ARDUINO
VERKSTAD
EDUCATION

CLASE DE TECNOLOGÍAS CREATIVAS

CASTILLA LA MANCHA

Co-financiado por:

Arduino Verkstad 2014

Tabla de contenidos

CREA
PROGRAMA
APRENDE
EXPERIMENTA

Introducción

La clase de tecnologías creativas en el aula (CTC) es una serie de metodologías y experimentos dirigidos a transformar la forma en que la tecnología se enseña en colegios secundarios de todo el mundo.

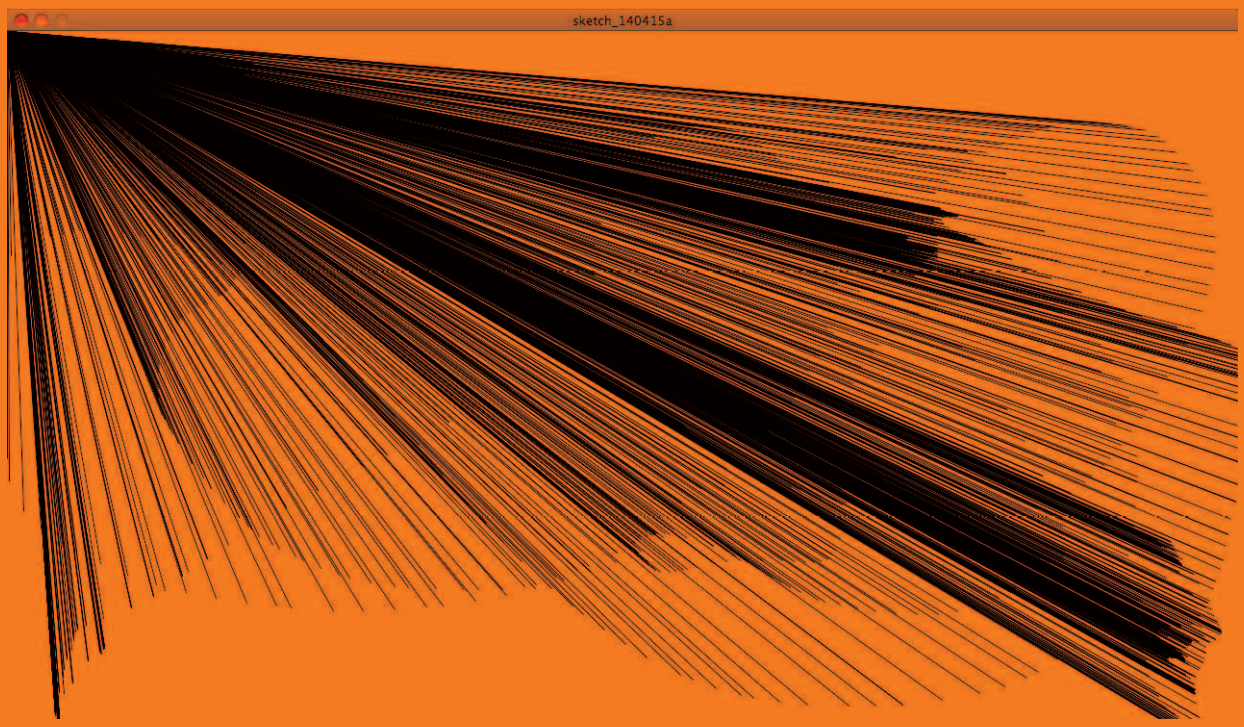
Estos experimentos introducen conceptos básicos acerca de la programación, la electrónica y la mecánica de forma lúdica. El proyecto desafía las formas tradicionales de enseñanza y motiva a los profesores y estudiantes a utilizar Internet como la principal herramienta para obtener información y documentar los proyectos que los alumnos construyen.

CTC está compuesta de un kit con más de 20 ejercicios explicados a detalle, además de un sitio web donde se puede realizar y documentar el proceso de ejecución del curso, y un grupo de colaboración en línea, donde los maestros que realizan la misma clase se reúnen, junto con un mediador, para compartir sus hallazgos y hacer preguntas técnicas.

Este proyecto se ha realizado en las regiones de Castilla La Mancha y Madrid, y durante el 2014 llegará a Cataluña, Ecuador y Skåne. Por lo general, fundaciones e instituciones públicas otorgan apoyo para el desarrollo del proyecto CTC. Hasta el momento, la Fundación Telefónica, la Comunidad de Castilla La Mancha, y la Fundación La Caixa, han patrocinado un total de 125 escuelas (más de 3.500 niños) para participar en este proceso.

Estamos planeando lanzar OpenCTC para permitir que cualquier escuela en todo el mundo pueda unirse a nosotros en el proceso de aprendizaje conjunto sobre los usos creativos de la tecnología.

Arduino Verkstad Malmö


sketch_140415a

SEMANA 1
Código

CONCEPTOS

PROYECTOS

Esta primera semana vas a hacer una serpiente roja interactiva, un reloj con post-it y un juego inspirado en Newton. Aprenderás lo básico de programación así como algo de la teoría detrás de los gráficos en las pantallas. Esta segunda parte es importante debido a que, aunque este curso trate sobre programación, el objetivo es siempre crear algo útil y divertido. Para hacer uso de la pantalla en tus proyectos, es bueno saber cómo funciona. Cuando esta semana esté terminada, estaremos encantados de ver qué otros juegos o proyectos interactivos serás capaz de crear.

Processing

Los códigos de programación son comandos comprensibles para los humanos. El código se transforma en los programas que los computadores entienden. Los computadores sólo entienden en realidad ceros y unos, por lo que se necesita una manera de traducir nuestro código humano al código de los computadores. Este es el motivo por el que escribimos el código en un editor de texto con propiedades especiales. Este editor es el software de desarrollo llamado Entorno de Desarrollo Integrado, abreviado por sus siglas del inglés IDE (Integrated Development Environment), el cual es capaz de traducir el código al lenguaje máquina. Cuando el IDE transforma nuestro código al lenguaje máquina, es lo que se llama compilar.
Cada ordenador, smartphone, servidor de internet, ascensor, horno – actualmente cualquier dispositivo con inteligencia digital – tiene diferentes capacidades y por lo tanto requiere de un lenguaje máquina algo diferente al resto. Existe un lenguaje de programación llamado Java que, una vez compilado, es capaz de utilizarse fácilmente en diferentes dispositivos. Hay muchas maneras diferentes de programar en Java IDE, pero nosotros nos centraremos en utilizar Processing.

Processing es código abierto y software libre, esto significa que se puede descargar de Internet, instalarlo sin coste alguno, y modificar su código. Processing funciona en cualquier tipo de ordenador de sobremesa o portátil en los sistemas operativos: Windows, Mac OSX y Linux.

El IDE de Processing es muy sencillo, hay una barra de herramientas que puedes utilizar para:


Poner en marcha un programa

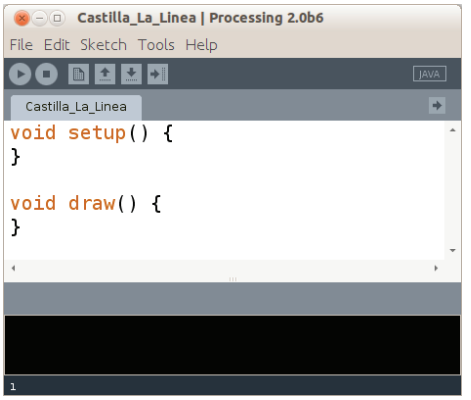

Detener un programa


Crear un nuevo programa


Abrir un programa


Almacenar el programa en el disco duro del ordenador


Exportar el programa



Processing crea programas que se ejecutan directamente dentro de tu ordenador. También puedes exportarlos y enviarlos a los ordenadores de otras personas. Los programas, una vez exportados en códizgo máquina, se llaman aplicaciones. Puedes exportar el mismo programa para que se ejecute en un ordenador, en un teléfono, o en un sitio web.

Para hacer tus propios programas, lo único que necesitas saber es cómo utilizar un teclado y cómo funciona la pantalla del ordenador. Esto te será útil cuando escribas tus propios programas interactivos.

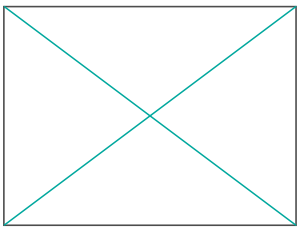
Pantallas y píxels

Las pantallas se utilizan para representar imágenes. Esas imágenes pueden ser dinámicas, como en las películas, o estáticas, como en las fotografías. Se puede incluir texto, como en los documentos, o solo gráficos. La mayoría de las veces, las imágenes en la pantalla son una mezcla de muchas cosas; incluyen animaciones, textos, estadísticas, etc.

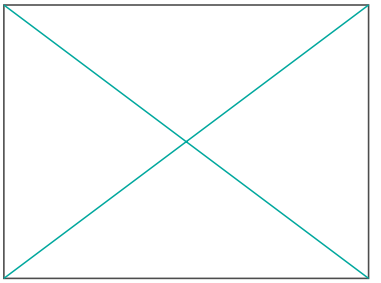
Pixels – Resolución

Las pantallas están hechas de puntos que llamamos pixels. Por lo general, cuantos más pixels tenga una pantalla, mejor será. Expresamos la calidad de la pantalla en función de su cantidad de pixels, esa cantidad es lo que llamamos resolución. Normalmente, hablamos de una pantalla en función de su anchura y altura. Por ejemplo, muchos proyectores tienen una resolución de 1024×768 pixels

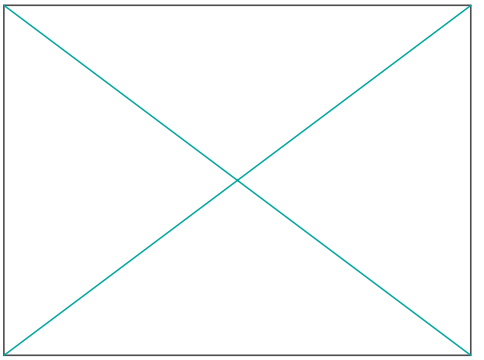
Esto significa que tiene 1024 pixels de ancho y 768 pixels de alto.



640 x 480



800 x 600



1024 x 768

Colores

Cada píxel se puede iluminar de muchos colores diferentes. Una vez más, de cuantos más colores se pueda pintar el píxel, mejor será la pantalla. Los colores en la pantalla del ordenador se expresan como una combinación de tres componentes de color diferentes: rojo (R), verde (G) y azul (B). Cualquier color que representes en una pantalla, se puede hacer mezclando los componentes RGB en diferentes cantidades.

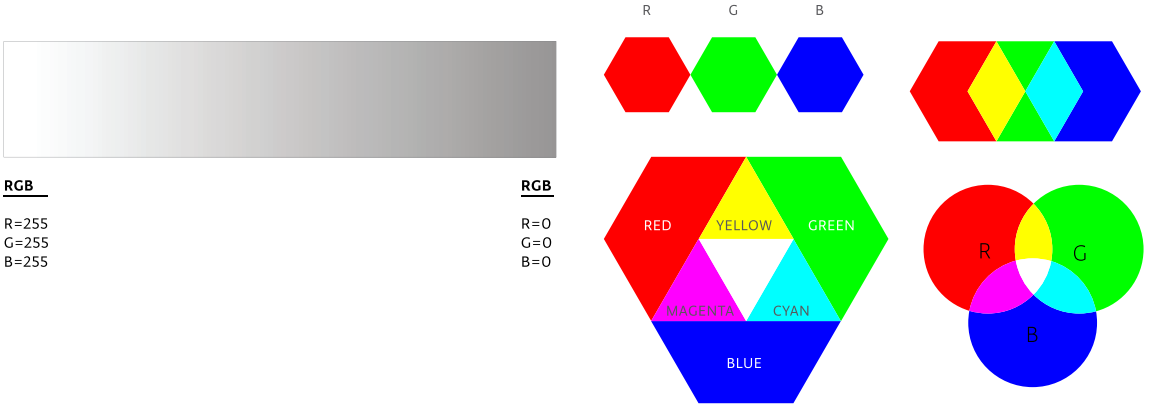
La cantidad de color para cada componente se expresa con un número entre 0 y 255. 0 representa el valor mínimo de un color, por ejemplo, si tiene 0 del componente rojo, significa que ese color no contiene rojo, sin embargo, si tiene 255 del componente rojo, significa que está saturado de color rojo.

Si las coordenadas son 255,255,255 – los píxels producirán blanco. Si los valores R,G,B son 0,0,0 – los píxels mostrarán negro. Si gradualmente cambias los valores R,G,B en la misma proporción desde 255,255,255 a 0,0,0, verás que el color cambia de blanco a negro.

La siguiente imagen muestra cómo el color de un píxel se funde de blanco a negro y cómo cambian los valores de R, G, y B:

Coordenadas

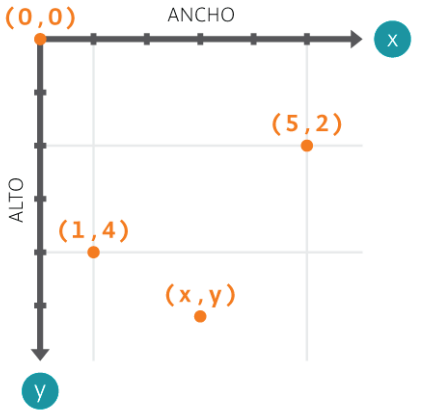
Puedes escribir programas que cambien los pixels de la pantalla. Puedes incluso cambiar un solo pixel. La acción de acceso a un solo pixel en la pantalla es lo que llamamos direccionar un pixel. La ubicación de un pixel se determina mediante coordenadas.



Si recuerdas lo que aprendiste en clase de matemáticas y física, puedes dibujar gráficos, puntos, líneas y curvas usando sus coordenadas. Esta misma idea se aplica a la pantalla del ordenador, la única diferencia, es que el llamado origen de coordenadas de la pantalla del ordenador es la esquina superior izquierda.

Cada pixel tiene 2 coordenadas:

La coordenada horizontal es la que llamamos coordenada X (anchura, o en inglés width).
La coordenada vertical, es la que llamamos coordenada Y (altura, o en inglés height).



Línea

La mejor manera de aprender acerca de la programación es escribiendo tus propios programas. Vamos a empezar por hacer el programa lo más corto posible usando Processing. Será un programa para dibujar una línea. Esto te va a ayudar a entender cómo funciona.

Abre el IDE y haz click en el icono  para crear un nuevo programa, esto te mostrará un nuevo programa vacío. Escribe el siguiente código:

```
1   line(0, 0, 100, 100);
```

Ejecuta el programa apretando el botón ***ejecutar***:  , que es el primero en la barra de herramientas. Esto es lo que deberías ver ahora por pantalla:

Sólo utilizamos un único comando aquí:

`line(x1, y1, x2, y2):` Dibuja una línea desde la coordenada x1,y1 hasta la x2,y2. En este ejemplo, desde 0,0 – esquina superior izquierda, hasta 100,100 – esquina inferior derecha. Esto es debido a que la ventana es 100 por 100 píxels.

Lo que también necesitas saber es:

- Debes terminar cada línea de código con un puto y coma, '`;`'. Si te olvidas, el código no podrá ser compilado ni ejecutado.
- Por defecto, las ventanas de Processing son de 100×100 píxels con un fondo gris. El color por defecto para dibujar es el negro. Pero por supuesto, todo esto se puede cambiar.

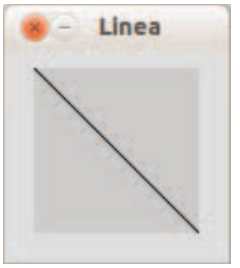
Vamos a escribir un programa que dibuje dos líneas de diferentes colores sobre un fondo blanco:

```
1  background(255);
2  line(0, 0, 100, 100);
3  stroke(0, 0, 255);
4  line(0, 100, 100, 0);
```

Los nuevos comandos utilizados aquí son:

- `background(gray)`: Establece el color de fondo desde 0 – negro, hasta 255 – blanco. También puedes utilizar `background(red, green, blue)` para establecer el color que tú quieras.
- `stroke(red, green, blue):` Establece el color de la línea. Cada valor de color puede ser desde 0 hasta 255. En este caso, la línea es de color azul puesto que red=0, green=0 y blue=255.

Cuando ejecutes el programa, lo que deberías ver por pantalla es:



Sólo utilizamos un único comando aquí:

- `line(x1, y1, x2, y2)`: Dibuja una línea desde la coordenada x1,y1 hasta la x2,y2. En este ejemplo, desde 0,0 – esquina superior izquierda, hasta 100,100 – esquina inferior derecha. Esto es debido a que la ventana es 100 por 100 píxels.

Lo que también necesitas saber es:

- Debes terminar cada línea de código con un puto y coma, ';'. Si te olvidas, el código no podrá ser compilado ni ejecutado.
- Por defecto, las ventanas de Processing son de 100×100 píxels con un fondo gris. El color por defecto para dibujar es el negro. Pero por supuesto, todo esto se puede cambiar.

Vamos a escribir un programa que dibuje dos líneas de diferentes colores sobre un fondo blanco:

Vamos a escribir un programa que dibuje dos líneas de diferentes colores sobre un fondo blanco:

```
1  background(255);
2  line(0, 0, 100, 100);
3  stroke(0, 0, 255);
4  line(0, 100, 100, 0);
```


Los nuevos comandos utilizados aquí son:

- `background(gray)`: Establece el color de fondo desde 0 – negro, hasta 255 – blanco. También puedes utilizar background(red, green, blue)para establecer el color que tú quieras.
- `stroke(red, green, blue)`: Establece el color de la línea. Cada valor de color puede ser desde 0 hasta 255. En este caso, la línea es de color azul puesto que red=0, green=0 y blue=255.

Cuando ejecutes el programa, lo que deberías ver por pantalla es:

Tus programas se ejecutarán siempre de arriba a abajo línea por línea. Este programa empieza estableciendo el color de fondo en blanco. Luego dibuja la línea negra igual que en el ejemplo anterior, desde la coordenada 0,0 hasta la 100,100. En tercer lugar, establece el color de línea a azul, justo antes de dibujar esta segunda línea desde la coordenada 0,100 – esquina inferior izquierda, hasta la 100,0 – esquina superior derecha.

No olvides guardar tu programa en el disco duro o en una memoria USB si quieres llevártelo a casa.

¡Sigue experimentando!

- Añade más líneas de diferentes colores y cambia el color de fondo.
- Cambia el orden del programa. ¿Qué ocurre si, por ejemplo, cambias el color de fondo al final?

Variables

Las variables son algo que usas todo el tiempo en programación. Son como un contenedor para diferentes tipos de datos. Para cada variable, necesitas especificar qué tipo de datos contendrán, cuál es el nombre de la variable y qué valor se le asigna.

Piensa en ellas como en botes. Digamos que tienes dos botes, uno para galletas y otro para palabras, estos son los tipos de datos. Ahora hay que darles un nombre a cada bote; cookieJar ***(bote de galletas)*** y jarOfWord ***(bote de palabras)***. Ahora tú decides que poner en cada bote. En el cookieJar pones una galleta de doble cocholate y en el jarOfWord decides poner "Arduino". Ahora cada bote tiene un valor.

Puedes cambiar el contenido de los botes, su valor, en cualquier momento, pero siempre y cuando sea del mismo tipo. Por ejemplo, puedes cambiar la galleta de doble chocolate por una oreo y "Arduino" por "spaceinvader".

Para hacerlo más claro, vamos a escribir un programa utilizando variables. Escribiremos un programa que dibuje, de nuevo, dos líneas, pero esta vez utilizaremos estas variables:

```
1   int value1 = 0;
2   int value2 = 100;
3   line(value1, value1, value2, value2);
4   line(value1, value2, value2, value1);
```

Nuevos comandos:

- `int variableName = value`: Crea una variable del tipo integer, un número entero. Puedes llamarla con cualquier nombre de tu elección, pero para hacerlo más sencillo para el futuro (probablemente tú o alguien más leerán el código y querrán entender lo que se hizo en el pasado) asegúrate que el nombre va acorde con el contexto en el que se usa la variable. value es el valor que decides darle al contenido de tu variable.

Lo que hace el programa es dibujar una línea desde la coordenada (0,0) hasta la (100,100) y luego dibuja una segunda línea desde la coordenada (0,100) hasta la (100,0). Puesto que `value1` contiene el valor 0, en cualquier lugar del código donde escribamos `value1` significará 0. Esto significa que si quieres cambiar la cruz que hemos dibujado, basta con cambiar el valor de la variable en lugar de cambiar el valor en los ocho lugares diferentes dentro de `line()`. Pruébalo tu mismo y verás.

Tipos de datos

Los tipos de datos más comunes que utilizarás son los siguientes:

- `int`: Número entero, p.ej., 2, 99 o 532.
- `float`: Número decimal, p.ej., 2.76, 8.211 o 900.3.
- `boolean`: Puede ser verdadero o falso.
- `char`: Un caracter, p.ej. 'r', '2' o '%'.
- `String`: Una secuencia de carateres, p.ej. "hola", "¡Me encanta programar!" o "&%!@¤".

Processing incluye algunas variables de sistema para hacerlas más accesibles dentro de tus programas. Un ejemplo es `width` y `height`. Estas variables devuelven la anchura (width) y altura (height) de la ventana de tu programa. Éste es un pequeño ejemplo:

```
1  size(400, 200);
2  ellipse(width/2, height/2, width, height);
```

Comandos nuevos:

- `size(width, height)`: Establece el tamaño de la ventana en píxels.
- `ellipse(x, y, x diameter, y diameter)`: dibuja un elipse centrado en las coordenadas `x` e `y`. El tamaño se establece con `x diameter` e `y diameter`. Cuando estos dos parámetros son iguales, el resultado es un círculo.
- `width`: la anchura de la ventana de programa.
- `height`: la altura de la ventana de programa.

Este programa empieza estableciendo el tamaño de ventana. Luego dibuja la elipse en el medio de la ventana. No importa el tamaño de la venta, el elipse la llenará siempre por completo. Intenta cambiar el tamaño de la ventana.

Setup y draw

Los tipos de programas que hemos hecho hasta ahora son los llamados programas estáticos. Esto significa que nunca cambian. Se ejecutan una única vez y cuando llegan a la última línea de código, se paran. Si queremos que un programa sea interactivo, tenemos que habilitar la entrada de datos continuamente mientras el programa se ejecuta. Esto sólo es posible si la ejecución es continua.

Con Processing, puedes crear programas que se ejecuten continuamente utilizando la función `draw()`. Esta función repetirá el bloque de código una y otra vez hasta que el programa se pare. Sin embargo, no todo el código escrito necesita ser repetido continuamente. Para el código que sólo necesita ser ejecutado una única vez, debes usar la función llamada `setup()`.

Vamos a escribir de nuevo un programa que dibuje una línea, pero esta vez utilizando las nuevas funciones:

```
1  void setup() {
2  size(300, 300);
3  }
4
5  void draw() {
6  line(0 ,0, width, height);
7  }
```

Nuevos comandos:

`setup()`: El código dentro de las llaves se ejecuta una única vez cuando el programa empieza. Sólo necesitamos establecer el tamaño de la ventana una vez, al principio del programa.

`draw()`: El código entre llaves se ejecuta una y otra vez. Se ejecuta línea por línea, de arriba a abajo hasta la última línea, donde vuelve a empezar desde el principio.

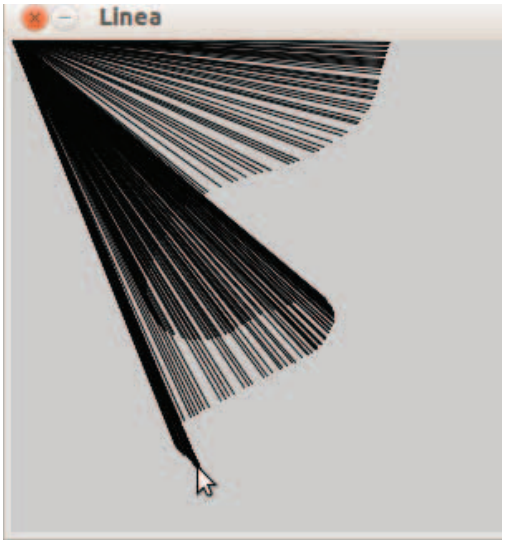
Este programa establece el tamaño de ventana a 300 x 300 pixels y luego dibuja una línea sobre toda la ventana una y otra vez. En este momento, todavía no se puede apreciar que esto esta pasando una y otra vez, así que vamos a escribir otro programa para hacerlo más visible.

```
1  void setup() {
2  size(300,300);
3  }
4
5  void draw() {
6  line(0, 0, mouseX, mouseY);
7  }
```

Nuevos comandos:

- `mouseX`: La coordenada X del puntero del ratón.
- `mouseY`: La coordenada Y del puntero del ratón.

Este programa te permite interactuar con la pantalla mientras mueves el ratón, como muestra la siguiente imagen. Esto es porque cada vez que el código dentro de `draw()` se ejecuta, una nueva línea se dibuja. El segundo punto de la línea cambia dependiendo de la posición del puntero del ratón.



Como puedes ver, se deja un rastro de líneas por donde vas moviendo el ratón. Esto es porque cada línea dibujada no es eliminada ni se dibuja nada más por encima. ¿Qué crees que ocurrirá cuando ejecutes el siguiente programa?

```
1  void setup() {
2  size(300,300);
3  }
4
5  void draw() {
6  background(255);
7  line(0, 0, mouseX, mouseY);
8  }
```

Hemos añadido una línea de código al principio de `draw()`, esto sirve para establecer el color de fondo a blanco. Como puedes ver, este programa solo muestra una línea todo el tiempo. Esto se debe a que, como hemos explicado antes, el código dentro de `draw()` se ejecuta una y otra vez, línea por línea de arriba a abajo. Por tanto, la primera cosa que pasa cada vez que `draw()` se ejecuta es que se crea el fondo blanco, sobre escribiendo todo lo anterior. Después una nueva línea es dibujada, y luego el fondo se vuelve a poner en blanco... Eso se repetirá una y otra vez por siempre.

Serpiente roja

En este miniproyecto vas a aprender a programar un objeto que se mueve en la pantalla, pero que además deja el trazado del movimiento del ratón. Iremos escribiendo el código paso por paso, añadiendo nuevas funcionalidades en cada uno.

Dibuja un círculo

Empezaremos dibujando un círculo rojo.

```
1  /*
2   * Red Snake (Serpiente Roja)
3   *
4   * Programa para crear una serpiente roja moviendose por la pantalla
5   *
6   * Paso 1:
7   *  - crea un programa en una ventana de 400x400 pixels
8   *  - dibuja un circulo rojo de 30 pixels de diametro
9   *  - haz que el dibujo no tenga borde, usa el comando 'noStroke()'
10  *
11  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
12  */
13
14 void setup() {
15   size(400, 400);
16 }
17
18 void draw() {
19   noStroke();                // Dibuja formas sin 'borde'
20   fill(255, 0, 0);           // Haz que las figuras sean de color rojo
21   ellipse(100, 100, 30, 30); // Circulo de 30 pixels de diametro
}
```

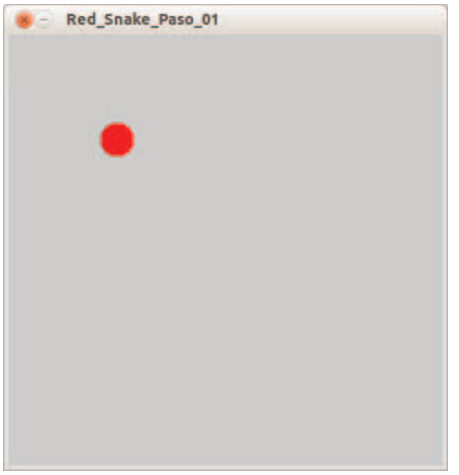
Nuevos comandos:

- `noStroke()`: Este comando es usado para no dibujar el contorno de las siguientes figuras
- `fill(red, green, blue)`: Establece el color utilizado para rellenar las siguientes figuras. En este ejemplo rellenará el círculo de color rojo.

Veamos los otros comandos utilizados:

- `size(ancho, alto)`: establece el tamaño de la ventana del programa en píxels.
- `ellipse(x, y, diametro x, diametro y)`: dibuja un elipse con centro en x,y. El tamaño se establece con diametro x y diametro y. Cuando estos dos parámetros son iguales, el resultado es un círculo.

Lo que el programa hace es primero establecer el tamaño de la ventana a 400 x 400 píxels. Nosotros decidimos no utilizar contornos y utilizar el color rojo para el relleno. Por último, dibuja un círculo en las coordenadas 100,100 con un diámetro de 30 píxels.

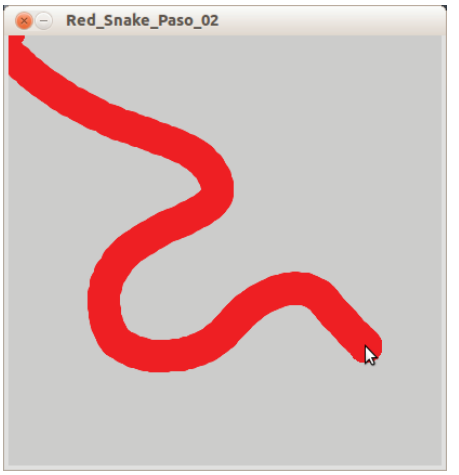


Haz que el círculo se mueva

En este paso haremos que el círculo se mueva con el ratón, dejando un rastro en la pantalla.

```
/*
 * Red Snake (Serpiente Roja)
 *
 * Programa para crear una serpiente roja moviendose por la pantalla
 *
 * Paso 2:
 *  - haz que el circulo se mueva en funcion de las coordenadas del
 *    raton (usa 'mouseX' y 'mouseY')
 *
 * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
 */

void setup() {
  size(400, 400);
}

void draw() {
  noStroke();
  fill(255, 0, 0);
  ellipse(mouseX, mouseY, 30, 30); // Circulo de en funcion de las coordenadas del raton
}
```

Todo lo que hemos hecho aquí ha sido reemplazar las coordenadas de la elipse con mouseX y mouseY. Esto hace que el círculo siempre siga al puntero del ratón. Al no redibujar el fondo de la pantalla, hacemos que el círculo deje su trazo.

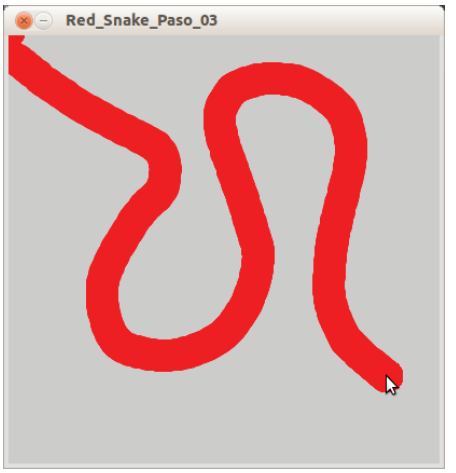


Cambia gradualmente el color

Queremos ahora que el color cambie durante el programa, y para hacerlo es más conveniente utilizar una variable para el color en lugar de un número constante. Simplemente declara la variable rojo con la que modificar el valor del color.

```
1  /*
2   * Red Snake (Serpiente Roja)
3   *
4   * Programa para crear una serpiente roja moviendose por la pantalla
5   *
6   * Paso 3:
7   *  - almacena la intesidad de rojo en una variable
8   *
9   * (veras como este programa no cambia en nada frente al Paso 2)
10  *
11  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
12  */
13
14 int rojo = 255;
15
16 void setup() {
17   size(400, 400);
18 }
19
20 void draw() {
21   noStroke();
22   fill(rojo, 0, 0);
23   ellipse(mouseX, mouseY, 30, 30); // Circulo de en funcion de las coordenadas del raton
24 }
```

Fíjate que esto no causará ningún cambio en la aplicación, como puedes ver en la siguiente imagen.



Cambia gradualmente el color II

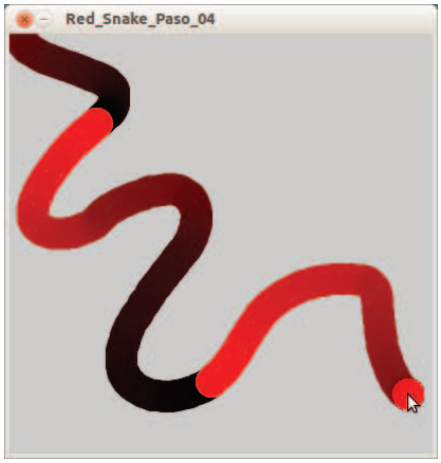
Ahora vamos a cambiar el color. El cambio de color dependiendo del tiempo puede hacerse de diferentes maneras. El color puede cambiar entre 0 y 255 (negro y rojo respectivamente). El siguiente ejemplo muestra como reducir la cantidad de rojo en cada instante que se ejecuta draw().

```
1  /*
2   * Red Snake
3   *
4   * Program to create a red snake moving through the screen
5   *
6   * Step 4:
7   *  - chage the level or red with the time
8   *
9   * (c) 2013 D. Cuartielles, Arduino Verkstad, Sweden
10  */
11
12 int rojo = 255;
13
14 void setup() {
15   size(400, 400);
16 }
17
18 void draw() {
19   rojo = rojo - 1;            // Make the red a little darker
20   if (rojo < 0) rojo = 255;   // Once it reaches black, make it light red again
21
22   noStroke();
23   fill(rojo, 0, 0);
24   ellipse(mouseX, mouseY, 30, 30);
25 }
```

Nuevos comandos:

- `if( test ){ statements }`: Comprueba si `test` es cierto. En este ejemplo, si rojo es menor que 0. Si es así, los `statements` entre llaves son ejecutados. En este ejemplo, establece `red` a 255 de nuevo. Si por el contrario, `test` es falso, el programa procede a ejecutar el código después de las llaves. Utilizando declaraciones "if" te permite decir al programa que código ejecutar.

Al principio de este programa `red` es 255, pero cada vez que `draw()` se ejecuta, `red` se reduce en una unidad. Esto hace que el color del círculo se haga más y más negro en cada iteración. Cuando `red` ha descendido mucho, tal que es menor que 0, se reinicia a 255. Este es el motivo por el que el color cambia de repente de negro a rojo.



Usa la función sin()

Para cambiar los colores más gradualmente, utilizaremos una función sinusoidal que oscile continuamente entre negro y rojo.

```
1  /*
2   * Red Snake (Serpiente Roja)
3   *
4   * Programa para crear una serpiente roja moviendose por la pantalla
5   *
6   * Paso 5:
7   *  - cambia la intensidad del rojo en funcion del tiempo
8   *  - para un mejor efecto visual y evitar que el rojo cambie
9   *    de forma brusca, utiliza una funcion matematica (por ejemplo
10  *    el seno 'sin()')
11  *  - necesitaras una variable para contar el paso del tiempo
12  *  - el '20' en la ecuacion de calculo del rojo determina cada cuanto
13  *    se repite un color
14  *
15  * Nota: la funcion 'sin()' devuelve un numero decimal ('float' o en coma flotante)
16  *       y por eso hay que emplear la funcion 'int()' para poder asignarla a la
17  *       variable que almacena el color rojo
18  *
19  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
20  */
21
```

```
22 int rojo = 255;
23 int tiempo = 0;
24
25 void setup() {
26   size(400, 400);
27 }
28
29 void draw() {
30   tiempo = tiempo + 1;  // Incrementa el tiempo en una unidad
31   rojo = int(128 * (1 + sin(tiempo * 2 * PI / frameRate / 20)));  // Cambia el rojo,
repite el color cada 20s
32
33   noStroke();
34   fill(rojo, 0, 0);
35   ellipse(mouseX, mouseY, 30, 30);
36 }
```

Nuevos comandos:

- `sin( angle )`: Esta función se utiliza para calcular el seno de un ángulo. En nuestro caso, no tiene nada que ver con ángulos pero, ¿te acuerdas de la gráfica del seno de la clase de matemáticas? Bien, esa gráfica es el por qué es útil para nosotros. Siempre que quieras crear una oscilación suave y continua de un movimiento o, en este caso, un cambio de color, sin() es muy práctico de usar.
- `PI`: Esto es una variable constante que contiene el valor del número pi.
- `frameRate`: Esto es otra variable que te da el *framerate* (frecuencia de ejecución) del programa.

En este ejemplo, puedes ver como el color cambia gradualmente de forma continua. Esto se debe a que el color se establece por una función senoidal dependiendo de la frecuencia de ejecución del sketch (framerate) y la variable `time` que cuenta las veces que `draw()` se ha ejecutado.

Cambiando de forma

El siguiente paso es cambiar la forma de la serpiente. Esto se consigue fácilmente cambiando el tamaño del círculo, utilizando también una función senoidal.

```
 *
 * Red Snake (Serpiente Roja)
 *
 * Programa para crear una serpiente roja moviendose por la pantalla
 *
 * Paso 6:
 *  - cambia el tamaño del circulo en funcion del tiempo
 *  - al igual que antes, necesitaras crear una variable
 *    para almacenar el valor del tamaño
 *  - puedes usar la misma funcion que para el color, pero
 *    cambiando la frecuencia
 *  - para mejor efecto visual, añade transparencia al color
 *
 * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
 */

int rojo = 255;
int tiempo = 0;
int diametro = 50;

void setup() {
  size(400, 400);
}

void draw() {
  tiempo = tiempo + 1;
  rojo = int(128 * (1 + sin(tiempo * 2 * PI / frameRate / 20)));
   diametro = int(50 * (1 + sin( tiempo * 2 * PI / frameRate / 5))); // Modifica el
diametro del circulo con el paso del tiempo

  noStroke();
  fill(rojo, 0, 0, 50);  // Añade un 50% de transparencia al color
  ellipse(mouseX, mouseY, diametro, diametro);
}
```

Nuevos comandos:

- `fill(red, green, blue, alpha)`: Hemos añadido un cuarto parámetro `alpha` a la función `fill()`. Esto establece la transparencia del color y su rango va de 0 a 255.

Sigue experimentando

- La modificación más sencilla que puedes hacer es cambiar el color de la serpiente. Puede ser tan fácil como mover la variable `red` al segundo o tercer parámetro de la función `fill()`.
- Puedes también añadir otras variables para cambiar los otros parámetros del color de forma independiente.

Reloj de Post-it

En este miniproyecto vas a crear un reloj donde los números estarán hechos a partir de fotografías de notas adhesivas o Post-it. Para ello, aprenderás a utilizar imágenes en tus programas y almacenarlas en arrays (explicado luego). De nuevo, iremos avanzando paso por paso

Inserta una imagen en tu programa

En primer lugar, vamos a aprender a crear un programa capaz de mostrar una imagen. En Processing hay un tipo de variable (también conocido como clase) llamada `PImage`. Las variables de este tipo almacenan una imagen que podrás cargar de un fichero usando la función `loadImage()`.

```
1  /**
2   * Post-it Clock (Reloj de Post-it)
3   *
4   * El objetivo de este proyecto es crear un reloj usando imagenes para
5   * representar los numeros.
6   *
7   * Paso 1:
8   *  - crea un programa que muestre una imagen
9   *  - para cargar la imagen en tu programa, selecciona una imagen y arrastrala
10  *    sobre el IDE, asi se almacenara en una carpeta accesible a tu programa
11  *  - puedes comprobar que se ha creado una carpeta 'data' dentro de la carpeta
12  *    de tu programa, abre la carpeta desde el menu 'Sketch --> Show Sketch Folder'
13  *  - las imagenes:
14  *     + son variables del tipo 'PImage'
15  *     + se muestran con la funcion 'imgage()'
16  *     + se cargan con 'loadImage()'
17  *
18  *  Nota: antes de introducir una imagen, asegurate de que no es demasiado grande,
19  *        reducela de tamaño a 400x400 o similar para que encaje en tu pantalla
20  *
21  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
22  */
23
24 PImage im;
25
26 void setup() {
27   size(400, 400);   // Hacemos el programa del tamaño de la imagen
```

```
28    im = loadImage("foto.jpg");  // Nombre de la imagen
29  }
30
31  void draw() {
32    image(im, 0, 0);  // Para mostrar la imagen en la pantalla, coordenadas 0, 0
33  }
```

Nuevos comandos:

- `PImage`: Un tipo de variable que puede contener una imagen.
- `loadImage(image file)`: Carga la imagen `image file` localizada dentro del directorio actual del sketch. `image file` debe escribirse exactamente igual que con el nombre del fichero. Mira más abajo cómo colocar la imagen en la carpeta correcta.
- `Image(image name, x, y)`: Muestra la PImage `image name` en las coordenadas `x`, `y`. La esquina superior izquierda es el eje de coordenadas.

Añadir imágenes

Para que el programa tenga acceso a la imagen, debes arrastrar el fichero de la imagen en el IDE de Processing, como muestra la siguiente captura:

La imagen se almacena en una subcarpeta del programa llamada data. Esta captura de pantalla muestra la estructura de los ficheros donde puedes encontrar el archivo de la imagen. También puedes encontrarlo abriendo la carpeta desde el menú 'Sketch –> Mostrar carpeta del Sketch'.

Cuando ejecutes el programa, la imagen de abajo es lo que deberias ver, un collage con cuatro Post-its como imagen de fondo para tu ventana de programa:

En este programa vamos a crear una variable `image` llamada `im`. En `setup()` almacenamos una imagen llamada foto.jpg en nuestra variable, utilizando `loadImage()`. En `draw()` mostramos la imagen usando `imagen()`. Puesto que nuestra ventana de programa se establece al mismo tamaño que la imagen, dicha imagen encajará perfectamente con la ventana. Puedes cambiar la imagen con cualquier otra que hayas hecho tu mismo o descargado de Internet. Asegúrate de que el tamaño de ventana y de la imagen sean iguales para que el programa tenga una buena apariencia.

Mostrando dos imágenes

Para poner dos imágenes una al lado de la otra, debes declarar una variable con distinto nombre pero del mismo tipo PImage y cargar la segunda imagen en el programa. Deberás ajustar las coordenadas de esta segunda imagen.

```
1  /**
2   * Post-it Clock (Reloj de Post-it)
3   *
4   * El objetivo de este proyecto es crear un reloj usando imagenes para
5   * representar los numeros.
6   *
7   * Paso 2:
8   *  - modifica el programa para que muestre dos imagenes
9   *  - necesitaras dos variables, una para cada imagen
10  *  - la funcion 'imgage()' usa coordenadas, muestra las imagenes una junto a otra
11  *  - incluye la segunda imagen en el progama arrastrandola sobre el IDE
12  *
13  *  Nota: si tus imagenes son de 400x400, necesitaras un programa de 800x400 pixels
14  *
15  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
16  */
17
18 PImage im;
19 PImage im2;
20
21 void setup() {
22   size(800, 400);   // Hacemos el programa dos veces el tamaño de la imagen en ancho
23   im = loadImage("foto.jpg");
24   im2 = loadImage("foto2.jpg");  // Nombre de la segunda imagen
25 }
26
27 void draw() {
28   image(im, 0, 0);
29   image(im2, 400, 0);  // Para mostrar la segunda imagen en la pantalla, coordenadas
400, 0
30 }
```

Simplemente hemos añadido otra `PImage` llamada `im2`. Esta imagen es del mismo tamaño que la primera, por lo que la ventana debe ser el doble de grande en el eje x para que quepan las dos imágenes.

Arrays

Cuando trabajamos con muchas imágenes podemos utilizar algo llamado arrays que permiten un acceso más fácil a ellas. Un array es como una caja con muchos compartimentos. Cada compartimento puede tener diferentes objetos y se pueden encontrar referenciando dicho compartimento.

Por ejemplo, un array llamado CajaDeFruta tiene 3 compartimentos y puede contener 3 objetos. El compartimento 1 tiene un plátano, el compartimento 2 tiene una fresa y el compartimento 3 tiene una cereza. Si quieres encontrar la cereza, mirarás en el compartimento 3 de la CajaDeFruta. El orden de los arrays siempre empieza en 0, por lo que CajaDeFruta[0] contiene el plátano, CajaDeFruta[1] contiene la fresa y CajaDeFruta[2] la cereza.

Los arrays pueden ser de un cierto tipo como entero o `int` o una cadena como `String`. El máximo número de variables que pueden almacenar debe ser declarado con el array – 3 en el ejemplo de arriba. El siguiente programa, que es una modificación del anterior, muestra como usar un array del tipo `PImage` para almacenar dos imágenes.

```
1  /**
2   * Post-it Clock (Reloj de Post-it)
3   *
4   * El objetivo de este proyecto es crear un reloj usando imagenes para
5   * representar los numeros.
6   *
7   * Paso 3:
8   *  - modifica el programa para que use arrays
9   *  - los arrays:
10  *    + son variables que pueden almacenar mas de un dato
11  *    + tienen un solo nombre, por ejemplo 'img'
12  *    + los datos se distinguen porque "van numerados", es lo que llamamos "indexados"
13  *    + el primer dato es el 0, el segundo el 1, el tercero el 2 ...
14  *    + un dato se pide con el nombre del array seguido del numero entre
15  *      cochetes 'img[3]'
16  *    + se declaran usando el tipo de variable con corchetes (mira el codigo)
17  *    + al declararlos hay que decir cuantos datos van a contener
18  *
19  *  Nota: este programa hace lo mismo que el anterior, solo que de forma mas eficiente
20  *
21  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
22  */
23
24 PImage im[] = new PImage[2]; // Declaracion de un array de 2 imagenes
25
26 void setup() {
27   size(800, 400);
28   // La primera imagen va a la primera posicion del array img[]
29   im[0] = loadImage("foto.jpg");
30   // La segunda imagen va a la segunda posicion del array
31   im[1] = loadImage("foto2.jpg");
32 }
33
34 void draw() {
35   image(im[0], 0, 0);
36   image(im[1], 400, 0);
37 }
```

Nuevos comandos:

- La declaración de las imágenes es un poco diferente: `PImage im[] = new PImage[2]`, esto significa que declaramos un array del tipo `PImage` con dos bloques (o compartimentos) de datos.
- Cuando mostramos y cargamos las imágenes, usamos `img[0]`, para la primera posición del array, y `img[1]`, para la segunda.

A pesar de que el programa ha sido modificado, no hay ninguna diferencia en el resultado.

Usando dos Arrays

Para en adelante simplificar la asignación de las variables imagen, el truco es almacenar los nombres de todas las imágenes en un array de strings (cadenas de texto). Aquí un ejemplo:

```
1  /**
2   * Post-it Clock (Reloj de Post-it)
3   *
4   * El objetivo de este proyecto es crear un reloj usando imagenes para
5   * representar los numeros.
6   *
7   * Paso 4:
8   *  - modifica el programa de modo que los nombres de imagenes se almacenen
9   *    en un array, de este modo tendras dos arrays, uno para los nombres de
10  *    imagenes y otro para las imagenes
11  *  - toma ventaja del hecho de que se puede hacer un array declarando sus
12  *    contenidos entre llaves '{dato, dato2 ...}' con los datos separados por comas
13  *  - en el programa ejemplo, "foto.jpg" es la posicion 0 del array,
14  *    "foto2.jpg" es la 1
15  *
16  *  Nota: este programa hace lo mismo que el anterior
17  *
18  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
19  */
20
21 PImage im[] = new PImage[2];
22 String imFile[] = {"foto.jpg", "foto2.jpg"};  // Array de nombres de fichero
23
24 void setup() {
25   size(800, 400);
26   im[0] = loadImage(imFile[0]);  // Primer fichero a la funcion 'loadImage()'
27   im[1] = loadImage(imFile[1]);  // Segundo fichero
28 }
29
30 void draw() {
31   image(im[0], 0, 0);
32   image(im[1], 400, 0);
33 }
```

Como podrás observar cuando ejecutes este programa, el resultado es el mismo que para los otros. La gran pregunta es, ¿Por qué utilizar esto para hacer lo mismo? Y la respuesta es simple: lo hace mucho más fácil para seguir trabajando desde aquí. Si necesitas cambiar algo, por ejemplo, sólo lo tendrás que hacerlo en un único lugar y no buscar por todo el programa. Cuando añadas más imágenes y para programas muy complejos, esto es importante.

El bucle for()

En este paso aumentaremos el número de imágenes que se muestran de 2 a 4. Todavía utilizaremos los 2 arrays de los pasos anteriores, pero para hacer el código más eficiente incluso, introduciremos una cosa nueva: el bucle for(). Esta es una de las funciones más prácticas y usadas en programación. Hace exactamente lo que parece, iterar, al igual que draw(). Veamos un ejemplo para seguir explicando como funciona.
Descarga las imágenes banana.jpg, peninsula.jpg, postit.jpg y tortilla.jpg

```
/**
 * Post-it Clock (Reloj de Post-it)
 *
 * El objetivo de este proyecto es crear un reloj usando imagenes para
 * representar los numeros.
 *
 * Paso 5:
 *  - añade mas imagenes a tu programa (aqui te pasamos un par mas)
 *  - recuerda que tienes que arrastrarlas sobre el IDE o copiarlas directamente
 *    en la carpeta 'data' de tu programa
 *  - usa el bucle 'for()' para cargar todas tus imagenes en el array de forma
 *    mas eficiente
 *  - el bucle 'for()':
 *    + se usa en programacion para ejecutar acciones que se repiten varias veces
 *    + es util para recorrer arrays
 *    + tiene tres parametros separados por ';' (punto y coma)
 *    + for( inicializacion; comprobacion; incremento )
 *
 *  Nota: busca mas informacion sobre for en la referencia de Processing
 *
 * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
 */

PImage im[] = new PImage[4];   // Haz tu array mas grande para que le quepan 4 imagenes
String imFile[] = {"postit.jpg", "peninsula.jpg", "tortilla.jpg", "banana.jpg"};

void setup() {
  size(800, 800);  // Haz tu programa mas grande para acomodar todas las imagenes
  for (int i = 0; i < 4; i = i + 1) {
    im[i] = loadImage(imFile[i]);  // Carga la imagen segun el contador 'i'
  }
}

void draw() {
  image(im[0], 0, 0);   // Muestra las imagenes
  image(im[1], 400, 0);
```

```
37    image(im[2], 0, 400);
38    image(im[3], 400, 400);
39  }
```

Nuevos comandos:

- `for(initiator; test; update){ statements }`: Esta función permite repetir una pieza de código tantas veces como necesites. La primera cosa que sucede dentro del `for()` es la inicialización, `initiator`. En este ejemplo esta declaración es un entero, `i`. Luego comprueba si `test` es verdadero o falso, tal y como hicimos con el `if` en el proyecto anterior, ¿recuerdas?. En este ejemplo `test` comprueba si i es menor que 4. Si así es, el código entre llaves se ejecutará. A continuación, lo que ocurre es el `update`, en este caso se trata de incrementar `i` una unidad. Después, `test` se comprueba de nuevo y si, nuevamente es cierto, se repite lo anterior. Puesto que siempre actualizamos añadiendo 1 a `i`, en algún momento i será igual que 4, lo que significa que `test` será falsa (4 no es menor que 4). En ese momento, se abandonará el bucle.

Aquí está el resultado:

Lo que hemos hecho es crear dos arrays con 4 posiciones cada uno. Uno para las imágenes y otro para el nombre de las mismas. En `setup()` establecemos el tamaño de la ventana para que quepan las 4 imágenes. A continuación, utilizamos el bucle `for()` para que cargue las imágenes.

El bucle se ejecuta 4 veces, tantas como imágenes queremos cargar. Esto se establece en el `test` con la condición `i<4` y en el `update` escribiendo `i=i+1`. La mejor parte de esto es que podemos usar la variable `i` para acceder a las posiciones de nuestros arrays. Por tanto, la primera vez que entramos en el `for`, `i` es igual a 0, lo que significa que cargamos la imagen `imFile[0]` ("postit.jpg") y la guardamos en `im[0]`. La siguiente vez, i es 1, etc. De esta manera, no tenemos que asignar todas las imágenes una por una con variables separadas, simplemente dejamos al bucle `for()` hacer todo el trabajo. Con esto conseguimos un código más compacto.

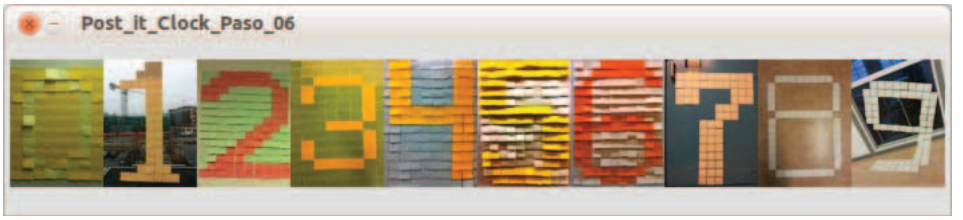
Múltiples imágenes

Puesto que el objetivo de este programa es crear un reloj, el siguiente paso es utilizar diferentes imágenes que representen los números y usarlas en nuestro programa. Para tu conveniencia, te damos una serie de imágenes hechas con notas de Post-It.

Descarga las imágenes que representan los números: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9

```
/**
 * Post-it Clock (Reloj de Post-it)
 *
 * El objetivo de este proyecto es crear un reloj usando imagenes para
 * representar los numeros.
 *
 * Paso 6:
 *  - crea imagenes que representen numeros (por si acaso, aqui tienes unas
 *    hechas con post-it)
 *  - modifica el programa para que muestre todos los numeros del 0 al 9
 *  - puedes usar 'for()' para mostrar las imagenes
 *  - este programa te servira como base para cualquier programa donde quieras
 *    representar numeros, si por ejemplo quieres mostrar el 5 en la pantalla,
 *    solo tienes que invocar 'image(img[5], coorX, coorY)'
 *
 *  Nota: como quieres mostrar todos los numeros, tendras que hacer imagenes mas
 *        pequeñas, por ejemplo de 70 pixels de ancho
 *
 * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
 */

PImage im[] = new PImage[10];   // Array para 10 imagenes
String imFile[] = {"0.jpg", "1.jpg", "2.jpg", "3.jpg", "4.jpg", "5.jpg", "6.jpg",
"7.jpg", "8.jpg", "9.jpg"};

void setup() {
  size(700, 95);  // 10 imagenes de 70 pixels de ancho y 95 de alto
  for (int i = 0; i < 10; i = i + 1) {
    im[i] = loadImage(imFile[i]);
  }
}

void draw() {
  for (int i = 0; i < 10; i = i + 1) {
    image(im[i], 70 * i, 0);  // Muestra las imagenes en secuencia
  }
```

No hemos introducido ningún comando nuevo aquí. Lo que hemos hecho ha sido crear 10 posiciones en cada array. Hemos establecido el tamaño de ventana a 700×95 para que quepan las 10 imágenes de 70×95. Tenemos un bucle `for()` que itera 10 veces para cargar cada una de las imágenes, y otro bucle `for` en `draw`. Este bucle itera diez veces y simplemente se usa para mostrar las imágenes. Puesto que las imágenes son de 70 píxels de ancho, podemos establecer la coordenada x para cada imagen como `70*i`.



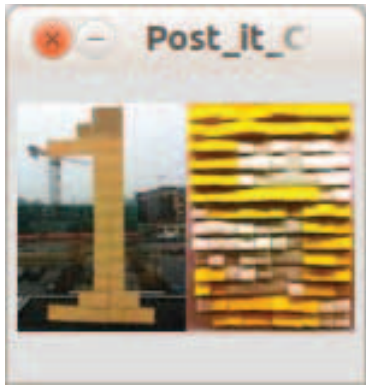
El Tiempo

En este paso te enseñaremos como mostrar el tiempo. Ya has aprendido como acceder a diferentes variables del sistema con Processing. Por ejemplo, ya hemos usado mouseX, mouseY y frameRate. Hay variables que podemos utilizar para obtener el tiempo actual; hour (), minute () y second (). Empecemos con cómo mostrar el tiempo por sólo las horas:

```
1  /**
2   * Post-it Clock (Reloj de Post-it)
3   *
4   * El objetivo de este proyecto es crear un reloj usando imagenes para
5   * representar los numeros.
6   *
7   * Paso 7:
8   *  - toma la hora del ordenador y representala con las imagenes
9   *  - la hora del sistema se obtiene con la funcion 'hour()'
10  *  - ten en cuenta que la hora tiene dos digitos y por lo tanto
11  *    tienes que buscar la forma de separarlos
12  *
13  * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
14  */
15
16 PImage im[] = new PImage[10];   // Array para 10 imagenes
17 String imFile[] = {"0.jpg", "1.jpg", "2.jpg", "3.jpg", "4.jpg", "5.jpg", "6.jpg",
"7.jpg", "8.jpg", "9.jpg"};
18
19 void setup() {
20   size(140, 95);  // 2 digitos de 70 pixels de ancho y 95 de alto
21   for (int i = 0; i < 10; i = i + 1) {
22     im[i] = loadImage(imFile[i]);
23   }
24 }
25
26 void draw() {
27   int h = hour();             // Toma la hora del reloj del ordenador y almacenalo en
una variable
28   int h_dec = int(h / 10);    // Extrae el digito de mayor peso de la hora (decenas)
29   int h_uni = h - h_dec * 10; // Extrae el digito de menor peso de la hora (unidades)
30
31   image(im[h_dec], 0, 0);     // Muestra el digito de las decenas
32   image(im[h_uni], 70, 0);    // Muestra el digito de las unidades
33 }
```

Nuevos comandos:

- `hour()`: Devuelve la hora actual como un número de 0 a 23.
- `int(data)`: Convierte `data` a un entero. P.ej. `int(2.545)` devuelve 2,`int(233.9999)` devuelve 233 etc.



