

PRÁCTICAS DE ROBÓTICA CON VR-VEX



Javier Fernández Panadero

Febrero 2022



PRESENTACIÓN	2
INTRODUCCIÓN	3
PRIMEROS PASOS	5
PRIMER PROGRAMA	6
DIBUJANDO	8
REPETICIONES	14
VARIABLES	16
RECAPITULANDO	19
SENSORES (el parachoques)	20
SENSORES (el ojo de abajo)	25
CONDICIONALES	31
SENSORES (distancia)	34
ACTUADORES (el imán)	37
Modificar variables durante el programa. Espiral	44
Búsqueda	47
Búsqueda con espiral creciente	47
ACUMULADORES	52
EL MONITOR	52
Barrer la zona. Localizar y empujar.	57
¡Hagamos un robot aspiradora!	60
Listas y cronómetro	69

PRESENTACIÓN

Aquí os dejo uno más de los tutoriales que voy escribiendo para mis alumnos y que comparto también con la Comunidad de la que tanto aprendo.

Sabéis que os los voy enlazando en esta página de mi blog [para profes](#)

Se agradece difusión y apoyo.

Quien quiera y pueda, puede invitar [a un ko-fi por aquí](#)



Juntos somos más.

Javier Fernández Panadero

Febrero 2022

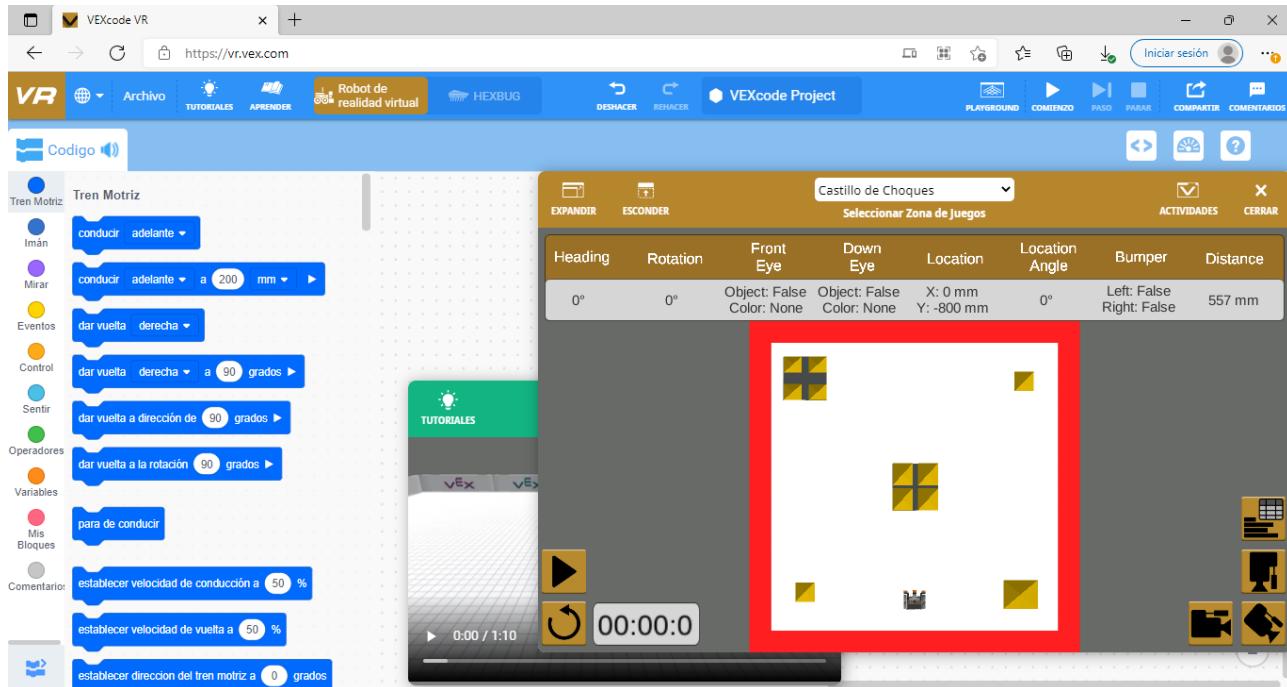
INTRODUCCIÓN

En estas prácticas de robótica vamos a usar el simulador [VEXcode VR](#)

Se trata de un simulador de “realidad virtual” en el que podremos ver al robot como se comportaría en un entorno real.

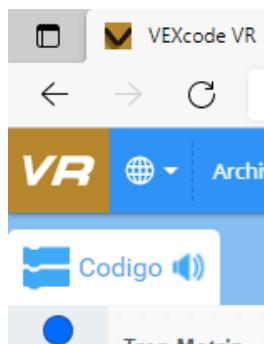
PRIMEROS PASOS

Al entrar veréis algo similar a esto:



1. SELECCIÓN DEL IDIOMA.

Lo tenéis arriba a la izquierda. Hay una “bola del mundo”.

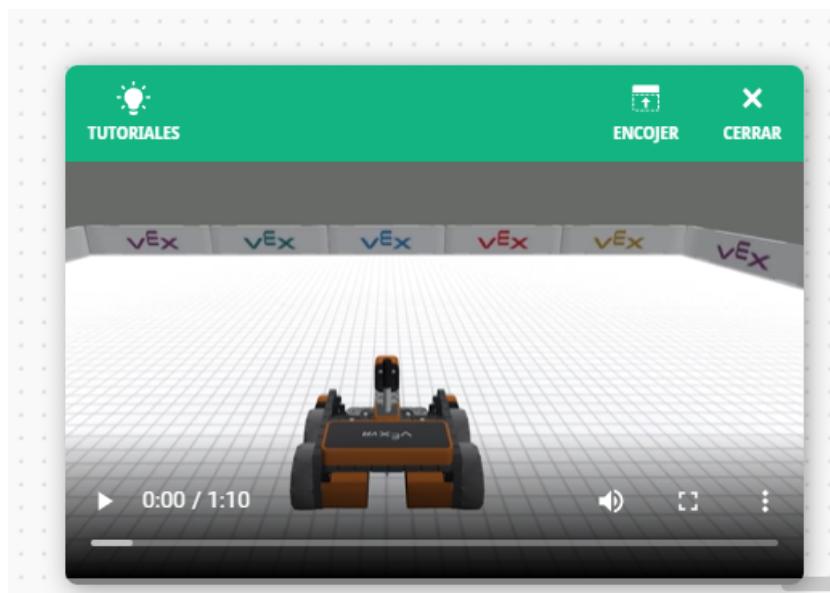


2. VER/OCULTAR/ENCOGER VIDEOTUTORIALES

Tenéis pequeños vídeos explicando como se hacen las acciones más básicas con bloques. Al entrar te sale abierta la ventana de videotutoriales.

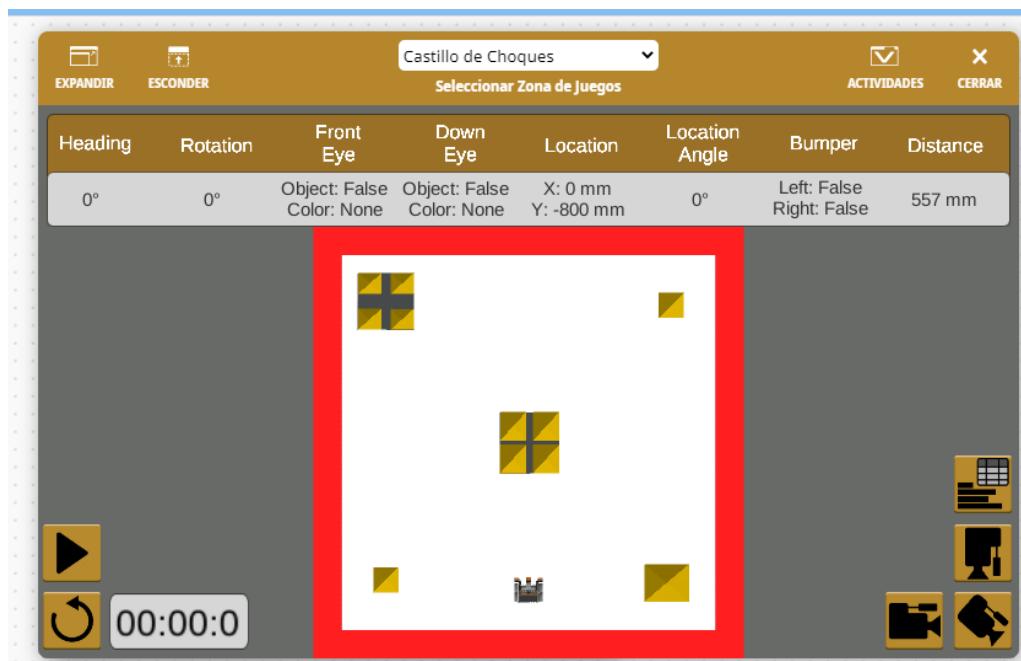
Haciendo click en la bombilla se abre un página donde puedes ver otros.

Si conoces Scratch es probable que muchas cosas te resulten familiares, en caso contrario, consulta en estos videotutoriales lo necesario.

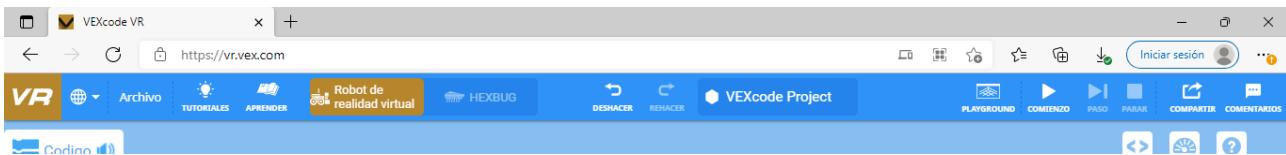


3. VER/OCULTAR/ENCOGER PLAYGROUND

Llaman Playground al “mundo” virtual en el que se va a mover el robot. Hay varias “zonas de juego” distintas y haremos prácticas diferentes en cada una de ellas.



4. BARRA SUPERIOR



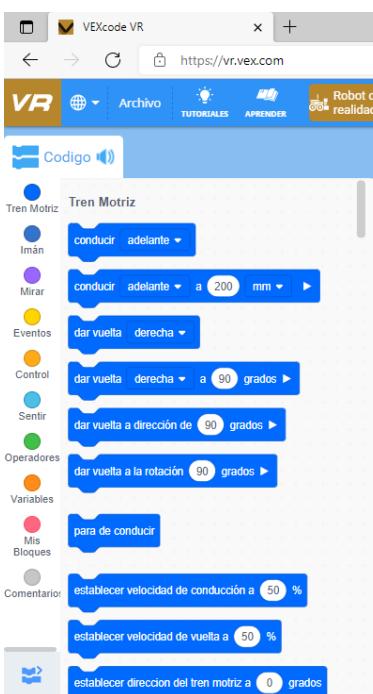
En esta barra superior, si has cerrado los videotutoriales o el playground, puedes volver a abrirlos.

El resto de funciones las iremos usando según necesitemos.

5. BLOQUES DE CÓDIGO

En la parte izquierda tienes disponibles los bloques de código para escribir el programa.

Puedes “esconderlo” dándole al ícono que hay debajo a la izquierda



Verás que son bloques muy similares a los que se usan en Scratch.

Puedes ir bajando y ver todos los bloques o hacer click en el círculo de cada color (Tren motor, Imán, etc.) y te llevará directamente a ese conjunto de bloques.

OJO: Estos bloques pueden cambiar según la zona de juegos que hayas elegido en el playground.

Elige ANTES DE EMPEZAR A PROGRAMAR la zona de juegos en la que quieras que se ejecute el programa.

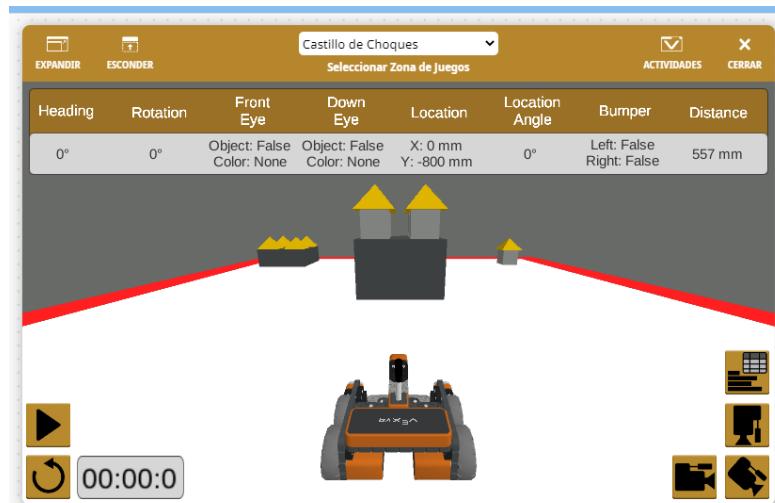
PRIMER PROGRAMA

Empecemos.

Oculta o encoge los videotutoriales.

Elige en el playground la zona “Castillo de choques”.

Para que sea más divertido elige un punto de vista “subjetivo” en el playground. Lo tienes abajo a la derecha. En la captura siguiente está en “visión desde atrás”, pero lo podéis más subjetivo aún con el ícono de la cámara horizontal.



Una vez hecho esto, “esconde” el playground, para tener sitio para poder programar.

Vamos a escoger un bloque para empezar y arrastrarlo hasta la zona central. Lo más sencillo. Que empiece a andar...

Así que, en el submenú “Tren motriz” elegimos “conducir adelante”



Vemos que el coche no hace nada...

Esto está bien. Una cosa es ESCRIBIR el programa y otra EJECUTARLO.

Para ejecutarlo, hay que darle al PLAY.

Tienes dos botones PLAY

- Uno abajo a la izquierda en el playground
- Otro en la barra azul superior en la parte derecha.

Muestra de nuevo el Playground para ver lo que le pasará al robot y dale al PLAY.

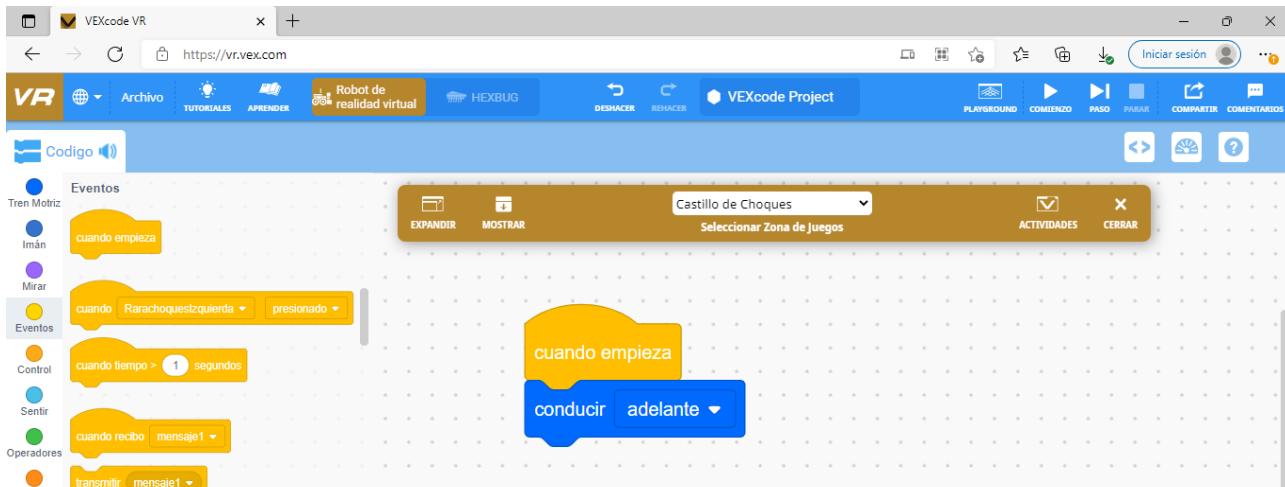
Vemos que el coche no hace nada, de nuevo...

Para que funcione nos falta un detalle.

PLAY se refiere a que el programa se inicie.

Pero nuestro bloque está “en el aire”, hay que especificar que queremos que se ejecute cuando el programa se inicie.

Para eso tenemos que usar este bloque “cuando empieza”, que lo encontráis en el grupo “eventos”.



Esto significa que cuando le demos al PLAY, se ejecutarán todos los bloques que estén debajo de “cuando empieza (el programa)”.

OJO: Ahora sí va a funcionar. Muestra el playground antes de darle al PLAY y observa lo que sucede.

¡¡¡AHORA SÍ!!!

Como veis le ha dado igual que hubiera un castillo de bloques o que se cayera al llegar al final del suelo...

Esto nos lleva a una de las máximas de la programación.

Lo mejor de la programación es que hace lo que le dices

Lo peor de la programación es que hace lo que dices.

Da igual que te hayas equivocado al elegir un bloque, da igual que quisieras que se parara al chocar con el castillo o que no cayera por el barranco... si no se lo dices, explícitamente, no lo va a hacer.

Esto a veces lo percibimos como “malo” porque a veces me gustaría que me leyera el pensamiento y “decidiera” pararse, pero pensad lo bueno que es que no tome otras decisiones que las que yo le haya dicho. Así tengo el control.

Por lo tanto, de aquí en adelante tendremos que tener mucho cuidado con lo que queremos y DARLE INSTRUCCIONES CLARAS Y DETALLADAS.

Objetivos cumplidos:

- Conocer el entorno básico de programación
- Ejecutar el primer programa
- Entender que hay que dar órdenes precisas para conseguir el resultado

DIBUJANDO

1. CUADRADO

Nuestro robot viene equipado con diferentes sensores y actuadores, de los que nos iremos aprovechando para nuestras prácticas.

