iteración.**

Si comparáis este programa con el de antes, veréis que funciona mejor, que cuando pulso arriba y derecha a la vez, el otro se lía un poco, pero este se mueve en diagonal de forma bastante suave y amigable.

MIREMOS ALREDEDOR

Hemos ido un poco a lo loco, vamos a echar un ojo alrededor.

Por ejemplo:

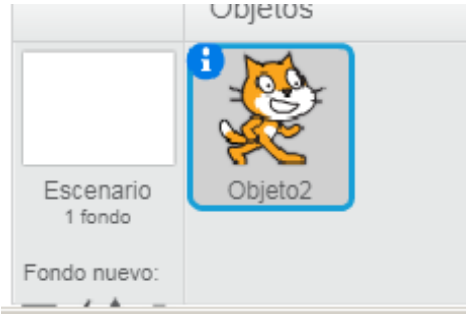
¿Qué tamaño tiene el escenario?
Haz click sobre el gato y arrástralo por el escenario verás que justo abajo a la derecha del escenario salen las coordenadas x e y en las que te encuentras.
El centro es x = 0, y = 0
Los extremos en x son -240 y 240
Los extremos en y son -180 y 180. Medidos ambos en “pasitos”.

¿Dónde están los objetos?
Escoge un momento el apartado de bloques MOVIMIENTO.
Ahora arrastra el gato a alguna posición y suéltalo.
Mira el bloque que sirve para desplazarte a un punto concreto.

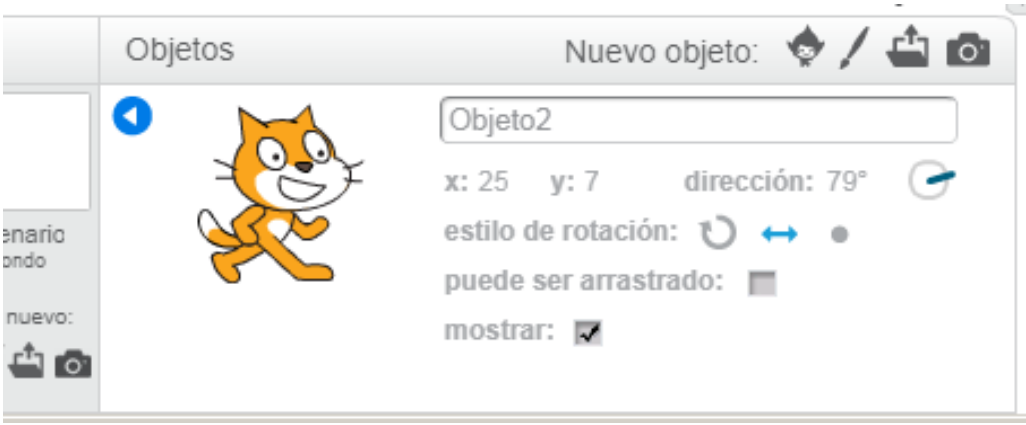
Él sólo ha seleccionado por defecto las coordenadas del punto en el que has dejado el gato. Eso nos será útil en el futuro.

Variar características de los objetos
Ya hemos visto que un objeto puede ejecutar un código (lo que en programación orientada a objetos se llaman **métodos**), pero también pueden cambiarse sus propiedades (**atributos**). Para que entiendas la diferencia, si tu coche fuera un objeto, su color, o su número de puertas serían atributos, y arrancar sería un método, una “función” que puede hacer.

Mira en la parte inferior derecha donde están listados todos los objetos que tenemos (hasta ahora sólo uno)



Pulsa en la “i”

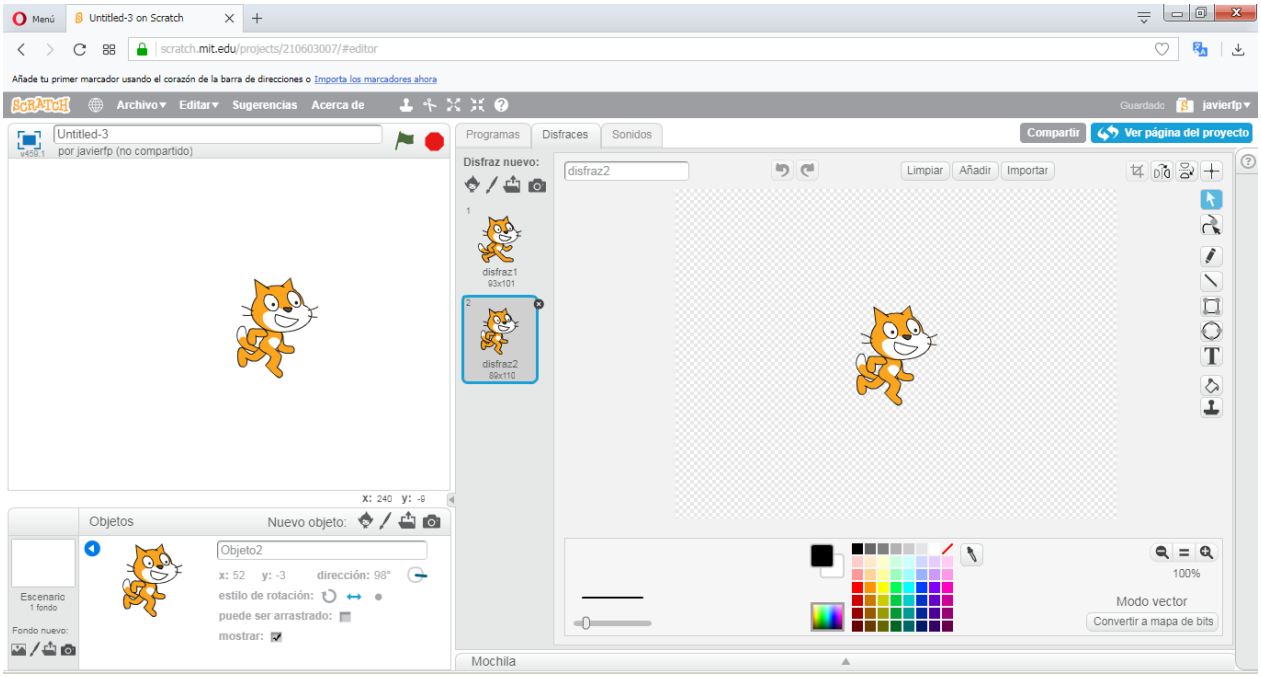


Ahí tienes sus atributos, algunos puedes cambiarlos directamente ahí, aunque también puedes modificarlos durante el programa con los bloques correspondientes.

- El nombre del objeto (ponle uno chulo, ya que estamos)
- Ves su posición en el escenario (aunque no te deja cambiarla ahí, tienes que mover el gato con el ratón)
- La dirección en la que mira, 79º (eso sí puedes cambiarlo ahí)
- El estilo de rotación es para que, por ejemplo, si queremos que cuando vaya hacia abajo no se ponga cabeza abajo, pues le ponemos el segundo estilo, en el que sólo cambia izquierda y derecha, o el último, en el que se queda en la misma orientación del cuerpo. Pruébalos.
- Mostrar u ocultar un objeto será un atributo que usaremos en tiempo de ejecución para que las cosas aparezcan o desaparezcan a nuestra conveniencia.

Disfraces

Id arriba al centro y veréis que hay tres pestañas: PROGRAMAS, DISFRACES Y SONIDOS. Elegid disfraces.



Aquí me vais a disculpar que no me entretenga y os muestre sólo la cantidad de cosas que podéis hacer.

Lo primero que vemos es que el objeto tiene dos disfraces. Luego veremos como cambiar de uno a otro y simular que “anda”.

Las herramientas de dibujo son las habituales, usar formas sencillas, dar color a líneas, colorear formas cerradas, copiar, poner unas cosas encima de otras, etc. Si dejáis el ratón sobre el icono os saldrá un texto de ayuda.

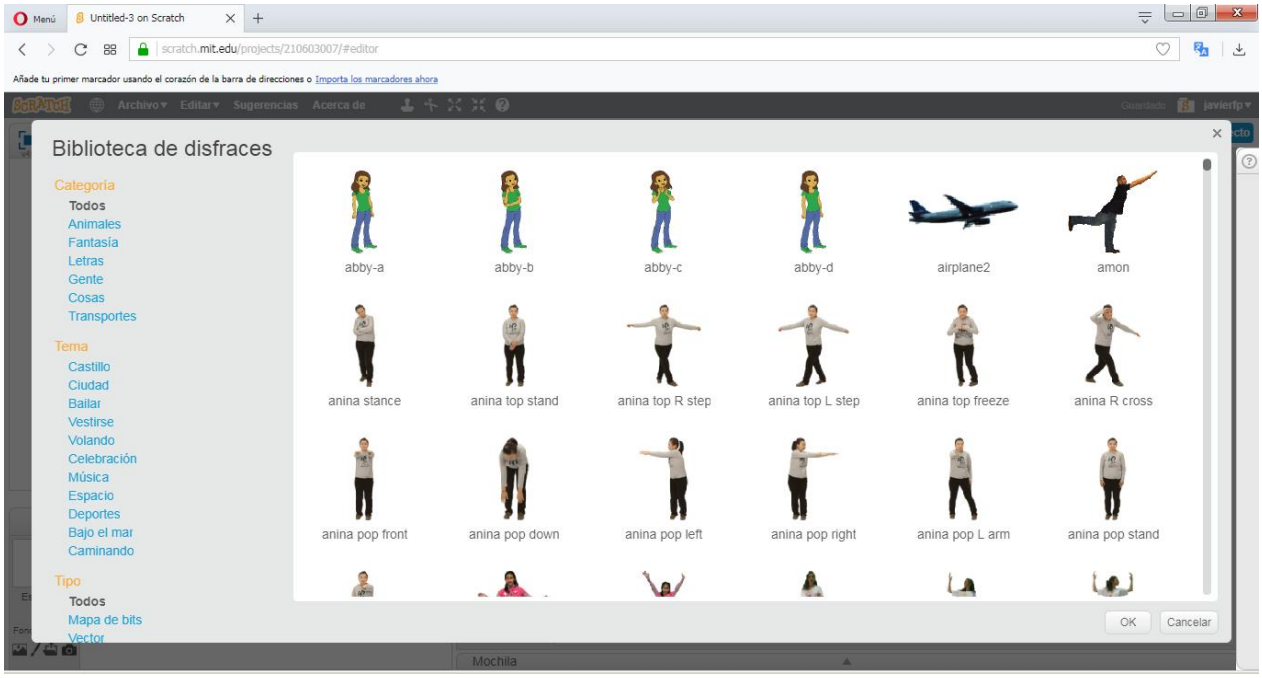
Os diré también que podéis importar otras imágenes que tengáis y retocarlas. De forma, que sí, **podéis poner vuestra cara** a algún bicho y **que se mueva por vuestra casa** y demás. Posibilidades infinitas.

Fijaos que en la parte central arriba, donde pone disfraz nuevo, existe incluso la posibilidad de tomar una imagen con la cámara en ese mismo momento.

Quizá a algunos os interese ver que en la parte inferior derecha podéis cambiar de dibujo vectorial a mapa de bits para usar las posibilidades que tiene una y otra forma de edición.

Por supuesto el número de disfraces no tiene por qué ser dos.

Donde pone disfraz nuevo, arriba en el centro, veis que además de dibujarlo vosotros, cogerlo de una carpeta, hacer una foto… también tenéis la opción de darle a una "cabecita" y poder elegir entre todos los disfraces que ya tienen todos los objetos que trae ya Scratch, aunque nosotros sólo hayamos usado el gato.



Veis que abby tiene cuatro, pero la tal anina ¡tiene 13!

Resumo.

Podéis editar los disfraces que tiene tu objeto, puedes dibujar nuevos, puedes hacerte una foto, puedes incluir un archivo, puedes poner como disfraces los disfraces que tienen otros objetos… un universo.

Sonidos

Pues sí, fíjate que también hay una pestaña de sonidos…
Nuestro gato sólo tiene uno, un maullido, pero también puedes subir archivos de sonido o ¡¡incluso grabarte tú!!

De nuevo no me alargaré en las posibilidades de edición de este aspecto, pero cuenta con él para tus programas.

Fondos

Otro objeto que podemos usar son los fondos para poner en el escenario:

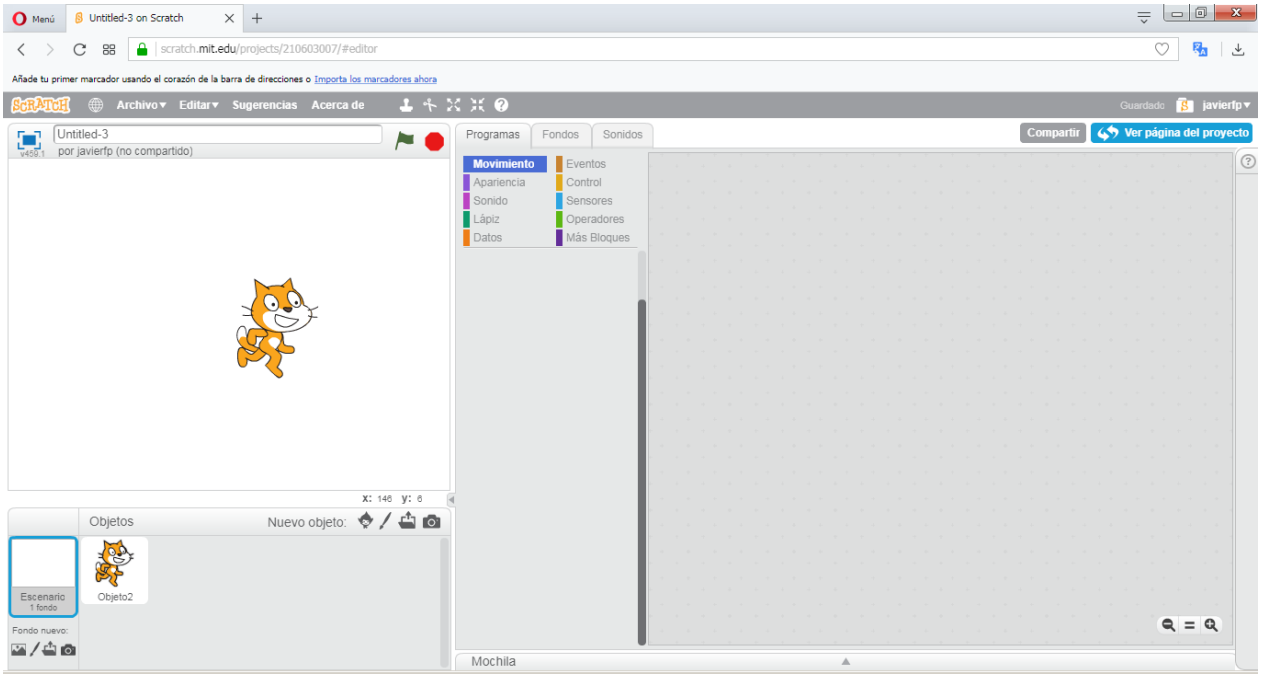
- Por decorar
- Por ilustrar distintos estados de la aplicación: comienzo, fin, etc.
- Para pasar de un sitio a otro, en un juego de plataformas, p.ej.
- Y mil cosas más… como siempre.

El objeto escenario está aquí al lado del gato.

El escenario es el espacio que tenemos ahora en blanco, y puede “vestirse” con distintos fondos que sería el equivalente de los disfraces al objeto “gato”.

Haz click en él y observa qué le pasa a la pantalla.

Ha quedado así, ¡vaya!



¡¡Ha desaparecido el programa que habíamos hecho!!

Si le dais a las pestañas de arriba: PROGRAMAS, FONDOS Y SONIDOS…

No tenemos el maullido del gato, tenemos un sonido como de una gota, no tenemos el programa del gato y en lugar de la pestaña disfraces nos sale esta de “fondos”.

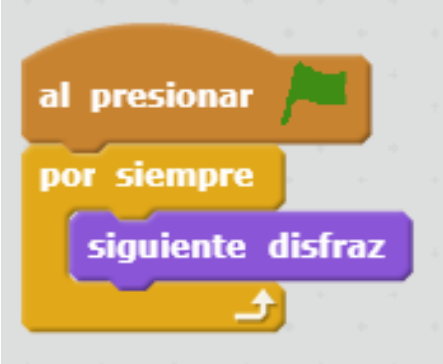
¡¡Pues, claro!!: **¡¡¡¡El escenario es otro objeto!!!!**

No tiene los sonidos del gato porque esos eran **atributos** del gato.
No tiene los disfraces del gato porque eran **atributos** del gato también.
No tiene el programa del gato porque esos eran **métodos** del gato. Así de simple.

Pero eres completamente libre de crear, importar, elegir los fondos que te gusten (vienen incluidos unos cuantos curiosos), editarlos, añadirles sonido y programar ese escenario para que sus fondos cambien como a ti te parezca.

Práctica 4. Cambio de disfraz

Sólo por jugar, añadamos un hilo a nuestro programa de movimento (práctica 3) un hilo para que nuestro gato esté por siempre cambiando de disfraz y parezca que vaya andando.



Ejecútalo… y verás que al pobre le ha dado *el tembleque*.

No es la única forma de hacerlo, podría haber usado el bloque

Y este es el bloque que usaréis cuando queráis hacer un cambio más concreto. Por ejemplo, si mi objeto se “muere” en un juego, pues cambiaré al disfraz que le he hecho tirado con equis en los ojos y lo seleccionaré específicamente.

Aún así, hay varios objetos que tienen un par de disfraces y están pensados para ir de uno a otro, por ejemplo, hay un simpático murciélago que parece que bate las alas.

Pero no creas que me he olvidado de que el gato no se mueve correctamente, porque cambia demasiado rápido de disfraz, tan rápido como se pasa la ejecución de un bloque a otro, de hecho en un lenguaje de programación más eficiente, sería tan rápido que ni se vería.

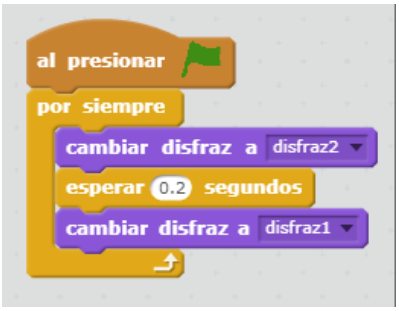
Pero no hay problema, en el grupo de bloques de CONTROL vas a encontrar la manera de hacer que esté un tiempo en cada disfraz.

Aquí puedes ver dos maneras de hacerlo. Quizá te sorprenda que sólo tenga una espera en un caso y dos en el otro. Como siempre te aconsejo que leas las órdenes como si tú fueras el gato y las ejecutes, sonriendo por ejemplo. Disfraz1 =serio, Disfraz2= sonriendo.

Código de la izquierda:
Aprieto la bandera
Repetiré una y otra vez
Paso a sonreír
Espero 0,2 segundos
Ahora toca volver arriba (otra iteración)
Paso a estar serio
Espero 0,2 segundos…

Repetiré una y otra vez
Paso a sonreír
Espero 0,2 segundos
Paso a estar serio
Espero 0,2 segundos
Vuelvo arriba a repetir
Vuelvo a sonreír…

Código de la derecha:
Aprieto la bandera
En ambos estoy 0,2 segundos en cada disfraz, aunque en el de la derecha podría escoger que estuviera tiempos distintos en cada uno.



Ahora imagina qué pasaría (y pruébalo) si quitas una de las esperas del segundo código.

Sonrío, espero, me pongo serio e inmediatamente subo arriba a repetir y me cambio a sonreír. No estoy nada de tiempo en el disfraz1!

Recuerda esto que te puede jugar malas pasadas en intermitentes y cosas parecidas.

MÁS OBJETOS

Práctica 5. Un objeto persigue a otro

Llevamos mucho tiempo con el gato… pongamos a alguien que le persiga.

Yo voy a probar con un murciélago que me mola.

Click en nuevo objeto… y lo eliges entre los que hay (o lo creas, ya sabes).

No tiene el pobre un sonido divertido, pero sí que tiene dos disfraces que me permite hacer el mismo juego que con el gato y que parezca que aletea.

TRUCO: Como queremos poner el mismo código de cambio de disfraz que hicimos con el gato una manera de copiar el código de un objeto a otro es la siguiente. Haces click en el objeto gato, y ahora arrastras su código hasta dejarlo “caer” sobre el murciélago en la lista de objetos (no en el escenario). Verás que el código no se pierde en el gato, y si ahora seleccionas el murciélago y miras sus PROGRAMAS verás que está ahí lo que copiaste.

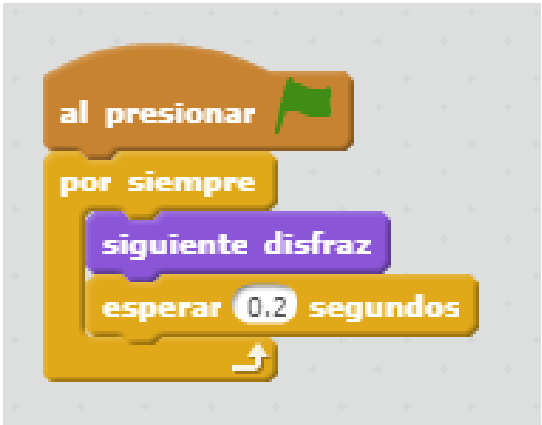
Este sería el código del gato

Como ves es idéntico al que hicimos antes… salvo un refinamiento que hemos hecho al principio. Es lo que llamamos **inicializar**.

Al principio las cosas quiero que estén de cierta manera: los objetos en ciertas posiciones, las puntuaciones a cero (si es un juego), las vidas al máximo, los tiempos a cero y un largo etcétera.

A veces esas cosas, por casualidad, están donde deben, pero así nos aseguramos de que todo irá bien.

Yo he puesto al gato en el medio mirando a la derecha.

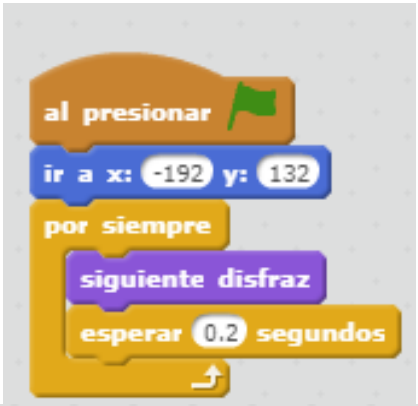


Y este hilo para que mueva los pies.

Si te fijas, podemos disparar con el mismo evento (pulsar la bandera) varios hilos de programación.

De hecho será lo más normal que al activar la bandera se pongan en marcha un montón de hilos en varios objetos.

Y este sería el código del murciélago



Este es el hilo que me lo posiciona al empezar en la esquina superior izquierda (para que no me pille inmediatamente. Recuerda que el gato está en el centro).

Y me hace aletear.
Recuerda que este código lo pegué y retoqué del gato para ahorrarme trabajo.

He usado el bloque "apuntar hacia" que cuando hay varios objetos me propone en el menú que se despliega: "apuntar al ratón" o a cada uno de los objetos que tenga.
Avanza un poco, espera un pelín y repite.

¿Qué es la "nota amarilla"?

Es fácil de adivinar que se trata de un **comentario** que puedes añadir donde quieras con tan sólo hacer click con el botón derecho.

El comentario se puede desplegar o reducir para que no estorbe.

Es muy importante comentar el código.

Ojo que no he dicho, "es muy divertido comentar el código", "no se pierde nada de tiempo comentando el código", "a todo el mundo le encanta comentar el código". He dicho que es muy importante.

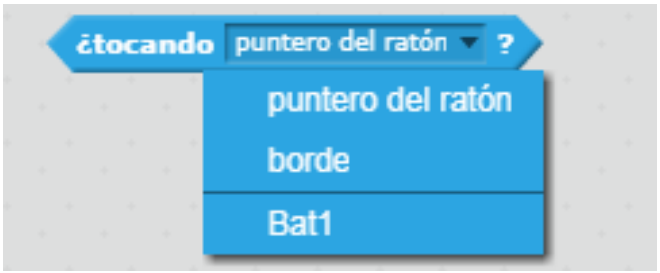
Si quieres que otros puedan entender tus programas, comenta el código.**Y te recuerdo que "otro" también eres tú dentro de tres semanas porque ya no te acordarás** de

por qué pusiste ese bloque ahí, o cómo se llama la variable que has usado para esto otro…

Así que te recomendamos que comentes profusamente el código y que observes lo mucho que lo agradeces cuando estés viendo el código de otros para aprender o por trabajo. Dicho queda.

Si quieres añadirle una floritura al asunto, el gato podría maullar cuando le pillara el murciélago.

Para eso en los bloques SENSORES encontrarás este, que verás que nos ayudará mucho en el futuro para saber si el ratón anda por encima de ese objeto, si ese objeto está tocando a otro (lo que nosotros buscábamos) o si está tocando un borde (quizá puedes querer que dé la vuelta o perder un punto o cambiar a la pantalla siguiente…)



Te recuerdo también que este es un bloque de “sí o no”, booleano, que podemos poner en el hueco de la condición de un “condicional.

Vamos a ello.

Quedaría así (lo pongo en el código del gato):

Si lo ejecutas verás que cuando te pila no deja de maullar. Si eso te parece mal, pon una espera después del maullido y te dará tiempo a escapar antes de que vuelva a hacerlo.

Si usas “tocar sonido y esperar”, no espera a que te vayas… espera a que acabe el sonido que hayas elegido. Según lo que estés programando te puede interesar o no. Quizá aquí nos vengan bien ambas cosas, que haga el maullido completo y que espere un momentín a que me pueda escapar.

VARIABLES

Vamos a refinar este programa poniendo vidas.
Muchos de los programas se irán complicando así: oye, ¿y si le hacemos tal y cual?
De hecho, estoy casi seguro que tú los estás haciendo con un escenario puesto…

Para las vidas acudiremos a lo que llamamos **variables.**

Una variable es como una caja, un lugar donde voy a guardar información. En este caso las vidas que nos van quedando.

Importante: Una cosa es el nombre de la variable: vidas; y otra es el valor que tiene esa variable: por ejemplo 3.

Para crear una variable, tienes que ir al bloque DATOS y darle a “Crear una variable”

Ponle el nombre que quieras y dile que vamos a poder usarla en todos los objetos, esto es lo que se llama una **variable global,** frente a una variable que sólo se usa en un objeto o una función que llamaríamos **variable local**.

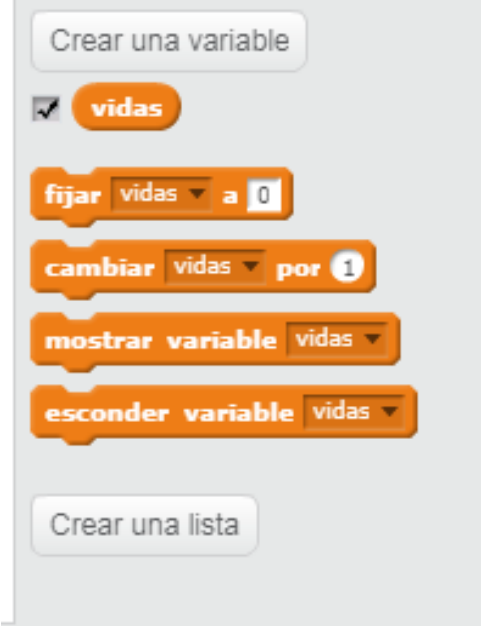
> **NOTA**: Estos comentarios sobre hilos, orientación a objetos, etc. son para que los profesores vean cómo con este entorno tan sencillo estamos tocando temas profundos y para ir preparando a los aprendices para el futuro.

Si quieres ir cogiendo costumbres de buen programador, sería interesante que no pusieras nombre a las variables a lo loco. Scratch es muy amable y te va a tolerar mayúsculas y minúsculas, tildes, palabras separadas y tal, pero mejor hacer las cosas como dios manda.