La primera cosa que se hace en `draw()` es almacenar el número de horas en la variable `h`. Ahora este número tiene dos dígitos pero nuestras imágenes sólo contienen uno, por lo que tendremos que calcular qué dígitos mostrar. El primer dígito se almacena en `h_dec` y se calcula como `hour ()`, `minute ()` and `second ()`. El segundo dígito se almacena en `h_uni` y se calcula como `h - h_dec * 10`.

Digamos que la hora actual son las 14. Esto significa que `h` es igual a 14, `h_dec` es 1 y `h_uni` vale 4. Estas variables ahora pueden ser usadas para decidir qué imagen mostrar.

El reloj final

Este es el último paso del proyecto. En este paso vamos a añadir los minutos y los segundos a nuestro reloj utilizando el mismo método que en el paso anterior con las horas.

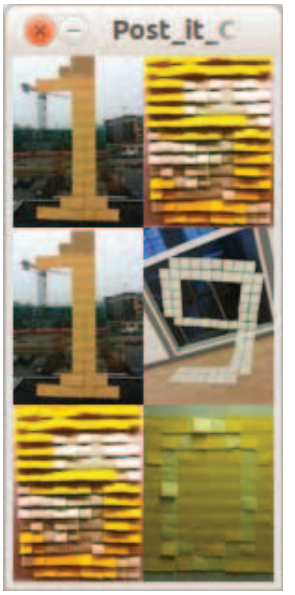
```
/**
 * Post-it Clock (Reloj de Post-it)
 *
 * El objetivo de este proyecto es crear un reloj usando imagenes para
 * representar los numeros.
 *
 * Paso 8:
 *  - añade los minutos y los segundos a tu reloj
 *  - los minutos del sistema se obtienen con la funcion 'minute()'
 *  - los segundos se obtienen con la funcion 'seconds()'
 *
 * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
 */

PImage im[] = new PImage[10];   // Array para 10 imagenes
String imFile[] = {"0.jpg", "1.jpg", "2.jpg", "3.jpg", "4.jpg", "5.jpg", "6.jpg", "7.jpg",
"8.jpg", "9.jpg"};

void setup() {
  size(140, 285);  // 6 digitos en tres filas y dos columnas
  for (int i = 0; i < 10; i = i + 1) {
```

```
    im[i] = loadImage(imFile[i]);
  }
}

void draw() {
  int h = hour();
  int h_dec = int(h / 10);
  int h_uni = h - h_dec * 10;

  image(im[h_dec], 0, 0);
  image(im[h_uni], 70, 0);

  int m = minute();            // Toma los minutos del reloj del ordenador y almacenalos
en una variable
  int m_dec = int(m / 10);    // Extrae el digito de mayor peso de los minutos (decenas)
  int m_uni = m - m_dec * 10; // Extrae el digito de menor peso de los minutos (unidades)

  image(im[m_dec], 0, 95);    // Muestra el digito de las decenas
  image(im[m_uni], 70, 95);   // Muestra el digito de las unidades

  int s = second();            // Toma los segundos del reloj del ordenador y almacenalos
en una variable
  int s_dec = int(s / 10);    // Extrae el digito de mayor peso de los segundos (decenas)
  int s_uni = s - s_dec * 10; // Extrae el digito de menor peso de los segundos (unidades)

  image(im[s_dec], 0, 190);   // Muestra el digito de las decenas
  image(im[s_uni], 70, 190);  // Muestra el digito de las unidades
}
```

Nuevos comandos:

- `minute()`: devuelve los minutos actuales como un valor de 0 a 59.
- `second()`: devuelve los segundos actuales como un valor entre 0 y 59.

Declaramos las variables para los minutos y segundos y calculamos los dígitos a mostrar de la misma manera que en el paso anterior.



Sigue experimentando

Usando las mismas variables de sistema, crea un temporizador para la cocina, por ejemplo un reloj que cuente hacia atrás con imágenes.

Caza la manzana

En este mini proyecto programarás un juego de ordenador. El objetivo de este ejercicio es llevar la programación un poco más lejos y crear un pequeño videojuego donde nuestro aguerrido héroe, el famoso científico Newton, intenta no perder la oportunidad de que la manzana le caiga en la cabeza.

Vamos a crear, paso a paso, un programa en el que Newton coleccione puntos durante medio minuto, al recibir tantos manzanazos en la cabeza como sea posible.

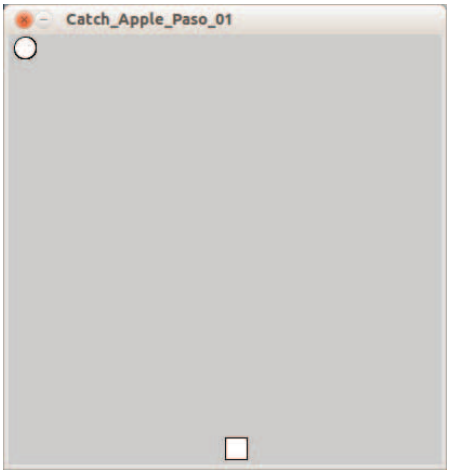
Crea una manzana y un científico

Crea la pantalla del juego. Las manzanas, por ahora, serán círculos que caen del cielo, mientras que Newton será un cuadrado al fondo de la pantalla.

```
1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will no let the chance
7   * go of having an apple hitting his head.
8   *
9   * Note: if you don't know about Newton's myth with the apple, you should
10  *       check it out on an encyclopedia, internet, or the like
11  *
12  * Step 1:
13  *  - creat the game screen, the apples will be circles falling from the sky, while
14  *    Newton will be represented by a square. We will change those later
15  *
16  * (c) 2013 D. Cuartielles, Arduino Verkstad, Sweden
17  */
18
19 void setup() {
20   size(400, 400);  // Make the screen of an average size
21 }
22
23 void draw() {
24   ellipse(15, 15, 20, 20);  // Draw the apple at the top of the screen
25   rect(width / 2, height - 25, 20, 20);  // Draw Newton at the bottom of the screen
26 }
```

Nuevos comandos:

`rect(x, y, width, height)`: Dibuja un rectángulo. `x` e `y` establecen la posición de la esquina superior izquierda, `width` y `height` establecen el tamaño.
De momento, Newton no se puede mover del centro de la ventana para coger la manzana, que está situada a su izquierda.



Toma el control de Newton con el teclado

Haz que Newton (el cuadrado) se mueva con las flechas del teclado. Para acceder al teclado desde Processing, considera lo siguiente: cada tecla del teclado tiene un `keyCode`. El `keyCode` es el valor o símbolo que representa una tecla específica. `keyCode` es una variable de sistema interna que puede ser usada para detectar qué tecla ha sido pulsada. Para ver que tecla ha sido apretada, vamos a usar una función llamada `keyPressed()`. Veremos que también necesitamos una variable para guardar las coordenadas del eje X (horizontal) del cuadrado, de forma que lo podamos ajustar cada vez que una tecla sea pulsada.

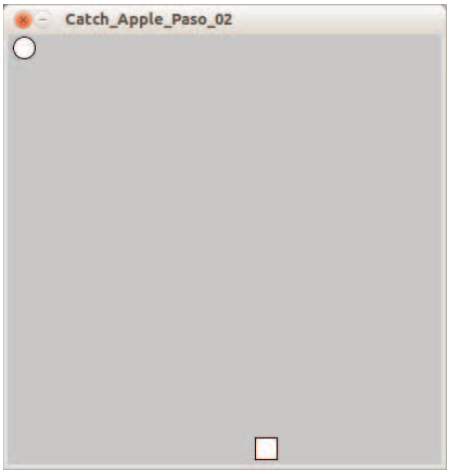
```
1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will no let the chance
7   * go of having an apple hitting his head.
8   *
9   * Step 2:
10  *  - make Newton (square) move using the keys in the keyboard
11  *  - each key in a keyboard has a 'keyCode' represting a value or
12  *    symbol code
13  *  - to check which key was pressed, create a function called 'keyPressed()'
14  *  - to move the square you will need a variable to store its X coordinate
15  *    (horizontal movement)
16  *
17  * (c) 2013 D. Cuartielles, Arduino Verkstad, Sweden
18  */
19
20 int nX = 0;
21
22 void setup() {
23   size(400, 400);  // Draw the sketch at a not-too-small size
24 }
```

```
25
26 void draw() {
27   background(200);  // Clear the screen
28   ellipse(15, 15, 20, 20);
29   rect(nX, height - 25, 20, 20);  // Draw Newton with a varaible X coordinate
30 }
31
32 void keyPressed() {
33   // If the right key was pressed, increase the X coordinate in 3 units
34   if (keyCode == RIGHT) {
35     nX = nX + 3;
36   }
37   // If the left key was pressed, decrease the X coordinate in 3 units
38   if (keyCode == LEFT) {
39     nX = nX - 3;
40   }
41 }
```

Nuevos comandos:

- `keyPressed()`: esta función se llama cada vez que una tecla es pulsada. Esto significa que cualquier código escrito dentro de esta función será ejecutado al pulsar una tecla.
- `keyCode`: devuelve el valor de la tecla que has pulsado.

La variable `nX` se usa para establecer la posición x de Newton. Cada vez que se pulsa el cursor derecho, `nX` se incrementa con 3 y cada vez que se pulsa el cursor izquierdo, `nX` se decrementa en 3. Ahora ya puedes mover el cuadrado por la pantalla de izquierda a derecha, pero ten cuidado, porque puede salirse de los límites de la pantalla.



Limita los movimientos del cuadrado

Limita el movimiento de Newton dentro de la ventana. Para esto vas a usar las funciones condicionales '`if-else`', con las que comprobarás que la coordenada X esté siempre dentro de la ventana del programa. Es decir, tendrá que ser mayor que 0 y menor que la anchura de la ventana o `width`.

```
/*
 * Catch the Apple (Caza la Manzana)
 *
 * Creacion de un videojuego con Processing. El objetivo de este ejercicio es
 * llevar la programacion un poco mas lejos y crear un pequeño videojuego donde
 * nuestro aguerrido heroe, el famoso cientifico Newton, intenta no perder la
```

```
 * oportunidad de que la manzana le caiga en la cabeza.
 *
 * Paso 3:
 *  - limita el movimiento de Newton dentro de la ventana
 *  - para esto vas a usar las funciones condicionales 'if-else'
 *  - comprobaras que la coordenada nX esta siempre dentro de la
 *    ventana del programa, es decir, habra de ser mayor que 0 y
 *    menor que la anchura de la ventana o 'width'
 *
 * (c) 2013 D. Cuartielles, Arduino Verkstad, Suecia
 */

int nX = 0;

void setup() {
  size(400, 400);
}

void draw() {
  background(200);
  ellipse(15, 15, 20, 20);
  rect(nX, height - 25, 20, 20);
}

void keyPressed() {
  // Incrementa las coordenadas en 3 unidades
  if (keyCode == RIGHT) {
    nX = nX + 3;
  }
  // Decrementa las coordenadas en 3 unidades
  if (keyCode == LEFT) {
    nX = nX - 3;
  }
  // Limita el valor de la coordenada X
  if (nX < 0) {
    nX = 0;
  }
  if (nX > width - 20) {  // Por la derecha tienes que tener en cuenta el tamaño del cuadrado
    nX = width - 20;
  }
}
```

En este caso, no hay una gran diferencia en lo que se verá en la pantalla del programa, con la excepción de que el cuadrado ya no se saldrá de los límites.

Manzanas que caen

Modifica el programa para hacer que la manzana (círculo) caiga de lo alto de la pantalla. Para esto, necesitarás una variable que contenga la coordenada Y del círculo y, cuando la manzana toque el suelo, que aparezca otra arriba.

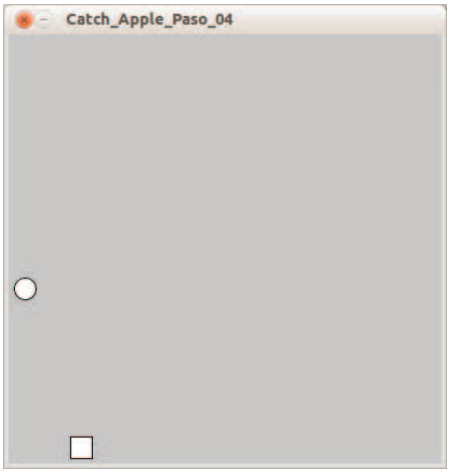
```
/*
 * Catch the Apple
 *
 * Create a videogame using Processing. The game for this exercise is
 * getting your programming skills a little further and making a computer
 * game where our hero, the famous scientist Newton, will no let the chance
 * go of having an apple hitting his head.
 *
 * Step 4:
 *  - modify the program to make the apple fall
 *  - you will need a variable to store the Y coordinate for the apple
 *  - when the apple touches the ground, you will need to “hang it” from the tree again
 *
 * (c) 2013 D. Cuartielles, Arduino Verkstad, Sweden
 */

int nX = 0;
int mY = 0;  // Apple’s Y coordinate

void setup() {
  size(400, 400);
}

void draw() {
  background(200);

  mY = mY + 1;  // Increase apple’s coordinate
  if (mY > height) {
    mY = 15;    // If the apple touches the ground, lift it again
  }
  ellipse(15, mY, 20, 20);   // Make the Y coordinate into a variable
  rect(nX, height - 25, 20, 20);
}

void keyPressed() {
  // Increase the coordinates in 3 pixels
  if (keyCode == RIGHT) {
    nX = nX + 3;
  }
  // Decrease the coordinates in 3 pixels
  if (keyCode == LEFT) {
    nX = nX - 3;
  }
  // Limit the X coordinates
```

```
45    if (nX < 0) {
46      nX = 0;
47    }
48    if (nX > width - 20) {
49      nX = width - 20;
50    }
51  }
```

Hemos creado una variable `mY` para almacenar la coordenada Y de la manzana. Cada vez que `draw()` se ejecuta, `mY` se incrementa en 1, haciendo que la manzana vaya acercándose al suelo de la ventana. Una vez que la manzana llega al suelo, es decir, `mY` es mayor que `height`, debe reaparecer mágicamente en lo alto de la ventana.

Un poco de azar

Hasta ahora, las manzanas siempre salen de la misma posición en lo alto de la pantalla, así es bastante predecible. Para cambiar la X de origen y que cada vez salga de un sitio distinto del árbol, haremos uso de una función llamada `random()` que permite generar números aleatorios. Necesitarás una nueva variable para almacenar la posición en X de la manzana. Ten en cuenta que la coordenada habrá que cambiarla solo cuando la manzana llegue al suelo, porque si no cambiará aleatoriamente durante su caída.




```
1   /*
2    * Catch the Apple
3    *
4    * Create a videogame using Processing. The game for this exercise is
5    * getting your programming skills a little further and making a computer
6    * game where our hero, the famous scientist Newton, will no let the chance
7    * go of having an apple hitting his head.
8    *
9    * Step 5:
10   *  - the apple is always falling from the same place on the screen, change its X
11   *    coordinate when created to show up at different places
12   *  - the 'random()' function will create a random number, it returns a number
13   *    between 0 and whatever number you pass to it as a parameter between the brackets
14   *  - you will need a new variable to store the X position of the apple
15   *  - the coordinate value has to be changed only when the apple touches the ground
16   *
17   * (c) 2013 D. Cuartielles, Arduino Verkstad, Sweden
18   */
19
20  int nX = 0;
21  int mY = 0;
```

```
22 int mX = 15; // Apple's X coordinate
23
24 void setup() {
25   size(400, 400);
26 }
27
28 void draw() {
29   background(200);
30
31   mY = mY + 1;
32   if (mY > height) {
33     mY = 15;
34      mX = int(random(width - 20));  // Initialize the X coordinate of the apple to a
random number
35   }
36   ellipse(mX, mY, 20, 20);  // Include the changes to the X coordinate to the circle's
movement
37   rect(nX, height - 25, 20, 20);
38 }
39
40 void keyPressed() {
41   // Increase the coordinates in 3 pixels
42   if (keyCode == RIGHT) {
43     nX = nX + 3;
44   }
45   // Decrease the coordinates in 3 pixels
46   if (keyCode == LEFT) {
47     nX = nX - 3;
48   }
49   // Limit the X coordinates
50   if (nX < 0) {
51     nX = 0;
52   }
53   if (nX > width - 20) {
54     nX = width - 20;
55   }
56 }
```

Nuevos comandos:

- `random(high)`: genera un número aleatorio entre 0 y el número high. Puedes también usar `random(low,high)` para generar un número entre `low` y `high`.

Con este cambio en el programa, podrás ver que las manzanas salen desde cualquier punto en lo alto de la pantalla.
Nuevos comandos:

Detección de colisión

Detecta que la manzana aterriza en la cabeza de Newton para poder contar puntos; la acción de detectar que dos objetos se chocan en la pantalla se llama detección de colisión. Utiliza las funciones condicionales `if-else` para ver si el círculo está sobre el cuadrado y, de ser así, cambia el color del cuadro a rojo. Para hacer el programa más sencillo, crearemos una variable que almacene las coordenadas Y del cuadrado, y así poder hacer las comparaciones de un modo más sencillo.

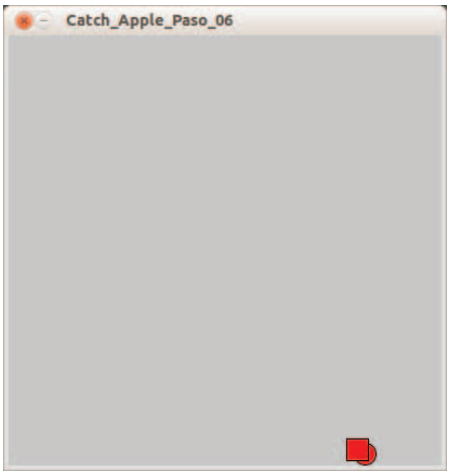
```
1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will no let the chance
7   * go of having an apple hitting his head.
8   *
9   * Step 6:
10  *  - detect the apple landing on Newton's head to be able of counting points;
11  *    checking whether two objects on a screen touch each other is called
12  *    "collision detection"
13  *  - use the conditional functions 'if-else' to check whether the circle is
14  *    on top of the square, if so, change the filling color to red
15  *  - to make the program easier, make a variable to store the square's Y coordinate
16  *
17  * (c) 2013 D. Cuartielles, Arduino Verkstad, Sweden
18  */
19
20 int nX = 0;
21 int nY = 0;  // Y coordinate for the square
22 int mY = 0;
23 int mX = 15;
24
25 void setup() {
26   size(400, 400);
27   nY = height - 25;  // Init the coordinate Y for the square to be at the end of the
screen
28 }
29
30 void draw() {
31   background(200);
32
33   mY = mY + 1;
34   if (mY > height) {
35     mY = 15;
36     mX = int(random(width - 20));
37   }
38
39   fill(255);  // By default fill in the shapes white
40
41   // Collision detection
```

```
42    if (mY + 10 > nY && mY - 10 < nY + 20) {  // Is the circle at the same height as the
square?
43      if (mX + 10 > nX && mX - 10 < nX + 20) { // Is the circle on top of the square?
44        fill(255, 0, 0);  // Change the filling color to red
45      }
46    }
47
48    // Lines to understand collision detection
49    // uncomment them to test how things work
50    //line(0,mY-10,width,mY-10);
51    //line(mX-10,0,mX-10,height);
52    //line(0,mY+10,width,mY+10);
53    //line(mX+10,0,mX+10,height);
54
55    ellipse(mX, mY, 20, 20);
56    rect(nX, nY, 20, 20);    // Include a variable to control the Y coordinate
57  }
58
59  void keyPressed() {
60    // Increase the coordinates in 3 pixels
61    if (keyCode == RIGHT) {
62      nX = nX + 3;
63    }
64    // Decrease the coordinates in 3 pixels
65    if (keyCode == LEFT) {
66      nX = nX - 3;
67    }
68    // Limit the X coordinates
69    if (nX < 0) {
70      nX = 0;
71    }
72    if (nX > width - 20) {
73      nX = width - 20;
74    }
75  }
```

Nuevos comandos:

- `if(test && test) {statements}`: esto se utiliza cuando varias comprobaciones deben ser realizadas en un único `if()`. En este ejemplo, comprobamos si `mY+10 > nY` y si `mY-10 < nY+20`. Si estas dos comprobaciones son ciertas simultáneamente, ejecutamos el código entre llaves.

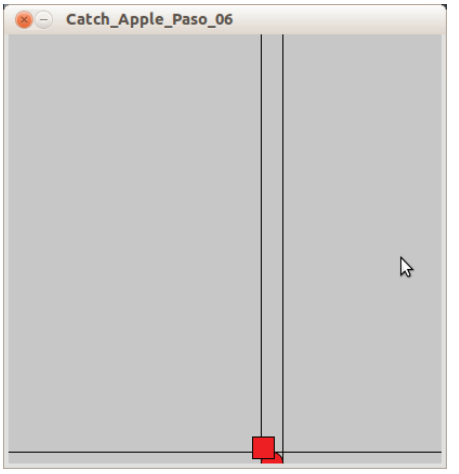
La detección de colisión consiste en dos `if()`, el primero comprueba si el círculo está a la misma altura que el cuadrado. Si es así, el segundo `if()` comprueba si el círculo está arriba del cuadrado. De ser así, el `fill()` colorea el círculo en rojo. Una colisión debe ser similar a esto:



Por otra parte, si activas las siguientes líneas en el programa:

```
1  // lines of code to understand how collision
works
2  // erase the comment in order to see the code
3  line(0,mY-10,width,mY-10);
4  line(mX-10,0,mX-10,height);
5  line(0,mY+10,width,mY+10);
6  line(mX+10,0,mX+10,height);
```

Verás una serie de líneas en la pantalla enmarcando el movimiento de la manzana. Puedes emplearlas para ver cómo funciona la detección de colisión.



Más rápido

¿No quieres que las manzanas caigan más rápido? Modifica la velocidad de las manzanas para hacer el juego algo más interesante. Para ello, crea una variable `float` para que `mV` almacene la velocidad de caída de la manzana. Puedes cambiar la velocidad modificando el valor de `mV`. Al mismo tiempo, para poder controlar mejor el movimiento de la manzana en el eje Y, vamos a modificar el tipo de variable de `mY` para que sea `float`. ¿Recuerdas `float` de la sección de variables? Una variable con datos del tipo `float` puede almacenar números decimales.

```
/*
 * Catch the Apple
 *
 * Create a videogame using Processing. The game for this exercise is
 * getting your programming skills a little further and making a computer
 * game where our hero, the famous scientist Newton, will no let the chance
 * go of having an apple hitting his head.
 *
 * Step 7:
 *  - modify the way the apples fall to reach higher speeds
 *  - introduce a variable to store the speed
 *  - declare that variable as 'mV' as a floating point number
 *  - you can modify the speed by changing just this variable
 *  - to control the apple's movement on the Y axis, modify the type of
 *    the 'mY' variable to be a 'float'
 *
 * (c) 2013 D. Cuartielles, Arduino Verkstad, Sweden
 */

int nX = 0;
int nY = 0;
float mY = 0;  // Make mY into a float
int mX = 15;
float mV = 3;  // Apple's falling speed

void setup() {
  size(400, 400);
  nY = height - 25;
}

void draw() {
  background(200);

  // Apple's movement
  mY = mY + mV;  // Introduce the speed as an increment
  if (mY > height) {
    mY = 15;
    mX = int(random(width - 20));
  }

  fill(255);

  // Collision detection
  if (mY + 10 > nY && mY - 10 < nY + 20) {
    if (mX + 10 > nX && mX - 10 < nX + 20) {
      fill(255, 0, 0);
    }
  }

  ellipse(mX, mY, 20, 20);
  rect(nX, nY, 20, 20);
```

```
52 }
53
54 void keyPressed() {
55   // Increase the coordinates in 3 pixels
56   if (keyCode == RIGHT) {
57     nX = nX + 3;
58   }
59   // Decrease the coordinates in 3 pixels
60   if (keyCode == LEFT) {
61     nX = nX - 3;
62   }
63   // Limit the X coordinates
64   if (nX < 0) {
65     nX = 0;
66   }
67   if (nX > width - 20) {
68     nX = width - 20;
69   }
70 }
```

En lugar de incrementar `mY` en uno, vamos a incrementarlo con `mV`. A parte de la velocidad, nada ha cambiado en el programa, por lo que sólo hay una pequeña diferencia entre este y el programa anterior.

A Newton le gusta la gravedad, dale más

Modifica la caída de las manzanas para que responda a la aceleración de la gravedad. De este modo, las manzanas irán cada vez más rápido cuanto más tiempo lleven cayendo. Como sabes, la velocidad se calcula a partir de la aceleración, y del mismo modo, la posición a partir de la velocidad. Para hacer estas operaciones de la forma más sencilla posible, introduce una variable en coma flotante (`float`) que represente la aceleración.

```
1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will no let the chance
7   * go of having an apple hitting his head.
8   *
9   * Step 8:
10  *  - mofify the apples' falling speed for them to follow the gravity
11  *  - as you know, speed can be calculated from acceleration, and position
12  *    from speed
13  *  - introduce a variable representing acceleration, make it a 'float'
14  *
15  * (c) 2013 D. Cuartielles, Arduino Verkstad, Sweden
16  */
17
18 int nX = 0;
19 int nY = 0;
```

```
float mY = 0;
int mX = 15;
float mV = 0;     // Apple's initial speed is zero
float mA = 0.05;  // Apple's intial accerlation (0.98 would be too much)

void setup() {
  size(400, 400);
  nY = height - 25;
}

void draw() {
  background(200);

  // Apple's movement
  mV = mV + mA;  // Estimate the speed according to the acceleration
  mY = mY + mV;  // Estimate the position according to the speed
  if (mY > height) {
    mY = 15;
    mX = int(random(width - 20));
    mV = 0;  // Apples start falling at zero speed
  }

  fill(255);

  // Collision detection
  if (mY + 10 > nY && mY - 10 < nY + 20) {
    if (mX + 10 > nX && mX - 10 < nX + 20) {
      fill(255, 0, 0);
    }
  }

  ellipse(mX, mY, 20, 20);
  rect(nX, nY, 20, 20);
}

void keyPressed() {
  // Increase the coordinates in 3 pixels
  if (keyCode == RIGHT) {
    nX = nX + 3;
  }
  // Decrease the coordinates in 3 pixels
  if (keyCode == LEFT) {
    nX = nX - 3;
  }
  // Limit the X coordinates
  if (nX < 0) {
    nX = 0;
  }
  if (nX > width - 20) {
    nX = width - 20;
  }
}
```

Cada vez que incrementes `mY` incrementaremos también `mV` un poco. De esta manera, la velocidad será un poco más grande cada vez. A parte de la aceleración, nada ha cambiado en el programa, por lo que sólo hay una pequeña diferencia entre este programa y el anterior.

Cuenta los puntos

Implementa un contador que muestre cuántas manzanas golpearon a Newton. Para mostrar el texto, puedes utilizar la función `text()`. Además, necesitarás un contador para anotar los puntos.

Nota: como empiezas a tener muchas variables en tu programa, es recomendable que añadas comentarios para recordar qué hace cada una.

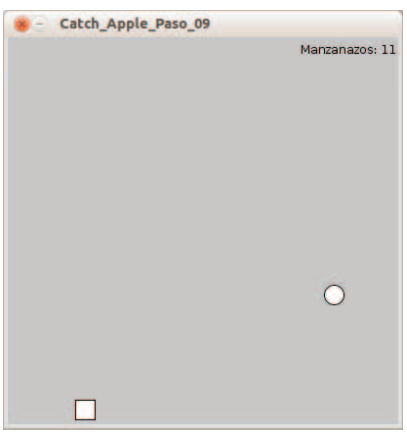
```
/*
 * Catch the Apple
 *
 * Create a videogame using Processing. The game for this exercise is
 * getting your programming skills a little further and making a computer
 * game where our hero, the famous scientist Newton, will no let the chance
 * go of having an apple hitting his head.
 *
 * Step 9:
 *  - implement a counter to show home many apples Newton caught
 *  - the function 'text()' will allow add text to the sketch
 *  - you will also need a counter to follow the points scored while playing
 *  - you start to have many variables in your program, it is considered good
 *    practice to add comments to explain each one of them
 *
 * (c) 2013 D. Cuartielles, Arduino Verkstad, Sweden
 */

int nX = 0;      // X coordinate, Newton
int nY = 0;      // Y coordinate, Newton
float mY = 0;    // Y coordinate, apples
int mX = 15;     // X coordinate, apples
float mV = 0;    // Y speed, apples
float mA = 0.05; // Y acceleration, apples
int p = 0;       // Points

void setup() {
  size(400, 400);
  nY = height - 25;
}

void draw() {
  background(200);

  // Apple's movement
  mV = mV + mA;
  mY = mY + mV;
  if (mY > height) {
```

```
39      mY = 15;
40      mX = int(random(width - 20));
41      mV = 0;
42    }
43
44    fill(255);
45
46    // Collision detection
47    if (mY + 10 > nY && mY - 10 < nY + 20) {
48      if (mX + 10 > nX && mX - 10 < nX + 20) {
49        fill(255, 0, 0);
50        // If collision increase the points
51        p = p + 1;
52      }
53    }
54
55    ellipse(mX, mY, 20, 20);
56    rect(nX, nY, 20, 20);
57
58    // Show the points on the screen
59    fill(0);
60    text("Hits: " + p, 3 * width / 4, 20); // Text to the right on the screen
61  }
62
63  void keyPressed() {
64    // Increase the coordinates in 3 pixels
65    if (keyCode == RIGHT) {
66      nX = nX + 3;
67    }
68    // Decrease the coordinates in 3 pixels
69    if (keyCode == LEFT) {
70      nX = nX - 3;
71    }
72    // Limit the X coordinates
73    if (nX < 0) {
74      nX = 0;
75    }
76    if (nX > width - 20) {
77      nX = width - 20;
78    }
79  }
```

Nuevos comandos:

- `text(text, x, y)`: escribe un texto en la pantalla en las coordenadas `x` e `y`.

Hemos declarado una nueva variable `p` que se incrementa en una unidad cada vez que se detecta una colisión. La última cosa que hacemos en `draw()` es mostrar el contador de los puntos en la esquina superior derecha.



Uoops, error

Te habrás dado cuenta que tu programa ahora mismo está contabilizando puntos de más. Cada vez que la manzana cae sobre la cabeza de Newton... eh... cuando el círculo toca el cuadrado, tu contador sube más o menos 5 puntos. Para corregir esto tienes un par de opciones:

- Vuelve a lanzar la manzana en cuanto se detecte colisión.
- Deja de contar cuando haya colisión, hasta que se lance una nueva manzana.

Para evitar cualquiera de estas dos posibilidades, necesitas una variable de tipo `boolean` donde almacenar el estado de la manzana. ¿Recuerdas qué era una variable `boolean`? Podemos usar esta variable para decirle al programa si contar puntos o no.

```
1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will no let the chance
7   * go of having an apple hitting his head.
8   *
9   * Step 10:
10  *  - the points are being counted extra, fix it
11  *  - you can achieve this in two different ways:
12  *    + you can throw the apple again as soon as you detect collision, or
13  *    + stop counting once there is a collision and until a new apple
14  *      falls from the tree
15  *  - you will need a 'boolean' variable to check whether to add points or
16  *    not, you will have to reset it to 'true' each time a new apple
17  *    falls from the top of the tree
18  *  - los puntos se estan contabilizando extra, corrigelo
19  *
20  * (c) 2013 D. Cuartielles, Arduino Verkstad, Sweden
21  */
22
23 int nX = 0;      // X coordinate, Newton
24 int nY = 0;      // Y coordinate, Newton
25 float mY = 0;    // Y coordinate, apples
26 int mX = 15;     // X coordinate, apples
27 float mV = 0;    // Y speed, apples
28 float mA = 0.05; // Y acceleration, apples
29 int p = 0;       // Points
30 boolean pCount = true;  // Check whether to count points or not
31
32 void setup() {
33   size(400, 400);
34   nY = height - 25;
35 }
36
```

```
void draw() {
  background(200);

  // Apple's movement
  mV = mV + mA;
  mY = mY + mV;
  if (mY > height) {
    mY = 15;
    mX = int(random(width - 20));
    mV = 0;
    // When throwing a new apple it will be possible
    // to start counting points again
    pCount = true;
  }

  // Collision detection
  if (mY + 10 > nY && mY - 10 < nY + 20) {
    if (mX + 10 > nX && mX - 10 < nX + 20) {
      fill(255, 0, 0);
      // If collision increase the points
      if (pCount) p = p + 1;
      pCount = false;  // Whenever you make it at this point, do not
                       // count any more points
    }
  }

  ellipse(mX, mY, 20, 20);
  rect(nX, nY, 20, 20);

  // Show the points on the screen
  fill(0);  // Text color
  text("Hits: " + p, 3 * width / 4, 20); // Text alligned to the right
}

void keyPressed() {
  // Increase the coordinates in 3 pixels
  if (keyCode == RIGHT) {
    nX = nX + 3;
  }
  // Decrease the coordinates in 3 pixels
  if (keyCode == LEFT) {
    nX = nX - 3;
  }
  // Limit the X coordinates
  if (nX < 0) {
    nX = 0;
  }
  if (nX > width - 20) {
    nX = width - 20;
  }
}
```

Nuevos comandos:

- `if( boolean )`: comprueba si `boolean` es `true`. Puedes también comprobar si es falso escribiendo `if( !boolean )`.

Ahora prueba el programa para ver cómo cuenta un único punto por manzana. Esto es porque sólo añadimos puntos cuando la variable `boolean pCount` es `true` y cambia a `false` justo después de contar el punto. Cuando la manzana es relanzada, `pCount` se vuelve a poner en `true.`

Y el tiempo empieza

El objetivo del juego es recibir tantas manzanas como sea posible en un periodo de tiempo determinado. Para poder contabilizar esto, sólo necesitamos un contador de tiempo, que también se muestre en la pantalla, para que los jugadores sepan cuándo ha terminado la partida. La duración óptima del juego es de medio minuto, si es demasiado corto, no habrá forma de conseguir puntos mientras que, si es demasiado largo, se hará aburrido. Para medir el tiempo, puedes usar una función llamada `millis()` que contabiliza los milisegundos que han pasado desde la última vez que se le llamó. De este modo puedes ver si han pasado 30000 milisegundos (30 segundos) para terminar el juego. Para almacenar el tiempo, necesitas una variable de tipo `long`. La función `noLoop()` le dirá a tu programa que termine una vez hayas llegado al final del mismo. Finalmente, usa `text()` para mostrar el tiempo que queda en la pantalla.

```
1  /*
2   * Catch the Apple
3   *
4   * Create a videogame using Processing. The game for this exercise is
5   * getting your programming skills a little further and making a computer
6   * game where our hero, the famous scientist Newton, will no let the chance
7   * go of having an apple hitting his head.
8   *
9   * Step 11:
10  *  - add a time counter to limit the game duration
11  *  - an optimal duration for the game is half a minute, there is a
12  *    function called 'millis()' that counts the milliseconds that
13  *    passed since the last time you called it. In this way you can check if
14  *    30000 milliseconds passed (or 30 seconds) in order to end the game
15  *  - to store the time you need a variable of type 'long'
16  *  - end the game with the function 'noLoop()'
17  *  - use 'text()' to show the remaining time on the screen
18  *
19  * (c) 2013 D. Cuartielles, Arduino Verkstad, Sweden
20  */
21
22 int nX = 0;      // X coordinate, Newton
23 int nY = 0;      // Y coordinate, Newton
24 float mY = 0;    // Y coordinate, apples
25 int mX = 15;     // X coordinate, apples
26 float mV = 0;    // Y speed, apples
27 float mA = 0.05; // Y acceleration, apples
```

```
int p = 0;       // Points
boolean pCount = true;  // Check whether to count points or not
long t = 0;      // Store the time

void setup() {
  size(400, 400);
  nY = height - 25;
  t = millis();  // Initialize the time counter
}

void draw() {
  background(200);

  // Apple's movement
  mV = mV + mA;
  mY = mY + mV;
  if (mY > height) {
    mY = 15;
    mX = int(random(width - 20));
    mV = 0;
    // When throwing a new apple it will be possible
    // to start counting points again
    pCount = true;
  }

  fill(255);

  // Collision detection
  if (mY + 10 > nY && mY - 10 < nY + 20) {
    if (mX + 10 > nX && mX - 10 < nX + 20) {
      fill(255, 0, 0);
      // If collision increase the points
      if (pCount) p = p + 1;
      pCount = false;  // Whenever you make it at this point, do not
                       // count any more points
    }
  }

  ellipse(mX, mY, 20, 20);
  rect(nX, nY, 20, 20);


  // Count the time
  float timer = (millis() - t) / 1000;  // Count how much time has passed in seconds

  // GAME OVER
  if (timer >= 30) {  // If time reaches 30 seconds, end the game
    noLoop();
  }

  // Show the time on the screen
  fill(0);
  text("Time: " + (30 - timer), 10, 20);
```

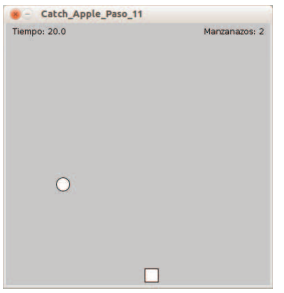
```
81
82    // Show the points on the screen
83    fill(0);
84    text("Hits: " + p, 3 * width / 4, 20);
85  }
86
87  void keyPressed() {
88    // Increase the coordinates in 3 pixels
89    if (keyCode == RIGHT) {
90      nX = nX + 3;
91    }
92    // Decrease the coordinates in 3 pixels
93    if (keyCode == LEFT) {
94      nX = nX - 3;
95    }
96    // Limit the X coordinates
97    if (nX < 0) {
98      nX = 0;
99    }
100   if (nX > width - 20) {
101     nX = width - 20;
102   }
103 }
```

Nuevos comandos:

- `long`: este es un tipo de datos para números enteros largos. Es conveniente usarlos cuando manejamos variables temporales, porque estos datos pueden ser muy grandes. ¿Recuerdas cómo explicamos que una variables es un contenedor de datos? Bien, se puede decir que `long` es un contenedor más grande que `int`. Un `int` se queda sin espacio antes que un `long`.
- `noLoop()`: detiene la ejecución continua dentro de `draw`. Para volver a ejecutar el programa tendrás que llamar a `loop()`.

La variable de tiempo `t` se inicializa en el `setup()`. Puesto que `millis()`cuenta la cantidad de milisegundos que el programa se ha ejecutado, `t` tomará un valor cercano a 0. Digamos que, para hacerlo más fácil, es igual a 0. Después de haber dibujado a Newton y la manzana, creamos una nueva variable llamada `timer` que calcula cuanto tiempo ha pasado. Esto se hace con `(millis()-t)/1000`. Cuando han pasado 10 segundos, este cálculo sería: `(10000-0)/1000`. Si `timer` es mayor que 30, llamamos a `noLoop()` para parar el programa. Pero puesto que `timer` no es mayor que 30, utiliza `text()` para mostrar cuanto tiempo queda.

El juego con contador de tiempo se verá como sigue:



Añade imágenes al juego

Incluye imágenes para la manzana, el fondo y Newton. Las puedes crear tú, buscarlas en Internet, o usar las que te adjuntamos. Es importante que las imágenes sean de tipo PNG si quieres que haya transparencia entre las imágenes y el fondo. Ten en cuenta que, al cambiar las formas por imágenes, las proporciones también cambian, por lo que tendrás que hacer encajar esos valores en la parte del programa dedicado a la detección de colisiones.

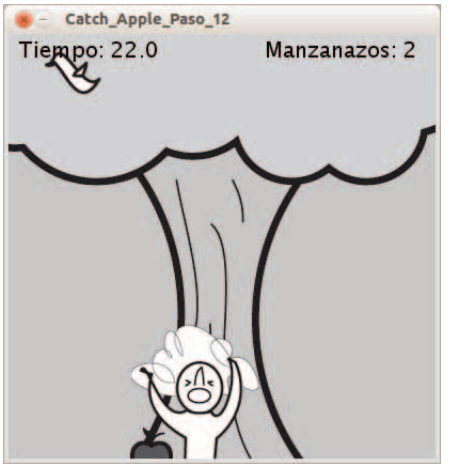
```
/*
 * Catch the Apple
 *
 * Create a videogame using Processing. The game for this exercise is
 * getting your programming skills a little further and making a computer
 * game where our hero, the famous scientist Newton, will no let the chance
 * go of having an apple hitting his head.
 *
 * Step 12:
 *  - include images to show the apple, the background picture and Newton
 *  - you can make your own images or look for them on the internet
 *  - it is important that you use PNG images if you want to have transparency
 *    between the background and the other images
 *  - check that when you change the images, you are also changing the game's
 *    proportions, you might have to adjust that in the code
 *
 * (c) 2013 D. Cuartielles, Arduino Verkstad, Sweden
 */

String[] imFiles = {"fondo.png", "manzana.png", "newton1.png", "newton2.png"};
PImage[] im = new PImage[4];

int nX = 0;      // X coordinate, Newton
int nY = 0;      // Y coordinate, Newton
float mY = 0;    // Y coordinate, apples
int mX = 15;     // X coordinate, apples
float mV = 0;    // Y speed, apples
float mA = 0.05; // Y acceleration, apples
int p = 0;       // Points
boolean pCount = true;  // Check whether to count points or not
long t = 0;      // Store the time

void setup() {
  size(400, 400);
  nY = height - 135;
  t = millis();

  // Load the images
  for(int i = 0; i < 4; i = i + 1) {
    im[i] = loadImage(imFiles[i]);
  }
}

void draw() {
```

```
  background(200);
  image(im[0], 0, 0, width, height);  // Background image

  // Apple's movement
  mV = mV + mA;
  mY = mY + mV;
  if (mY > height) {
    mY = 15;
    mX = int(random(width - 20));
    mV = 0;
    // When throwing a new apple it will be possible
    // to start counting points again
    pCount = true;
  }

  fill(255);

  // Collision detection
  if (mY + 50 > nY && mY < nY + 135) {
    if (mX + 40 > nX && mX < nX + 128) {
      fill(255, 0, 0);
      // If collision increase the points
      if (pCount) p = p + 1;
      pCount = false;  // Whenever you make it at this point, do not
                       // count any more points
    }
  }

  image(im[1], mX, mY);  // Apple
  if(pCount) {
    image(im[2], nX, nY);  // Newton looking for apples
  } else {
    image(im[3], nX, nY);  // Newton got an apple
  }

  // Count the time
  float timer = (millis() - t) / 1000;  // Count how much time has passed in seconds

  // GAME OVER
  if (timer >= 30) {  // If time reaches 30 seconds, end the game
    noLoop();
  }

  // Show the time on the screen
  fill(0);
  textSize(20);  // Increase the font size
  text("Time: " + (30 - timer), 10, 20);

  // Show the points on the screen
  fill(0);
  textSize(20);  // Increase the font size
  text("Hits: " + p, 3 * width / 4, 20);
}
```

```
98
99  void keyPressed() {
100   // Increase the coordinates in 3 pixels
101   if (keyCode == RIGHT) {
102     nX = nX + 3;
103   }
104   // Decrease the coordinates in 3 pixels
105   if (keyCode == LEFT) {
106     nX = nX - 3;
107   }
108   // Limit the X coordinates
109   if (nX < 0) {
110     nX = 0;
111   }
112   if (nX > width - 20) {
113     nX = width - 20;
114   }
115 }
```

Para cargar las imágenes utilizamos el mismo método que en los proyectos anteriores, utilizando arrays. En lugar de cambiar el color de las formas cuando una colisión sea detectada, utilizamos el boolean `pCount` para decidir qué imagen de Newton mostrar.

El resultado final del juego se ve como sigue:



¡Sigue experimentando!

Para mejorar este juego puedes hacer varias cosas:

- Haz tus propias imágenes.
- Crea una pantalla de inicio y que se pase al juego una vez se presione un botón.
- Crea una pantalla final que muestre el resultado una vez se haya terminado el tiempo.
- Haz posible reiniciar el juego cuando el tiempo haya terminado. No te olvides de reiniciar todas las variables necesarias.
- Usa superpoderes científicos, haz que Newton se pueda mover con aceleración y no solo a velocidad constante.

SEMANA 2
Deportes

CONCEPTOS

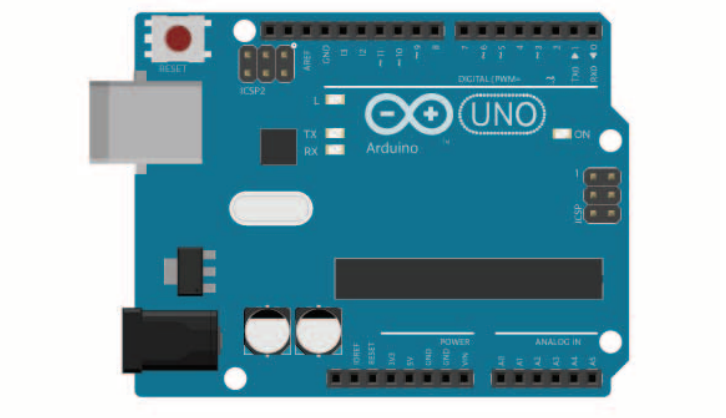
PROYECTOS

Esta semana vamos a construir algunos proyectos relacionados con los deportes. Os juntaréis en grupos y construiréis pequeños juegos electrónicos que simularán diferentes deportes. Al final de la semana, os reuniréis todos y haréis las Electrolimpiadas, donde tendréis que competir para ver quién es el mejor en las distintas disciplinas.

Pero antes de comenzar, necesitaremos saber algunas cosas sobre las herramientas que vamos a utilizar. Veamos brevemente qué es Arduino y algunos conceptos básicos sobre tecnología digital.

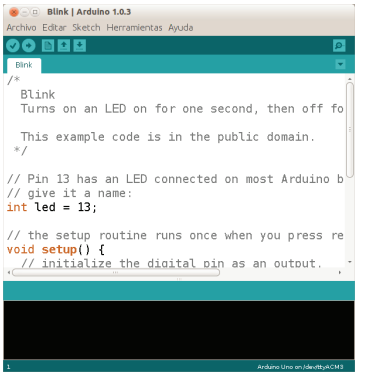
Qué es Arduino

Las placas Arduino son pequeños ordenadores con los que puedes leer información de diferentes sensores, así como controlar luces, motores y muchas otras cosas. La gran mayoría de los sistemas que nos rodean son ordenadores de diferentes tamaños. Los ordenadores no necesitan tener teclado ni pantalla. Hay ordenadores en el microondas de la cocina, dentro de los ascensores para detectar qué botón pulsas y, en los coches. Hay más de 70 ordenadores similares a Arduino... hay ordenadores por todas partes.



Puesto que Arduino, a diferencia del ordenador que usas normalmente, no tiene pantalla ni teclado, se necesita un programa externo ejecutado en otro ordenador para poder escribir programas para la placa Arduino. Éste software es lo que llamamos Arduino IDE. IDE significa "Integrated Development Environment" (Entorno de Desarrollo Integrado), y es un término común para llamar a este tipo de desarrollo de software. Escribes tu programa en el IDE, lo cargas en el Arduino, y el programa se ejecutará en la placa.

El IDE de Arduino es muy sencillo y parecido a Processing. Hay una sencilla barra de herramientas que puedes utilizar para:



Verificar si tu programa va a funcionar.

Cargar el programa a la placa de Arduino.

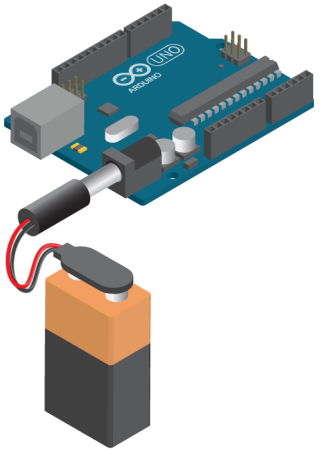
Crear un programa nuevo.

Abrir un programa.

Guardar el programa en el disco duro del ordenador.

(En la parte derecha de la barra de herramientas se encuentra el Monitor Serial) abre una ventana de comunicación con la placa Arduino.

Las placas Arduino se conectan a tu ordenador utilizando un cable USB, al igual que cualquier otro periférico, como la impresora, el teclado o incluso, un mando de videojuegos. Arduino necesita estar conectado al ordenador a través del cable USB para cargar un programa. El cable USB sirve también para suministrar energía a la placa, pero también puedes alimentarla usando una fuente de energía externa, como una batería o un transformador apropiado.



Cuando programes tu Arduino, debes asegurarte de que el IDE está configurado correctamente para la placa que estés utilizando. Compruébalo en el menú 'Herramientas → Puerto Serial' para ver que el puerto esté configurado correctamente, y en 'Herramientas → Tarjeta' para comprobar que esté asignado 'Arduino Uno'.

Nota: para saber a qué Puerto Serial está conectado tu Arduino, sigue estos pasos:

1. Desconecta Arduino del ordenador.
2. Comprueba 'Herramientas → Puerto Serial' para ver si está disponible.
3. Conecta Arduino al ordenador.
4. Entra en 'Herramientas → Puerto Serial' de nuevo y verás un puerto nuevo, ése es tu Arduino.
5. Selecciona ese puerto.

Señales digitales

El alfabeto Español consta de 27 símbolos, más otros tantos para las mayúsculas, y 10 símbolos para los números. A diferencia de los humanos, los ordenadores trabajan y se comunican con sólo 2 símbolos; '1' y '0'. Esto es lo que llamamos señales digitales. Usando una combinación de éstos símbolos, las máquinas digitales puedes representar todo lo que hay en el Universo.

Una característica común a todos los ordenadores es que utilizan la lógica binaria. Esto quiere decir que pueden solucionar múltiples operaciones utilizando solo dos símbolos básicos para representar toda la información. Los seres humanos, por ejemplo, utilizamos muchos símbolos. El abecedario español, tiene 27 símbolos en minúscula, más las mismas en mayúscula, además de 10 símbolos numéricos. Los ordenadores solo utilizan dos: '1' y '0'. Combinando estos dos números, las máquinas digitales pueden representar prácticamente todo lo que hay en el Universo.
Arduino representa un '1' con 5 Voltios, y un '0'con 0 Voltios. Cuando escribes un programa para Arduino, tú representas un '1' escribiendo `HIGH` y '0' escribiendo `LOW`. Cuando compilas el código, lo que realmente pasa es que lo que has escrito con símbolos humanos se traduce a unos y ceros, el lenguaje que los ordenadores entienden.

Al hablar de señales digitales y Arduino, hablamos de entradas y salidas. Una entrada digital significa que Arduino está recibiendo datos digitales de un sensor, por ejemplo, un botón. Cuando leemos desde un botón, Arduino recibirá bien 5V, `HIGH` o 0V, `LOW`, dependiendo si el botón está pulsado o no. Una salida digital significa que Arduino está mandado datos digitales a un actuador, como por ejemplo, un LED. Para encender un LED, Arduino manda 5V, `HIGH`; y para apagarlo, manda 0V, `LOW`.

on	off
1	0
5V	0V
HIGH	LOW

Lógica binaria

Otra característica común en todos los ordenadores es que utilizan lógica binaria. Lógica binaria significa que sólo hay dos posibilidades. Puesto que un ordenador utiliza sólo dos símbolos, '0' y '1', decimos que ellos también emplean la lógica binaria. En este caso, '1'representa `TRUE`, mientras que '0' representa `FALSE`. Esto es así porque la lógica binaria también puede ser usada para hacer preguntas como '¿Ha alcanzado la temperatura 20 grados?', la respuesta es o bien cierta, o bien falsa, y por tanto puede ser representada por '0' ó '1'.

Cuenta en binario

Los seres humanos tenemos un total de 10 símbolos para representar los números: del 0 al 9. Combinándolos, podemos representar cualquier número – 13, 648, 2015, etc. De la misma forma que cualquier cosa en el Universo puede ser representada por unos y ceros. Sólo se necesita un sistema para combinarlos.

Imagina cuatro interruptores que pueden estar encendidos o apagados. Cada interruptor tiene su propio valor, '8','4','2' y '1'. Cuando todos están apagados, representan '0' – 0000. Para representar el número uno, simplemente encendemos el interruptor '1' – 0001. Para representar el número dos, encendemos el interruptor del '2' – 0010. Ahora piensa cuidadosamente – ¿Cómo puedo representar el número tres? Basta con encender el interruptor del '1' y el del '2'– 0011; ya que 2 más 1 son 3.

¿Empiezas a entender el funcionamiento detrás del sistema binario? Probemos a obtener el número siete. Encendemos los interruptores '4', '2' y '1' – 0111. ¿Y para el número 11? Encendemos los interruptores '8', '2' y '1' – 1011.

Mira en la siguiente tabla para entender como funciona:

	128	64	32	16	8	4	2	1	
0	0	0	0	0	0	0	0	0	= 0
1	0	0	0	0	0	0	0	1	= 1
2	0	0	0	0	0	0	1	0	= 2
3	0	0	0	0	0	0	1	1	= 2 + 1
4	0	0	0	0	0	1	0	0	= 4
5	0	0	0	0	0	1	0	1	= 4 + 1
6	0	0	0	0	0	1	1	0	= 4 + 2
7	0	0	0	0	0	1	1	1	= 4 + 2 + 1
8	0	0	0	0	1	0	0	0	= 8
9	0	0	0	0	1	0	0	1	= 8 + 1
10	0	0	0	0	1	0	1	0	= 8 + 2
11	0	0	0	0	1	0	1	1	= 8 + 2 + 1
12	0	0	0	0	1	1	0	0	= 8 + 4
13	0	0	0	0	1	1	0	1	= 8 + 4 + 1
14	0	0	0	0	1	1	1	0	= 8 + 4 + 2
15	0	0	0	0	1	1	1	1	= 8 + 4 + 2 + 1

Blink

Durante la introducción a la programación vimos como hacer pequeños programas y animaciones usando el ordenador. Encendíamos y apagábamos los píxels en la pantalla del ordenador. Como sabes, la placa Arduino no tiene pantalla, pero tiene un LED – una pequeña lámpara que puede encenderse y apagarse fácilmente usando un programa. Se puede decir que Arduino viene con una pantalla de un solo pixel.

Ese LED en placa, está conectado al Pin digital 13. Como puedes ver en la placa, todos los pins están numerados y agrupados por funcionalidad. Hay un grupo de 14 pins (numerados del 0 al 13) que son pins digitales y luego otro grupo de 6 pins (etiquetados de A0 a A5) que son los analógicos. Más adelante explicaremos los pins analógicos, ahora nos centraremos en los digitales.

Veamos cómo controlar el LED en tu Arduino utilizando un comando sencillo. El primer ejemplo es el que llamamos Blink, que significa encender y apagar el LED repetidamente.