En esta vamos a usar un “lápiz” con el que podemos ir haciendo un dibujo según se va moviendo.

Empezamos cambiando la zona de juegos a “Mapa de cuadrícula”.

Elegimos la cámara para la vista superior, para tener más control de lo que hacemos.

Vamos a intentar un movimiento más preciso a ver qué sucede.



Hemos andado una cuadrícula. Eso ya nos da una idea del tamaño de las cosas.

En todo caso, en la barra superior del playground tenéis los valores de muchas variables y el estado de los sensores.

A screenshot of the RoboBlockly playground. At the top, there are buttons for "EXPANDIR", "ESCONDER", "Mapa de Cuadrícula" (selected), "ACTIVIDADES", and "CERRAR". Below is a table with sensor data:

Heading	Rotation	Front Eye	Down Eye	Location	Location Angle	Bumper	Distance
0°	0°	Object: False Color: None	Object: False Color: None	X: -900 mm Y: -700 mm	0°	Left: False Right: False	1645 mm

On the left, there are play and reset buttons, and a timer showing "00:03:9". On the right, there are icons for camera, video, and other sensors. The central area shows a 2D grid workspace with a small robot icon at the bottom-left corner.

Los iremos viendo todos poco a poco, en este momento vamos a concentrarnos en algunos de ellos.

Si miras en Location, verás que ha cambiado el valor de la coordenada Y. (Si no te habías fijado, ejecuta de nuevo y verás cómo vuelve a cambiar)

También habéis notado que el cronómetro seguía andando aunque no hubiera más instrucciones (bloques) que ejecutar y que habéis tenido que darle al STOP en el que se había convertido el PLAY.

Si queréis volver a la posición inicial, tenéis el botón de REINICIAR justo debajo.



La barra de datos del playground puede sacarse u ocultarse con el botón que tenéis donde están las "cámaras"



Como habréis notado, nos hemos movido, pero NO HEMOS PINTADO NADA en el suelo. Eso se debe a que TENEMOS EL LÁPIZ "SUBIDO".

Si queréis que al moverse el robot vaya dibujando una línea, tenéis que bajar el lápiz.

Para eso tenéis que usar este bloque que está en el grupo "Mirar"



Si reiniciáis y ejecutáis este código, veréis cómo se dibuja una línea.



Con todo esto...

HAGAMOS UN CUADRADO.

Pista: usa este bloque también



¡¿Te atreves a intentarlo tú primero sin mirar la solución?!

Solución:



¿Qué tal ha ido? No demasiado difícil, ¿verdad?

Quizá te haya resultado pesado tener que sacar uno a uno los bloques...

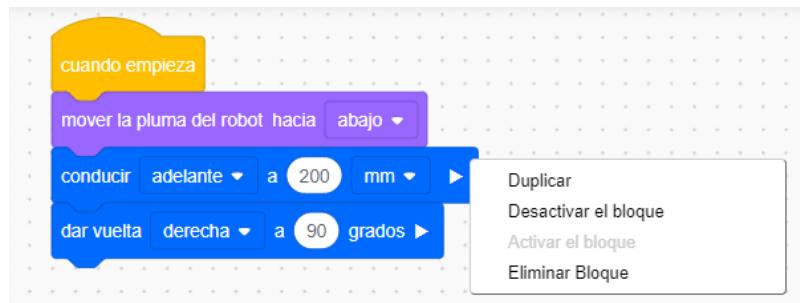
Siempre que te pase eso, ten en cuenta que...

Si necesitas hacer algo “razonable”,

seguro que hay una manera rápida de conseguirlo

Nosotros no somos pioneros. Estamos aprendiendo cosas que se han estado haciendo durante décadas por millones de personas... ¿Crees que eres el primero al que le hubiera apetecido poder hacer una copia o un duplicado rápido? Cuando te encuentres en esa tesitura, busca esa “solución rápida” o pregúntala.

En nuestro caso, si hacéis click con el botón secundario (el derecho, típicamente) sobre los bloques, te da la opción de duplicarlos.

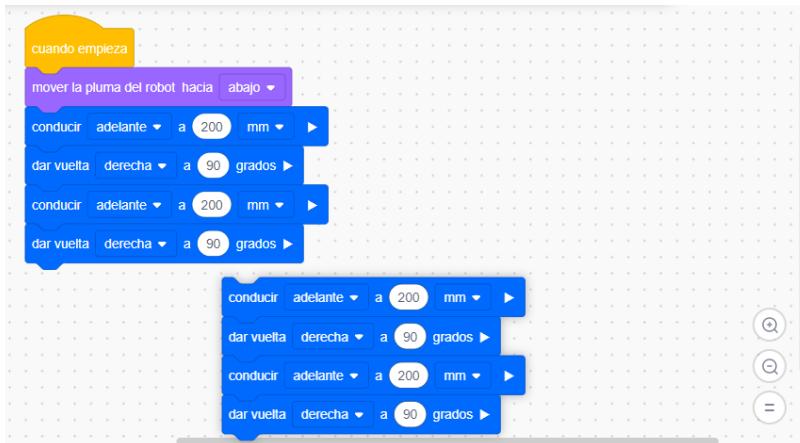


Así que, los dos siguientes me salen “gratis”



Podríais pensar que ya está, hacemos esto TRES veces más y listo, pero en realidad podemos ser incluso más rápidos.

Si una vez pegados esos dos bloques, duplico el conjunto de cuatro, ya me salen todos los bloques que necesito para el programa.



Así que no os olvidéis. Si tenéis una necesidad “razonable”, es muy probable que ya esté resuelta. Buscadla y os ahorrará trabajo innecesario.

2. TRIÁNGULO

Has visto que hemos sacado los bloques sin modificarlos, hemos avanzado la distancia que venía en el bloque y hemos girado 90° , pero **hay valores en los bloques que pueden modificarse**.

De momento, puedes probar a hacer click donde pone 200 mm y cambiar el número o cambiar la unidad (esto último es un menú desplegable, dándole a la flecha).

Comprueba que el hecho de que el robot vaya hacia delante o hacia atrás, o que el giro sea a la derecha o la izquierda también puede cambiarse en el menú desplegable.

HAGAMOS UN TRIÁNGULO (equilátero).

Elige una medida diferente a 200 mm e intenta descubrir qué ángulo tienes que girar para poder dibujar un triángulo antes de mirar la solución.

Pista 1: En el cuadrado dábamos una vuelta completa (360°) en cuatro movimientos. En el triángulo lo haremos en tres...

Pista 2: El ángulo que gira el robot no es el ángulo interior del triángulo... sino el exterior.

SOLUCIÓN

Como vamos a dar la vuelta completa (360°) en tres “pasos”, cada paso tiene que ser de $360/3 = 120^\circ$.



Si quieres probar con otros polígonos regulares es así de sencillo. Tendrás que hacer tantos “pasos” como lados tenga el polígono y, en cada giro, hacer un ángulo igual a 360° entre el número de pasos.

Prueba a hacer un pentágono o un octógono. Ya sabes duplicar bloques, así que, en realidad, será rápido y sencillo.

Puede que se te caiga el robot al intentar dibujar. Para solucionarlo, mueve el robot primero hasta una posición más central y dibuja después el polígono.

Objetivos cumplidos:

- Mover el robot y hacerlo girar.
- Uso de la pluma (pintar según se escribe)
- Entender que hay maneras de optimizar el trabajo

REPETICIONES

Supongo que no soy el único al que le molesta un poco esto de repetir bloques, más allá de que lo hayamos hecho más rápido con el truco de duplicar.

De nuevo la pregunta, ¿seguro que no hay una forma más eficiente de hacer esto?

Igual que a mí me dicen, “Lava cinco platos” en lugar de decirme “lava un plato, lava un plato, lava un plato...” hasta cinco veces, ¿no hay una manera de decirle al robot “repita 5 veces estas instrucciones”?

Por supuesto que sí. Es a lo que llamamos BUCLES y para esto de los polígonos nos viene que ni pintado.

El bloque que necesitamos está en el grupo “Control”.



Por supuesto, se puede poner más de un bloque en el “agujero” (se irá agrandando según los vayamos metiendo), y también puede elegirse cuántas repeticiones vamos a hacer, aunque *por defecto* hayan puesto 10.

NOTA: Llamamos valores por defecto a los que tiene configurados un sistema cuando lo iniciamos. Por ejemplo, el color de la pluma con el que “pintamos” o que el lápiz estuviera “subido” son valores por defecto en esta aplicación.

DIBUJA UN POLÍGONO USANDO UN BUCLE

Intenta volver a hacer un polígono, pero usando un bucle... la solución en la página siguiente.

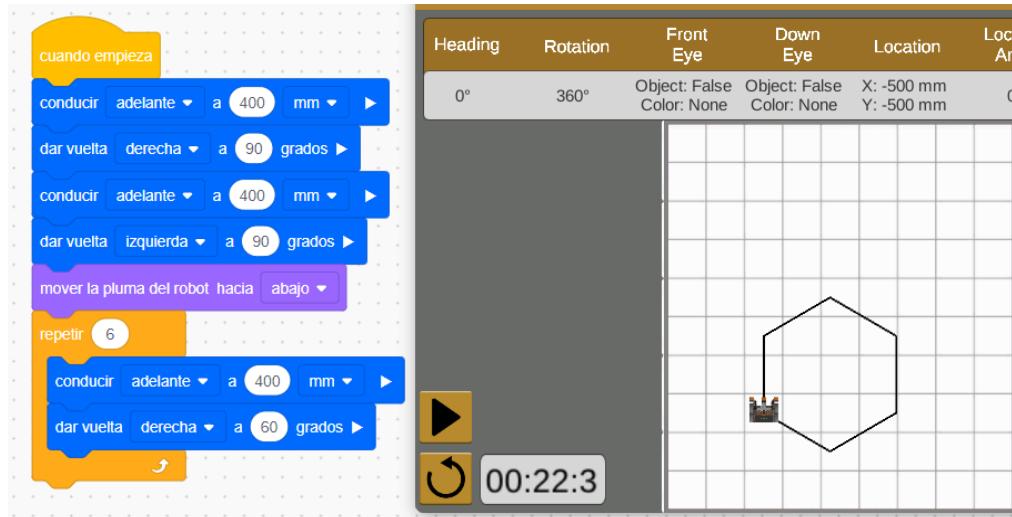
Y, si te sale...

DIBUJA DOS POLÍGONOS SEPARADOS

Pista: Atención con el lápiz...

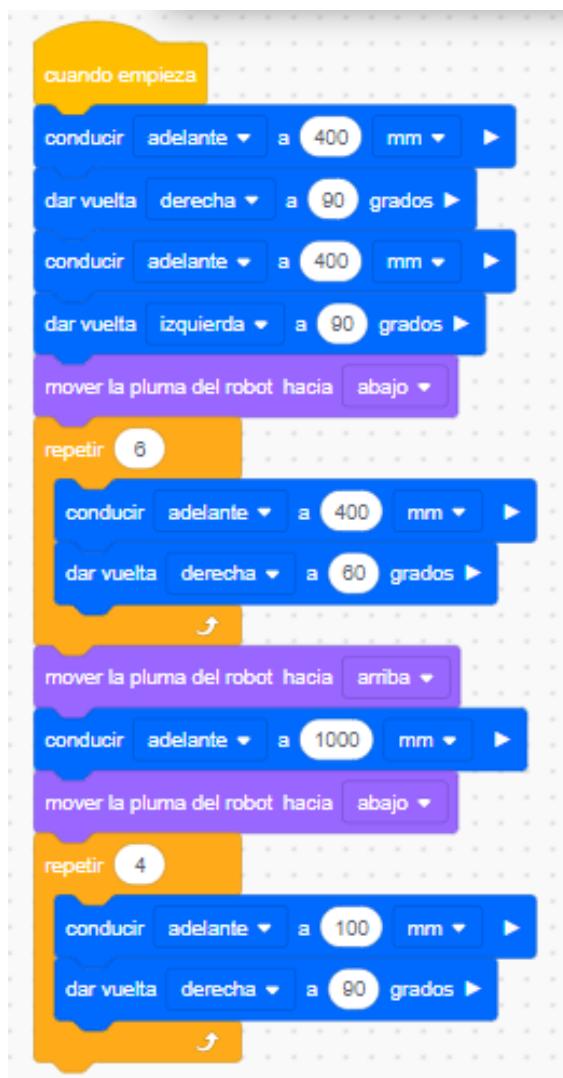
OJO: Quizá tengas el playground encima de los bloques y no veas lo que pasa cuando los ejecutas. Desplaza la ventana de playground un poco hacia un lado y verás cómo se van iluminando los bloques que se están ejecutando en cada momento. Es una manera muy buena de entender lo que está ocurriendo.

SOLUCIÓN 1



Fíjate que no he querido bajar el lápiz hasta estar en la posición más centrada, para no dejar una línea dibujada en ese camino.

SOLUCIÓN 2



¿Te acordaste de volver a bajar el lápiz para pintar el segundo polígono?

Quizá te parezca que queda un poco feo que el robot acabe “encima” del dibujo tapándolo.

Si quieras, una vez que has terminado el dibujo, puedes volver a levantar el lápiz y desplazar al robot una cierta distancia para que no tape nada.

Objetivos cumplidos:

- Hacer bucles con un número de repeticiones
- Mejorar el uso de la pluma
- Ver la ejecución paso a paso

VARIABLES

Vamos a hacer algo más potente, que ya sabemos unas cuantas cosas.

En realidad no hace falta hacer un programa para el triángulo, otro para el cuadrado, otro para el pentágono...

Podemos hacer un programa que sea capaz de dibujar CUALQUIER polígono.

Si lo intentamos decir en “lenguaje natural” sería algo como esto:

Repite N veces: avanza y gira 360/N

Donde N sería el número de lados que quisiéramos hacer, tres para el triángulo, cuatro para el cuadrado...

¿Cómo podemos expresar esto en el lenguaje del robot?

Para eso necesitamos UNA VARIABLE.

Podríamos decir que una variable es una caja donde guardamos información.

Las variables tienen estas características:

- **Nombre:** Cómo nos referimos a ellas en el programa (en nuestro ejemplo la llamaremos “lados”).
- **Tipo:** Qué clase de información guardan: números, letras, palabras, bits... (En nuestro caso será una variable numérica, guardaremos el número de lados del polígono)
- **Valor:** Qué dato concreto están guardando. Esta puede cambiar de ejecución en ejecución o durante el programa, por eso se llaman “variables” (en nuestro caso tendrá un valor 3 para el triángulo, cuatro para el cuadrado, etc.)

Para crear una variable, tendremos que ir al grupo “variables” de la parte de los bloques. Allí encontrarás las variables que ya hayas creado y podrás crear nuevas.

En nuestro caso, aún no tenemos ninguna. Así que le damos a “Hacer una variable” y aparece una ventana emergente para que le demos nombre.

Escogemos un nombre que nos ayude a recordar qué representa. Como dijimos, una buena opción es *lados*.

Cuando le des a aceptar verás que hay bloques nuevos. Veamos cómo funcionan algunos de ellos, más adelante iremos viendo las demás funcionalidades.



El primer bloque representa el valor de la variable (3, 4, etc)

El segundo bloque sirve para establecer el valor de la variable al número que pongas en el lugar donde está el cero. Así es como conseguimos que la variable tenga un valor concreto.

Procederemos así:

Al principio del programa estableceremos el valor de la variable al número de lados que queramos para cada ejecución.



Para hacer la cuenta $360/\text{lados}$ necesitamos los bloques del grupo “Operadores”.

Allí tenemos este que es el de la división



- En el primer “hueco” escribimos 360
- En el segundo hueco arrastramos el bloque del valor de la variable



OJO: Ponemos el bloque NO escribimos la palabra “lados”.

Nos quedará así



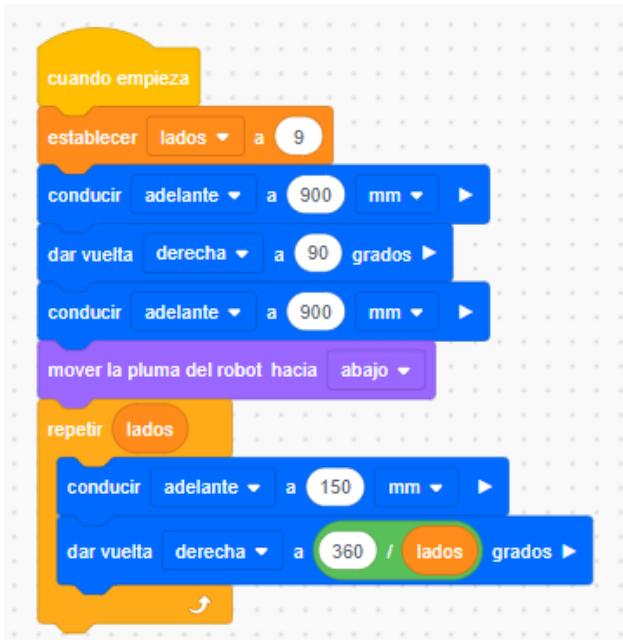
Y ese será el ángulo que haya que arrastrar al hueco del bloque “dar vuelta”.

¿Te atreves a intentarlo antes de mirar la solución en la página siguiente?

HAZ UN PROGRAMA QUE DIBUJE UN POLÍGONO CON EL NÚMERO DE LADOS COMO VARIABLE.

HAZ UN PROGRAMA QUE DIBUJE UN POLÍGONO CON EL NÚMERO DE LADOS Y LA LONGITUD DE LOS LADOS COMO VARIABLES.