- **Muy importante:** usa **un nombre que indique con claridad qué representa** esa variable, por ejemplo: puntos, vidas, password, etc. No la llames x, t, z…
- Usa minúsculas, no uses caracteres no internacionales, como letras con tildes, la ñ y demás. Aquí no pasa nada, pero en el futuro te la puede liar
- No separes palabras, para esto puedes usar dos maneras muy populares
    a. estaEsMiVariable (A esto se le llama camelcase -estilo de camello)
    b. Esta_es_mi_variable (Usando guiones bajos)
- También es práctica común empezar los nombres con minúscula

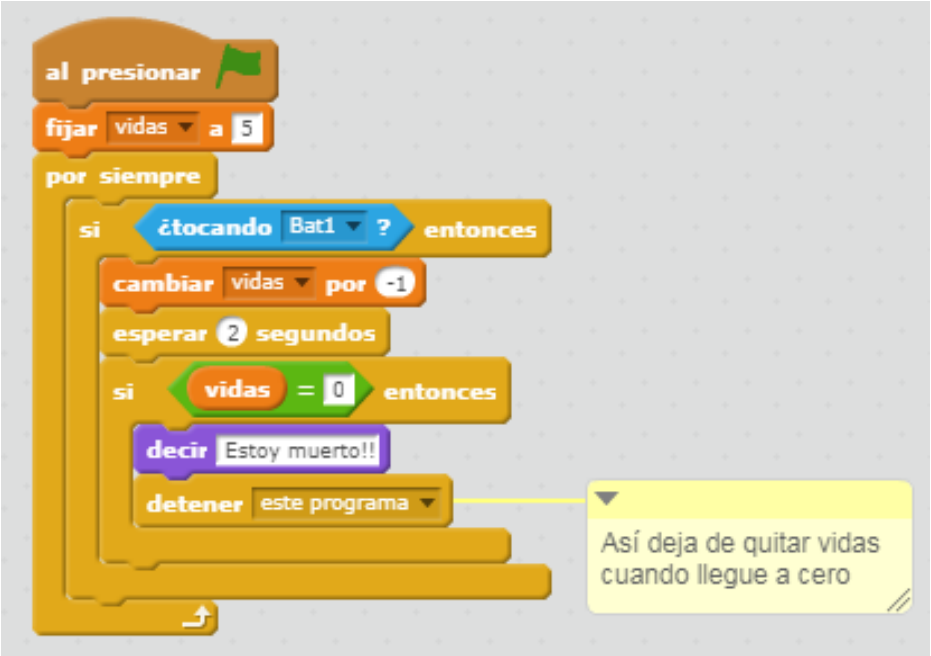
Práctica 6. Murciélago persigue al gato y le quita vidas.

Creemos la variable vidas. Una vez creada te salen un montón de bloques nuevos



El bloque fijar te permite darle un valor a la variable
El bloque cambiar le sumará el número que pongas, puede ser negativo, si lo que quieres es restar.
Los bloques mostrar y esconder sirven para que el cartel con el valor de la variable, que verás que ha aparecido solo en el escenario, se oculte o se muestre durante el programa.
Si lo quieres dejar siempre puesto o quitado bastará con que marques la casilla de verificación que al principio, por defecto ya ves que la muestra.
El bloque de bordes redondeados con el nombre de la variable representa precisamente el valor de esa variable.

NOTA: Aprovecho para recordarte que haciendo click con el botón derecho sobre cualquier bloque te permite llamar a la ayuda y te mostrará una explicación muy clara y un ejemplo sencillo. No dudes en usarla siempre que lo necesites.



Y ahí va el código para llevar la cuenta de las vidas

Al principio pongo la variable vidas al valor 5.
Después si el gato toca al murciélago le resto una vida (le sumo -1).
Espero un momento, para escapar.

Uso un condicional para ver si el valor de la variable vidas es cero. En ese caso el gato dice que está muerto y detiene este hilo de programación, el de contar vidas.
Si pruebas el código verás que puedo seguir moviendo el gato y que el murciélago me sigue persiguiendo, pero las vidas no bajan de cero.

Ese bloque (detener) puede usarse (y lo usaremos!!) para parar uno o varios hilos de programación cuando nos interese. Está en el grupo de bloques CONTROL.

Como ves puede detener todos los hilos de todos los objetos que hay en tu programa, o bien el hilo en el que esté (como en nuestro caso) o bien el resto de hilos de ese objeto. Esto último lo haremos por ejemplo si en un juego pasamos a una fase final y decimos: oye, STOP a todos los hilos, de movimiento, de “aleteo”, los puntos… etc.

Fíjate en algo curioso, este bloque por debajo no permite que le pongas nada.

Muy lógico, si detiene este hilo como poco, ¿qué orden quieres poner debajo? Porque no se van a ejecutar.

Esto es lo que os decía de Scratch no te deja hacer cosas que darían errores de programación. Cuando aprendáis otros lenguajes, veréis que no todos son así...

Para los más valientes os dejo como ampliación:

EXTRA:
Haz que si las vidas son cero el murciélago desaparezca

- Poniendo un condicional en el código del murciélago con el bloque esconder (está en APARIENCIA)

Pon un fondo que diga GAME OVER

- Pon un condicional en el código del escenario y cambia al fondo GAME OVER (En el editor de fondos puedes usar letras, si quieres escribirlo tú mismo)

Práctica 7. El tiempo en tus manos ver.1

Con estos dos bloques podemos acceder al tiempo y tomar decisiones.

Tendremos cuidado de reiniciar el cronómetro en cada ejecución (como cuando inicializábamos variables o posiciones) con el primer bloque.
Con el segundo bloque accedemos al valor del tiempo (en segundos) desde el reinicio.

Por ejemplo, podemos añadirlo a nuestro juego del gato para que cuando se acaben las vidas muestre el tiempo que hemos conseguido mantenernos vivos.

Podría ser algo así

Mira cómo escondo el tiempo para que no sepas cuánto has hecho hasta el final.

Reinicio el cronómetro para tener una lectura correcta del tiempo.

Ejecuto un bucle infinito que termino cuando se da la condiciòn que yo quiero (vidas=0).

Guardo en ese momento el tiempo en la variable tiempo y lo muestro.

Detengo el programa.

Esto de detener los programas así a lo bruto tiene su interés como todo en tecnología, puede ser más o menos apropiado a nuestros intereses.

¿Cuándo nos va a interesar más?
Cuando tengamos que detener abruptamente nuestro hilo
Cuando tengamos que detener más hilos en este objeto u otros (típicamente cuando pasemos de una “situación” a otra, como fin de partida, por ejemplo)
Si es algo más sencillo, un bucle que tenga que ejecutarse hasta que se cumpla una condición, o una situación que tenga que esperar a que se cumpla una condición, podemos usar otros bloques más sencillos.

Bucles con condición de paro

Con el primer bloque dejaremos parada la ejecución de este hilo hasta que se cumpla la condición. Scratch se encarga de comprobarlo sin que tengamos que hacer nada.
Con el segundo bloque se repetirán una y otra vez los bloques de dentro y en cada vuelta se evaluará la condición. Cuando se cumpla, se sale del bucle y se sigue con el hilo.

En nuestro caso, ¿cuál te parece más apropiado?

A mí me gusta mucho más el primero, aunque ambos arrojan el mismo resultado.

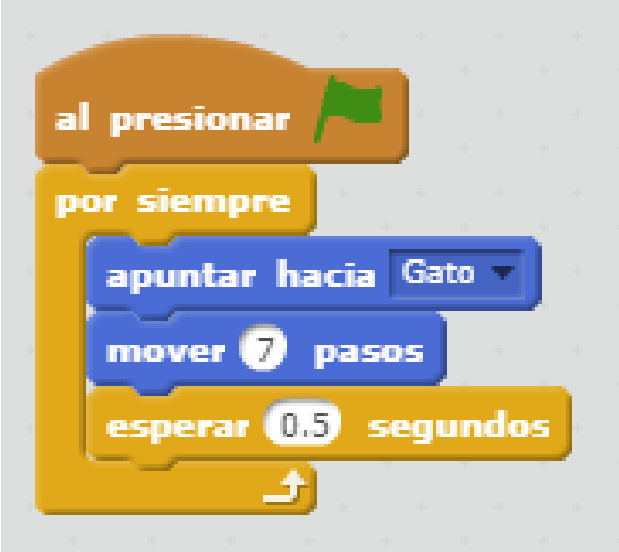
Fíjate que en el segundo se guarda el valor del cronómetro una y otra vez en la variable, un montón de acciones que son irrelevantes y costosas para el programa que queríamos hacer.

Cuando programamos no solo queremos que el resultado sea correcto, sino que sea eficiente, claro, sencillo… todo eso hace un buen programa.

Práctica 8. El tiempo en tus manos ver.B

Os propongo ahora que el comportamiento del murciélago vaya cambiando a lo largo del tiempo. Por ejemplo, que vaya aumentando la velocidad a la que se mueve.

Podríamos hacerlo a “saltos” o de manera continua, yo me voy a decantar por esta segunda para hablaros de **operaciones matemáticas.**



Esto es lo que teníamos.
Podemos actuar en dos lugares:
a) Acortando el tiempo de espera
b) Cambiando los pasos que se mueve

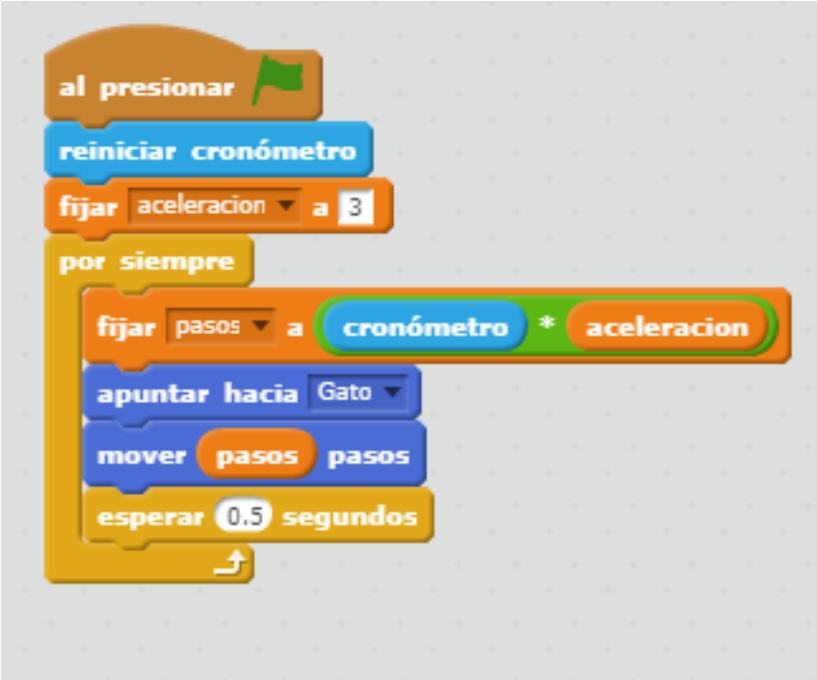
Voy a optar por lo segundo pero te animo a que pruebes las dos posibilidades.

Aquí lo interesante es que veas que en esos círculos con números, pueden meterse bloques con valores constantes o que vayan variando con el programa.

NOTA: Puede ser que en algunos momentos pienses que estoy usando variables de más, pero es una buena práctica poner las cosas en variables. Hace más legible el código y también nos permite hacer cambios generales en el programa, cambiando esas variables, por ejemplo cuando las usamos como “constantes”.

Voy a definir una variable aceleración a la que fijaré su valor al iniciar el programa, por ejemplo a 2. Como te digo, en mi caso, la pienso más como una constante.

La usaré varias veces en el programa (pudiendo sustituirla por el número 2 y ahorrarme una variable) pero si me parece que el murciélago va muy rápido o muy lento, en mi caso sólo tendré que cambiar ese valor al principio, mientras que en el otro caso me tengo que poner a buscar los 2 que he puesto, a ver en qué hilos, y cambiarlos todos.



Ahí voy. El producto está en el grupo OPERADORES
Bueno, de hecho me he calentado y he añadido otra variable.

Antes de que el murciélago avance, calculo los pasos que quiero que dé multiplicando el valor del reloj por el valor de “aceleración”.

Como te dije, esto me permite cambiar sólo en un sitio el valor de la aceleración. Que parece ser muy lenta.

Así que la subo a 3 a ver si le doy más emoción. Y… ¡ya te digo, va que se las pela!

En operaciones, además de las básicas: suma, resta, multiplicación y división tenemos otras curiosas, que agradecemos al MIT que pusiera.

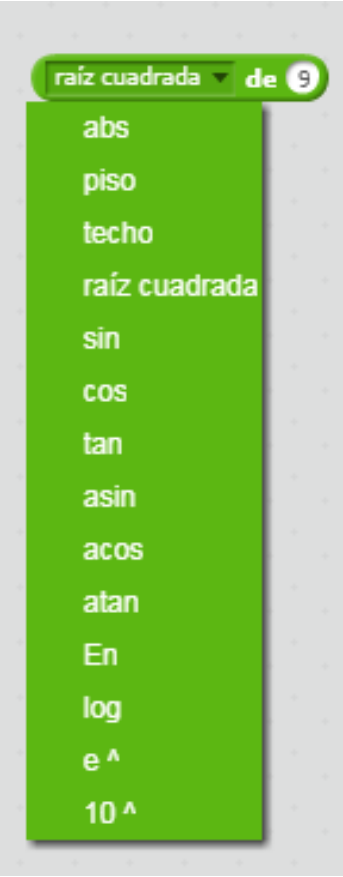
Pero antes, nota que los bloques de operaciones son “redonditos”, porque caben en agujeros redondos. Igual que los hexágonos arrojaban valores booleanos, (de sí o no). Las operaciones arrojan valores numéricos que pueden ponerse en los lugares donde Scratch nos pone números por defecto.

La primera te da el resto de dividir un número entre otro, lo que se llama a veces **módulo** y otras **residuo**.

La segunda me redondea el valor que pongo. Te dejo a ti que de deberes que veas qué sale con el número que he puesto.

Iremos hablando de algunas de ellas, pero no te despistes que en último bloque hay un montón juntas.

Mira



Por orden:

Valor absoluto de un número
El entero más próximo por debajo (floor)
El entero más próximo por arriba (ceiling)
La raíz cuadrada
Seno
Cos
Tangente
Arco seno
Arco coseno
Arco tangente
Logaritmo en base e (mala traducción de ln)
e elevado a…
10 elevado a…

MENSAJES

Práctica 9. Los objetos se comunican entre sí

Como has visto cada objeto (incluido el escenario) tiene su propio código, su propia programación.

No van del todo a su bola, porque se comunican a través de los valores de las variables globales, que unos objetos pueden cambiar y otros "darse cuenta" a través de un condicional, por ejemplo, y reaccionar. Como hemos hecho con las vidas.

También hacíamos un poco de trampa con el bloque "detener" que era capaz de detener los hilos de programación de otros objetos.

Y finalmente con algunos bloques del grupo SENSORES que sabían si dos objetos se estaban tocando, por ejemplo.
Pero ahora vamos a hacerlo más limpiamente, mediante los mensajes.

Los tenéis en el grupo EVENTOS

En un momento concreto de un hilo, por la razón que sea, queremos que las cosas "cambien". Por ejemplo, se han acabado las vidas, has alcanzado una puntuación máxima, has adivinado un valor, ha terminado la introducción de una aplicación y ahora empieza el funcionamiento normal, ha terminado la aplicación… y un enorme etcétera.

En ese momento enviamos el mensaje, este irá a todos los objetos (**broadcast)**, que puede que reaccionen o no a él.

Si comenzamos un hilo "al recibir mensaje1" en un objeto este hilo se activará cuando el mensaje se mande.

Por supuesto, en algunos casos podemos añadir en ese hilo el bloque "detener otros programas en el objeto" de forma que este mensaje cambiará totalmente el comportamiento del objeto reduciéndolo solamente a lo que manda el hilo que se dispara con el evento recibir mensaje.

Por ejemplo, si al terminar una partida quiero que el gato deje de responder a mis controles y se quede en el centro dando saltos, pondré un hilo con ese tipo de movimiento y el bloque "detener otros programas en el objeto" para que se paren los hilos que organizaban el comportamiento "normal" del gato.

Sería algo así.

En algún objeto tendríamos algo parecido a

Como ves se puede cambiar el nombre al mensaje y se pueden tener preparados muchos mensajes distintos.

Al igual que en las variables, es muy conveniente ponerles un nombre claro de su función.

Y en el gato tendríamos algo así

Cuando el gato reciba el mensaje, detiene todos sus hilos (salvo este).
Se va al centro de la pantalla
Se pone a cambiar de disfraz

EXTRA: Te desafío... a que cuando se acaben las vidas, envíes un mensaje de “game over” y que reaccione todo el mundo. (Sí, también puede reaccionar el propio objeto que lo manda)

Por ejemplo, cambiemos este código que teníamos

Aquí íbamos mirando si es se tocaban, en caso afirmativo le quitábamos una vida, esperábamos un momento para que se pudiera separar, luego comprobábamos si le quedaban vidas. En caso de que no fuera así, deteníamos el hilo de cálculo de vidas.

En lugar de ese detener… enviemos un mensaje, y que reaccionen el gato, el murciélago y el escenario.
Pondrás un hilo en cada objeto “al recibir game over” de forma que:

- El fondo del escenario cambie (si quieres poner que pase de uno a otro de distintos colores, te puede parecer que parpadea)
- El murciélago y el gato desaparezcan o se vayan a donde quieras, se queden quietos, bailen, cambien a un disfraz de “perdedor”... tú verás.

Me queda por contarte para qué vale este bloque

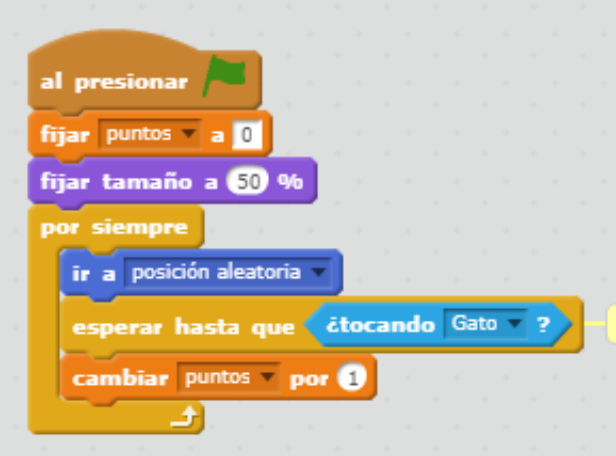
Sencillo. **Envía el mensaje y se queda esperando a que terminen los hilos que ha disparado** antes de seguir ejecutando los bloques que tenga debajo.

P.ej.: En un juego que pases de una puntuación, salga un momento una pantalla de “Nuevo Record” (un cambio de fondo) y luego siga ejecutándose normalmente.

ALEATORIEDAD

Práctica 10. Busca manzanas ver.1

Pongamos a nuestro gato a buscar manzanas. Le pondremos una en una posición aleatoria, esperaremos a que se la coma, le sumamos un punto y luego la sacamos otra vez.



Se me hacía un poco grande la manzana así que la he reducido un poco de tamaño.

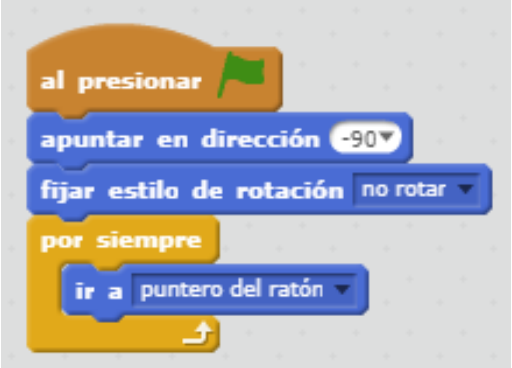
Verás que la mando a una posición aleatoria con un bloque muy sencillo.

Después me quedo esperando a que la toque el gato, sumo un punto, y repito: la mando a otra posición aleatoria...

No es la única manera de ir a una posición aleatoria. Se puede usar este bloque que está en el grupo OPERADORES

En mi caso puedo poner un bloque en cada coordenada y controlar incluso en qué rango quiero que me ponga la manzana, por ejemplo si pongo en la coordenada x un número al azar entre 0 y 240 sólo aparecerá en la mitad derecha del escenario.

Para mover el gato, me he permitido dejar los cursores a un lado y aprovecharme del ratón que me da un movimiento más rápido y fluido.



Los dos primeros bloques son para que el gato empiece mirando a un lado y luego no se mueva hacia un lado y otro (aunque el movimiento del ratón es tan rápido que no le da tiempo a hacerlo)

Después sólo le pido que vaya directo al lugar donde tengo el ratón.
Verás que va a ¡toda pastilla!

Si quieres hacerlo más chulo:

Que con la bandera verde:
Desaparezcan el gato y la manzana
En el escenario se ponga un fondo de inicio, se espere un momento y mande el mensaje “empezar

Al recibir empezar:
Aparece el gato y la manzana con sus hilos normales. Los mensajes pueden disparar varios hilos sin problemas.
El escenario cambia a fondo de partida normal

Cuando los puntos lleguen a cierto valor, se envía el mensaje “game over”

Al recibir game over
El escenario cambia a fondo final
La manzana desaparece
El gato pasa al centro, quieto y diciendo la puntuación

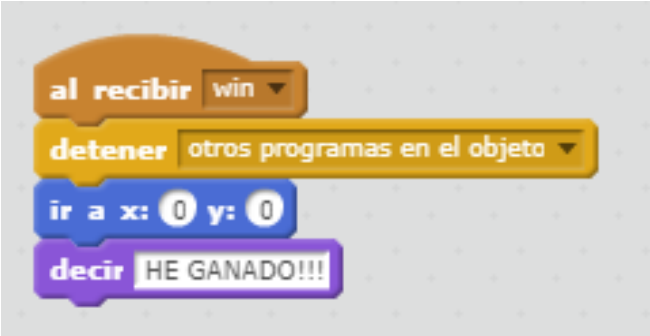
El bloque decir lo tienes en el grupo APARIENCIA

El valor de los puntos lo tienes porque definiste la variable antes.

Por ejemplo, así puedo enviar el mensaje

Manzana

Gato



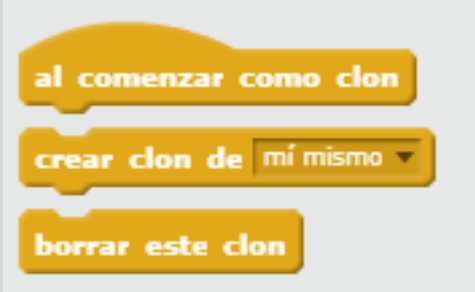
Si te fijas, en la manzana, el cambio de posición repetido por siempre se interrumpe cuando se cumple la condición de que los puntos sean 20, en ese momento escondo la manzana, envío el mensaje “win” y detengo este hilo.

En el gato, se paran todos los hilos del gato (para poder hacer lo que quiera yo) y lo mando al centro a celebrar su victoria.

CLONES

Pensemos en otro juego, hagamos que aparezcan manzanas, pero quiero que aparezcan VARIAS manzanas a la vez.

En este caso haremos **clones**. Sólo tendremos un gato y una manzana, pero iremos haciendo copias de ella (en programación orientada a objetos serían **instancias).**



Están en el grupo CONTROL

Los clones los creamos con el segundo bloque y puedo clonar el objeto en el que se está ejecutando el bloque o bien puedo clonar **otro** objeto, aunque la orden de clonado se mande desde este objeto.

En el objeto clonado pondremos el "disparador" del hilo que ejecutará el clon "al comenzar como clon" y debajo pondremos su código.

Cuando queramos que desaparezca, pues borramos el clon.

Práctica 11. Coge manzanas ver.2

El código del gato sería como antes.

Este sería el primer hilo que pondría en la manzana

Como ves, es un puro creador de clones

Me aseguro que la manzana aparezca (como aquí jugamos a que desaparezca cuando la toco, así que es mejor estar seguros de que está visible cuando quiero yo)

Me doy un tiempo entre clon y clon para no agobiar al pobre gato.

El segundo hilo, claro, sería, ¿qué quiero que haga el clon cuando sale?

Primero hago la manzana más pequeña y elijo una posición aleatoria para colocar este clon.

Miro si toca al gato, en caso afirmativo, sumo un punto. En caso negativo no hace nada más volver a preguntarlo hasta que se dé el caso positivo.

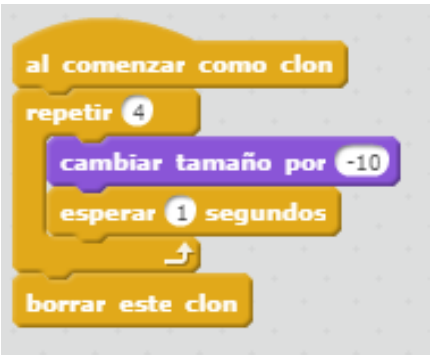
Si he sumado el punto, miro si ha llegado a la puntuación máxima. En caso afirmativo escondo la manzana, y lanzo el mensaje "win" para hacer lo que hablamos en el ejemplo anterior.

Si aún no ha llegado a esa puntuación la ejecución del hilo sigue y le pido que borre el clon y se acabó la vida de ese clon, no del objeto.

Ejecutadlo y ved que, si esperáis, los clones van apareciendo y puede haber más de uno a la vez (si no te los comes).

Ved también que las posiciones son distintas para cada clon, porque la elección de la posición la hago en el hilo de cada clon, no en un hilo principal del objeto.

Si me dejáis que hagamos una floritura más… ¿Qué os parece si hacemos que según aparezca el clon comience a reducirse hasta que desaparezca si no te lo comes a tiempo?



Tengo dos hilos que se disparan al crear un clon. El de antes que controla si te lo comes y este que lo va reduciendo.

¡Prueba a ver qué tal se te da!

¿Te apetece complicarlo un poco más?

¿Qué te parece si le damos puntos diferentes al gato según el tamaño de la manzana que se come?

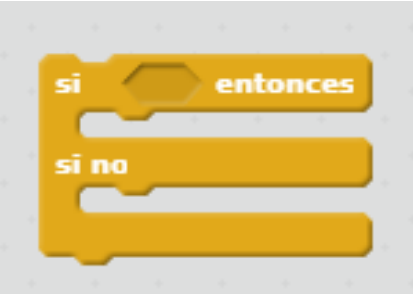
Aquí te pongo un ejemplo de resolución no demasiado elegante, en el que, al tocar la manzana me pregunto luego en varios condicionales por el tamaño de la manzana y asigno puntos diferentes en los distintos casos.

Fíjate que como sumo fracciones de punto, no es seguro que en ningún momento llegue justo a 20, así que he cambiado el condicional del final para ganar cuando supere los 20 puntos.

Esto es más habitual en robótica porque las variables que se miden (luz, distancia, sonido, etc.) no toman valores concretos, así que, en lugar de preguntarnos si la distancia es igual a 50 cm, nos preguntamos si la distancia es mayor que 50 cm (si lo que nos preocupa es alejarnos)

¿Por qué digo que es poco elegante?
Fíjate que aunque se cumpla el primer condicional, va a seguir mirando los otros. Por ejemplo, si el tamaño es 50, sumará el punto, y luego se preguntará si es 40, que no lo es, así que no suma el medio punto, pero luego se pregunta otra vez… y vaya, después del primer condicional ya podríamos saber que no va a cumplir los siguientes.

Esto se puede solucionar con **condicionales anidados**.



Hay un bloque que no hemos usado aún, no nos hizo falta

Es muy similar al que estábamos usando, lo que pasa es que, además de ejecutar un código cuando se cumple la condición por la que nos preguntamos, también puede ejecutar otro código cuando no se cumple.

Por ejemplo, mira la edad y hace una cosa si son mayores de 18 y otra si no son mayores de 18.

Para anidar condicionales, lo que hacemos es preguntarnos así:

¿Se cumple la condición1?
Sí. Chupi
No. ¿Se cumple la condición2?

Por ejemplo,
¿Vienes solo?
Sí. Guay
No. ¿Vienen tus padres?

Así, si nos dice que viene solo, no le preguntamos si vienen sus padres, sólo se lo preguntamos en el caso en el que sabemos que no viene solo. Con bloques sería:

Aplicado a nuestro caso

Si te fijas, no se evalúan todos los condicionales en todos los casos. Sólo evalúo los “anidados” cuando el anterior no se ha cumplido.

Con estas cosas hay que tener cuidado que a veces no los escribimos bien y no contemplamos todos los casos.

Una manera es dar los valores posibles o uno de cada rango que queramos valorar y ver si funciona como debe.

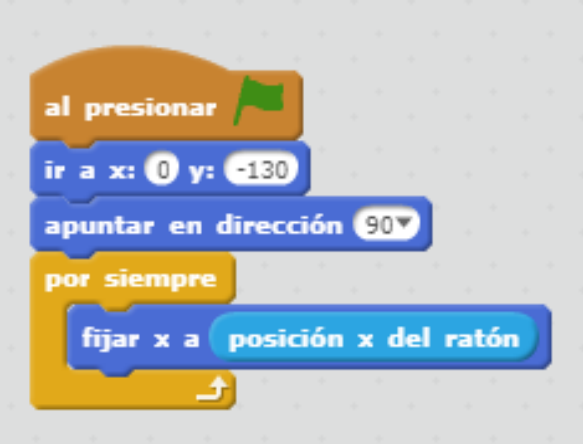
Si queréis podemos intentar una aproximación más elegante, haciendo el puntaje una función del tamaño y lo liquidamos con un sólo bloque.