Al igual que los programas de Processing que siempre necesitan tener una función `setup()` y una función `draw()`, los programas Arduino necesitan las funciones `setup()` y `loop()`:

- `setup()`: Esta parte del programa sólo se ejecuta al principio. Aquí podrás configurar las funcionalidades de los pins, ya sean `inputs` (entradas) u `outputs` (salidas), o si van a realizar una función más compleja como enviar señales a través del cable USB.
- `loop()`: Esta parte funcionará infinitamente (o hasta que desconectes la fuente de alimentación). Los comandos en el `loop` serán ejecutadas en orden, una tras otra. Cuando llegamos al último comando, comenzará de nuevo desde el principio.

El programa Blink más básico funciona de la siguiente manera:

```
1  void setup() {
2  pinMode(13, OUTPUT);
3  }
4  void loop() {
5  digitalWrite(13, HIGH);
6  delay(1000);
7  digitalWrite(13, LOW);
8  delay(1000);
```

Aquí puedes ver tres comandos diferentes:

- `pinMode(pinNumber, INPUT | OUTPUT | INPUT_PULLUP)`: Se utiliza para determinar si el Pin digital en tu Arduino está escribiendo `(OUTPUT)` o leyendo `(INPUT | INPUT_PULLUP)` las señales desde/hacia el entorno.
- `digitalWrite(pinNumber, HIGH | LOW)`: Se utiliza para hacer que un Pin digital concreto escriba 5 voltios `(HIGH)` ó 0 voltios `(LOW)` a una salida.
- `delay(time)`: Detiene el programa durante cierta cantidad de tiempo. El tiempo se expresa en milisegundos. Si quieres detener el programa durante 2 segundos, deberías escribir `delay(2000)`.

Recordemos también lo que vimos durante la semana de Processing:

- Cada línea termina con punto y coma ';'.
- Los bloques de código están contenidos entre llaves '{ }'.
- Las funciones comienzan con una definición como void.
- Las funciones tienen parámetros que figuran entre paréntesis '( )', y pueden tener tantos parámetros como necesites.

Pero volviendo a lo que hace el programa, éste enciende el LED marcado con la 'L' en tu Arduino, espera 1 segundo, lo apaga y espera otro segundo. Luego esto lo repite una y otra vez hasta el infinito. Cárgalo en la placa utilizando la flecha derecha (el segundo botón de la barra de herramientas) y comprueba cómo funciona.

¡Sigue experimentando!

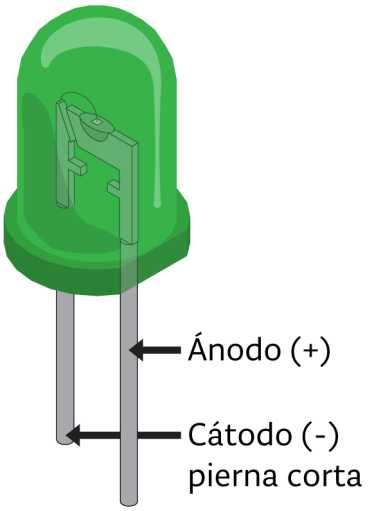
Ahora puedes probar a ver qué sucede cuando cambias los tiempos dentro de la función `delay`. ¿Qué ocurre si haces los `delays` más pequeños?, ¿la luz parpadea más rápido o más lento?

También puedes utilizar la luz para simular los latidos del corazón. Comprueba tus latidos midiendo tu pulso, notarás que siempre obtienes dos pulsaciones, algo así como tock tock... pausa... tock tock... pausa... ¿Puedes simularlo con un LED? (SUGERENCIA: puedes copiar y pegar el código).

Por último, ¿qué sucede cuando el `delay` se convierte a un intervalo verdaderamente pequeño? Digamos que lo cambias a `delay(10)`, ¿todavía puedes ver el parpadeo de la luz?

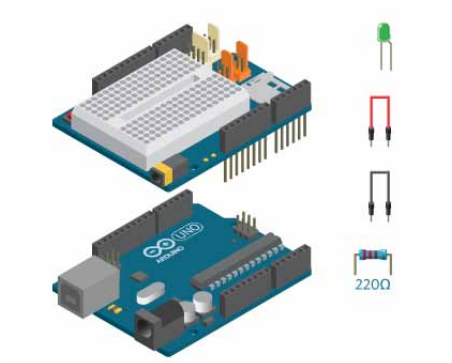
Añádele tus propios LEDs

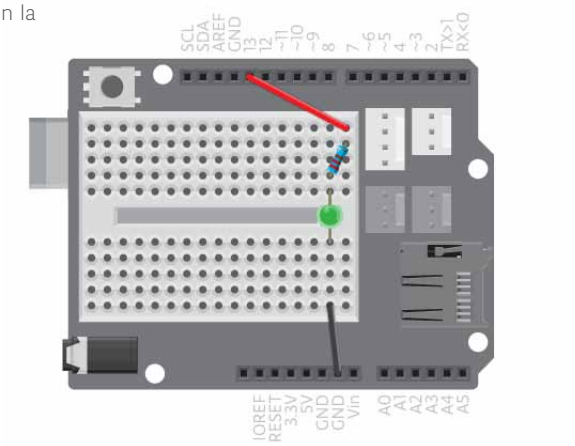
¿Quieres añadirle al circuito tus propios LED? Cuando trabajas con un LED, es importante que recuerdes que tienen polaridad. Los LED están construidos para mostrarla. El Pin largo, se denomina ánodo, y es el positivo, cuando lo conectamos al circuito, éste debe estar conectado al extremo positivo del alimentador. El Pin corto, o cátodo, debe estar conectado a tierra (0 voltios).



Los LED no se romperán si los conectas con la polaridad opuesta, simplemente no funcionarán. Sin embargo, por los LED no puede circular mucha corriente, para asegurarte de que no se quemen, debes poner una resistencia delante de ellos. El valor de la resistencia suele variar, en nuestro caso, utilizaremos 220 ohm para los LED.

Para hacer más fácil la construcción de circuitos, usaremos la Shield Básica Educativa. Una vez montada en tu Arduino, la shield amplia las capacidades de la placa, además de añadirle características extra. Para hacer el circuito con un LED, haremos uso de la breadboard. Lee más acerca de la breadboard y el Basic Education Shield en la sección de referencia.





Si haces un circuito como el que hemos mostrado anteriormente, tendrás que cambiar el programa Blink para direccionarlo al Pin número 5 en vez de al Pin número 13. El programa puede parecerse al siguiente:

```
1  void setup() {
2  pinMode(5, OUTPUT);
3  }
4
5  void loop() {
6  digitalWrite(5, HIGH);
7  delay(1000);
8  digitalWrite(5, LOW);
9  delay(1000);
10 }
```

También puedes utilizar variables que contengan el número del Pin, así será más fácil cambiar el programa en caso de que sea necesario.

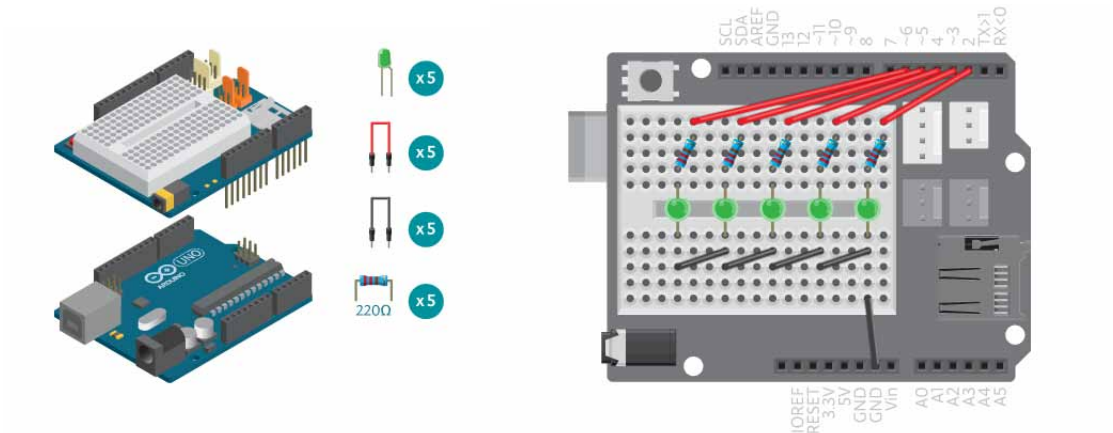
Puede que recuerdes las variables de la primera semana del curso. Sólo recordarte que las variables son lugares dentro de la memoria del ordenador donde podemos almacenar números, caracteres o incluso cadenas de texto (`strings`) completas. La variables tienen un tipo (como `int` para número y `strings` para cadenas de texto) y un nombre al que le puedes asignar lo que quieras.

En este caso, queremos que una variable almacene el número que representa el Pin que estamos cambiado con nuestro programa. Escribiendo el programa de esta forma, sólo tienes que reemplazar el número del Pin en un único lugar del código, en lugar de en tres si decidiéramos conectar el LED en otro pin de nuevo. Veamos cómo sería:

```
1  void setup() {
2  pinMode(5, OUTPUT);
3  }
4
5  void loop() {
6  digitalWrite(5, HIGH);
7  delay(1000);
8  digitalWrite(5, LOW);
9  delay(1000);
10 }
```

`ledPin` es una variable. En este caso, los valores posibles van de 0 a 13: cualquiera de los pins digitales de la placa Arduino.

Por supuesto, es posible conectar y controlar más de un solo LED a tu Arduino. En los proyectos de esta semana verás que al usar varios LEDs obtenemos lo que llamamos VU-Metro. Un VU-Metro es una linea de LEDs agrupados. Existen varias funciones que hemos preparado para que puedas controlar muchos LEDs más fácilmente. Lee más acerca del VU-Metro en la sección de referencia.



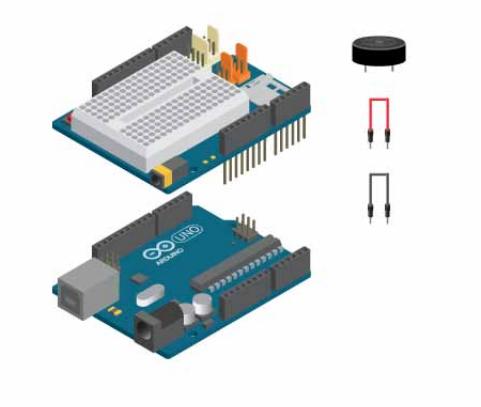
Nota: Te recomendamos que no utilices los pines 0 ni 1 a no ser que realmente los necesites. Estos pines se utilizan para que Arduino envíe información al ordenador, si les conectas cualquier cosa, Arduino no será capaz de comunicarse y por lo tanto, no será posible reprogramarlo.

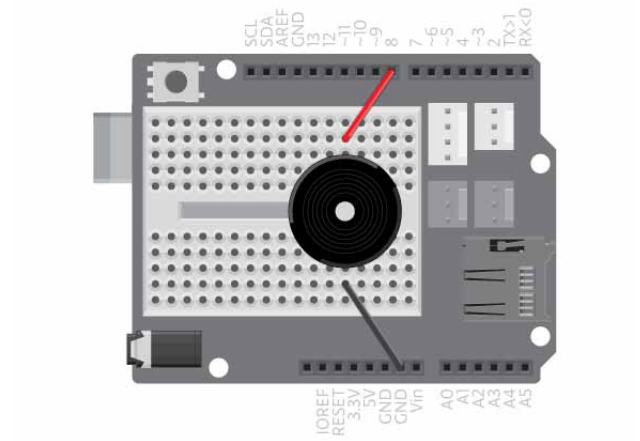
Beep

Vamos a jugar un poco con sonidos. Como probablemente ya sabrás, el sonido son vibraciones. El sonido de una guitarra por ejemplo, se produce por las vibraciones de las cuerdas. Por lo tanto, para producir sonido con Arduino necesitamos algo que genere vibraciones. En el siguiente experimento haremos esto mediante un zumbador piezoeléctrico que pite.

El Zumbador Piezoeléctrico

El micrófono de contacto, también conocido como piezo zumbador, es un componente electrónico formado a partir de la combinación de dos discos de distintos materiales. Uno de ellos es metálico y el otro, generalmente es de cerámica, y ambos tienen propiedades piezoeléctricas. Cuando se le aplica un voltaje al componente, los materiales se repelen produciendo un "click" audible (chasquido). Poniendo a cero la diferencia de tensión, hará que los materiales vuelvan a su posición inicial, produciendo de nuevo un sonido de "click".





Conecta el zumbador al pin digital 8 y a tierra. No importa cual de los pines está conectado a que pin. Para hacer vibrar al piezoeléctrico ahora, necesitamos hacer que el pin digital cambie entre HIGH y LOW repetidamente, esto es, alternar entre 5V y OV. Se parece bastante a hacer parpadear a un LED, ¿verdad?. De hecho, el programa para que el zumbador suene es muy parecido al ejemplo "Blink".

```
1   int speakerPin = 8;
2
3   void setup() {
4   pinMode(speakerPin, OUTPUT);
5   }
6
7   void loop() {
8   digitalWrite(speakerPin, HIGH);
9   delay(1000);
10  digitalWrite(speakerPin, LOW);
11  delay(1000);
12  }
```

Como has podido comprobar, la única diferencia en este programa es que usamos la variable `speakerPin` en vez de `ledPin`.

Este programa hace que el material dentro del piezoeléctrico se repela y vuelva a su posición normal en un intervalo de 2 segundos. Lo que escuchas es un "click" cada segundo. Pero para poder generar un tono, necesitamos hacer que el piezoeléctrico oscile más rápido. Podemos conseguirlo simplemente reduciendo el tiempo de retraso. Prueba cambiando el `delay` a un milisegundo:

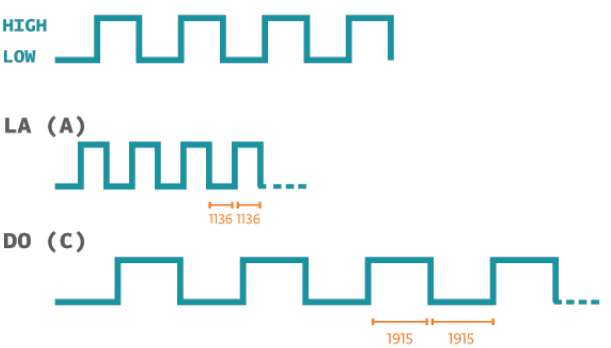
```
1  int speakerPin = 8;
2
3  void setup() {
4  pinMode(speakerPin, OUTPUT);
5  }
6
7  void loop() {
8  digitalWrite(speakerPin, HIGH);
9  delay(1);
10 digitalWrite(speakerPin, LOW);
11 delay(1);
12 }
```

Si has conectado el zumbador correctamente, deberías estar escuchando un sonido muy fuerte saliendo de tu Arduino. La única manera de detenerlo es o bien desconectarlo de la placa, o quitando el cable del pin 8.

Tonos

Ahora ya sabes como generar sonido con Arduino y un zumbador piezoeléctrico cambiando el pin entre '0' y '1' un determinado número de veces por segundo. Esta cantidad es la frecuencia, y obtendrás diferentes tonos dependiendo de su valor. A alta frecuencia, esto es, más oscilaciones por segundo, obtendrás sonidos muy agudos, mientras que a baja frecuencia, obtendrás sonidos más graves.

Los tonos tienen nombres: Do (C), Re (D), Mi (E), Fa (F), Sol (G), La (A), Si (B). De estos tonos, Do (C) tiene la frecuencia más baja a 262 Hercios, mientras que Si (B) tiene la más alta a 494 Hz. Esto significa que necesitamos hacer oscilar al pin 262 veces por segundo para que suene un Do (C) y 494 veces por segundo para que suene un Si (B).



Si quieres hacer sonar tonos diferentes, tienes que cambiar la cantidad de tiempo en que el pin está en ON y OFF. Basta con cambiar el valor de la función `delay(1)`. Sólo hay un problema: para tocar los tonos, deben ser controlados con más precisión. En vez de utilizar la función delay, deberías usar la función `delayMicroseconds`. Como recordarás, el `delay` hace una pausa del orden de milisegundos, mientras que el `delayMicroseconds` la hace en microsegundos, es decir, 1000 veces menor que un milisegundo.

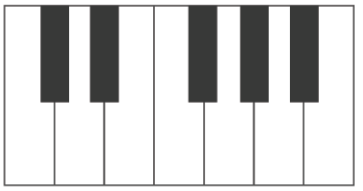
La siguiente ecuación describe la relación entre las dos funciones:

```
1   delay(1) = 1000 * delayMicroseconds(1) = delayMicroseconds(1000)
```

Esto quiere decir que el programa que has utilizado para tocar Beep, también se puede escribir como:

```
1   int speakerPin = 8;
2
3   void setup() {
4   pinMode(speakerPin, OUTPUT);
5   }
6
7   void loop() {
8   digitalWrite(speakerPin, HIGH);
9   delayMicroseconds(1000);
10  digitalWrite(speakerPin, LOW);
11  delayMicroseconds(1000);
12  }
```

Esto nos muestra un aspecto básico de la programación: ¡puedes hacer lo mismo de distintas maneras! Este programa y el anterior hacen lo mismo. Si quieres que tu zumbador piezoeléctrico toque diferentes tonos, sólo tienes que cambiar el valor '1000' por los otros valores en el delay de la tabla de abajo. Melodías.



TONO	FREQUENCIA (en hertz)	RETARDO
Do (C)	261.63	1915
Re (D)	293.66	1700
Mi (E)	329.63	1519
Fa (F)	349.23	1432
Sol (G)	392.00	1275
La (A)	440.00	1136
Si (B)	493.88	1014

Una melodía es una combinación de notas. Sin embargo, si quisiéramos tocar una melodía en un zumbador cómo hemos hecho en los ejemplos anteriores, no acabaríamos nunca. En lugar de tener que calcular el tiempo del delay dependiendo de la frecuencia, y decidir cuantas veces hacer oscilar al pin dependiendo de la duración de la nota, hemos hecho una función llamada `play()`. Puedes leer acerca de cómo usarla para tocar melodías en la sección de referencia.

Entradas digitales

Al igual que puedes asignar '0' ó '1' a un pin en Arduino, puedes leer el valor generado por un sensor que esté conectado a él. Al trabajar con pines digitales, utilizaremos sensores que solo pueden estar en ON y OFF.

Un simple cable es un buen ejemplo de un `INPUT` (entrada) digital. Puedes conectar un cable a, por ejemplo, el pin 5 y encenderlo conectando la otra parte del cable o bien a 5 voltios o a GND (0 voltios) en el conector de tu Arduino.

¡Prueba esto! Coge unos de los conectores de tu kit, conéctalo al pin 5 y programa el siguiente código en tu Arduino:

```
1  int inputPin = 5;
2  int ledPin = 13;
3
4  void setup() {
5  pinMode(inputPin, INPUT);
6  pinMode(ledPin, OUTPUT);
7  }
8
9  void loop() {
10 if (digitalRead(inputPin) == HIGH) {
11 digitalWrite(ledPin, HIGH);
12 } else {
13 digitalWrite(ledPin, LOW);
14 }
15 }
```

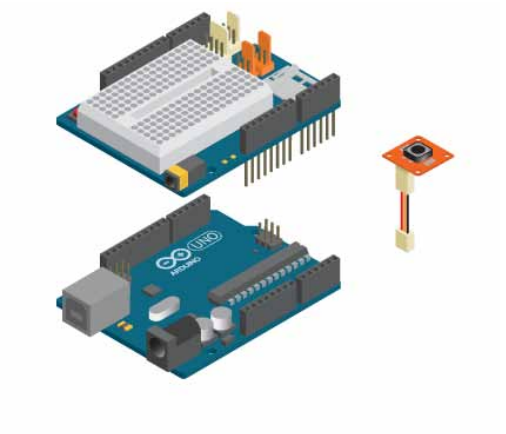
Al insertar el cable en el interior de la cabecera de 5 voltios, verás que el LED en el Pin 13 se enciende y, cuando lo insertas en la cabecera marcada como GND, el LED debería quedarse en OFF (se debería apagar).

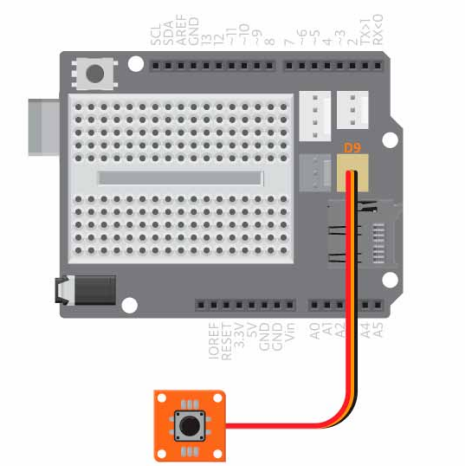
Este programa introduce algunas cosas nuevas:

- Primero, vemos que puedes utilizar muchas variables, en este caso, tenemos una para el LED `ledPin` y otra para el cable `inputPin`.
- Al igual que un `pinMode` se puede utilizar para asignar un pin como `OUTPUT`, puedes asignar el pin como INPUT.
- Hay un comando, `digitalRead(inputPin)`, que leerá el valor del voltio en el `inputPin` y devolverá si está a `HIGH` o `LOW`. Devolverá `HIGH` cuando el cable esté a 5V y `LOW` cuando se conecte a GND.
- El programa utiliza una instrucción condicional `if` para comprobar el valor del `inputPin`.
- Cuando quieres comparar dos valores en un programa, debes utilizar dos veces el signo 'igual a', es decir, `==`. Esto hace que el ordenador entienda que los quieres comparar (en lugar de asignar). Este "doble igual" es lo que llamamos un operador. Hay otros operadores que comparan otro tipo de valores.

Botón

Vamos a probar en lugar de ello con un botón. Para este ejemplo necesitarás montar la Shield Básica Educativa en Arduino y conectar un botón tinkerKit al conector D9 con un cable tinkerKit.
Puedes usar el mismo código del ejemplo anterior, pero necesitarás cambiar el valor del inputPin a 9. Esto es porque el conector D9 está de hecho, conectado al pin 9.






```
1   int inputPin = 9;
2   int ledPin = 13;
3
4   void setup() {
5   pinMode(inputPin, INPUT);
6   pinMode(ledPin, OUTPUT);
7   }
8
9   void loop() {
10  if (digitalRead(inputPin) == HIGH) {
11  digitalWrite(ledPin, HIGH);
12  } else {
13  digitalWrite(ledPin, LOW);
14  }
15  }
```

Como puedes ver, el LED se enciende cuando el botón está pulsado, y se apaga cuando lo sueltas.

¡Sigue experimentando!

- Escribe un programa que haga sonar un sonido al pulsar el botón.

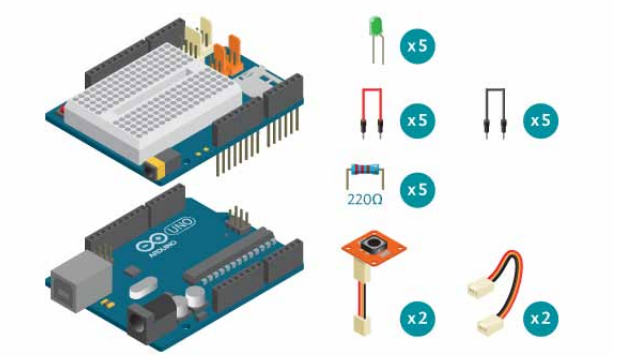
Nota: Recuerda que el conector D9 está conectado al pin digital 9. Si tienes algo a D9, no podrás usarlo para conectar cualquier otra cosa.

Pong

¡Juega a una versión simplificada del clásico juego arcade, Pong!
En este juego, un "pong" se moverá a través de cinco LEDs (VU-meter). Según los jugadores vayan pulsando el botón, rebotará de un lado a otro. Los jugadores deben pulsar el botón en el momento preciso con el fin de devolver el pong.

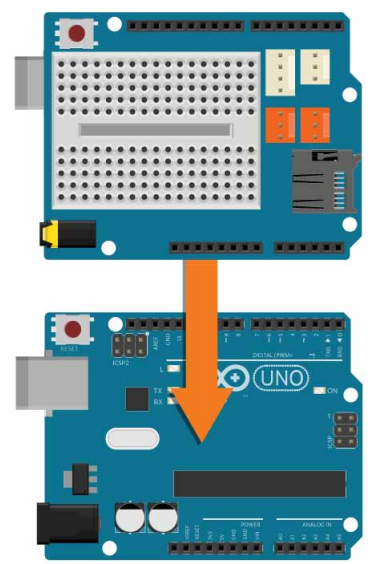
Materiales

- 1 placa Arduino
- 1 Shield Básica Educativa
- 2 botones Tinkerkit
- 2 cables Tinkerkit
- 5 LEDs
- 5 resistencias de 220 ohm
- 5 cables negros
- 5 cables de colores

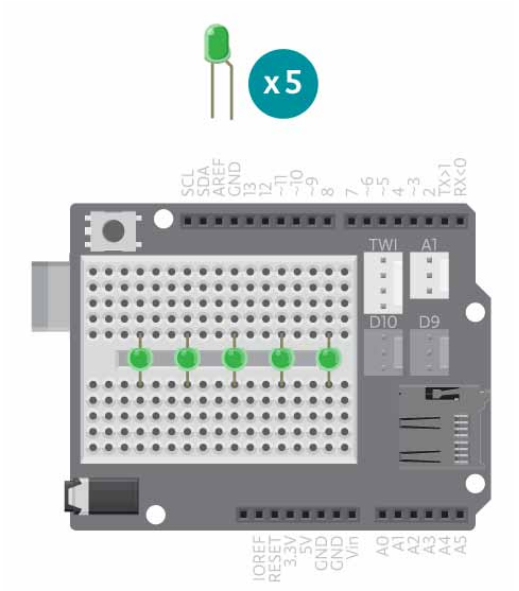


Instrucciones

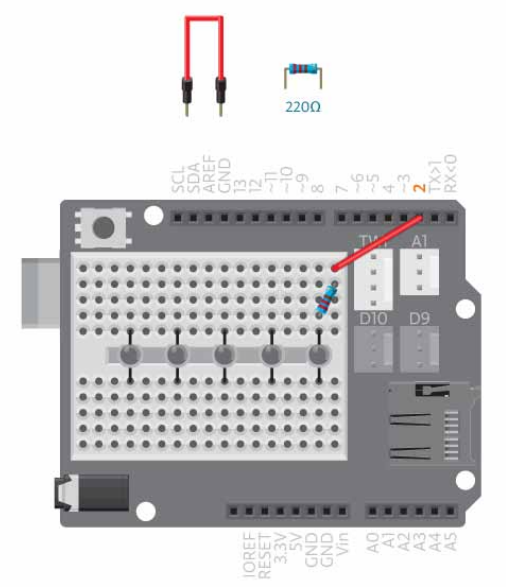
1. Coloca la shield en la parte superior de tu placa Arduino.



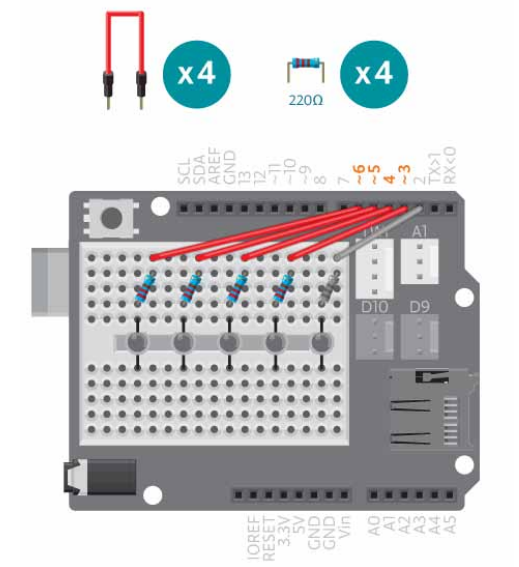
2. Conecta 5 LEDs a través del puente de la breadboard.



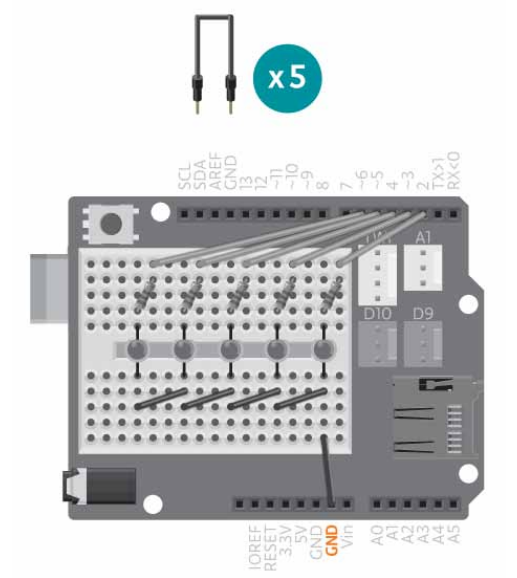
3. Conecta la resistencia de 220 ohm al Pin digital 2. Conecta la resistencia a la pata larga del primer LED (ánodo).



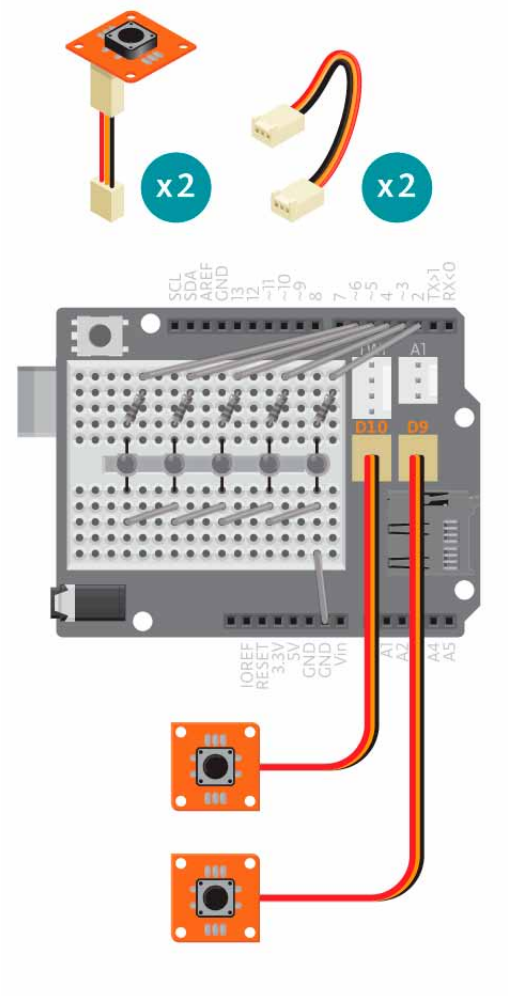
4. Conecta cada uno de los Pins digitales 3 hasta 6 a su LED correspondiente siguiendo el mismo método.



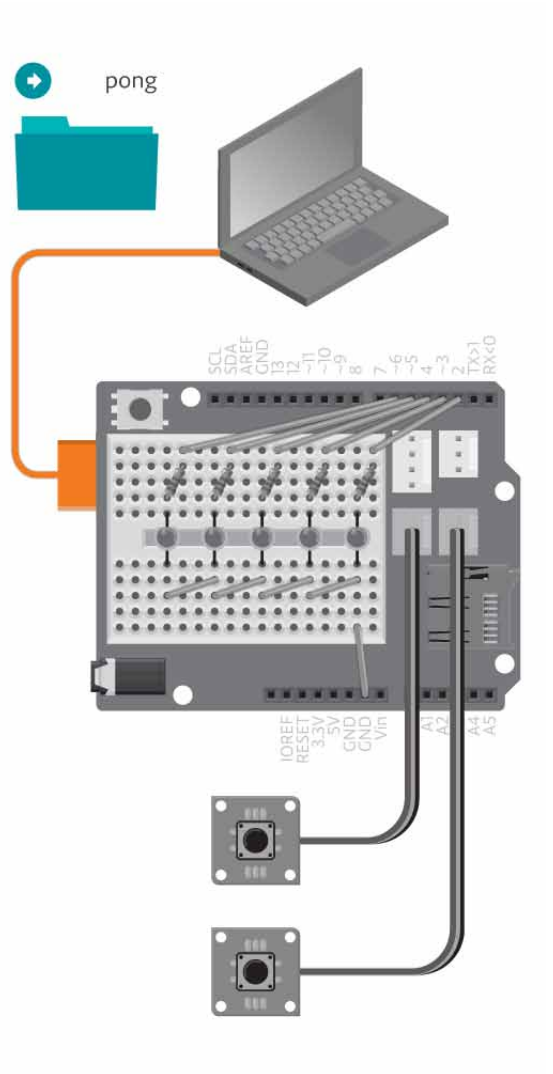
5. Conecta la pata corta de los LED a un Pin GND en Arduino utilizando los cables negros.



6. Conecta los botones Tinkerkit a los puerto D9 y D10.



7. Conecta los botones Tinkerkit a los puerto D9 y D10.
Conecta Arduino al ordenador. Carga el ejemplo Pong y prueba el juego.



Código

Puedes encontrar el código en Archivo -> Ejemplos -> BasicEducationShield-> Sports -> Pong

```
/*
  Pong

  Play a simplified version of the classic arcade game, Pong!

  In this game, a “pong” will move across five LEDs (VU-meter)
  and bounce back and forth as players press the button.
  Players must press the button at the right time in order to
  return the pong.

  (c) 2013 Arduino Verkstad
*/

#include <BasicEducationShield.h>
/*
  An array of pin numbers to which LEDs are attached
  the defaults are 2 to 6 but you can choose any of the digital
  pins. Just remember to leave digital pin 9 and 10 for the buttons.
*/
int ledPins[] = {2, 3, 4, 5, 6};
int pinCount = 5;
VUMeter vuMeter;

Button button1 = Button(9); //the button connected to digital pin 9
Button button2 = Button(10); //the button connected to digital pin 10

int ledTime = 100; //determines how fast the LEDs will switch
int pressTime = 200; //determines how long time a player has to press the button
int buttonNotPressed = 0; //this keep track on who missed to press the button

void setup(){
  //if your are using other pins than 2 to 6 you need to configure that here
  vuMeter.config(pinCount, ledPins);

  vuMeter.begin(); //does the same as pinMode, LEDs are outputs
  button1.begin(); //does the same as pinMode, buttons are inputs
  button2.begin(); //does the same as pinMode, buttons are inputs

  vuMeter.scrollLeft(ledTime, 1); //The game starts by scrolling the LEDs to the left
}

void loop(){
  /*
  if button1 is pressed within the press time, the game will continue
  by scrolling the LEDs to the right
  else if button1 is not pressed, the program will jump to gameOver()
  */
```

```
48    if(button1.released(pressTime)){
49      vuMeter.scrollRight(ledTime, 1);
50    }
51    else{
52      buttonNotPressed = 1; //Keep track on where we are in the game
53      gameOver();
54    }
55
56    /*
57    if button2 is pressed within the press time, the game will continue
58    by scrolling the LEDs to the left
59    else if button2 is not pressed, the program will jump to gameOver()
60    */
61    if(button2.released(pressTime)){
62      vuMeter.scrollLeft(ledTime, 1);
63    }
64    else{
65      buttonNotPressed = 2; //Keep track on where we are in the game
66      gameOver();
67    }
68  }
69
70  /*
71    When a player doesn't press the right button within the right
72    time it is game over. Inside the function gameOver() you can
73    decide how the LEDs should blink.
74    Use vuMeter.blink(LED,delayTime,numberOfBlinks) to make one specific LED blink
75    Use vuMeter.blinkAll(delayTime,numberOfBlinks) to make all LEDs blink
76  */
77  void gameOver(){
78    vuMeter.blinkAll(100,10);
79
80      if(buttonNotPressed==1) vuMeter.scrollRight(ledTime, 1); //if button1 was not
pressed, scroll LEDs to right to start over
81     else if(buttonNotPressed==2) vuMeter.scrollLeft(ledTime, 1); //if button2 was not
pressed, scroll LEDs to left to start over
82  }
```

Cómo funciona

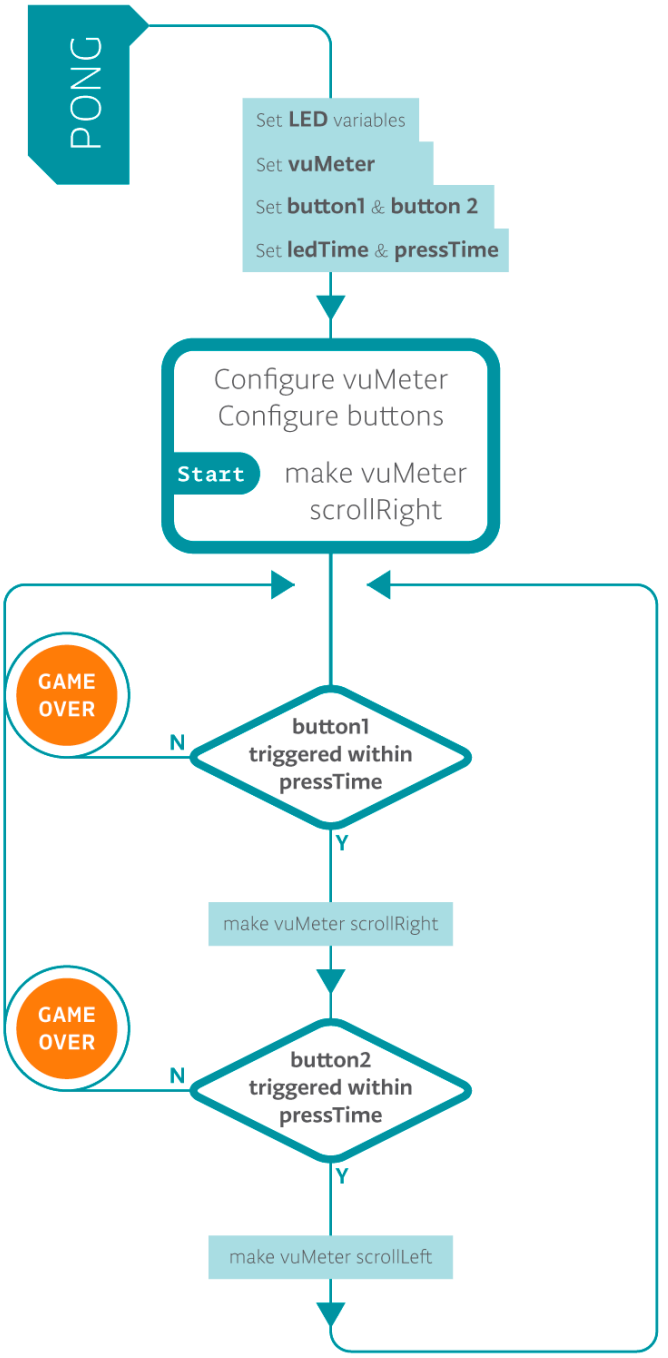
El juego comienza con el desplazamiento de los LEDs a la izquierda. Cuando el último LED se apaga, el botón "uno" debe ser pulsado dentro del tiempo establecido. Si lo haces, el LED se desplazará de nuevo a la derecha. Esta vez, cuando el último LED esté apagado, tienes que pulsarse el botón "dos" dentro del tiempo de pulsación. Si así lo has hecho, la función `loop()` se ejecutará de nuevo. Siempre que un jugador tarde demasiado tiempo en pulsar el botón, el juego finalizará. La función `(gameOver)` se activará y todos los LEDs parpadearán. Inmediatamente los LEDs se desplazarán a la izquierda y el juego comenzará de nuevo.

¿No funciona?

1. Consulta la ilustración y vuelve a comprobar las conexiones. Asegúrate de que el shield y los cables estén firmemente conectados.

2. ¿No puedes golpear el pong? Intenta intercambiar los botones, si sigue sin funcionar, mira en la referencia para depurar botones.

3. ¿El VU-meter no funciona correctamente? Mira en la referencia para depurar el VU-metro.



¡Sigue experimentando!

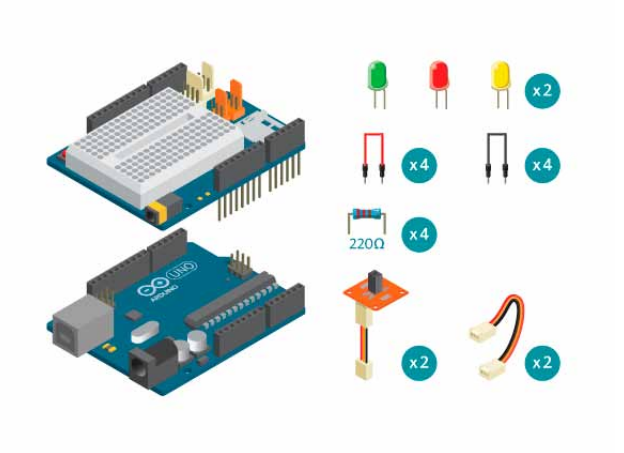
- Intenta cambiar el parpadeo de los LEDs cuando finaliza el juego. Mira el código de ejemplo VUMeterTest para ver los comandos que puedes utilizar.
- Haz una caja para la placa. Utiliza un material que permita que los LEDs brillen a través de él. Prueba también a hacer unas cajas para los botones.
- Puedes intentar cambiar los Pins digitales a los que están conectados los LEDs y ver qué pasa. ¡No te olvides de cambiarlos tanto en las conexiones como en el código! Y no te olvides que los Pins D9 y D10 ya están siendo utilizados para los botones.

Esgrima

¡Comprueba tu tiempo de reacción retando a un oponente!
En este juego, dos jugadores sujetarán sus espadas con sensores tilt. El primero que gire su espada cuando el LED verde se encienda aleatoriamente, gana.

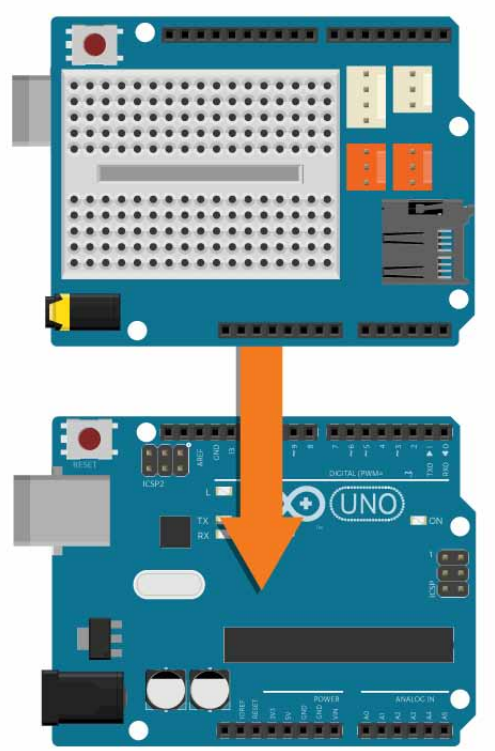
Materiales

- 1 placa Arduino
- 1 Shield Básica Educativa
- 2 interruptores Tilt Tinkerkit
- 2 cables largos Tinkerkit
- 1 LED grande rojo
- 1 LED grande verde
- 2 LEDs amarillos
- 4 resistencias de 220 ohm
- 4 cables negros
- 4 cables de colores

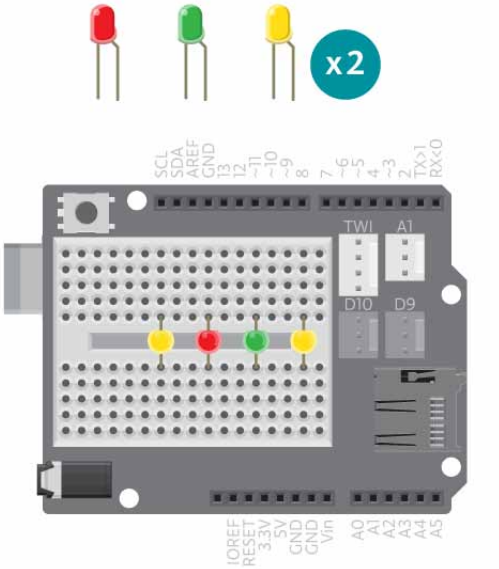


Instrucciones

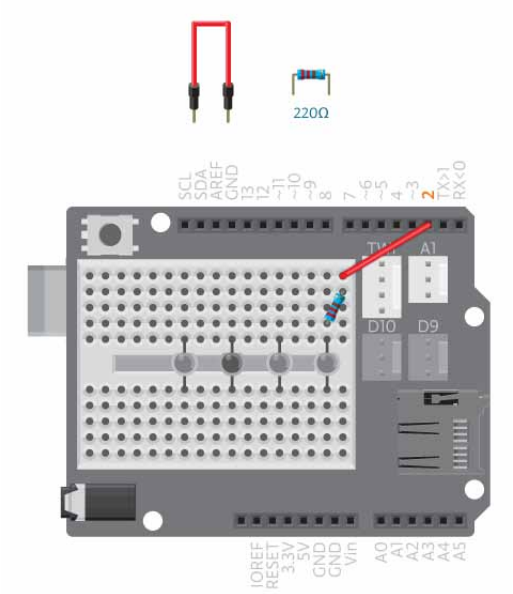
1. Conecta la shield a la parte superior de tu placa Arduino.



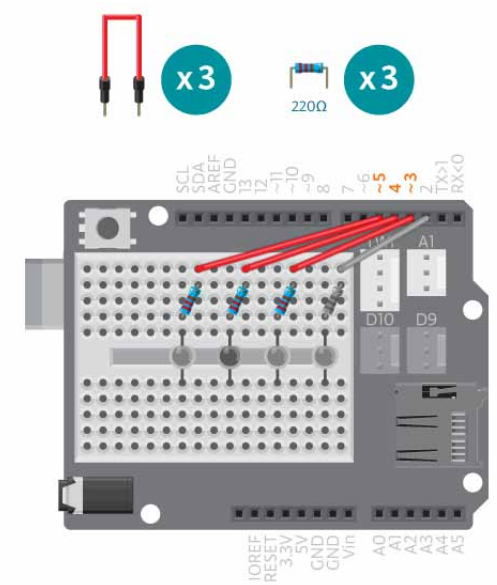
2. Conecta cuatro LEDs a través del puente de la breadboard en el siguiente orden: amarillo, rojo, verde, amarillo.



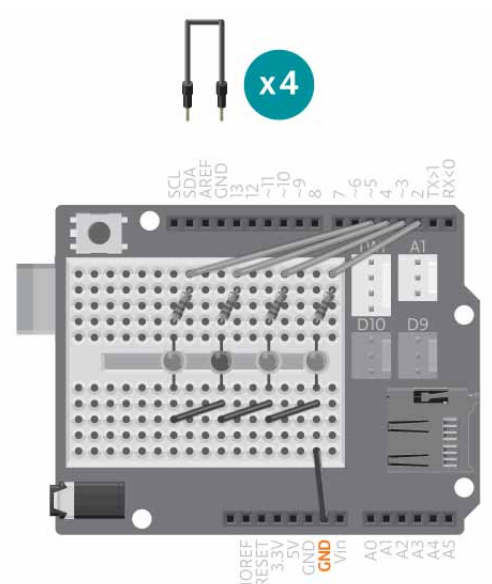
3. Conecta una resistencia de 220 ohm al Pin digital 2. Conecta la resistencia a la pata larga del primer LED.



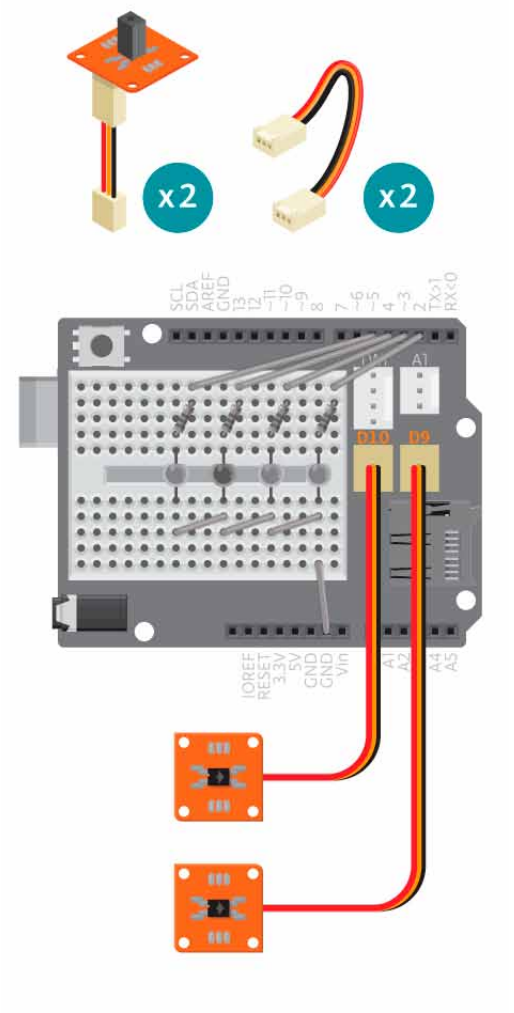
4. Conecta cada uno de los Pins digitales del 3 al 5 a su LED correspondiente siguiendo el mismo método.



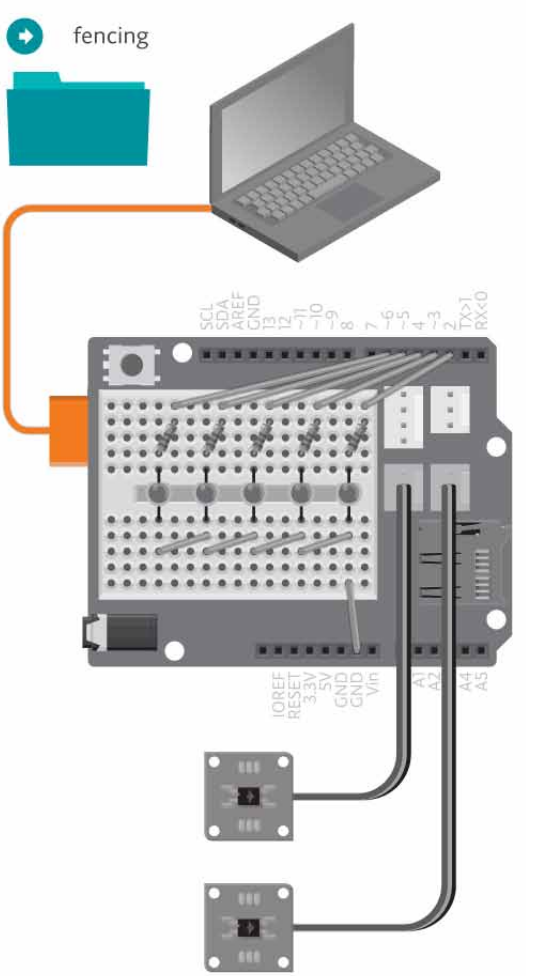
5. Conecta la pata corta de los LEDs a un Pin de Arduino GND utilizando los cables negros.



6. Conecta los módulos TinkerKit Tilt a los pines D9 y D10.



7. Conecta Arduino al ordenador. Carga el ejemplo "fencing" y prueba el juego.



Código

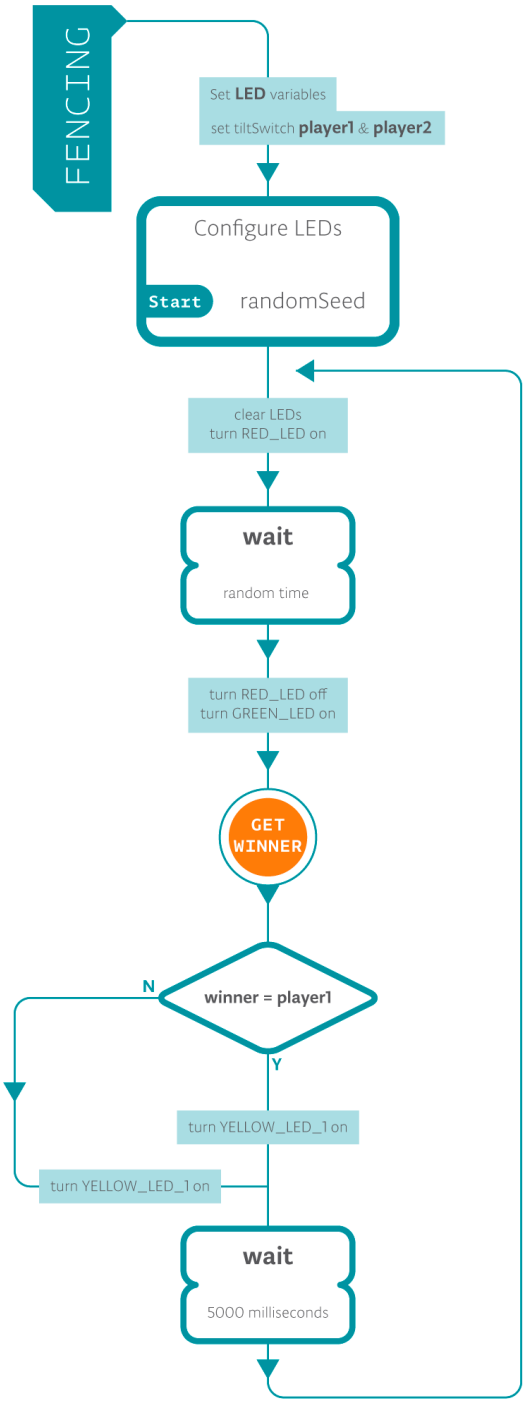
Puedes encontrar el código en Archivo -> Ejemplos -> BasicEducationShield-> Sports -> Fencing

```
/*
  Fencing
  Test your reaction time against an opponent!

  In this game, two players will hold tilt switch swords.
  When the green LED randomly lights up, the first person
  to swing their sword wins.
  (c) 2013 Aduino Verkstad
*/

#include <BasicEducationShield.h>

//Position of the leds in VU-meter is represented
//by their names here. So we can use names to find LEDs later
#define YELLOW_LED_1 0
#define GREEN_LED 1
#define RED_LED 2
#define YELLOW_LED_2 3

//An array stores which pins the VU-meter is connected
int ledPins[]={2,3,4,5};
//How many pins are used by VU-meter
int pinCount=4;
VUMeter lights;

TiltSwitch player_1 = TiltSwitch(9);
TiltSwitch player_2 = TiltSwitch(10);

void setup(){
  lights.config(pinCount,ledPins);

  //Initializing components
  lights.begin();
  player_1.begin();
  player_2.begin();

  //We need this for generating random number later
  randomSeed(analogRead(0));
}
void loop(){
 lights.clear();

 //Red led means both of you should hold the tilt switch sword up right
 lights.on(RED_LED);

 //Wait for a random period of time, between 3 seconds
 //And 6 seconds. Get ready!
```

```
48   delay(random(3000,6000));
49
50   lights.off(RED_LED);
51   //When the green led turns on, game starts
52   lights.on(GREEN_LED);
53
54   //Swing your swords as fast as you can, the faster one
55   //will be returned by getWinner()
56   int winner=getWinner();
57
58   //The yellow led by side of the winner will light up
59   if(winner==1){
60     lights.on(YELLOW_LED_1);
61   }else{
62     lights.on(YELLOW_LED_2);
63   }
64   delay(5000);
65 }
66
67 //The function below waits for either of the tilter
68 //switch to be swang. The first one to swing
69 //will be returned by its number
70 int getWinner(){
71    do{
72      if(player_1.pressed(1)){
73        return 1;
74      }else if(player_2.pressed(1)){
75        return 2;
76      }
77    }while(true);
78 }
```

Cómo funciona

Primero se ilumina el LED rojo indicando que ambos deben sujetar sus espadas y esperar. Hay un tiempo de espera aleatorio antes que el LED rojo se apague y el verde se encienda. Luego el juego espera a que una de las espadas ataque, y el primero en hacerlo será el ganador. El LED amarillo que esté en el lado del ganador se iluminará unos segundos y comienza el juego de nuevo.



¿No funciona?

1. Revisa la ilustración y vuelve a comprobar las conexiones. Asegúrate de que el shield y los cables estén firmemente conectados.

2. ¿Has sido el primero en darle pero ha ganado tu amigo? Intenta intercambiar vuestras espadas.

3. ¿Sigue sin pasar nada? Asegúrate de que sigues las reglas y mira la referencia para depurar interruptores tilt.

4. ¿El VU–meter no funciona correctamente? Mira la referencia para depurar el VU-meter.

¡Sigue experimentando!