SOLUCIÓN 1:



SOLUCIÓN 2:



Objetivos cumplidos:

- Saber lo que es una variable
- Distinguir entre nombre, tipo y valor de una variable
- Distinguir cómo dar valor a una variable y cómo usar ese valor en un bloque

Como veis, sólo hace falta cambiar el valor de la variable antes de cada ejecución y te hace un polígono distinto.

Con muy pocas líneas hemos hecho un programa muy general gracias a usar un bucle y una variable.

Si quieras hacerlo más general, podrías poner que la longitud del lado fuera otra variable.

Veréis que he hecho algo muy parecido.

Crear la variable primero en el grupo "variables" y darle un nombre.

Darle valor a la variable longitud al principio del programa.

Usarla en el bloque que quiera, (conducir, en este caso), mediante el pequeño bloque naranja que me aparece al crearla.

RECAPITULANDO

Veamos qué cosas hemos aprendido, para no perdernos.

- Sabemos colocar bloques que se ejecuten según le damos al play.
- Sabemos hacer bucles con cierto número de repeticiones.
- Sabemos utilizar variables: Crearlas, cambiar su valor y usar su valor en otro bloque.
- Hemos visto que los bloques “redondeados” tienen un valor numérico (puede ser el de la variable o el resultado de una operación, p.ej.)

Pero, la verdad, es que nuestro robot es un poco “bruto”. Hace lo que le mandamos y le da igual que se caiga por el borde o que empuje bloques que haya por el medio, como hemos visto. No le importa “el mundo” ni “los demás”... es un egoísta ;)

Vamos a ponerle remedio.

Lo interesante de los robots “de verdad” es, precisamente, que sean capaces de percibir “el mundo” con sus sensores y tomar decisiones para sus tareas. Por ejemplo: “vaya, qué cerca estoy del borde, voy a parar antes de caerme”. “vaya, me acabo de chocar con una pared, voy a dar marcha atrás”. “vaya, he encontrado un objeto de color verde, justo lo que estaba buscando, voy a recogerlo”.

Pero recordemos que dijimos al principio que el robot no iba a hacer nada que no le dijéramos que hiciera, así que, tendremos que darle la orden de mirar “al mundo” y explicarle de forma muy concreta cómo actuar según lo que “vea”.

¡Vamos al lío!

SENSORES (el parachoques)

Este robot virtual tiene varios sensores que replican aquellos que ya existen en las versiones reales.

Sensores de color, de presión (parachoques), de distancia (por ultrasonidos -como los murciélagos), giróscopos o brújulas que te indican la dirección a la que apuntas, encoders y otros sistemas que te indican la distancia recorrida, combinaciones de estos que te dan la posición en la que te encuentras, etc.

Iremos viendo cómo funcionan según los vayamos usando.

Empecemos con los parachoques que son sencillitos. Son básicamente un pulsador que se activa cuando chocamos con algo.

Vayamos a la zona de juegos “Castillo de bloques dinámico”... y vamos a darnos unos cabezazos.

Si buscamos entre los bloques, en el grupo “Sentir”, veréis este bloque.



Por supuesto lo de “Rarachoques” es una errata en la traducción. La disculpamos y agradecemos el esfuerzo de tener la aplicación en tantos idiomas. No seamos *tiquismiquis* ;)

Fíjate que es un tipo de bloque diferente a los que conocíamos. No es un bloque de “acción” ni tampoco es un bloque de valor numérico, tiene los bordes angulosos.

Si leemos lo que pone, veremos que se trata de una pregunta. “¿Está el parachoques izquierdo presionado?”

Esa pregunta tiene solo dos respuestas posibles y excluyentes. Sí o No. Esto es lo que se llama un valor *booleano*.

Los bloques de esta forma aparecerán en otras circunstancias, pero siempre les pasará igual, tomarán el valor sí o no (verdadero o falso, si lo prefieres).

HAGAMOS QUE EL ROBOT SE PARE SI ENCUENTRA UN OBSTÁCULO

Empecemos por algo simple. Que se mueva en línea recta hasta que choque con algo.

Fíjate que no sabemos cuánta distancia va a recorrer en cada ejecución, depende de cómo estén colocados los bloques. Así que, nuestro robot REACCIONARÁ a los distintos entornos que se encuentre.

Hay varias maneras de hacerlo.

Te espero un poco si quieres probar tú alguna antes de mirar las soluciones que te propongo yo... porque hay más de una.

Pista: Vas a necesitar bloques que están en el grupo “Control”

SOLUCIÓN 1:



Curiosamente esta primera solución, es casi “lenguaje natural”. Mirad, lo leo.

“Cuando empieza (la ejecución) conduce adelante y espera a que el parachoques esté presionado. Entonces, para de conducir.”

Reiniciad y veréis que va cambiando la disposición de los bloques, pero se sigue parando cuando se choca.

Os recuerdo que podéis poner la cámara en “punto de vista subjetivo”, y veis los golpes en primera persona ;)

¡Analicemos!

El bloque “Espera hasta que...” es en realidad **un “bucle disfrazado”**, porque lo que hace es mirar una y otra vez al parachoques, a ver si nos hemos chocado, y no deja seguir la ejecución al bloque siguiente hasta que se cumple la condición.

Curiosamente, **este bucle no es como los que usamos antes** (que tenían un número de repeticiones fijas).

En este tipo de bucle el número de repeticiones puede ser muy pequeño o muy grande. Por ejemplo, si nuestro sistema mira cada décima de segundo al parachoques, imaginad cuántas veces ha mirado en una ejecución. Y además será un número diferente según esté el obstáculo más cerca o más lejos.

A esto se le llama BUCLE CONDICIONAL.

Que siga repitiéndose o se pare depende de que se cumpla una condición, no del número de repeticiones.

Nos gustan mucho, os lo confieso. Porque precisamente así podemos adaptarnos a distintas circunstancias en lugar de andar distancias fijas, giros de ángulos concretos, etc.

Vamos a poner otras soluciones pero esconden el mismo comportamiento condicional. “Haz hasta o mientras algo pase”.

NOTA: A cada repetición de un bucle se le llama ***iteración***, a partir de ahora usaremos también este término.

¿Te animas a buscar tú otra solución antes de mirar a la página siguiente?

SOLUCIÓN 2:



Lo que ocurre aquí es lo siguiente:

- Se comprueba que no está presionado el parachoques (porque dice HASTA QUE...)
- Si no lo está, ejecutamos UNA VEZ los bloques de dentro del bucle. En este caso, "conducir adelante".
- Al acabar de ejecutar los bloques de dentro, volvemos a comprobar la condición.
- Si sigue sin cumplirse, hacemos otra **iteración**, o sea, ejecutamos de nuevo, una vez, los bloques de dentro.

Y así, hasta que se cumpla la condición. En ese momento, abandonamos el bucle y seguimos con los bloques que tenga debajo el programa.

NOTA: Al analizar esta segunda solución se ve que la primera solución realmente ocultaba un bucle en un solo bloque.

Pero hay más formas de hacerlo... ¿Se te ocurren?

SOLUCIÓN 3:



Leamoslo si queréis, que se entiende bastante.

“Cuando empieza (la ejecución) y mientras no esté presionado el parachoques izquierdo, conduce adelante. Si eso cambia, para de conducir.”

De nuevo es un bucle condicional. Después de cada iteración (en nuestro caso sólo de un bloque, conducir adelante, se vuelve a comprobar la condición, y mientras se cumpla, seguimos en el bucle).

Es decir lo mismo con “hasta que” o con “mientras que”.

Lava vasos hasta que el fregadero esté vacío.

Lava vasos mientras el fregadero NO esté vacío.

Como son maneras contrarias de expresarlo, me aparece ese “NO”, que tengo que incluir en el programa.

Es un bloque que tenéis en el grupo “operaciones”.

Como veis por su forma es un bloque booleano, que solo puede tomar valores “verdadero o falso”.

Al ponerlo así toma el valor contrario a otro bloque.

Si el bloque azul es “verdadero”, el verde es “falso” y viceversa.

- Si es verdad que el parachoques está presionado, es falso que no esté presionado.
- Si es falso que el parachoques está presionado, es verdad que no está presionado.

POR SI ACASO:



Quizá hayas pensado que podrías usar este bloque y no te haya salido.

Si no es así, no te preocupes y sáltate esto, que ya lo veremos. Si es así, piensa que con este bloque solo se comprueba UNA VEZ si te estás chocando... y la clave para no matarse está en mirar y mirar y mirar... en repetir... en los bucles!

Aunque te confieso que con esto y alguna cosilla más... ¡sí se puede hacer!

Objetivos cumplidos.

- Entender el uso de sensores
- Practicar el sensor del parachoques
- Entender los bloques booleanos
- Entender distintas formas de bucles condicionales

SENSORES (el ojo de abajo)

Seguimos en la misma zona de juegos del playground (Castillo de choques dinámico)

Quizá ya lo habéis notado, pero si ponéis el coche en marcha podéis comprobar que el límite de este espacio no es una pared sino una línea roja EN EL SUELO, de forma que el parachoques no nos va a evitar caernos si llegamos hasta allí.

Si os parece bien, en este capítulo **en lugar de chocarnos y quedarnos ahí, os propongo que nos empeñemos en llegar hasta el otro extremo y evitemos los obstáculos que encontremos.**

Para evitar los obstáculos podríamos usar el parachoques, pero para la parada final... no.

Para esto usaremos un sensor de luz que tiene el robot apuntando al suelo (el ojoAbajo) y que es capaz de detectar colores.

Sacad el bloque correspondiente, os recuerdo que tanto el ojo (delantero o abajo) como el color hay que escogerlo adecuadamente de los menús desplegables.



Os propongo que probéis este código... y penséis conmigo por qué se ha caído por el barranco...



Vaya, Javi, pero si tiene una pinta estupenda.

“Repite conducir hasta que el ojo de abajo vea rojo”.

Perfecto. Sin fisuras... Salvo que se cae... Dadle una vuelta os veo en la próxima hoja.

¿Lo habéis visto?

Volvamos a lo primero que dijimos cuando empezamos

Lo peor de la programación es que hace lo que le dices y lo mejor es que hace lo que le dices.

Pero sólo lo que le dices...

Ese robot ha recibido una y otra vez la orden de andar. “Tira p’alante, que no hay rojo”, “Tira p’alante, que no hay rojo”, “Tira p’alante, que no hay rojo” y otra vez, y otra vez... órdenes redundantes en el sentido de que YA estaba andando, pero bueno, hay que mirar todo el rato.

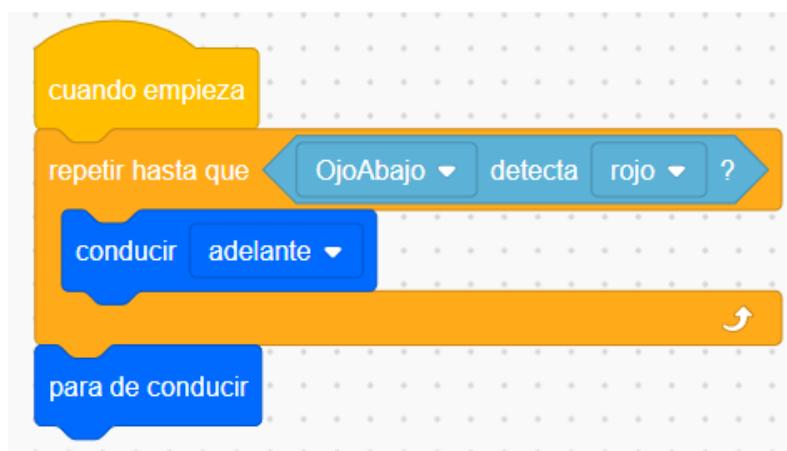
¿Qué pasa cuando termina el bucle?

Que ya no le dan más veces esa orden de “Tira p’alante, que no hay rojo”, PEEEERO, ¿¿hay alguien que le diga EXPLÍCITAMENTE que se pare??

No. Y por eso sigue andando.

Ya nadie le repite la orden, pero estaba andando y no le han dicho que deje de hacerlo, así que sigue andando.

El código adecuado sería así:



Esto nos será útil en la meta, pero falta ver cómo evitamos los obstáculos.

Hagamos la parte de esquivar obstáculos y luego vemos cómo las juntamos.

Yo os propongo.

- Esperar a que choquemos.
- Nos paramos.
- Giramos a la derecha, andamos un poco.
- Giramos a la izquierda (para ponernos de frente de nuevo) e intentamos seguir.

¿Intentáis primero antes de pasar a la solución de la página siguiente?

SOLUCIÓN:

Digamos que esta sería la idea de “el regate”



Tiro para delante.

Espero hasta chocar

Paro

Giro derecha

Avanzo un poco

Me pongo de frente de nuevo.

Ahora la idea sería repetir esto, tantas veces como fuera necesario.

Por ejemplo, si el obstáculo es muy ancho, después de andar un poco hacia un lado, al ponerme de frente de nuevo, me encontraré con que aún lo tengo delante, así que tendré que repetir la operación...

¿Cuántas veces? Las que sean necesarias.

¿Y si no termina nunca? Pues por siempre.

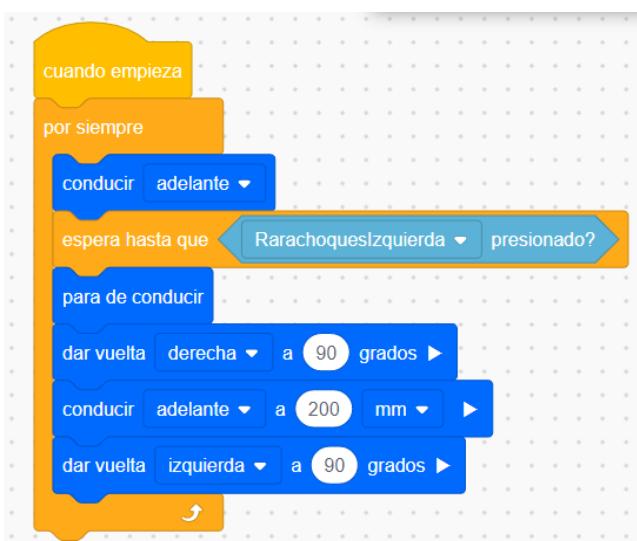


Esto es lo que se llama un BUCLE INFINITO.

Hace una iteración tras otra, sin pararse en el número, ni en si se cumple o se deja de cumplir alguna condición. Simplemente se repite infinitamente.

Está claro que alguna cosa nos tendremos que inventar para que se pare en algún momento, pero la idea es tener a algo con una vigilancia constante, sin cesar.

Así tendríamos esto:



Alguien podría pensar que me falta un “conducir adelante” al final del bucle, cuando ya estamos de frente otra vez. Pero fíjate que no hace falta, porque cuando nos ponemos de frente se acaba la iteración y tenemos que empezar con la siguiente que justo su primer bloque es... “conducir adelante”.

Si ponéis ese bloque al final también funciona, pero es innecesario.

Y esto sería un “esquivador eterno”...

Pero el suelo no es infinito y se cae.
¡Arreglámoslo!

¿Y si sustituimos el bucle infinito por un bucle condicional que esté mirando el sensor del suelo?



Ñam, tiene muy buena pinta.

Probadlo

¿Qué tal?

Tortazo épico, ¿verdad?

NO FUNCIONA.

Para saber qué es lo que falla, os animo a que miréis el código según se mueve el robot y veáis qué bloques se están ejecutando en cada ocasión.

Intentad darle una vuelta antes de seguir leyendo.

¿Cómo lo veis? Habéis averiguado lo que pasa?

Os cuento:

Cuando llega a la frontera el pobre robot está mirando a ver si se choca en lugar de estar comprobando el suelo.

¿Por qué ocurre esto?

En un bucle condicional se trabaja así, como ya hemos visto: se lleva a cabo una iteración completa, se comprueba la condición y se comienza otra iteración.

De forma que mientras estamos dentro de una iteración la condición NO se evalúa.

Dentro de nuestro bucle tenemos un “ESPERA” y ahí se queda parada la iteración, esperando un choque.

Solo se termina la iteración cuando se choca con algo, ejecuta el “regate” y en ese momento y sólo en ese momento, al final del regate, es cuando mira al suelo a ver si está rojo y, si no lo está, vuelve a quedarse esperando un choque.

De esta forma, lo más normal será que llegue al borde esperando un choque... y se caiga.

Necesitamos evaluar AMBAS cosas durante todo el viaje: si hemos llegado a la frontera y si nos vamos a chocar. No podemos quedarnos esperando a ver si sucede una o la otra.

¡Así que HAGAMOS DOS HILOS DE PROGRAMACIÓN!



Cuando le demos al PLAY empezarán a ejecutarse estos dos hilos al mismo tiempo, mandando órdenes al robot según los valores de los sensores.

¿Es esto posible en todos los lenguajes de programación o en todos los entornos? No.

Pero como en este sí, pues aprovecho para que lo veáis: un programa MULTIHILO.

Os animo a que observéis qué bloques se ejecutan en cada momento y si entran en conflicto o no, y cómo se resuelve la cosa. Es curioso. Os avanzo que funciona.

FUNCIONAMIENTO.

Cuando le damos al PLAY el coche recibe la orden de avanzar y tenemos a un hilo esperando a chocarse y al otro esperando por el color rojo.

Cuando se choca, se ejecuta el regate, con sus paros, giros y demás, sin que el otro hilo intervenga, ya que sigue esperando a que el suelo se vuelva rojo.

Cuando ya no quedan más obstáculos, el coche está andando porque es la última orden que recibió, pero el primer hilo está esperando a que se choque sin ordenarle nada más.

En el momento en el que toca rojo, el segundo hilo manda la orden de paro y el coche se para, el primer hilo sigue en funcionamiento, esperando un choque que nunca más se producirá y sin dar una orden que pudiera contradecir al segundo hilo.

VENTAJAS:

Es cómoda a la vista, se entienden bien los dos procesos.