Con este bloque no puntuamos igual que lo que teníamos, sería una puntuación proporcional al tamaño. Tiene la ventaja de que si la reducción de las manzanas fuera más gradual, este bloque funcionaría teniendo en cuenta todos los casos posibles de tamaño de manzana.

Práctica 12. Caen manzanas

Vamos a hacer un juego parecido pero para recoger las manzanas que caen del cielo.

Voy a mover el gato con el ratón, pero sólo la coordenada x. La coordenada y la dejaré fija. Para averiguar a qué altura me gusta, arrastro el gato usando el ratón, lo coloco donde me guste y leo las coordenadas x e y (en el bloque "ir a" de MOVIMIENTO).

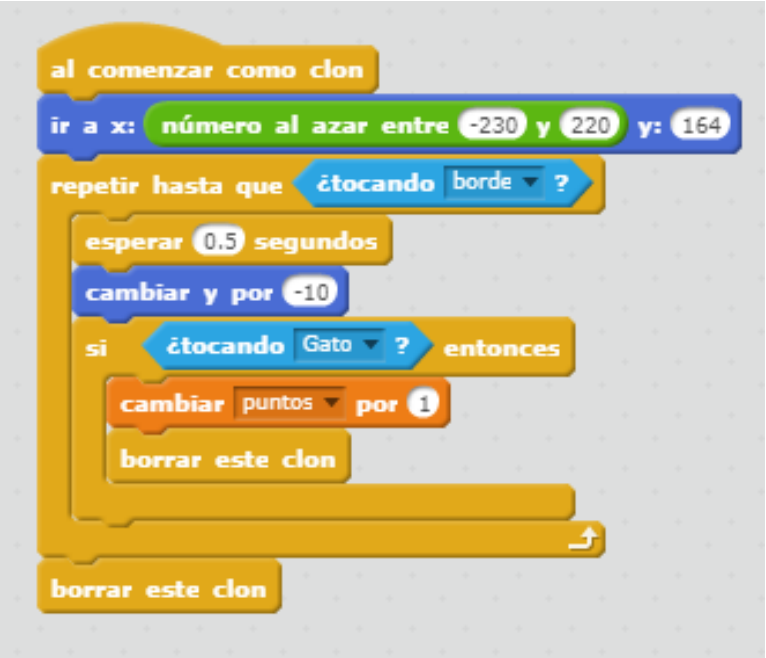


El bloque posición x del ratón pertenece, como podrías adivinar por su color al grupo SENSORES.
También podría haber usado este bloque

Prueba cómo se mueve, fácil y bien, ¿no? Esto es útil para multitud de juegos que quieras programar.

Vamos a por las manzanas.

Aquí ves el hilo generador de clones, de paso inicializo los puntos a cero y dejo las manzanas a su tamaño reducido.



Moviendo “a mano” la manzana he visto que la posición buena de altura es 164 más o menos. Así que hago que cada clon aparezca siempre a esa altura, pero en una posición x aleatoria entre esos límites (que son los que no me “cortan” el dibujo de la manzana, comprobado a mano).

Voy cambiando la coordenadaY poco a poco hasta que toque un borde (por construcción, sólo puede ser el de abajo) y allí desaparezca.

Cada vez que muevo el clon un poco hacia abajo compruebo si el gato está tocando la manzana para contar un punto más porque se la ha comido. Después, por supuesto, borro la manzana.

Si lo pruebas, verás que no va mal, pero si queremos ser finos… ¿te das cuenta de que hay un tiempo durante el que puedo estar tocando la manzana sin comérmela?

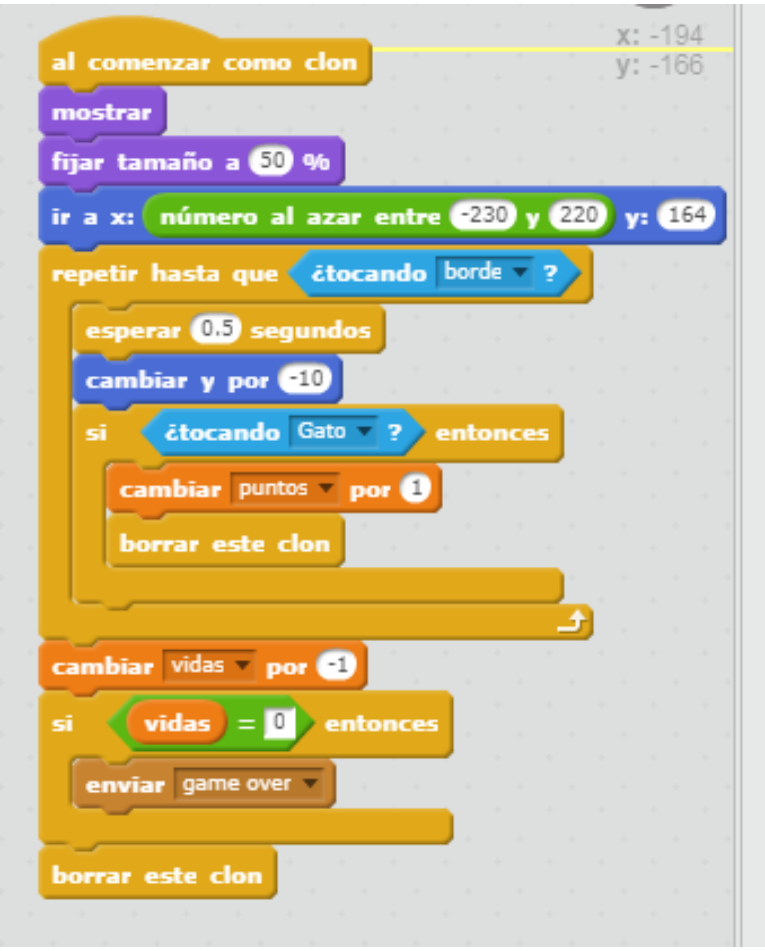
Efectivamente, **ese “esperar 0.5 segundos” detiene ese hilo de programación durante ese tiempo,** así que ni se comprobará durante ese tiempo si está tocando el borde (no pasa nada, porque el movimiento de la coordenadaY se hace a saltos) ni se comprobará si está tocando al gato. Ejecútalo, fíjate y lo notarás.

Si fuera crucial que en cuanto el gato tocara la manzana, esta se diera por comida. Habría que poner en un hilo el asunto del movimiento y en otro el asunto de los puntos.

NOTA: O bien hacerlo de una manera más fina que suele usarse mucho en el mundo Arduino, donde sólo tenemos un hilo de programación. Los curiosos pueden consultar mi manual sobre el tema, pero os avanzo que lo que se hace es (en lugar de esperar) mirar el tiempo cada vez que pasas por ese punto del bucle y si ha pasado el tiempo suficiente andar, y si no, esperar a la próxima vuelta del bucle. Así no tienes que parar el hilo y no dejas de mirar el sensor.

Práctica 13. Caen manzanas con límite de vidas

Refinemos. En el programa anterior si las manzanas llegaban abajo desaparecían y no podía cogerlas y ganar puntos, pero podríamos hacer que, cuando una manzana llegara al suelo, se perdiera una vida, y al cabo de varias vidas, la partida.



Verás que es muy similar al código del programa anterior, salvo al final.

Si el bucle "repetir hasta que tocando borde" se completa sin que se cumpla que toques gato y se borre, significa que has llegado al suelo, al borde.

Así que lo que ponga después se ejecutará una vez que la manzana esté en el suelo.

Como ves, quita una vida y borra el clon.

Salvo que nos hayamos quedado sin vidas, entonces manda el mensaje "game over".

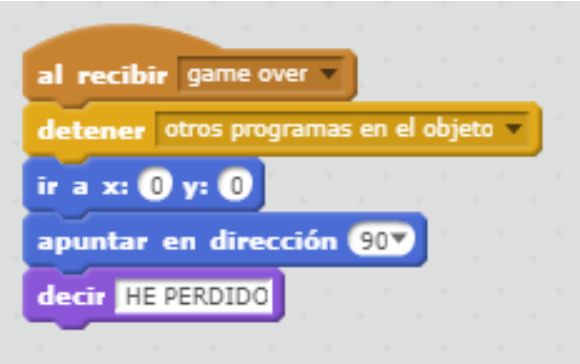
Ese mensaje actúa sobre las manzanas y sobre el gato.

En el caso de las manzanas

Detiene los otros hilos de las manzanas.

- El que acabo de poner
- El bucle generador de clones

En el caso del gato



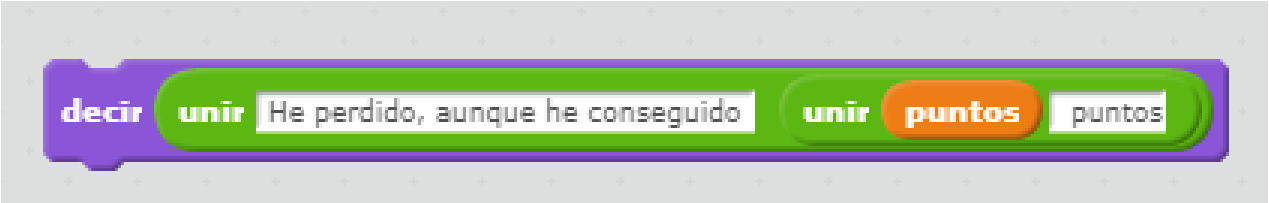
Detiene todos los hilos del gato. El único que había, aparte de este, es el que gobernaba el movimiento del gato usando el ratón.

Lleva al gato al centro de la pantalla y dice que ha perdido.

Si queréis también podría decir cuántos puntos ha conseguido hacer. Usando las herramientas para concatenar texto del apartado OPERADORES

La herramienta es la primera, en el segundo bloque lo que he hecho es unir tres para que veas que puedes poner lo que quieras, no solamente dos palabras. Quiero decir, incluso variables.

Mira y fíjate en los espacios necesarios después de “conseguido” y antes de “puntos”.



Y este es el efecto. Mola

Pensemos en algunos refinamientos más que podríamos incluir…

Por ejemplo, podríamos hacer que la caída de las manzanas fuera acelerándose, como en la vida real. Vayamos paso a paso.

Práctica 14. Una manzana cae acelerada

Hagámoslo primero sin clones, sólo una manzana.

Ya sabes que la fórmula que rige la caída es $y = \frac{1}{2} g t^2$

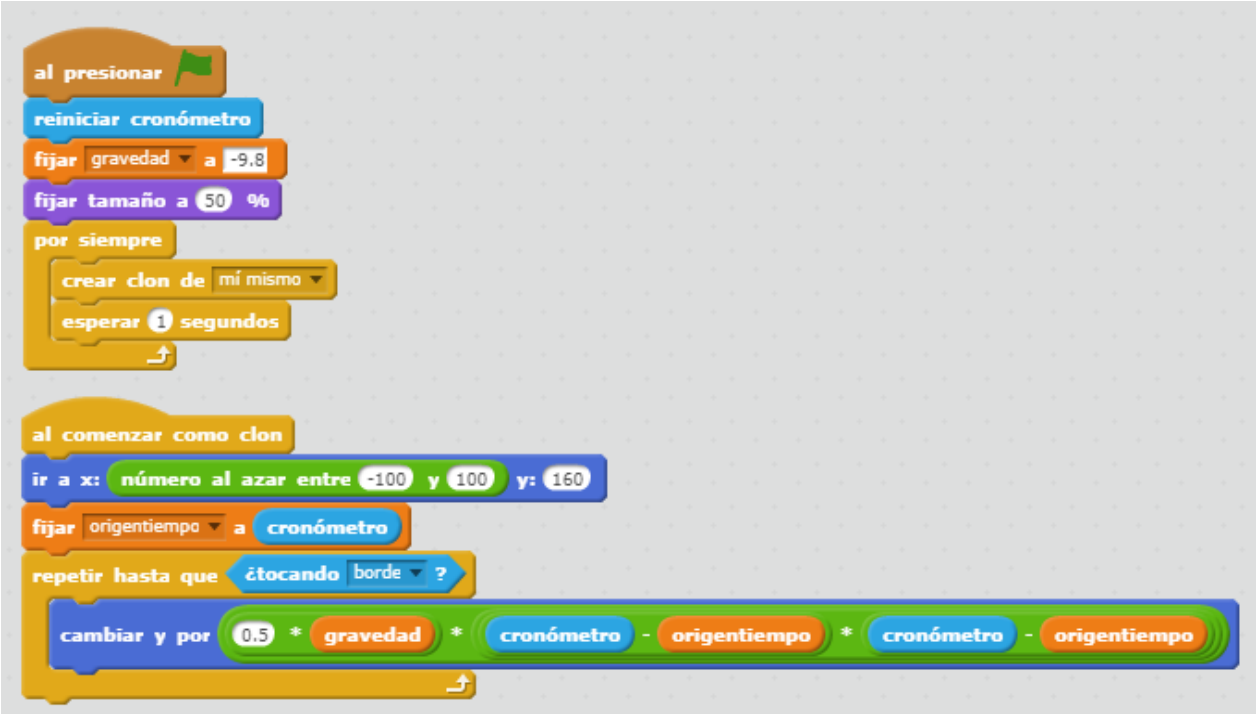
Eso vale si el tiempo inicial es cero, si no hay que sustituir t por (tiempo-tiempoinicial)

Dejo la gravedad como una variable (aunque hace el papel de constante) para que si se te hace mucho o poco sólo tengas que cambiarla en ese bloque inicial, además de la claridad que da al código.

Si pensamos ahora en generalizar esto para hacerlo con clones… tenemos un problema, porque el origen de tiempos será diferente para cada clon.

Si ponemos este hilo para cada clon que sea creado, el origen de tiempos se irá sobreescribiendo cada vez que aparezca uno y nos estropeará el movimiento.

No lo creas, compruébalo.



Cada vez que sale una manzana nueva se detienen todas y empiezan a caer como si se las soltara desde el punto en el que están.

Cuando creas una variable recordarás que nos preguntaban si queríamos que fuera aplicable a todos los objetos o sólo a uno… pero es que nosotros queremos que sea aplicable sólo a un clon, una variable muy local.

Aquí vienen en nuestra ayuda **las funciones**

FUNCIONES

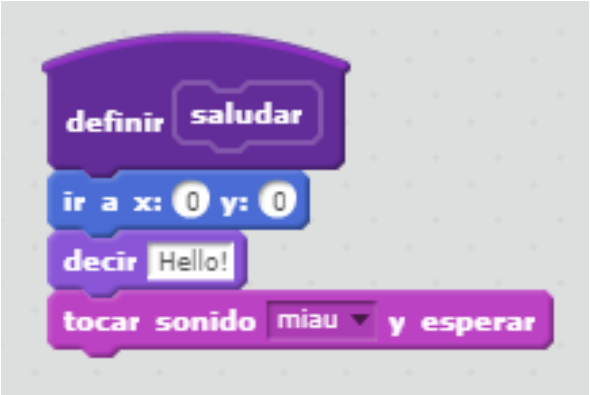
Una función es un **cachito de código que vamos a resumir en un solo bloque**.

Se definen en el grupo MÁS BLOQUES.

Por ejemplo, voy a crear la función saludar.

Me pide que le ponga nombre a la función y aparece un bloque que pone “saludar” y un “iniciador” en la zona de programación, para que yo vaya poniendo debajo el hilo, el cachito de código, que ejecutará la función cuando se la llame.

Muy sencillo.



Cuando se invoque a la función saludar, el gato irá al centro, dirá Hello y sonará el maullido.

Aunque Scratch lo permite, lo normal no es ejecutar una función según se inicia el programa. Lo habitual es que tú la tengas definida y la invoques cuando quieras.

En ese sentido **una función es como una "habilidad"** que tiene el programa.

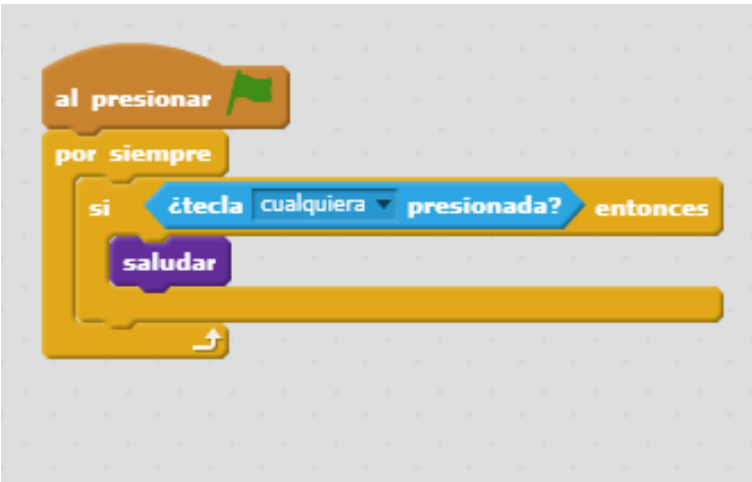
Por ejemplo, tú sabes barrer, fregar, ir a comprar, bailar… pero no haces todo constantemente desde que te despiertas hasta que te duermes.

Simplemente te comportas así:

Repetir {Fregar} hasta platosSucios = 0

Sólo se activan cuando se "invocan".

Así que yo podría hacer mi código principal así.



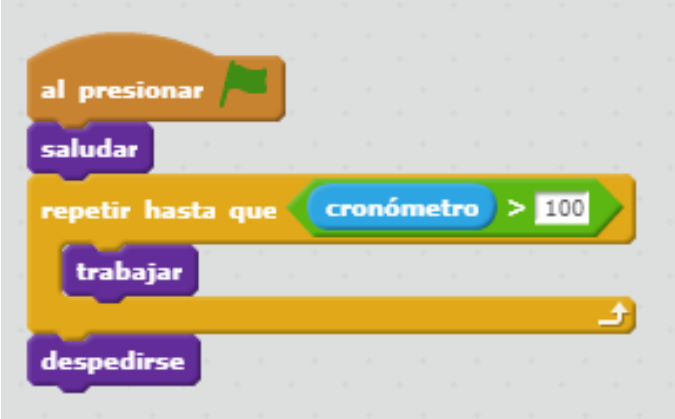
Este es el programa principal, la función se invoca cada vez que se pulsa una tecla.

Podría ser llamada mil veces o ninguna, y yo, como programador, no lo sé, porque eso **lo decidirá el usuario**.

Pero la "habilidad" que nos da la función está siempre disponible durante el programa

De esta forma, **una manera de programar es ir "encapsulando" la complejidad de lo que se hace en funciones** y mantener en el programa "principal" básicamente las estructuras de control: bucles, condicionales y demás.

Algo así como



Cuando empieza el programa ejecutará la función saludar, después repite trabajar una y otra vez hasta que nos pasamos del tiempo estipulado y, en ese momento, nos despedimos.

De esta manera la estructura es sencilla de programar y ver

La dificultad de programa completo la hemos dividido en partes:

- Por un lado la estructura de control (bucles, condicionales, etc.)
- Por otro lado cada una de las funciones que hacen las tareas.

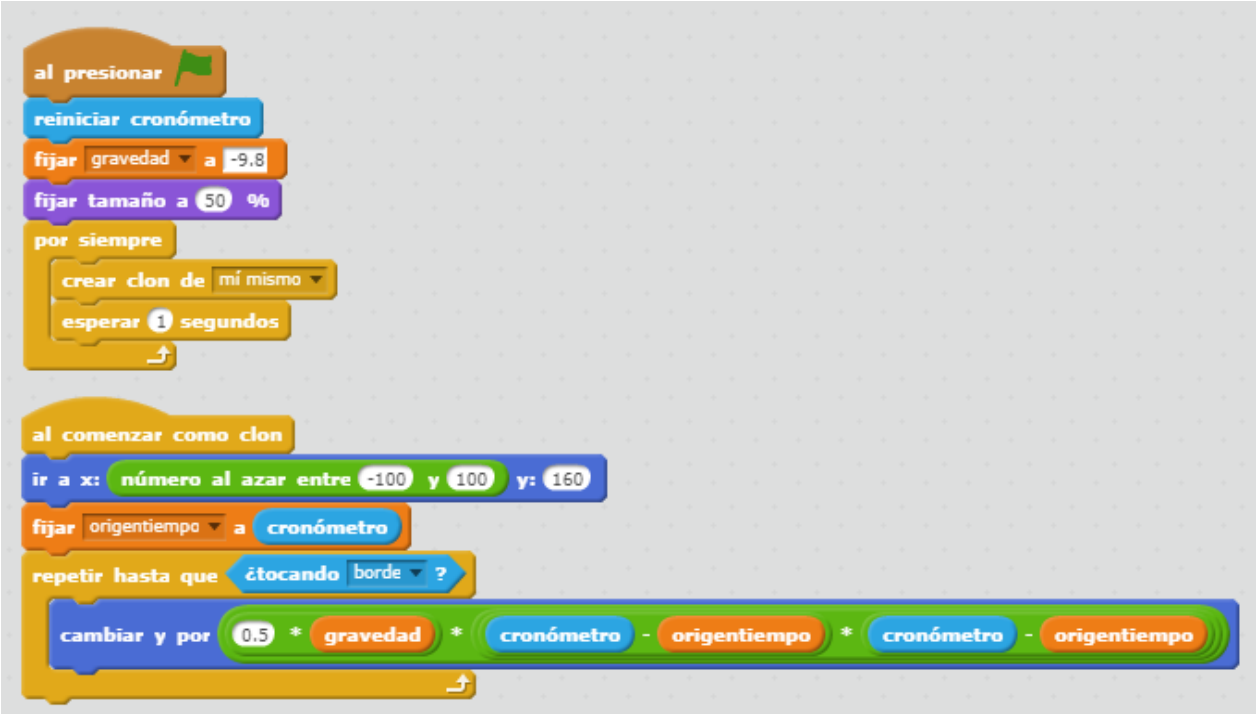
Ahora podemos concentrarnos en programar el código de cada una de las tareas (las funciones) que necesitamos, sin que nos líen las otras funciones, ni la propia estructura de control del programa.

DIVIDE y VENCERÁS, ya sabéis.

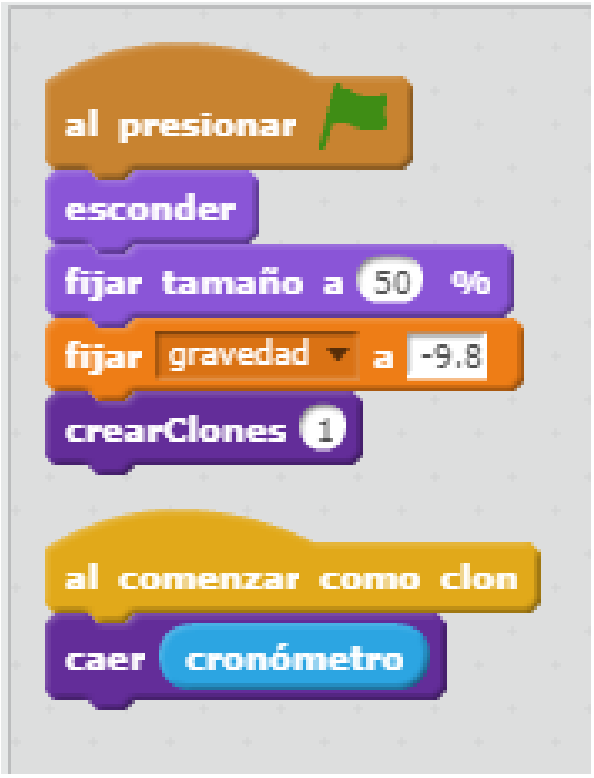
Práctica 15. Manzanas aceleradas caen

Hagamos un programa donde el gato se mueve por abajo usando el ratón (como ya sabemos) y que vayan apareciendo manzanas (usando clones) que al caer se vayan acelerando.

Habíamos hecho esto, pero no iba bien, recuerda que cada vez que aparecía un clon nuevo el origentiempo se reseteaba. Y nosotros queremos que cada clon tenga su propio origen de tiempos.



Vamos a meter funciones locamente (código de la manzana)



Al empezar, inicializamos la visibilidad de la manzana, su tamaño y la gravedad.

Después llamamos a la función crear clones… y ya veremos luego cómo la hacemos.

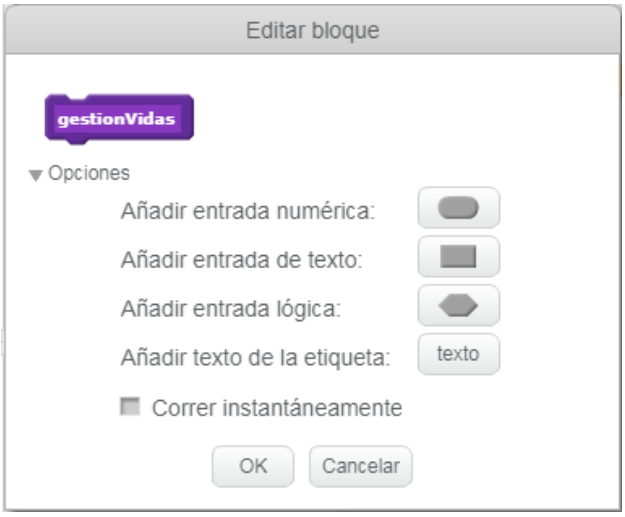
Cuando un clon es creado, llamamos a la función caer… y ya veremos luego cómo la hacemos.

Esa es la filosofía...

Empecemos por la función crear clones, que será básicamente ese bucle infinito con una espera para que no salgan todos a la vez.

Hasta ahora no nos habíamos preocupado de ese tiempo, pero podría ser interesante que fuese posible cambiarlo, por ejemplo, con una **variable local.**

Si le das con el botón derecho al bloque crearClones una de las opciones será editar. Elígela, haz click en opciones y verás desplegada esta ventana.



Las opciones que ves son los **parámetros** que permite la función. Valores que le voy a pasar a la función cuando la llame para que los use, en nuestro caso el tiempo de espera entre la creación de clones.

Como eso será una entrada numérica, selecciono la primera, le pongo un nombre como dios manda ;) y OK.

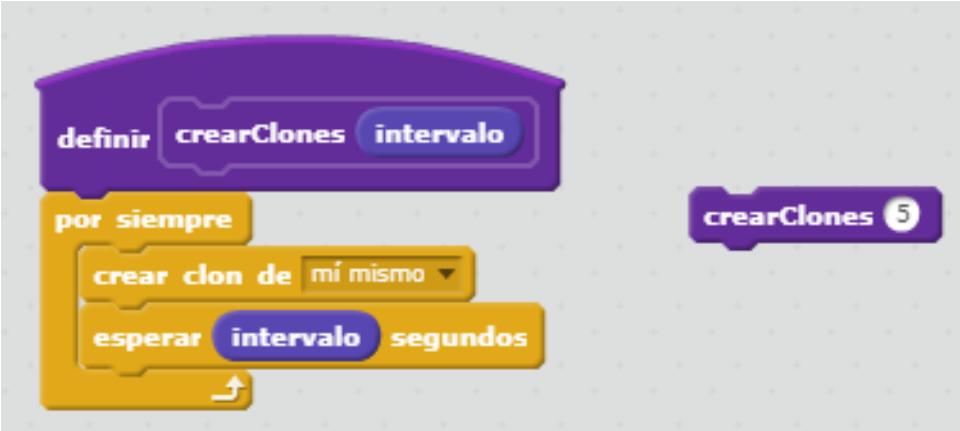
Así que ahora tengo esto

Ya tengo la etiqueta para crear el hilo de la función y el bloque para llamarla, que ahora admite un parámetro, un número.

Ese número será el valor de la **variable local "**intervalo" dentro de la función.

A ver si con el ejemplo…

Este es el hilo, la definición, de la función y al lado el bloque con un ejemplo de cómo podría invocarla.



Si la invoco como crearClones 5 dejará cinco segundos entre clon y clon, pero también la puedo invocar como crearClones 6 o con cualquier otro número.

Por cierto, para usar el bloque “intervalo” hay que arrastrarlo desde el bloque de inicio de la definición de función.

Si buscas la variable intervalo en el apartado DATOS, donde están todas las variables… verás que NO ESTÁ. Esta es una variable local, sólo tiene sentido y valor dentro de la función en la que está, por eso decimos que es LOCAL, insisto.

El asunto de creación de clones está resuelto. Ya pusimos donde queríamos que se ejecutase esa función y la hemos definido.