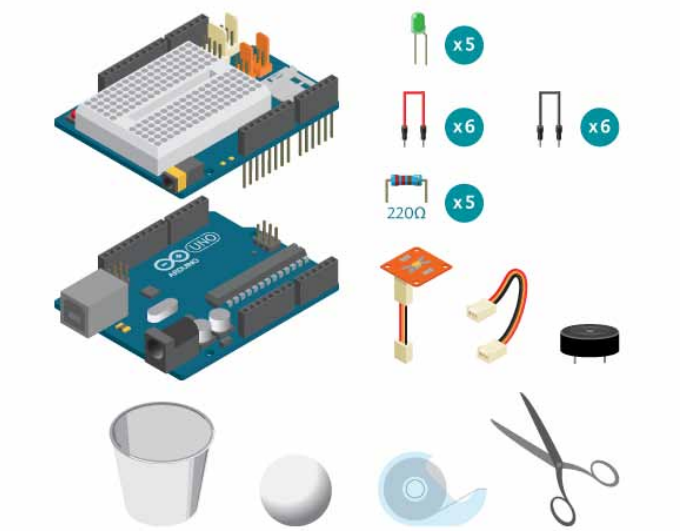
- Crea un par de espadas para ponerles los interruptores tilt dentro.
- Cambia las reglas del juego. ¿Puedes modificarlo de forma que la persona que agite la espada arriba y abajo cinco veces sea el ganador?

Baloncesto

¡Mete una canasta!
En este juego, los jugadores tratarán de hacer botar una pelota de ping pong y meterla en una taza. Para ganar, mete cinco puntos. Las anotaciones se registran utilizando un LDR, Light Dependent Resistor (resistencia dependiente de luz).

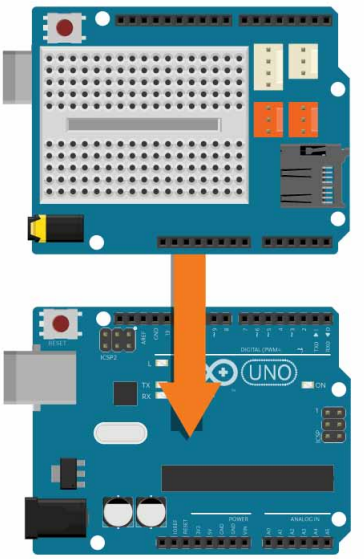
Materiales

- 1 placa Arduino
- 1 Shield Básica Educativa
- 1 Tinkerkit LDR
- 1 cable Tinkerkit
- 1 altavoz o piezoeléctrico
- 5 LEDs
- 5 resistencias 220 ohm
- 6 cables negros
- 6 cables de colores
- 1 taza de plástico
- 1 pelota de ping pong
- cinta adhesiva
- tijeras/cuchillo

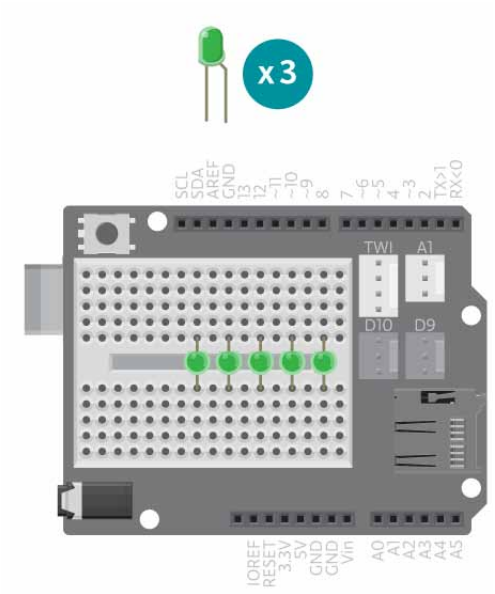


Instrucciones

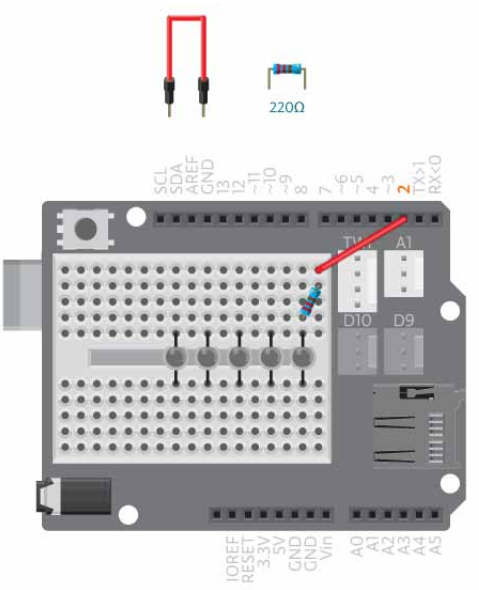
1. Conecta la shield a la parte superior de tu placa Arduino.



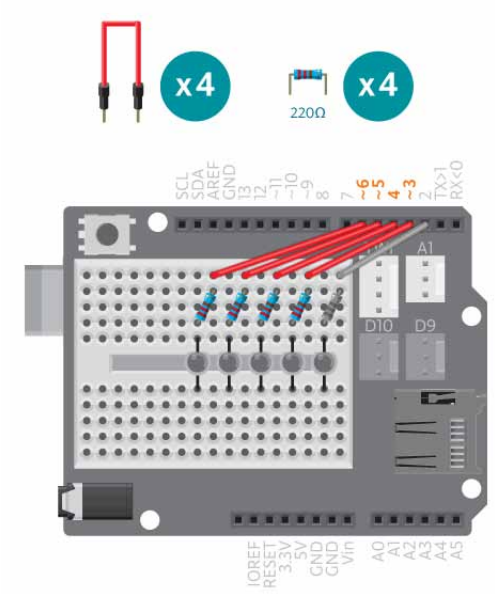
2. Conecta cinco LEDs a través del puente de la breadboard.



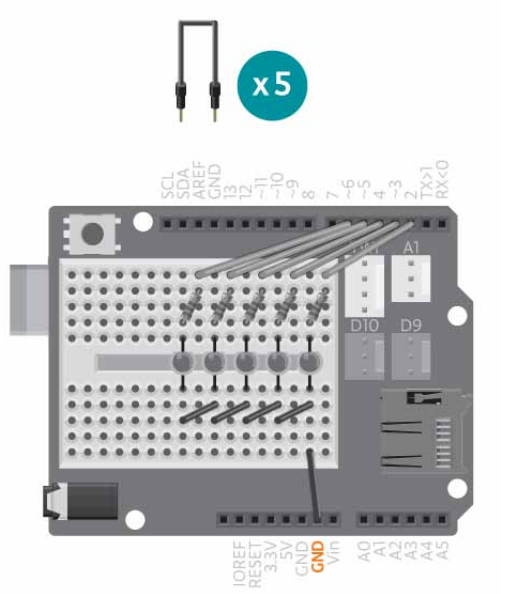
3. Conecta la resistencia de 220 ohm al Pin digital 2. Conecta la resistencia a la pata larga del primer LED.



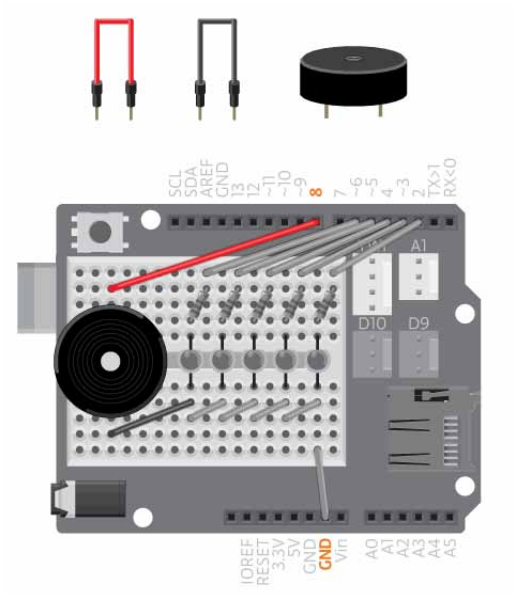
4. Conecta cada Pin digital de 3 a 6 a su LED correspondiente siguiendo el mismo método.



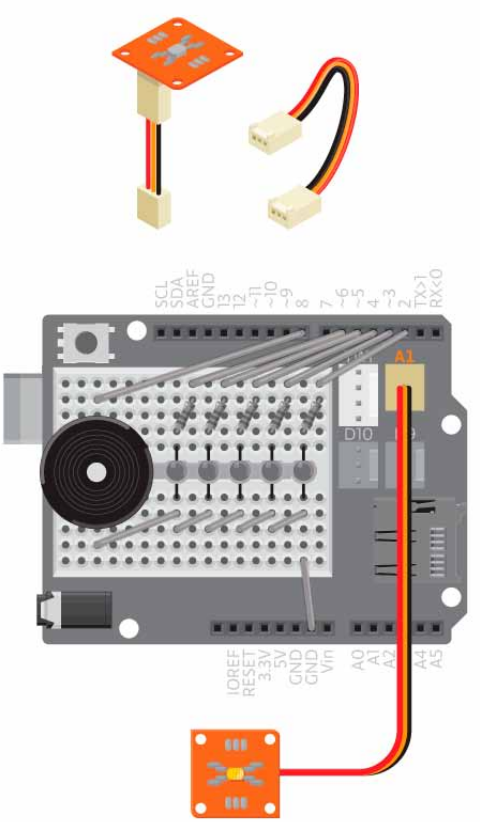
5. Conecta la pata corta de los LEDs a un Pin GND de Arduino utilizando los cables negros.



6. Conecta con un cable rojo el zumbador al Pin digital 8 y con uno negro a GND.



7. Conecta el módulo TinkerKit LDR al conector A1 de tres puertos.

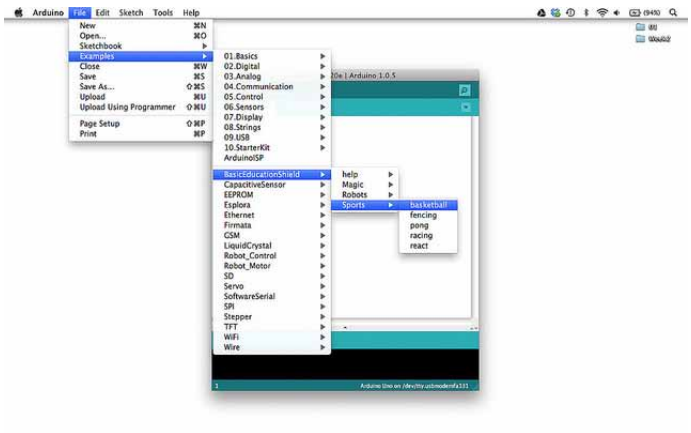


8. Carga el ejemplo LDRtest para probar las condiciones de iluminación. Abre el monitor serial y sujeta la LDR verticalmente. Toma nota del valor al descubierto (uncovered).

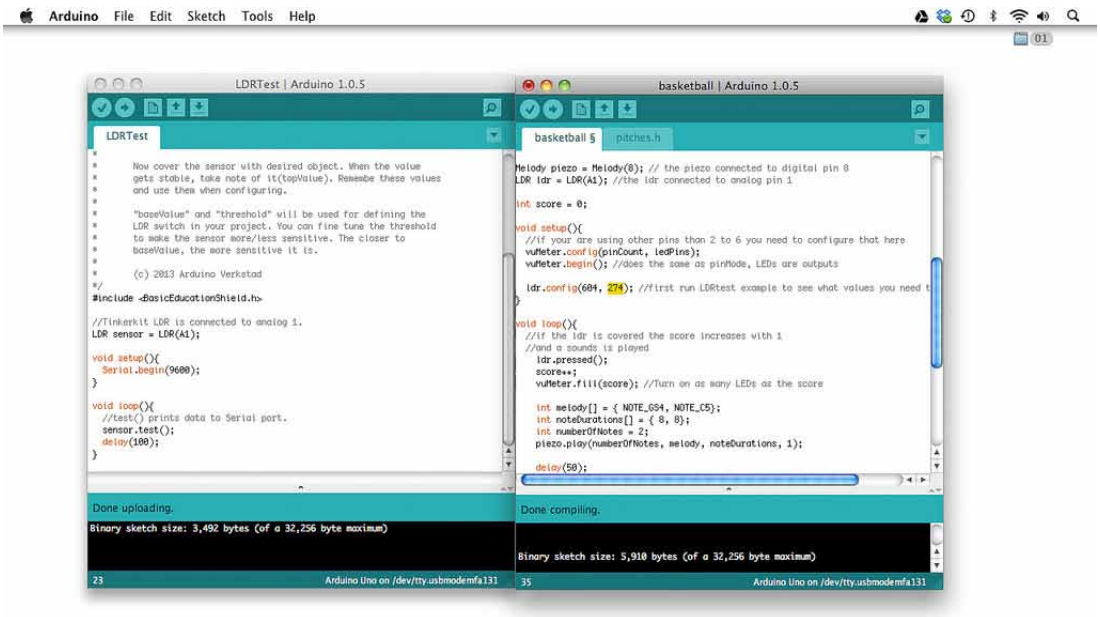
9. Carga el ejemplo LDRtest para probar las condiciones de iluminación. Abre el monitor serial y sujeta la LDR verticalmente. Toma nota del valor al descubierto (`uncovered`).

Coloca una pelota de ping-pong encima de la LDR. Anota el valor al cubierto (`covered`). Haz el promedio de los dos números, lo que te dará es el valor umbral `threshold`.

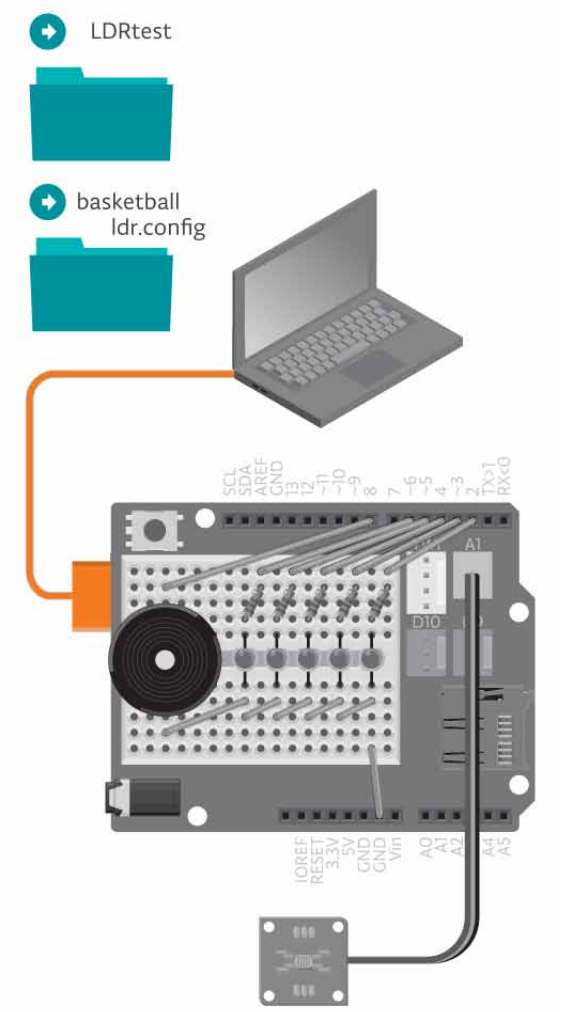
10. Abre el ejemplo Basketball.



11. Busca la línea `ldr.config (# # # # # #)`, y cambia los parámetros `# # #` al valor al descubierto (`uncovered`) y el valor umbral (`threshold`), respectivamente.



12. Carga Basketball y prueba el juego.



13. Corta un agujero en el fondo del vaso de plástico para que se ajuste la LDR.

14. Coloca la LDR en el agujero, y pégalo (no cubras la LDR). Pega el vaso de plástico a tu mesa, para que se mantenga de pié.

Código

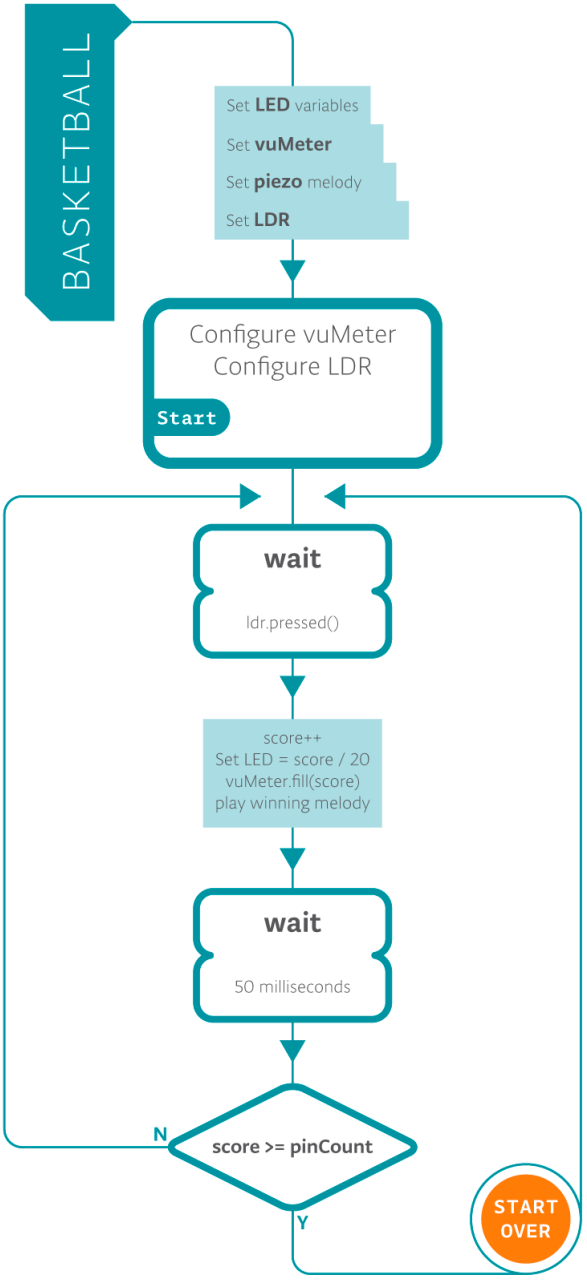
Puedes encontrar el código en Archivo -> Ejemplos -> BasicEducationShield-> Sports -> Basketball

```
/*
  Basketball

  Score a goal!

  In this game, players will try to bounce a ping pong ball
  into a cup. Make five points to win. The score is tracked
  using a light dependent resistor (LDR).

  (c) 2013 Arduino Verkstad

*/


#include <BasicEducationShield.h>
#include “pitches.h”
/*
An array of pin numbers to which LEDs are attached
the defaults are 2 to 6 but you can choose any of the digital pins
*/
int ledPins[] = {2, 3, 4, 5, 6};
int pinCount = 5;
VUMeter vuMeter;

Melody piezo = Melody(8); // the piezo connected to digital pin 8
LDR ldr = LDR(A1); //the ldr connected to analog pin 1

int score = 0;

void setup(){
  //if your are using other pins than 2 to 6 you need to configure that here
  vuMeter.config(pinCount, ledPins);
  vuMeter.begin(); //does the same as pinMode, LEDs are outputs

  ldr.config(800, 600); //first run LDRtest example to see what values you need to put
here
}

void loop(){
  //if the ldr is covered the score increases with 1
  //and a sounds is played
    ldr.pressed();
    score++;
    vuMeter.fill(score); //Turn on as many LEDs as the score

    int melody[] = { NOTE_GS4, NOTE_C5};
    int noteDurations[] = { 8, 8};
```

```
47      int numberOfNotes = 2;
48      piezo.play(numberOfNotes, melody, noteDurations, 1);
49
50      delay(50);
51
52    if(score>=pinCount) startOver(); //If the score equals the amount of LEDs you start
over
53  }
54
55
56  void startOver(){
57    score=0; //reset the score
58
59    int melody[] = { NOTE_C5, NOTE_G4,NOTE_G4, NOTE_A4, NOTE_G4, 0, NOTE_B4, NOTE_C5};
60    int noteDurations[] = { 4, 8, 8, 4,4,4,4,4 };
61    int numberOfNotes = 8;
62    piezo.play(numberOfNotes, melody, noteDurations, 1);
63
64    vuMeter.blinkAll(50,10);
65
66  }
```

Cómo funciona

La variable `score` irá contando en tu marcador. Cada vez que cubres la LDR, esta detecta un valor superior al umbral y la puntuación se incrementará con 1. Según la puntuación, se irán encendiendo los LEDs que haya en el VU-meter. Se reproducirá una pequeña melodía. Al final de la función `loop()` el programa verifica si la puntuación es mayor o igual a la cantidad de LEDs conectados a la placa. Si lo es, el programa salta otra vez a startOver(). Allí, el marcador se pone a cero otra vez, sonará una melodía de victoria y todos los LEDs parpadearán. Después de esto, el programa comienza de nuevo en `loop()`.



¿No funciona?

1. Revisa la ilustración y asegúrate de que están bien hechas todas las conexiones.

2. ¿No anota tus puntos? Intenta cambiar el threshold más cerca de baseValue. Mira la referencia del sensor LDR para corregir errores.

3. ¿El VU-meter no funciona correctamente? Mira la referencia VU-meter para corregir posibles errores.

¡Sigue experimentando!

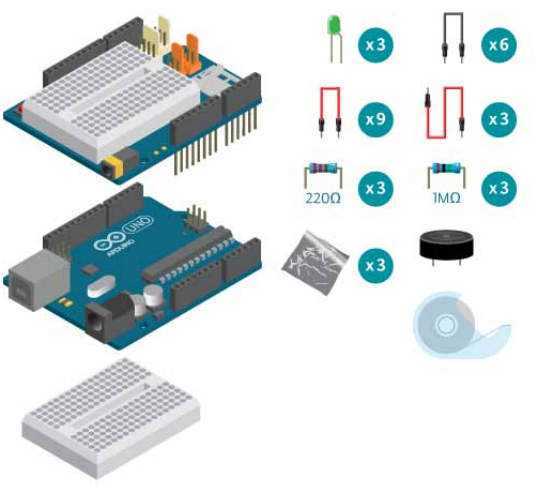
- ¿Puedes añadir más LEDs y aumentar la puntuación? ¡No olvides añadirlos tanto físicamente a la placa como al código del programa!
- ¿No te acaban de gustar los sonidos que emite el piezo? Prueba a hacer tus propias melodías.
- Intenta cambiar la manera en que parpadean los LEDs cuando finaliza el juego. Mira el código de ejemplo VUMeterTest para ver los comandos que puedes utilizar.

Reacciona

¡Pon a prueba tu capacidad de reacción! En este juego, uno de los tres LEDs se iluminarán aleatoriamente. Tienes que golpear el sensor capacitivo correspondiente dentro del tiempo de reacción. Si no lo consigues, el juego finaliza.

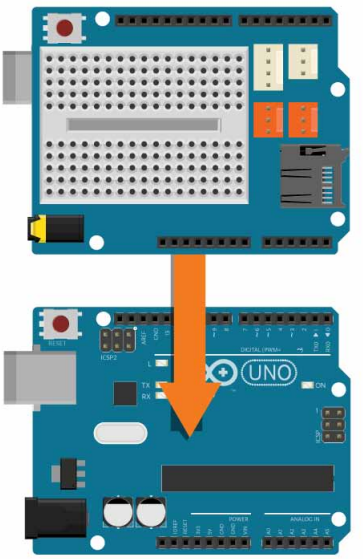
Materiales

- 1 placa Arduino
- 1 Shield Básica Educativa
- 1 altavoz piezo
- 3 LEDs
- 3 resistencias de 220 ohm
- 3 resistencias de 1Mohm
- 6 cables negros
- 12 cables de diferentes colores (3 largos)
- papel de aluminio
- cinta adhesiva
- 1 breadboard

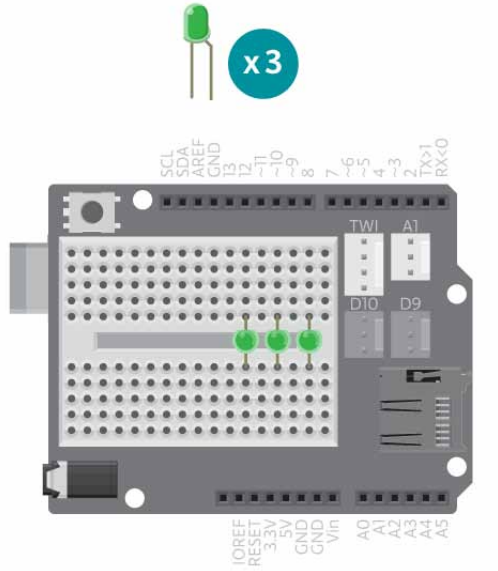


Instrucciones

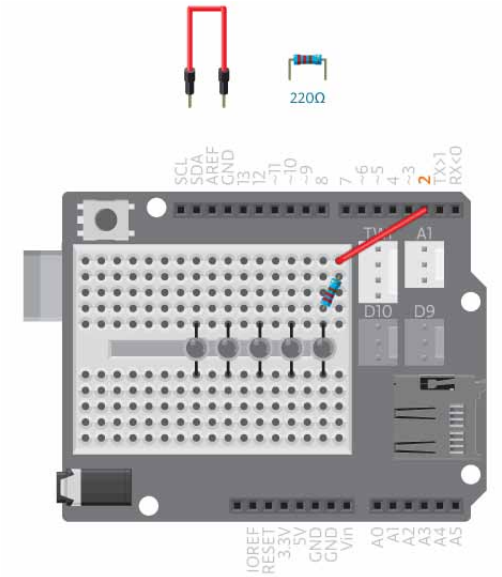
1. Conecta la shield a la parte superior de tu placa Arduino.



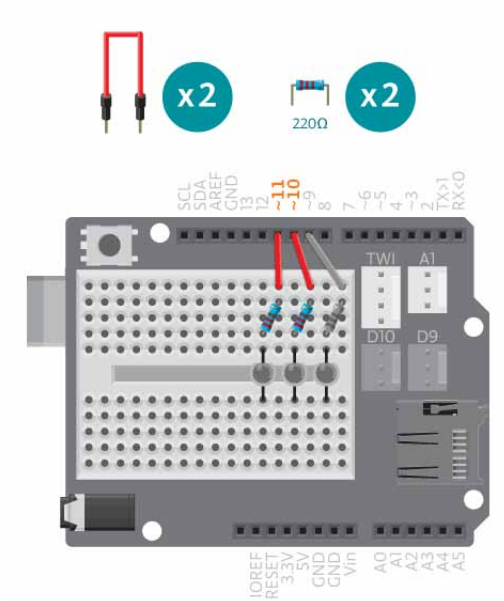
2. Conecta los tres LEDs a través del puente de la breadboard (revisa los Pins).



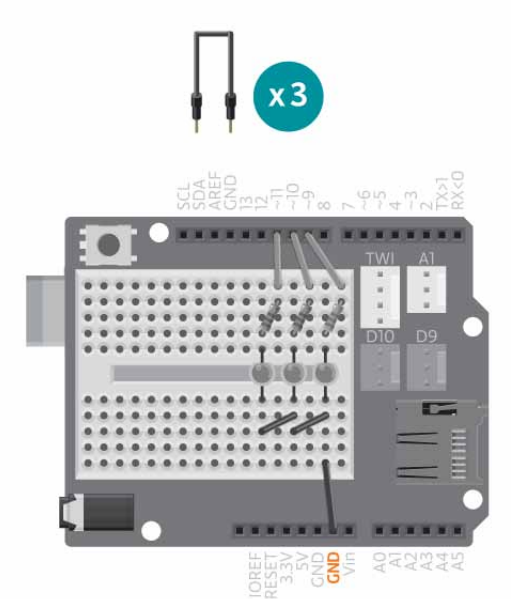
3. Conecta una resistencia de 220 ohm al Pin digital 9. Conecta la resistencia a la pata larga del primer LED.



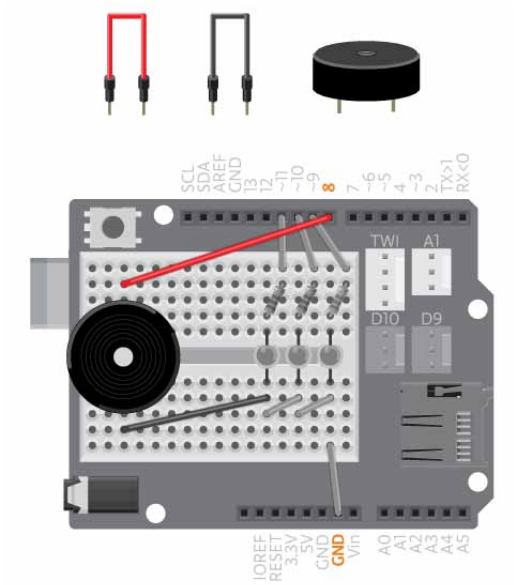
4. Conecta cada pin digital del 10 al 11 a su LED correspondiente siguiendo el mismo método.



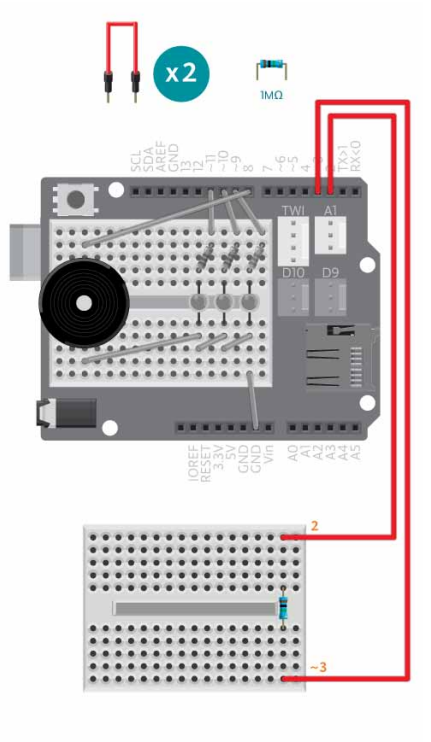
5. Conecta la pata corta de los LEDs a un Pin de Arduino GND utilizando cables negros.



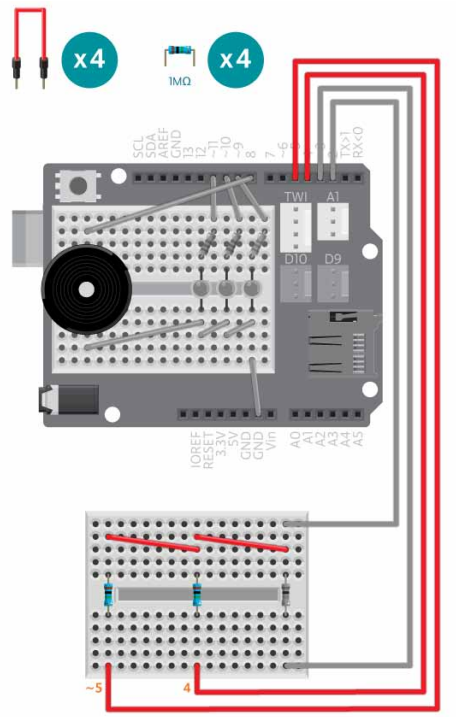
6. Conecta el cable rojo del altavoz piezo al Pin digital 8 y su cable negro a GND.



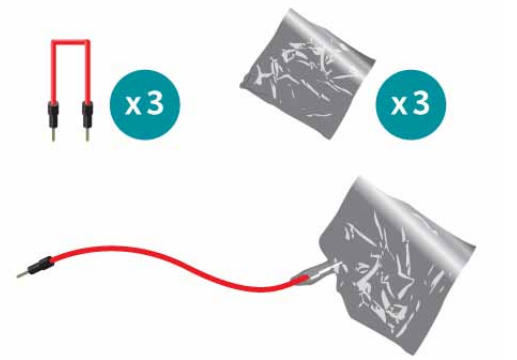
7. Conecta una resistencia de 1MOhm entre los pines digitales 3 y 2, en la breadboard.



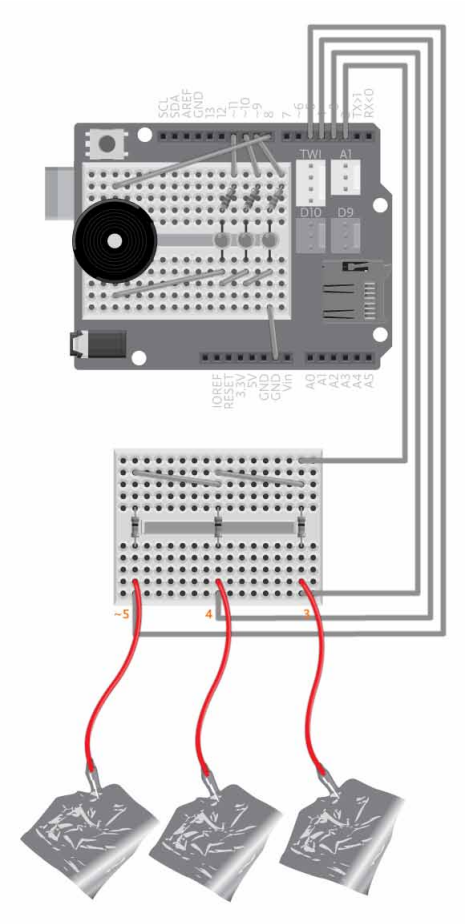
8. Conecta los Pins digitales 4 y 2 y los Pins digitales 5 y 2 siguiendo el mismo método.



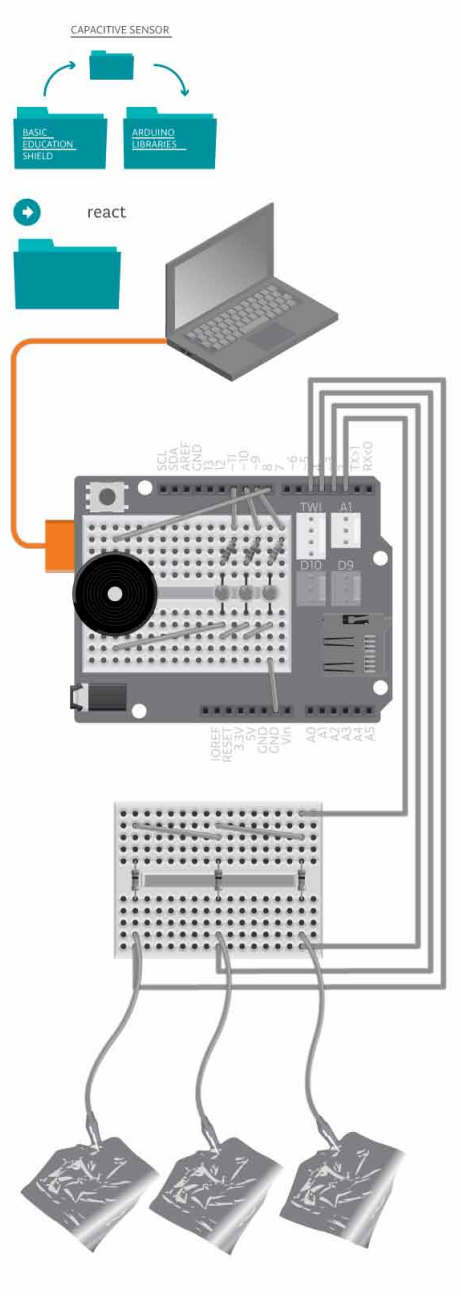
9. Haz un sensor de contacto cortando un cuadrado de papel de aluminio de 5 cm x 5 cm. Envuelve una esquina de la lámina cuadrada a un cable suelto, el cable de metal debe estar en contacto con el papel de aluminio.



10. Haz tres sensores de contacto, conecta cada pin digital del 3 al 5. Pega los tres sensores firmemente a la mesa, asegurándote de que no se toquen.



11. Conecta Arduino al ordenador. Mueve la carpeta CapacitiveSensor de la carpeta BasicEducationShield a la librería de la carpeta Arduino. Carga el ejemplo React y prueba el juego.



Código

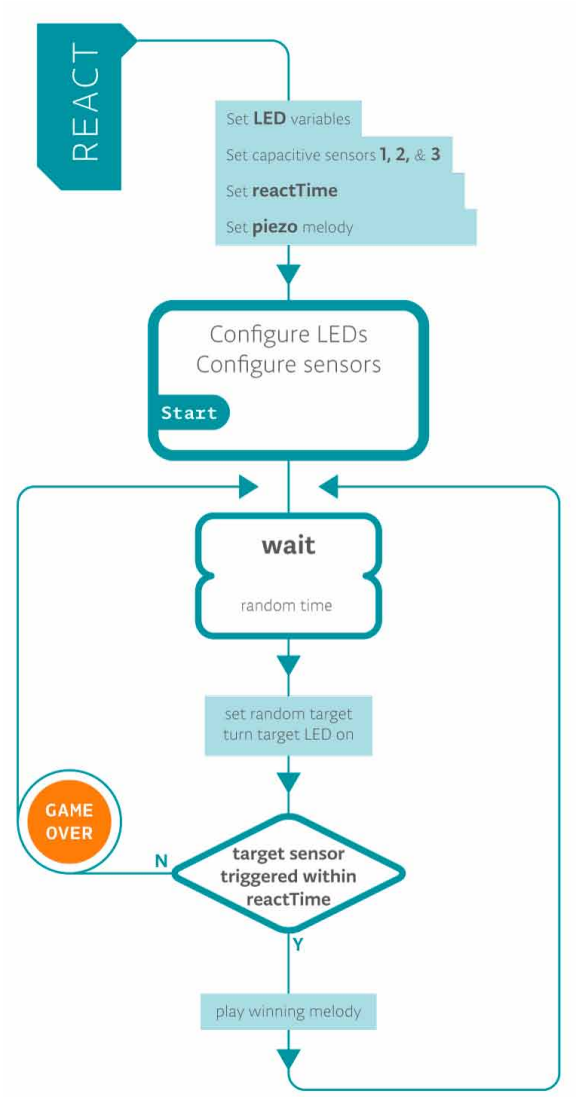
Puedes encontrar el código en Archivo -> Ejemplos -> BasicEducationShield-> Sports -> React

```
/*
  React
  Test your reaction time!

  In this game, one of three LEDs will randomly light up.
  You must tap the corresponding capacitive sensor as quick
  as possible. If you don't react fast enough, the game is over.

  (c) 2013 Arduino Verkstad
*/

#include <CapacitiveSensor.h>
#include <BasicEducationShield.h>
#include "pitches.h"

//Define the 3 LEDs
int ledPins[] = {9, 10, 11};
int pinCount = 3;
VUMeter LEDs;

//There're 3 pads for pressing
CapacitiveSwitch pad[3];

//You have 500 milliseconds to press the pad
int reactTime = 500;

// the piezo connected to digital pin 8
Melody piezo = Melody(8);

void setup(){
  LEDs.config(pinCount, ledPins);
  LEDs.begin();

  //Configure the pads
  pad[0] = CapacitiveSwitch(2,3);
  pad[1] = CapacitiveSwitch(2,4);
  pad[2] = CapacitiveSwitch(2,5);
  pad[0].config(100);
  pad[1].config(100);
  pad[2].config(100);
}

void loop(){
  //Wait for a random time before each turn begins
  delay(random(50, 2000));

  //pick a target between the 3 pads
```

```
48    int target = random(0, 3);
49    //Light up the LED
50    LEDs.on(target);
51
52    //If the pad corresponding to the LED is pressed
53    if(pad[target].pressed(reactTime)){
54      LEDs.off(target);
55
56      //Play the winning sound
57      int melody[] = { NOTE_GS4, NOTE_C5};
58      int noteDurations[] = { 8, 8};
59      int numberOfNotes = 2;
60      piezo.play(numberOfNotes, melody, noteDurations, 1);
61    }
62    else{
63      //Else if the reaction is too slow, run the function gameOver()
64      gameOver();
65    }
66  }
67
68  void gameOver(){
69    //Turn all LEDs on
70    LEDs.fill(pinCount);
71
72    //Play a melody
73    int melody[] = { NOTE_E2, NOTE_C2};
74    int noteDurations[] = { 2, 1};
75    int numberOfNotes = 2;
76    piezo.play(numberOfNotes, melody, noteDurations, 1);
77
78    LEDs.blinkAll(100, 10);
79    LEDs.fill(0); //Tun all LEDs off
80  }
```

Cómo funciona

Primero el juego espera durante un período de tiempo aleatorio, antes de elegir un número entre 0 y 2. Este número representa el LED que se encenderá y el sensor asociado a él. El LED se enciende, y el sensor correspondiente espera a ser pulsado. Si tienes la habilidad de pulsarlo dentro del `reactTime`, un sonido de victoria sonará . Entonces el proceso comenzará de nuevo. Sin embargo si fallas, la función `gameOver()` se activará, todos los LEDs se encenderán y sonará el sonido de fin de juego. Después, el juego se reiniciará.



¿No funciona?

1. Consulta la ilustración y vuelve a comprobar las conexiones. Asegúrate de que el shield y los cables estén firmemente conectados.

2. Aumenta el tiempo permitido de reacción para hacer el juego más fácil y probar que funcione. ¡Asegúrate de volver a cambiarlos cuando todo esté funcionando!

3. ¿El VU-meter no funciona correctamente? Mira la referencia para corregir el VU-Meter.

4. ¿El sensor capacitivo no funciona? Mira la referencia para corregir el sensor capacitivo

5. ¿No puedes cargar el código? Asegúrate de que la librería CapacitiveSensor esté colocada dentro de la carpeta de librerías en la carpeta de Arduino.

¡Sigue experimentando!

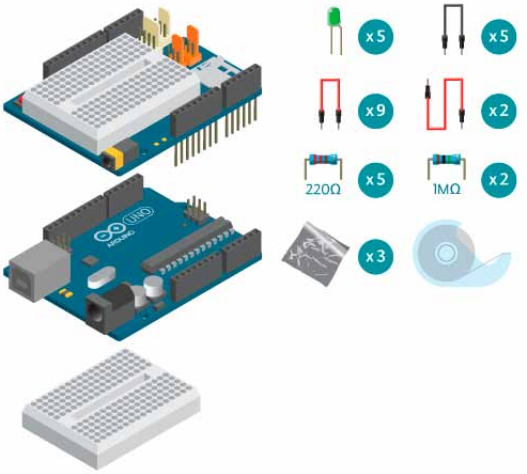
- ¿Se puede hacer el juego más difícil? Cambia el tiempo de reacción y el tiempo de espera.
- ¿No te acaban de gustar los sonidos que emite el piezo? Prueba a hacer tus propias melodías.
- Intenta cambiar la forma en que el LED parpadea cuando se acaba el juego. Mira el código de ejemplo VUMeterTEst para saber los comandos que puedes usar.
- ¿Se pueden agregar más LEDs y sensores? ¡No olvides agregar ambos tanto a la placa físicamente como al código del programa!

Carreras

¡Corre con tus dedos lo más rápido que puedas!
En este juego, el corredor tiene que golpear dos sensores repetidamente con el fin de completar vueltas. Un LED se encenderá por vuelta. Cuando todas las vueltas se hayan completado, los LED parpadearán celebrando la victoria.

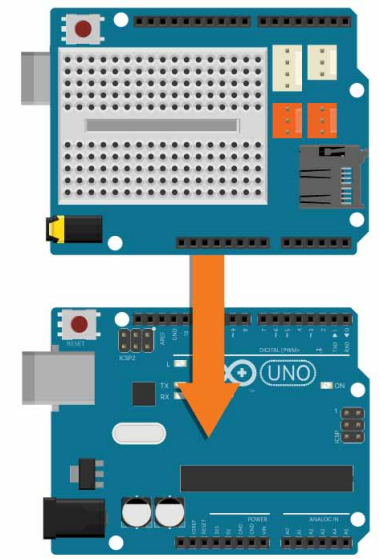
Materiales

- 1 placa Arduino
- 1 Shield Básica Educativa
- 5 LEDs
- 2 resistencias de 1Mohm
- 5 resistencias de 220 ohm
- 5 cables negros
- 11 cables de colores (2 largos)
- papel de aluminio
- cinta adhesiva
- 1 breadboard pequeña

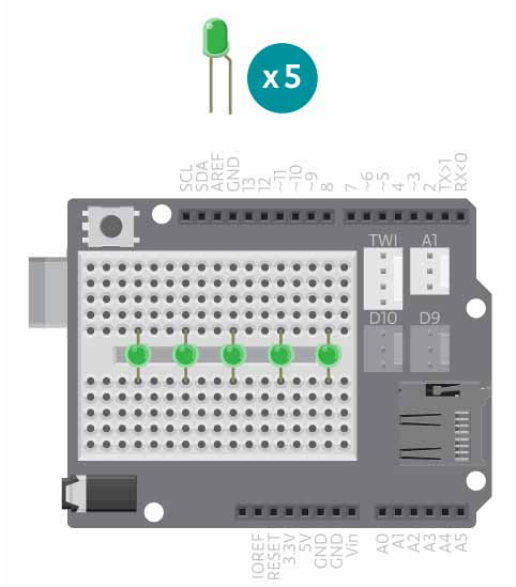


Instrucciones

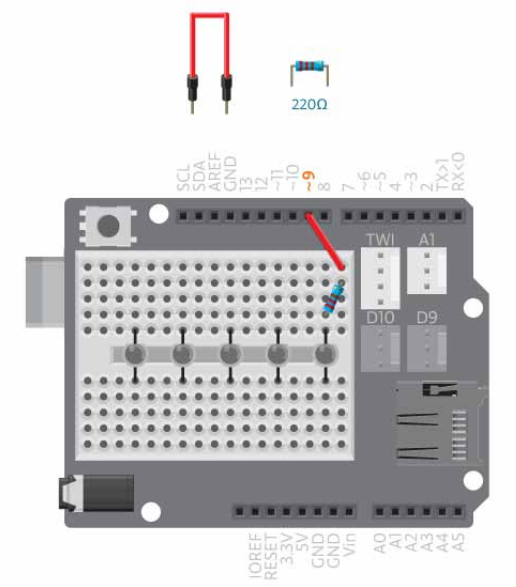
1. Conecta la shield a la parte superior de tu placa Arduino.



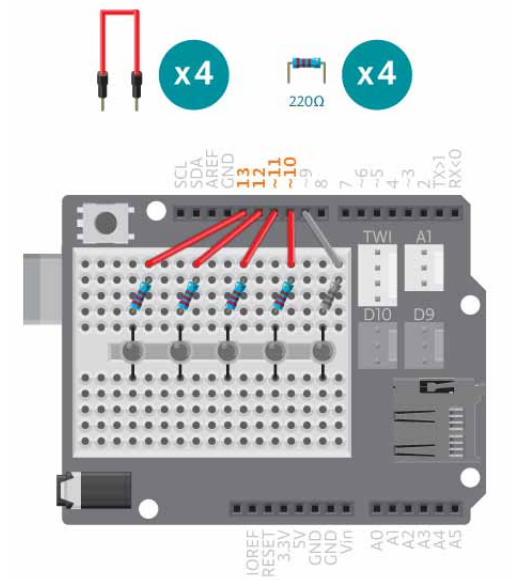
2. Conecta cinco LEDs individuales a través del puente de la breadboard.



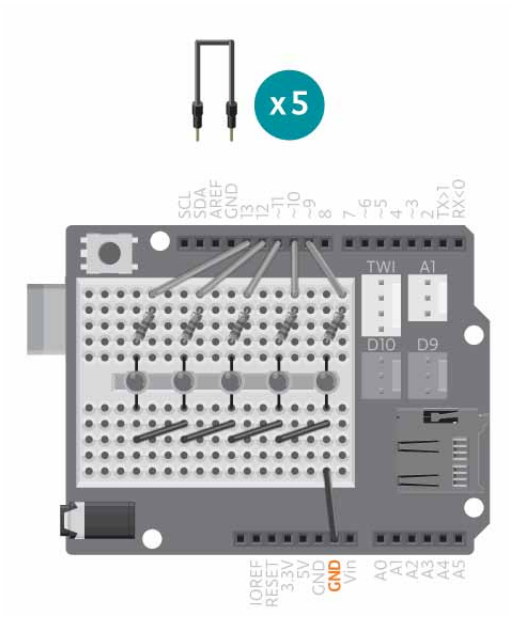
3. Conecta una resistencia de 220 ohm al Pin digital 9. Conecta la resistencia a la pata larga del primer LED.



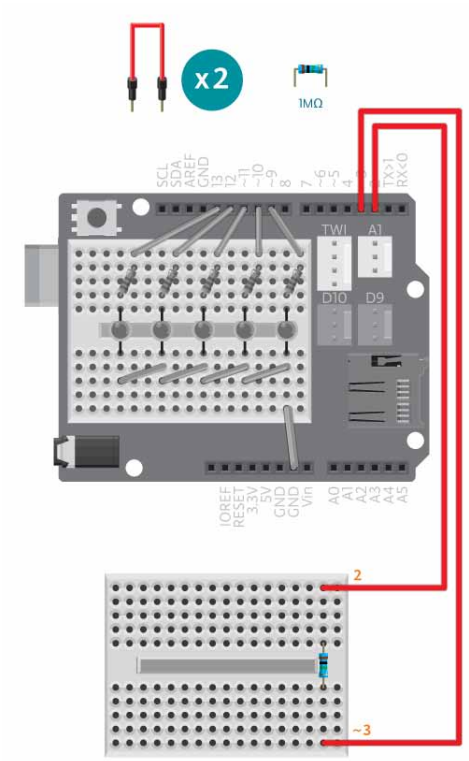
4. Conecta cada uno de los Pins digitales del 10 al 13 a su LED correspondiente siguiendo el mismo método.



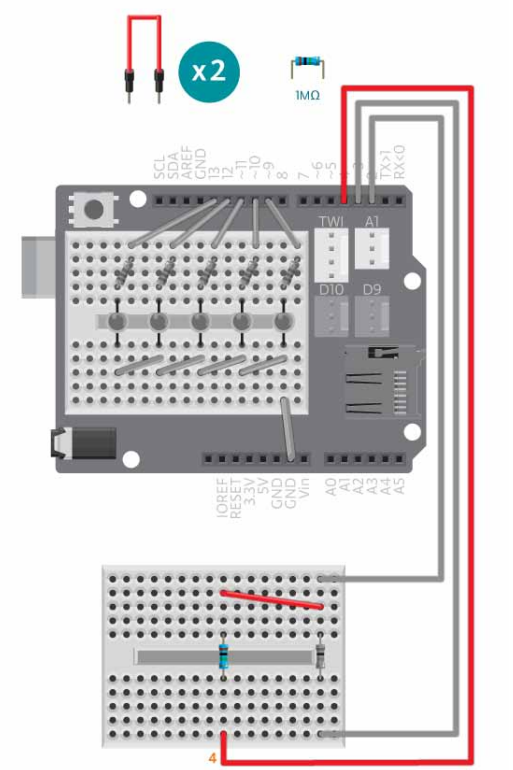
5. Conecta la pata corta de los LED a un pin de Arduino GND utilizando cables negros.



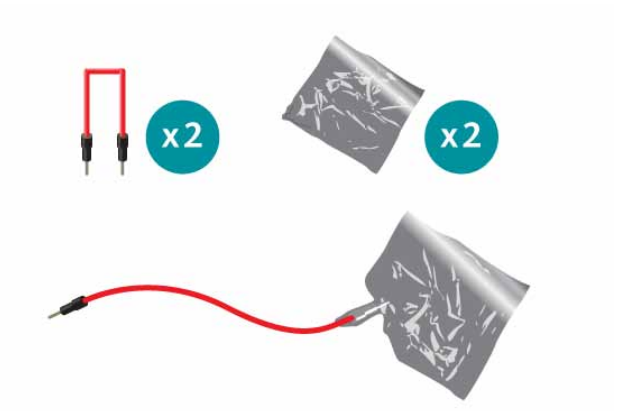
6. Conecta una resistencia de 1Mohm entre el Pin digital 2 y el Pin digital 3.



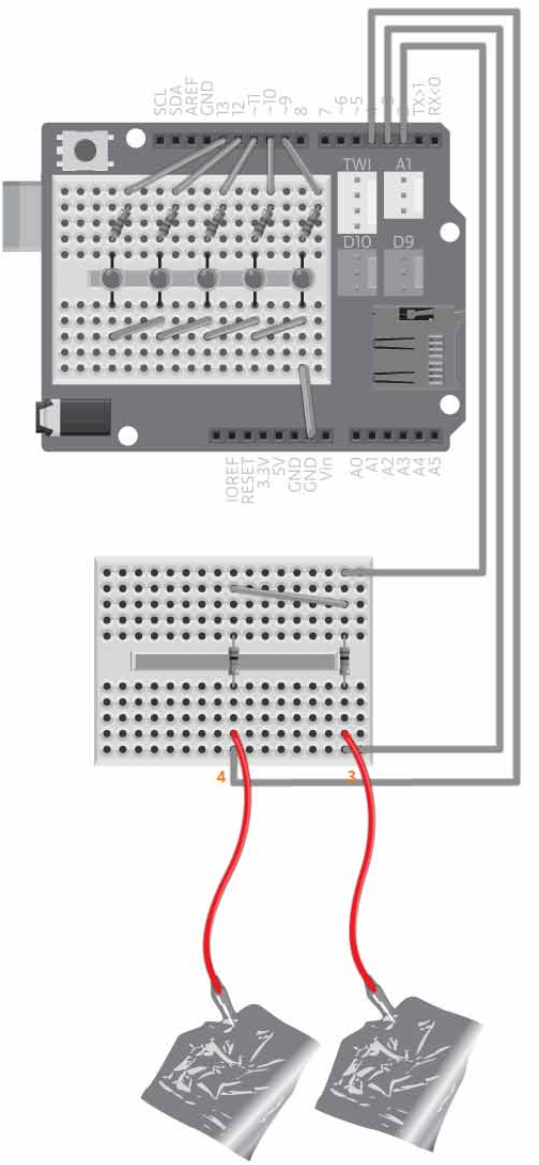
7. Conecta el Pin digital 4 siguiendo el mismo método.



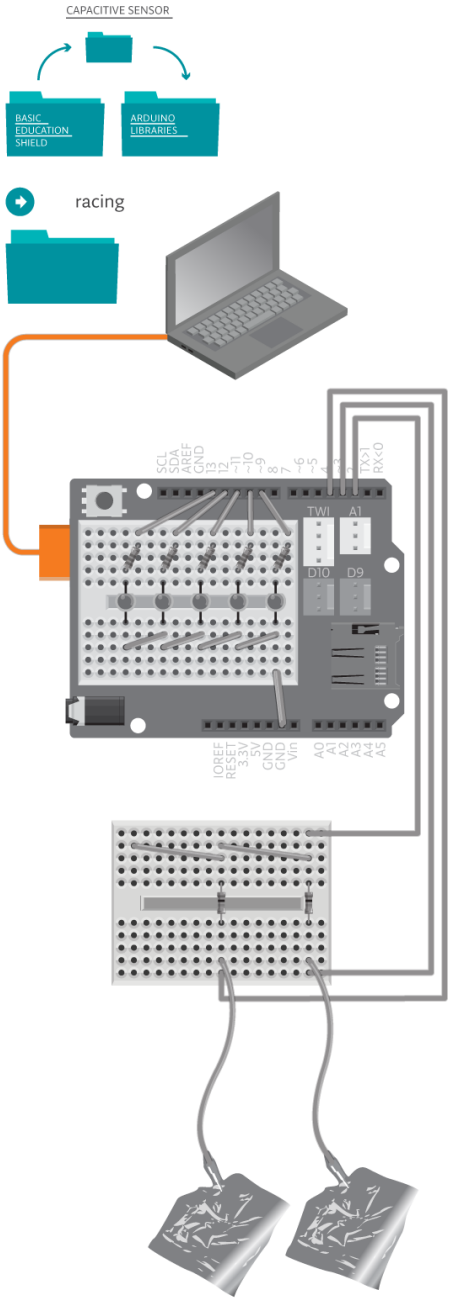
8. Haz un sensor de contacto cortando un cuadrado de papel de aluminio de 5 cm x 5 cm. Envuelve una esquina de la lámina cuadrada a un cable suelto, el cable de metal debe estar en contacto con el papel de aluminio. Repite el proceso.



9. Conecta cada uno de los sensores de contacto a los Pins digitales 3 y 4. Pega los dos sensores firmemente a la mesa asegurándote de que no se toquen.



10. Conecta el conjunto Arduino al ordenador. Mueve la carpeta CapacitiveSensor de la carpeta BasicEducationShield a la librería de la carpeta Arduino. Carga ejemplo Carreras y prueba el juego.