DESVENTAJAS:

Tiene problemas de “conurrencia”, de conflictos entre los hilos.

No siempre es posible por el lenguaje o el entorno.

¿Se te ocurre cómo hacerlo con un hilo solo?

Pista: Para eso necesitamos a los CONDICIONALES

Objetivos cumplidos.

- Sensores de color
- Bucles infinitos
- Programación multihilo

CONDICIONALES

Este es el bloque condicional más sencillo. Sirve para tomar decisiones.



Si es verdad lo que pongo en el hueco (¿veis que tiene forma de booleano?) entonces ejecuto los bloques que meto dentro.

Si no es verdad, no hago nada y paso al bloque que haya debajo del condicional.

Por ejemplo: Si he chocado, me paro.



Pero a veces nos interesa también hacer algo si no se cumple la condición, entonces usamos este otro bloque, más completo.

Por ejemplo: Si he chocado, me paro. Si no, avanzo.



Este tipo de decisiones son muy comunes en los programas.

Tomo un dato del entorno, lo evalúo y, según lo que salga, hago una cosa u otra.

Por ejemplo, te preguntan la edad en la puerta de un local. Si eres mayor de edad, puedes pasar. Si no, no puedes pasar.

Pensad que en este ejemplo último, es una decisión que se toma una vez. Dime tu edad, aplico el condicional y fin del asunto.

Esto en robótica no suele ser así, sobre todo con sensores, porque los valores cambian y puedo querer que mis decisiones cambien.

Imagina un Perrito robot que quiero que me siga y se quede pegado a mi pie.

Yo le pondría, precisamente, este condicional



Si me estás tocando el zapato, quédate quieto, pero si no, avanza hasta que llegues hasta donde estoy.

El problema de estos condicionales es que, insisto, sólo se evalúan una vez. Así que, cuando encienda el robot, mirará a ver si tiene que andar o parar y se quedará así para siempre.

Probemos con nuestro robot.

Si le ponéis esto, al arrancar tomará la decisión y quedará tomada. Como no está tocando ningún obstáculo, empezará a andar y se llevará por delante lo que se encuentre.



¿Cómo lo arreglamos? Metiendo este condicional dentro de un bucle para que se evalúe una y otra vez, una y otra vez, dándole la oportunidad al “robot” de cambiar de opinión.

Veréis que funciona.

Pues aquí tienes la pista que necesitabas para hacer el esquivador que se detenga en el borde con un solo hilo...

HAGAMOS UN ESQUIVADOR CON UN HILO, BUCLE INFINITO Y CONDICIONALES

Anímate a hacer alguna prueba antes de mirar la solución

SOLUCIÓN:

La estructura es: Un bucle infinito, con una sucesión de condicionales que van mirando cada sensor y tomando la decisión que sea necesaria.



En cada iteración.

- Primero miro si he llegado al final. Si he llegado, paro. Si no, avanzo.
- Luego miro si me he chocado. Si es así, ejecuto el regate, si no, no mando ninguna orden.

¡Y vuelta a empezar!

De esta forma la vigilancia es constante. De ambas cosas. No es exactamente a la vez, pero casi, ya sabéis lo rápidos que son los ordenadores.

Podrías pensar que le podíamos poner el comando de avance a la parte del choque o a ambas. Os animo a que lo probéis y veáis los problemas que da.

Supongo que habréis visto que la estrategia no es perfecta, nos chocamos por el parachoques izquierdo, durante el regate también nos podemos chocar...

Un detalle curioso es que, cuando llegamos al final, el robot está parado, pero el programa no. Sigue evaluando una y otra vez los condicionales, saliéndole que hay que seguir parado. Esto no es óptimo tampoco, claro.

Es sólo un ejemplo sencillo para aprender mientras nos entretenemos. Si queréis mejorarlo puede ser también un interesante ejercicio, por supuesto. A programar se aprende practicando ;)

Objetivos cumplidos.

- Usar condicionales if, if-else
- Controlar varios sensores en un mismo hilo

SENSORES (distancia)

Ir chocando con las cosas es poco “elegante”, así que, dado que tenemos un sensor de distancia por ultrasonidos en nuestro robot, nos pararemos antes de chocar sabiendo que hemos encontrado un obstáculo.

Este sensor arroja dos posibles resultados.

- Si está viendo un objeto cerca o no (valor booleano -verdadero o falso-)
- A qué distancia está (valor numérico)

Para este juego vamos a usar este último. Daos cuenta de la forma del bloque redondeada indicando que se trata de un valor numérico.



En la práctica estos sensores no pasan por todos los valores de las distancias en sus medidas, así que no vamos a usar condiciones del estilo “distancia = 15 cm” porque nunca llegarían a cumplirse. Si queremos no acercarnos más de 15 centímetros, mandaremos parar cuando se cumpla “distancia < 15 cm”.

NOTA:

P.ej.: las medidas que va arrojando el sensor podrían ser 16,3 15,9 15,3 14,8 14,6 ...

Si ponemos distancia = 15 no se cumple nunca la condición y ¡ya estamos más cerca de lo que queremos!

En cambio si pongo que se pare cuando distancia < 15, en 14,8 ya se para. No consigo una distancia exacta (es imposible), pero sé que va a parar aproximadamente donde quería.

Vamos a usar este sensor para **resolver un laberinto, sin saber cuál es su forma**.

Usaremos una técnica clásica. Sencilla, aunque a veces tarda un poco y falla con algunos tipos de laberintos (los que tienen “islas”, p.ej., por si queréis investigar más)

Se trata de ir pegado a una pared, bien a la izquierda o a la derecha, y no dejar nunca estar pegado a la pared por el lado elegido.

Si pudiéramos mover los sensores de sitio pondríamos uno a un lado del robot para controlar que vamos pegados a la pared. Como no puede ser, **lo que haremos será cabecear un poco después de cada avance, a ver si seguimos viendo la pared por ese lado**.

Pongamos primero la zona de juegos en el playground “Laberinto de pared dinámico”.

Si le dais a recargar, veréis que salen distintos laberintos. La idea es hacer un programa sencillo que sea capaz de resolverlos todos.

Vamos a aprovecharnos de que el laberinto está hecho con paredes perpendiculares y usando una cuadrícula de 250 mm. Haremos que nuestros “pasos” tengan esa longitud y así evitaremos chocar con las paredes.

Vamos a elegir pegarnos a la pared de la derecha, como hemos dicho podéis hacerlo igual pero con la pared izquierda. Según sea el laberinto una estrategia será mejor o peor por puro azar, pero ambas nos llevarán a la salida.

¡Vamos a ello! Solución tomada de [aquí](#)

Esta idea es la “madre del cordero”



Veamos la estrategia

1. Si tengo espacio por delante..

- Ando un paso
- Giro a la derecha

2. Si no..

- Giro a la izquierda

Se trata de avanzar e intentar girar a la derecha para ir siguiendo la pared. En caso de no poder andar, al girar a la izquierda sigues manteniendo la pared a tu derecha y quizás por ahí encuentres “hueco” para seguir en la próxima iteración.

Porque claro, esto hay que repetirlo en un bucle hasta que encontremos la salida.

En el laberinto el suelo de la meta está pintado de rojo y negro. Así que si repetimos el “avance y cabeceo” hasta que el suelo se vea rojo, habremos conseguido programar la estrategia ganadora.



Como veis he aumentado la velocidad del robot porque tarda un poquillo. Los valores por defecto están al 50%.

Hemos empezado girando también para evitar que el robot se salga por la puerta que está entrando. (en algunos laberintos pasa si no se pone, compruébalo)

Y al final del bucle, tengo que decirle que se pare claro... para que no siga avanzando por la pared.

Si queréis profundizar:

- Probad en los distintos laberintos a ver qué tal funciona.
- Podéis probar a hacerlo “a izquierdas”.
- Bajad el “lápiz” y ved qué camino ha ido siguiendo.
- Calculad tiempos y distancias recorridas
- Intentad buscar otras estrategias de resolución de laberintos e intentar programarlas.

En todo caso, no debe pasarse por alto la **potencia de un programa tan pequeño** para resolver (casi) **cualquier laberinto** que se le presente sin saber de antemano cómo es.

Este es el estilo de robots al que queremos llegar, que sea capaz de moverse por entornos desconocidos, tomar datos del ambiente y decidir adecuadamente su comportamiento.

Objetivos cumplidos:

- Usar el valor numérico de la distancia que da el sensor de distancia
- Resolución de laberintos desconocidos con algoritmo general
- Uso de condicionales

ACTUADORES (el imán)

Además de sensores para saber qué pasa en su entorno, los robots tienen actuadores para poder hacer cosas en él.

Ya hemos usado los motores de sus ruedas, y “la pluma”. Otros robots suelen tener luces para señalizar comportamientos, dar información, alarmas. Pueden tener pantallas donde mostrar valores o resultados de operaciones, etc.

En este juego vamos a usar un imán para poder recoger objetos y trasladarlos a otras posiciones.

Entendemos que están simulando un electroimán, de manera que si le damos corriente atraerá el objeto y si lo apagamos dejará de atraerlo cayendo al suelo.

Vamos a seleccionar la zona de juego del playground “Movedor de discos”.

Aunque por la cuadrícula y el punto de vista podríamos hacer un programa que fuera al lugar en el que están los discos directamente, es más interesante, hacer que el robot los “busque” y actúe en consecuencia cuando se los encuentre (recogiéndolos y llevándolos al cuadrado marcado de su color).

Como siempre, **te animo a que hagas una prueba tú antes de mirar la solución.**

PROGRAMA:

- AVANCE HASTA QUE ENCUENTRE UN DISCO
- LO RECOGE
- LO TRAE DE VUELTA
- LO SUELTE AL LLEGAR

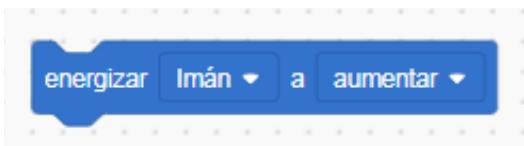
SOLUCIÓN

La primera parte se parece a cosas que ya hemos hecho. “Andar hasta que un sensor detecte alguna cosa”

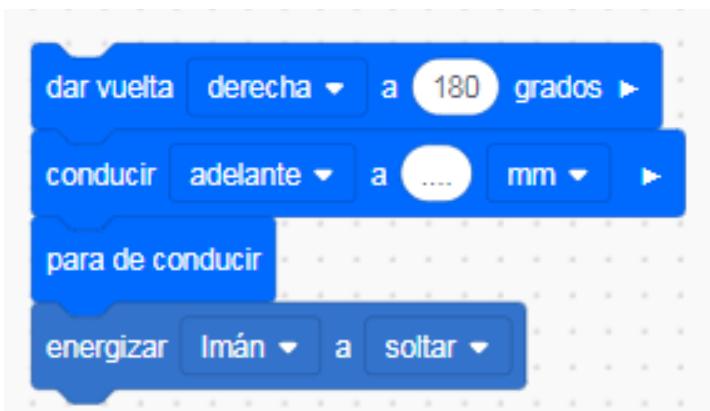
Sería algo así



Después debemos “encender el imán” para que recoja el disco. Aquí la traducción deja un poco que desear, las opciones son “aumentar” que sería “encender” y “soltar” que sería “apagar”.



Al encender el imán el disco se pega, ahora nos tocaría andar hacia atrás. Y este sería el problema más interesante de este juego, porque tenemos que volver la misma distancia que hemos andado, pero no sabemos cuánto es.



Este sería el código de la parte final

Damos la vuelta, andamos una distancia (que aún no sabemos).

Paramos.

Soltamos el disco.

Veamos cómo saber la distancia recorrida.

Para esto tenemos que usar variables y apuntar la posición inicial y la posición en la que encontramos el disco. La diferencia entre ambas será la distancia que tengamos que recorrer de vuelta.

Para guardar valores usamos las VARIABLES, ya sabes.

Así que necesitamos dos variables.

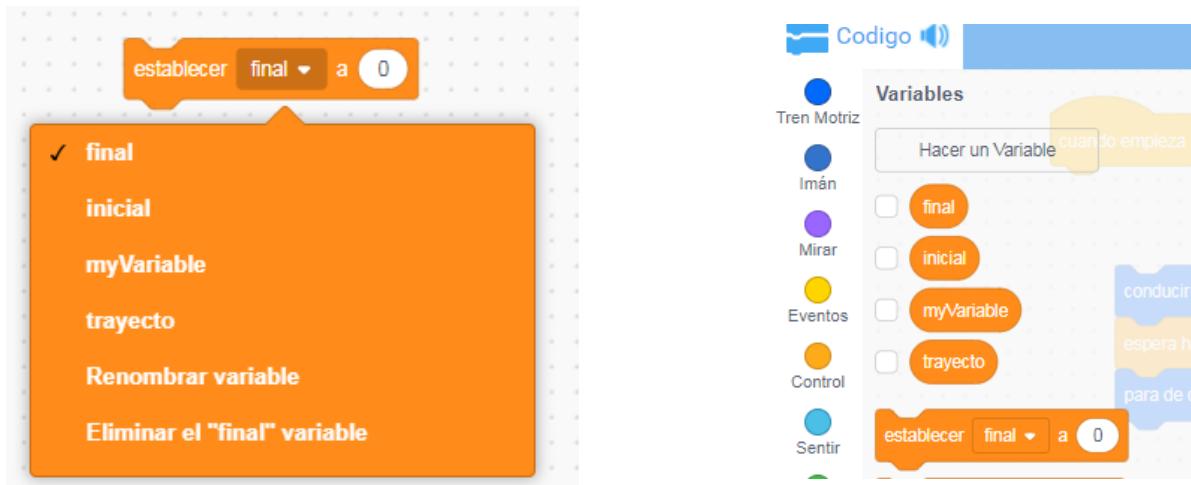
- Posición inicial
- Posición final
- Distancia recorrida (la resta de las dos anteriores)

Como ya dijimos lo mejor es darles nombres que nos hagan deducir de qué se trata.
posicionInicial, inicial_pos, inicio, etc.

Ve al grupo de bloques “variables” y crea esas tres variables.

Yo he elegido como nombres: inicial, final y trayecto.

En el bloque de establecer variable puedes elegir en el menú desplegable cuál quieras usar y en el grupo variables tienes los bloques para usar su valor.



Lo que vamos a hacer es “medir” en qué lugar estamos tanto al empezar como al encontrar el disco.

Para eso usaremos en el grupo de bloques “sentir” tienes un sensor de posición con el que puedes obtener en qué lugar estás en cada momento. Este valor también se muestra en la parte superior del playground, donde pone “Location”, pero hay que usar este bloque para poder “decírselo” al robot.



Así que en esos dos momentos “tomaríamos nota” con el siguiente bloque



Y para saber el trayecto que hay que recorrer de vuelta, calcularíamos la diferencia usando el bloque resta que tenemos en el grupo de bloques “operadores”



Quedaría así el programa completo.



Como ves con este programa tan sencillo, tenemos una capacidad de moverse en un entorno desconocido bastante interesante.

No sabe dónde estará el disco, pero es capaz de detectarlo y calcular la distancia que tiene que volver.

Hagamos otro programa

PROGRAMA

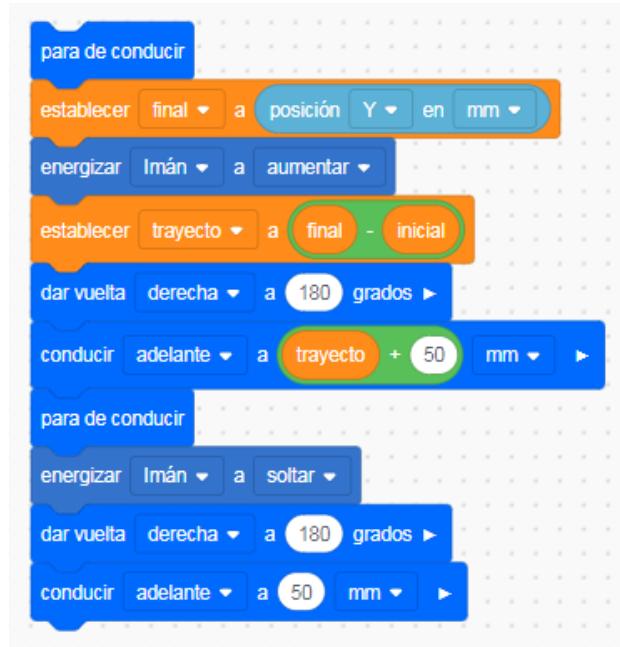
- Recoge discos uno a uno hasta que ya no queden más.

De nuevo te animo a que intentes.

Pista: Piensa que cuando ya no haya más discos, lo que ocurrirá es que se encontrará con la pared del final.

SOLUCIÓN:

El procedimiento de recogida funciona bien, pero cuando haya dejado un disco en “casa”, si le decimos que vaya a buscar otro, igual lo que hace es encontrar el que acaba de traer y se lía. Así que, vamos a dejarlos un poco más atrás y luego vamos a avanzar un poco antes de ponernos a buscar discos nuevos



Los dos ajustes que decíamos son:

Ese bloque “trayecto +50 mm” para dejarlo un poco más atrás.