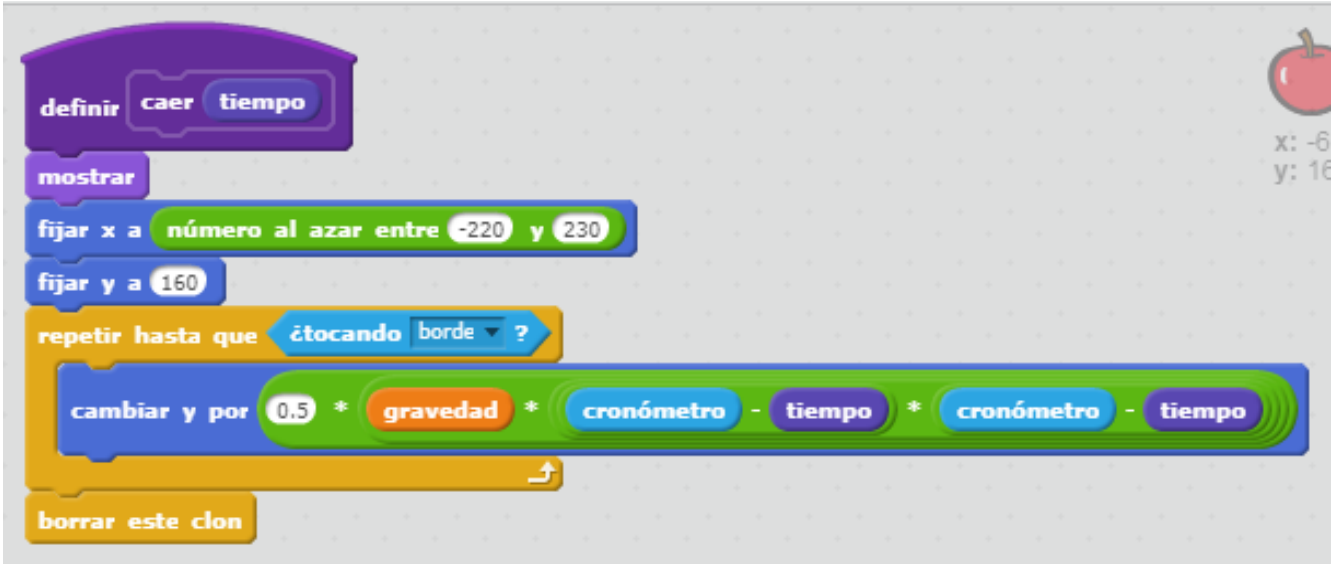
Ahora vamos a lo de caer. ¿Recuerdas que teníamos un problema con que el origentiempo de cada clon queríamos que fuese distinto? ¿Y si ese tiempo lo dejamos como una variable local de la función caer? Así cuando cada clon invocase a la función caer con el tiempo de creación, eso se quedaría allí… y ya estaría todo resuelto.

Verás:

Le pongo un parámetro a la función caer, que será el tiempo en el que empieza la caída. El código me queda así

El valor del cronómetro en ese momento se mete en la función guardado en la variable local “tiempo” y ya lo puedo usar sin que me varíe cuando lo haga el reloj.

Y la función



Recuerda que la llamamos así

Al llamarla con el cronómetro como parámetro, si en ese momento el cronómetro tiene el valor 2 segundos, ese es el número que se queda metido en la variable local tiempo.

Y ahora, aunque el cronómetro sigue cambiando, no lo hará la variable tiempo, así que podemos tomarla como el origen de tiempos y será distinta cuando se invoque esa función desde los distintos clones.

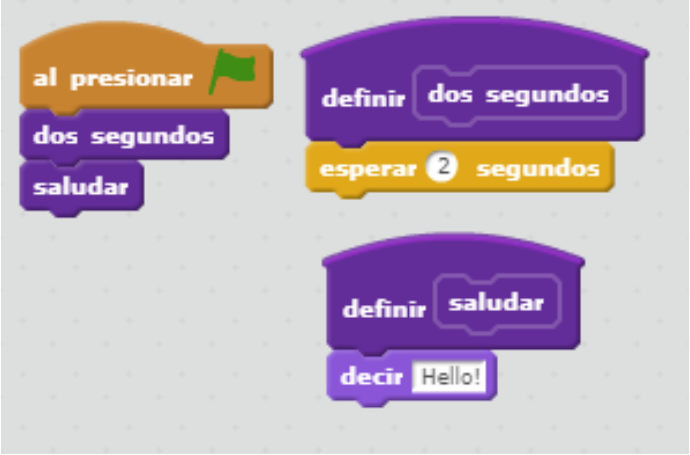
Hasta aquí ya hemos conseguido el movimiento de los clones de manzanas como queríamos, si además queremos llevar la gestión de puntos y de vidas… hay que hacer alguna cosilla más.
Podríamos pensar en hacer esto.

Hacer una función gestionPuntos que se activara en ese hilo y anduviera mirando todo el rato si el gato toca a las manzanas o no, pero me ha asaltado una duda:

¿Cuando llamo a una función manda la orden y sigue con el hilo o espera a que termine de ejecutarse antes de seguir con el hilo?

Las dos cosas son pensables o podríamos tener la posibilidad de elegir una cosa u otra, ¿recuerdas que en los mensajes era así?

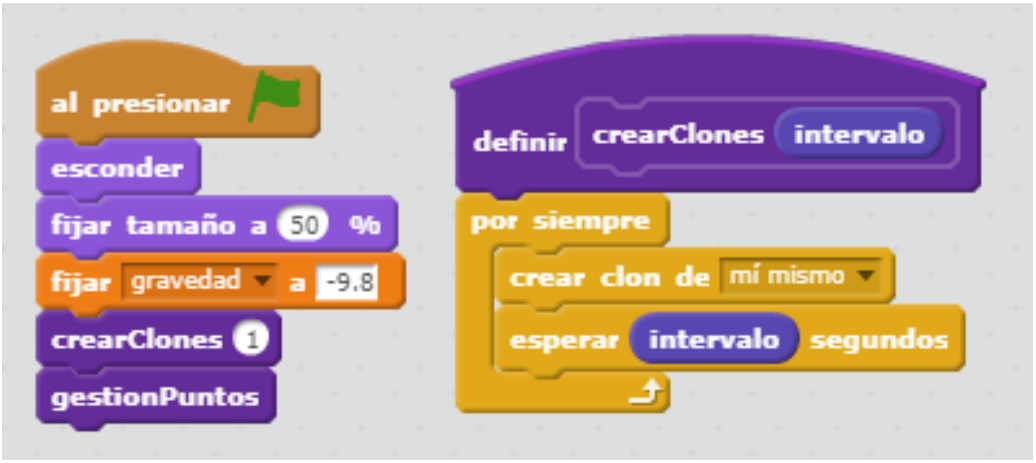
Así que me he abierto otro programa y he hecho esto



Y mi pregunta es si el saludo va a salir según presione la bandera o va a tardar dos segundos…

Mola que lo pruebes, pero vaya, tarda dos segundos. Así que cuando invoco a una función, no se sigue ejecutando el hilo del que veníamos hasta que la función no se acabe.

En nuestra primera propuesta de antes tenemos un problema



La función crearClones tiene un bucle infinito, así que no va a volver NUNCA al hilo principal…

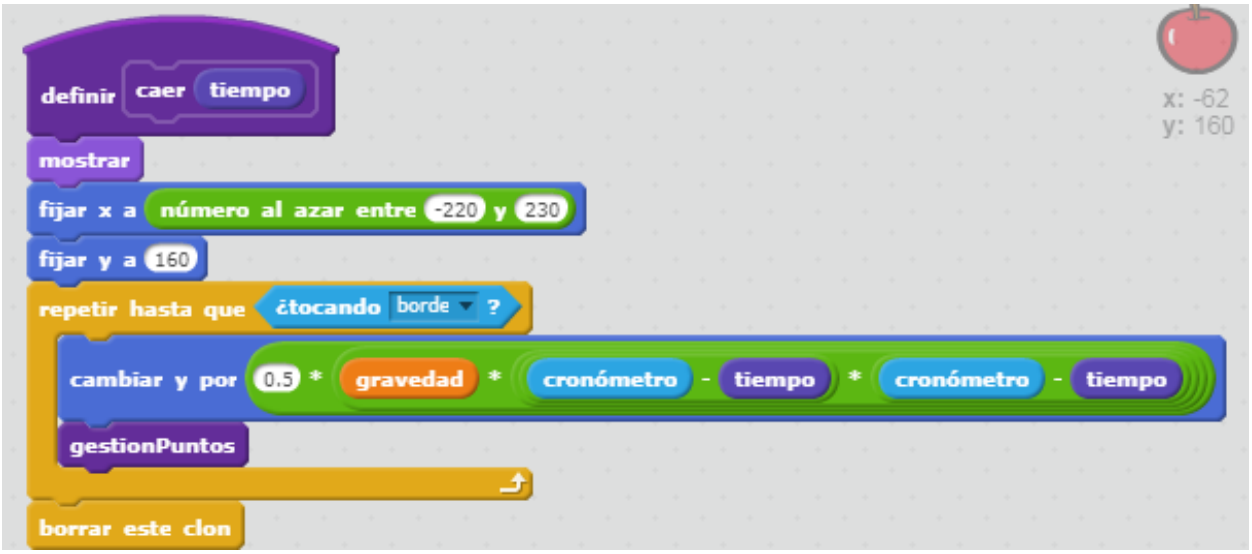
La función gestionPuntos tiene que ser llamada desde otro hilo. Pero no pasa na', se llama y listo.

Pero tenemos un problema si pensáis en hacer esto

Este al presionar la bandera no nos hace relacionarnos con el clon… probadlo y veréis que no os coméis ni una manzana.

Nos quedan dos opciones, o disparamos la gestión de puntos con cada clon (en lugar de con la bandera), o bien ponemos ese condicional dentro del bloque caer.

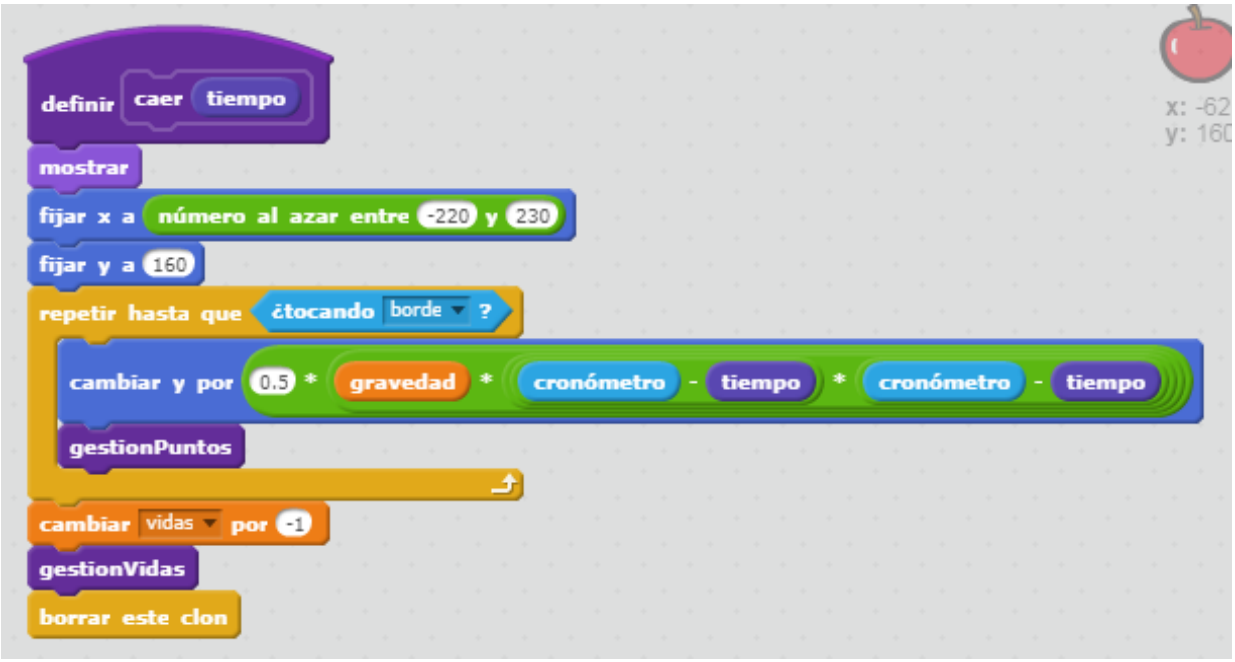
Apuesto por esta segunda opción… pero voy a crear una función en lugar de poner el condicional tal cual.



¿Veis? Después de cada cambio en la coordenadaY invoco a la función gestionPuntos que voy a definir así

Después de cada cambio se evalúa este condicional y se cambian los puntos y borra el clon si procede y si no, pues nada, de vuelta al "repetir hasta tocando borde" y hacemos otra iteración

Y sí, ¡sorpresa! **SE PUEDE INVOCAR A UNA FUNCIÓN DESDE DENTRO DE OTRA!!** Nos faltaría quitarnos una vida cada vez que la manzana llegue al suelo…muy bien: HAGAMOS OTRA FUNCIÓN: gestionVidas



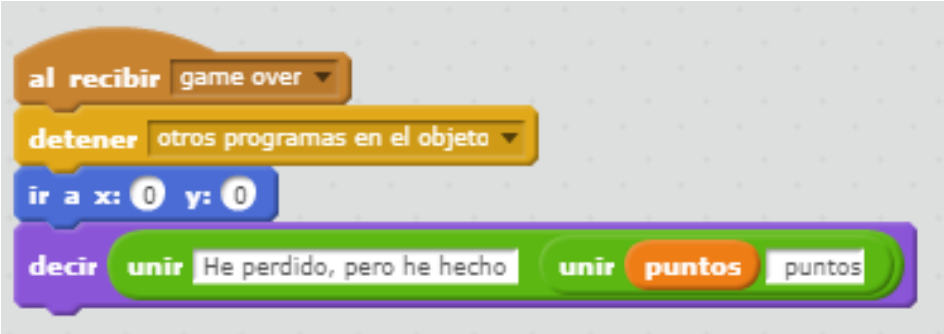
Cuando salgo del bucle "hasta tocar borde" es que la manzana ha llegado abajo sin que se la haya comido el gato. Así que quito una vida y miro en la función gestionVidas a ver cómo va la cosa…

Facilito, un condicional, simplemente para comprobar si estoy ya sin vidas.

Si es así mando el mensaje game over para cerrar todos los hilos y si no, pues tranquilamente vuelvo a donde se llamó esta función. Se borra el clon y a otra cosa.

Hace un rato advertimos de que cuando llamamos a una función el hilo desde el que se hace la llamada se queda esperando a que la función acabe para seguir, y desde nuestro hilo caer se llama a ¡DOS FUNCIONES! Entonces, ¿va a funcionar mal? Mira, no, funciona estupendamente, esas dos funciones no tienen esperas ni bucles infinitos y se ejecutan en un plis plas ;)

Le he puesto estos hilos al gato y a la manzana, respectivamente
Gato



Manzana

Quizá te llame la atención el esconder… pero es que si no se me quedan las manzanas que estaban cayendo por ahí "colgadas".

Nos ha llevado un ratillo, pero hemos aprendido un montón de cosas… repasa y verás.

EXTRA: Si quieres complicar tu juego te propongo que elijas un nuevo objeto, una naranja por ejemplo y que sean "tóxicas", quiero decir, que si las coges te quitan una vida, pero si las dejas llegar hasta el suelo que te den un punto.

Un comentario sobre FUNCIONES Y LIBRERÍAS

Es fácil imaginar que todos los programadores andan haciendo funciones para sus programas… y tampoco es difícil imaginar que muchos de ellos están escribiendo las MISMAS funciones, cosas que casi todo el mundo necesita. Por ejemplo, una manera de ordenar una lista de números.

Sería estupendo que los programadores compartieran sus funciones y que pudiéramos incluirlas en nuestros programas (**importarlas**). Pues así es.

Antes te dije que las funciones son como habilidades: fregar, barrer, limpiar el polvo, etc. y que tú las tenías "importadas" o escritas en tu programa para llamarlas cuando te parezca oportuno.

Algo muy útil es agrupar esas funciones por "temas" en paquetes que llamamos **librerías** de forma que puedes importar librerías completas llenas de funciones que invocarás o no según tu necesidad.

En nuestra metáfora:
LibreríaHogar, que tiene las funciones: barrer, fregar, limpiar el polvo, etc.
LibreríaCarpintería, que tiene las funciones: cortar, atornillar, lijar, etc.

En el mundo de la programación tendrás librerías para matemáticas, estadística, para controlar distinto hardware, etc.

Programadores profesionales y aficionados están constantemente creando, ampliando, modificando librerías y es muy interesante estar al día. Si te acabas metiendo en este mundo visita foros de programación y lugares como Github.

En Scratch el equivalente a una librería sería una **extensión**, lo tienes en el apartado MÁS BLOQUES y si pruebas a añadir una de las que se proponen, verás que aparecen bloques nuevos que "esconden" un pequeño hilo de programación (no accesible desde aquí) implementando la función correspondiente.

INTERACCIÓN CON EL USUARIO

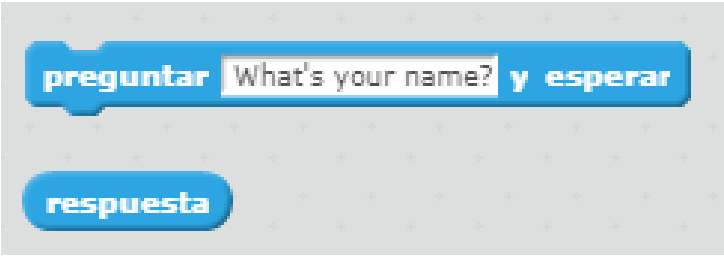
Vamos a hacer ahora uno más facilito… que nos hemos dado una buena palicilla con el último.

Nuestros programas anteriores están bastante limitados por cómo los hemos definido, porque en la práctica, más allá de los choques de objetos y tal, no hacen mucho caso al usuario, pero en Scratch es posible PREGUNTAR DIRECTAMENTE AL USUARIO en tiempo de ejecución, cuando el programa está en marcha.

No sólo eso, sino que también puede tomar la respuesta del usuario y manipularla a su antojo.

Práctica 16. Hablemos con el gato.

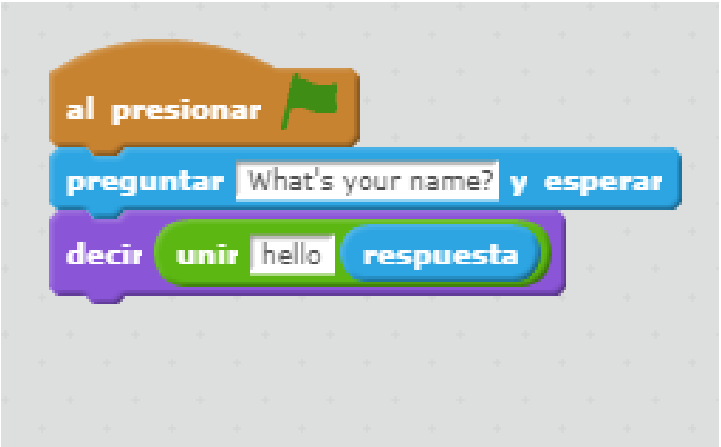
Tanto el bloque para hacer la pregunta como el que nos guarda la respuesta los tenéis en el grupo SENSORES



Fíjate que el bloque que te pregunta, detiene el hilo porque está esperando tu respuesta.

Tenlo en cuenta si tienes varios hilos.

Te propongo algo muy facilito.



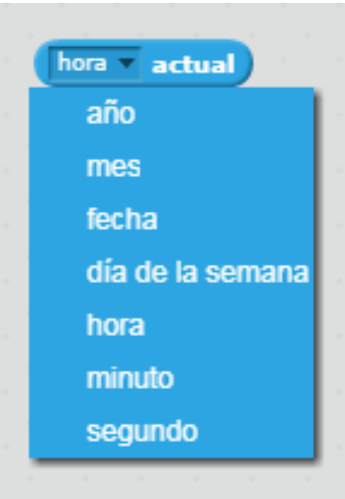
Verás que aparece una caja de texto, como un chat, que espera a que escribas algo (o no), pero que le des a “intro”.

La variable respuesta se SOBREESCRIBE cada vez que haces una pregunta, así que será mejor que guardes esa respuesta en otra variable si la quieres usar luego.

Vayamos guardando respuestas, pues.

Te aconsejo dejar que la respuesta se muestre en el escenario para que veas como los valores se van guardando en las tres variables: respuesta, nombre y fechaNacimiento según va progresando el programa.

La verdad es que podríamos volver a nuestro gato más perspicaz usando un bloque tan curioso como este:



Es divertido ver cómo funciona.
Toma la hora del reloj de tu ordenador.

Sobre el formato de la fecha, el día de la semana… bueno, ¿sabes qué? Míralo tú mismo. Es sencillo, métele el bloque al gato para que lo diga… y a ver qué dice.

De esta forma podrías poner al gato que te dijese tu edad, como la resta del año y la fechaNacimiento. Sería un programa que funcionaría siempre sin tener que actualizarse, no como si pones el año en el que me lees menos fechaNacimiento, que solo acierta este año.

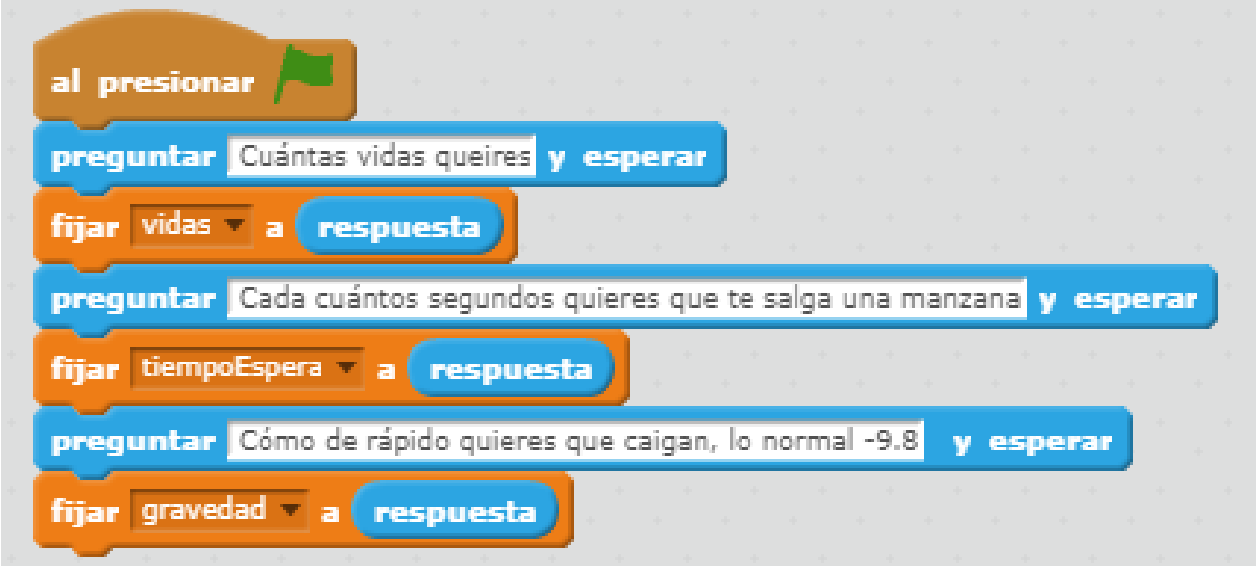
EXTRA: Con todo lo que sabes te desafío a…

- Haz que el gato te pregunte el nombre y guárdalo
- Haz que te pregunte el año de nacimiento y lo guarda
- Haz que calcule tu edad
- Si eres mayor que 18 que te diga una cosa
- Si no que te diga otra.

De la misma forma muchos de los programas que hemos ido haciendo podrían haber comenzado con unas cuantas preguntas por parte del gato para ajustar los valores que elegíamos nosotros como programadores.

Por ejemplo, en el juego de las manzanas que caen, recuerda que elegíamos la gravedad, el número de vidas inicial y el tiempo entre clones.

Sería fácil dejarle la capacidad de elegir eso al usuario en cada ejecución.

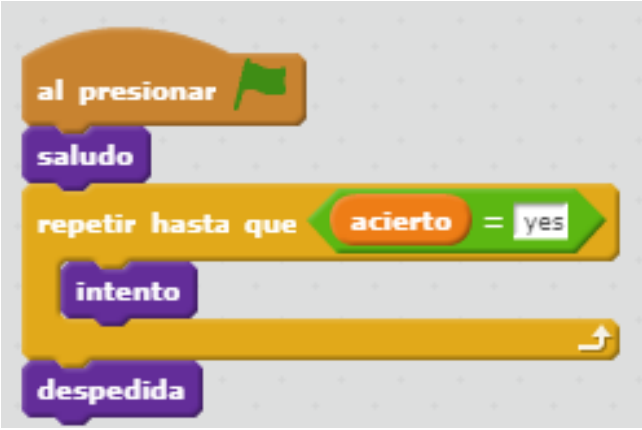


Después claro, tendríamos que llamar a la función generadora de clones así

Esta tipo de introducción puede hacerse al presionar la bandera y después de terminar enviar un mensaje de inicio para que el programa comience su ejecución normal.

Práctica 17. El gato nos adivina un número

El gato nos saluda, nos pide que pensemos un número e intentará adivinarlo.



¿Qué te parece esto?

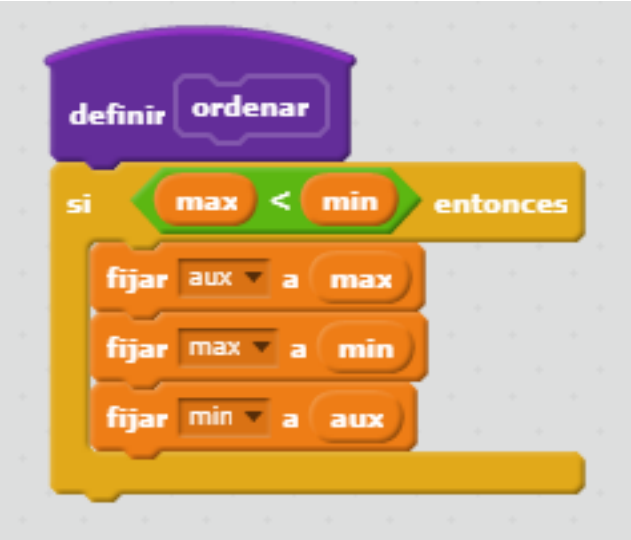
¿Una tontería?
¿Un programa vacío?

Es la estructura lógica de lo que queremos hacer, ahora tendremos que programar esas funciones una a una tranquilamente, pero es todo mucho más claro.

Empecemos por la función saludo

Fíjate que gracioso, que incluso tenemos un bloque para saber el nombre de usuario en la comunidad Scratch de quien está ejecutando nuestra aplicación.

Guardo los extremos que me da de pista el usuario, pero como igual me los da cambiados, voy a definir otra función para ordenarlos y así aprendemos más cosas.



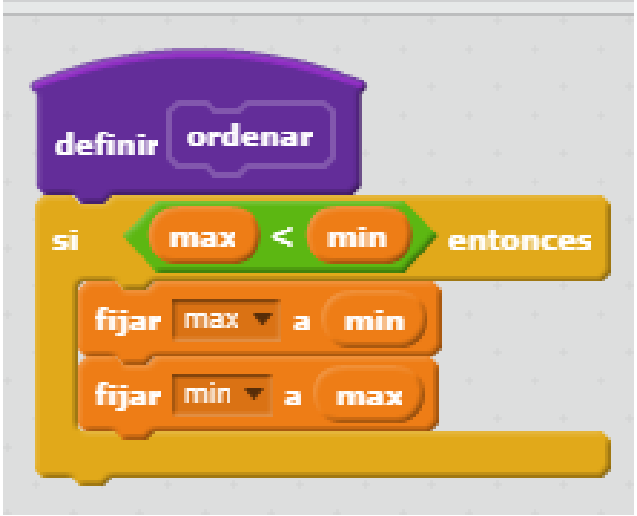
Esta función sólo hace algo si el máximo es menor que el mínimo.

En ese caso tenemos que intercambiar los valores de las variables máximo y mínimo.

Piensa en que cada variable es una caja y que guarda un valor. Como no puedo hacer el intercambio de las dos variables en un solo paso, necesito una caja extra. Te pondré el valor de las tres variables en cada paso.