Código

Puedes encontrar el código en Archivo -> Ejemplos -> BasicEducationShield -> Sports -> Racing

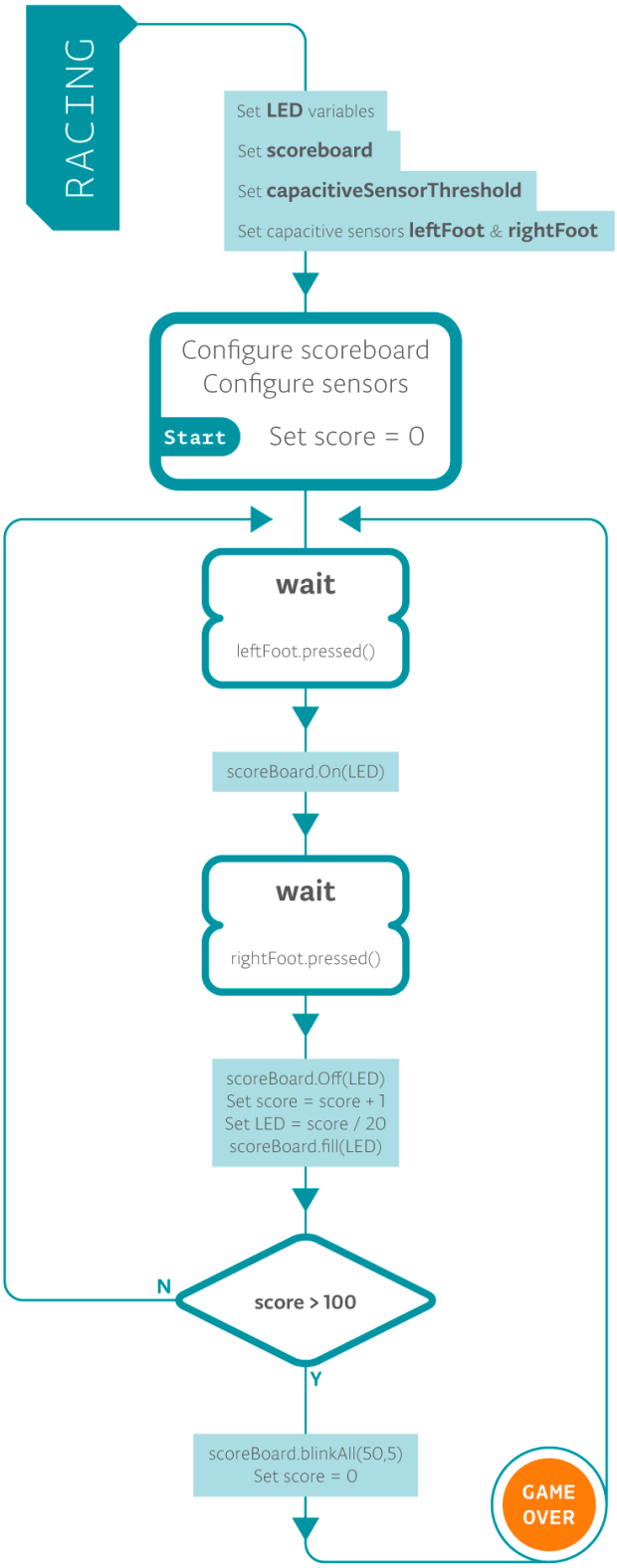
```
/*
  Racing

  Run with your fingers as fast as you can.

  In this game, the player must tap two sensors repeatedly
  in order to run laps. One LED will light up per lap.
  When all laps are completed, LEDs will blink in victory.

  (c) 2013 Arduino Verkstad
*/

#include <CapacitiveSensor.h>
#include <BasicEducationShield.h>

/*
  An array of pin numbers to which LEDs are attached
  the defaults are 2 to 6 but you can choose any of the digital pins
  just remember to leave digital pin 9 and 10 for the buttons
*/
int ledPins[] = {9, 10, 11, 12, 13};
int pinCount = 5;
//This variable will let us keep track on which LED to turn on
int LED;
VUMeter scoreBoard;

//Configure the capacitive sensors
int capacitiveSensorThreshold=400;
CapacitiveSwitch leftFoot=CapacitiveSwitch(2,3);
CapacitiveSwitch rightFoot=CapacitiveSwitch(2,4);

int score;

void setup(){
  //initializing the game, set up all the components and variables
  score=0;

  //Connect scoreboard
  scoreBoard.config(pinCount,ledPins);
  scoreBoard.begin();

  //initialize left and right “foot”
  leftFoot.config(capacitiveSensorThreshold);
  rightFoot.config(capacitiveSensorThreshold);
}

void loop(){
```

```
48   //Wait for the left foot to be pressed
49    leftFoot.pressed();
50    scoreBoard.on(LED);
51
52   //Wait for the right foot to be pressed
53   rightFoot.pressed();
54   scoreBoard.off(LED);
55
56   score=score+1; //After both feet are pressed, add one point
57
58   //Every 20 points light up a led
59   LED =score/20;
60   scoreBoard.fill(LED);
61
62  //When you get 100 points, you win
63   if(score>100){
64     //if you win, blink all leds for celebration
65     //See vuMeter in refence list to make your own blink animation
66     scoreBoard.blinkAll(50,5);
67     //and reset the game
68     score=0;
69   }
70 }
```

Cómo funciona

Debes alternar entre los dos sensores capacitivos -aprieta el de la izquierda y luego el de la derecha-. Por cada 20 "pasos", un LED se encenderá. Los LEDs se irán encendiendo y apagando con cada nueva vuelta.

Una vez hayas completado 100 "pasos", todos los LEDs parpadearán para la celebración. Después de eso, el juego se reiniciará.



¿No funciona?

1. Revisa las ilustraciones y comprueba tus conexiones. Asegúrate de que el shield y los cables están firmemente conectados.

2. ¿Al golpear las láminas no ocurre nada? Mira la referencia CapacitiveSwitch para depurar el sensor capacitivo.

3. ¿VU-meter no funciona correctamente? Mira la referencia VU-Metro? para depurar VU-meter.

4. ¿No se puede cargar el código? Asegúrate de que la biblioteca CapacitiveSensor esté colocada dentro de la carpeta de librerías (libraries) en tu carpeta Arduino.

¡Sigue experimentando!

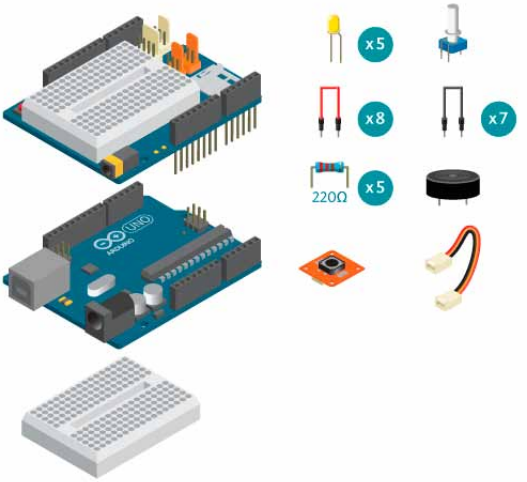
- Cambia la manera en que los LEDs parpadean cuando se acaba el juego. Mira el código de ejemplo VUMeterTest para conocer los comandos que puedes usar.
- ¿El juego es demasiado sencillo? ¡Intenta hacer que dure más!
- ¿Puedes colocar el papel de aluminio en otro lugar? ¿Puedes utilizar otra cosa que reemplace el papel de aluminio (tal vez a tus amigos)?
- ¡Cronométralo para ver quién corre más rápido!

Simón dice

¡Pon a prueba tu memoria con este juego!
Los LEDs parpadearán en una secuencia que deberás recordar y repetir. Si lo haces correctamente, el juego se vuelve más y más desafiante.

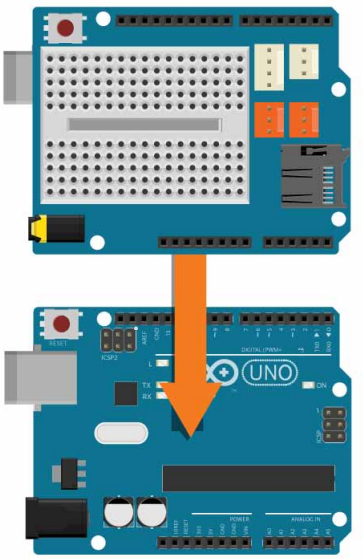
Materiales

- 1 placa Arduino
- 1 Shield Básica Educativa
- 1 botón Tinkerkit
- 1 cable Tinkerkit
- 1 potenciómetro
- 1 piezo
- 5 LEDs
- 5 resistencias de 220 ohm
- 1 breadboard
- 7 cables negros
- 8 cables de colores

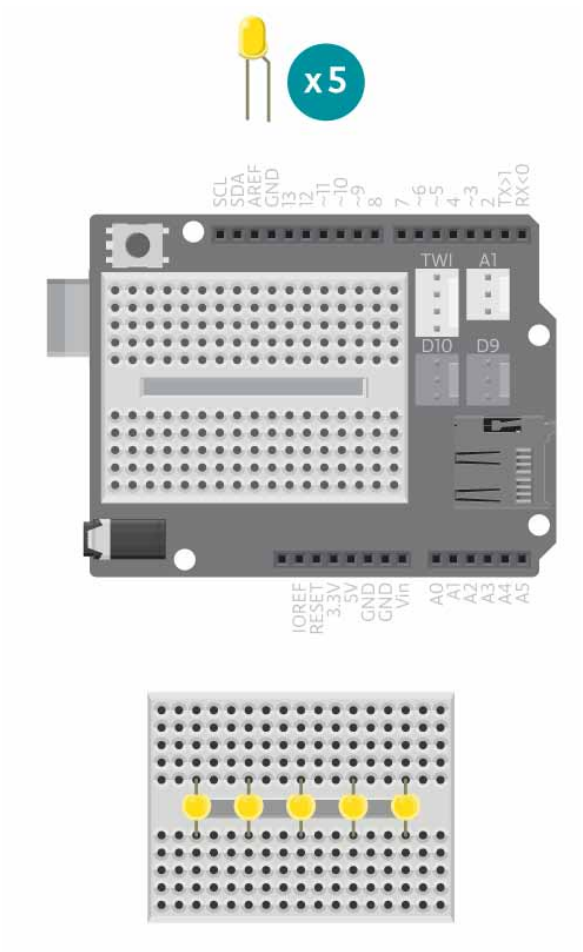


Instrucciones

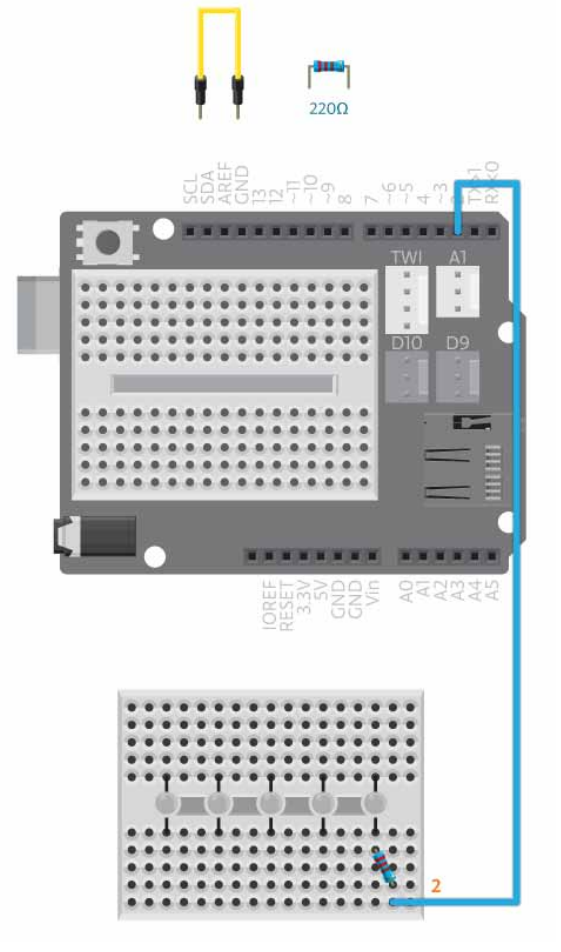
1. Conecta el shield encima de la placa Arduino.



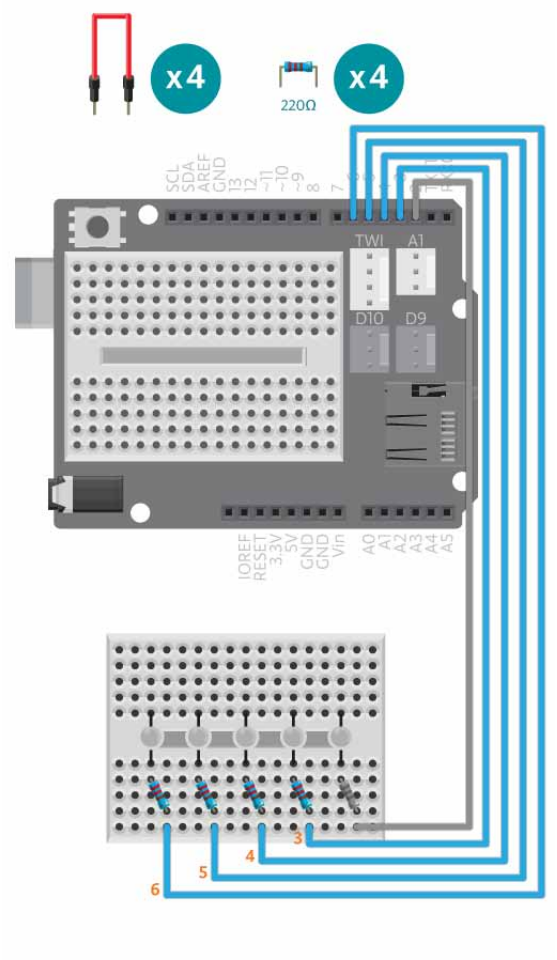
2. Conecta cinco LEDs a lo largo del puente de la breadboard.



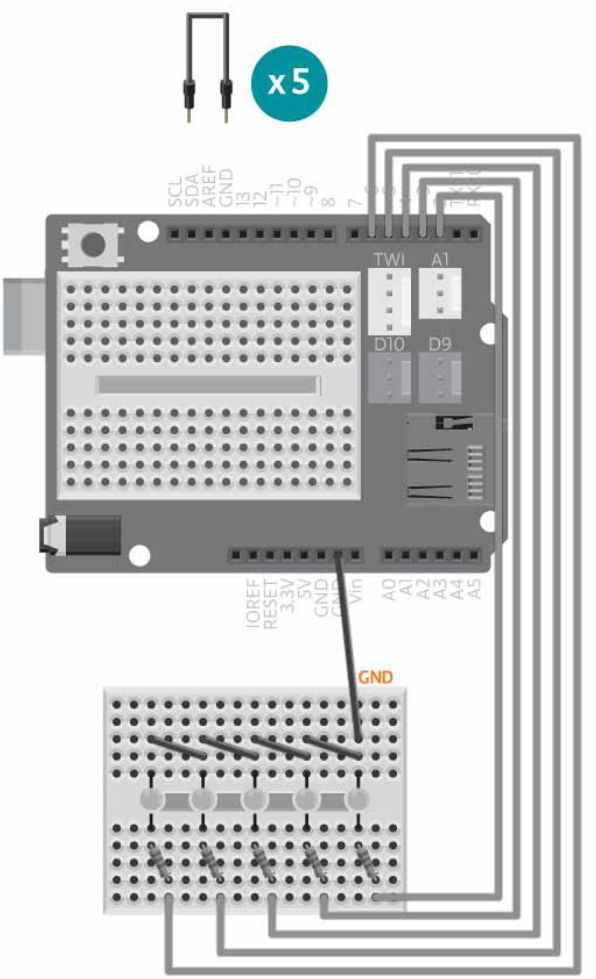
3. Conecta una resistencia de 220 ohm al pin digital 2. Conéctala a la pata larga del primer LED.



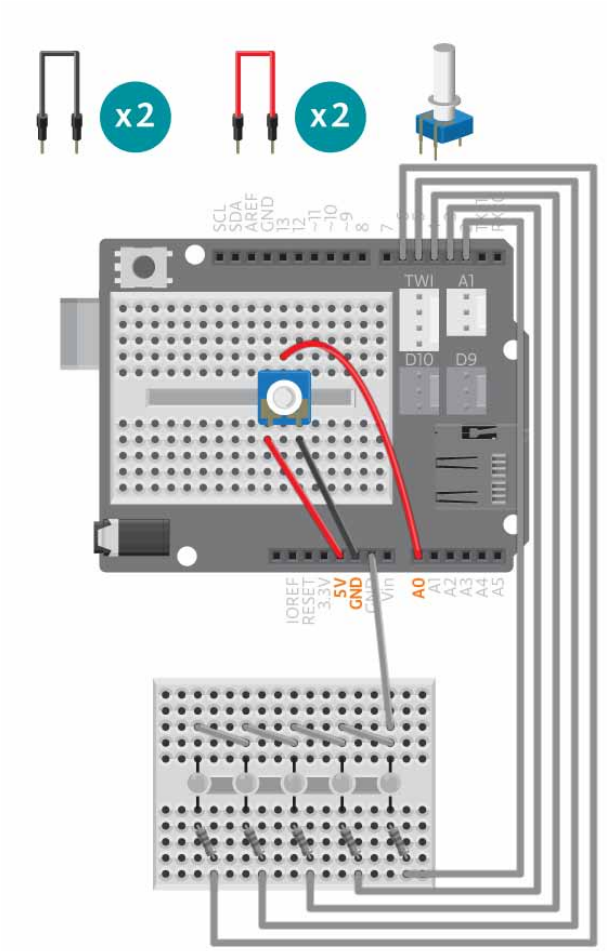
4. Conecta cada uno de los pins digitales 3 a 6 al LED correspondiente siguiendo el mismo método.



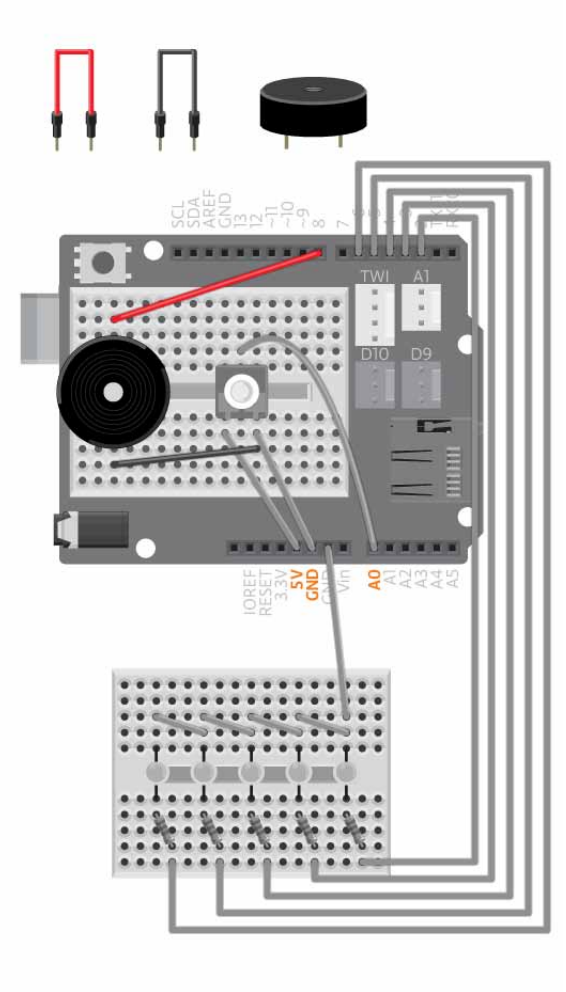
5. Conecta la pata corta de los LEDs a un pin GND de Arduino utilizando los cables negros.



6. Conecta el potenciómetro a la breadboard del shield. Conecta el pin de en medio del potenciómetro a A0, el pin izquierdo a GND y el derecho a 5V.

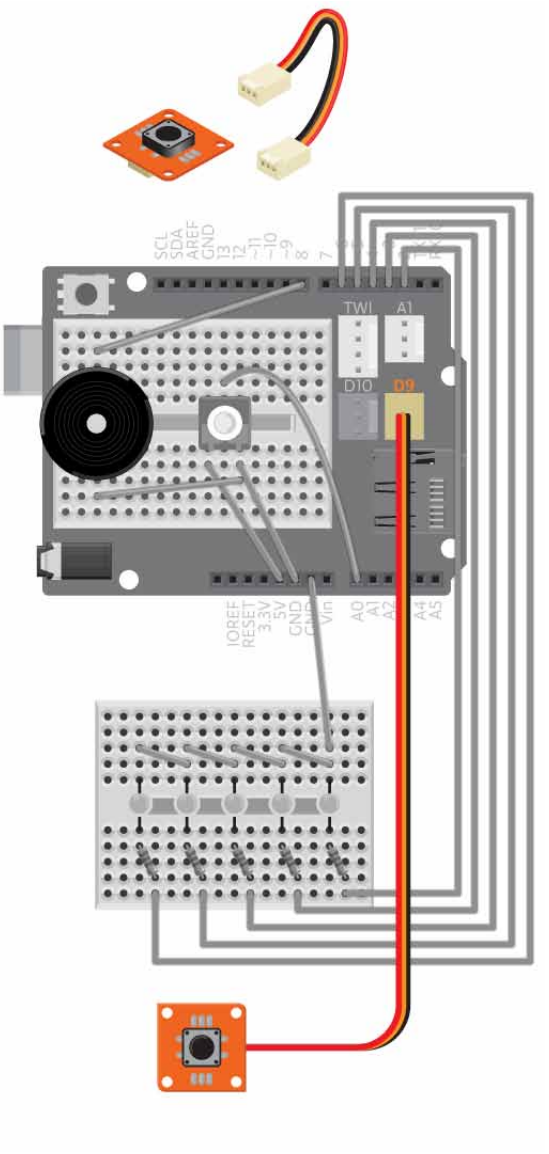


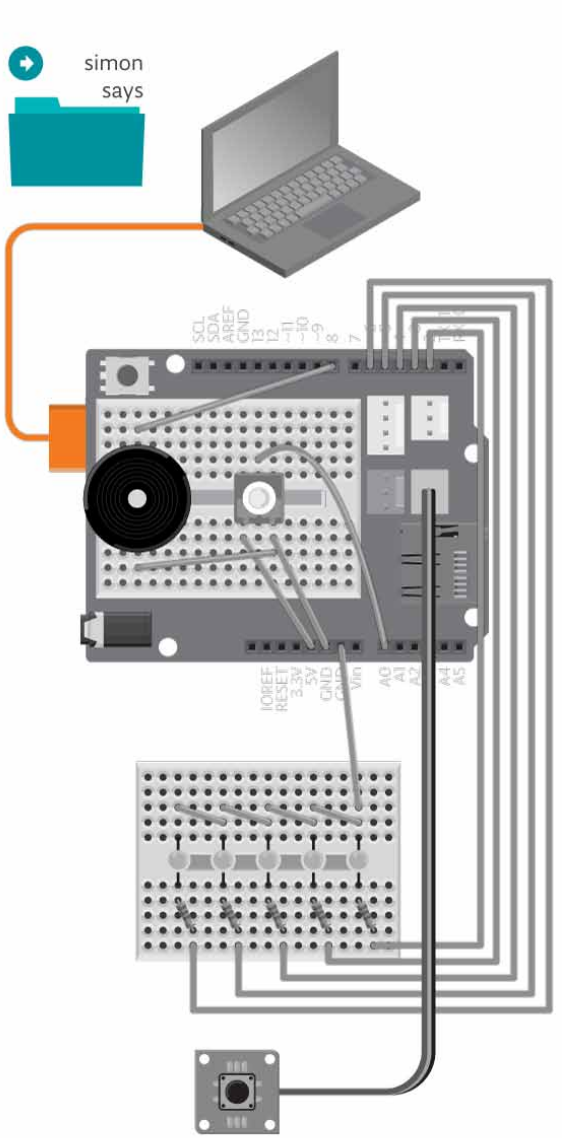
7. Conecta el piezo a GND y al pin digital 8.



8. Conecta el botón tinkerKit a D9.

9. Conecta el montaje Arduino al ordenador, carga el ejemplo SimonSays y prueba el juego.





Código

Puedes encontrar el código en "Archivo -> Ejemplos -> Deportes -> SimonSays"

```
/*
  Simon Says

  Test your memory with this game!

  LEDs will blink in a pattern that you have to remember and repeat.
  If you get it right, the game gets more and more challenging.

  (c) 2014 Aduino Verkstad
*/

#include <BasicEducationShield.h>

/*
  An array of pin numbers to which LEDs are attached
  the defaults are 2 to 6 but you can choose any of the digital
  pins.
*/
int ledPins[] = {2, 3, 4, 5, 6};
int pinCount = 5;
VUMeter vuMeter;

Knob pot=Knob(A0);//a knob is connected to A0

Button button = Button(9); //a button is connected to digital pin 7

Melody piezo=Melody(8);//a piezo is connected to digital pin 8

//Set the game parameters
int turns_begin=2;//a new game starts with 2 turns
int turns_max=10;//the most difficult game has 10 turns

int game[10];//array for storing the “simon says”
int turns=2;//for storing the number of turns in current game
int blinkTime=500;//how fast does “simon says” blink

void setup(){
  //initialize the components
  vuMeter.config(pinCount, ledPins);
  vuMeter.begin();
  pot.setLevels(5);
  button.begin();

  //Create a random seed, so we can call random() later.
  randomSeed(analogRead(A5));
}
void loop(){
```

```
  newGame();
  simonSays();
  delay(1000);
  getInputs();
}

void newGame(){
  vuMeter.blinkAll(200, 3);
  vuMeter.clear();
  delay(500);
  //Generate simon says, it'll be stored in an array
  //So we can compare with player's input later
  for(int i=0;i<turns;i++){
    game[i]=random(pinCount);
  }
}
void simonSays(){
  //Display simon says to the player.
  for(int i=0;i<turns;i++){
    vuMeter.on(game[i]);
    delay(blinkTime);
    vuMeter.off(game[i]);
    delay(blinkTime);
  }
}
void getInputs(){
  //Get "i" inputs where "i" matches the number of
  //simon says in this round.
  for(int i=0;i<turns;i++){
    int input;
    //button.released() stops the program, so
    //let's do it in short pulses, in between
    //we can change the vuMeter display.
    while(!button.released(10)){
     vuMeter.clear();
     vuMeter.on(pot.getLevel());
    }
    //When button is released, get the adjusted
    //value from the knob
    input=pot.getLevel();
    if(input==game[i]){
      //if your input is right, play a score sound
      //and continue.
      piezo.effect_score();
    }else{
      //Otherwise, gameover. Stop the function from
      //continuing by an empty return.
      gameOver();
      return ;
    }
  }
  //When all the inputs matched simon says, you win
```

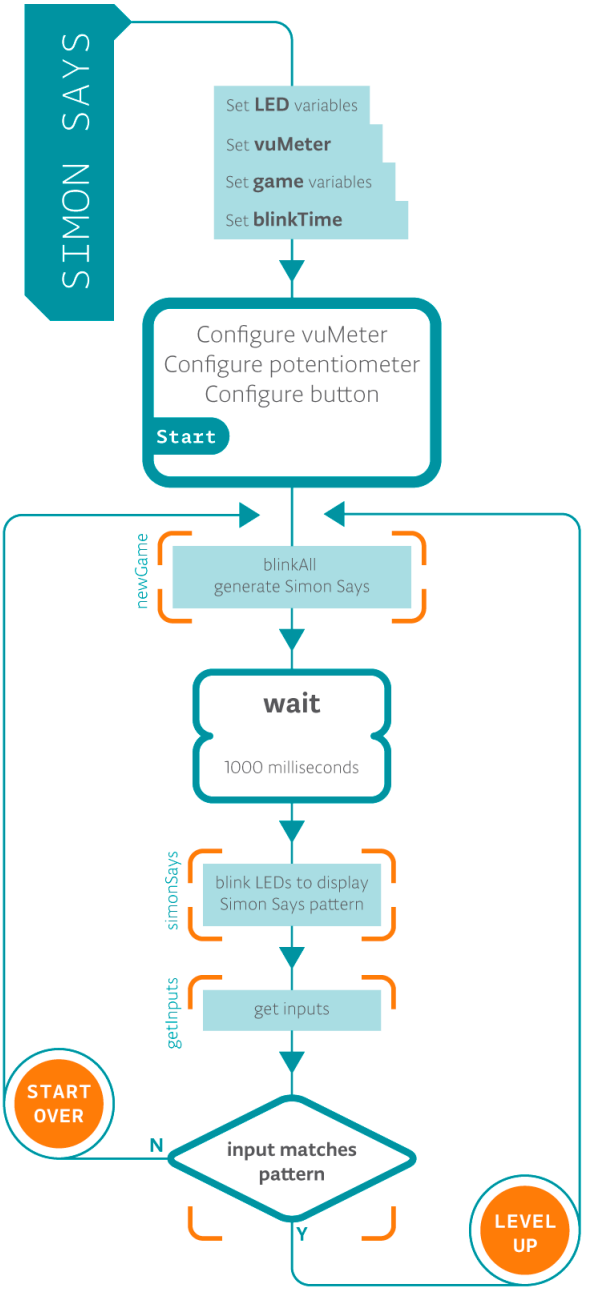
```
100   //this round and level up.
101   delay(500);
102   levelUp();
103 }
104 void gameOver(){
105   //When it's gameover, difficutly resets
106   turns=turns_begin;
107   //And play the gameover sound
108   piezo.effect_gameover();
109 }
110 void levelUp(){
111   //When level up, adds more difficulty until it reaches maximum
112   if(turns<turns_max){
113     turns++;
114   }
115   //And play a wining sound
116   piezo.effect_win();
117
118 }
```

Cómo funciona

El juego empieza generando tantos números aleatorios como el valor de la variable `turns`. Los números aleatorios pueden ir de 0 hasta el número de LEDs usados. Esto es lo que sucede en `newGame()`.

Nos movemos a `simonSays()` donde los LEDs correspondientes a los números aleatorios generados parpadean en secuencia. Esperamos 1000 milisegundos y esperamos a las entradas en `getInputs()`. Comprobamos tantas entradas como veces en `turns`. Cada vez que una entrada es detectada, comprobamos si es el mismo número que el generado aleatoriamente por orden. Si no lo es, saltamos a `gameOver()` donde el juego se resetea. Si es correcto, seguimos comprobando las entradas restantes.

Cuando comprobamos tantas entradas como veces en `turns` sin obtener ningún error, vamos a `levelUp()`. `turns` se incrementa en uno y suena un sonido de victoria antes de volver a empezar el juego.



¿No funciona?

1. Revisa las ilustraciones y comprueba tus conexiones. Asegúrate de que el shield y los cables están firmemente conectados.

2. ¿El botón no funciona? Mira la referencia para depurar el botón.

3. ¿El VU–meter no funciona correctamente? Mira la referencia para depurar el VU-meter.

¡Sigue experimentando!

- Haz el juego más desafiante haciendo que vaya más rápido.
- Hazlo un juego de dos jugadores. Graba la secuencia del primer jugador y deja que el segundo la repita.

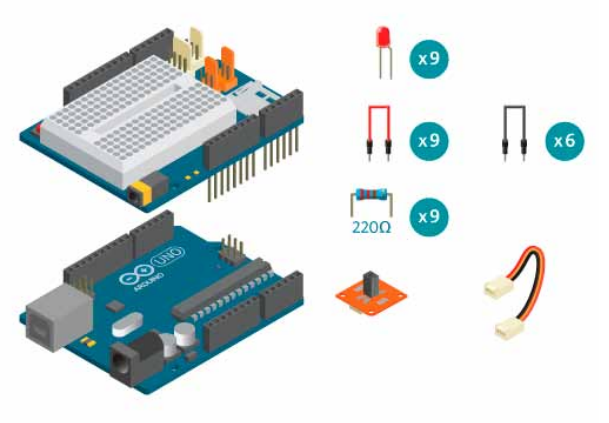
Dado digital

Utiliza este dado digital la próxima vez que juegues a un juego de mesa.

"Lanzas" el dado agitando el sensor tilt. Los LEDs mostrarán diferentes números, cambiando cada vez más lentamente, hasta que se detiene en un número específico. No te apresures a pensar que se ha detenido en tu número deseado, que podrías decepcionarte...

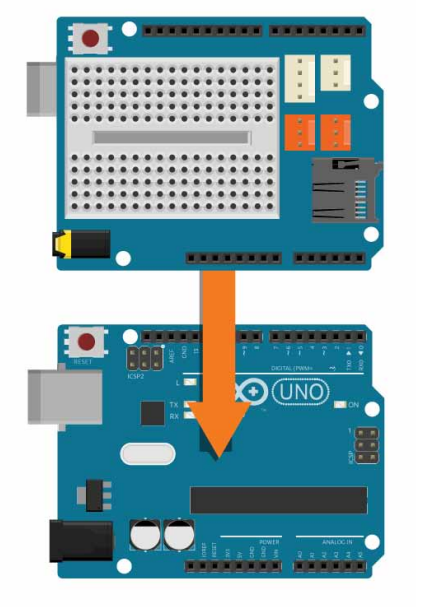
Materiales

- 1 placa Arduino
- 1 shield Básico Educativo
- 1 interruptor tilt Tinkerkit
- 1 cable Tinkerkit
- 9 LEDs
- 9 resistencias de 220 ohm
- 6 cables negros
- 9 cables de colores

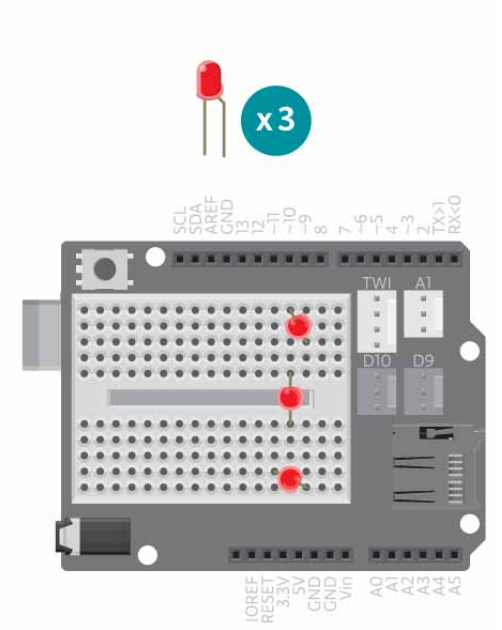


Instrucciones

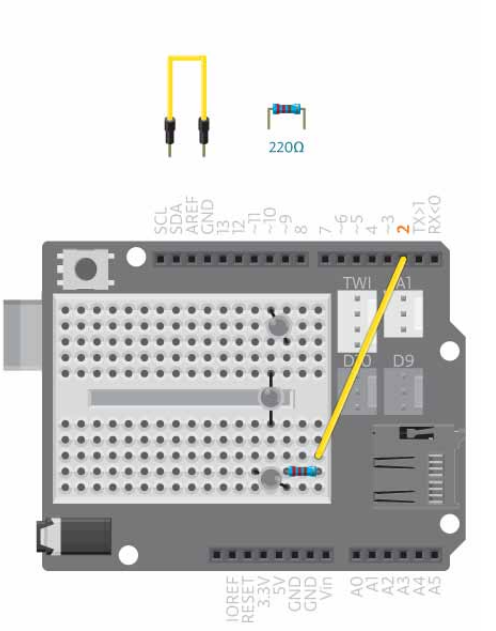
1. Conecta la shield a la placa Arduino.



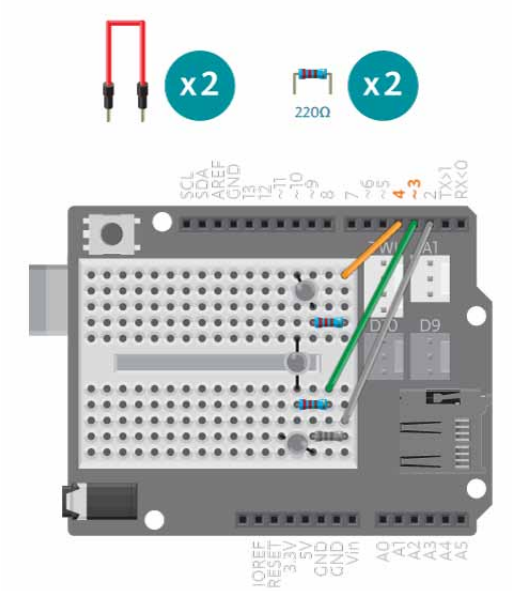
2. Conecta 3 LEDs a la breadboard.



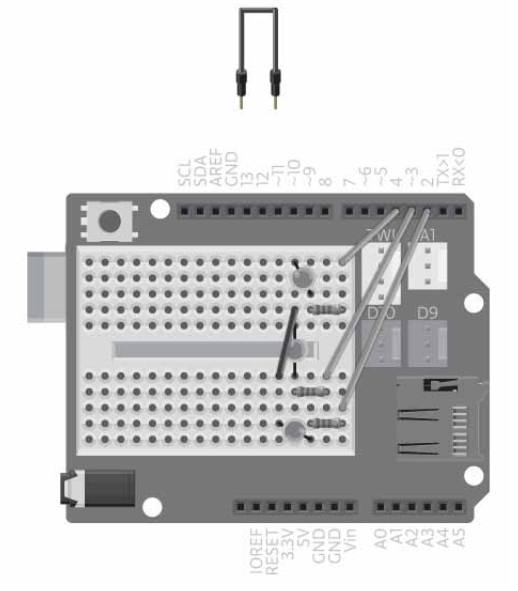
3. Conecta una resistencia de 220 ohm al pin digital 2. Conecta la resistencia a la pata larga del LED.



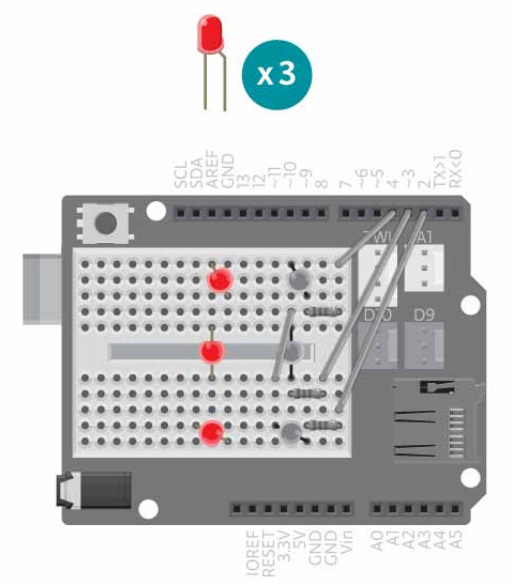
4. Conecta los pines digitales 3 y 4 al correspondiente LED siguiendo el mismo método.



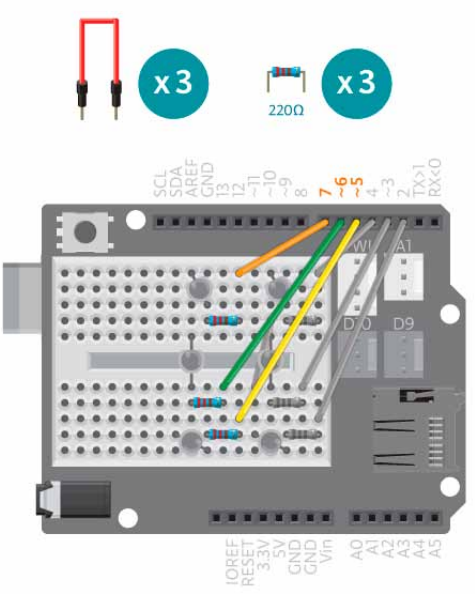
5. Conecta la pata corta del LED con un cable negro.



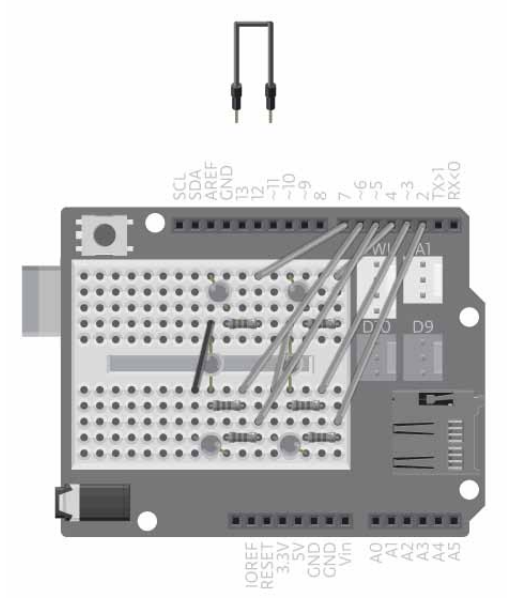
6. Conecta los siguientes 3 LEDs a la breadboard.



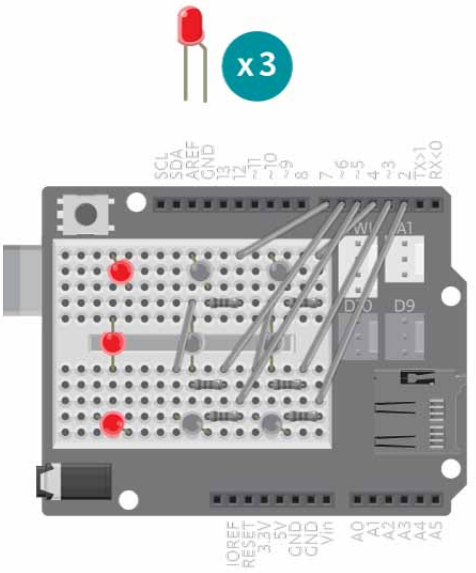
7. Conecta los pines digitales 5 y 7 al correspondiente LED a través de una resistencia de 220 ohm.



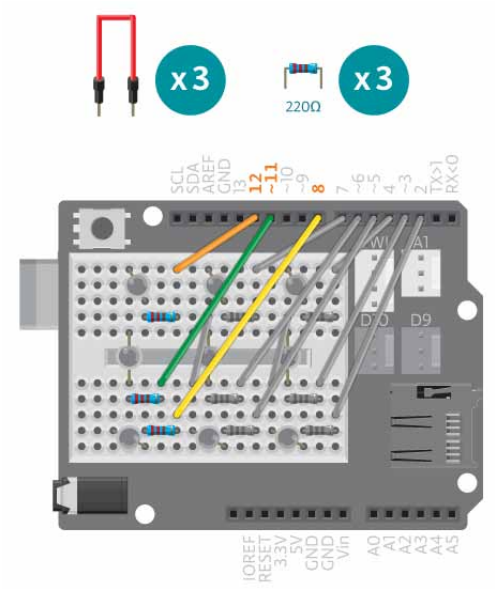
8. Connect all short the legs of the LEDs with 2 black jumper wires.



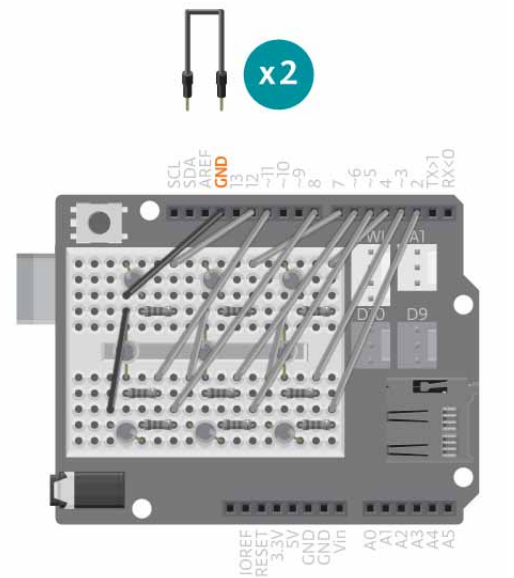
9. Conecta los 3 últimos LEDs a la breadboard.



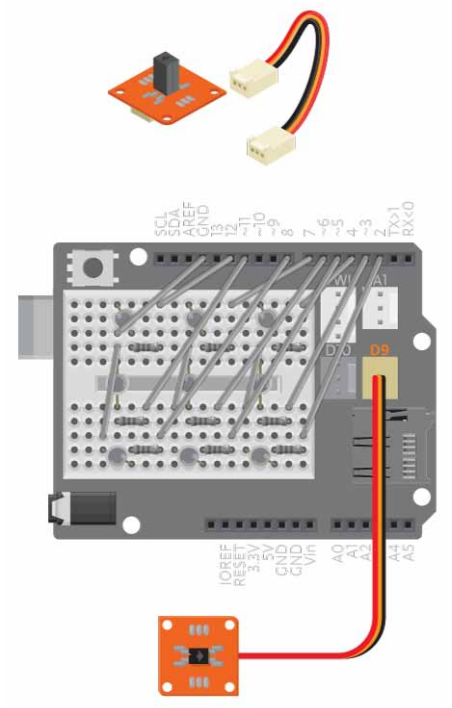
10. Conecta los pines digitales 8, 11 y 12 al LED correspondiente mediante una resistencia de 220 ohm.



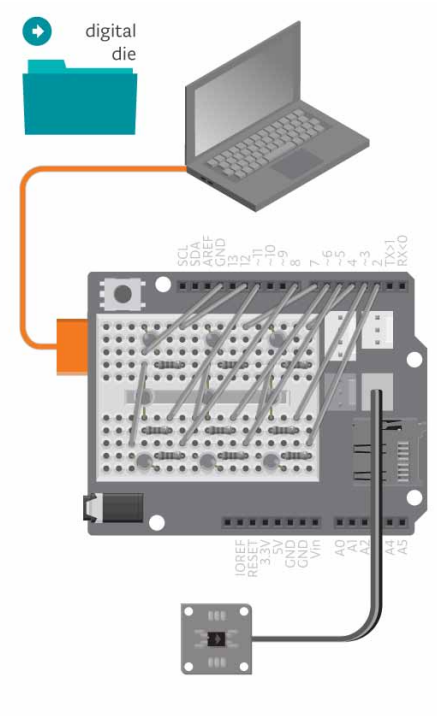
11. Conecta las restantes patas cortas de los LEDs con 3 cables negros a un pin GND de Arduino.



12. Conecta el interruptor tilt tinkerKit a D9.



13. Conecta el montaje Arduino al ordenador, carga el ejemplo Digital Die y prueba el juego.



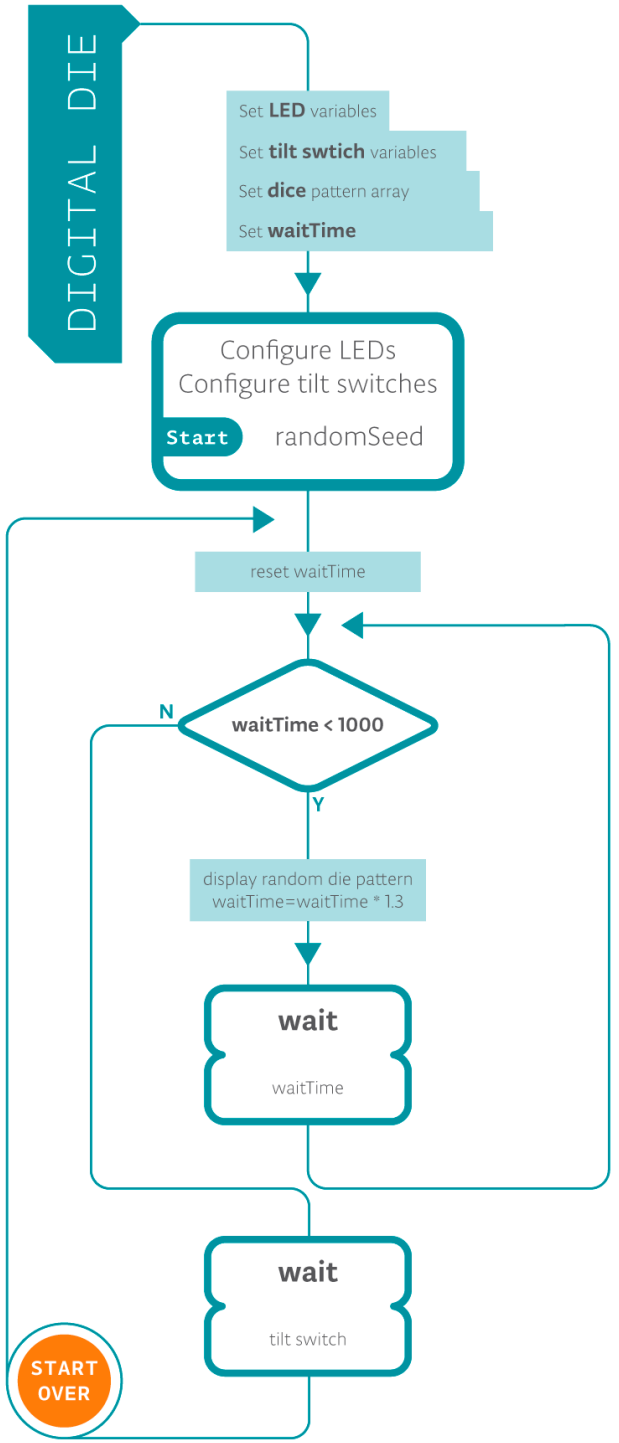
Código

Puedes encontrar el código en "Archivo -> Ejemplos -> Deportes -> Digital Die"

```
/*
  Digital Die

  Use this digital die next time you play a board game.

  You "throw" the die by shaking a tilt sensor. The LEDs
  will show different numbers, waiting a longer and longer
  time for each number,  until it finally stops. Don't be
  too fast to cheer believing it stopped on your desired
  number or you might get disappointed ...

  (c) 2014 Arduino Verkstad
*/

#include <BasicEducationShield.h>

//declare the pins used for leds, 9 leds in total
int pinCount=9;
int ledPins[] = {2, 3, 4, 5, 6, 7, 8, 11, 12};

/*
declare the tilt switch, it's connected to tinkerkit
port 9
*/
TiltSwitch ts=TiltSwitch(9);

/*
  Define the patterns of die values. Each pattern is
  an array of 9 integers, indicating the on/off state
  of each led.
  And because there're 6 possible patterns, we need a
  2-dimensional array to define all the data. It's a
  big array of 6 elements, each element is an array of
  9 integers.
*/
int dice[6][9]={
1
  {
  0,0,0,
  0,1,0,
  0,0,0
  },

2
  {
  1,0,0,
  0,0,0,
```

```
  0,0,1
  },

3
  {
  1,0,0,
  0,1,0,
  0,0,1
  },

4
  {
  1,0,1,
  0,0,0,
  1,0,1
  },

5
  {
  1,0,1,
  0,1,0,
  1,0,1
  },

6
  {
  1,1,1,
  0,0,0,
  1,1,1
  }
};

/*
  wait time between the die rolls to a different face.
  Notice it's using float type here? Read on!
*/
float waitTime=1;

void setup(){
  //Configure each pin as an output.
  for(int i=0;i<pinCount;i++){
    pinMode(ledPins[i],OUTPUT);
  }

  //initialize the tilt switch.
  ts.begin();

  //generate the random seed. We use the value from
  //A0, since it's not connected to anything, it should
  //generate some random noises. Perfect for our purpose
  randomSeed(analogRead(A0));
}
```

```
100 void loop(){
101   //Reset the wait time
102   waitTime=1;
103 
104   /*
105     Imagine when you throw a dice, it'll bounce around,
106     showing a few values before laying still.
107 
108     Let's keep generating new values until it's stable
109     (when time between new values become long enough)
110   */
111   while(waitTime<1000){
112     /*
113       Generate a random dice value.
114       The dice value can be 1 to 6, in the array it's
115       dice[0] to dice[5]. random(0,6) generate a value
116       between 0 to 6, that would be 0 to 5 in effect.
117     */
118     int value;
119     value=random(0,6);
120 
121     //Display the dice value
122     displayDice(value);
123 
124     /*
125       See why waitTime have to be float? If it's an integer,
126       multiply it by 1.3 will make it lose everything behind
127       the decimal mark. We use 2 as the starting value,
128       2*1.3 should be 2.6, losing the fractional parts means
129       it'll be 2 in the end, so 2*1.3=2! It'll
130     */
131     waitTime=waitTime*1.3;
132     delay(waitTime);
133   }
134 
135   /*
136     Now the dice is stable, wait untill the tilt switch is
137     activated again. ts.pressed() stops the whole program
138     until it's activated.
139   */
140   ts.pressed();
141 }
142 
143 void displayDice(int num){
144   //Show the dice value by turning on/off the right leds
145   for(int i=0;i<pinCount;i++){
146     digitalWrite(ledPins[i],dice[num][i]);
147   }
148 }
```



Cómo funciona

Declaramos las seis figuras diferentes del dado, donde '0' representa un LED apagado y un '1' un LED encendido. El programa genera un valor aleatorio entre 0 y 5, donde 0 es el primer patrón de LEDs, 1 es el segundo, etc. El patrón de LEDs se muestra, y entonces esperamos tantos milisegundos como el valor de `waitTime`. Repetimos este procedimiento e incrementamos `waitTime` cada vez, hasta que `waitTime` es mayor que 1000. El último número generado será el patrón que el dado mostrará al final. El programa se detendrá hasta que detecte una nueva agitación del sensor tilt. Cuando el sensor es agitado, el programa comienza de nuevo.

¿No funciona?

1. ¿El dado no muestra la combinación correcta de los LEDs encendidos? Revisa las ilustraciones y comprueba tus conexiones. Asegúrate de que los cables están firmemente conectados.

2. ¿No funciona correctamente el interruptor tilt? Asegúrate de que no está boca abajo cuando el dado se esté "agitando" todavía. Mira la referencia para corregir el Sensor Tilt.

¡Sigue experimentando!

- Haz que los LEDs muestren algo más que sólo las figuras de un dado.
- Usa una pila de 9V para alimentar a Arduino y crea una caja que incorpore todos los componentes de forma que la tengas que agitar entera para "lanzar" el dado.

SEMANA 3
Magia

CONCEPTOS

PROYECTOS

Esta semana vamos a construir algunos proyectos relacionados con la magia. Para alucinar del todo, os juntaréis en grupos y construiréis uno de los proyectos más increíbles pero también, de los más sencillos. Serás un músico reproduciendo música con diferentes objetos al azar, un artista que experimenta con el arte temporal o simplemente crearás un monstruo que proteja tus galletas.

Para realizar el trabajo, podrás adentrarte en el reino de las señales analógicas. Aprenderás los conceptos básicos de lectura y escritura analógica, además de jugar con sensores que dan valores analógicos. También podrás conocer cómo funciona el Puerto Serial, que le da la potencia a Arduino para comunicarse con el ordenador, ampliando sus capacidades.

Leyendo en analógico

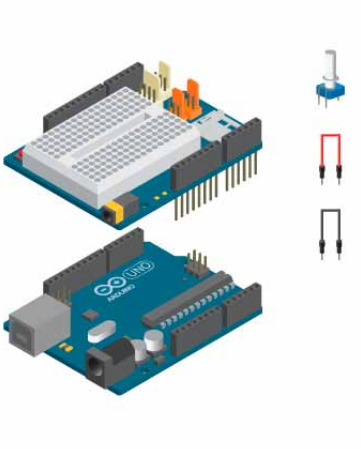
Como has aprendido anteriormente, las señales digitales solo tienen dos estados opuestos: 1 ó 0. Si presionas un botón, su estado cambiará de uno a otro. Un LED está encendido o apagado.

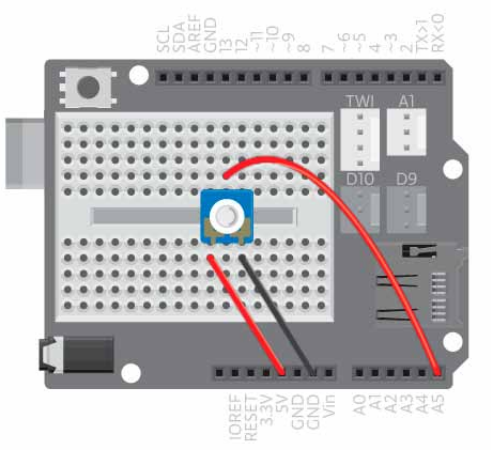
El mundo real sin embargo, no es digital. El agua por ejemplo, no sólo está caliente o fría, puede también estar templada. Para poder medir eso, y otras cosas del mundo real, no podemos usar sólo señales digitales. En lugar de ello, empleamos señales analógicas.