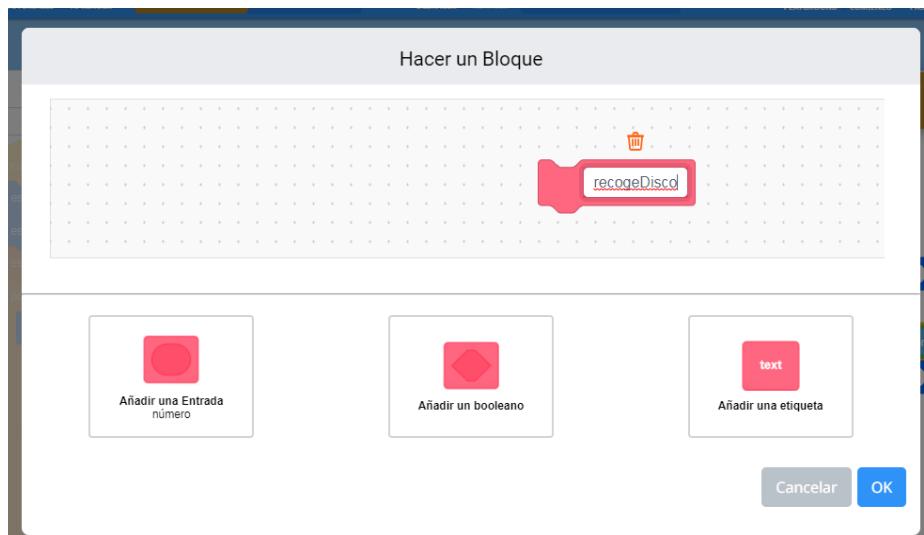
El bloque final que avanza 50 mm desde donde ha dejado el disco.

Ahora la idea sería meter esto en un bucle para que se hiciera el número de veces necesario.

Vamos a introducir un elemento nuevo que nos será muy útil, CREAR BLOQUES NUEVOS.

A este conjunto de bloques lo hemos llamado “procedimiento de recogida” y podríamos crear un bloque que se llame “recogerDisco” y así nos queda todo resumido y más claro.

Si vais al grupo de bloques “Mis bloques” y creáis uno, os va a pedir que le pongáis nombre.



Entonces os aparecerán dos bloques. Uno en el grupo de bloques y otro en la zona de programación.



Lo que hacemos es poner nuestro algoritmo de recogida debajo de la definición y usaremos el bloque “recogerDisco” cuando lo necesitemos.

Ahora podríamos definir otro bloque donde explicaríamos cómo hacemos la vuelta.



La idea es que, cuando llegue a la pared, dará media vuelta, volverá, y se parará cuando esté a 100 mm de la pared de la llegada después de ponerse otra vez mirando hacia delante.

¿Te imaginas ya cómo sería el programa general? A la vuelta, la solución.

De esta forma el programa general queda muy “limpito”



Aumento la velocidad primero para no esperar tanto.

Mido la posición inicial (la necesito para calcular las vueltas)

Hago un bucle infinito donde:

- Empiezo a andar
- Miro si tengo un objeto debajo.
- Si lo tengo aplico el protocolo “recoger”
- Después miro si estoy muy cerca de una pared
- Si lo estoy, es que ya no hay más discos y queremos salir del bucle. Para salir del bucle usamos el bloque “romper” (en inglés, BREAK).
- Una vez fuera, aplicamos el procedimiento de regreso.

Esto que hemos hecho, es habitual en programación.

Hemos separado nuestro problema en TRES problemas menores. “*Divide y vencerás*”

- El flujo general del programa
- El procedimiento de recogida
- El procedimiento de vuelta

De esta forma nos podemos concentrar en cada cosa por separado, además de resultar visualmente mucho más claro cómo se comporta el programa general.

Objetivos cumplidos:

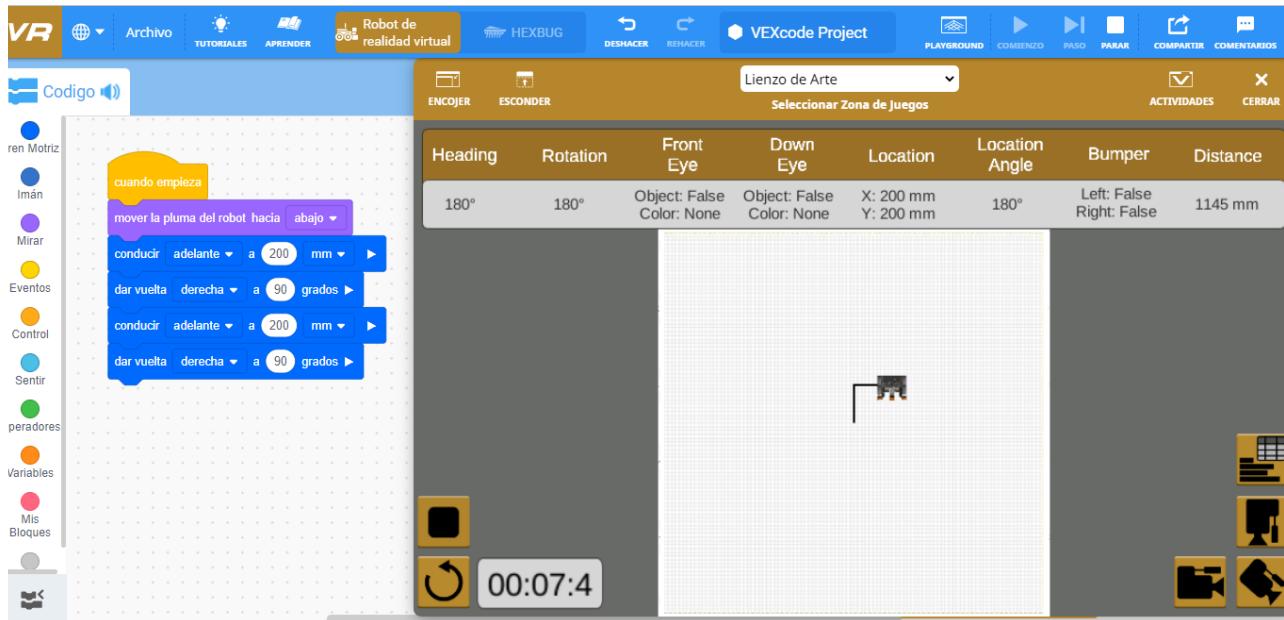
- Manejar variables
- Obtener la posición del robot
- Bloques definidos por el usuario (FUNCIONES)
- Bloque BREAK para salir de bucles.
- Cambio de valor de las variables

Modificar variables durante el programa. Espiral

Como ya hemos dicho las variables reciben ese nombre porque pueden cambiar su valor durante la ejecución.

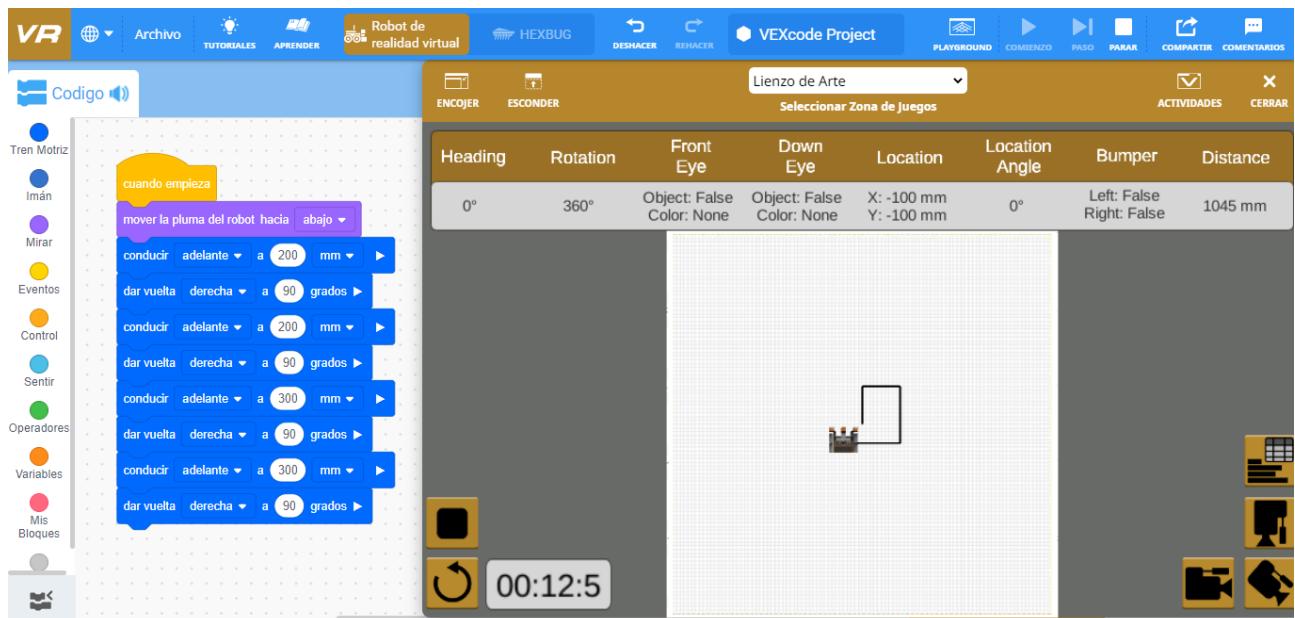
Vamos a intentar dibujar una espiral para mostrarlo.

Empieza ejecutando estos bloques, te dibujará dos lados.



Si queremos dibujar una espiral NO basta con repetir estas instrucciones... haríamos un cuadrado. Lo que hay que hacer es bajar UN POCO MÁS de la distancia que subimos.

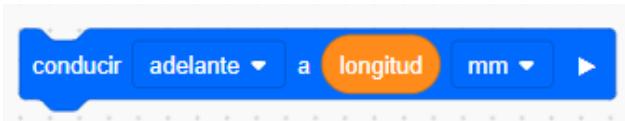
Vamos a probarlo.



Ahora tendríamos que hacer otros dos lados con UN POCO MÁS de distancia de nuevo.

No podemos hacer directamente un bucle porque las distancias de avance son diferentes cada dos lados que dibujamos. Pero esto se arregla estupendamente con una variable.

Vamos a crear una variable que se llame longitud. Y pondremos



Haremos un bucle de la siguiente forma



PEEEERO aún nos falta un detalle. Si ejecutáis esto os va a hacer cuadrados.

Tenemos que aumentar la longitud al final de cada iteración del bucle.

Así que elegiremos este bloque, que aún no habíamos usado.



Este bloque lo que hace es añadir uno a la variable elegida.

Si antes del bloque la variable longitud vale 7, después de él, la variable longitud valdrá 8.

¡Ya lo tenemos! Prueba el siguiente código:



Hay que empezar bajando el lápiz, que si no, no pintamos.

Establecemos la longitud del primer trazo.

Vamos a repetir diez veces

En la primera iteración hacemos dos trazos de 200.

Después de hacer esos trazos, le sumamos cien unidades a la longitud.

En la segunda iteración, hacemos dos trazos de 300 y volvemos a sumar cien unidades a la longitud.

Si queréis lo podemos dejar mejor todavía. Vamos a dejar como variable ese “aumento” de la longitud.

Crea otra variable que se llame paso.

Dale un valor antes del bucle.

Pon la variable paso donde estaba el “100”.



De esta forma puedes elegir cambiando los valores iniciales de longitud y paso la espiral que quieras hacer.

Te dejo como desafío que crees otra variable para elegir cuántas “vueltas” quieras dar.

Le das un valor después de los valores que le has dado a longitud y paso, y la pones en el bucle repetir donde está el “10”.

Objetivos cumplidos:

- Cambiar el valor de una variable durante la ejecución del programa.

Búsqueda

Vamos a programar el robot para que busque un objeto en el suelo (en nuestro caso una línea de color).

Para eso seleccionaremos la Zona de juegos en “Detector de línea”.

Hay diversos patrones para “buscar” según en la situación en la que te encuentres.

Vamos a probar el típico que se usa cuando no se conoce la distancia a los bordes o ni siquiera estás en una región limitada por bordes. Se trata de hacer una espiral que va creciendo.

Búsqueda con espiral creciente

Podemos usar el mismo programa de movimiento en espiral que vimos en el apartado anterior.



Por supuesto, habrá que modificarlo.

Ahora mismo, baja la pluma para pintar, establece la longitud de los segmentos en 200 y el paso (el incremento que irá añadiendo) a 100. Despues hace diez veces una “L”, aumentando la longitud al final de cada iteración para que la espiral vaya creciendo.

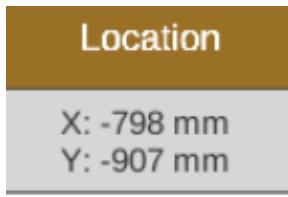
Primera cosa que modificaremos. Que no queremos repetir diez veces, sino hasta que encuentre lo que andamos buscando. Por ejemplo, el color rojo. Por lo tanto nos quedará así:



Como ves hemos cambiado un bucle con número de repeticiones fijas, por un bucle condicional.

Probemos a ver qué tal va.

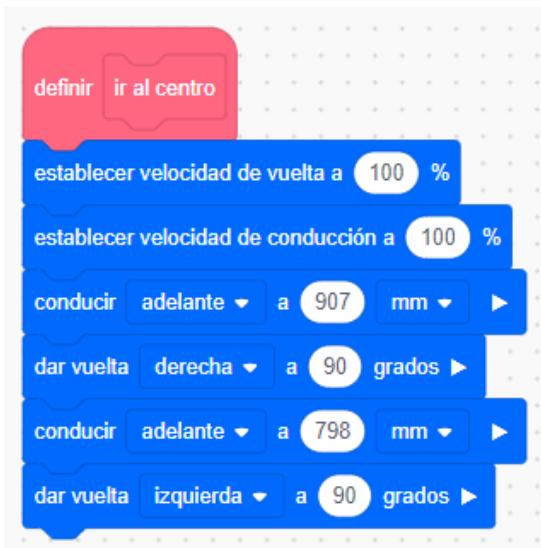
¡Espera, aún no podemos! Fíjate que *tenemos el robot en una esquina*, llevémoslo primero al centro.



Como podemos leer la posición exacta en la parte de arriba de la ventana “playground”, sabemos lo que tenemos que desplazarnos.

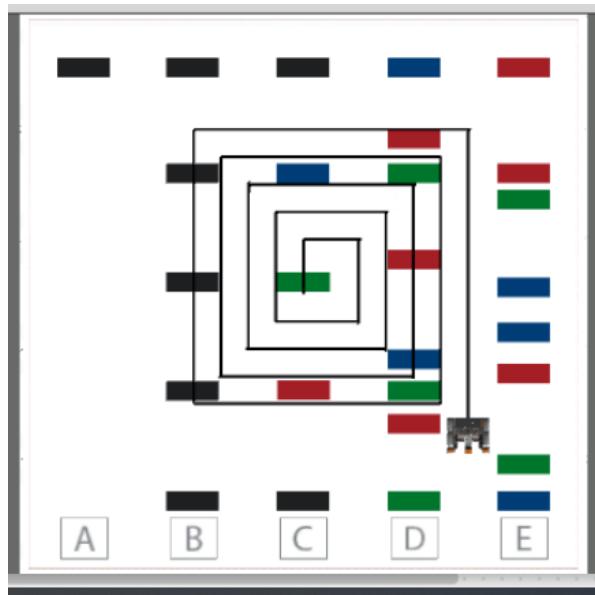
Bastará con movernos hacia la posición X = 0, Y = 0

Sería así:



Me disculparéis que he aprovechado para aumentar la velocidad y definir ese movimiento como un bloque ;)

Pues, como podéis ver... NO FUNCIONA. Ha pasado por el rojo varias veces sin detectarlo.



¿QUÉ SUCDE?

Pensémoslo, es sencillo. El robot solo hace... ya sabéis, lo que le decimos. Repasemos el programa como si las órdenes nos las dieran a nosotros y fuéramos nosotros los que estuviésemos buscando ese color rojo en el suelo.

Lo copio de nuevo para analizarlo.



Baja la pluma, ok.

Establece el valor de la longitud y el paso, ok.

Mira a ver si hay rojo y ejecuta los comandos de dentro del bucle... y aquí está el asunto.

Una vez que no ha detectado rojo, hace los dos tramos del camino SIN VOLVER A MIRAR SI HAY ROJO hasta el final del bucle.

No va mirando según anda.

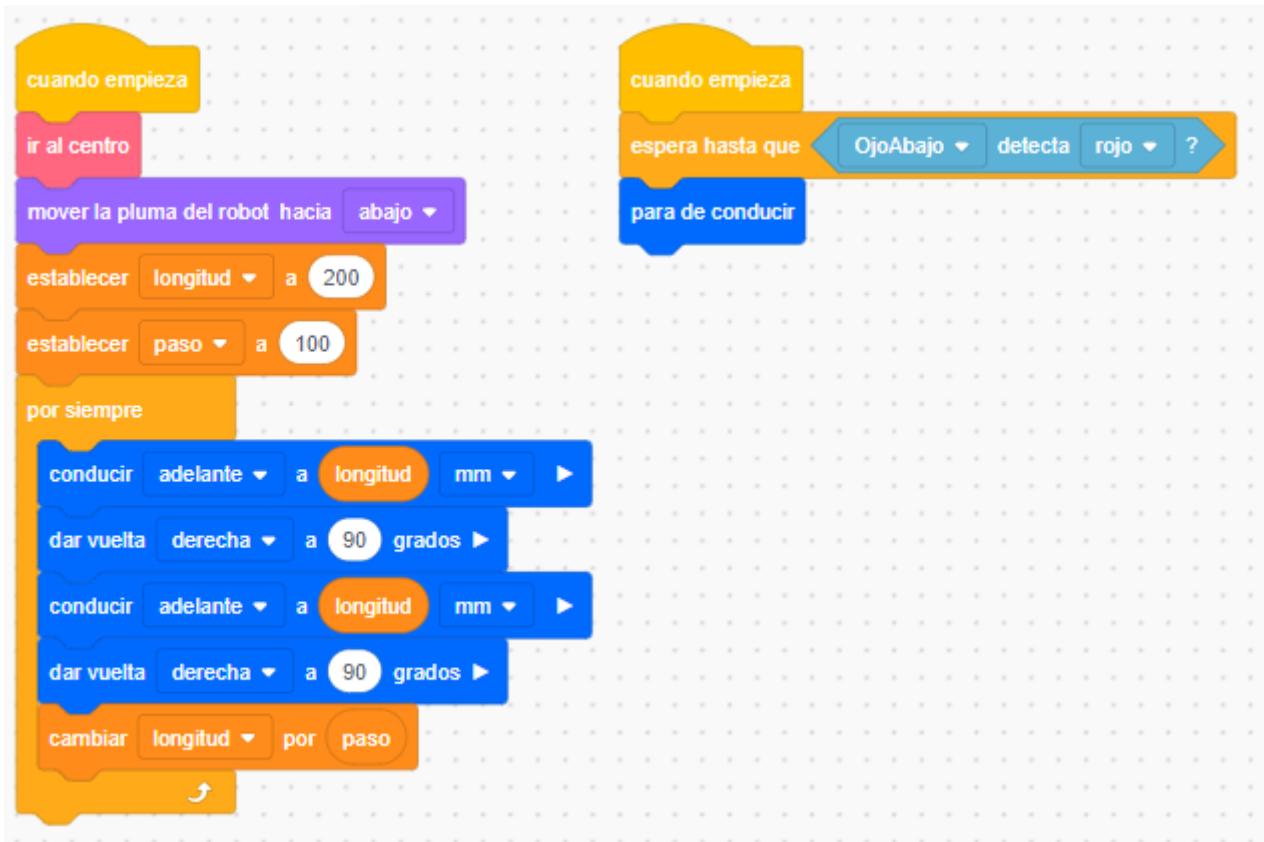
Si no da la casualidad de que justo al final de una iteración esté sobre el una línea roja, nunca la encontrará.