Por ejemplo si nos dicen que min es 6 y max es 4
Antes de empezar
max 4 min 6 aux (sin asignar)
Paso1 (fijar aux al valor de max)
max 4 min 6 aux 4
Paso2 (fijar max al valor de min)
max 6 min 6 aux 4
Paso3 (fijar min al valor de aux)
max 6 min 4 aux 4

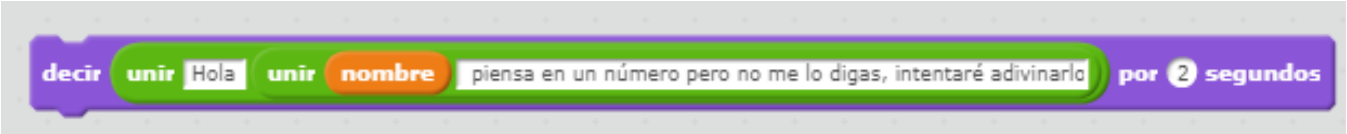
Qué pasaría si intentamos hacerlo sin la variable auxiliar, digamos



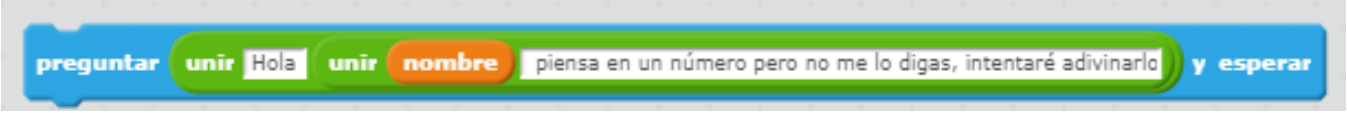
Antes de empezar
max 4 min 6
Paso1 (fijar max al valor de min)
max 6 min 6
Paso2 (fijar min al valor de max)
max 6 min 6

Después del primer paso, hemos perdido el valor 4, por eso necesitamos guardarlo en aux.
Pruébalo y lo verás.

Como habrás visto ya por todo lo que llevamos juntos, las posibilidades de ir introduciendo mejoras son ilimitadas. Por ejemplo, mi programa es un poco “cagaprisas” sólo me deja dos segundos para pensar el número…



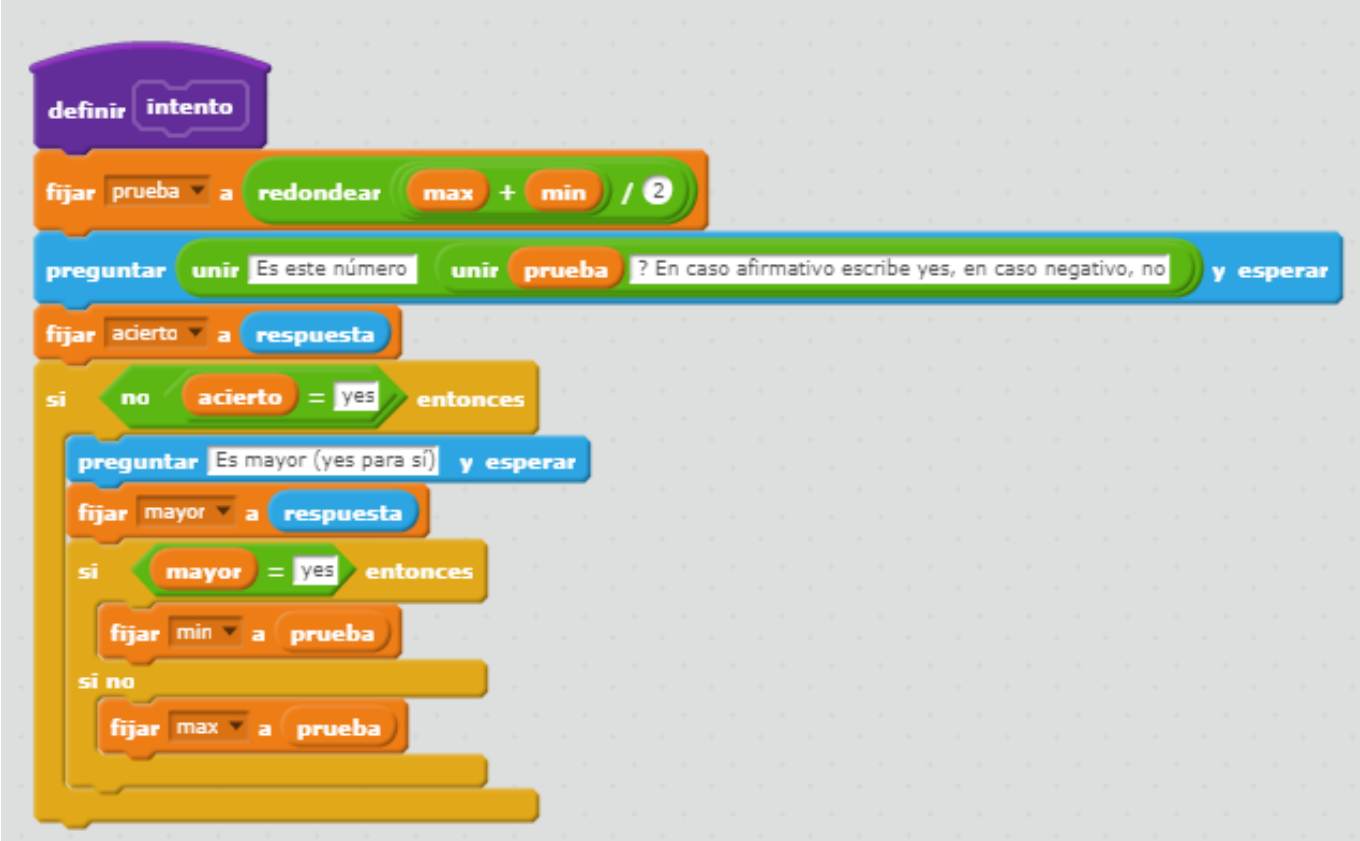
Puedes usar este truco para ser más “user-friendly”.



De esta forma el programa espera a que pulses intro para seguir en lugar de fijar un tiempo. Por supuesto deberías advertírselo en el texto (pero es que ya no me cabía en la hoja). En la variable respuesta se guarda un valor nulo, pero no nos importa, cuando necesitemos otro valor, preguntaremos de nuevo.

NOTA: A estas alturas te habrás dado cuenta de que en las variables estamos guardando números o texto a nuestro antojo sin especificarle nada a Scratch y sin que se moleste por ello. Si sigues en esto de la programación y comienzas a usar otros lenguajes, verás que esto de los **tipos de variables**, no se reduce sólo a estos dos, y que pueden llegar a ser un dolor de cabeza. De momento le agradecemos a Scratch la liberación de este lío, que somos principiantes… pero quedáis avisados para el futuro.

Ahora nos tocaría la función “intento”, la madre del cordero.



La función calcula la media entre el máximo y el mínimo y pregunta si es ese el número. Guarda la respuesta del usuario en la variable acierto
En el caso de que sea yes, no hace nada, porque el bucle con condición de paro del hilo principal tiene como condición que pare cuando acierto sea yes, así que allí se resolverá todo.
Si no es yes la respuesta es porque el número será mayor o menor que el que ha elegido el usuario. Así que cambia el máximo o el mínimo, a ese valor, según corresponda, y el bucle del hilo principal volverá a llamar a esta función para ver si hemos acertado.

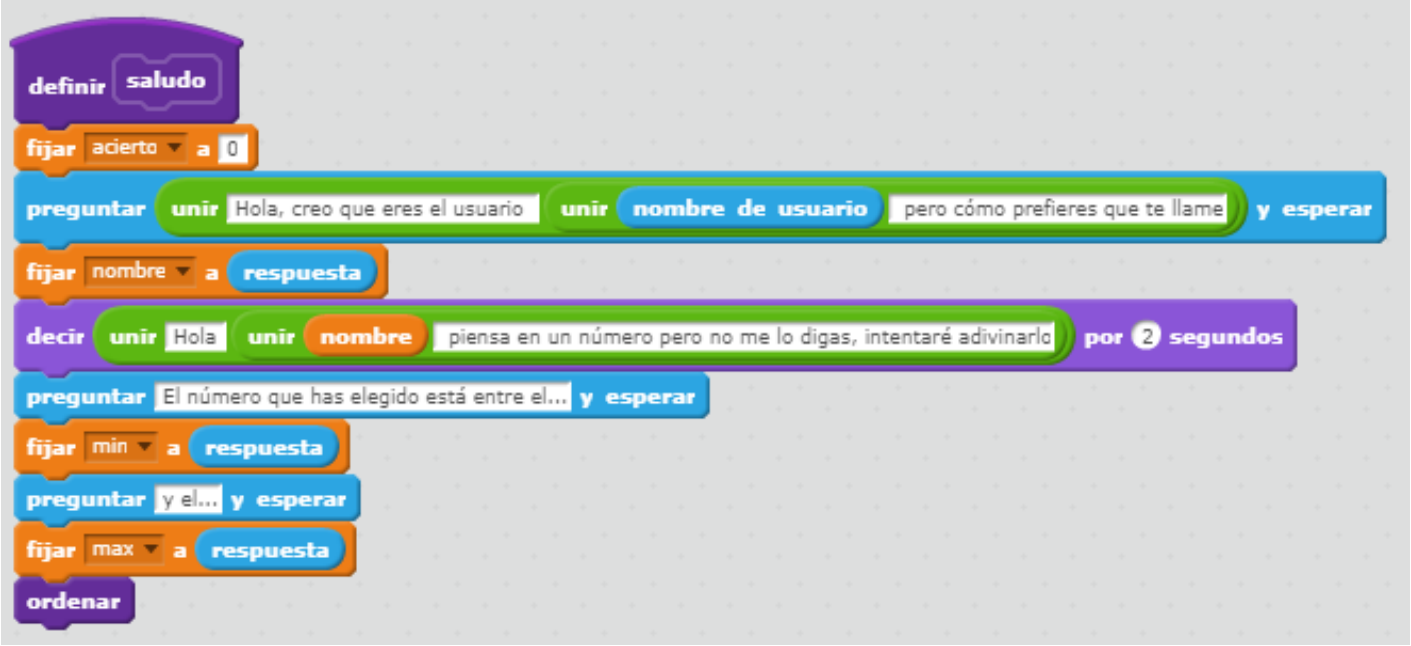
La he probado y va de fábula… la primera vez. La segunda vez, me ha dicho que el número es el mismo que tenía antes, ¿cuál es el problema? Pues que no he inicializado las variables, así que como terminé la ejecución anterior con acierto igual a yes… pues ahora nada más empezar, ya estaba a yes, así que se ha parado el bucle.

Por eso te decía hace unas páginas que es una buena costumbre inicializar las variables en cada ejecución. Puede que sea innecesario, pero no cuesta mucho y te soluciona este tipo de problemas.

Aquí podría bastar con poner acierto a valor no, porque esa es la única variable que se "cuestiona" en un condicional antes de asignarle un valor en la ejecución presente. Me refiero al principio del bucle "repetir hasta que acierto = yes".

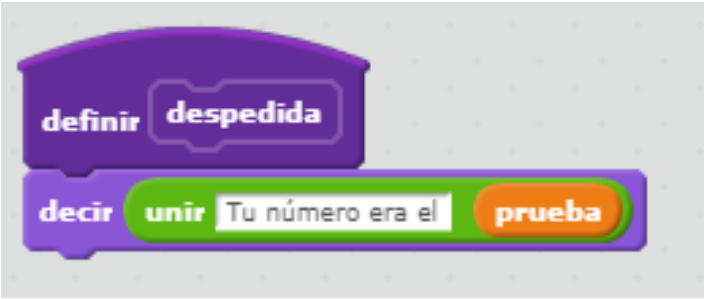
Podríamos pensar en ponerlo en la función intento, pero no vale, porque antes de que se llame a esa función se evalúa la variable acierto en el bucle.

Así que hay que hacerlo en la función saludo, como lo que evalúa es si tiene el valor "yes", en realidad puedes inicializarlo a cualquier otro valor: "no", a cero, a 27… pero tienes que sobreescribir el valor "yes" que se ha quedado puesto de la ejecución anterior.



Para despedirnos puedes hacerlo tan florido como quieras, ya sabemos mucho…

Lo más sencillo



Práctica 18. Le adivinamos un número al gato.

En este caso le toca al gato pensar un número y darnos pistas.

Vamos a hacerlo de otra manera, en lugar de hacer los bloques como saludo, intento y despedida. Voy a hacer un bucle infinito de intentos del que saldremos cuando acertemos deteniendo todo y enviando un mensaje que active un hilo final. A ver si os gusta.

Empezamos porque el gato nos pregunte entre qué números queremos que escoja.

Ahora lo suyo sería ordenarlo también como acabamos de hacer… aunque me da pereza volver a escribir el código, pero…

> **NOTA**: Una estupenda actitud para encarar la tecnología es la siguiente
>
> - ¿Este problema que tengo yo es algo extraño o que me parezca que nadie lo debe haber tenido antes?
>
> Si la respuesta es no, es muy fácil que ya estén implementadas soluciones. Gasta cinco minutos en buscarlas si la tarea no es demasiado fácil. Te ahorrará tiempo. Si es fácil, hazla y punto ;)

Llevado a nuestro caso. Esto de reutilizar código de un proyecto Scratch a otro… no es algo muy descabellado, ¿lo habrán contemplado ya? La respuesta es que sí.

Se llama **mochila** y está en el borde inferior de la ventana. Despliégala primero y arrastra ahí el código que quieras tener disponible. Abre otro proyecto y mira en la mochila, ahí tendrás el código que guardaste y podrás arrastrar ahora a este nuevo proyecto.

Dos detalles.

- Se puede guardar más de un hilo
- Si tienes abiertas dos ventanas con los dos proyectos, tendrás que guardar en la mochila el código del proyecto previo que quieras compartir, recargar la ventana del nuevo proyecto y ya te aparecerá allí disponible.

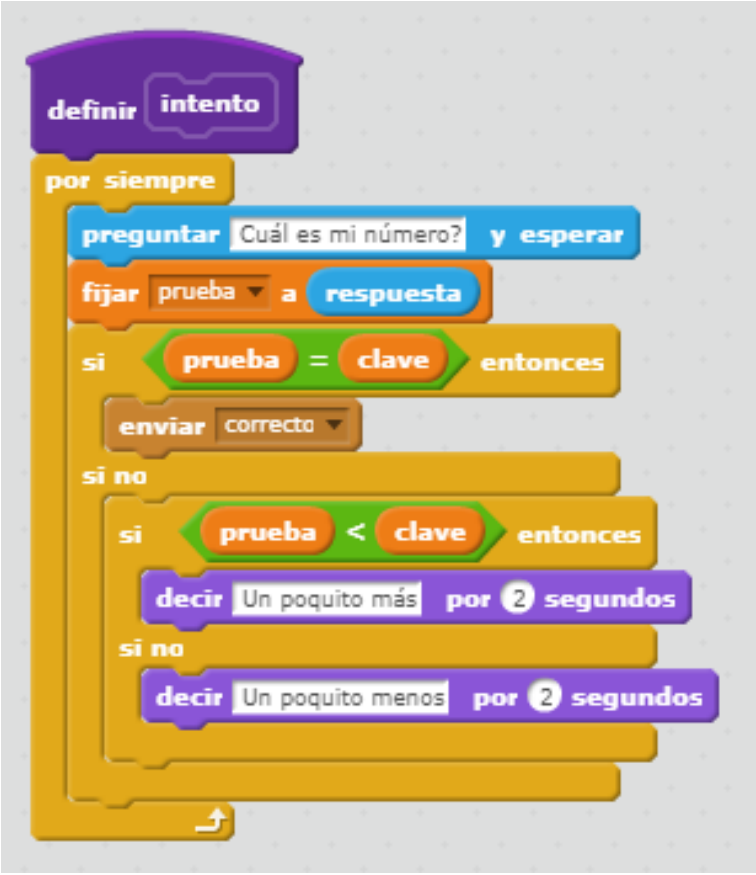
Así que empezamos. La estructura general será algo como esto

La función saludo

Nos pregunta el límite inferior y el superior entre los que tendrá que elegir el número.
Los ordena, por si se los hemos dado mal
Elige un número al azar entre esos límites

Inicializamos prueba a algo que no es un número para evitar que se nos quede con el "número correcto para la vez siguiente", aunque en este programa eso no da error, porque prueba toma valor del usuario antes de ser evaluada en un condicional. Pero mejor prevenir.

La función intento



Nos pregunta el número
Si acertamos manda un mensaje para activar el estado final del programa
Si fallamos nos indica si es mayor o menor y vuelve a preguntar, infinitas veces...

Para despedirnos

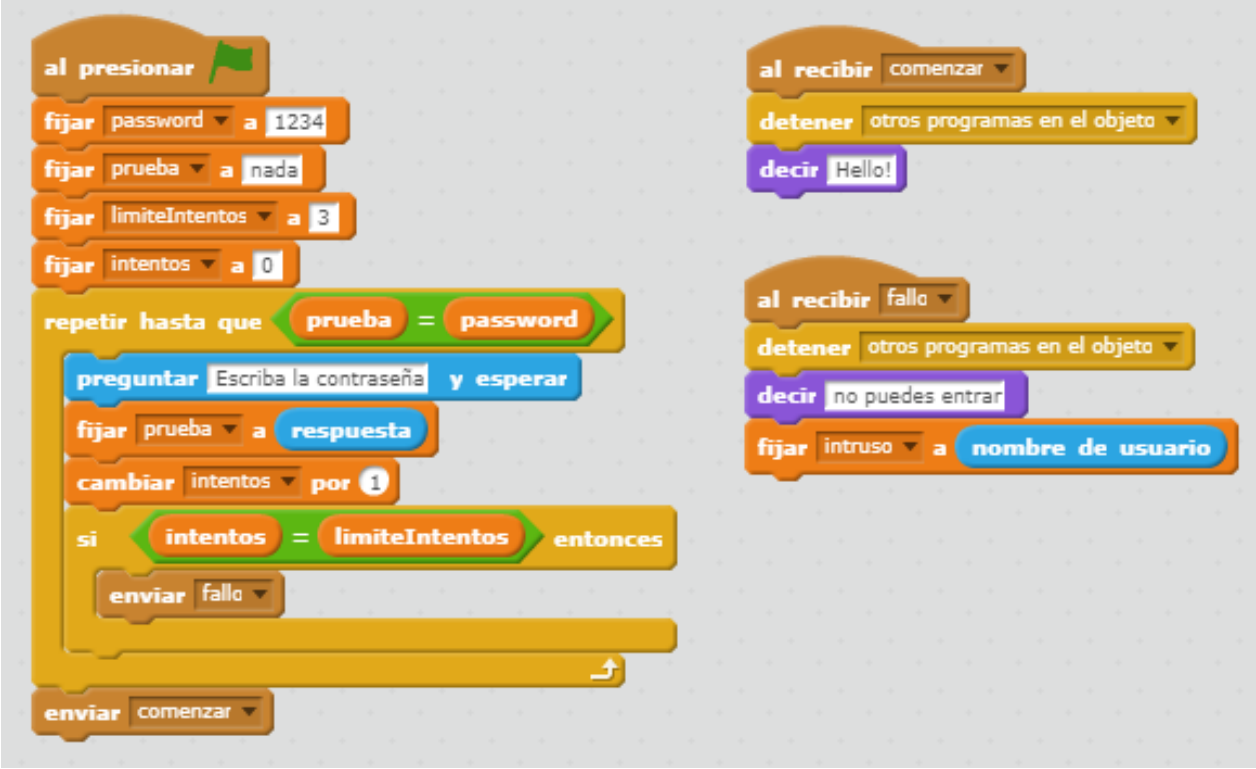
Nos dice que muy bien y apaga a todo el mundo… menos a este hilo, en caso contrario desaparece el bocadillo de “MU BIEN”.

EXTRA: Una mejora pensable sería limitar el número de intentos. Para eso podemos usar una variable que funcionara como las vidas de un juego, que se actualizara en cada intento y que enviara un mensaje de game over si alcanza su tope. ¿Fácil?

Práctica 19. Pedir contraseña

Algo que puede hacer antes de cada programa es pedir una contraseña, es un caso particular de lo que hemos hecho antes para averiguar un número…

Sencillo, ¿verdad? Ahora pongamos un límite de intentos



Fíjate en la macarrada de meter al usuario en una variable como intruso…

¿Qué te parece si, ya que le tenemos guardado, no le dejamos entrar cuando lo vuelva a intentar al dar a la bandera?

Simplemente antes de empezar, pongamos un condicional a ver qué usuario es el que está entrando.

Prueba a fallar una vez y verás como no te deja.

El hecho de tener las variables a la vista muestra muy bien el funcionamiento. Fíjate bien.

Curiosamente, esto que nos ha pasado un par de veces de tener almacenado un valor que no nos interesaba de una ejecución anterior, ahora nos resulta una ventaja.

Así es la tecnología en general, los objetos y los sistemas tienen características que no son buenas o malas en general, sino apropiadas o no para la aplicación que estamos pensando.

Seguro que ya estás pensando en que un intruso es poco, que estaría bien poder almacenar más de uno. Pero claro, ¿cómo lo hacemos? ¿Con cuántas variables? ¿Cómo decirle al programa que cuando ya ha escrito uno, vaya a la siguiente variable?

LISTAS

Nuestro anterior problema se soluciona con listas.

En programación os encontraréis cosas que sirven para simplificar, como nos ha pasado con las funciones.

No es que no se pudiera hacer de otra manera, pero esta es mejor.

Las listas son un conjunto de valores, ordenado (hay un primer elemento, un segundo, etc.)

Usamos listas cuando lo que estamos guardando tiene cierta relación entre sí y es más interesante que guardarlo cada elemento como una variable separada.

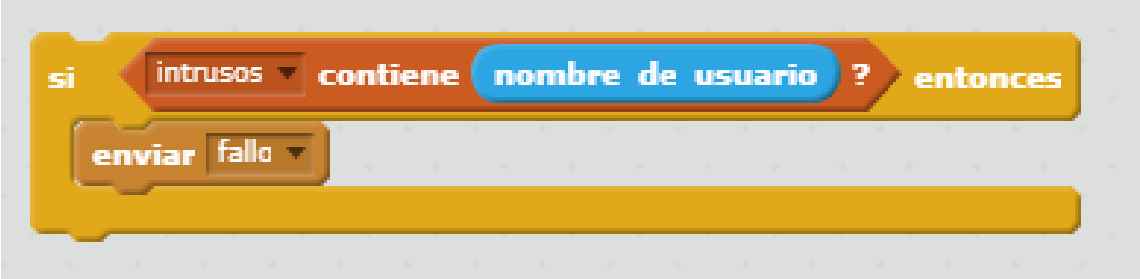
Hay estructuras de datos más complejas que las listas en programación, ya las verás en el futuro, pero en Scratch con las listas ya podemos hacer muchas cosas.

Sigamos con el ejemplo anterior, vamos a crear una lista que se llame intrusos, puedes hacerlo en el grupo DATOS, donde están las variables.

Verás que aparecen un montón de bloques

Añadir es poner un elemento más al final de la lista
Puedes **borrar** un elemento concreto por su lugar en la lista, puedes borrar el último o la lista completa.
Puedes **insertar** un elemento en una posición concreta en lugar de simplemente añadirlo
También puedes **reemplazar** un elemento
Usar elementos concretos
Saber la **longitud** de la lista
Preguntar **si algo está contenido** en nuestra lista (p. Ej. en nuestro caso si el usuario que llega es un intruso ya listado)
Y, por último, **esconder o mostrar** la lista en el escenario.

En nuestro caso, sustituiríamos el condicional que verificaba la variable intruso por



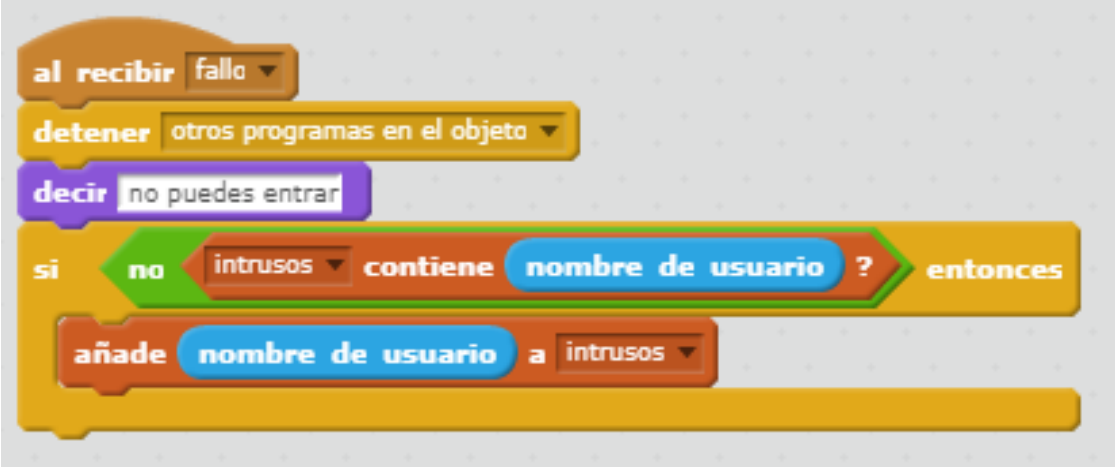
E iríamos añadiendo intrusos así

En lugar de usar “fijar intruso a nombre de usuario”.

Espera, ¿lo has probado contigo mismo, varias veces?

Es que funciona mal… me inscribe una y otra vez en la lista.

Mejor así:



Prueba con compañeros y amigos a ir llenando esa lista desde sus distintos usuarios.

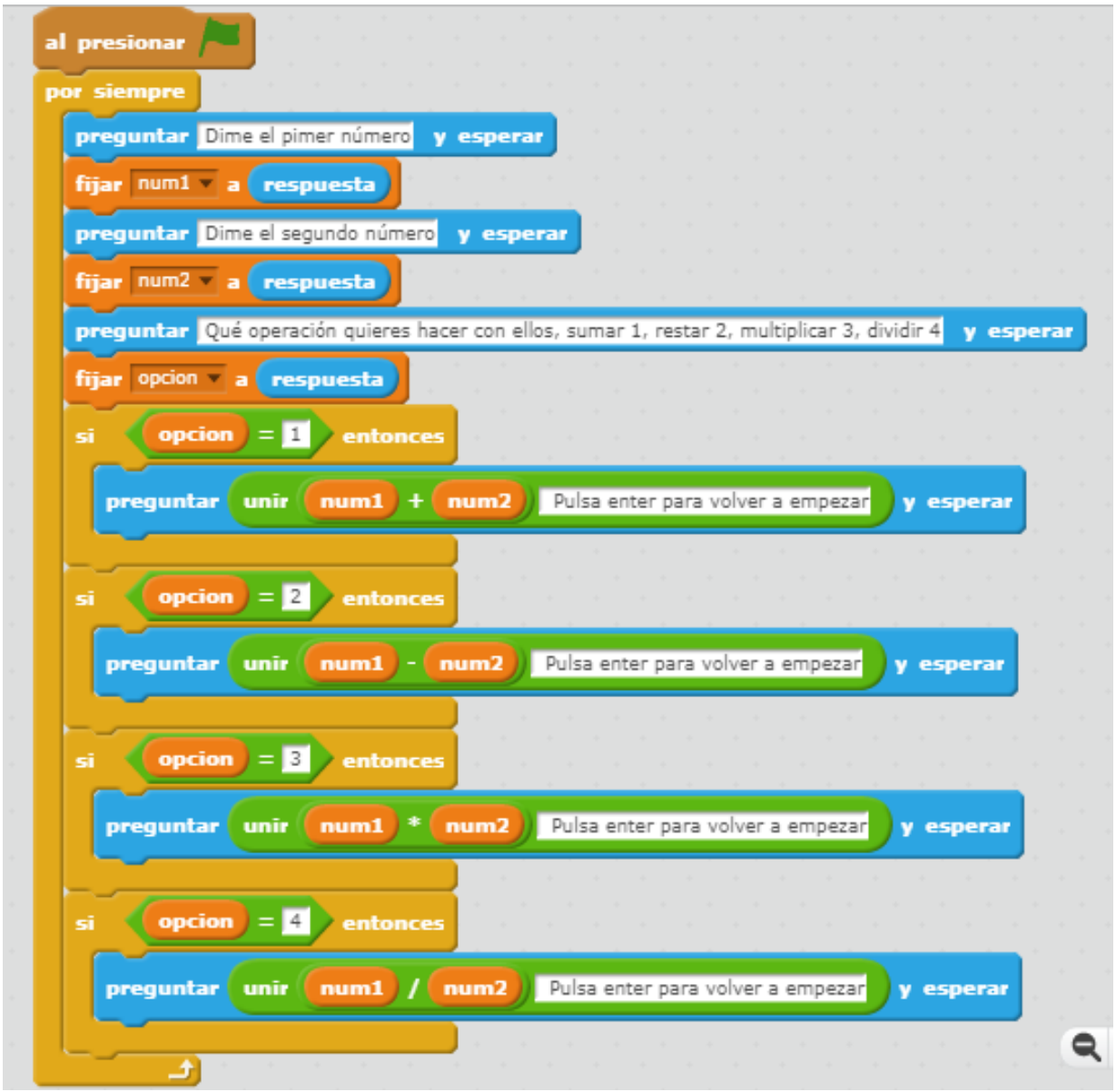
NOTA: Probar el código contemplando todas las situaciones posibles es la única manera de saber que es correcto. Diseña tu batería de pruebas y pásasela a tu código sin piedad.