En vez de 2 estados opuestos, las señales analógicas tienen niveles continuos. Así que si tienes un sensor de luz, puedes obtener muchos valores diferentes que expresan cómo de iluminada está la habitación, y no sólo clara/oscura. O en el caso de un termómetro, te dice la temperatura mediante un número, en lugar de fría/caliente.

Con Arduino, puedes obtener los valores analógicos de los Pins analógicos. Sobre la placa puedes ver un grupo de Pins marcados como analog in, que llevan el nombre de A0 a A5. Cuando se les aplica tensión, ellos reportan valores de 0 a 1023; de este modo, cuando no hay voltaje en un Pin, la lectura es 0. Si le aplicas 5V, la lectura será 1023. Con una entrada de 2.5V, te dará 512. Para leer estos valores desde un Pin analógico tienes que utilizar la función `analogRead()`, en lugar de `digitalRead()`.

Para seguir explicando las señales analógicas, necesitamos introducir el potenciómetro. Un potenciómetro es un control que funciona mediante el giro de un brazo. Por ejemplo, el regulador de volumen de un estéreo es un potenciómetro. Con dos Pins exteriores al potenciómetro, conectados a GND y 5V respectivamente, puedes utilizar el potenciómetro para controlar la cantidad de tensión que quieres que haya en el Pin central, siempre dentro del rango de 0V a 5V.





Aquí vamos a experimentar un poco usando un potenciómetro para controlar el parpadeo de un LED. Conecta el Pin central del potenciómetro al Pin analógico A5, los otros Pins conéctalos a 5V y GND. Usaremos el LED on-board, incluido en la placa, como salida.

```
1   int ledPin = 13;
2
3   void setup() {
4   pinMode(ledPin, OUTPUT);
5   }
6
7   void loop() {
8   int val = analogRead(A5);
9   digitalWrite(ledPin, HIGH);
10  delay(val);
11  digitalWrite(ledPin, LOW);
12  delay(val);
13  }
```

Puedes encontrar un nuevo comando:

- `analogRead (pinNumber)`: Este comando coge la lectura de un Pin analógico especificado por la variable `pinNumber`, que puede ser desde A0 hasta A5. El valor leído variará entre 0 (0V) y 1023 (5V).
- Los pines analógicos sólo se pueden usar como entradas, por lo que no es necesario declarar el pin mode en el `setup`.

La lectura del potenciómetro conectado al pin analógico A5 cambiará el tiempo de retardo, delay, y por lo tanto la velocidad con la que parpadea el LED on-board del pin 13. Cuando gires el brazo del potenciómetro, podrás ver el cambio en la frecuencia de parpadeo.

¡Sigue experimentando!

- El valor de `analogRead()` tiene 1024 niveles – entre 0 y 1023. ¿Se te ocurre una manera de procesar un valor utilizando menos niveles? Imagina que sólo necesitamos 10 valores diferentes, manteniendo 0V como 0 pero 5V como 10, en lugar de 1023.(SUGERENCIA: Busca una función llamada map() en la referencia de Arduino.)
- ¿Qué sucede si reduces el número de niveles de `analogRead` a solamente dos valores, 0 y 1? Intenta aplicarlo al ejemplo del botón digital. ¿Sientes que has entendido la relación entre las señales analógicas y digitales?

Escribiendo en analógico

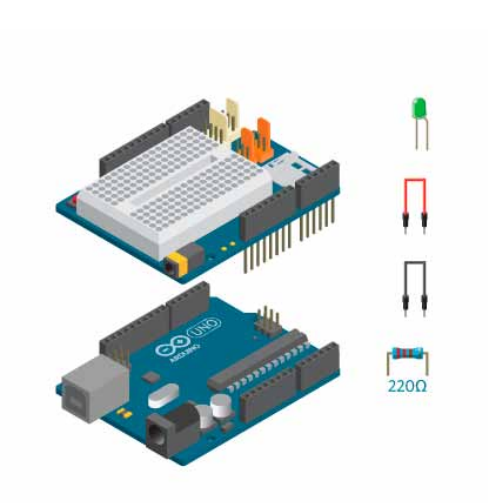
Al igual que puedes leer señales analógicas, también puedes generar valores analógicos. Arduino utiliza Pins con Modulación Controlada por Pulsos, PWM (del inglés Pulse Width Modulation) para enviar los valores analógicos. Echa un vistazo a los Pins digitales que llevan al lado un símbolo de corriente (~).

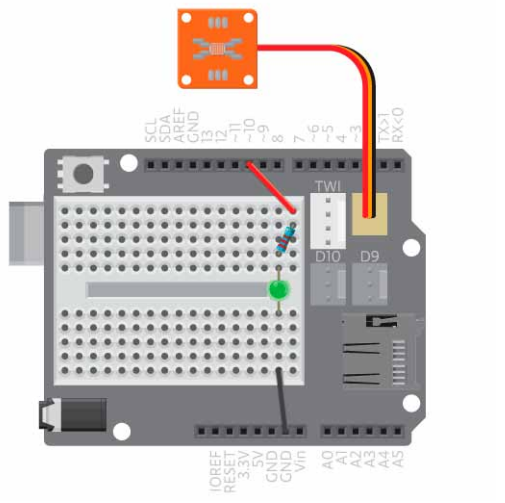
Cuando se utiliza un Pin digital normal para escribir `HIGH` o `LOW`, se obtiene bien 5V o 0V. Estos Pins PWM tienen una habilidad diferente, puedes utilizarlos para obtener un nivel de tensión en cualquier punto entre 5V y 0V. Con esto, puedes atenuar un LED sin problemas entre ON y OFF.

Para utilizar esta habilidad especial de los Pins PWM, tendrás que usar la función `analogWrite()`. Se necesitan dos parámetros, el número del Pin PWM, y el nivel de salida. El nivel de salida es un número entre 0 y 255. 0 es igual a `digitalWrite(pin, LOW)`, 255 es igual a ejecutar `digitalWrite(pin, HIGH)` y puedes llegar a cualquier punto entremedias.

Nota: Ten cuidado con la diferencia entre analogRead y analogWrite: analogRead tiene 1024 niveles de lectura, mientras que analogWrite sólo tiene 256. analogRead utiliza Pins analógicos, mientras que analogWrite utiliza Pins digitales tipo PWM.

Veamos un ejemplo. Coge un LED y una resistencia de 220 Ohmios (con líneas de color rojo, rojo, marrón). Conecta el Pin largo del LED a la resistencia y el otro Pin de la resistencia al pin digital 10. Conecta el brazo corto del LED a GND.





No es necesario llamar a la función interna `pinMode` cuando utilizas `analogWrite`. Así que puedes dejar la configuración vacía aquí.

Después de cargar el código de abajo, verás que el LED se va iluminando poco a poco hasta llegar a su máxima intensidad, y luego se repite.

```
1  int ledPin = 10;
2  int fade = 0;
3
4  void setup() {
5  // nothing here
6  }
7
8  void loop() {
9  analogWrite(ledPin, fade);
10 delay(10);
11 fade = fade + 10;
12 if (fade > 255) fade = 0;
13 }
```

Aquí se ha utilizado un nuevo comando:

- `analogWrite(pinNumber, fadeLevel)`: escribe una señal analógica a un Pin PWM. `pinNumber` es el número de Pin PWM que estás utilizando, `fadeLevel` es un número entre 0 y 255. 0 es igual a digital `LOW`, 255 es igual a digital `HIGH`.

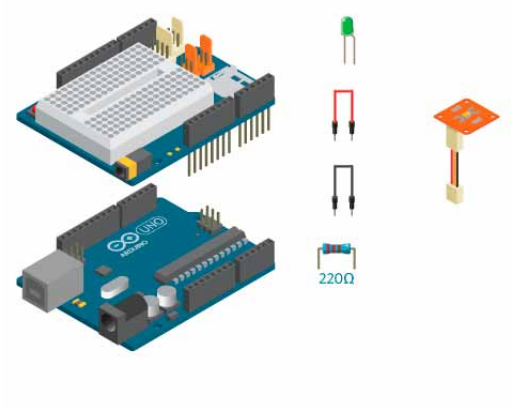
¡Sigue experimentando!

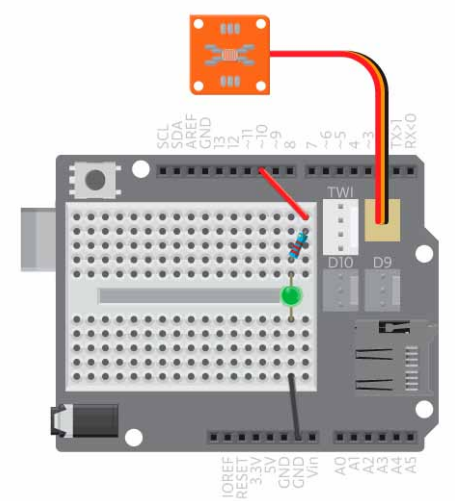
- Intenta hacer que el LED se desvanezca a negro después de que alcance el brillo completo, en lugar de quedarse oscuro de repente. A esto lo llamamos una luz de respiración (breath).
- ¿Se puede cambiar la velocidad de la "respiración" para que se desvanezca más rápido o más lento?
- ¿Se puede utilizar un potenciómetro para controlar el LED? Recuerda la diferencia entre `analogRead()` y `analogWrite()`.

LDR

Ahora vamos a aprender acerca de un sensor analógico real. El LDR, es una resistencia dependiente de la luz (sus siglas provienen del inglés ***Light Dependent Resistor***), detecta la luminosidad y según la cantidad de luz que le llegue, el sensor devolverá un valor analógico diferente. Puedes hacer cosas muy interesantes con ella, como una lámpara que se enciende automáticamente cuando la habitación se oscurece o que un robot siga una linterna... pero vamos a empezar por lo básico.

Consigue un sensor LDR TinkerKit y un cable TinkerKit y conéctalos entre sí. Luego conecta el otro extremo del cable al puerto analógico de tres pins TinkerKit A1 en la shield Básica Educativa.





Ahora coge un LED y una resistencia de 220 ohm, conecta el LED al Pin 10 como se describió en ejercicios anteriores.

Sube el código que sigue, y experimenta un poco con el sensor LDR. Apúntalo a la luz o cúbrelo con tu mano. Observa cómo se comporta el LED.

```
1   int ledPin=10;
2   int ldrPin=A1;void setup() {
3   //nothing here
4   }void loop() {
5   int ldrValue=analogRead(ldrPin);
6   int ledValue=map(ldrValue,0,1023,0,255);
7
8   analogWrite(ledPin, ledValue);
9   delay(10);
10  }
```

Los comandos usados son:

- `map(valor, desdeBajo, desdeAlto, hastaBajo, hastaAlto)`: Reasigna un número de un rango a otro. `valor` es el valor que queremos reasignar.`desdeBajo` y `desdeAlto` son el valor mínimo y máximo que el valor puede tomar. En este caso desde 0 a 1023. `hastaBajo` y `hastaAlto` son el mínimo y el máximo valor que queremos. En este caso, desde 0 hasta 255.

Utilizamos `analogRead()` para leer el valor del LDR, cuyo rango va de 0 a 1023. Entonces usamos la función `map()` para transformar dicho valor de forma que el nuevo sea proporcional a un valor entre 0 y 255. Si `ldrValue` es 1023, `ledValue` será 255, si `ldrValue` es 512 `ledValue` será 127. Una vez el valor haya sido reasignado, lo utilizamos para encender el LED. El LED se hará oscuro o luminoso dependiendo de la cantidad de luz que el LDR detecte.

¡Sigue experimentando!

- Intenta hacer la lámpara automática que hemos mencionado anteriormente. Cuando la lectura del LDR es más baja que un valor concreto o umbral, la luz se enciende. De lo contrario, se apaga. Utiliza un LED para simular la lámpara.
- ¿Recuerdas el ejemplo Beep de la semana pasada? ¡Haz que funcione con un LDR! Recuerda asignar los valores correctamente.

Calibración de Sensores

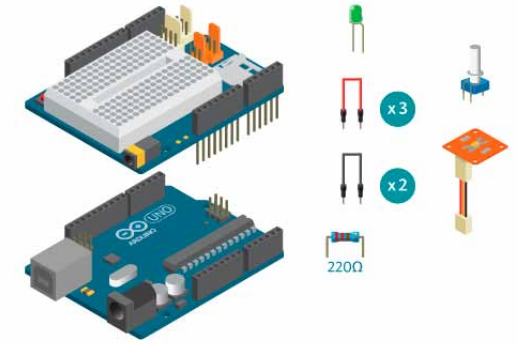
El LDR es un sensor fuertemente influenciado por configuraciones exteriores. Es decir, un proyecto con LDR funcionará de manera diferente en diferentes sitios porque la configuración lumínica es diferente. En ocasiones funcionará a la primera pero a veces tendremos que calibrarla para que funcione en lugares diferentes.

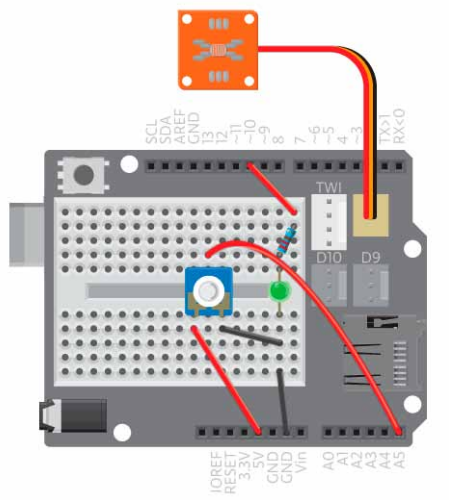
Coge la lámpara automática como ejemplo. Digamos que la queremos utilizar para otro propósito. Esta vez, se encenderá si le colocas un trozo de papel sobre la LDR.

Necesitarás experimentar un poco, intenta buscar el umbral para un trozo de papel. ¿Has conseguido que funcione? Muy bien, mueve el LDR por debajo de la mesa, ¿todavía funciona?

Si es que no, probablemente ya sabemos cuál es el problema. El papel no ha cambiado mucho la lectura de la LDR, mientras que la iluminación de tu habitación sí lo ha hecho (al mover el sensor bajo de la mesa). Por lo que necesitarás encontrar una manera mejor de cambiar el umbral dinamicamente.

Ahora busca un potenciómetro y conéctalo a la entrada analógica A5 como en ejercicios anteriores. Reasigna el valor del `analogRead()` del potenciómetro a 0~255 y utiliza esto como el umbral.






```
1   int ledPin=10;
2   int ldrPin=A1;
3   int potPin=A5;
4
5   void setup() {
6     pinMode(ledPin,OUTPUT);
7   }
8
9   void loop() {
10    int ldrValue=analogRead(ldrPin);
11    int threashold=analogRead(potPin);
12
13    if(ldrValue>threshold){
14      digitalWrite(ledPin,LOW);
15    }else{
16      digitalWrite(ledPin,HIGH);
17    }
18    delay(10);
19  }
```

Ahora, en lugar de ajustar el umbral en el código, puedes hacerlo girando el potenciómetro hasta que el LDR detecte el papel. Prueba a ponerlo otra vez bajo de la mesa y verás cómo de simple y práctico puede ser un potenciómetro que reasigne los valores.

Acabas de aprender una manera de calibrar sensores analógicos, ¡utilízalo cuando construyas tus propios proyectos!

Puerto serie

Las placas Arduino se conectan a tu ordenador usando un cable USB. El modo en que las placas "hablan" con el ordenador consiste en una técnica llamada Puerto Serial o Puerto Serie. Este se puede usar para intercambiar datos relativamente complicados entre Arduino y el ordenador. En lugar de señales digitales o analógicas puedes enviar o recibir texto (string).

Mediante comunicación serie puedes comunicarte también con otros programas. Puedes por ejemplo, utilizar Arduino para leer el estado de un botón y mandar los datos a un sketch de processing que cambie el color de la pantalla cuando el botón esté presionado.

El puerto serie usa los Pins digitales 0 y 1:

El Pin 0, RX o recepción, por aquí llegan los datos a Arduino.
El Pin 1, TX o transmisión, por aquí salen los datos desde Arduino.

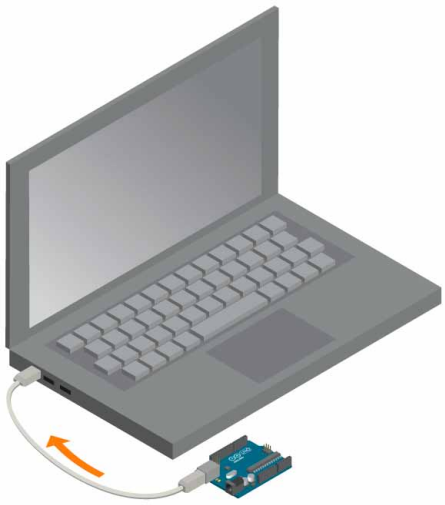
Por lo tanto, no uses la función `digitalRead()` ni `digitalWrite()` en esos Pins si vas a utilizar comunicación serie.

Enviando al ordenador

Instrucciones

Para enviar un mensaje al ordenador, necesitarás tres comandos diferentes: `Serial.begin()`, `Serial.println()` o `Serial.print()`.

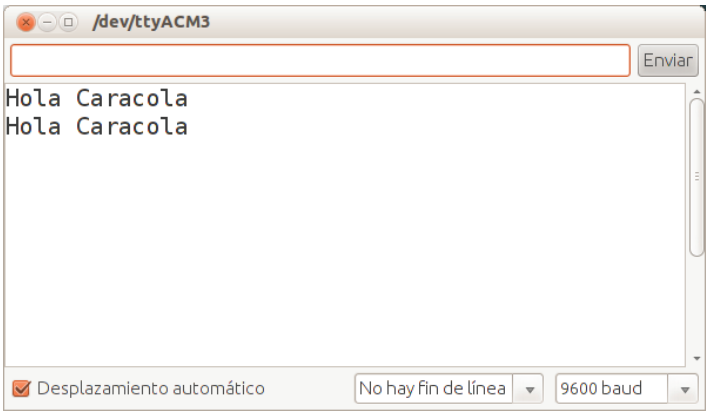
Carga el código de abajo en tu placa y haz click en Herramientas -> Monitor Serial en el IDE de Arduino.



```
1  void setup() {
2  Serial.begin(9600);
```

```
3  }
4
5  void loop() {
6  Serial.println(“Hola Caracola”);
7  delay(1000);
```

Verás una pequeña ventana con el texto “Hola Caracola”, apareciendo una vez por segundo. Esta ventana muestra todo lo enviado a tu ordenador a través del puerto serial de Arduino.

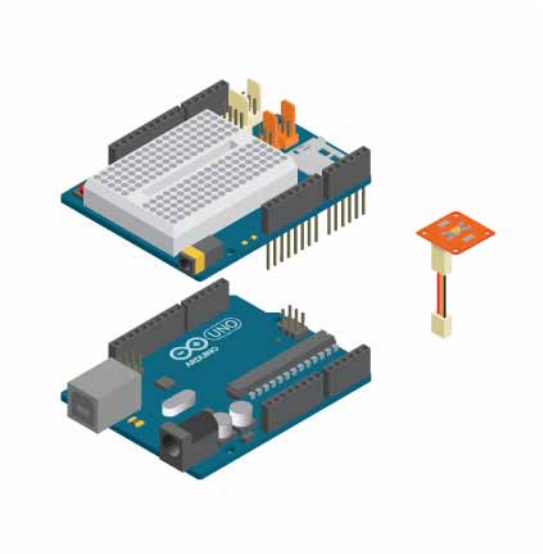


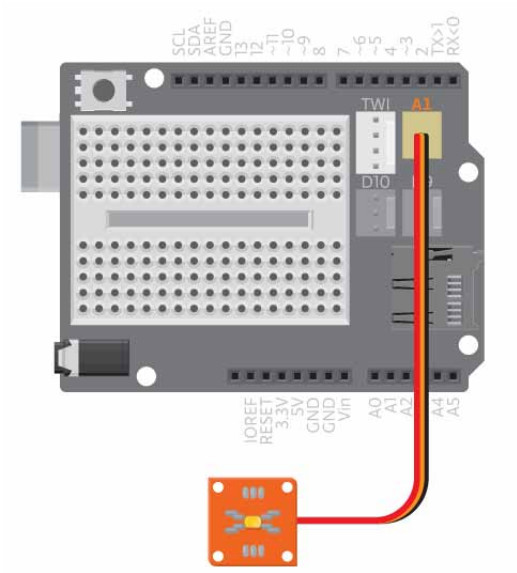
Los comandos utilizados arriba son:

- `Serial.begin(velocidad)`: Inicia la comunicación serial. Velocidad es lo rápido que los mensajes son transmitidos. Es importante que tanto el emisor (en este caso la placa Arduino) y el receptor (en este caso el monitor serial) se comuniquen a la misma velocidad. De otra manera, sería como dos personas que intentan hablar, uno chino y el otro en español. Por ahora, utiliza 9600. Esta es la velocidad más común cuando se trabaja con Arduino y Processing.
- `Serial.println(mensaje)`: Muestra un mensaje a través del puerto serial, con un salto de línea al final. Así que cada vez que llames a la función `Serial.println()`, el mensaje siguiente se iniciará en una nueva línea. El `mensaje` es una cadena de texto (tipo `String`), reemplázala con el texto que quieras enviar.
- `Serial.print(mensaje)`: Muestra un mensaje a través del puerto serial. Así que cada vez que llames a `Serial.print()`, el siguiente mensaje aparecerá justo después, sin nueva línea. El `mensaje` es una cadena de texto reemplázala con el texto que quieras enviar.

Enviando valores del LDR

La cosa más importante en la que utilizarás la comunicación serial es para testear tus programas. De esta manera, más que mandar mensajes estáticos por el puerto serial, puedes mandar valores dinámicos que cambien con el tiempo. Esto es útil cuando quieras usar un sensor analógico pero que no sabes exactamente qué valores lee.





¿Te acuerdas del ejemplo anterior donde utilizábamos un potenciómetro para calibrar nuestro programa LDR? En lugar de ello, ahora vamos a mostrarte cómo usar la comunicación serial para hacerlo.

En este ejemplo leeremos el valor analógico del LDR e imprimiremos el valor por el monitor serial. Conecta un TinkerKit LDR al puerto de tres pines A1. Carga el siguiente código y abre el monitor serial.

```
1  void setup() {
2  Serial.begin(9600);
3  }void loop() {
4  int lecturaSensor = analogRead(A1);
5  Serial.println(lecturaSensor);
6  delay(1000);
7  }
```

No se necesita ningún nuevo comando aquí. Leemos el valor del LDR conectado al puerto A1 e imprimimos dicho valor por el monitor serial. El `delay()` está aquí sólo para hacer más fácil la lectura de los datos. Cuando tengas una buena idea sobre qué valores obtendrás, sabrás que valor utilizar como umbral.

Nota: Acuérdate de que el conector de tres pins A1 está realmente conectado al pin A1, por lo que no puedes usar ningún otro sensor que ese pin analógico.

Recibiendo del ordenador

Puedes utilizar el comando `Serial.read()` para obtener datos que llegan a Arduino a través del puerto serial.

```
1  int ledPin=13;
2  int incomingByte;
3
4  void setup() {
5  Serial.begin(9600);
6  pinMode(ledPin,OUTPUT);
7  }
8  void loop() {
9  if(Serial.available()>0){
10 incomingByte=Serial.read();
11 if(incomingByte=='H'){
12 digitalWrite(ledPin, HIGH);
13 }
14 if(incomingByte=='L'){
15 digitalWrite(ledPin,LOW);
16 }
17 }
18 }
```

Abre el monitor del puerto serie desde Herramientas -> Monitor Serial o con el icono al final de la barra de tareas. Hay un cuadro de entrada de texto con el botón enviar al lado. Escribe H en el cuadro de entrada y haz clic en el botón enviar. Deberías ver que se enciende el LED conectado al Pin 13 de tu Arduino. Teclea L y envía. El LED se apagará. Asegúrate de utilizar mayúsculas.

Los comandos utilizados son:

- `Serial.available()`: Comprueba si hay señales procedentes del puerto serial. Devuelve bien `true` (verdadero) o `false` (falso). Lee las señales sólo cuando estén disponibles.
- `Serial.read()`: Lee un byte del puerto serie.

Sólo queremos leer del puerto serial cuando haya datos entrantes. Por tanto, primero comprobamos con `Serial.available` si hay algo que leer. En ese caso, leemos con `Serial.read()` y lo guardamos en `incomingByte`. Luego comprobamos si `incomingByte` es igual a 'H' o 'L' y encendemos o apagamos el LED conforme a esto.

¡Sigue experimentando!

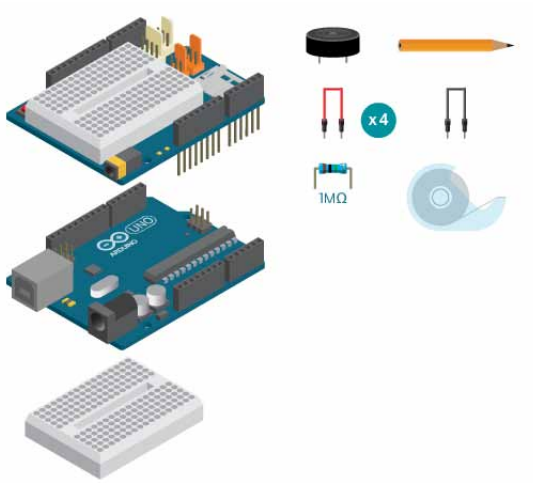
- Consulta cualquiera de los ejemplos en Archivo –> Ejemplos –> Castilla –> Help, casi todos hacen uso del puerto serie. Por ejemplo, controla el programa breath light con las señales del puerto serial.
- Si dispones de dos baterías de 9V, haz que un Arduino controle al otro. El Arduino que recibe las señales debe tener el código anterior. El Arduino que envía debe mandar las señales "H" o "L" alternativamente.

Drawdio

Dibuja un sonido con drawdio. Quizás pienses que este lápiz es mágico, y simplemente puede serlo. Drawdio convierte (casi) todo que sea conductivo en un instrumento musical. Puedes hacer música dibujando o tocando un punto de diferentes cosas conductivas a tu alrededor. Mira algunos ejemplos de cómo utilizarlo en esta página (en inglés).

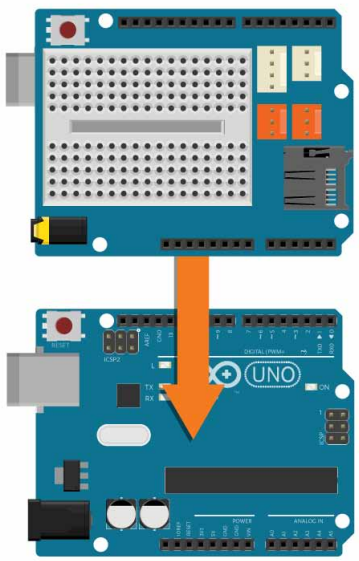
Materiales

- 1 placa Arduino Uno
- 1 Shield Básica Educativa
- 1 piezo
- 1 resistencia de 1MOhm
- 1 cable negro
- 4 cables de colores
- 1 lápiz 9B
- cinta adhesiva
- 1 breadboard

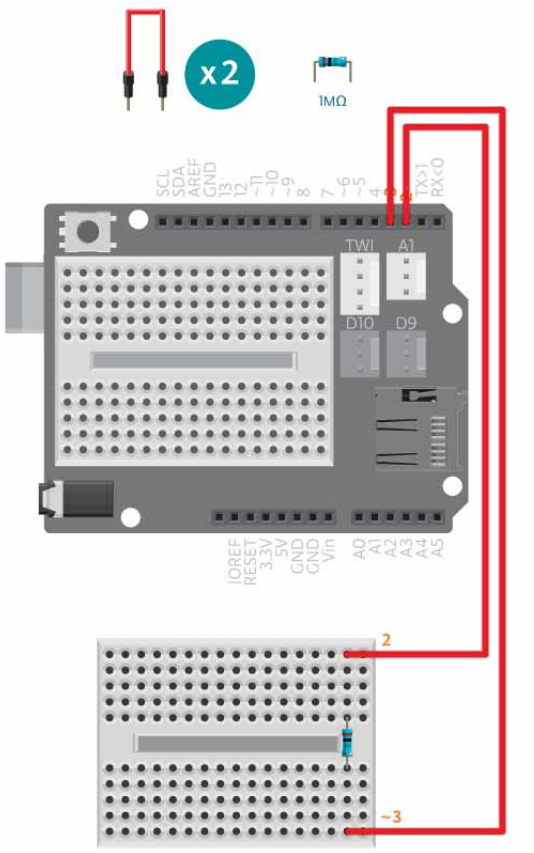


Instrucciones

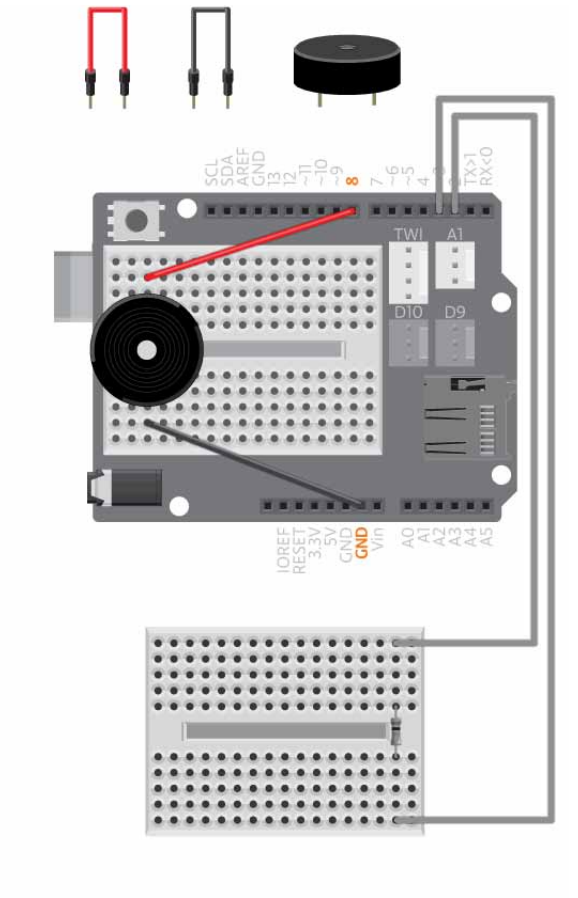
1. Conecta la shield a la parte superior de tu placa Arduino.



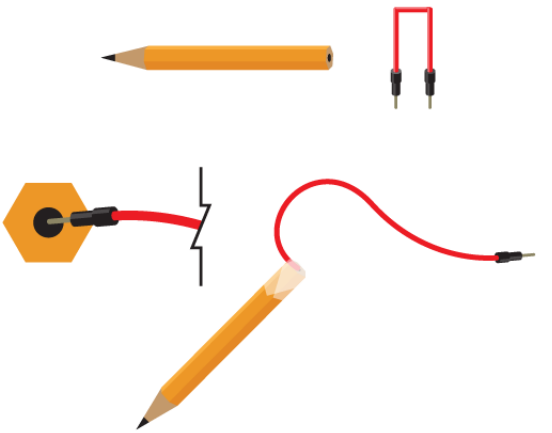
2. Conecta una resistencia de 1MOhm entre los Pins digitales 2 y 3.



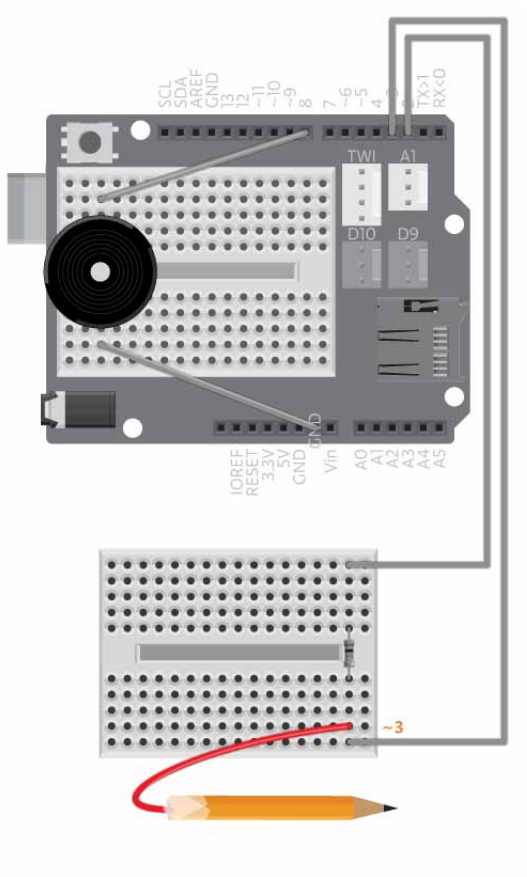
3. Pon el piezo en la breadboard y conecta una de sus patas al Pin 8 y la otra a GND.



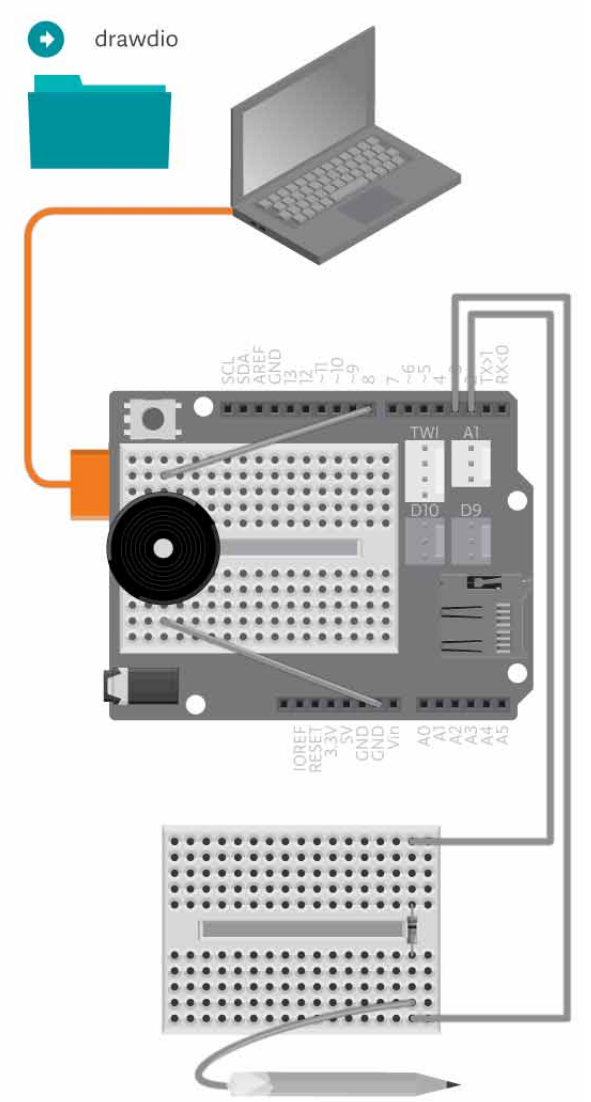
4. Pega el extremo de un cable a una parte expuesta del grafito del lápiz, asegúrate de que hacen contacto.



5. Conecta el otro extremo del cable al Pin 3.



6. Conecta tu Arduino al ordenador y carga el ejemplo Drawdio.



Código

Puedes encontrar el código en Archivo -> Ejemplos -> BasicEducationShield-> Magic -> Drawdio

```
/*
  Drawdio

  Draw audio with drawdio. You might think that this pen is magic,
  and it might just be. Drawdio turns (almost) everything that is
  conductive into an instrument. You can either make music by
  drawing a picture or by touching the tip of it to different
  conductive things around you.

  (c) 2013 Arduino Verkstad
  Inspiration from Jay Siver's DRAWDIO
  http://web.media.mit.edu/~silver/drawdio/

*/

#include <CapacitiveSensor.h>
#include "pitches.h"
#include <BasicEducationShield.h>

//Capacitive switch connected between 2 and 3
CapacitiveSwitch me=CapacitiveSwitch(2,3);

//A piezo speaker connected to digital pin 8
Melody speaker=Melody(8);

//The range of capacitive sensor values to be
//mapped to music tones. See example
//"help/CapacitiveSwitchTest" about how to get
//the values
int lowerThreshold=80;
int upperThreshold=900;

//Define the lowest and highest tone. Defined
//in pitches.h
int toneLow=NOTE_C3;
int toneHigh=NOTE_B6;

void setup(){
  //Nothing to do here. Magic?
}
void loop(){
  //Get value of the sensor. If it's smaller
  //than threshold, it'll be 0
  int value=me.getValue(lowerThreshold);

  //map the sensor value to tones. The map()
  //function maps values smaller than lower
```

```
48   //threshold to the lowest desired value. If
49   //you want to keep drawdio quiet when you're
50   //not playing, you have to deal with 0 separately
51   int pitch;
52   if(value>0){
53     pitch=map(value,lowerThreshold,upperThreshold,toneLow,toneHigh);
54   }else{
55     pitch=0;
56   }
57
58   //Play the tone to the speaker.
59   speaker.playTone(pitch,10);
60 }
```

¿Cómo funciona?

Drawdio mide la capacidad de los objetos. El programa mapea los valores de capacidad en su entrada a la frecuencia de un sonido. Para cualquier valor por debajo de un umbral, no habrá sonido.

¿No funciona?

1. Los objetos metálicos son conductores pero pueden no funcionar para Drawdio. El agua, la fruta, los humanos, las plantas y los lápices funcionan mejor para Drawdio. Usa Archivo -> Ejemplos -> BasicEducationShield-> Help -> CapacitiveSwitchTest para experimentar con la capacidad de los diferentes objetos que te rodean.

2. Si usas lápices de dibujo, asegúrate de que las líneas y las formas son anchas y gruesas. Intenta tocar uno de los extremos del objeto cuando utilices Drawdio.

3. Mira la referencia del sensor capacitivo para más explicaciones.

¡Sigue experimentando!

- Dibuja un instrumento molón con Drawdio y toca música con el.
- Afina el tono de Drawdio para que sea más preciso con los objetos que toca.
- Cambia los sonidos que Drawdio toca.
- ¡Utiliza el material más insólito que encuentres para Drawdio!

POV

Con POV, persistencia de visión (del inglés Persistence Of Vision), puedes, por ejemplo, convertir una rueda de bici en un display con tan solo unos pocos LEDs. Mira este video para entenderlo mejor.

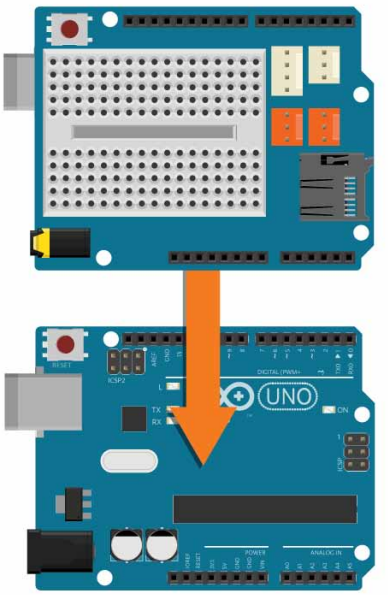
En este proyecto harás uno de estos displays por ti mismo. Haz que dibuje largas líneas o que incluso muestre texto, un patrón o una imagen mientras pedaleas. El efecto es más visible en la oscuridad, por lo que si vas en bici, ten cuidado de no chocar contra un árbol o un amigo.

Materiales

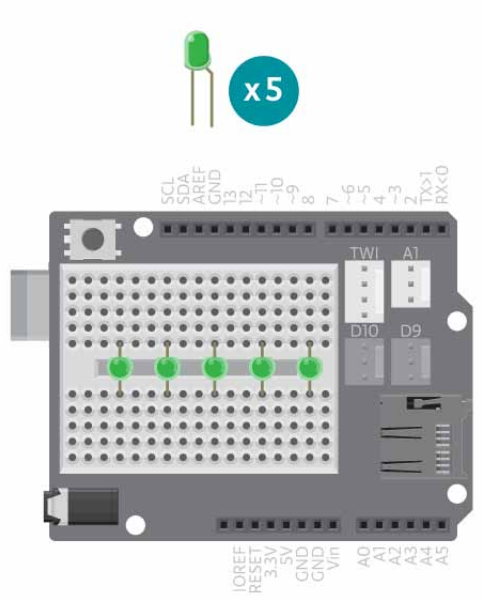
- 1 placa Arduino Uno
- 1 Shield Básica Educativa
- 5 LEDs
- 5 resistencias 220 Ohm
- 5 cables negros
- 5 cables de colores
- 1 pila 9V
- 1 portapilas 9V

Instrucciones

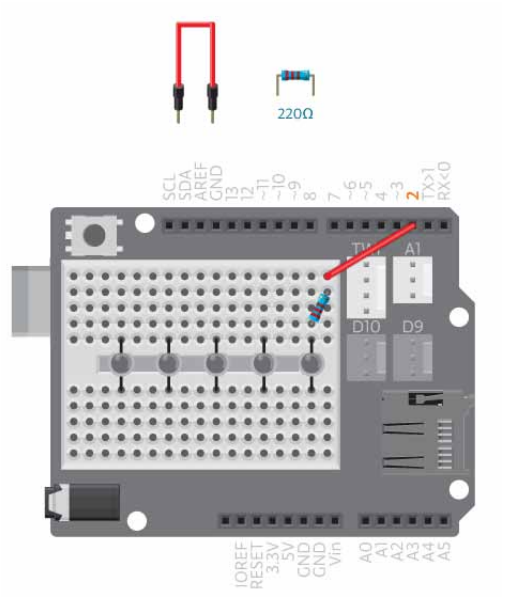
1. Coloca la shield en la parte superior de tu Arduino.



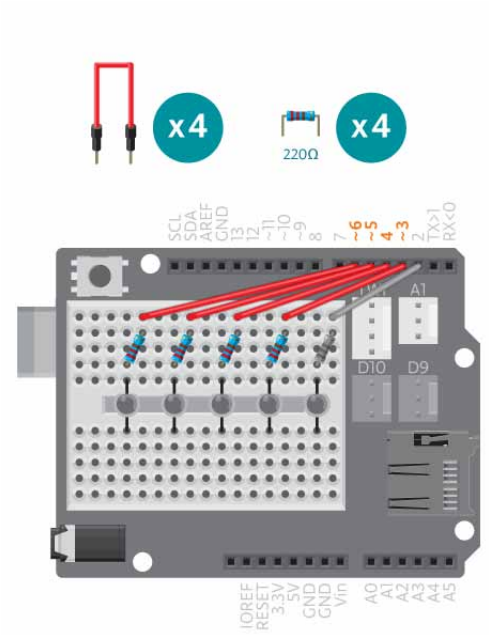
2. Conecta cinco LEDs a través del puente de la breadboard.



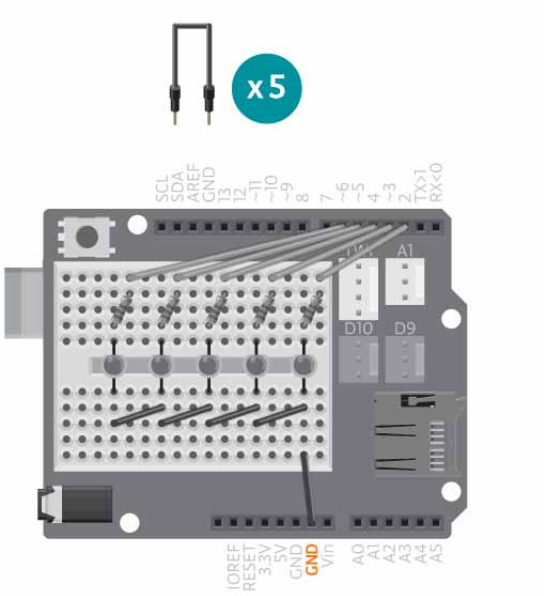
3. Conecta una resistencia de 220 Ohm a la pata larga del primer LED. Conecta la otra pata de la resistencia con un cable, al Pin digital 2.



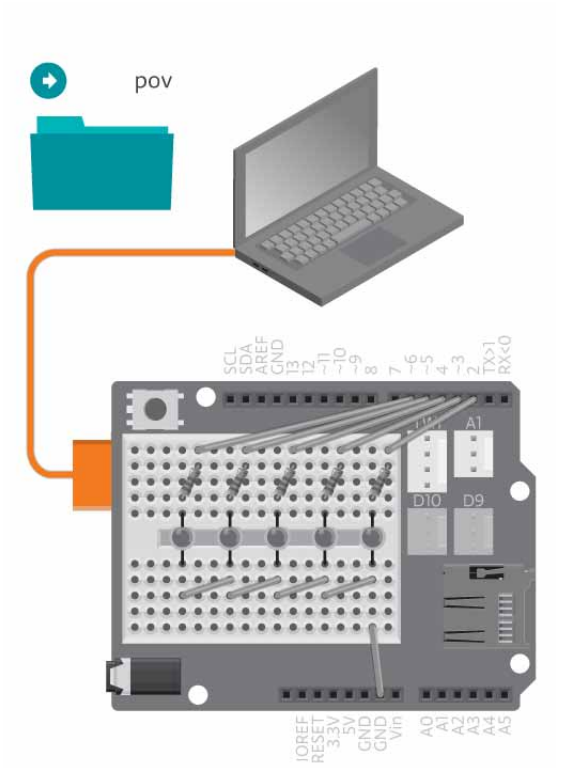
4. Repite el paso anterior para que los LEDs queden conectados desde el Pin 3 al Pin 6.



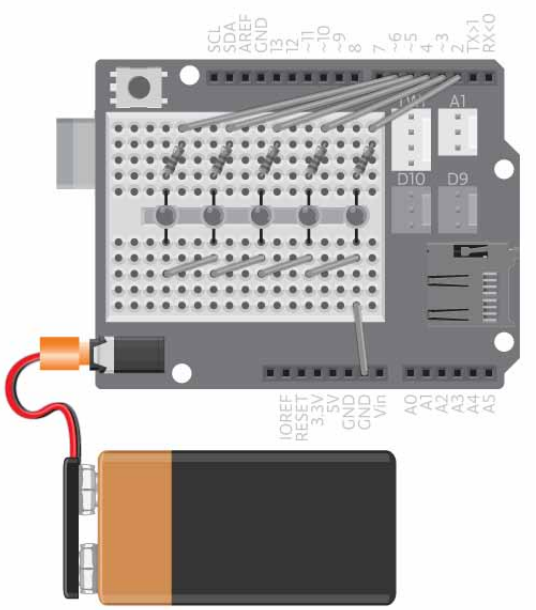
5. Conecta las patas cortas de todos los LEDs con cables negros a GND.



6. Conecta tu Arduino al ordenador y carga el ejemplo POV.



7. Desconecta tu Arduino del puerto USB y conecta la batería de 9V al portapilas y, a continuación, a tu Arduino.



Código

Puedes encontrar el código en Archivo -> Ejemplos -> BasicEducationShield -> Magic -> POV.

```
/*
  POV (Persistence of Vision)

  With POV, persistence of vision, you can take eg a bike wheel
  and make it into a display with just a few LEDs. Check out
  this video to see what we mean:
  http://www.youtube.com/watch?v=-TvGvVWS3IE

  In this project you will make one of these displays yourself.
  Wave it with long strokes or even hold it while running to
  display a text, a statement or a picture. The effect is most
  visible in a dark place so if you're running, make sure not
  to run in to a tree or a friend.

  (c) 2013 Arduino Verkstad
*/

```

```
#include <BasicEducationShield.h>
/*
An array of pin numbers to which LEDs are attached
the defaults are 2 to 6 but you can choose any of the digital pins
*/
int ledPins[] = {2, 3, 4, 5, 6};
int pinCount = 5;
VUMeter vuMeter;

int rowCount = 0; // Stores the number of rows
int rowLength = 22; // Width of the message, copy this number to the message array
int delayTime = 9; // Time it takes to show a row in milliseconds

// The message where 0 is LOW and 1 is HIGH
boolean message[5][22]={
  // H H H H   O O O O   L L L L   A A A A
  {0,1,0,0,1,0,0,1,1,0,0,1,0,0,0,0,0,1,1,0,0,0},
  {0,1,0,0,1,0,1,0,0,1,0,1,0,0,0,0,1,0,0,1,0,0},
  {0,1,1,1,1,0,1,0,0,1,0,1,0,0,0,0,1,1,1,1,0,0},
  {0,1,0,0,1,0,1,0,0,1,0,1,0,0,0,0,1,0,0,1,0,0},
  {0,1,0,0,1,0,0,1,1,0,0,1,1,1,1,0,1,0,0,1,0,0}

};

void setup(){
  // If your are using other pins than 2 to 6 you need to configure that here
  vuMeter.config(pinCount, ledPins);

  vuMeter.begin(); //does the same as pinMode, LEDs are outputs
}

void loop(){
 // If the whole array has been drawn
  if(rowCount == rowLength) {
    rowCount = 0; // Reset the rowCount
  } else {
    // Shows the message
    for (int i = 0; i < pinCount; i++) {
      // Checks if the array says HIGH
      if (message[i][rowCount] == 1) {
        vuMeter.on(i);
      } else {
        vuMeter.off(i);
      }
    }
    rowCount++;
  }

  delay(delayTime);// This is the delay per row

}
```

¿Cómo Funciona?

El programa hace parpadear los LEDs de acuerdo a la información del array llamado message. El efecto de persistencia visual se consigue al hacer que el VU-metro se mueva conforme a este parpadeo.

¿No funciona?

1. ¿El VU-metro no funciona correctamente? Mira la referencia del VU-metro para corregir posibles errores.

¡Sigue experimentando!

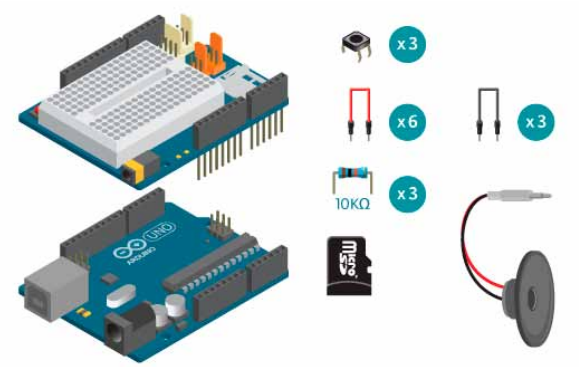
- Prueba otros mensajes con una longitud diferente.
- Cambia el mensaje a un patrón o un dibujo.
- Hazte una foto agitando el POV. ¡Mira el efecto que hace!

Boombox

¡Es hora de poner en práctica tus habilidades de DJ con el BoomBox mágico y adquirir habilidades como estas! El BoomBox es un reproductor de sonido sencillo que puedes usar para tocar música, o simplemente reproducir cualquier sonido. Viene con tres sonidos pregrabados, ¡pero estamos seguros que tú puedes grabar sonidos mucho mejores!

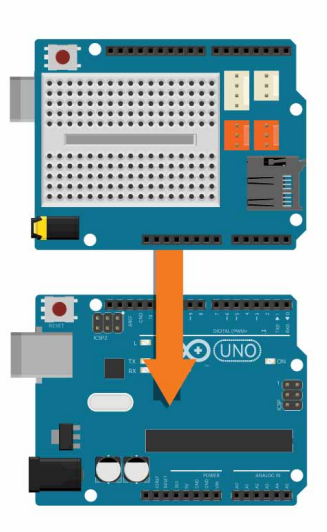
Materiales

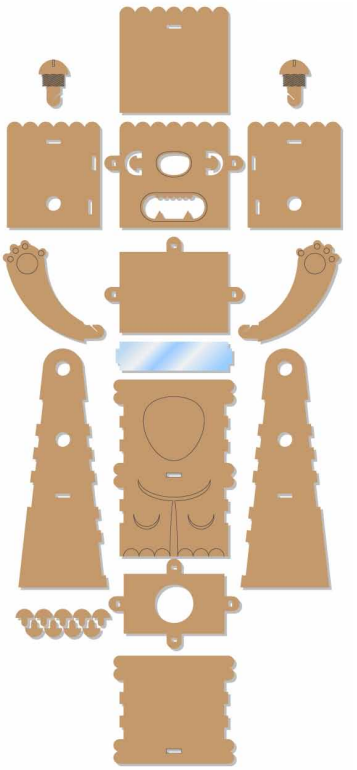
- 1 placa Arduino Uno
- 1 Shield Básica Educativa
- 1 altavoz
- 1 tarjeta micro SD
- 3 botones
- 3 resistencias 10kohm
- 3 cables negros
- 6 cables de colores



Instrucciones

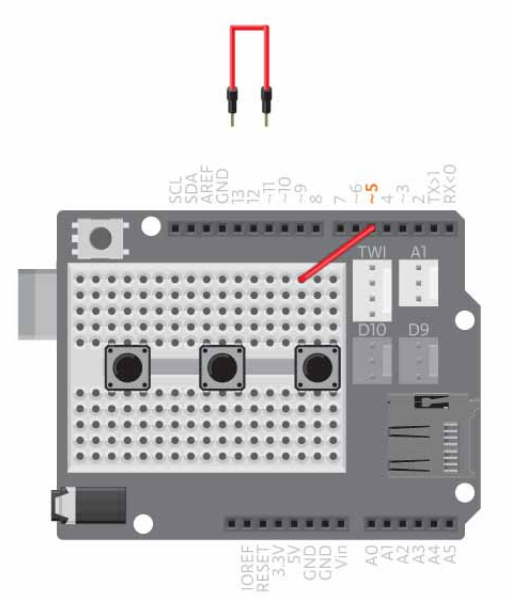
1. Conecta la shield a la parte superior de tu placa Arduino.



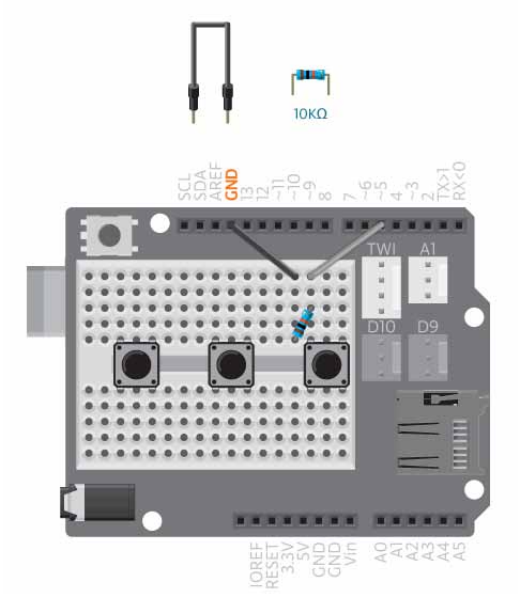


2. Conecta tres botones a través del puente central de la breadboard.

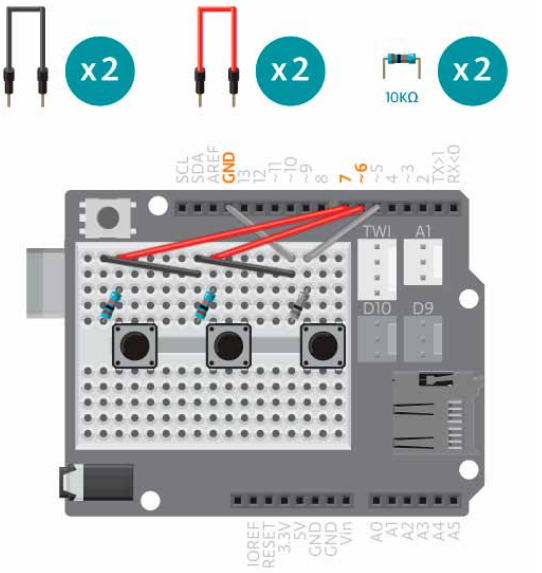
3. Conecta una pata del botón al Pin digital 5.