Esta estrategia no nos sirve. Necesitamos estar mirando TODO el tiempo a ver si detectamos rojo.

Así que vamos a hacer una estrategia a DOS hilos.

En un hilo hacemos la espiral y en el otro hilo miramos constantemente si detectamos rojo.

Podría ser algo así. Probemos



Como veis, tampoco acaba de funcionar. Parece que se quiere parar, pero no se para.

Lo que sucede es que hay dos hilos ejecutándose a la vez. Uno le ha dicho que se pare, pero el otro sigue mandando órdenes para que se dibuje la espiral.

Si queremos que se detengan todos los hilos, tenemos que “parar” el proyecto entero.

Para eso tenemos que usar el bloque “Stop project” que se ha traducido incorrectamente como “proyecto de parada” pero debería decir “Parar proyecto” (Está en el grupo “control”)

Quedaría así:



¡Por fin FUNCIONA!

Verás que también se ha parado el cronómetro. Eso no nos había ocurrido hasta ahora, porque habíamos parado el robot, pero no habíamos detenido el programa.

Objetivos cumplidos:

- Búsqueda en espiral
- Programas multihilo. Conflictos
- Bloque “Parar proyecto”

ACUMULADORES

Te propongo una variante para el programa que acabamos de hacer.

En lugar de que se pare cuando encuentre el color rojo, que “apunte” en algún sitio que ha detectado un objeto, pero que siga haciendo la espiral. Al final que nos diga cuántos objetos se ha encontrado.

Ya sabes que cuando queremos “guardar” un valor usamos una variable. En este caso la inicializaremos a valor cero y después le iremos sumando uno cada vez que tengamos un encuentro. Este tipo de variables es muy común y se las llama acumuladores.

Podemos probar con este hilo a ver qué tal...



Tenemos un problema...

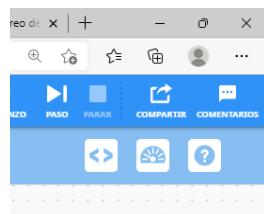
¿Dónde podemos ver el valor de esa variable?

Para eso necesitamos el “monitor”.

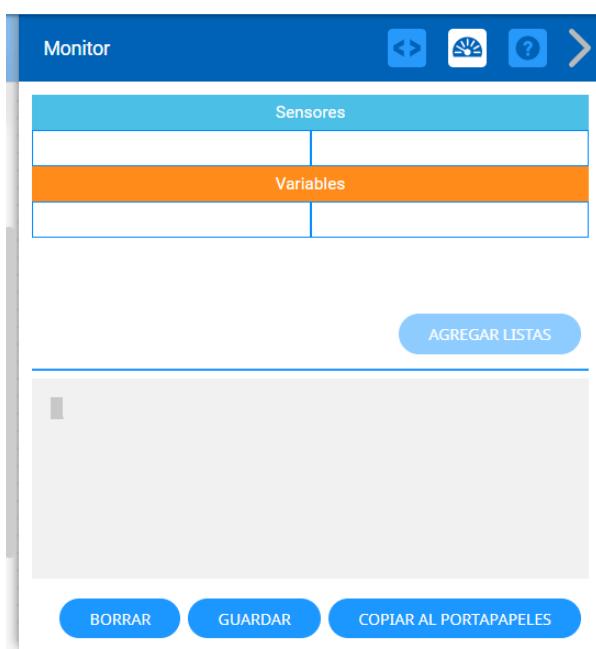
Vamos a ver cómo activarlo.

EL MONITOR

El monitor es una ventana donde nuestro robot puede mostrarnos valores de variables, valores de sensores, incluso escribirnos mensajes de texto. Para abrirlo tenéis que dar a ese ícono con forma de “abanico” que tenéis arriba a la derecha.



Y se abrirá esto:



Veis que tiene varias zonas.

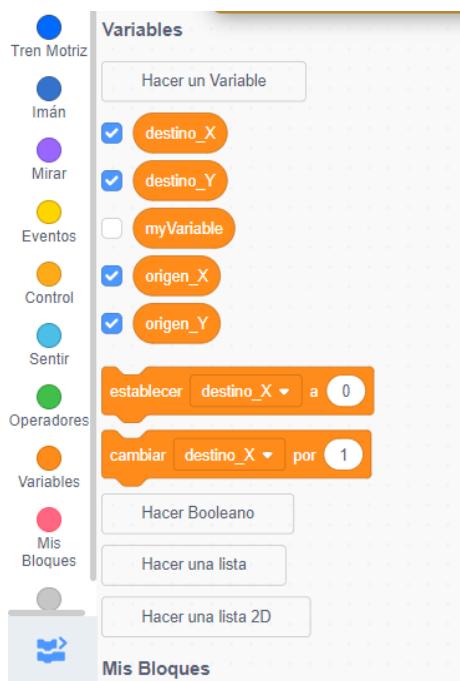
Una para mostrar valores de sensores

Otra para mostrar valores de variables

Una abajo para que nos escriba mensajes con texto.

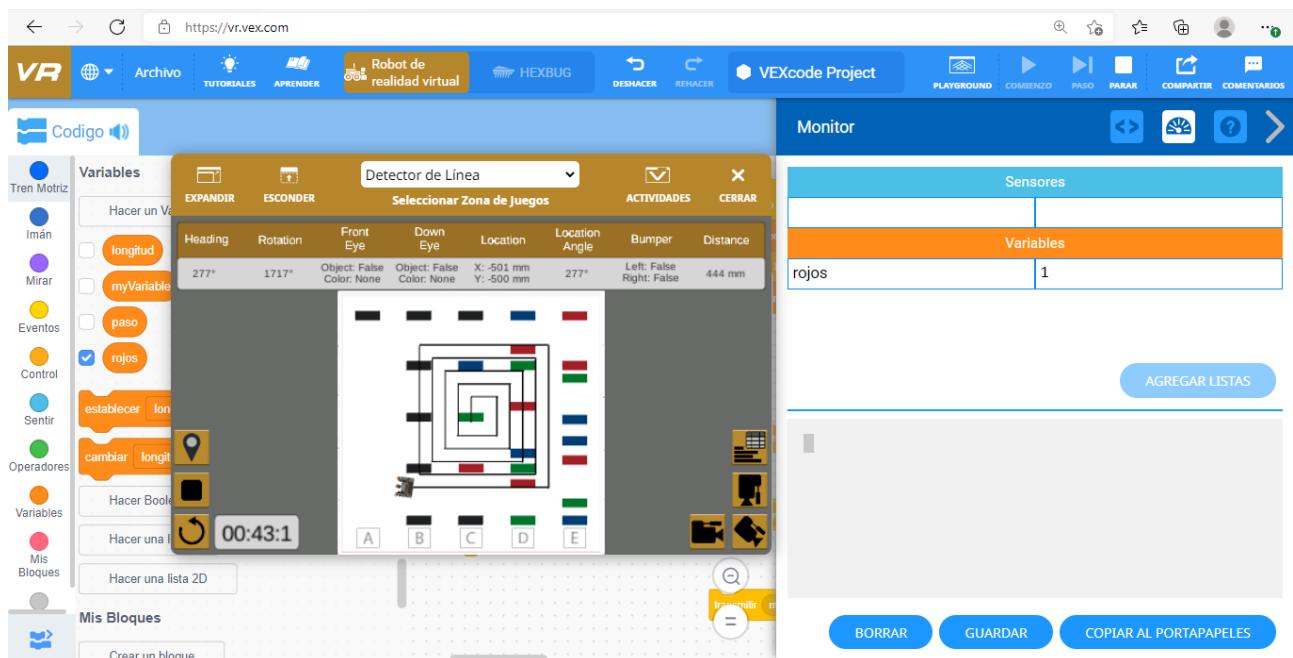
Para elegir qué sensores o variables queremos que se muestren, debemos “marcarlas” en la zona donde están los bloques de programación.

Id al grupo “variables” y veréis que al lado de los bloques de valor de las variables, hay unos cuadrados blancos. Haced click encima del cuadrado al lado de “rojos”, saldrá un “tick” y eso hará que esos valores se muestren en el monitor cuando ejecutemos el programa.



OJO: Tenéis que quitar lo de “esconder bloques” porque si no no podéis marcarlos.

Si ejecutáis, ahora veréis esto

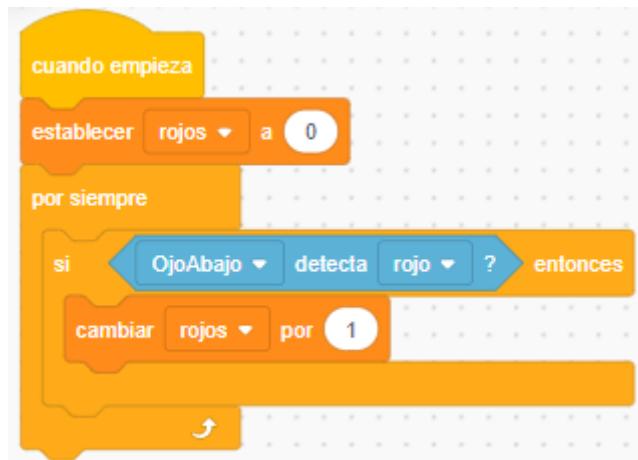


Pero, parece que sólo cuenta un encuentro...



con otro.

Vamos a necesitar un bucle



No vuelve a mirar a ver si se encuentra

Es lógico, nuestro primer intento no está bien. Fijaos, lo copio de nuevo.

Pone el acumulador a cero.

Espera al primer encuentro.

Suma uno al acumulador.

Y ya...

No vuelve a mirar a ver si se encuentra

En este caso va a repetir por siempre la pregunta “estamos viendo rojo”, en caso afirmativo sumará uno al acumulador.

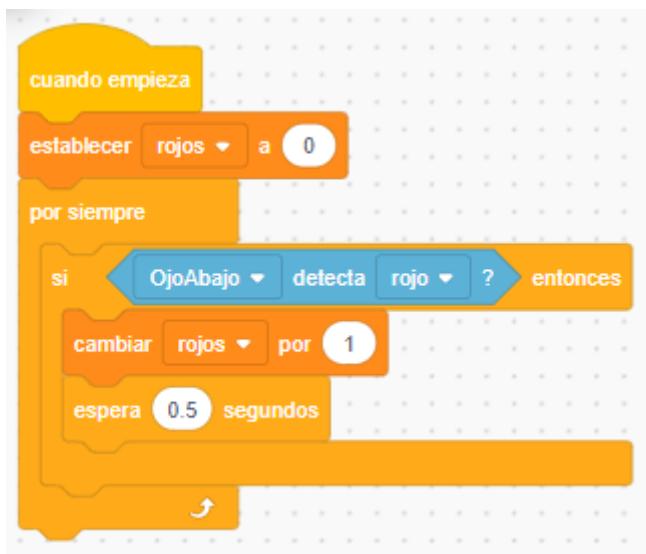
Probémoslo, a ver qué tal va.

Vaya, ejem... cuando pasa por el primer rojo cuenta... 28914. Un poco exagerado...

The screenshot shows the VEXcode VR software interface. On the left, the script editor displays a script with a 'repetir (10)' loop. Inside the loop, there is a condition 'si [color sensor (front)] detecta [rojo]' which triggers a 'cambiar [rojos] por (1)' block. The monitor on the right shows the 'Sensores' tab with 'rojos' set to 28914, indicating the script has run many cycles.

¿Qué pasa aquí?

Sencillo, el programa es muy rápido y mientras que está pasando le da tiempo a mirar y sumar todas esas veces. Vamos a arreglarlo de una forma sencilla.



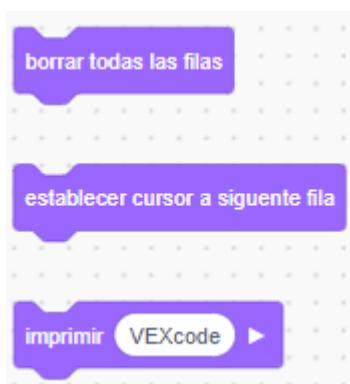
Si quieres, podemos añadir un hilo más, para que la espiral se pare cuando llegue a la pared.

Lo puedes hacer con el parachoques o, si quieres que seamos más finos, con el sensor de distancia.



Ya que tenemos el monitor abierto, te propongo una sutileza más. ¿Qué te parece si el propio robot nos DICE que ha terminado y cuántas líneas rojas ha encontrado?

Es sencillo de hacer con estos bloques



Con el primer bloque borramos mensajes de otras ejecuciones.

Con el segundo podemos cambiar de fila para escribir otra línea

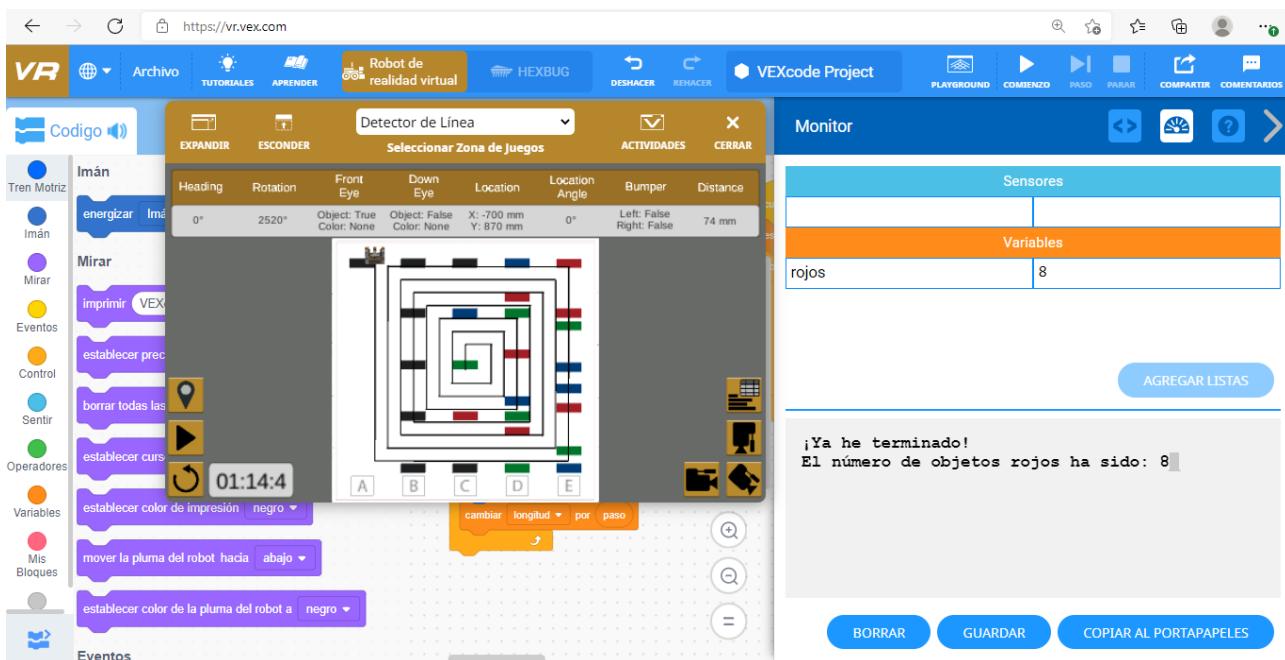
Con el tercero podemos escribir palabras, o bien meter bloques con esa forma redondeada, como los que dan los valores de las variables.



Empezamos borrando lo que pudiera haber escrito antes de nada.

Después, damos el mensaje de finalización, cambiamos de línea, escribimos otro mensaje y ponemos el valor de la variable “rojos”.

Quedando como sale en el pantallazo siguiente.



Objetivos cumplidos

- Variables acumuladoras
- Monitor de variables y sensores
- Sensibilidad y retardos
- Mensajes al usuario. Texto y variables.

Barrer la zona. Localizar y empujar.

Hemos sido muy sutiles... vamos a hacer un poco el cabra.

Elijamos la zona “Castillo de bloques dinámico”.

Saldrán construcciones en lugares aleatorios cada vez que reiniciemos.