Práctica 20. Muchos programas en uno

A veces mis alumnos me preguntan cuántas preguntas tiene un examen, y lo que yo me pregunto es ¿cuál es la diferencia entre un examen de diez preguntas y un examen de una pregunta con diez apartados?

Pues aquí podemos jugar a lo mismo. Un programa que tiene dentro cuatro programas.

Algo tan sencillo como una calculadora



Aunque así nos queda el código más bonito, ya sabes que es más eficiente si haces los condicionales anidados, porque tal y como lo hemos hecho, pasará por todos los condicionales, y si los anido, no necesariamente.

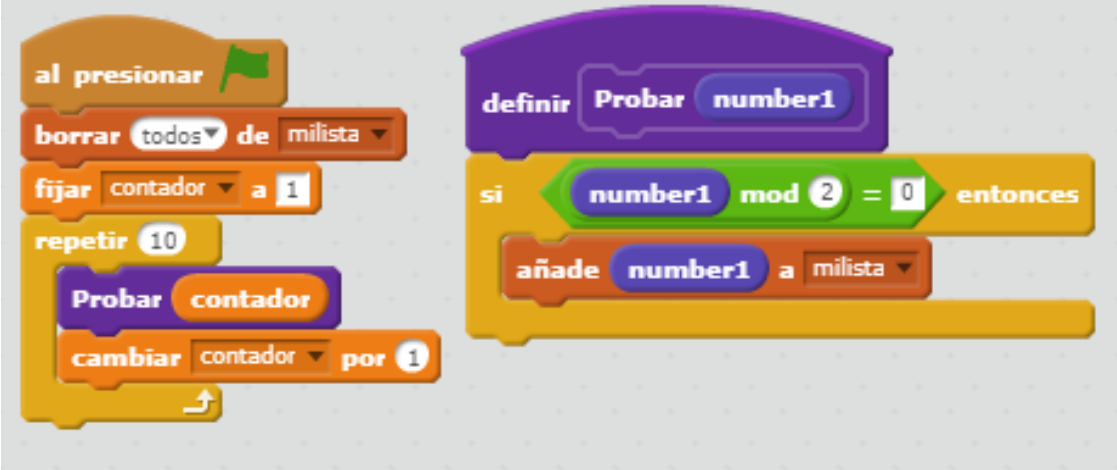
Pero también lo podemos hacer usando mensajes que disparen hilos distintos, incluso en objetos distintos. Así que en la práctica sería como tener varios programas de los que se elige uno en tiempo de ejecución por el usuario.

Usando los bloques detener, me ahorro tener que recorrer todos los condicionales restantes cuando acierte con la opción.

A los que visiten este manual como profes, buscando cosas para sus alumnos y sepan de programación, les recordará esta disposición a la estructura **switch-case** que se usa cuando se quiere inspeccionar una variable para casos disjuntos (sin intersección) y se añade un comando **break** después del código de cada caso.

Los condicionales son más interesantes quizá cuando buscamos que la variable esté en cierto intervalo o entre ciertos valores, más que si toma algún valor de un conjunto discreto, como es nuestro caso: uno, dos o tres.

Práctica 21. Hacer una lista con los números pares



Casi todo cosas que ya hemos visto:

El programa principal borra la lista, por si hubiera cosas que no queremos de antes. Pone el contador a uno y repite diez veces una llamada a la función Probar, a la vez que actualiza el contador.

La función probar comprueba una condición con la función módulo, que arroja el resto de dividir el primer número por el segundo. Aquí el primero es el número que se le pasa a la función, el valor del contador en cada vuelta, el divisor es dos (porque queremos saber si par) y comprobamos que el resto sea cero.

En caso afirmativo, el número es par, lo añadimos a la lista… y listo.

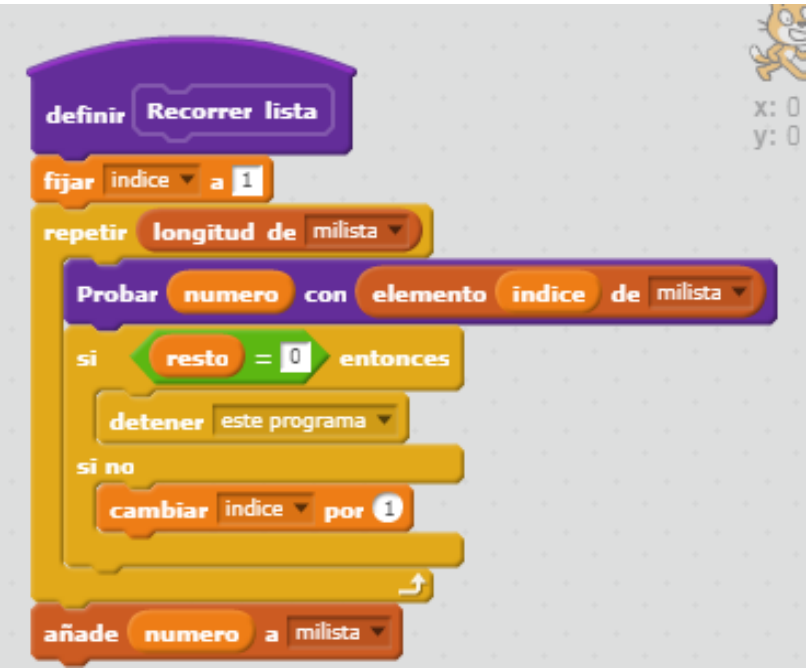
Práctica 22. Hacer una lista con los 20 primeros números primos

Que no se preocupen las agencias gubernamentales, que vamos a hacer poquitos (ya sabéis que son fundamentales en seguridad)

Lo primero borrar la lista claro. La empezamos con el 2 y vamos a comprobar explícitamente el 3.

Repetiremos, pero no hasta que hayamos mirado veinte números a ver si son primos, sino hasta que hayamos conseguido 20 primos en la lista.

"Recorrer lista" es la función que irá viendo si el número que estamos probando es divisible entre los primos de la lista, uno por uno.



Creamos la variable índice para poder recorrer la lista.

Repetimos hasta que la lista se acabe

Probamos si el número es divisible (ahora veremos esa función)

Si es divisible por alguno, interrumpimos el bucle, porque ya no será primo y si no lo es pasamos al siguiente de la lista.

Si llegamos al final del bucle (de la lista) y no hemos salido fuera por encontrar que era divisible… es porque es primo respecto a los que tenemos, así que pa'dentro a la lista.

La función Probar tiene dos argumentos, el número que estoy probando y el elemento de la lista que quiero ver si puede dividirlo de forma exacta

Usa la función mod que ya sabéis que es módulo o residuo, vaya, resto de la división, y lo guarda en la variable resto que será evaluada dentro de la función que ha llamado a esta.

Probadlo y veréis cómo funciona.

Si os perdéis un poco porque los números varían muy rápido, podéis poner esperas en algunos lugares para ver cómo van cambiando las variables, los índices y demás.

En todo caso, os aviso de que no está yendo muy rápido…

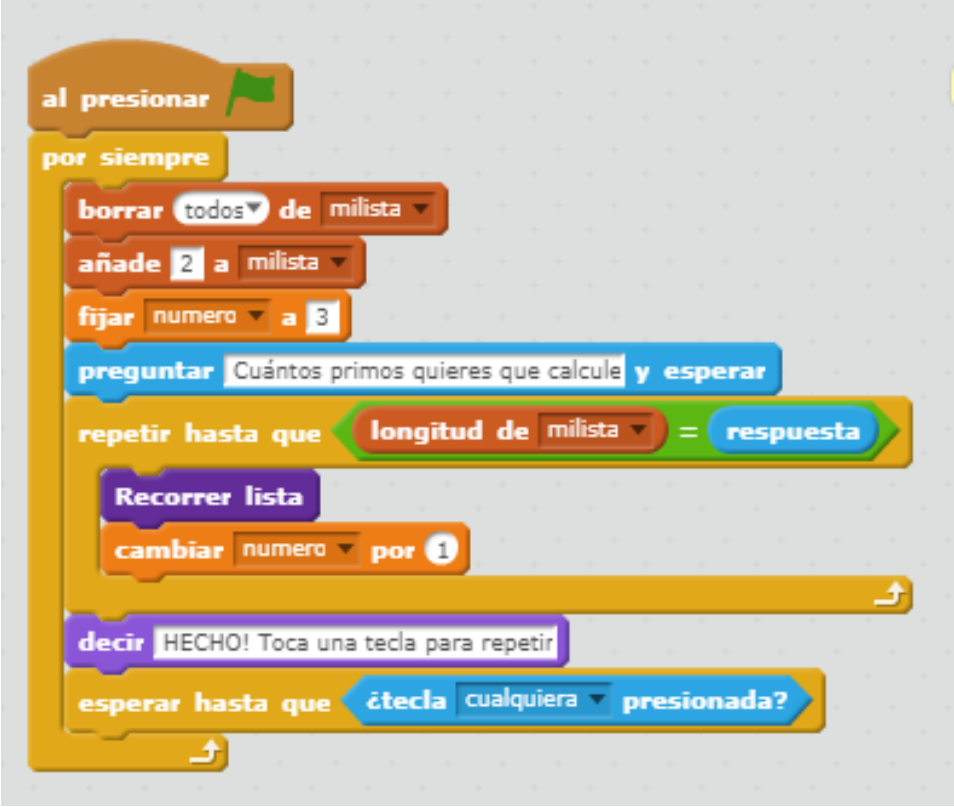
Aún no habéis puesto **el modo TURBO**

Id al menú editar y allí tenéis la posibilidad de activar el modo TURBO. Si lo hacéis veréis que os genera la lista de forma inmediata. Scratch tiene ligeramente ralentizados los procesos para que se vean los flujos de datos, las estructuras de control y demás.

En alguna ocasión esto me ha dado problemas de sincronía al trabajar con varios objetos. Me pasó con un murciélago que hacía volar hacia una ventana de un fondo. Yo quería que el tiempo de vuelo se emplease en ir reduciendo el tamaño, pero se deslizaba a una velocidad mientras se encogía a otra. Esto era porque el bucle que había puesto para ir reduciendo el tamaño iba "despacio" y acababa metiendo más tiempo del necesario. Puse el modo turbo y todo solucionado.

Práctica 23. Calcular una lista de primos de cualquier longitud

Sólo por refinarlo un poco más.



Limpio la lista
Añado el número 2
Escojo el 3 para empezar
Pregunto al usuario cuántos quiere

Hago el bucle teniendo como objetivo que la longitud de la lista sea la que dijo el usuario.

Finalmente espero tecla para repetir si se quiere.

Las funciones “Recorrer lista” y “Probar” son iguales que antes, sólo he cambiado el hilo principal.

Una ocasión estupenda para usar **la mochila**, y traerte esas funciones a este proyecto ¿verdad?

Ten cuidado cuando lo hagas porque no se te crean las variables automáticamente, asegúrate de que todo está definido y en su sitio.

Práctica 24. Calcular la letra del DNI

En España el documento de identidad lleva un número que acaba en una letra.

No se trata de “ampliar” los números combinando también letras, como en algunos sistemas de matrículas de vehículos, se trata de un **carácter de control.**

La letra se deduce de los números anteriores.

Cuando le das tu DNI a un operario y lo introduce en un ordenador, un programa calcula la letra con una fórmula y si coincide con la letra que tú le diste, lo da por bueno, si no, sabe que te has equivocado al decirlo (o él al teclearlo), aunque no sepa donde, y te pedirá que se lo digas de nuevo.

La fórmula consiste en dividir el número por 23 y observar el resto, según salga el resto te asignan una letra de esta lista:

Tomado de la página del Ministerio del Interior

RESTO	0	1	2	3	4	5	6	7	8	9	10	11
LETRA	T	R	W	A	G	M	Y	F	P	D	X	B

RESTO	12	13	14	15	16	17	18	19	20	21	22
LETRA	N	J	Z	S	Q	V	H	L	C	K	E

Si estáis pensando en que voy a escribir 22 condicionales estáis chalaos… ahora que sabemos listas, yo creo que es mucho más fácil, ¿no? El resto es casi el índice de la lista (menos uno). Ñam, ñam… yo creo que ya está.

¿Qué tal chavales? ¿Ya habéis hecho esto?

Miiiiiira que os lo tengo dicho:

Si en tecnología os ponéis a hacer algo, que no sea una cosa rara, y os resulta muy largo o pesado, seguro que alguien ya lo ha hecho más fácil y automatizado…

Y las cosas rocambolescas y ortopédicas… aunque funcionen por carambolas… acabarán cascando y ahora ponte a buscar el fallo.

Perdonadme la maldad de no avisar, pero a ver si así os acordáis para la próxima.

Hagamos las cosas sencillas y elegantes en la medida de lo posible.

Formemos una palabra con todas esas letras y luego preguntémonos por la posición de la letra. Es similar pero nos quedará mucho más lindo.

Escoger la estructura de datos en la que vas a meter la información es importante.

Por ejemplo, la edad es un número, porque querré hacer operaciones numéricas con ella: ver si es mayor que otra, sumar o restar años, hacer una media de edad.

En cambio el número de teléfono… no es un número. Me explicaré. Es un conjunto de caracteres numéricos, pero no representan un valor, no tiene sentido ver si tu número es mayor que el mío o a hacer la media de dos teléfonos. Lo podría tratar como una palabra.

En el caso de las letras del DNI como no voy a hacer nada más con esa lista, puedo considerar que es una palabra también, sin problema.

Mira qué limpito queda usando una “palabra”, lo que en programación se llama una **cadena de caracteres** o un **string**.

Cargamos en la variable letrasDNI las letras que nos dicen en el Ministerio.
Preguntamos para saber el número
Lo guardamos en una variable
Calculamos el resto
Lo usamos como el número de orden (+1) para encontrar la letra adecuada.
Hacemos que el gato lo diga.

Quizá te parezca una cosa rara lo de dividir por 23, la idea es que si cometes un pequeño error (te baila alguna cifra, o cambias una por otra) no te salga la misma letra.

De forma similar, pero con un algoritmo más complicado podrías plantearte calcular los dígitos de control del número de cuenta bancaria, que también los tiene.

En España usamos ahora el estándar internacional IBAN y nos queda así:

ES**12** 3456 7890 **12** 1234567890

Los dos números subrayados son dígitos de control.

RECURSIVIDAD

En programación se dice que una función es recursiva cuando se llama a sí misma y ese proceso se repite varias veces.

Un ejemplo de esto es el factorial, que es una función de bastante importancia en matemáticas, en probabilidad y estadística, por ejemplo.

Con un ejemplo lo entenderás.

5! = 5·4·3·2·1

7! = 7·6·5·4·3·2·1

¿Por qué digo que es recursivo? Fácil

7! = 7·6! = 7·6·5! Etcétera.

> **NOTA:** En programación llamamos "**lenguaje natural**" a cómo hablamos las personas, frente al **código máquina** que es el lenguaje que entiende el ordenador

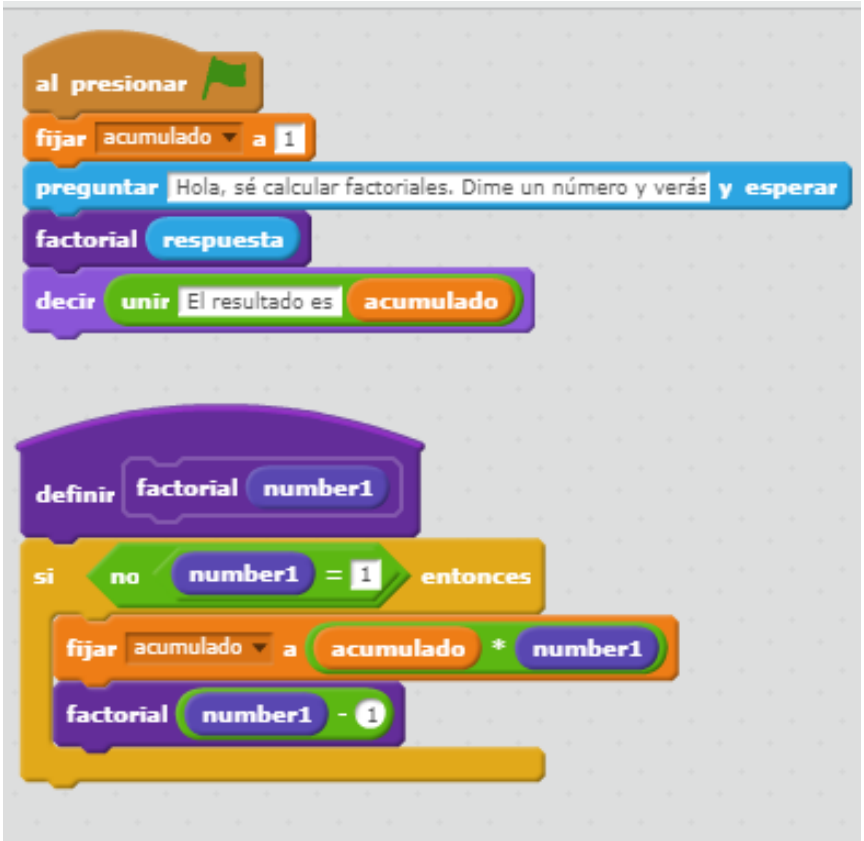
Bueno, pues si te digo en lenguaje natural la conversación sería…

- Oye, Paco, ¿cuánto es el factorial de 7
- Pues 7 por el factorial de 6
- Ah, muy bien, y ¿cuál es el factorial de 6?
- Pues 6 por el factorial de 5...

Así, hasta llegar al factorial de 1 que es 1.

Vamos a programarlo

Práctica 25. Calcula el factorial de un número



Fíjate cómo hemos inicializado la variable acumulado al valor 1, típico cuando luego queremos ir multiplicando cosas, igual que cuando queremos ir sumando cosas, inicializamos variables a cero.

Es en la función donde se produce la recursividad.

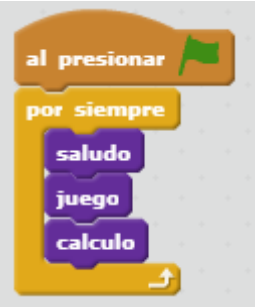
Se pregunta si el número no es el uno. Si lo fuera, la función termina y se va al hilo principal donde se da como valor el acumulado, 1.

Pero si le hemos pedido el factorial de otro número entonces empieza el movimiento.

Por ejemplo, si le pedimos el factorial de 3:

- Como no es el de 1, ejecuta el código
- Actualiza el valor de acumulado a 3·1, que era el valor anterior.
- Y pide el factorial de dos.
- Pero el factorial de dos tampoco es el de 1
- Actualiza el valor de acumulado a 2·3, que era el valor anterior
- Y pide el factorial de 1
- Ahora ya termina y va al hilo principal donde el gato nos dará el valor acumulado.

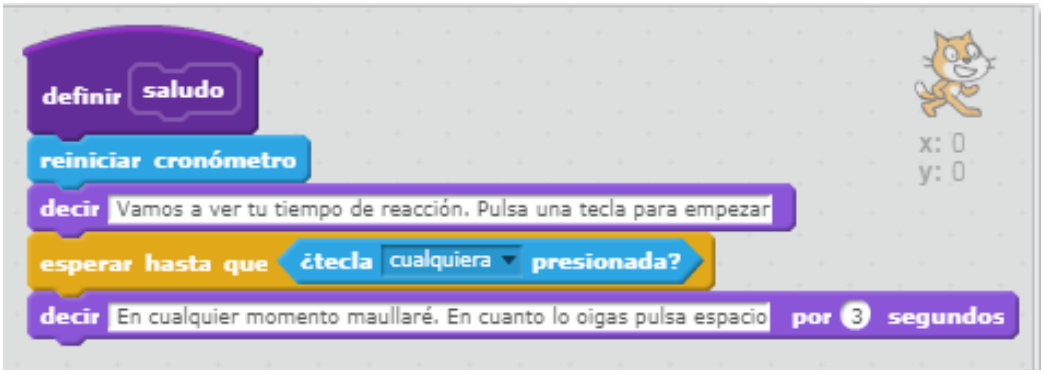
Práctica 26. Calcular el tiempo de reacción



Con esto ya sé que no te digo mucho con estos hilos, jeje…

Bueno, sí, te digo que estructures los programas y los hagas con funciones.

Discúlpame la insistencia.



Una presentación sencilla. Simplemente hacerte notar, una vez más, que puede ser interesante lo de esperar a que se pulsen teclas en lugar de poner esperas de tiempo concretas para que el usuario vaya más tranquilo.

Hace una cuenta atrás emocionante… Espera un tiempo aleatorio, además no empiezo en un segundo para que no pille el ritmo a la cuenta atrás.

Mide ambos tiempos y calcula la diferencia.

En sentido estricto habría unos pequeños retardos debidos a lo que tardan los bloques en ejecutarse, pero comparados con el tiempo de reacción que es de algunas décimas… despreciable

Quizá te sorprenda el condicional, pero es para detectar a los que estén tocando todo el rato el espacio para intentar aumentar sus posibilidades. Se supone que un tiempo de reacción menor que una décima no es posible (en atletismo te descalifican).

Pongo la espera a tocar una tecla para reiniciar el bucle, verás por qué.

La espera de dos segundos es para no tener lo que en robótica se llama "**rebote**" de un botón. Se trata de que mientras pulso y levanto el dedo, el ordenador me lee varias pulsaciones. En nuestro caso, la que me tocaba hacer al escuchar y luego la de volver a empezar.

Práctica 27. Tiempo de reacción “sin trampa”

Para evitar que nos anden pulsando todo el tiempo el espacio podemos implementar la siguiente solución en la que sólo se guardará una pulsación del espacio por partida.

La idea del bucle infinito es para que se pueda jugar una y otra vez sin tener que darle a la bandera.

Dentro de las funciones, tendremos esperas por teclas para volver a empezar.

Ponemos todo a cero.

Esperamos un momento para evitar el rebote con la pulsación de volver a jugar cuando acaben que se detecte como la siguiente que estamos esperando.

Esperamos a que el usuario esté listo.

Enviamos el mensaje empezar que dispara dos hilos:

El primero

Este vigila si se pulsa la tecla y guarda ese tiempo.

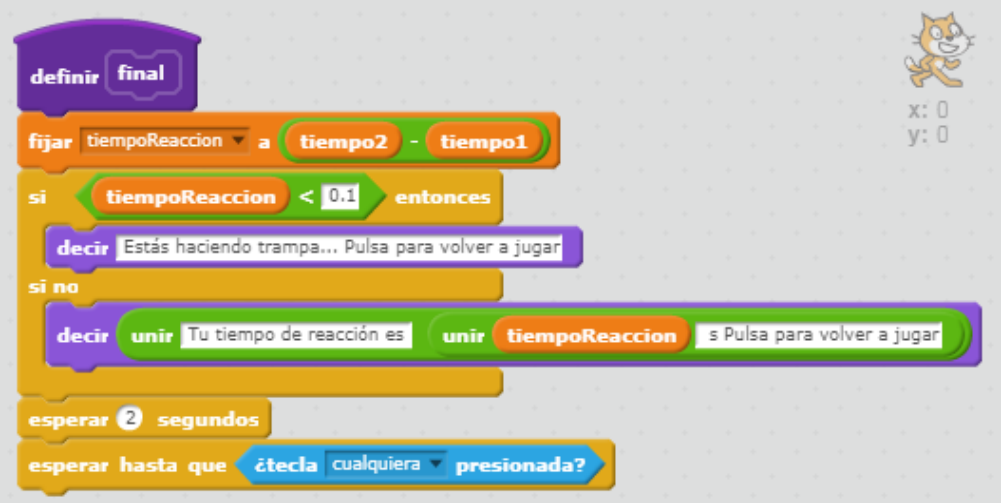
Se detiene para que sólo se pueda dar una pulsación

El segundo

Hace la cuenta atrás.
Espera un tiempo aleatorio
Apunta el tiempo inicial
Hace maullar al gato.

Recuerda que el mensaje empezar lo mandamos con la orden: enviar mensaje y ESPERAR, así que hasta que no terminen los dos hilos anteriores, nuestra función juego NO HABRÁ TERMINADO. Eso nos permite asegurarnos de que, antes de acabar la función, tendremos bien registrado el tiempo1 y el tiempo2.

Por lo tanto, cuando en el hilo principal terminemos la función juego y llamemos a la función final todo irá bien.



Calculamos el tiempo de reacción sin problemas, porque tenemos los datos suficientes al haber concluido los dos hilos que disparó la función juego con su mensaje.
Si el tiempo es menor que una décima, incluido el caso en el que sea negativo porque el usuario pulsó antes de que el maullido se produjera, el gato nos regañará. Si no, nos dará el tiempo calculado.
Después espera dos segundos para evitar rebotes (que la pulsación para el juego se tomase también como esta pulsación para reiniciarlo) y esperamos por si quiere repetir.

Práctica 28. Calibrando el modo TURBO

Ya te he contado que Scratch ralentiza la ejecución de los bucles para que pueda observarse el flujo del programa, y que eso podría darnos problemas.

¡Qué tal si ejecutamos dos bucles largos, por ejemplo, y vemos cuál es la diferencia al ejecutarlo en un modo u otro?

El programa es el siguiente

Pongo un bucle, porque fue lo que me pareció a mí que iba más ralentizado (te dejo a ti, la prueba de tus propias sospechas)

Ejecuto este programa en modo TURBO y esto es lo que veo

He palmado tres segundos… en doscientas iteraciones… 0,015 s por iteración.

Probemos en modo NORMAL… miedo me da.

Pues al final… hoy por hoy, con la versión online, parecen no diferenciarse mucho entre ellos, pero sí que ambos me meten un retardo apreciable.

Así que, tenedlo en cuenta si hacéis bucles largos y queréis sincronía con otros hilos.

Práctica 29. Limitando actividades en el tiempo

Hay muchas razones para limitar actividades en el tiempo, algunas podrían ser

- Tiempo límite para poner una contraseña
- En juegos, para acabar tareas, llegar a lugares, adivinar cosas...

Es un añadido que podéis incluir en cualquiera de vuestros proyectos

He puesto como disparador la recepción de un mensaje, pero podríais poner la bandera verde como iniciador si queréis limitar desde el principio.

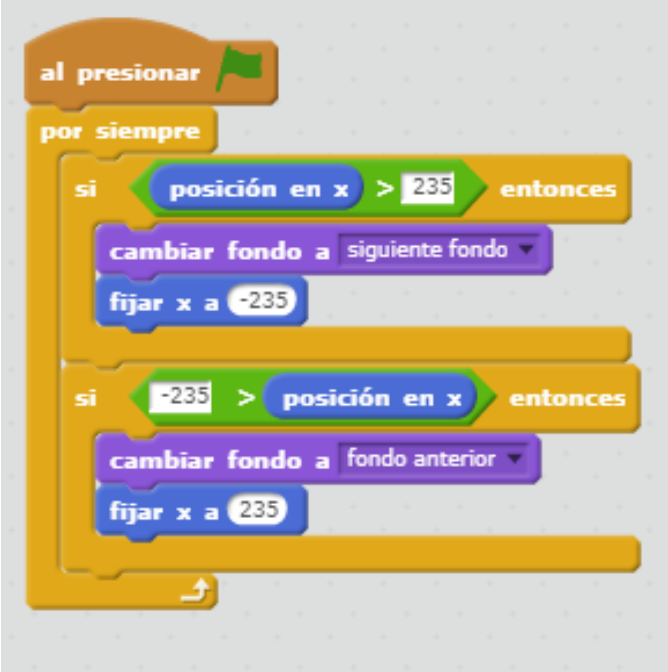
Después podéis enviar un mensaje que active otras tareas y detenga las actuales.