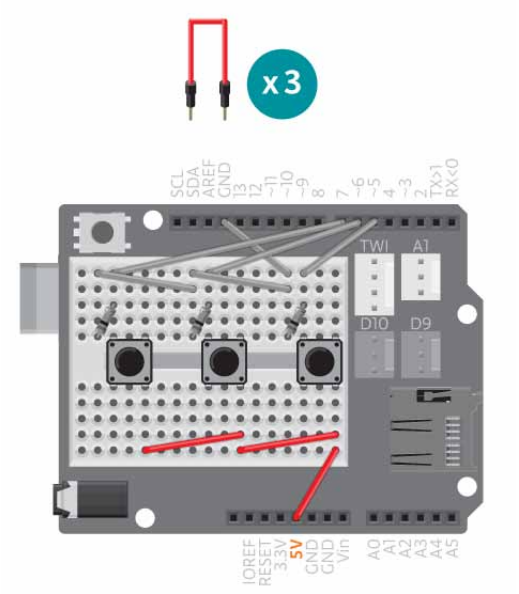
4. Conecta el mismo Pin a una resistencia de 10kohm, y la resistencia a GND.



5. Conecta los otros dos botones a los Pins digitales 6 y 7, añade sus resistencias y cables respectivos.

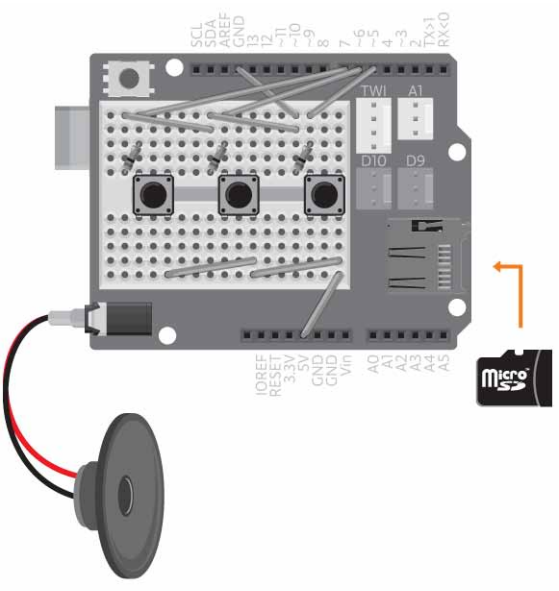


6. Conecta el otro extremo del botón a 5V.

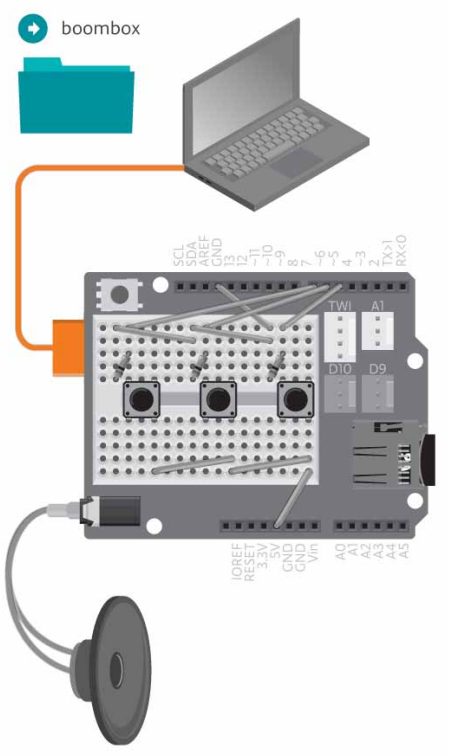


7. Graba 3 piezas sonoras wav con el ordenador, nómbralas 0.wav,1.wav, 2.wav. Guárdalas en el directorio raíz de la tarjeta SD.

8. Inserta la tarjeta SD en la ranura correspondiente del shield. Conecta tu Arduino al ordenador y carga el ejemplo Boombox.



9. Conecta Arduino al ordenador. Carga el ejemplo Boombox y prueba el juego.



Nota: Puedes utilizar Audacity http://audacity.sourceforge.net/) para grabar ficheros de sonido que encajen con Arduino. Para más detalles, mira en la referencia: Prepara Sonidos Wav.

Nota: Utiliza solo el altavoz de papel para la reproducción el audio ya que tendrá mejor calidad que el piezo.

Código

Puedes encontrar el código en Archivo -> Ejemplos -> BasicEducationShield -> Magic -> BoomBox

```
/*
  Boom Box

  This is your first step towards theese kinds of skills:
  http://www.youtube.com/watch?v=FcJCxe1VSLA&noredirect=1
  The boom box is a small sample player you can use to make music,
  or just random sound. It comes with three prerecorded samples
  but we probably don't have the same creative cleverness when it
  comes to recording as you do so you should definitely record your
  own samples too!

  (c) 2013 Arduino Verkstad
*/

#include "BasicEducationShield.h"

//We need to include the SD library to be able to read from an SD card
#include <SD.h>

//Declare a button group with 3 buttons. The
//sound player secrectly takes pin 3, 4 and 11,
//so don't use them,
ButtonGroup bg;
int buttons[]={5,6,7};

//There're 3 buttons in the button group.
int buttonsCount=3;

//Declare the sound player
Player player=Player();

void setup(){
  // Open serial communications and wait for port to open:
  Serial.begin(9600);

  // Initialize the sound player. Open the serial monitor to see
  //the sound files found on your micro SD card
  play er.begin();

  //Initialize the button group.
  bg.begin(buttonsCount,buttons);
```

```
}

void loop(){
  //Wait for one of the buttons to be pressed.
  //According to which button is pressed, it
  //returns either 0, 1 or 2
  int pressedButton=bg.pressed();

  //Play a different sound according to the
  //button pressed.
  switch(pressedButton){
    case 0:
      Serial.println(“sound 0”);
      player.play(“0.wav”);
      break;
    case 1:
      Serial.println(“sound 1”);
      player.play(“1.wav”);
      break;
    case 2:
      Serial.println(“sound 2”);
      player.play(“2.wav”);
      break;
  }

}
```

¿Cómo funciona?

`Player` permite a Arduino reproducir un sonido .wav desde la tarjeta SD.

El programa espera primero a que alguno de los 3 botones sea pulsado, y luego reproduce el sonido asociado a él. Después se repite desde el principio.

¿No funciona?

1. Prueba a corregir errores de los botones, mira la referencia para ver cómo se hace.
2. Corrige errores del reproductor de música. Para más detalles, mira la referencia Herramienta Soundwave.

¡Sigue experimentando!

- ¿Puedes añadirle más botones para reproducir más sonidos?

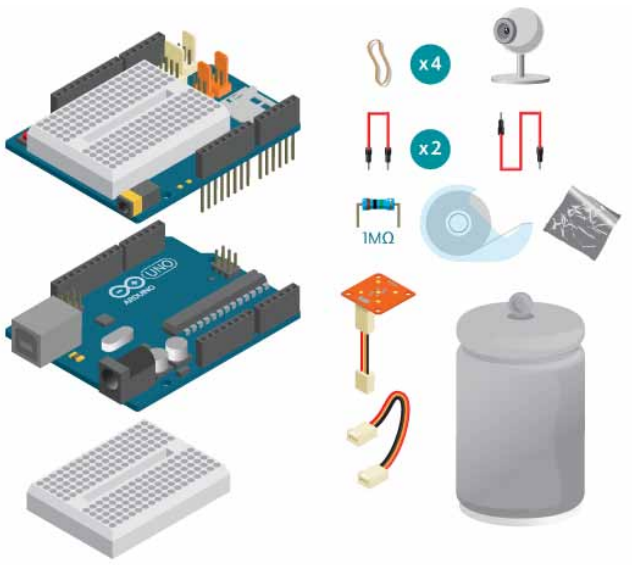
Monstruo de las galletas

¡Hemos descubierto que alguien se ha estado comiendo las galletas del bote de la cocina!
Hemos estado hablando sobre esto y llegamos a la conclusión de que lo mejor va a ser preparar una trampa para descubrir al culpable. Vamos a construir algo que haga una foto a lo que sea que abra el bote de las galletas.

Ningún monstruo de las galletas fue herido en la realización de este experimento.

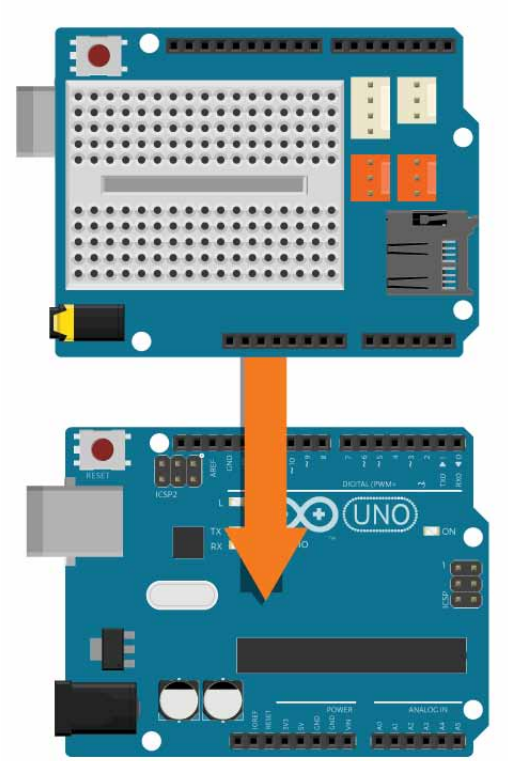
Materiales

- 1 placa Arduino Uno
- 1 Shield Básica Educativa
- 1 resistencia 1Mohm
- 3 cables (1 largo)
- 1 TinkerKit Ultra Bright White LED
- 1 cable TinkerKit
- webcam
- Kit Monstruo de las galletas
- goma elástica
- 1 bote de galletas metálico
- 1 breadboard pequeña

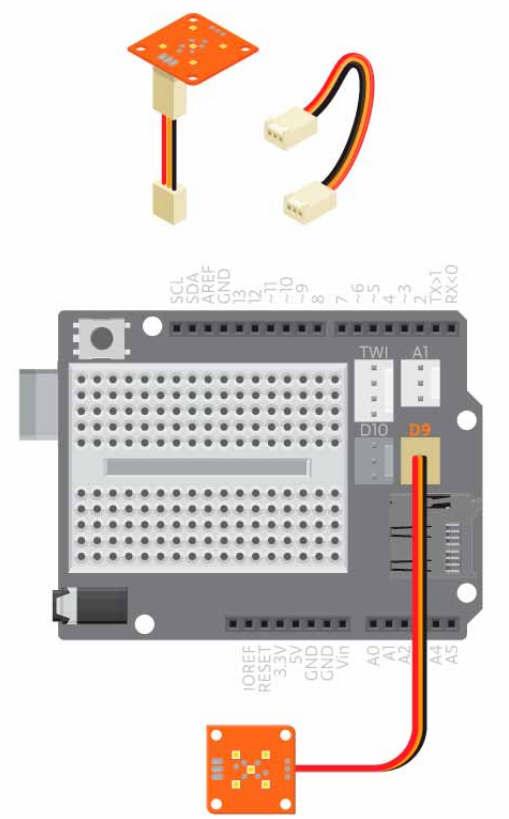


Instrucciones

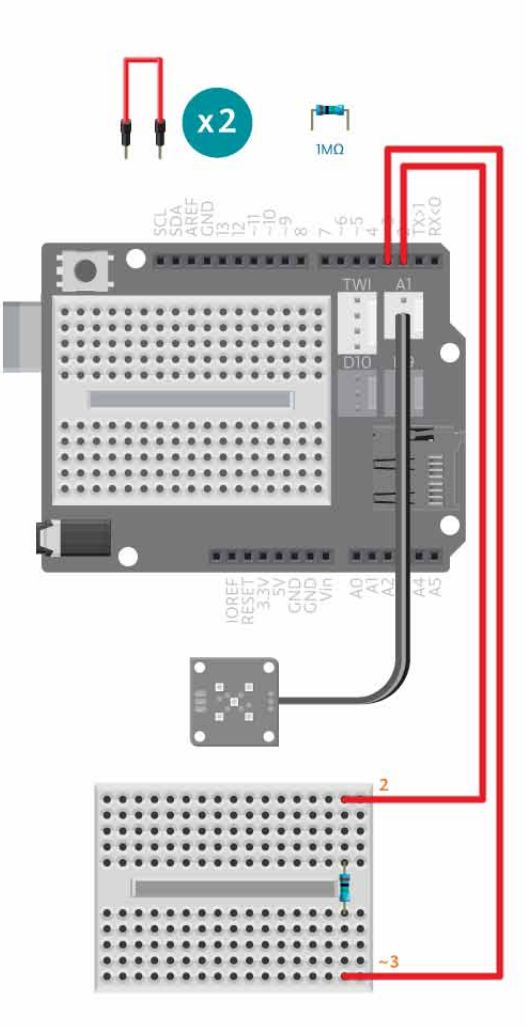
1. Conecta la shield a la parte superior de tu Arduino.



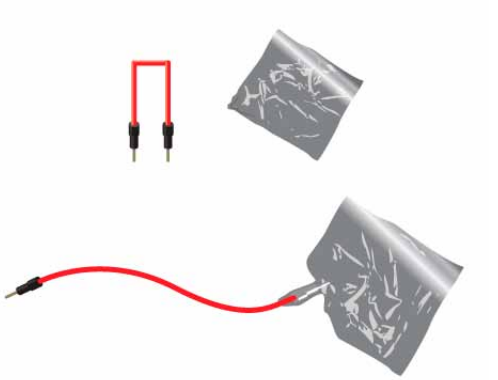
2. Conecta módulo TinkerKit Ultra Bright White LED al puerto digital 9.



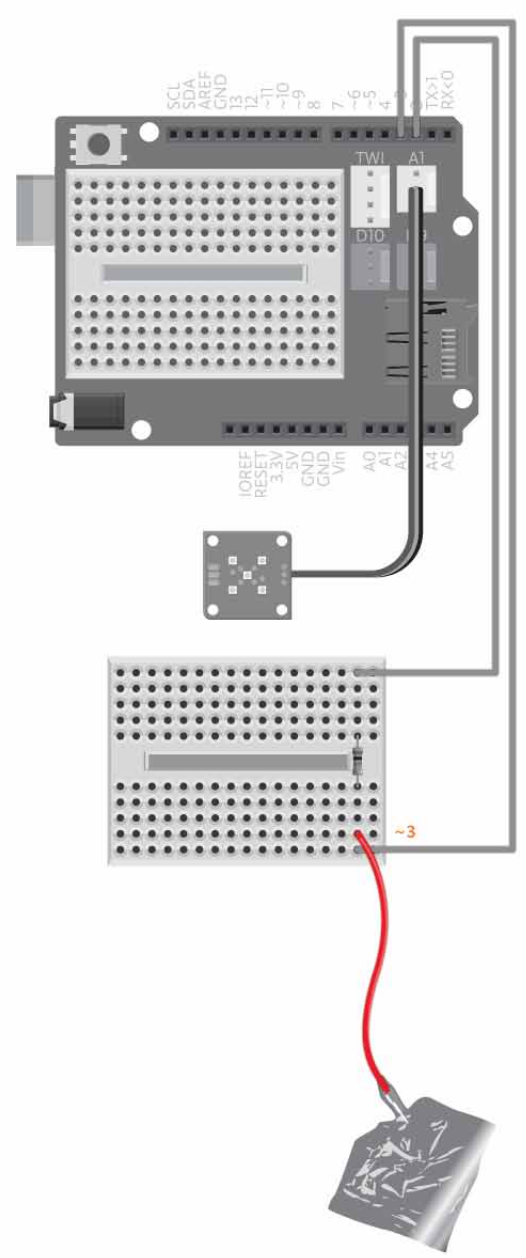
3. Conecta una resistencia de 1Mohm entre el Pin digital 2 y el Pin digital 3.



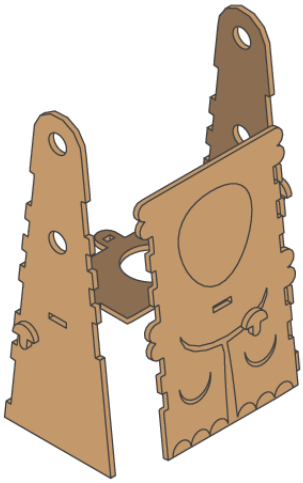
4. Haz un sensor capacitivo cortando un cuadrado de 20 cm (por lado) de papel de aluminio. Enrolla una esquina del cuadrado de papel de aluminio a un cable suelto –el metal del cable debe estar en contacto con el papel de aluminio-.



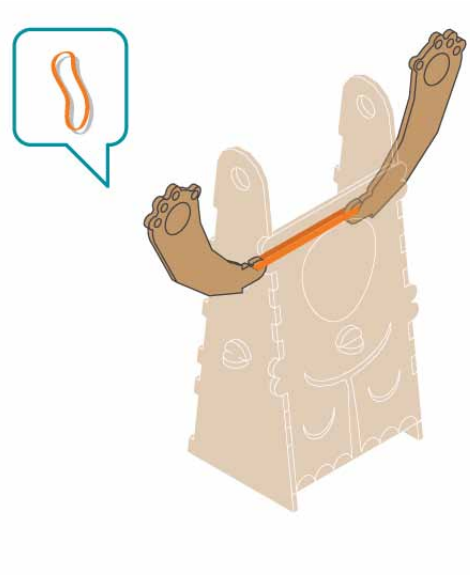
5. Conecta el otro extremo del cable suelto al pin digital 3 sobre la breadboard.



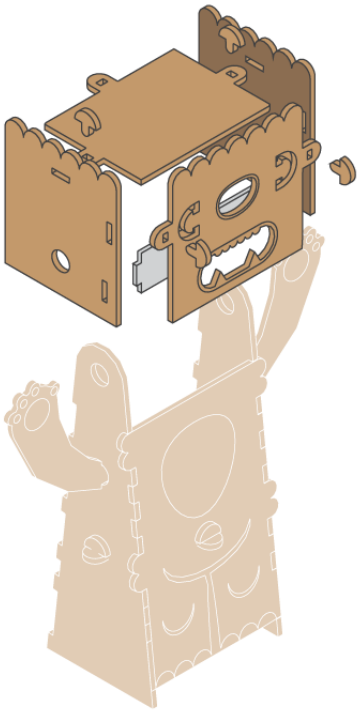
6. Construye el monstruo de las galletas.



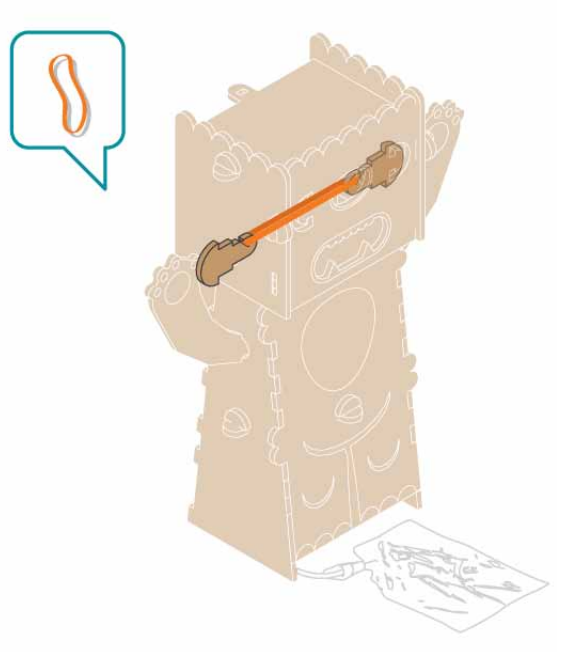
7.



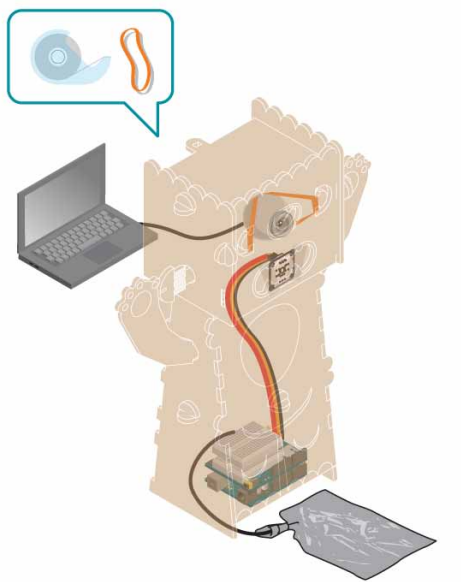
8.



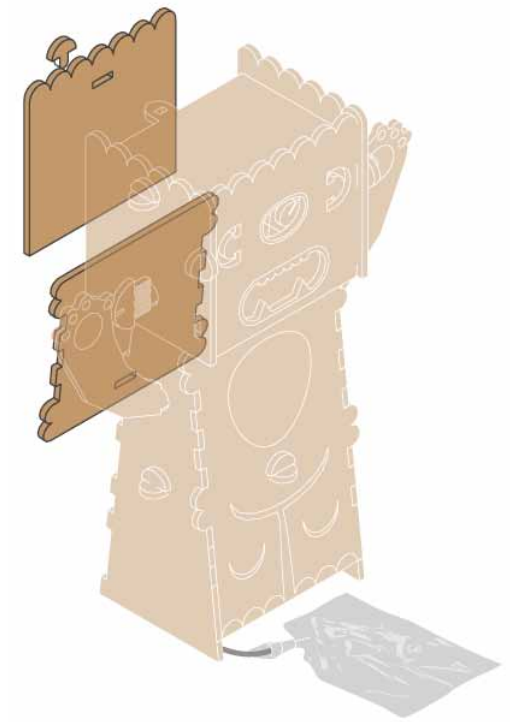
9.



10. Coloca la cámara en la cabeza del monstruo de las galletas. Coloca el módulo Ultra Bright White LED en la boca del monstruo de las galletas.

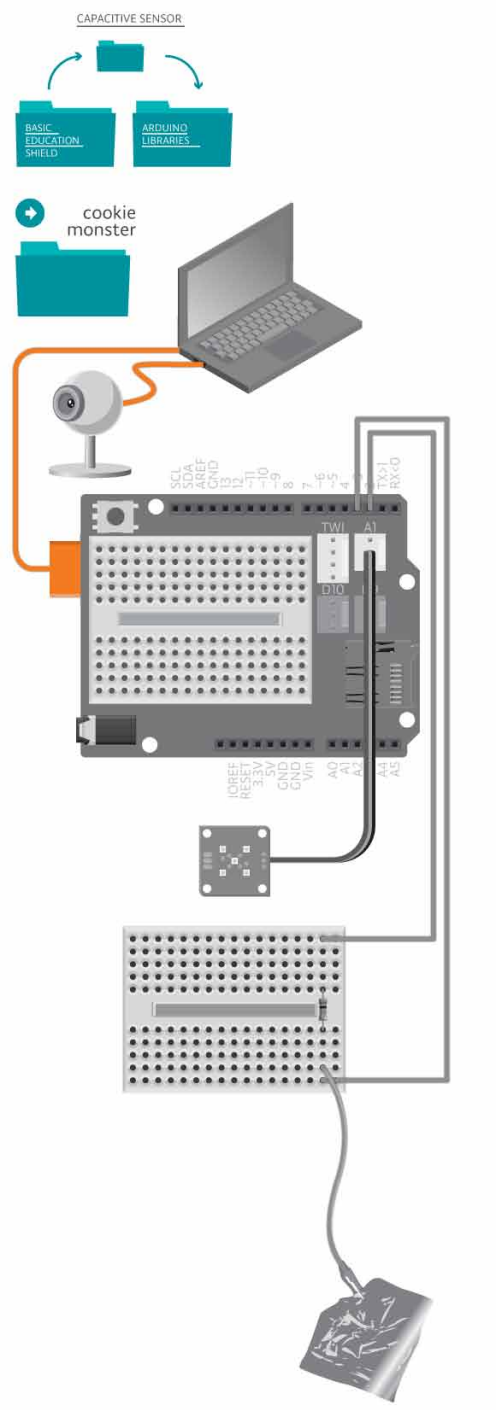


11.

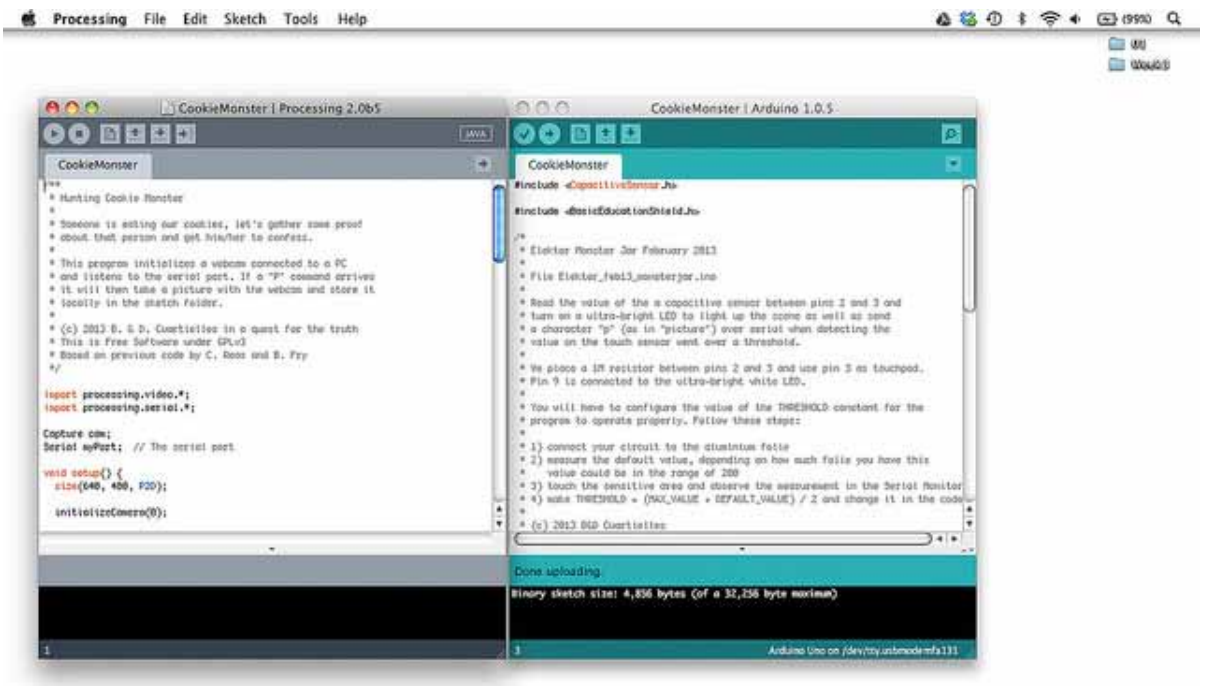


12. Coloca tu Arduino en el interior del cuerpo del monstruo. Coloca el sensor capacitivo en la parte delantera del monstruo de las galletas. Coloca el bote metálico sobre el sensor capacitivo, asegúrate de que las partes metálicas están en contacto.

13. Conecta el Arduino al ordenador y carga el ejemplo CookieMonster.



14. Mueve CookieMonster.pde de la carpeta Processing a la carpeta Processing de las librerías Arduino (Libraries) y ejecútalo desde Processing.



Código

Puedes encontrar el código en: Archivo -> Ejemplos -> BasicEducationShield-> Magic -> CookieMonster

```
1   #include <CapacitiveSensor.h>
2
3   #include <BasicEducationShield.h>
4
5   /*
6    * Elektor Monster Jar February 2013
7    *
8    * File Elektor_feb13_monsterjar.ino
9    *
10  * Read the value of the a capacitive sensor between pins 2 and 3 and
11  * turn on a ultra-bright LED to light up the scene as well as send
12  * a character “p” (as in “picture”) over serial when detecting the
13  * value on the touch sensor went over a threshold.
14  *
15  * We place a 1M resistor between pins 2 and 3 and use pin 3 as touchpad.
16  * Pin 9 is connected to the ultra-bright white LED.
17  *
18  * You will have to configure the value of the THRESHOLD constant for the
```

```
 * program to operate properly. Follow these steps:
 *
 * 1) connect your circuit to the aluminium folie
 * 2) measure the default value, depending on how much folie you have this
 *    value could be in the range of 200
 * 3) touch the sensitive area and observe the measurement in the Serial Monitor
 * 4) make THRESHOLD = (MAX_VALUE + DEFAULT_VALUE) / 2 and change it in the code
 *
 * (c) 2013 B&D Cuartielles
 *
 * This code is Free Software, licensed under GPLv3
 * Based on code by Paul Badger 2008
 *
 */

//Define the capacitive sensor
CapacitiveSwitch sensor=CapacitiveSwitch(2,3);
//Use Tinkerkit LED(or Tinkerkit LED matrix) here.
//Deinfe the LED
LED led=LED(9);

void setup()
{
   // configure the serial port
   Serial.begin(9600);

   //initialize components
   sensor.config(200);
   led.begin();
}

void loop(){
  //When someone attempts to open the
  //cookie jar, the sensor is activated
  if(sensor.pressed()){
    //Turn on the LED
    led.on();
    //Sends signal to processing, so a picture
    //will be captured
    Serial.print('p');
    //Wait 2 seconds before turning the LED off
    delay(2000);
    led.off();
  }
}
```

Ahí también podrás encontrar el código que necesitas ejecutar en Processing para que la cámara web capture las imágenes del ladrón de galletas.

```
/**
 * Hunting Cookie Monster
 *
 * Someone is eating our cookies, let's gather some proof
 * about that person and get him/her to confess.
 *
 * This program initializes a webcam connected to a PC
 * and listens to the serial port. If a "P" command arrives
 * it will then take a picture with the webcam and store it
 * locally in the sketch folder.
 *
 * (c) 2013 B. & D. Cuartielles in a quest for the truth
 * This is Free Software under GPLv3
 * Based on previous code by C. Reas and B. Fry
 */

import processing.video.*;
import processing.serial.*;

Capture cam;
Serial myPort;  // The serial port

void setup() {
  size(640, 480, P2D);

  initializeCamera(0);

  // List all the available serial ports
  println(Serial.list());
  // Open the port you are using at the rate you want:
  myPort = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
  if (cam.available() == true) {
    cam.read();
  }
  image(cam, 0, 0);

  String timeStamp=createTimeStamp();

  text(timeStamp, 10, height-10);



  // for the keyboard detection to work, you need to have
  // clicked on the application window first (aka focus)
  if(keyPressed) {
    if (key == 'p' || key == 'P') {
      captureImage(timeStamp);
```

```
    }
  }

  if (myPort.available() > 0) {
    int inByte = myPort.read();
    if(inByte == 'p') {
      captureImage(timeStamp);
    }
  }
}




void initializeCamera(int camNum){
  String[] cameras = Capture.list();

  if (cameras.length == 0) {
    println("There are no cameras available for capture.");
    exit();
  } else {
    println("Available cameras:");
    for (int i = 0; i < cameras.length; i++) {
      println("["+i+"] "+cameras[i]);
    }

    // The camera can be initialized directly using an element
    // from the array returned by list():
    cam = new Capture(this, cameras[camNum]);
    cam.start();
  }

  // we don't need the camera at full blast
  frameRate(1);

}

String createTimeStamp(){
  String timeStamp = String.format("%02d", hour());
  timeStamp += ":" + String.format("%02d", minute());
  timeStamp += ":" + String.format("%02d", second());
  timeStamp += " " + year();
  timeStamp += "/" + String.format("%02d", month());
  timeStamp += "/" + String.format("%02d", day());

  return timeStamp;
}

void captureImage(String timeStamp){
      saveFrame("pic-######.png");
      println("capturing Frame at: " + timeStamp);
}
```

¿Cómo Funciona?

Cuando alguien toca el bote de las galletas, el sensor capacitivo se activa y Arduino envía una "p" a Processing a través del puerto serial. El programa de Processing hace una foto a través de la webcam.

¿No funciona?

1. ¿Processing te está informando sobre un error y no aparece ninguna imagen? Asegúrate de que tengas la última versión de Processing y de que:

    - Arduino también esté conectado al ordenador y dentro de tu programa en Processing `myPort = new Serial(this, Serial.list()[0], 9600);` está usando el número del puerto serial de tu placa Arduino, lo puedes encontrar en el menú Herramientas -> Puerto serial de tu IDE de Arduino.
    - `initializeCamera()` está utilizando la cámara correcta de la lista de Processing.

2. ¿No captura ninguna imagen tocando el bote de galletas? Prueba a tocar el sensor directamente. Si no funciona, necesitas un bote de galletas que sea más conductor. Si no, mira la referencia del sensor capacitivo para corregir errores.

3. ¿Dónde está la foto del ladrón de galletas? Búscala en el folder sketch de Processing dentro de la carpeta de tu programa CookieMonster.

¡Sigue experimentando!

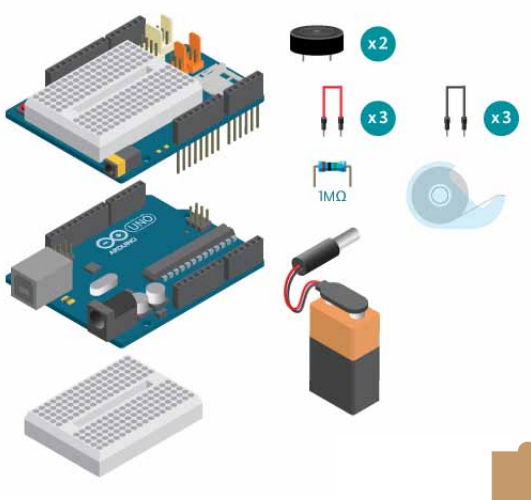
- Deja que el monstruo espere unos segundos antes de hacer la foto, ¡así pillarás al ladrón in fraganti!

Caja knock knock

Llama a éste ataúd para despertar al muerto. No te preocupes, el esqueleto no saldrá y te perseguirá, pero te contestará desde dentro.

Vale, sabemos que no hay ningún esqueleto real ahí. En realidad es un piezoeléctrico utilizado como sensor de toques. Después de que hayas montado este proyecto quizás se te ocurran otras formas más útiles de utilizar este sensor. Mira este cerrojo secreto detector de toques para inspirarte.

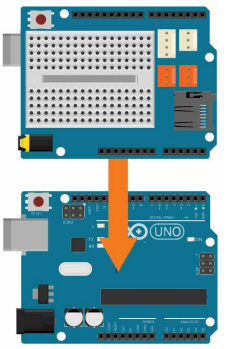
Materiales

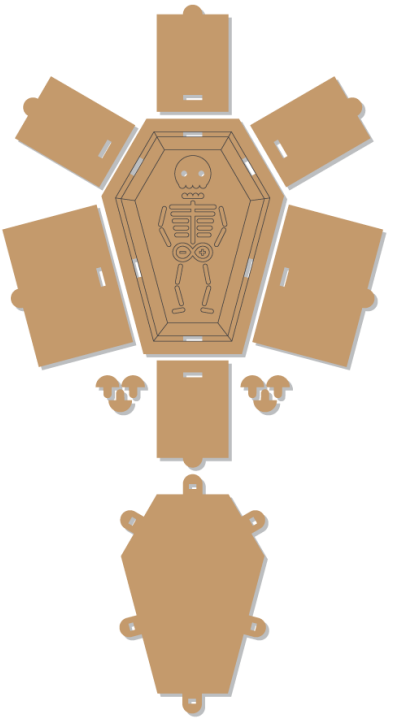


- 1 placa Arduino Uno
- 1 Shield Básica Educativa
- 2 piezos
- 1 resistencia de 1MOhm
- 3 cables negros
- 3 cables de colores
- 1 pila 9V
- 1 portapilas 9V
- Kit Knock-knock box
- cinta adhesiva
- 1 breadboard

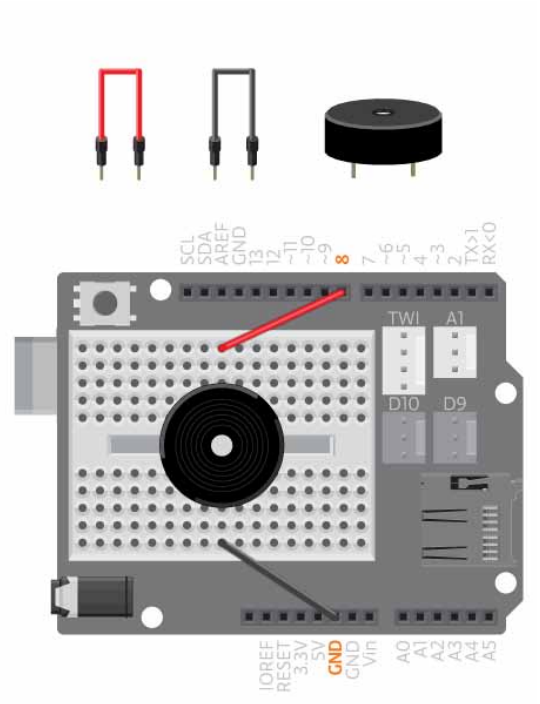
Instrucciones

1. Coloca la shield en la parte superior de tu Arduino.

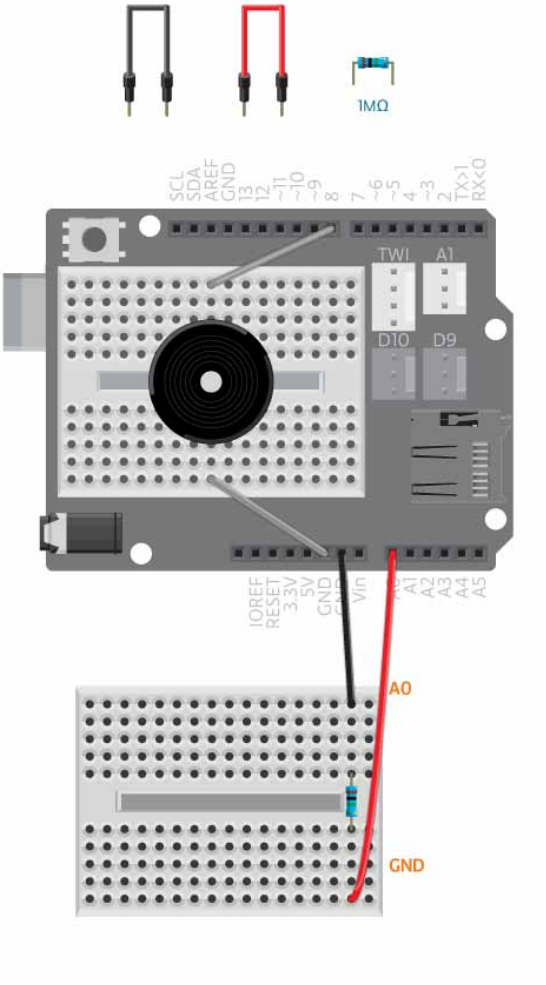




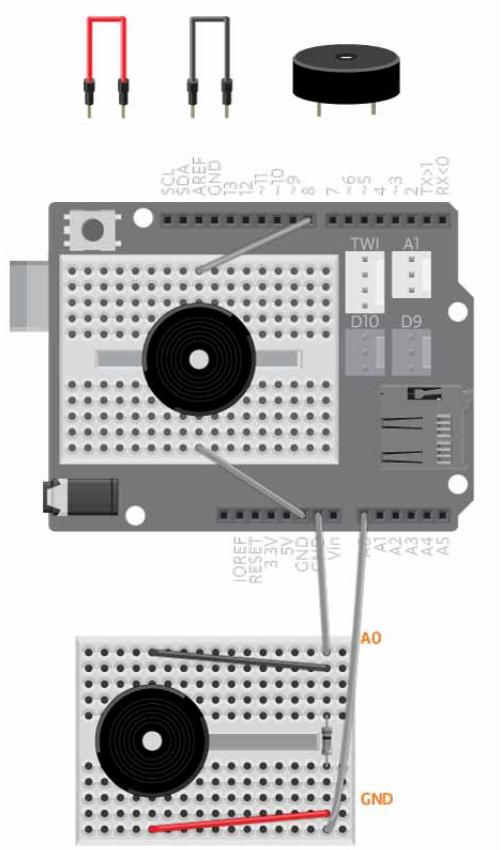
2. Conecta un piezo sobre la breadboard del shield entre el Pin digital 8 y GND.



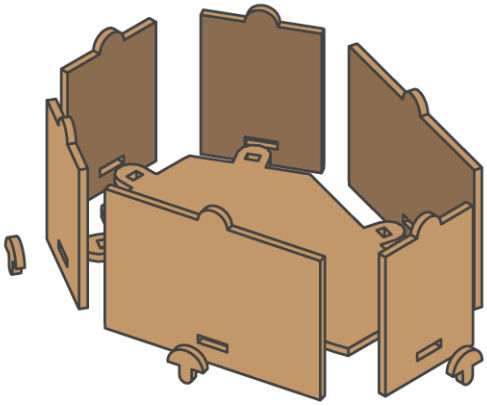
3. En otra breadboard distinta coloca la resistencia de 1MOhm, usa cables para conectarla entre el Pin analógico A0 y GND.



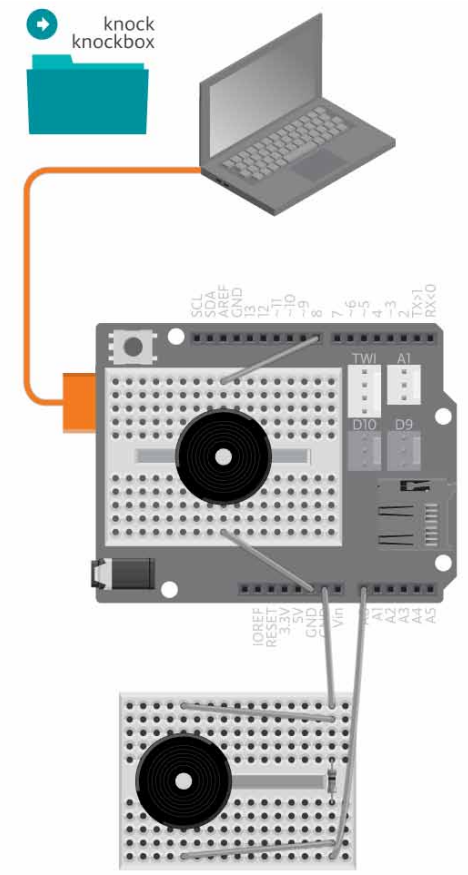
4. Conecta el otro piezo a los mismos Pins que la resistencia, entre A0 y GND. Esto actúa como un sensor de vibración. Asegúrate de que ambos piezos están conectados firmemente, usa cinta adhesiva si es necesario.



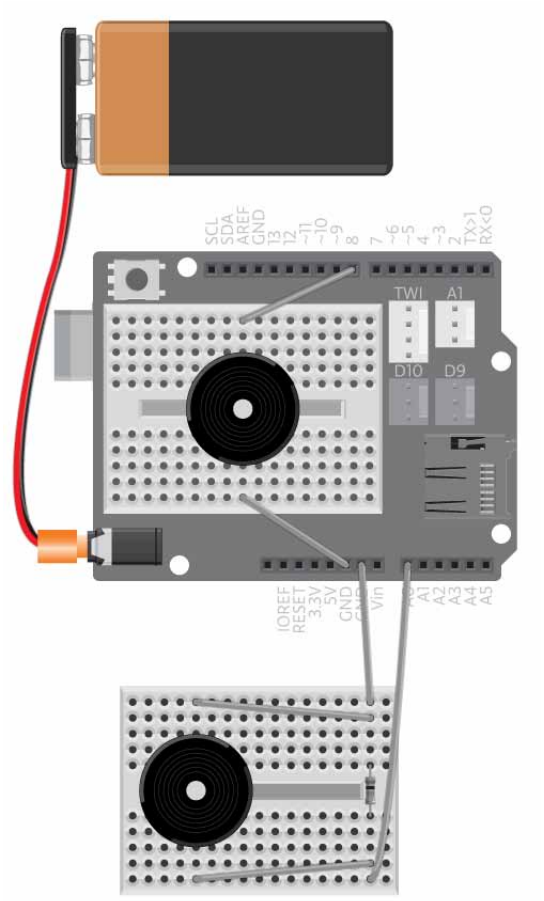
5. Construye la caja knock knock a partir del kit de madera.



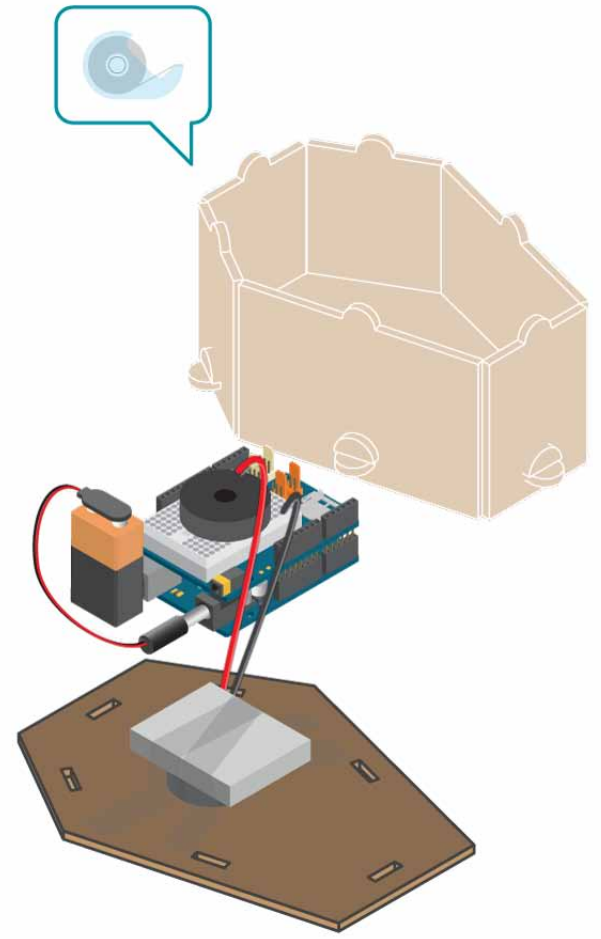
6. Conecta tu Arduino al ordenador y carga el ejemplo KnockKnockBox.



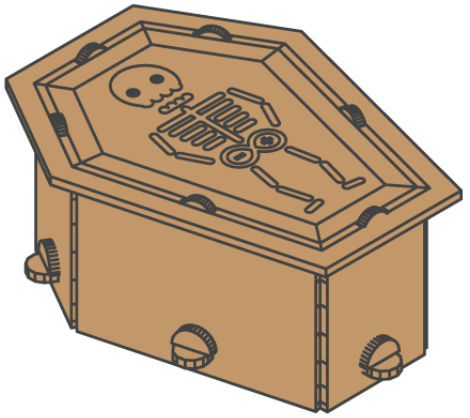
7. Coloca la pila de 9V en el portapilas. Desconecta Arduino del ordenador y conecta la batería a Arduino usando el conector.



8. Cierra la tapa de la caja. Asegúrate de que el piezo que actúa como sensor esté en contacto con la pared de la caja.



9.



Código

Puedes encontrar el código en Archivo -> Ejemplos -> BasicEducationShield-> Magic -> KnockKnockBox

```
/*
  Knock Knock Box

  Knock on this coffin to awake the dead. Don't worry,
  the skeleton won't get out and come after you but it
  will reply from inside.

  Ok, we know that you know that there isn't a real
  skeleton there. It's really a piezo used as a knock
  sensor. After you've made this project you might
  think of a handy way to use this sensor in other ways.
  Check out this secret knock detecting lock for some
  inspiration: http://www.youtube.com/watch?v=zE5PGeh2K9k

  (c) 2013 Arduino Verkstad
*/

#include <BasicEducationShield.h>

//The number of knocks that can be recorded
#define MAX_KNOCKS 30

PiezoKnockSensor sensor=PiezoKnockSensor(A0);
Melody speaker=Melody(8);

//An array for remembering the knock pattern
long timer[MAX_KNOCKS];

//If it has started recording
boolean started;

//Used for calculating if you have finished the pattern
long timeoutBase;

//If you stop knocking for the period of timeout, it'll
//stop recording
long timeout=2000;

//Keep track of the number of knocks you've knocked
int currentKnock;

void setup(){
  //define the threshold and debounce time of the knock
  //sensor. Threshold defines how hard you need to knock,
  //debounce time prevents the sensor from detecting
  //false knocks, but also limits how rapid you can knock.
  sensor.config(40,80);
```

```
48
49   //initializing the values
50   started=false;
51   timeoutBase=0;
52   currentKnock=0;
53   clearArray();
54 }
55
56 void loop(){
57   //Knock sensor waits for a short time if a knock is detected
58   //and then move on.
59   if(sensor.knocked(10)){
60     //If it's the first knock in the round, start recording
61     if(!started){
62       started=true;
63     }
64
65     long currentTime=millis();
66     //Reset timeout
67     timeoutBase=currentTime;
68     //Save the amount of milliseconds that have
69     //passed since the last knock
70     timer[currentKnock]=currentTime;
71     currentKnock++;
72   }
73   if(started){
74     //If recording has started and you stop
75     //knocking for the time of "timeout", it'll
76     //stop recording and play it back to you.
77     if(millis()-timeoutBase>timeout){
78       playback();
79       //reset the parameters, so a new round begins
80       started=false;
81       clearArray();
82       currentKnock=0;
83     }
84   }
85 }
86 void clearArray(){
87   //clean up values in the timer array
88   for(int i=0;i<MAX_KNOCKS;i++){
89     timer[i]=0;
90   }
91 }
92 void playback(){
93   //Play the knock pattern back to you through the
94   //speaker piezo
95   for(int i=0;timer[i]!=0;i++){
96     //Make a beep sound with tone 200 for 30 milliseconds
97     speaker.playTone(200, 30);
98     if(timer[i+1]){
99       //Wait the same amount of milliseconds that was detected
```

```
100       //between the knocks
101       delay(timer[i+1]-timer[i]);
102     }
103   }
104 }
```

¿Cómo Funciona?

La primera vez que golpeas la caja, comienza a grabarse el patrón de sonido. El sonido es detectado por el primero de los piezoeléctricos (el que está conectado al pin A0). Cada vez que el sensor es golpeado, Arduino registra el tiempo hasta el siguiente golpe en un Array. Si no se produce un nuevo golpe en más de 2 segundos, será considerado como el final de la grabación. Al entrar en este estado, el patrón de golpes será reproducido a través del otro altavoz piezoeléctrico.

¿No funciona?

1. Asegúrate de que la pila esté conectada correctamente y que por tanto el LED de alimentación esté encendido. Pruébalo con Arduino conectado al ordenador y la pila de 9V desconectada. Si funciona con el cable USB pero no con la pila, puede ser este el problema; prueba a cambiarla.
2. Recuerda que debes golpear sobre la pared derecha de la caja (la que tiene sensor piezo).
3. Mira la referencia del sensor de Knock para corregir errores.

¡Sigue experimentando!

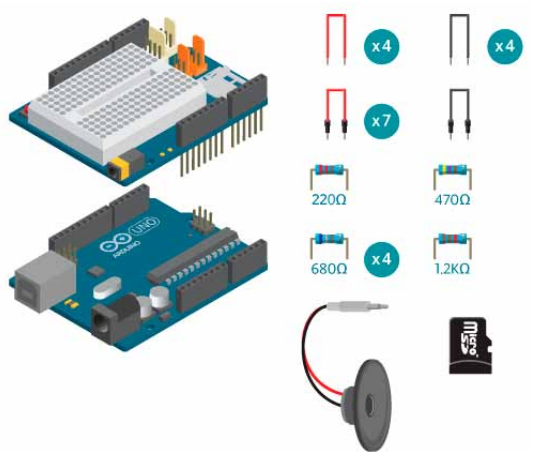
- Cambia el sonido con que la caja devuelve el golpe.

Secuenciador

Reproduce ritmos y cambia la secuencia en tiempo real. Es lo básico para convertirte en un artista del hiphop. Eso, y habilidades de rapero, pero eso es otro tipo de curso.

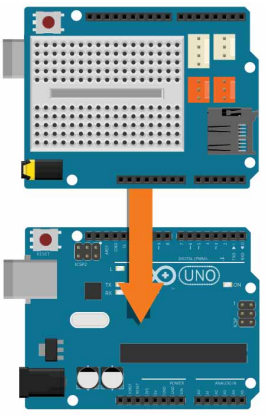
Materiales

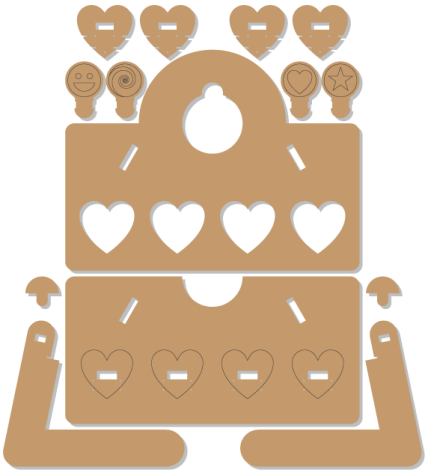
- 1 placa Arduino
- 1 shield Básica Educativa
- 4 resistencias de 680 ohm
- 1 resistencia de 220 ohm
- 1 resistencia de 470 ohm
- 1 resistencia de 1.2K ohm
- 1 altavoz de 8 ohm
- 1 tarjeta micro SD
- 4 cables sueltos negros
- 4 cables sueltos de colores
- 1 cable conector negro
- 7 cables de colores
- 1 kit Secuenciador



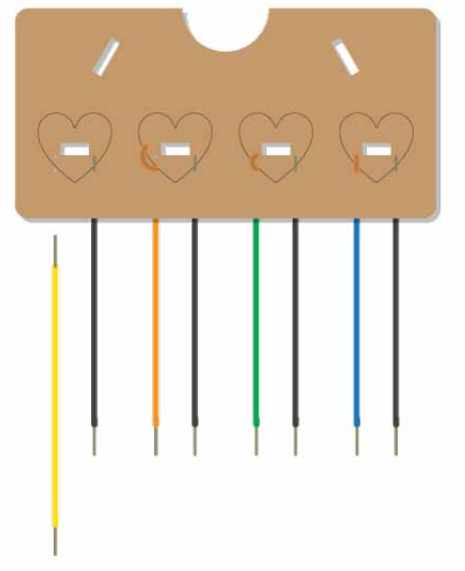
Instrucciones

1. Conecta la shield por la parte superior de Arduino.

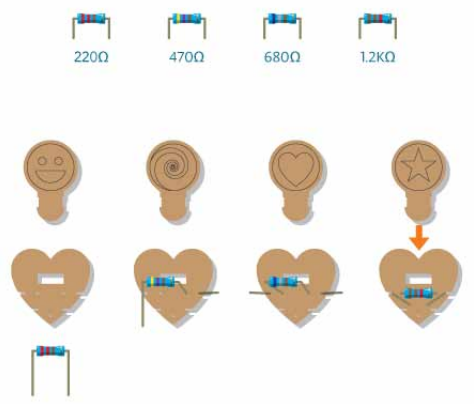




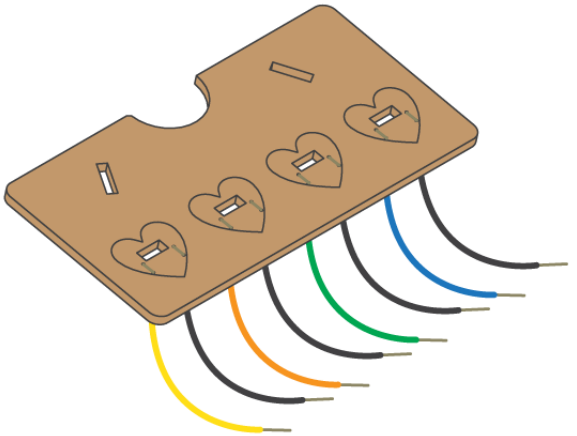
2. Pon el extremo de un cable negro pelado a través de los agujeros de uno de los lados de cada corazón. Utilizando un cable de color previamente pelado, haz lo mismo por el otro agujero de cada uno de los corazones.