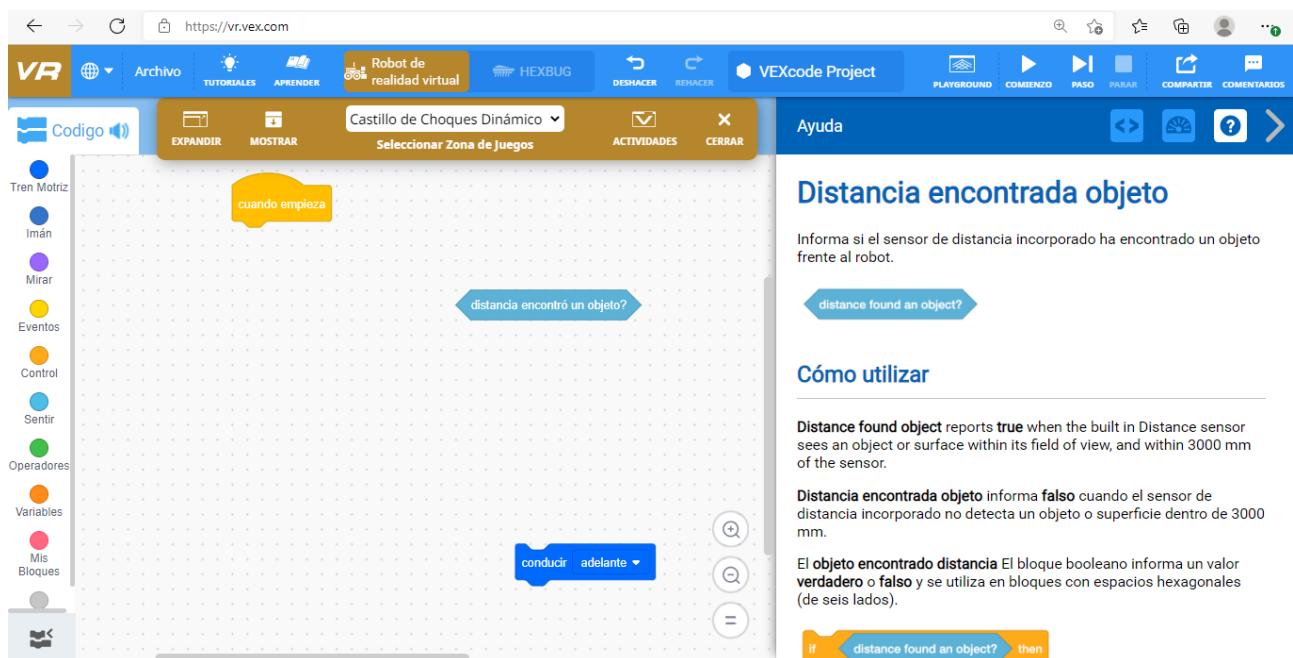
Vamos a “empujar al vacío” todos los bloques.

La idea será

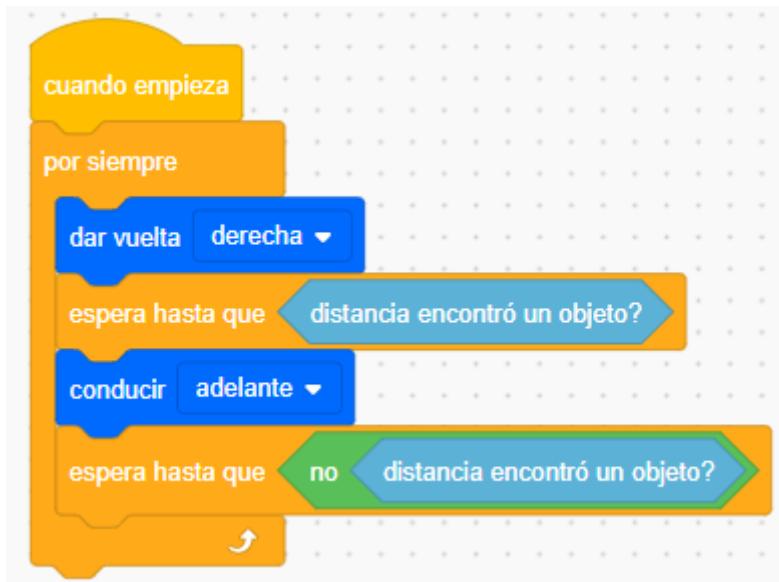
- Girar hasta que veamos un objeto.
- Ir hacia él
- Empujarlo hasta que se caiga.
- Buscar otro.

Para buscar un objeto usaremos el sensor de distancia, que tiene la facultad de darnos la distancia como ya hemos usado o bien, darnos un simple “Sí” cuando hay un objeto a menos de 3000 mm. Como nuestra zona es de 1800 mm, si hay algo, el sensor lo verá.

Esta información y otras sobre los bloques la puedes tener abriendo la ayuda y haciendo click en el bloque del que quieras información. Para abrir la ayuda pulsa en la interrogación que hay al lado del símbolo del monitor



Te propongo probar este código a ver qué tal



Hacemos un bucle infinito.

Se pone a girar

Espera hasta que vea un objeto

Se dirige hacia él (acabará empujándolo)

Cuando el objeto se caiga, o se le escape, terminará la iteración y empezará la siguiente...

Gira hasta que vea un objeto...

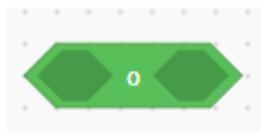
Crucemos los dedos a ver si consigue no caerse...

Hay veces que, cuando se cae el objeto, el robot comienza a girar para buscar otro y se "salva", pero en otras ocasiones está demasiado al borde.

Vamos a introducir un nuevo bloque.

Vamos a decirle al robot que deje de empujar cuando ya no vea objeto O cuando pise la línea roja.

Para eso necesitamos el siguiente bloque

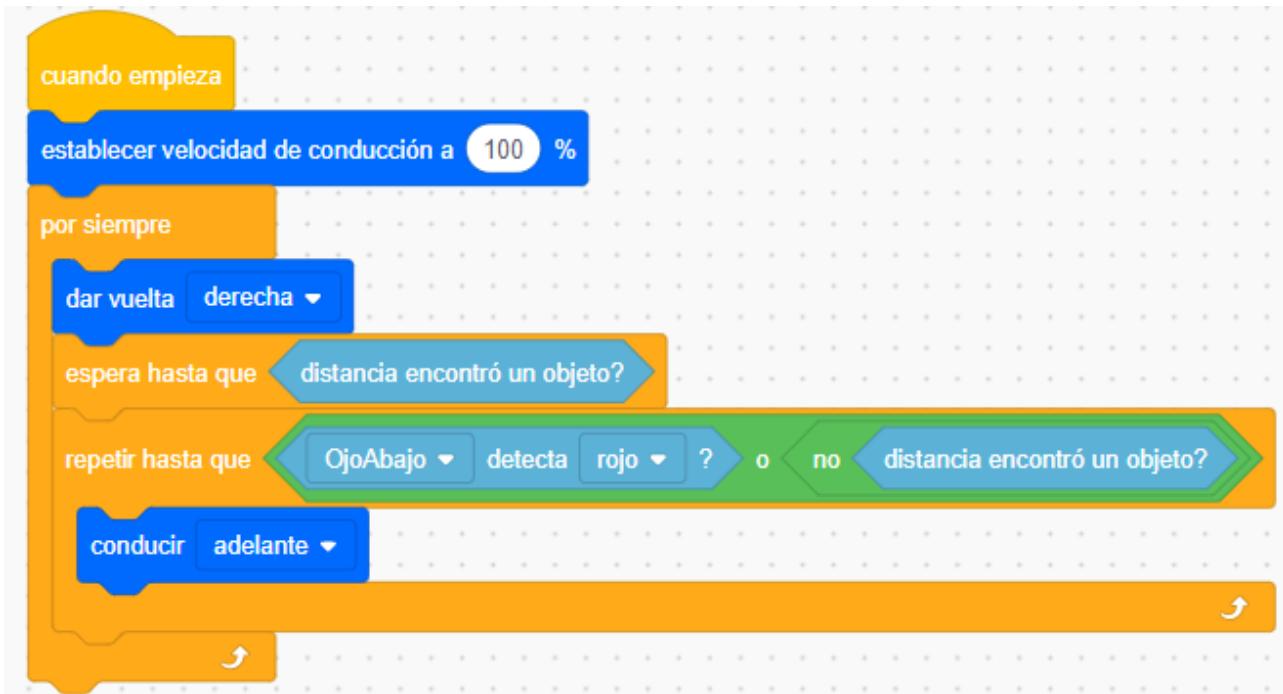


Y así nos queda la condición



Prueba con distintas velocidades y distintos escenarios (reinicia y cambiarán) a ver qué tal funciona.

El programa completo sería este



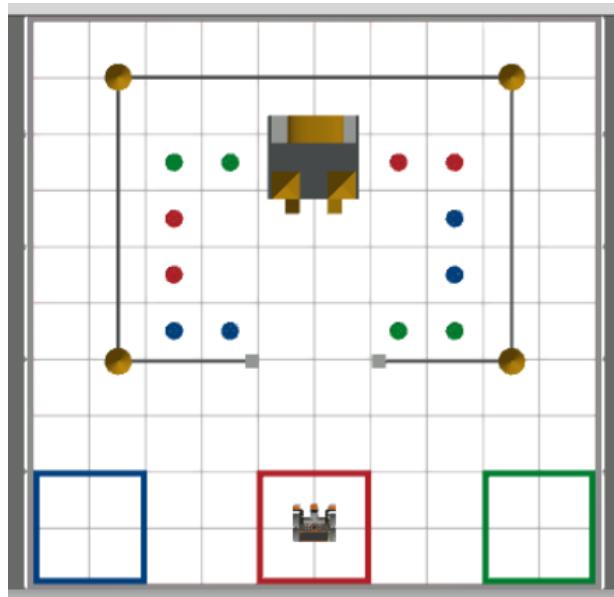
Como verás, a veces se quedan las piezas en el borde y por la “precaución” que gasta nuestro robot, o bien no las consigue tirar, o bien le cuesta mucho.

Objetivos conseguidos:

- Operadores O y NO.

¡Hagamos un robot aspiradora!

Empecemos por cambiar a la zona de juegos a “Transporte de discos”.



El recinto vallado será la habitación que queramos “limpiar”.

Los discos serán juguetes que nos hemos dejado tirados y queremos que el robot lleve a “su sitio”.

Los tres cuadrados exteriores, el lugar donde hay que guardar los discos, según su color.

Este proyecto hay que dividirlo en partes, para hacerlo más sencillo y comprenderlo mejor.

Habrá una parte que será “Buscar” (para moverse dentro del cuarto buscando discos)

Habrá otra parte que será “Guardar_discos” (para llevárselos cuando los encontramos)

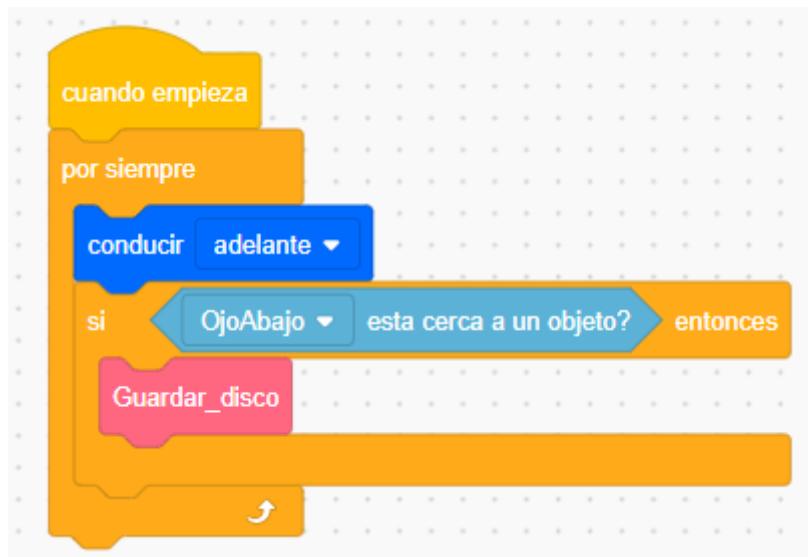
Habrá otra que será “Entrar_cuarto” (para volver a entrar a por más)

BUSCAR

Vamos a empezar por aquí.

Tenemos que movernos y si vemos un disco, activar la función “Guardar_discos”

La idea básica sería esta



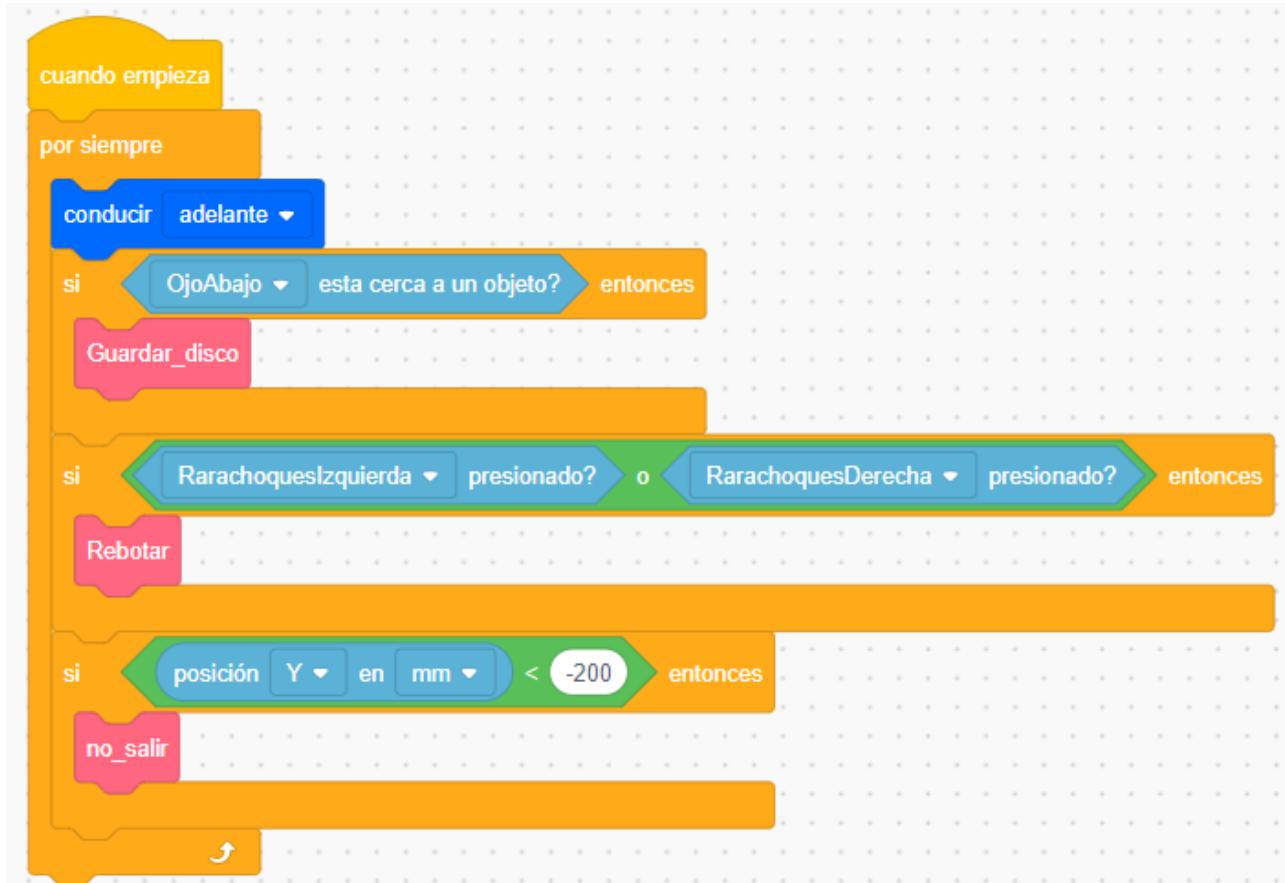
Aún no hemos definido la función **Guardar_discos**, estamos solo viendo cómo es el esquema general.

Pero este no es el único problema que puede tener el coche en su movimiento por el cuarto, ¿verdad?

Otros problemas:

- Chocarse con el objeto central o las paredes
- Salirse por la puerta

Así que necesitamos poner más condicionales.



La idea es:

“Tira p’alante y mira cómo te va...”

- Si ves un disco, recógelos y guárdalo
- Si el parachoques izquierdo o el derecho se activan, “rebota” para evitar el obstáculo
- Si tu coordenada Y es menor que la de la puerta, haz algo para no salir.
- Si no pasa nada de eso, vuelve a empezar el bucle y... sigue p’alante.

NOTA: En programación, los nombres de las variables y funciones no pueden tener espacios, así que cuando queremos separar letras es típico usar el guión bajo (está donde el guión normal, pero sale al usar las mayúsculas). También es usual que los nombres de las variables empiecen por minúscula.

NO_SALIR

Este es sencillo, si nuestro robot, en sus movimientos raros resulta que enfila hacia la puerta y se quiere salir, pongámoslo mirando hacia arriba y que ande un poco hacia adentro.



Básicamente lo ponemos mirando hacia arriba y que ande un poco hacia adentro (200 mm es el equivalente a un cuadro)

REBOTAR

Cuando choquemos contra un obstáculo, nuestra estrategia va a ser:

Andar un poco hacia atrás y cambiar la dirección del movimiento.



Ves que hemos usado un bloque nuevo que nos da números al azar entre dos valores que pongamos. Está en el grupo "Operadores".

De esta manera, cada vez que rebote cambiará a una dirección aleatoria y eso nos permitirá recorrer todo el "cuarto".