Una forma más sencilla, en un programa más sencillo podría ser un simple bucle con condición de paro y poner dentro la tarea cuya duración quieras limitar.

Prueba a rehacer cualquiera de los programas anteriores, añadiendo esta limitación. Por ejemplo, tratar de adivinar el número que elegía el gato, con un límite de tiempo, por ejemplo.

Práctica 30. Cambiar de fondo cuando el objeto llega al borde

Algo muy típico en programas para los chavales más jóvenes es hacer un juego de plataformas con personajes. En estos entornos es muy útil poder cambiar de fondo cuando el objeto “sale” por la izquierda o la derecha.



Muy sencillo, como verás.

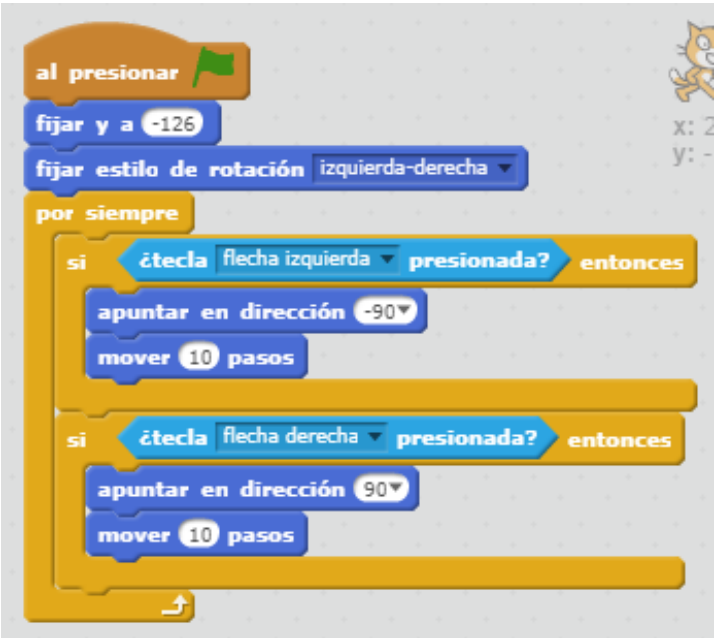
Miro constantemente si el gato está a la derecha, en ese caso cambio el fondo al siguiente y pongo al gato a la izquierda para que parezca que acaba de entrar al fondo.

En el cambio al otro lado hago lo contrario.

Si lo pruebas hasta llegar al último fondo verás que se vuelve circular. El siguiente al último fondo es el primero y el anterior al primero es el último.

Los bloques cambiar fondo admiten también cambiar a un fondo concreto que quieras, no tiene que ser siempre al siguiente o al anterior.

El código del gato podría ser



Ya muy conocido.

Pongo al gato debajo del todo y miro si se pulsan las teclas de izquierda o derecha para apuntar en esa dirección y moverme un poco.

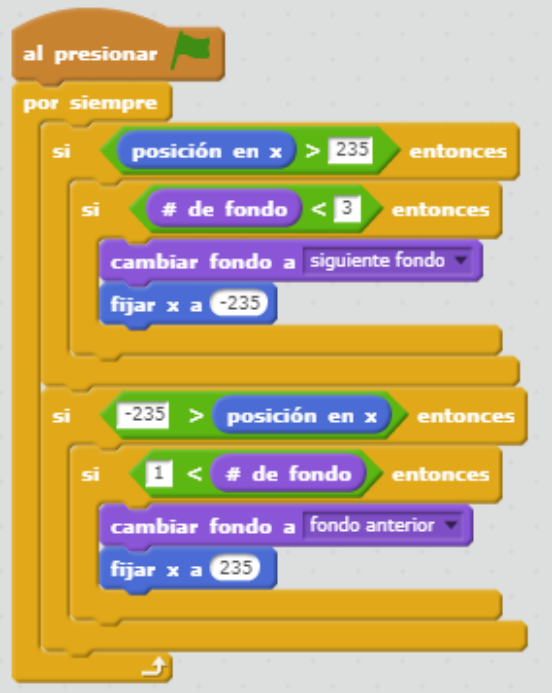
El estilo de movimiento lo pongo sólo de izquierda a derecha que así no se pone boca abajo.

También puedes combinar condicionales para preguntar si estás en el último fondo y que no te deje seguir.

En el grupo APARIENCIA del objeto escenario verás que puedes usar bloques como

Con estos puedes identificar el fondo en condicionales, y si quieres copiarlos para usarlos en el código de algún objeto, puedes arrastrarlo hasta el objeto (en la tabla de objetos, no en el escenario) y se copiarán los bloques en su espacio de programas, como ya hablamos antes.

Por lo tanto, si queréis que no sea circular el cambio de fondos, podemos hacerlo así.



Ya ves que simplemente ponemos una comprobación previa antes de cambiar el escenario en ambos casos, para saber si ya estamos en el primero o en el último.

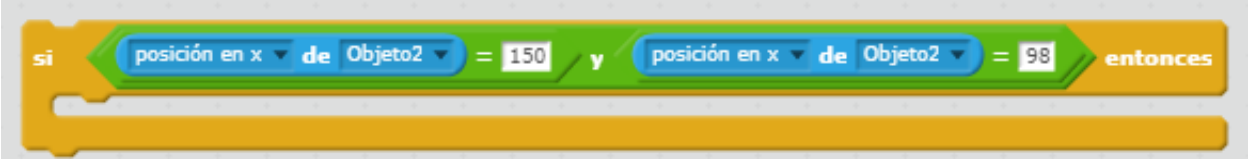
EXTRA: Hay efectos de colores divertidos para el gato, y también los tienes en los fondos. Si exploras los colores podrían simularte distintos momentos del día en un fondo. Una buena baza para contar historias.

Práctica 31. Cambiar de fondo con otros eventos

Podríamos cambiar de fondo usando cualquiera de los eventos que ya conocemos, típicamente para mostrar pantallas de comienzo, fin, etc.

Pero también podríamos querer que cuando un objeto esté en algún lugar concreto de la pantalla cambie ese fondo. Por ejemplo, estoy en un fondo con una puerta, cuando toco la puerta cambia el fondo al interior del edificio.

Para esto podemos usar directamente un condicional sobre las coordenadas del objeto, sabiendo en qué posición está la puerta, pondríamos algo así:



Esto nos obliga a estar en una posición concreta, pero la verdad es que los objetos son tan grandes como para que eso sea fácil de conseguir.

Con objetos pequeños tendríamos problemas en encontrar el pixel justo y podríamos darnos un “margen”.

Uniríamos en el bloque “Y” estas dos condiciones
Valor absoluto de la resta de la posición y el objetivo menor que el margen.

NOTA: Valor absoluto quiere decir que nos da igual que la resta nos salga positiva o negativa, que nos pasemos por arriba o por abajo. Lo que queremos es no irnos más cantidad del margen.

Otra manera de encontrar la puerta sería, poner encima un objeto que fuera la puerta y preguntarnos si nuestro objeto está en contacto con ese otro objeto que es la puerta.

Y otra forma diferente sería mirar si nuestro objeto está tocando un color concreto

Para seleccionar ese color, hacemos click en el cuadro, cambiará el cursor a una mano y hacemos click en el lugar donde esté el color que queramos identificar.

Una forma más específica incluso sería usar este otro bloque, en el que nos preguntamos si hay dos colores que están en contacto, porque quizá haya partes de un objeto que no importe que toquen un límite, pero otras sí. Por ejemplo, si me quiero poner un sombrero en la cabeza pero no en los pies. De nuevo los colores se seleccionan de la misma manera.

EXTRA: Si recuerdas el juego Worms o el de las motos de la película TRON (en un caso no se pueden tocar gusanos entre sí, y en el segundo los objetos iban "pintando" líneas que no podían tocar otros. Para simular eso podrías usar los bloques del grupo LAPIZ para crear los "caminos" de las motos de TRON u otros rastros.

Los bloques son bastante intuitivos, pero querría señalarte estos cuatro:

Borrar sirve para limpiar la pantalla, sería el equivalente a inicializar nuestras variables.

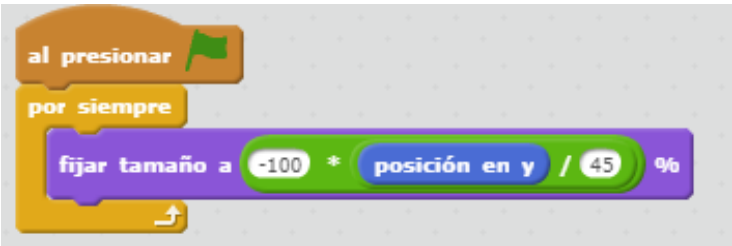
Sellar deja un dibujo de nuestro objeto sobre el fondo. NO ES UN OBJETO NUEVO NI UN CLON. Tampoco ejecuta código y se puede pintar por encima.

Subir y bajar lápiz es porque los objetos van pintando con su movimiento (incluso si están "escondidos") pero sólo si el lápiz está abajo, si está arriba pueden desplazarse sin pintar. Muy importante si queremos hacer figuras que no estén conectadas.

Práctica 32. Perspectiva

Algo divertido es añadir un poco de perspectiva a nuestras historias, podemos hacerlo de una manera tan sencilla como sobreentender que si está más arriba en la vertical es porque está más lejos… y si está más lejos pues se verá más pequeño.

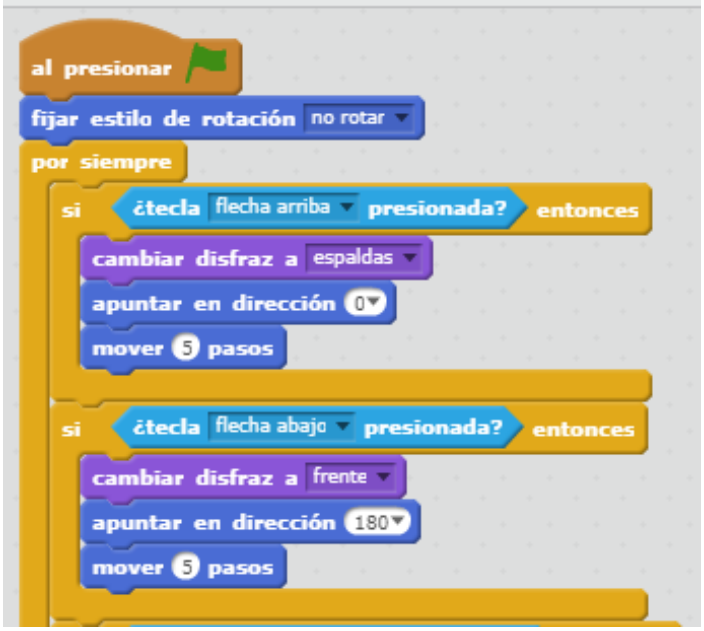
Puede parecer demasiado simple, pero añadido a un fondo, es muy efectivo, por ejemplo para simular un movimiento hacia el “interior” del fondo, sobre todo si le hacéis un disfraz en el que esté de espaldas.



Hago que el tamaño sea función de la coordenada y.
Pongo (-100) porque la coordenada y es negativa y así no le pido un tamaño negativo.

Divido por 45 para ajustar la perspectiva, prueba distintos números y te cambiará de distinta forma el tamaño.
Con ese 45, cuando el objeto esté en la base medirá un 400% de su tamaño (180/45=4)
A mitad de pantalla se queda al 0% (y = 0)

Este sería el código para el movimiento (he recortado el final por sabido ya)

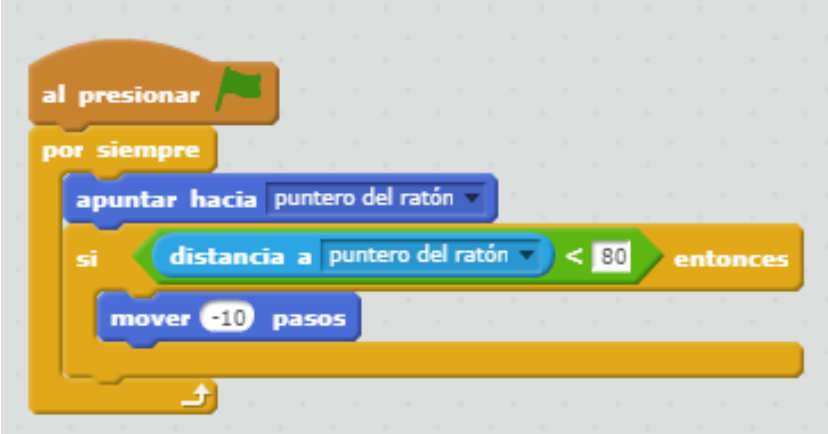


Como ves, en cada movimiento pongo un disfraz diferente y me quedan todos sin girar por el estilo de rotación elegido al principio.

NOTA: Ten en cuenta que según sea la perspectiva del fondo que tengas tendrás que ajustar ese 45 del que hablábamos antes y quizá también limitar el movimiento vertical, por ejemplo, porque haya una pared y no tenga sentido “subir” más.

Práctica 33. Objetos que me huyen

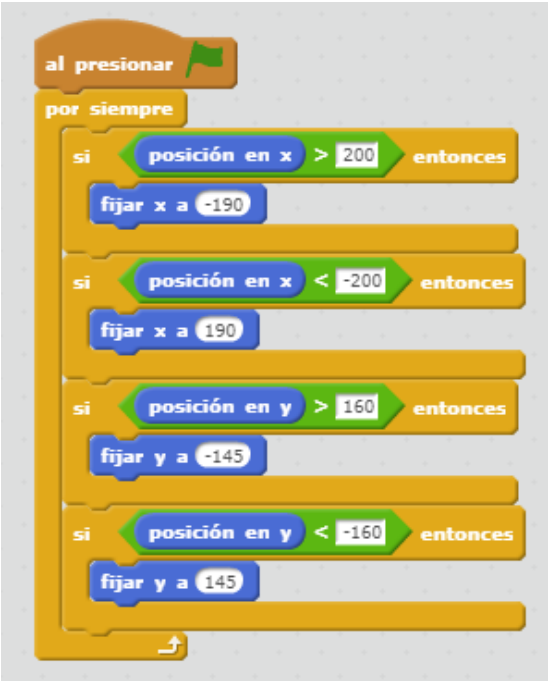
Scratch también nos permite saber la distancia entre objetos o entre un objeto y el ratón y eso puede servir para hacer un divertido movimiento de huida.



Como he puesto que me mire, no tengo que hacer consideraciones complejas sobre las coordenadas del gato y las mías, simplemente decirle que retroceda (avance negativo)

Si lo probáis veréis que lo echáis al borde, pero que se queda ahí y no hay manera de sacarlo… jeje.

Podríamos hacer que el escenario fuera “circular”, como pasa en mucho juegos, que si sales por arriba entras por abajo y que si sales por la izquierda entras por la derecha y viceversa.



Prueba con distintos modos de rotación, es divertido que te mire, pero no mola mucho que esté cabeza abajo… vaya, prueba tú y elige lo que prefieras.

Verás que lo detecto a 200, pero no lo mando a -200, lo mando un poco antes. Eso es porque si lo mando al otro extremo se activaría la otra condición y me lo mandaría de vuelta (!)

Práctica 34. Botones que cambian de color

El “feedback” al usuario, los sonidos, los cambios de color, etc. son elementos que generan una interacción más amigable.

Podríamos hacer que cuando el ratón pasase por encima de un objeto, este sufriera un cambio de color, indicándonos que se puede: pulsar, coger, mover… lo que sea que necesite nuestro programa. Seguro que recuerdas aventuras gráficas en las que nos pasábamos horas moviendo el ratón buscando algún objeto que fuera “clickable”.

Se puede hacer con un cambio de color del objeto.

Te voy a poner dos códigos que funcionan (usa sólo uno cada vez)

Este código comienza el bucle, mira si están el ratón encima, si lo está establece un efecto de color, si no lo quita.
Y repite el bucle una y otra vez.
Mientras estés sobre el botón se están mandando órdenes redundantes, hasta que te quites.

Aquí empezamos igual, pero nos quedamos esperando a ver si dejamos de estar encima para dar la orden de quitar efecto y volver a evaluar.
¿Crees que este ejecuta menos iteraciones del bucle?

Pues sí… y sobre todo, NO.
Ejecuta menos iteraciones del bucle “por siempre” que hemos escrito, pero es que nuestro bloque “espera hasta que” es en realidad un bucle “oculto” que mira una y otra vez a ver si el ratón está tocando, lo que sí que no está haciendo es volver a mandar las órdenes de poner color una y otra vez. Así que yo diría que es más eficiente.
También podéis hacer estas florituras cuando pulséis el botón, cambiando de disfraz o incluso poniendo algún sonido. Probad sin mirar…

¡SIN MIRAR HEMOS DICHO!

Casi seguro que os he pillado… ¿A que sólo habíais puesto “ratón presionado”? Pero fíjate que yo quiero ambas cosas, que el ratón esté sobre el botón y que se haga click.

Si sólo ponéis “ratón presionado” activaríais el botón haciendo click en cualquier parte del escenario. Comprobadlo.

EXTRA: Podéis poner una pequeña espera después de cambiar el disfraz, para que dé tiempo a verlo, también podéis esconder el botón después de ser activado, o volverlo a su disfraz inicial cuando hayáis dejado de pulsar y esté el ratón alejado. En fin, florituras y más florituras… las que queráis.

Práctica 35. Variables con controles deslizantes

Scratch permite la visualización de las variables en el escenario de varias formas:

- Con su nombre y su valor, como estamos haciendo todo el rato
- Sólo el valor
- Con un control deslizante

La segunda opción es una variante de la primera, más pensada (creo) yo para poner al lado un letrero vistoso, usando un objeto.

Para cambiar entre ellos se hace click sobre la caja de las variables en el escenario o, más finamente, botón derecho y se elige la forma que se desea.

Cuando está en modo “deslizador” si pulsas botón derecho verás que también puedes escoger los límites superior e inferior del valor, que por defecto están a 0 y 100.

Quería centrarme en la tercera, que permite al usuario cambiar valores en tiempo de ejecución sin necesidad de implementar el código, como solemos hacer: preguntar, recoger la respuesta en una variable, etc.

Por ejemplo:

Es sólo para que veáis cómo funciona.

En este caso el gato se mueve hacia la derecha y si llega al final se pone a la izquierda. De forma que si mantengo el espacio pulsado no deja de pasar de izquierda a derecha.

Así es como se ve la variable y su control.

Podríamos usar esto en muchos programas en los que hemos preguntado por valores al usuario, por ejemplo, en la gravedad del juego de las manzanas que caían.

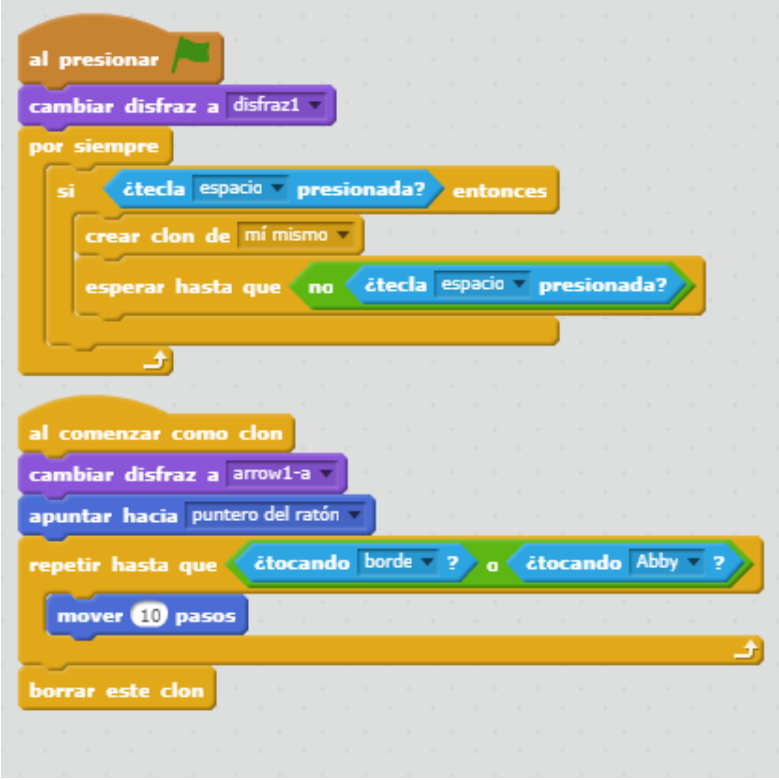
Práctica 36. Disparar

Bueno… sé que os gustan estas cosas, así que, disparemos aquí que hacemos menos daño.

Podríais pensar en un objeto como el que dispara, nuestro gato, por ejemplo y otro objeto como el proyectil. Y sin duda se puede hacer, incluso podéis clonar este último para tirar muchos proyectiles, pero he visto una manera más interesante. Clonar al propio tirador.

El truco consiste en que uno de los disfraces del tirador sea… una bala.

Así que, clonamos al tirador, le vestimos de bala… y a volar.



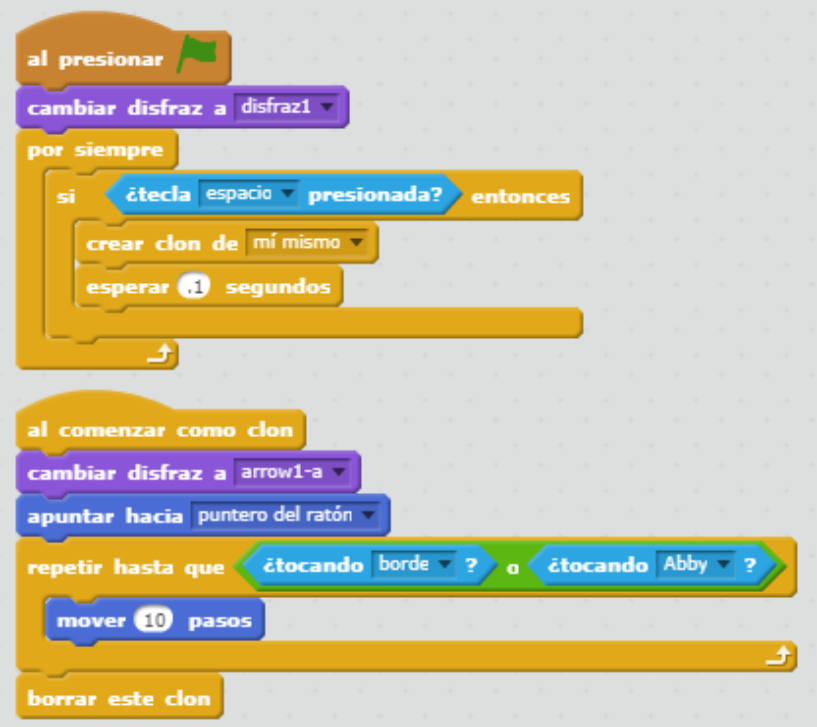
Nos aseguramos que el gato tiene el disfraz correcto. Cuando andamos jugando con alguna característica es bueno inicializar las cosas.

Miramos si se pulsa el espacio para crear un clon. Esperamos a que se suelte el espacio para generar el siguiente clon.

El clon cambia a disfraz bala, apunta al ratón, y avanza hasta que toca un borde o a la pobre Abby, y luego se borra.

Si metes el apuntar dentro del bucle, la bala sigue al ratón...

Si quieres poder disparar a ráfaga.

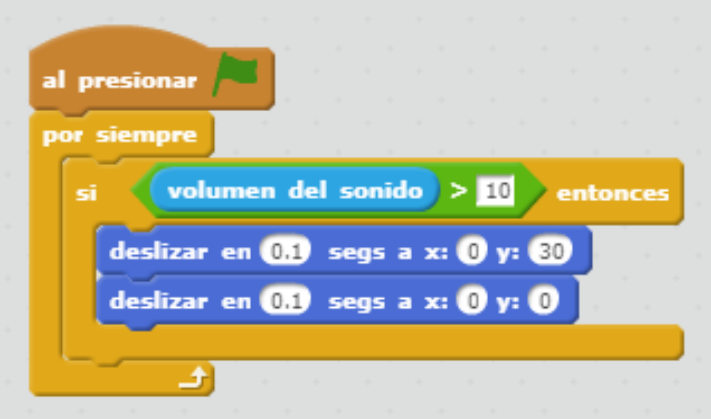


No esperamos a que se suelte el espacio, vamos creando clones con un pequeño retardo.

Aquí tienes un estupendo lugar de referencia donde te explican esta técnica y otras de disparo… y multitud de cosas de Scratch.
https://wiki.scratch.mit.edu/wiki/Shooting_Projectiles

Práctica 37. “Baila, gatito, baila!”

Scratch nos permite recoger el sonido del micrófono del ordenador y tomar decisiones basadas en el volumen que se detecta.



Con este código tan simple puedes hacer saltar al gato… con un chasquido de dedos

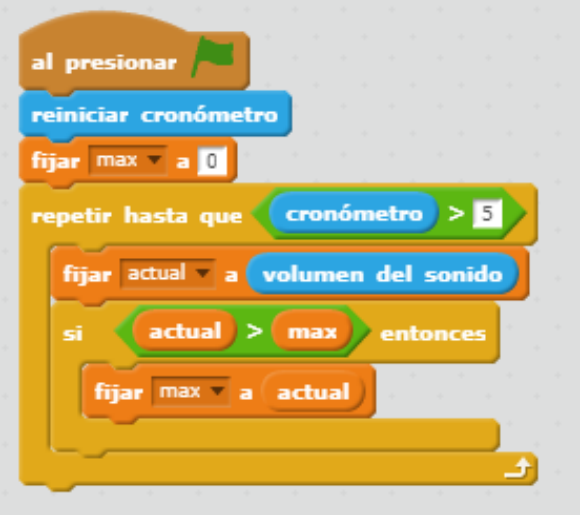
Pon visible en el escenario el volumen para que veas cómo responde a las condiciones ambientales

Programar esto es sencillo porque nosotros somos los programadores y los usuarios. Además también estamos en el lugar en el que se va a ejecutar el programa, pero, ¿podríamos hacer un programa que se adaptara a diferentes ruidos ambientes y fuera capaz de distinguir picos de volumen entre ese ruido? POS CLARO!

Práctica 38. “A ver quién grita más”

Esto es algo habitual en robótica, hacer un previo en el programa que detecte las condiciones ambientales para tomarlas como referencia.

Empecemos por guardar durante un tiempo el valor máximo del volumen



Ponemos el máximo en cero

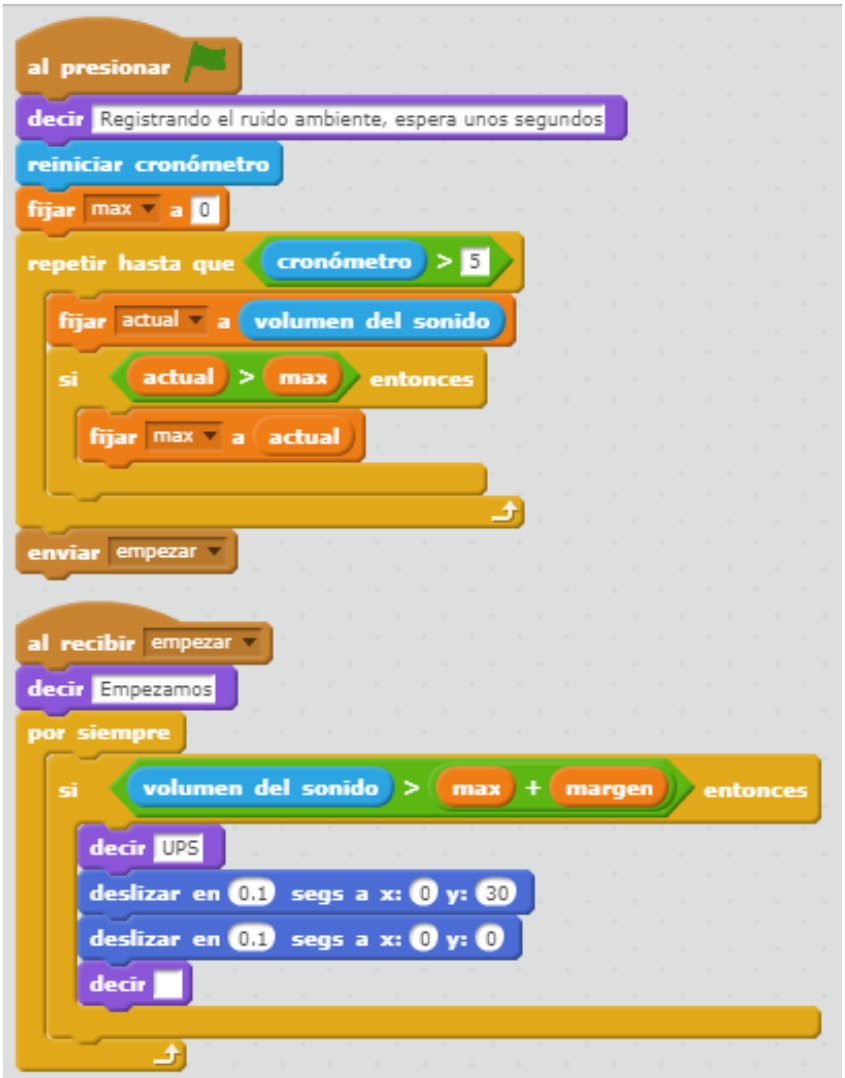
Nos damos un tiempo para registrar con el bucle

Miramos el valor actual y si es mayor que el máximo que teníamos, lo tomamos como nuevo máximo.

Después de esta primera toma de contacto con el ruido ambiente, empezaríamos a mirar a ver cómo es el sonido y activaríamos la

“alarma” cuando el sonido superase el máximo más allá de un margen.

Manos a la obra.



Nos avisa de que va a registrar el ruido.

Pone el máximo a cero

Repite durante cinco segundos la comparación de lo que oye, corrigiendo el máximo si procede.

Después de de ese tiempo manda el mensaje empezar

Avisa de que empieza

Mira si el sonido que registra es mayor que el máximo más cierto margen, si es así dice “ups” y salta. El último bloque es para que se calle después.

Si os extraña ver que **uso la variable margen pero no le doy valor nunca** (es espeluznante para cualquier programador) es que **la he puesto como un control deslizante** en el escenario para ir cambiando el margen en tiempo de ejecución y ver cómo se comporta el programa.

El máximo volumen que registra Scratch es 100… y con bien poquito follón. Así que si le gritáis mucho durante el registro del ruido, veréis que no conseguiréis hacer saltar a nuestro amigo.

No os he dicho nada sobre el mínimo… pero también podría interesarnos. Para eso usaríamos la estrategia contraria: Iniciamos el mínimo a 100 y lo vamos bajando con el valor actual que se registre durante el periodo de calibración.

Esta técnica te puede servir en muchas ocasiones, cuando uses sensores y tomes valores de la vida real.

El código conjunto sería

Como decíamos inicializamos el máximo abajo y el mínimo arriba.

Durante un tiempo

Leemos el valor del sonido y lo comparamos con ambos.

En caso de que el valor actual sea mayor que el máximo, será el nuevo máximo.

En caso de que el valor actual sea menor que el mínimo, será el nuevo mínimo.

Después de que pase el tiempo que nos fijamos, mandamos el mensaje de empezar.

Me gustaría preguntaros, ¿por qué creéis que he puesto el código de la izquierda en lugar de el de la derecha?

En el código de la derecha se comprueba cada vez ambos condicionales, siendo algo muy raro que los dos necesiten ser comprobados, ¿verdad?

¿Puede el valor actual ser mayor que el máximo y a la vez menor que el mínimo? Suena imposible… bueno, en realidad, ocurre sólo un par de veces.

Vamos a pensar iteración a iteración. Mira con atención la última columna.

Iteración	Actual	IZQUIERDA	DERECHA	CONDICIONALES CALCULADOS
0		max= 0 min= 100	max= 0 min= 100	IZQ / DCHA
1	actual =15	max= 15 min= 100	max= 15 min= 15	1/2
2	actual = 25	max= 25 min= 100	max= 25 min= 15	1/2
3	actual = 12	max= 25 min= 12	max= 25 min= 12	2/2
4	actual = 37	max = 37 min = 12	max= 37 min= 12	1/2

Fíjate, el código de la izquierda tarda un poco más en tener bien apuntados los valores máximo y mínimo, que en tiempo serán fracciones de segundo. A partir de ahí el código de la derecha siempre calculará dos condicionales, en cambio el de la izquierda se ahorra uno si el valor actual es mayor que el máximo, así que diríamos que es más eficiente, aunque los dos cumplen su cometido.

Práctica 39. Dibujando el sonido

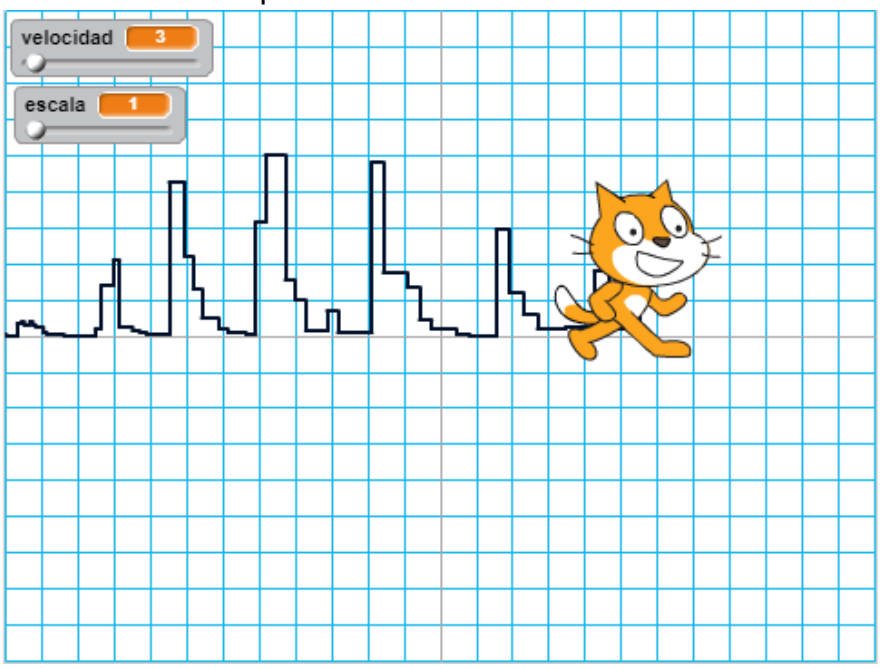
Como si hiciéramos un electrocardiograma…

Movamos al gato de izquierda a derecha a velocidad constante, pero cambiemos su altura según el volumen que vaya registrando.

Comienzo borrando la pantalla y subiendo el lápiz para que el desplazamiento del gato no me pinte líneas no deseadas.

Empezamos fijando el grosor de línea y su transparencia (tamaño e intensidad).
Borramos la pantalla y subimos el lápiz.
Llevamos el gato al medio y espera a que toquemos una tecla
Se calla ;)
Llevamos el gato a la izquierda, ponemos el reloj, bajamos el lápiz y nos ponemos a pintar moviendo la X de manera constante e Y según el volumen.
Velocidad y escala funcionan como constantes. Hacen que el gato vaya más rápido o despacio y que la gráfica salga más o menos alta.
Cuando llega a la derecha, borra, sube lápiz, vuelve a la izquierda, pone reloj a cero y otra vez dibuja tu sonido.

Queda así… si no pones modo TURBO



He puesto velocidad y escala como dos variables con control deslizante para poder ajustarlas.

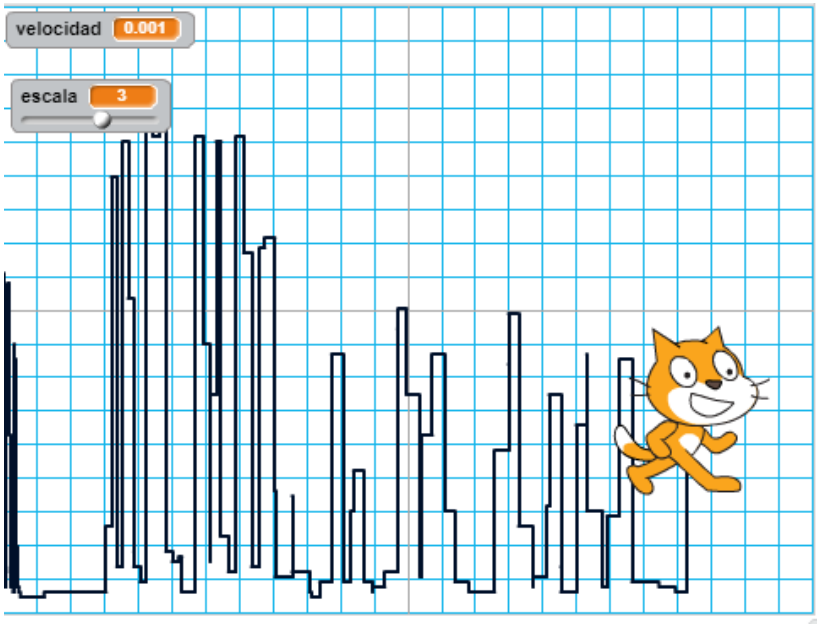
Como el sonido máximo es 100 y se llega rápido, y dado que la cuadrícula tiene un lado de 20 pasos, con una escala de 1 se ve bastante bien.

Si quisieras ver sólo un sonido ambiente muy bajo, sería mejor que ampliaras la escala. Ahí ya decides tú.

Como te decía antes, estamos haciendo un poco de trampa aprovechándonos de que NO estamos en modo TURBO, y el gato se mueve despacio.

Lo suyo sería ponerlo en modo TURBO y cambiar nosotros la velocidad o añadir esperas de la forma que considerásemos conveniente.

Aquí os dejo mi propuesta...



He añadido un bloque para fijar la velocidad en el código.

velocidad = 0.001
escala = 3

VÍDEO

Scratch permite la interacción con una cámara… esto es alucinante, porque abre toda una posibilidad de hacer "realidad aumentada" con nuestros muchachos y cuatro bloques.

Con el primer bloque encendemos y apagamos el vídeo, también puede invertirse la imagen.

Con el segundo podemo velar el vídeo, a 0% se ve nítido y a 100% desaparece.

Puede servir para tener un brillo cómodo o para hacer efectos estilo fade out (desvanecimiento) si se quiere.

El último bloque es el la clave del asunto, podemos detectar cambios y de dónde vienen Podemos detectar cambios sobre el escenario o sobre el objeto.

Ya os digo que las posibilidades son tremendas… a vosotros queda explorarlas, ya con todo lo que sabemos y combinando cosas podríamos, por ejemplo, hacer que un gatito abriera un ventana y que estuvieras tú dentro (!!) y que tus acciones pudieran cerrar la ventana o hacerle algo al objeto… en fin… recuerda también que puedes usar el volumen de tu voz para hacer cosas, el teclado… En ti queda.

Sólo para que veas cómo funciona y que puedas integrarlo en tus programas hagamos un par de juegos.

Práctica 40. Realidad aumentada 1. Coger manzanas

Haré aparecer manzanas en lugares aleatorios y las “cogeré” con mis movimientos, apuntándome unos puntitos ;)

Empezamos encendiendo el vídeo. La primera vez que lo hagas te pedirá permiso.

Pone los puntos a cero, esconde la manzana y se pone a crear clones.

Cuando empiezan los clones se muestran, ajustan tamaño y van a cualquier sitio.

En el último hilo miramos si hay movimiento de vídeo bajo ese clon y, en caso afirmativo, nos suma un punto y borra el clon. Hemos cogido la manzana.

No eres humano si no te tiras un rato braceando y riéndote, es la pera… jeje.

Bueno… ya más calmados.

No es perfecto, a veces cambia el brillo con un reflejo y lo detecta como un paso de “mi mano” y quita la manzana… y mil pegas que podríamos poner, pero que no pueden ocultar el hecho de que acabamos de hacer una aplicación de realidad aumentada con cuatro bloques y una lógica muy sencilla.

Por supuesto, las posibilidades de mejora son enormes.

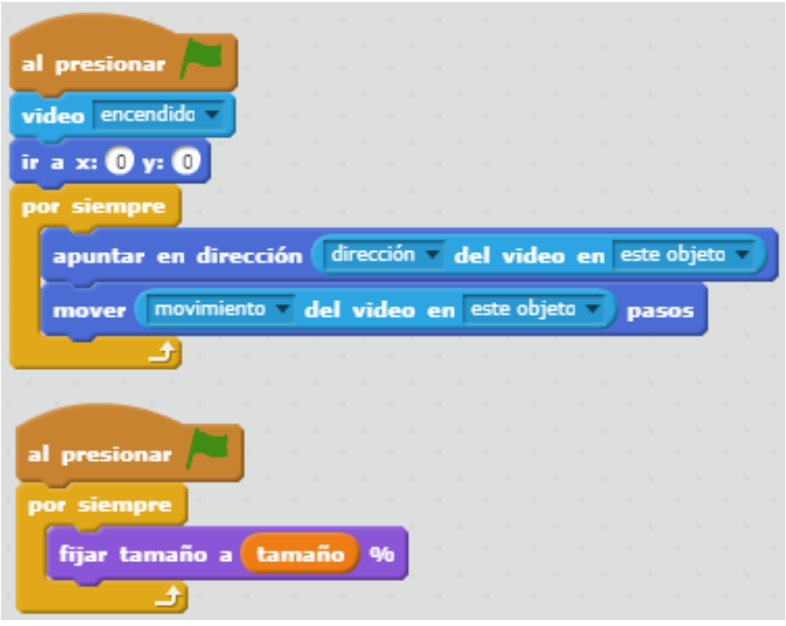
- Haz que las manzanas vayan menguando (cuidado con las esperas que paran los hilos)
- Puntúa diferente por las manzanas más pequeñas
- Pon objetos que no haya que coger y quiten vidas (hazlos pequeños, que si no…)
- Y todo lo que se te ocurra!!

Práctica 41. Detectar dirección de vídeo

Es estupenda también esta posibilidad de saber en qué dirección se está moviendo el vídeo y reaccionar… hagamos una prueba.

Vamos a intentar producir movimiento en una determinada dirección sobre un objeto.

Esto funciona sorprendentemente bien (yo he puesto una manzana…)



Encendemos el vídeo
Mando a la manzana al medio
Cambio la dirección según el vídeo
Muevo una cantidad de pasos igual al cambio del vídeo sobre el objeto. Esto último podríais cambiarlo para conseguir mejor efecto, pero así ya es muy bueno.

Lo último es sólo para probar tamaños (le he puesto un deslizador).

Es posible y no muy difícil controlar la posición y el movimiento de la manzana con las manos… incluso con la cabeza si la tienes debajo del objeto.

Se puede mantener "sujeta" entre las manos con facilidad o hacerla subir y bajar. También jugar al "tenis con ella y tus manos.

Mil ideas… te puedo proponer

EXTRA: Se me ocurre un juego con un límite de tiempo en el que haya que coger unos objetos y llevarlos hacia otros (frutas y canasta, por ejemplo). Con un límite de tiempo y con puntuación según lo que llevas, cuánto has tardado… tienes todos los elementos aprendidos para hacerlo.

Como en todo hay gente muy profesional, este proyecto cambia el movimiento de los objetos que le hacen de "fondo" produciendo una sensación de estar mirando por una ventana a un planeta, que cambia su perspectiva al mover tú la cabeza. Brutal.

https://scratch.mit.edu/projects/71971220/#player

SCROLLING

Algo muy interesante es tener (aparentemente) un fondo muy grande y que nuestro gato se pueda ir moviendo por él.

Lo primero es deciros que no se puede hacer con los fondos de los escenarios. Tendremos que **insertar objetos que nos funcionen como si fueran fondos**.

El truco consiste en mover ese "cuadro" mientras el gato se queda parado en el medio y que parezca que se desplaza.

Práctica 42. Scroll 1. Un paisaje que se mueve.

Empecemos por el principio.

Vamos a movernos con las flechas y meteremos la "cantidad" de movimiento en una variable: **scrollX**

Necesitamos que scrollX sea una **variable global** para que todos los "paisajes" (terrain les suelen llamar en inglés) puedan acceder a ella y puedan moverse correctamente.

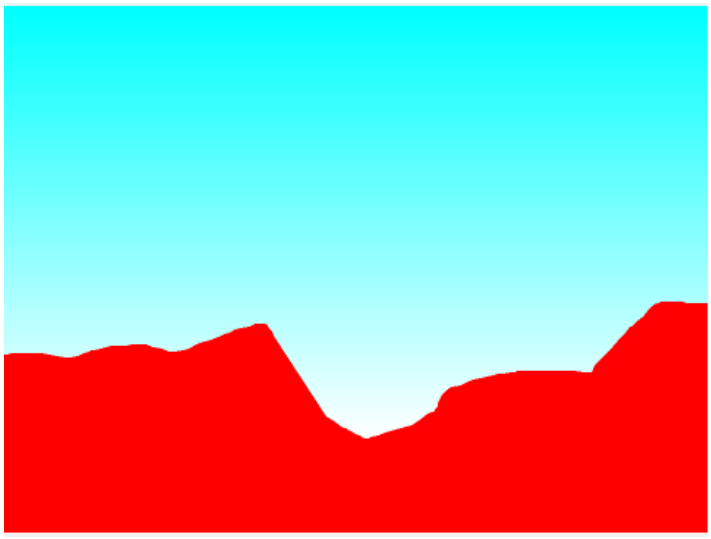
Le he puesto este código al gato

Nota el signo menos en la variable para la derecha y viceversa. Esto es porque si quiero que parezca que el gato se mueve a la izquierda, el paisaje debe moverse a la derecha.

He puesto que el gato mire hacia la izquierda o derecha para mayor realismo.

NOTA: He preferido **poner modo TURBO y poner yo las esperas** para moverme a la velocidad que quiera que no ponerlo y dejar que sea la ralentización de Scratch la que elija. Así se parece más a la programación de "verdad".

Cread un objeto paisaje, yo he hecho algo sencillote, pero bastará.



Y le he puesto este código

Muy simple, como veis.

Muestro el paisaje.
Pongo a cero la variable scrollX
Pongo el paisaje en el centro
Y ahora cambio constantemente la posición x del paisaje según me indique la variable, que al final son las pulsaciones que esté dando a las flechas.

Ya, ya… aparece el final del cuadro por el otro lado y no queda precioso, lo sé, pero es que acabamos de empezar. ¿Hasta aquí está claro? Ok, seguimos.

Qué te parece si ponemos un clon de mi paisaje a la derecha, fuera de la vista, pero pegado al primero, que también se desplace, y así parecerá que va seguido.

Pensemos un momento.

Mi paisaje original está en x = 0, para que no solape mi siguiente paisaje tiene que estar desplazado todo el ancho de la pantalla, esto son 480 pasos, ¿verdad?

Así que la posición del clon deberá ser

Lo que se “mueva” el gato, pero empezando más a la derecha.

Si quisiera hacer el viaje más largo con otro clon, tendría que mandarlo otra pantalla más allá.

Y si quiero hacer más clones… esto se apaña con una variable.

Me hago un índice para los clones y me quedaría así

Si clonNumber = 0 sería el paisaje “original”
Si clonNumber = 1 la primera extensión, etc.

Práctica 43. Scroll 2. Varios Paisajes

NOTA: El índice clonNumber necesita ser una variable local

A ver qué os parece este código

Empiezo escondiendo el escenario, pongo las variables a cero y el objeto en el centro.

Creo el primer clon cuando clonNumber vale cero. Como esa variable es LOCAL, no cambiará dentro del primer clon.

Ese clon es una **instancia** del objeto paisaje, como si fuera otro objeto, y las variables que tenga dentro, si son locales, sólo pueden cambiarse desde dentro.

Pero scrollX es una variable GLOBAL, así que sí responderá a los cambios de esa variable.

Prueba el código y verás que ahora tenemos un paisaje más “largo”.

Si quieres ver por qué esto es importante, créate un índice global y sustitúyelo por el que hemos hecho. Ejecuta y verás que los tres paisajes se van a la derecha del todo, porque reciben el cambio del índice desde el hilo principal.

Práctica 44. Scroll 3. Andamos sobre el suelo

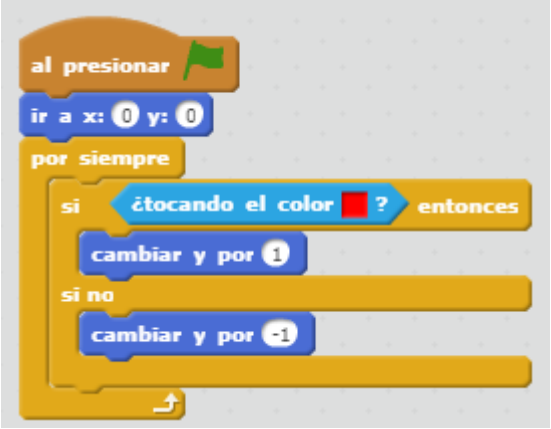
Refinemos…

Pensaba en proponeros otra cosa, pero me apetece mucho que el gato ande sobre el suelo… Y con colores seguro que sale a la primera.

Si pensáis que basta con tirar para arriba si tocamos el color rojo… va a salir a la segunda.

Eso te llevaría al gato para arriba y cuando el suelo bajase, no estaría tocando el rojo y no tendría forma de saber que el suelo hubiera descendido.

Así que metemos “gravedad” y ahora sí que funciona.



Digamos que si está tocando el rojo tira para arriba, pero si deja de tocarlo se “cae” un poquito.

Me queda un poco tembloroso, pero funciona ;)

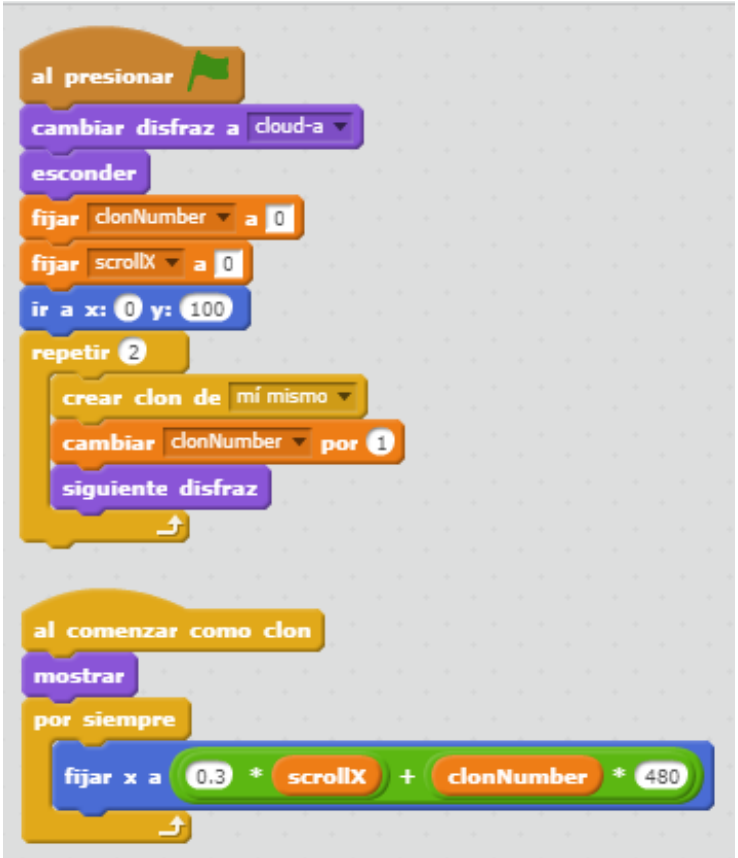
Práctica 45. Scroll 4. Dos paisajes desacoplados

Algo muy efectista es cuando hay distintos planos de profundidad en el desplazamiento.

Eso significa que tendremos varios objetos “paisaje” que se moverán a distinta velocidad. En mi caso voy a poner unas nubes.

Me voy a aprovechar de que el objeto nube tiene varios disfraces para hacer cada clon con un disfraz diferente, eso ayudará a reducir la impresión de “repetición del fondo”.

Este es el código que he usado

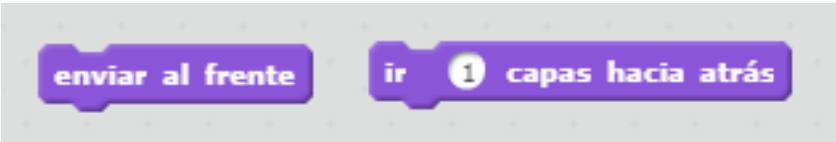


Te comento las diferencias:
Pongo un disfraz al principio.
Pongo el objeto más alto, para que las nubes estén arriba.
Cambio de disfraz en cada clon.

Y, lo más importante:
Multiplico la variable scrollX por 0,3 para que se mueva más lentamente.

Ajustad ese 0,3 hasta que os guste la velocidad.

Un detallito cuando pones varios objetos es decidir cuál quieres que te quede encima de cuál. Puedes crearlos en el orden que los desees, o bien jugar con estos bloques.



Cambiando ese 1 por -1 también te puedes traer el objeto hacia delante.

Como sabes que haciendo click en los bloques, estos se ejecutan, y como los hemos ido poniendo en un orden diferente al que queremos, he ido haciendo click en ese bloque “suelto” en cada objeto, hasta que los he tenido como he querido.

Los más detallistas se estarán preguntando si hay algún problema con usar el mismo nombre para el índice de los clones de un paisaje y los de otro… Ninguno, queridos, son variables LOCALES. En cada objeto toman valores diferentes.

Ajuste

Os habréis fijado que en los lados se quedan parados los paisajes y objetos si los pusierais, lo cual es un poco incordio. Una manera rápida de arreglarlo es superponer un marco, al menos lateral, y listo.

EXTRA: En este proyecto https://scratch.mit.edu/projects/168691186/#fullscreen tenéis una explicación muy interesante de cómo funciona este método de scrolling, y algunos detalles más. Me gustaría que os fijarais en que usan varios planos de profundidad: tienen suelo, unas rocas después, agua después, unas montañas y el cielo.

EXTRA: Otra interesante idea es situar objetos en distintas posiciones. Sabiendo que cada pantalla son 480 y que se tiene que mover (recuerda que el gato está quieto), tiene que venir al encuentro del gato.

2000 sería =	480·4 = 1920	Cuatro pantallas
	80	Un poquito más
	2000	Total.

Luego, una vez que te los encuentres, puedes comértelos y ganar puntos, chocar con ellos y perderlos… lo mismo que hemos hecho antes.

EXTRA: Nada nos impide poner disfraces también en el paisaje original para generar más variedad.

Práctica 46. Scroll 5. Scroll en dos dimensiones

Hay que duplicar todo lo hecho.... Empezando por crear scrollY

Para este caso, más que un camino muy largo que parezca variado nos puede interesar un mapa. Pensad entonces en hacer trocitos de 480 x 360 vuestro “mundo”.

360 porque en el eje vertical íbamos de -180 a 180, claro.

En lugar de generar clones, pondríamos “etiquetas” en los distintos trozos. Imagina que tenemos nueve partes en nuestro mapa. Se identificarán con dos valores (x,y)

La etiqueta de este “trozo” es (0,0) porque es el central.

Si pones (1,0) será el del medio a la derecha

Si pones (-1,1) será el de la esquina superior izquierda.

Y en tus manos lo dejo. El código es similar al visto, sólo presta atención a las pequeñas diferencias. Cambiar las X y las Y, los distintos rangos en alto y ancho de la pantalla, las variables. Pero te resultará sencillo.

Muchos son los programas que podéis hacer usando esta técnica: arcades, plataformas, etc.

FINALMENTE

Espero que os haya gustado y servido, como habéis visto es más una manera de abriros la puerta a este mundo que un conjunto de programas ya hechos que podáis copiar o jugar con ellos.

Mi intención es daros las **herramientas y elementos para que vosotros mismos construyáis vuestros programas.**

Por eso no he sido exhaustivo en todos los posibles programas que existen, no hemos mencionado clásicos como los laberintos, el Simón, pero sí creo **haber sido muy exhaustivo en las posibilidades de uso de los bloques, las estructuras y conceptos básicos de programación.**

Scratch llega hasta donde llega… pero es muy lejos. Aún así, como en cualquier disciplina, si te gusta y sientes que la herramienta que tienes se te queda corta, embárcate en aprender un lenguaje de programación más potente y a disfrutar del desafío de conseguir resolver problemas.

Si te interesa la robótica, en mi blog puedes encontrar dos manuales similares a este:

- Programación para Arduino usando Visualino
- Programación de mbot usando Mblock.

¡Hasta pronto!

Cualquier sugerencia o corrección será muy bienvenida y agradecida.
Este texto se publica con licencia Creative Commons, así que podéis descargarlo libremente, aunque **también podéis contribuir con una aportación económica** si tenéis la posibilidad y os apetece.

Podéis contactarme en:
email: javierfpanadero@yahoo.com
Twitter: @javierfpanadero
Blog: La ciencia para todos

Y quizá os interesen mis libros de divulgación científica bajo la serie
“La ciencia para todos”

JUNTOS SOMOS MÁS