Dependiendo de la geometría del recinto y de los valores que pongáis para los ángulos, esta forma de recorrer el cuarto funcionará mejor o peor.

Ya podéis ejecutar (aunque aún no recoge discos) y ver cómo se mueve dentro del cuarto y cómo vuelve si sale por la puerta.

GUARDAR DISCO

Esto ya no es tan fácil que pueda hacerse con un par de bloques.

Está claro que tenemos que recoger el disco, así que estos bloques serán necesarios seguro.



Pero para ver dónde lo llevamos, tendremos que ver de qué color es. El "Ojo de Abajo" es capaz de ver el color del disco aunque lo tengamos cogido con el imán.

Vamos a usar este bloque



Miraremos primero si es rojo...

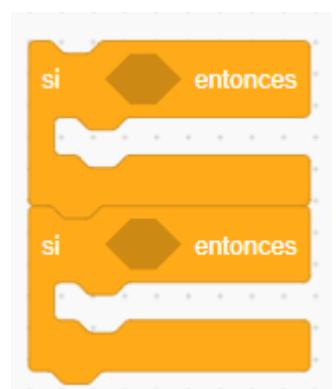
Si no lo es, entonces miraremos si es verde...

Si no lo es, entonces miraremos si es azul...

Como vamos a necesitar más huecos de los que trae, haremos click en el "triángulo con el más".

El último hueco sirve por si queremos hacer alguna acción si no se cumple ningún condicional. En nuestro caso no ocurre, miraremos todos los colores posibles, así que, alguno será el correcto.

NOTA: La diferencia entre usar el bloque anterior y esto que pongo a continuación...



es que, en este caso, aunque se cumpla el primer condicional el programa sigue mirando los siguientes condicionales.

En el que vamos a usar nosotros, sólo mira el siguiente si no se han cumplido los anteriores, por lo que es más eficiente.

Quedaría así:



Dentro de cada condicional habría que poner “Ir a casa roja”, “Ir a casa verde” o “Ir a casa azul”.

Como decíamos antes, el último caso (que no sea de ninguno de esos colores) no se va a dar por lo que queda vacío.

Vamos a concentrarnos ahora en la “vuelta a casa”. La verdad es que esos tres bloques “Ir a casa roja/verde/azul” iban a ser muy parecidos, salvo por la dirección del destino final.

Lo más “barato” es hacer un sólo bloque que tenga UN PARÁMETRO cuyo valor haga que vaya al destino rojo, al verde o al azul. Eso se consigue con “Añadir una entrada”.

Hacer un Bloque

Ir_casa

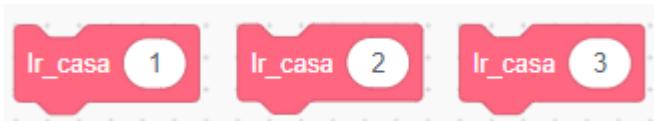
Añadir una Entrada número

Añadir un booleano

Añadir una etiqueta

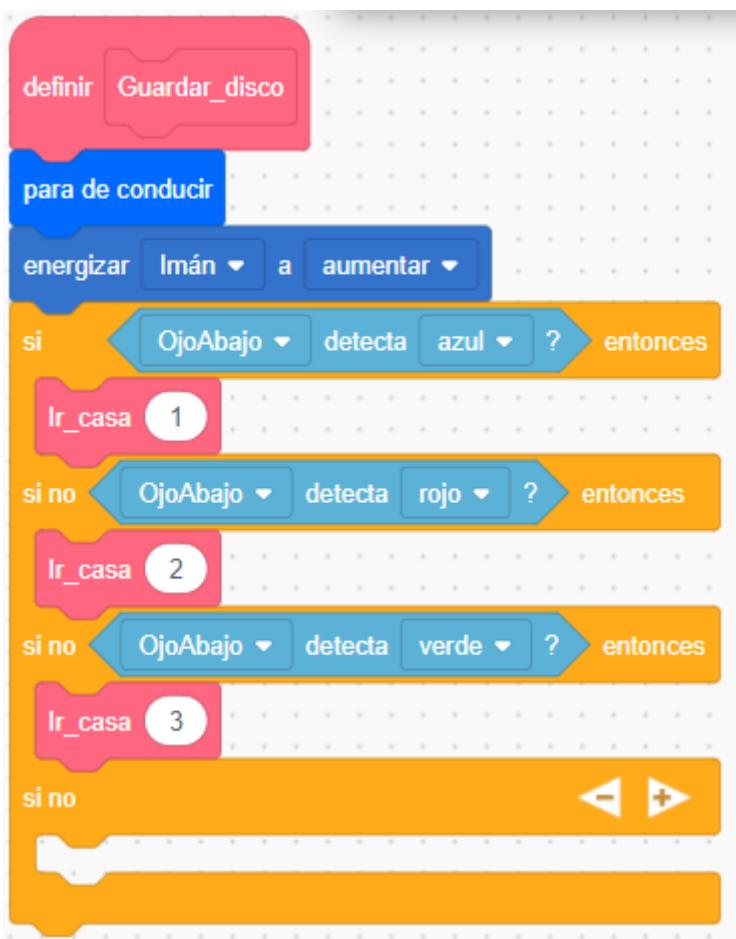
Cancelar OK

Esos bloques los usaremos así



Simbolizando los destinos azul, rojo y verde, respectivamente (en el orden que están en el simulador).

Tendríamos esto entonces



Ahora nos toca definir "lr_casa"

Como veis la cabecera para definir es un poco diferente



Haced click y arrastrad donde dice "number", veréis que sale un bloquecito pequeño que tendrá el valor del número con el que se llame a la función (en nuestro caso: 1, 2 o 3)

Lo primero es “enfilar” la puerta para poder salir.

Empecemos por “bajar” para evitar el obstáculo central.



Según estemos a la derecha o a la izquierda de la puerta, tendremos que movernos en una dirección diferente. Añadiremos estos bloques



NOTA: Quizá te sorprenda que busquemos una posición que sea mayor o menor que un número en lugar de buscar que fuera igual a un valor concreto. La cuestión es que cuando mires los sensores es casi imposible que justo te den cierto valor, se habrá pasado un poquito o estará un poquito corto.

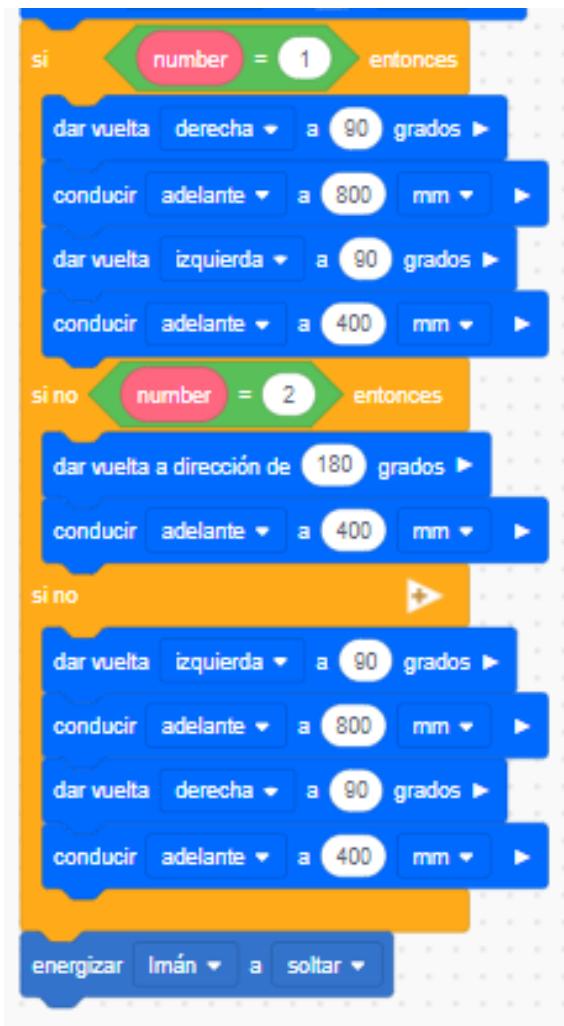
Si, por ejemplo, venimos al cero por el lado positivo, pues esperaremos a que el sensor nos dé un valor negativo pequeño y así sabremos que ya hemos llegado (aunque técnicamente nos hayamos pasado un pelín despreciable).

En este momento ya estamos aproximadamente en (0,0)

Primero habría que salir fuera del cuarto



Después habría que coger un rumbo distinto para llegar a cada destino de color y finalmente soltar el disco.



Comprueba que te coincidan los caminos con los que te pongo yo.

Fíjate que en el caso de la casilla roja solo hay que bajar un poco más.

Y al final, insisto, soltar el disco.

Nos faltaría hacer que entrara otra vez en la habitación a seguir buscando.

Para eso añadimos después de que suelte la pieza un bloque **volver**, que sería relativamente sencillo.



Se trata de ir a la posición $X = 0, Y = -400$ desde donde esté.

Podemos saber cuánto tenemos que andar restando las coordenadas del destino de las del origen.

Si sale negativo, el robot irá hacia atrás. Fíjate cuando viene de la casilla verde, p.ej.

Como has visto, una tarea que podría parecer imposible hacerla de una sola vez, resulta mucho más sencilla si se divide en tareas simples.

Insisto que el principio de “*Divide y vencerás*” es de aplicación general en muchos aspectos de la vida laboral y personal.

Objetivos cumplidos:

- **Atomizar una tarea compleja.**
- **Números aleatorios**
- **Condicionales anidados**
- **Bloques con parámetros**

NOTA para usuarios más avanzados:

1. Por el carácter básico del manual estamos moviendo el robot primero en un eje y luego en el otro, pero tenéis en el grupo “Operadores” la función arcotangente que, junto con Pitágoras podéis usar para moveros directamente de un punto a otro en un solo trayecto (sin obstáculos)

Listas y cronómetro

Vamos a ver dos funciones muy interesantes que aún no hemos tocado: el cronómetro y las listas.

El cronómetro nos permite:

- Saber cuándo ocurre algo.
- Tomar decisiones cuando el tiempo tome cierto valor

Sus bloques son

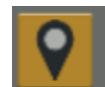


El primer bloque pone el cronómetro a cero. Puede hacerse dentro del programa tanto como se quiera.

El segundo bloque nos da el valor del cronómetro en el momento en el que se usa.

Escojamos la zona “Detector de línea”.

Si hacéis click en este botón que os aparece abajo a la izquierda el lugar de salida del robot. Escoged E.



podéis escoger

Nuestro objetivo es apuntar en qué instante se cruza cada línea.

Hablemos de las LISTAS.

¿Recordáis que cuando queríamos guardar un valor, usábamos una variable? En este caso, como vamos a guardar varios números necesitaríamos tener varias variables, pero es más sencillo guardarlos todos en una lista.

Las variables y las listas no son las únicas maneras de guardar datos. En este programa también se pueden crear listas bidimensionales (matrices), pero hay aún más. Simplemente recuerda para el futuro que usamos la que más conviene a la forma que tienen los datos que queremos guardar.

Para crear una lista, nos vamos al grupo “Variables” y allí tienes “Hacer una lista”.

Te preguntará la longitud de la lista (selecciona ocho, sabemos que solo hay ocho líneas)

Aparecen bloques nuevos

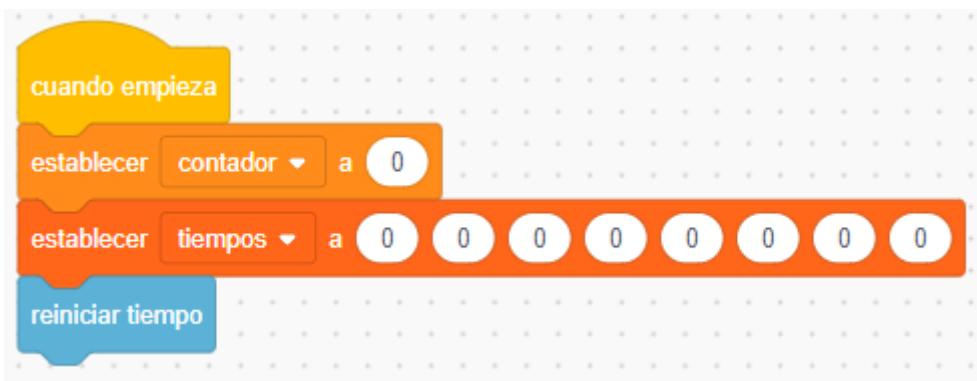


El primer bloque te permite usar cada valor de la lista, según su número de orden. En el ejemplo, el número uno.

El segundo bloque te permite dar valor a un elemento concreto de la lista. En el ejemplo, dará el valor cero al elemento número uno de la lista.

El tercer bloque te permite dar valor a la vez a todos los elementos de la lista.

Empezamos nuestro programa, como tantas veces, inicializando todo. Es una manera de asegurarse de que no tendrán valores pasados que nos estropeen el resultado.



Conduciremos hasta chocar



Pero dentro de ese bucle iremos comprobando si vemos líneas en el suelo o no con el siguiente condicional



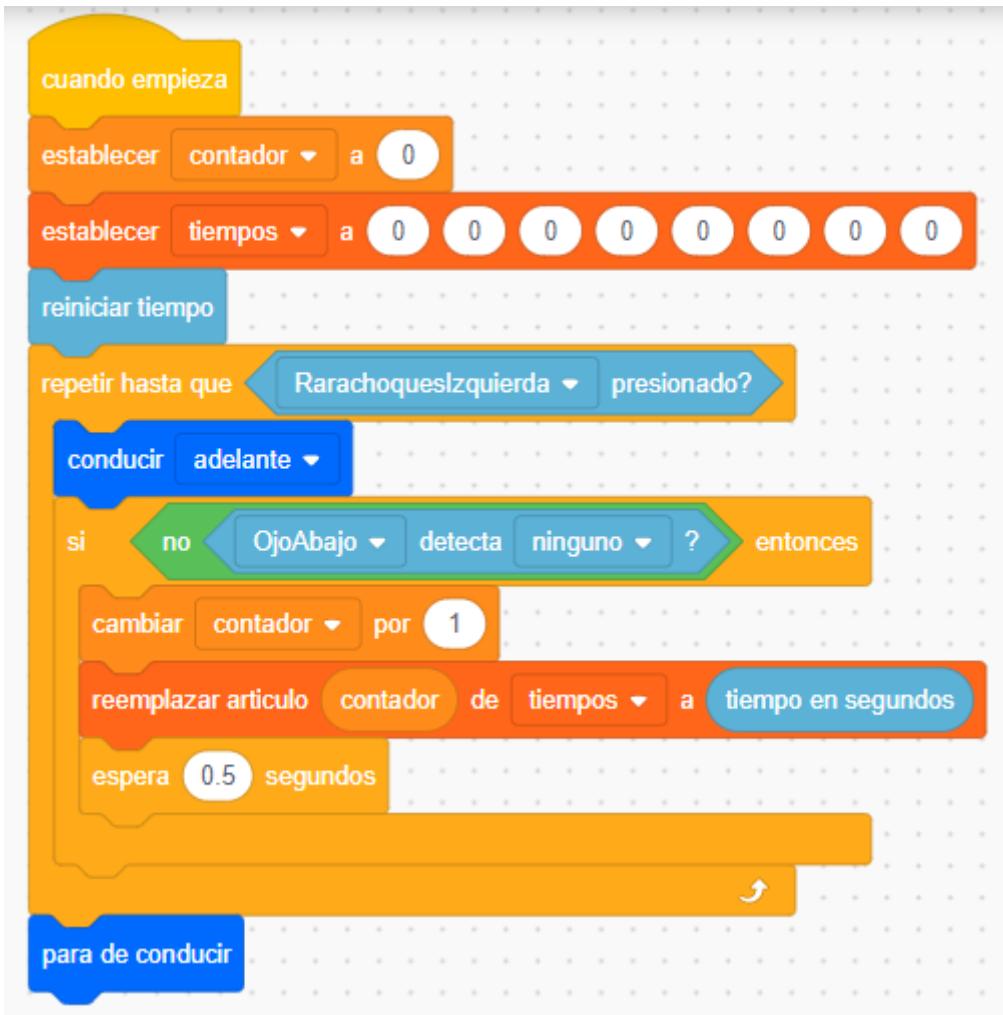
Si no detecta ninguno... ¡es que detecta ALGUNO! Con este truco puedes detectar todas las líneas del color que sean.

Si hemos encontrado algo, aumentamos el contador en una unidad.

Reemplazamos el elemento de la lista número “lo que marque el contador” por el valor del tiempo que marca el cronómetro.

Si quieras saber por qué he puesto la espera de medio segundo. Quítala y prueba ;)

Este es el programa completo



Saca el monitor y agrega la lista y marca el contador para que aparezcan allí y así ver cómo sucede todo.

Sensores	
heading	0°
rotation	0°
Front Eye	Object: False Color: None
Down Eye	Object: False Color: None
Location	X: 803 mm Y: 123 mm
Location Angle	0°
Bumper	Left: False Right: False
Distance	822 mm

Variables	
contador	5
tiempos	{0.368, 0.897, 2.208, 2.817, 3.473, 0, 0, 0}

Si queréis podéis añadir estos bloques al final para que os dé un mensaje el robot y así ver cómo podéis acceder al valor de un elemento de la lista



Objetivos cumplidos:

- **Uso del cronómetro. Reinicio y uso del valor**
- **Uso de las listas. Creación, cambio de valores y uso de valores.**

Despedida

Espero que os haya gustado y servido para aprender.

Como siempre no trato de agotar todas las posibilidades del programa. Mi intención es usarlo para ir introduciendo los conceptos claves de la programación.

Se agradece difusión y apoyo.

[Podéis invitarme a un ko-fi por aquí](#)

Juntos somos más.