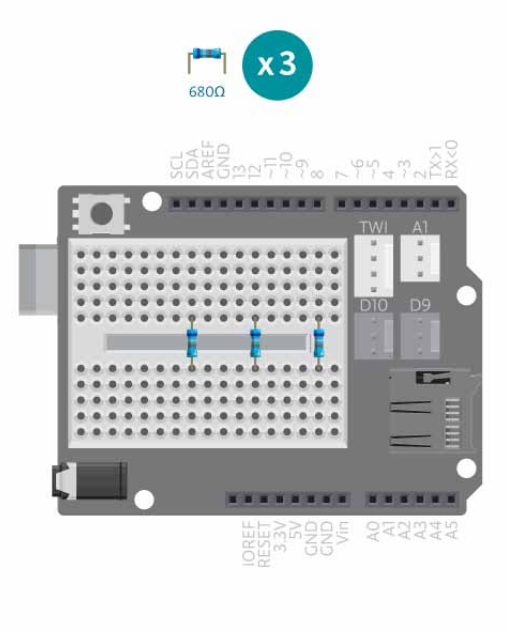
4. Monta la mesa del Secuenciador.
Coloca una resistencia en cada pieza con forma de corazón (220Ω, 470Ω, 680Ω, 1.2KΩ) y usa la pieza superior para sujetarla en su sitio.



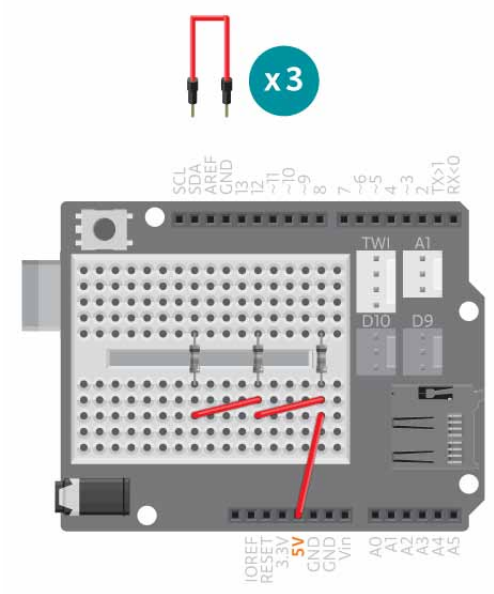
3. Monta la mesa del Secuenciador.



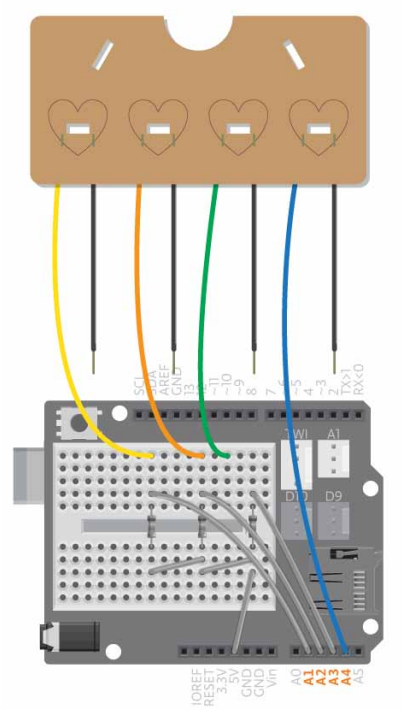
5. Conecta tres resistencias de 680 ohm a través del puente de la breadboard.



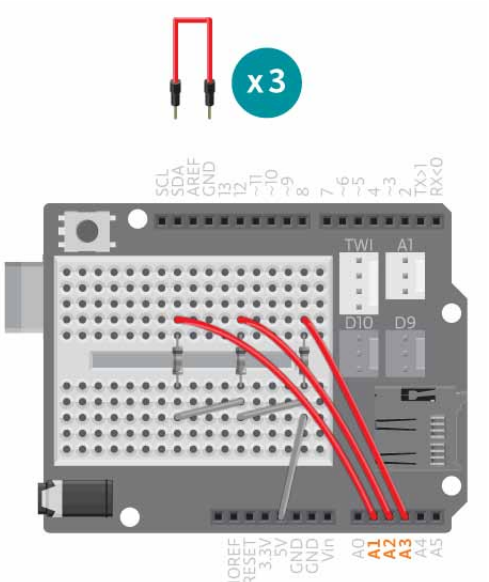
6. Conecta uno de los extremos de las resistencias a 5V.



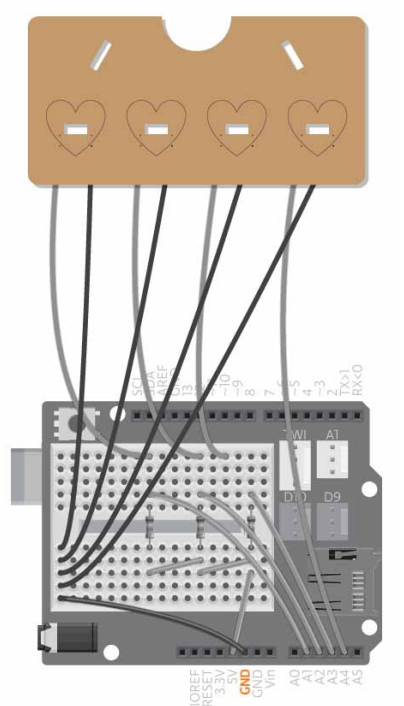
8. Conecta tres cables de colores de la mesa del secuenciador a la breadboard en A1, A2 y A3 y el cuarto directamente a A4.



7. Conecta el otro extremo de la resistencia a los pines analógicos A1 y A3. (El pin analógico A4 tiene una resistencia integrada en la shield, esto es el motivo por el que no necesitamos conectar una resistencia aquí).

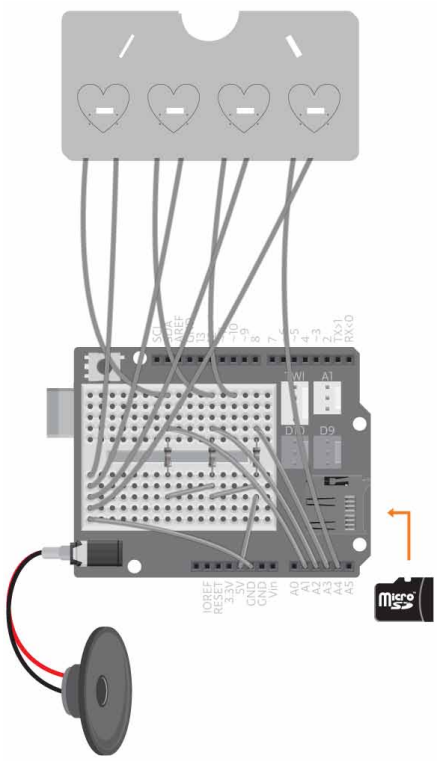


9. Conecta los cables negros de la mesa del secuenciador a GND.

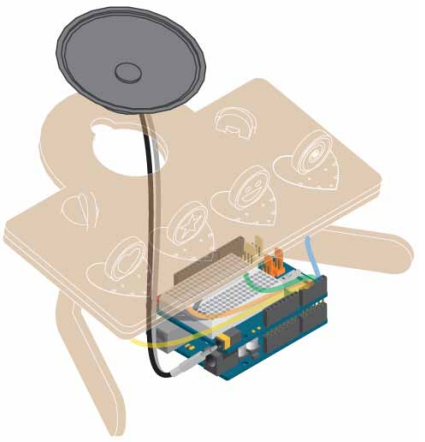


10. Mueve los archivos de sonido nombrados seq0.wav, seq1.wav, seq2.wav, seq3.wav, seq4.wav al directorio raíz de la tarjeta SD.

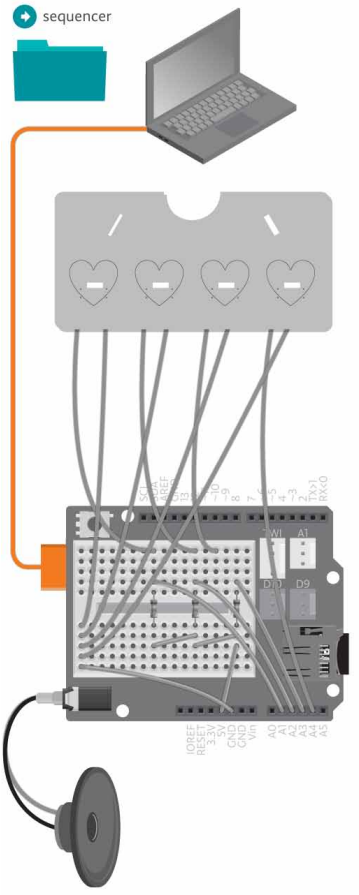
12. Coloca la tarjeta SD en el lector SD del shield.



11. Conecta el altavoz al conector de audio.



13. Conecta la placa Arduino al ordenador y carga el ejemplo Sequencer.



Código

Puedes encontrar el código en Archivo -> Ejemplos -> Educación Básica -> Magia -> Sequencer

```
/*
  Sequencer
  Play beats and change the sequence in real time. Basically
  all you need to become a hiphop artist. That, and rapping
  skills. But that's another course.

  (c) 2014 Arduino Verkstad
*/


#include "BasicEducationShield.h"

//We need to include the SD library to be able to read from an SD card
#include <SD.h>

//Declare the sound player
Player player=Player();

//There're 4 slots for 4 sequences. We use 4 analog pins
//to read them.
int analog_pins[]={4,3,2,1};
int pin_count=4;

//Template for the music file names.
char filename[]="seq0.wav";

void setup(){
  // Open serial communications and wait for port to open:
  Serial.begin(9600);

  // Initialize the sound player. Open the serial monitor to see
  //the sound files found on your micro SD card
  player.begin();

}
void loop(){
  //Loop through 4 pins and play the right sequence accordingly
  for(int i=0;i<pin_count;i++){
    int slot_value=analogRead(analog_pins[i]);
    int sequence=getSeq(slot_value);

    //Get the right file name by sequence
    filename[3]='0'+sequence;
    Serial.println(filename);
    //Play the file
    player.play(filename);
  }
```

```
48
49    //End of one loop
50    Serial.println("=======");
51  }
52  int getSeq(int analogVal){
53    if(analogVal>200 && analogVal<300){  //220 Ohm
54     return 1;
55    }else if(analogVal>360 && analogVal<460){  //470 Ohm
56     return 2;
57    }else if(analogVal>480 && analogVal<580){  //680 Ohm
58     return 3;
59    }else if(analogVal>600 && analogVal<700){  //1k2 Ohm
60     return 4;
61    }else{  //No resistor
62     return 0;
63    }
64  }
```

Cómo funciona

Cada hueco con forma de corazón está conectado a un pin analógico y a tierra (GND). Cada corazón tiene una resistencia de diferente valor que, al ser situada en los huecos, conecta los cables en los huecos. Puesto que cada corazón posee una resistencia diferente, tiene también una única lectura. Esto significa que podemos distinguir qué corazón está conectado a qué hueco.

En el programa usamos `analogRead()` para leer el valor analógico de un pin. Comprobamos qué corazón está conectado en `getSeq()`, esto es, qué resistencia está utilizada. Si es la resistencia de 220 ohm, la función devolverá un '1' de forma que `sequence` será igual a '1'. Utilizamos este valor para decidir que archivo reproducir. En este caso es "seq1.wav". Si ningún corazón está conectao al hueco, el archivo "seq0.wav" se reproduce, este es un archivo silencioso.

El proceso se repite 4 veces, una por cada pin analógico que estamos usando. Una vez finalizado, se repite de nuevo.

¿No funciona?

1. Revisa la ilustraciones y comprueba todas tus conexiones. Asegúrate de que los cables están firmemente conectados.
2. ¿Tienes problemas reproduciendo los archivos de sonido? Mira la referencia para corregir el reproductor de sonidos.
3. Asegúrate de que estás utilizando las resistencias correctas. Mira la referencia de resistencias para saber como hacerlo.

¡Sigue experimentando!

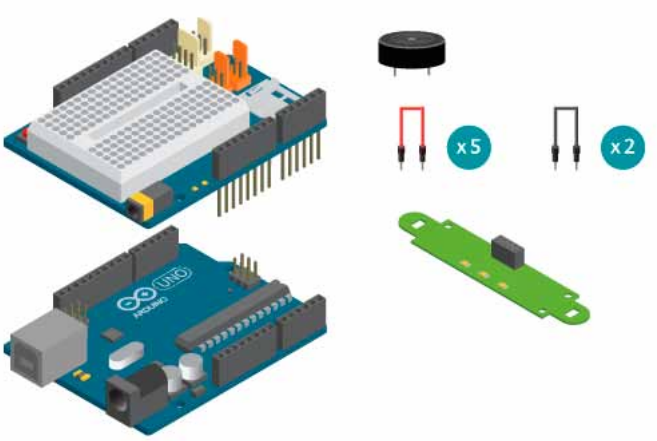
- Graba tus propios 5 sonidos wav y llamalos"seq0.wav", "seq1.wav", "seq2.wav", "seq3.wav" y "seq4.wav". Todos deben de tener la misma duración para un mejor resultado y "seq0.wav" debe ser silencioso. Puedes utilizar Audacity (http://audacity.sourceforge.net/) para grabar sonidos compatibles con Arduino. Mira la z para preparar sonidos Wav para más detalles.

Tocadiscos binario

Esto funciona parecido a un tocadiscos. La diferencia está en que, en lugar de utilizar una aguja en un disco de vinilo, utilizamos tres sensores IR en línea para leer el patrón de un disco de papel. Si eres de los que te gusta tanto lo musical como lo digital, te lo pasarás genial creando diferentes melodías para este proyecto.

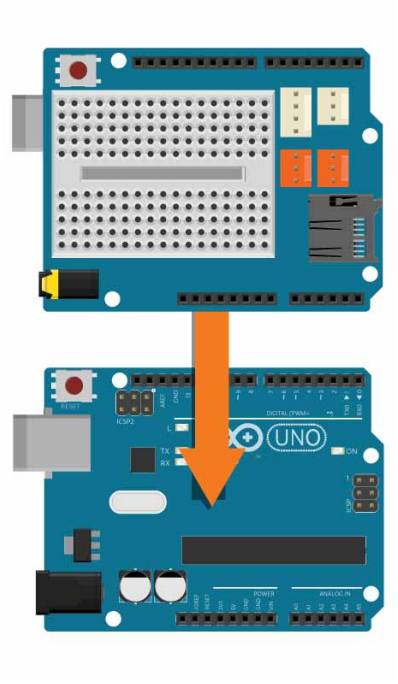
Materiales

- 1 placa Arduino UNO
- 1 shield Básica Educativa
- 1 Piezo
- 1 IRArray
- 2 cables negros
- 5 cables de colores
- Kit Binary LP
- disco de papel Binary LP

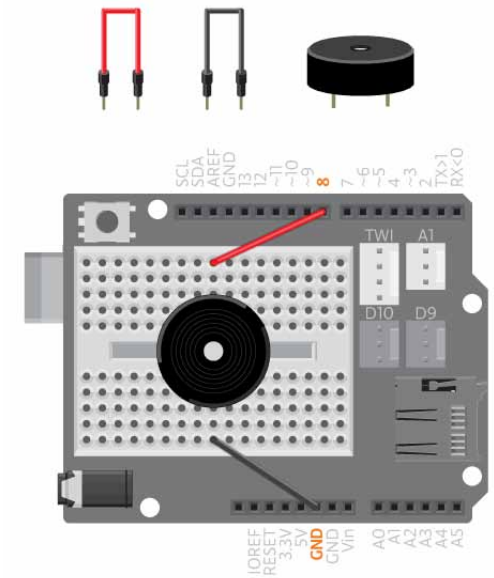


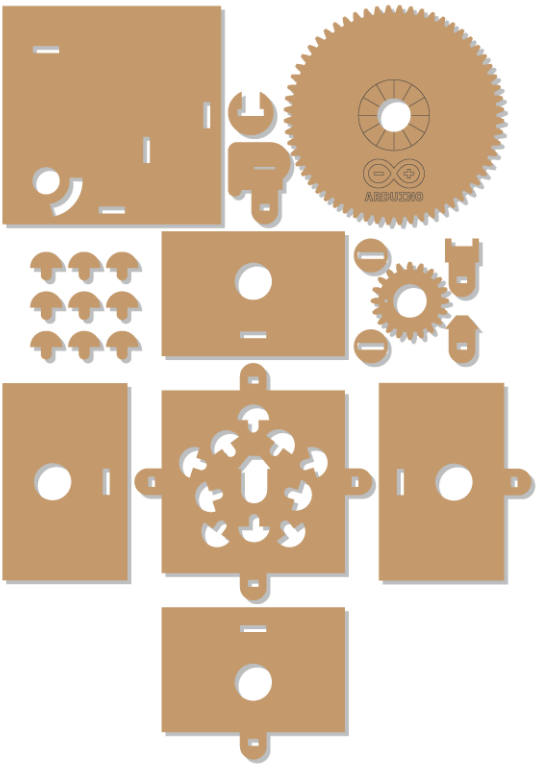
Instrucciones

1. Conecta la shield a la placa Arduino.

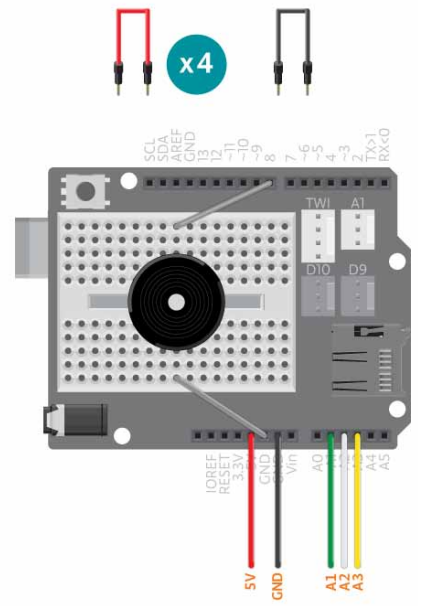


2. Conecta el altavoz piezoeléctrico a la breadboard y conecta una pata al pin digital 8 y el otro a GND.

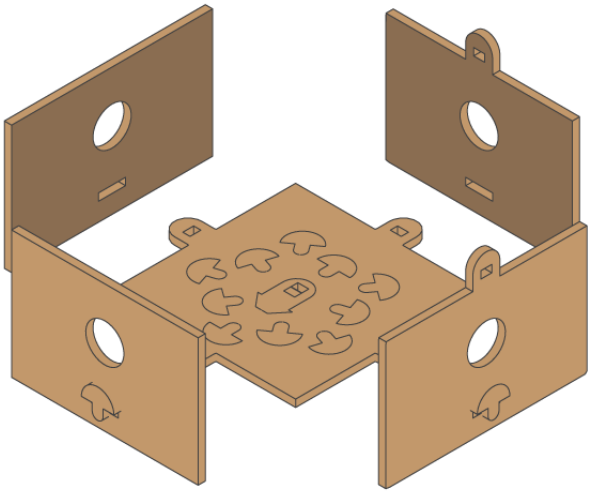




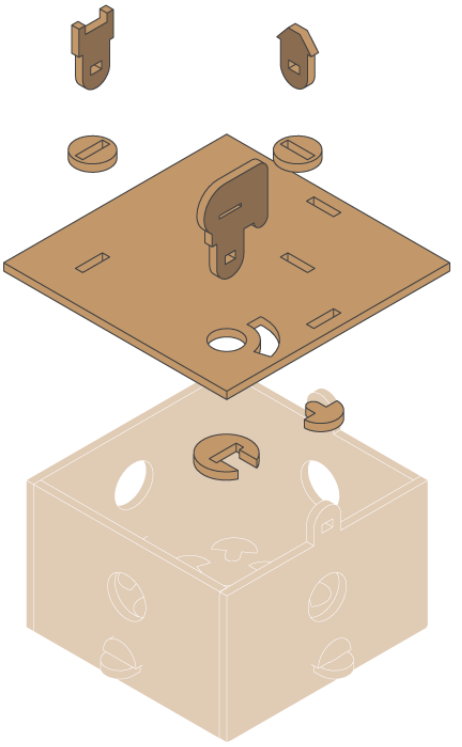
3. Conecta 5 cables a la placa, dejando suelto su otro extremo, uno negro a GND, uno rojo a 5V y 3 de diferentes colores a A1, A2 y A3. Después serán conectados al IRArray, así que recuerda qué color está conectado a qué pin.



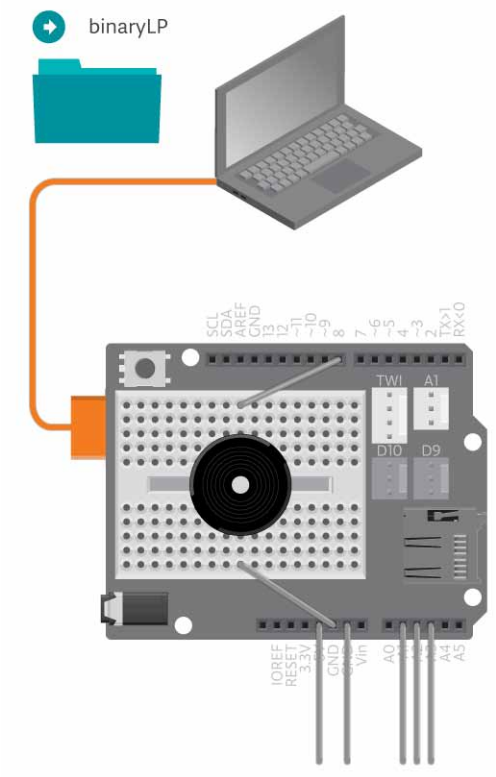
4. Monta la caja y la tapa. (Pero no pongas la tapa todavía).



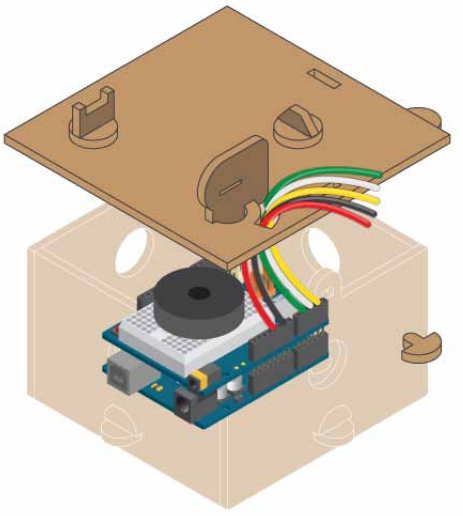
5.



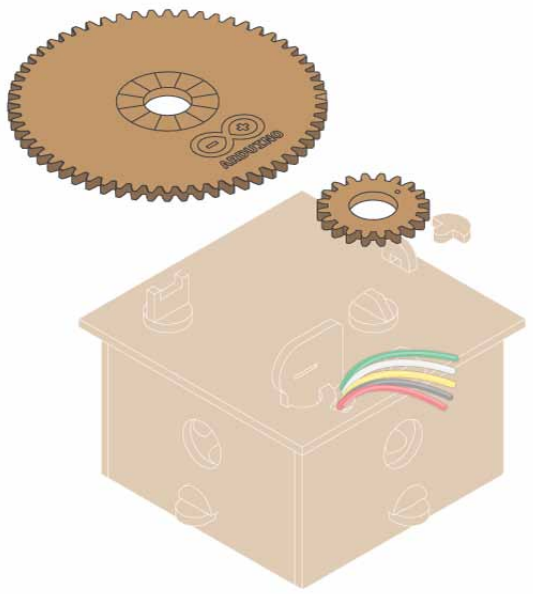
6. Introduce la placa Arduino en la caja, conéctala al ordenador, y carga el ejemplo BinaryLP.



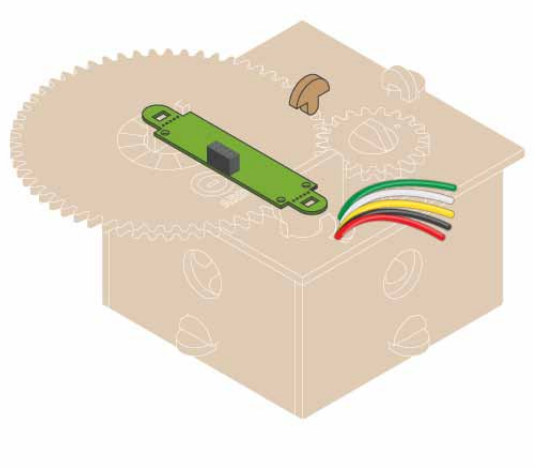
7. Pasa el extremo suelto de los cables a través del agujero de la tapa del tocadiscos.



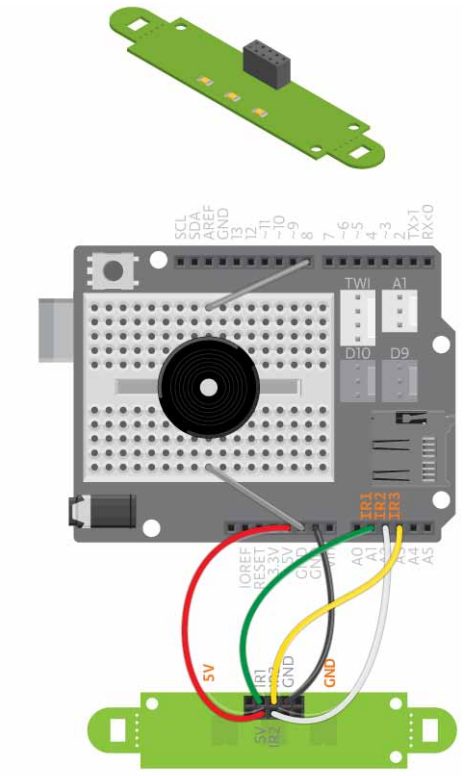
8. Pon la tapa en la caja, y los engranajes en la tapa.



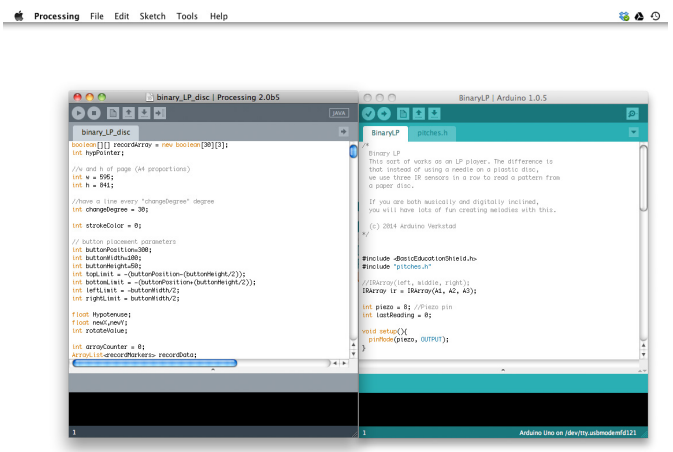
9. Coloca el IRArray en el soporte.



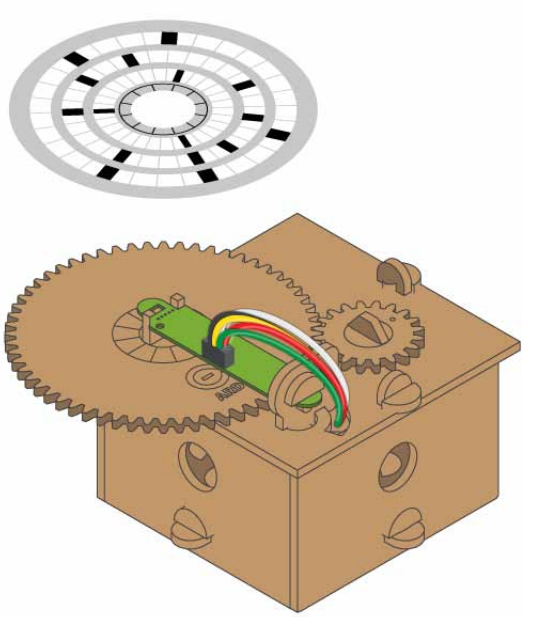
10. Conecta los cables sueltos al IRArray. 5V a 5V, GND a GND, A1 a IR1, A2 a IR2 y A3 a IR3.



11. Utiliza el sketch de processing en la carpeta de ejemplos para crear tu propio disco. Imprime el PDF creado y asegúrate de imprimir a tamaño real (no 'ajustar a página').



12. Pega el disco con celo a la plataforma giratoria, asegurándote de que las marcas están alineadas correctamente de forma que queden centradas.



Código

Puedes encontrar el código en Archivo -> Ejemplos -> ShieldBasicaEducativa-> Magia -> BinaryLP

```
/*
  Binary LP
  This sort of works as an LP player. The difference is
  that instead of using a needle on a plastic disc,
  we use three IR sensors in a row to read a pattern from
  a paper disc.

  If you are both musically and digitally inclined,
  you will have lots of fun creating melodies with this.

  (c) 2014 Arduino Verkstad
*/


#include <BasicEducationShield.h>
#include “pitches.h”

//IRArray(left, middle, right);
IRArray ir = IRArray(A1, A2, A3);

int piezo = 8; //Piezo pin
int lastReading = 0;

void setup(){
  pinMode(piezo, OUTPUT);
```

```
}

void loop(){
  //Read the binary pattern and get a number from 0 to 7
  int reading = ir.readBinary();

  //Play a note depending on the read value
  playNote(reading);

}

void playNote(int r){
  //If the last note we played is the same as the new note
  //we make a short break so that two notes are distinguished
  if(lastReading==r){
    noTone(piezo);
    delay(20);
  }

  //Play a different note depending on the value of b.
  //Check pitches.h to see which notes you can use.
  //In this case the scale is C Major.
  switch (r){
    case 0:
      break;
    case 1:
      tone(piezo, NOTE_C4);
      break;
    case 2:
      tone(piezo, NOTE_D4);
      break;
    case 3:
      tone(piezo, NOTE_E4);
      break;
    case 4:
      tone(piezo, NOTE_F4);
      break;
    case 5:
      tone(piezo, NOTE_G4);
      break;
    case 6:
      tone(piezo, NOTE_A4);
      break;
    case 7:
      tone(piezo, NOTE_B4);
      break;
  }

  //If r is more than 0 we save that value to lastByte
  if(r>0)lastReading = r;
}
```

Cómo funciona

Los tres sensores IR en el IRArray pueden detectar el blanco y negro del disco de papel. Esto significa que puede detectar 8 combinaciones diferentes de blanco y negro. ¿Recuerdas cómo contar en binario? Ahora te será útil. En el programa utilizamos el comando `readBinary()` para leer del disco y recibir un valor de 1 a 7. Si el patrón es blanco, blanco, negro, recibimos un '1'. Si el patrón es blanco, negro, blanco, recibiríamos un '2', etc. Entonces utilizamos ese valor leído para tocar una nota. Tú decides que nota reproducir para cada número en la función `playNote()`. Por ejemplo, si recibimos un '1' y tocamos `NOTE_C4` y si recibimos un '2' reproducimos `NOTE_D4`. En este ejemplo estamos usando los tonos de la escala Do mayor. Mira la tabla "pitches.h" para ver que otros tonos puedes utilizar. Si leemos un '0' no tocamos ninguna nota específica, en lugar de ello dejamos el espacio en blanco decidir cuánto tiempo debe ser reproducida la nota anterior. Cuanto más largo sea el espacio, más durará la nota. Si la última nota coincide con la actual, apagamos el altavoz por 20 milisegundos, de forma que se puedan distinguir las dos notas diferentes.

¿No funciona?

1. Revisa las ilustraciones y comprueba tus conexiones. Asegúrate de que los cables están firmemente conectados.
2. Corrige el IRArray, mira la referencia del IRArray.

¡Sigue experimentando!

- Compón dos melodías diferentes.
- Piensa una nueva forma de utilizar las lecturas binarias.

SEMANA **4**

Robots

CONCEPTOS

PROYECTOS

Esta semana vamos a construir robots. Os juntaréis en grupos y construiréis diferentes máquinas a partir de motores. La función básica de estos motores es muy sencilla pero verás que, dependiendo de cómo los utilices, podrás animar los elementos de maneras muy diferentes. Cada grupo hará su propio robot y al final de la semana, lo enseñaréis al resto de la clase.

Antes de nada, comenzaremos con una introducción a los distintos tipos de motores que existen y veremos unos ejemplos sobre cómo utilizarlos.

Tipos de motores

Para controlar los motores, utilizarás la placa Arduino. Básicamente, cualquier objeto electrónico que hayas visto o conoces, tiene alguna pieza en movimiento. Por ejemplo: impresoras, coches de juguete, cepillos de dientes eléctricos, etc.; todos, contienen motores. Los hay de muchos tipos, pero principalmente encontrarás tres tipos de motores:

Motores DC (corriente continua):

Si necesitas que algo gire pero sin precisión, este es tu motor. Para encontrar un motor DC en la vida real, busca el ventilador que se encuentra dentro de tu ordenador. También puedes encontrar uno muy pequeño en tu teléfono móvil. El motor DC es el que hace que tu teléfono vibre, haciendo girar un bloque de metal cuyo peso está distribuido de manera no uniforme.

Un motor DC puede funcionar libremente en ambas direcciones, es muy fácil controlar su velocidad pero no su posición. Tampoco es sencillo hacerlo parar de forma precisa. Viene con dos cables: alimentación y tierra. Por regla general un motor DC no puede ser alimentado directamente de la corriente proporcionada por un Pin digital de Arduino.

Los cables se pueden conectar a tierra o a un Pin digital. Para hacerlo girar, establece el Pin digital en HIGH y para que se detenga, pon el Pin digital en LOW. Para hacerlo girar en sentido contrario, cambia el orden de los cables.

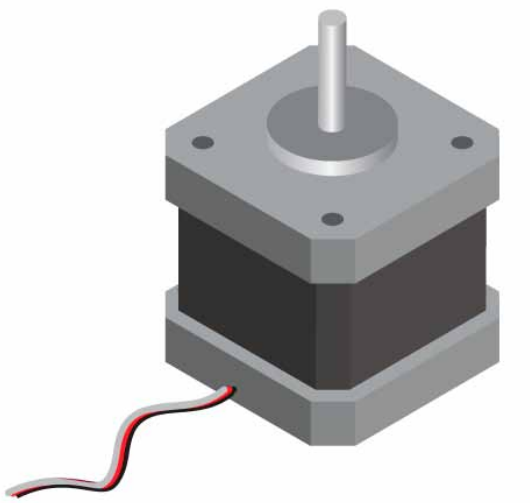
Es posible controlar la velocidad de un motor DC desde Arduino con una técnica llamada PWM usando un transistor. Con varios transistores dispuestos en un puente H, puedes incluso controlar la dirección sin tener que desconectar el motor.

Motores paso a paso:

Los motores paso a paso se pueden encontrar en cualquier objeto electrónico donde prima la precisión, como escáneres e impresoras. Un motor paso a paso puede, a diferencia del motor DC, ser muy preciso tanto en posición como en velocidad.

La rotación completa de los motores paso a paso se divide en pasos equitativos y puedes controlar la parada del motor en cada uno de estos pasos. Los pasos se miden en grados, normalmente 1.8, 3.6 o 7.2. Cuanto más pequeños sean los pasos, más preciso será. Esto hace que sea muy útil cuando se necesita un posicionamiento repetido.

Sin embargo, el motor paso a paso nunca será muy rápido. Un motor paso a paso tiene 4 o más cables. Por lo general, necesitas más de 5 voltios para alimentar un motor paso a paso, lo que significa que no se puede alimentar directamente desde Arduino. Sin embargo, podemos utilizar una fuente de alimentación externa para alimentar el motor y controlarlo desde Arduino a través de un transistor.



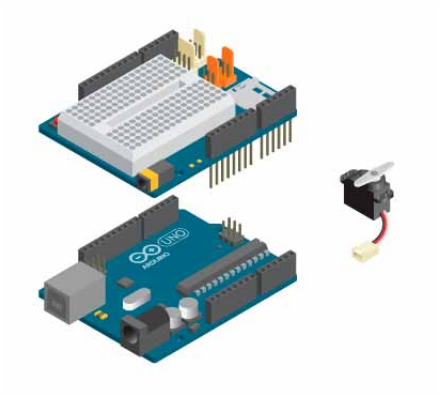
Servomotores

Los servomotores son ampliamente utilizados en robótica y en radio-control. Estos tipos de motores son los que vas a utilizar esta semana puesto que son muy sencillos de controlar y conectar desde Arduino.

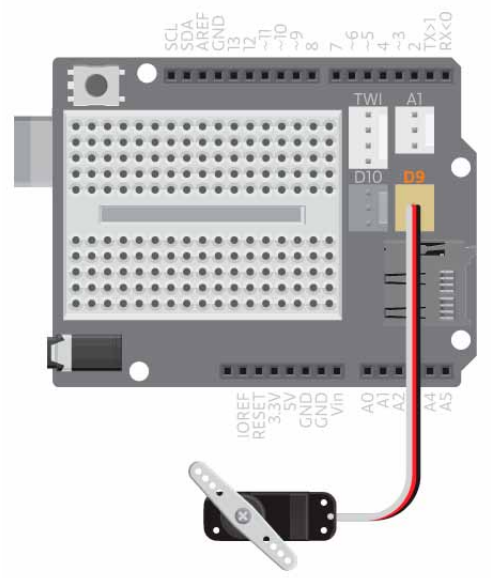
Tienen tres cables: uno para la energía, uno para tierra y otro para controlarlos. Hay dos tipos de servomotores: rotación estándar y rotación continua. El estándar puede girar 180 grados y puede ser controlado como el motor paso a paso a una posición precisa. La rotación continua puede, al igual que el motor DC, rotar en ambas direcciones, no tan rápido; pero puedes controlar tanto la velocidad como la dirección sin tener que utilizar transistores.

Servo estándar

Comencemos por comprobar cómo funciona el servomotor estándar.



Conéctalo al pin D9.



Para controlar los servos vamos a utilizar una librería llamada Servo. Una librería es un fragmento de código que ya ha sido escrito y que se puede instanciar (llamar) cada vez que necesites utilizarlo. En lugar de tener que escribir todo el código, solo tienes que añadir esta línea de código `#include <Servo.h>` a tu programa.
Escribe el siguiente en el Arduino IDE:

```
1   #include <Servo.h>
2
3   Servo myservo;
4
5   void setup() {
6   myservo.attach(9);
7   }
8
9   void loop() {
10  myservo.write(0);
11  delay(1000);
12  myservo.write(180);
13  delay(1000);
14  }
```

Estos son los diferentes comandos que utilizamos:

- `#include <Servo.h>`: Incluye el código que realmente utilizamos para controlar los motores.
- `Servo myservo`: Crea un objeto servo `myservo`. Este es el que utilizas para controlar el motor.
- `myservo.attach(pinNumber)`: Aquí es dónde indicas a qué Pin conectas el servo.
- `myservo.write(degrees)`: Ordena al servo a qué posición girar. Como solo puede girar 180 grados, este valor varía de 0 a 180.

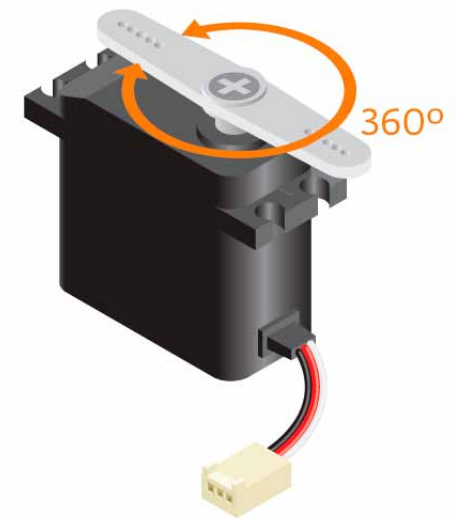
Lo que hace el programa es girar el servo a una posición de 0 grados, espera 1 segundo y luego lo gira a una posición de 180 grados.

¡Sigue experimentando!

- Cambia el ángulo del servo cambiando el valor dentro de la función write. Prueba añadir más posiciones al programa. Observa qué sucede si cambias el tiempo de retardo.
- Como puedes ver, los movimientos del servo son muy repentinos, especialmente si hay un gran salto entre dos posiciones. Esto se debe a que el servo cambia la posición prácticamente de forma instantánea. ¿Tienes alguna idea de cómo hacer los movimientos más suaves? En lugar de hacer que el servo salte directamente de una posición a otra, puedes aumentar o disminuir la posición poco a poco.
- Intenta escribir un programa que haga que el servo haga un barrido de un lado a otro, de una manera suave. (SUGERENCIA: mira el ejemplo Archivo -> Ejemplos -> Servo -> Sweep)

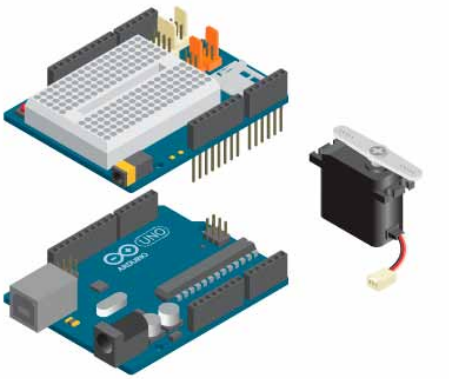
Servo de giro continuo

El servo continuo tiene el mismo aspecto que un servo estándar, pero es capaz de girar continuamente como un motor normal. No puedes controlar qué ángulo está señalando pero puedes especificar la rapidez con la que quieres que gire y en qué dirección debe girar.



Materiales

- 1 placa Arduino Uno
- 1 shield Básica Educativa
- 1 servo continuo

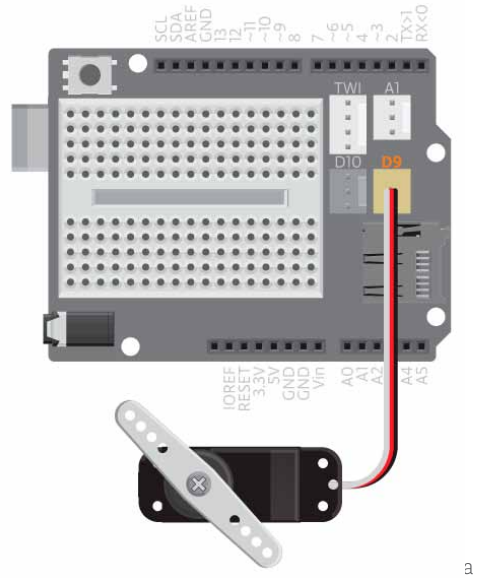


Instrucciones

1. Conecta el servo al puerto D9.

Abre Archivo -> Ejemplos -> BasicEducationShield -> Help -> ServoContinuo en el Arduino IDE.

```
1  /*  ContinuousServo
2  */
3
4  #include <BasicEducationShield.h>
5  #include <Servo.h>
6  Servo me;
7
8  void setup(){
9  me.attach(9);
10 }
11
12 void loop(){
13 //Make the servo rotate in speed 120.
14 me.write(120);
15
16 }
```



Este programa hará que el servo gire a una velocidad de 120. Pued a velocidad más rápida en una dirección y 0 la más rápida también pero en la dirección opuesta. 90 debería dejarlo parado. Este valor no es exacto y puede variar de un servo a otro, por lo que deberás calibrarlo.

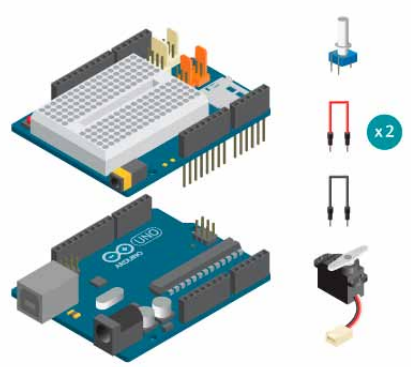
¿No funciona?

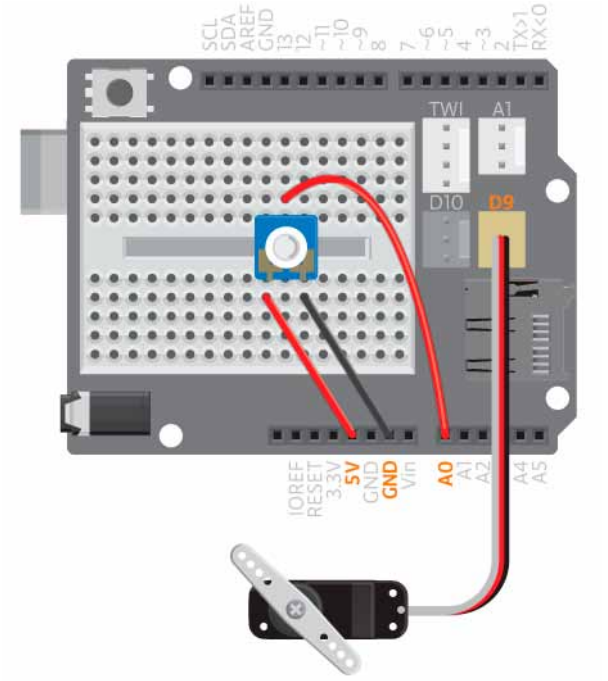
- Asegúrate que has conectado el servo al mismo pin que el utilizado en el código.

Servo controlado con entrada

Vamos a intentar utilizar una entrada para controlar el servo. En este ejemplo vamos a utilizar un potenciómetro.

Conecta el Pin central del potenciómetro a la entrada analógica A0. Conecta uno de los otros Pins a 5V y el último Pin a tierra. Conecta un servo estándar al pin D9.





Carga el siguiente código a tu placa Arduino:

```
#include <Servo.h>

Servo myservo;

int potPin = A0;
int val;

void setup() {
myservo.attach(9);
}

void loop() {
val = analogRead(potPin);
val = map(val, 0, 1023, 0, 179);
myservo.write(val);
delay(15);
}
```

A este ejemplo le hemos añadido:

- Dos variables: `potpin` es el número del pin utilizado para el potenciómetro y `val` para el valor de lectura desde el potenciómetro.
- En `loop()` comenzamos leyendo el valor del potenciómetro.
- Los valores que leemos del potenciómetro son 0 a 1023 pero necesitamos valores de 0 a 180 para controlar el servo. Para ello, utilizamos la siguiente función:
- `map(value, fromLow, fromHigh, toLow, toHigh)`

para ajustar los valores al rango que podemos utilizar con el servo. Esto significa que si leemos 1023 del potenciómetro obtendremos 180, si leemos 511 obtendremos 90, etc.

- Utilizamos una variable (`val`) para establecer la posición del servo.
- Utilizamos un pequeño `delay` para darle tiempo a girar al servo.

¡Sigue experimentando!

- Cambia el servo estándar con uno de giro continuo. Utiliza el mismo código pero suprime la función `delay`, cárgalo y vuelve a intentarlo. ¿Cómo actúan los diferentes servos y por qué? Piensa por qué la función de `delay` es necesaria cuando se controla el servo estándar pero no lo es con el continuo.
- ¿Qué más puedes utilizar para controlar los servos y qué puedes añadirles para que sean más útiles?

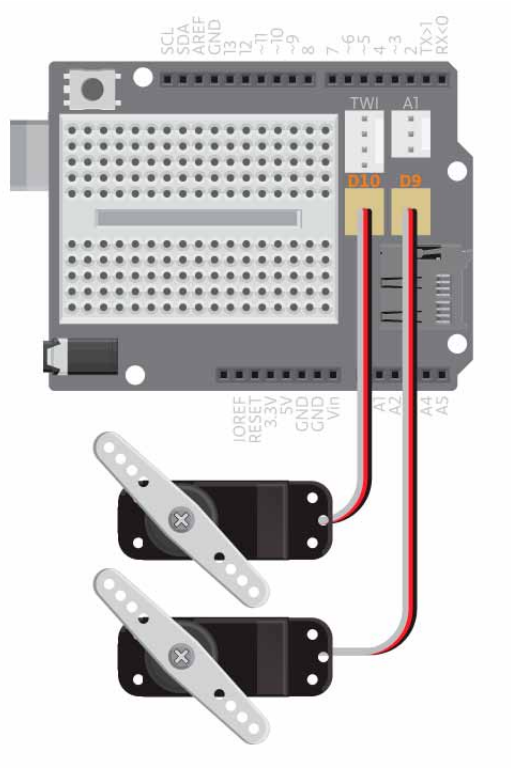
Utilizando dos servos

Cuando utilices dos servos a la vez, tu placa Arduino puede tener un problema con darles la suficiente corriente. Hay diferentes técnicas para controlar esto. Aquí vas a explorar la técnica de escribir señales a los servos por separado. Este control lo harás desde tu programa.

Conecta dos servos a la shield, uno a D9 y otro a D10.

Escribe el siguiente código en el IDE de Arduino:

```
#include <Servo.h>

Servo myservo1, myservo2;

void setup() {
myservo1.attach(9);
myservo2.attach(10);
}

void loop() {
myservo2.detach();
myservo1.attach(9);
myservo1.write(70);
delay(1000);
myservo1.write(120);
delay(1000);

myservo1.detach();
myservo2.attach(10);
myservo2.write(70);
delay(1000);
myservo2.write(120);
delay(1000);
}
```



El programa desconecta `myservo2` cuando va a utilizar `myservo1`. Hace girar `myservo1` un ángulo de 70 grados, espera 1 segundo y luego gira el mismo motor a 120 grados.

Del mismo modo, para hacer el mismo movimiento con `myservo2` necesitas desconectar mysevo1 y conectar `myservo2`.

¡Sigue experimentando!

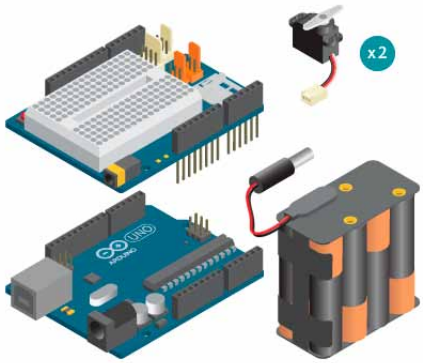
- Inventa un robot que use un motor de giro continuo y otro de giro estándar para que se mueva hacia adelante, ¿cómo lo harías?

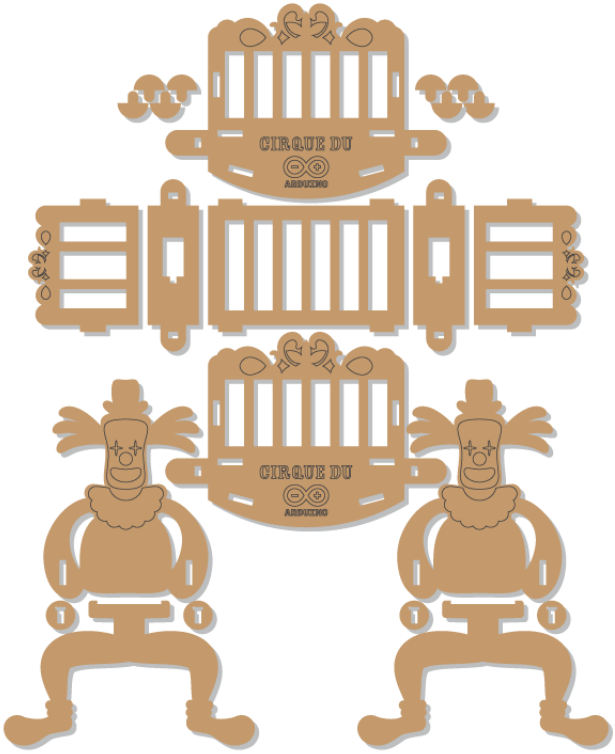
Robot gateador

Este pequeño robot puede gatear. Es todo lo que puede hacer, y tampoco es muy bueno en eso, pero es muy bonito y divertido.

Materiales

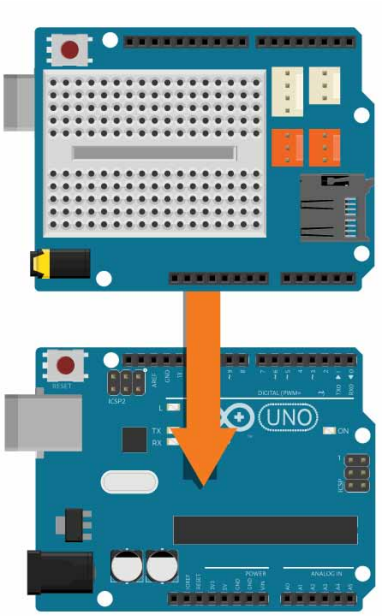
- 1 placa Arduino Uno
- 1 Shield Básica Educativa
- 2 motores servo estándar
- 8 Pilas AA
- 1 portapilas
- 1 conector a corriente
- 1 kit robot gateador



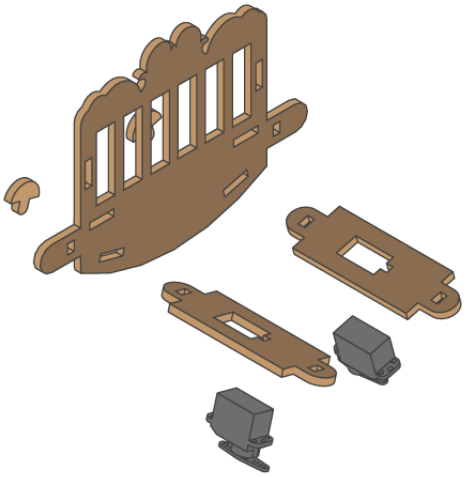


Instrucciones

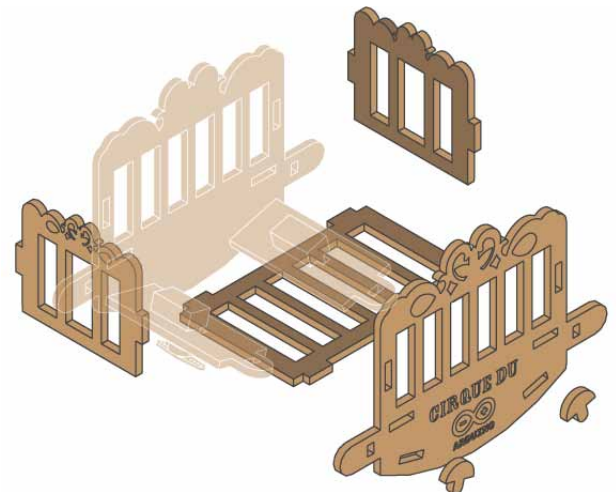
1. Conecta la shield en la parte superior de tu placa Arduino.



2. Monta el robot gateador.

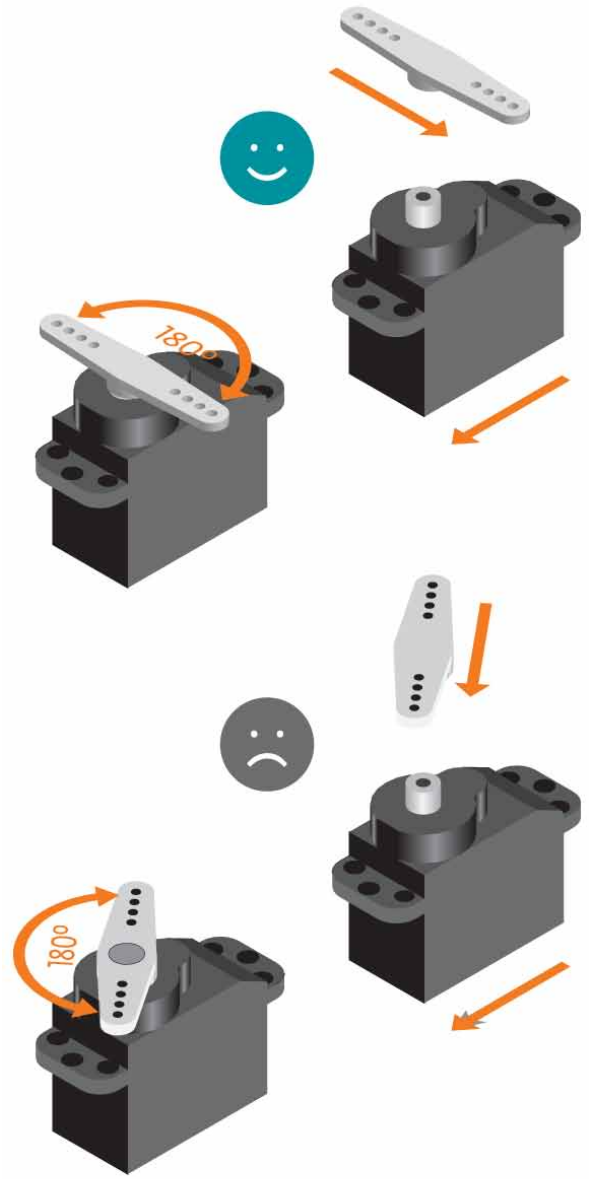


3.

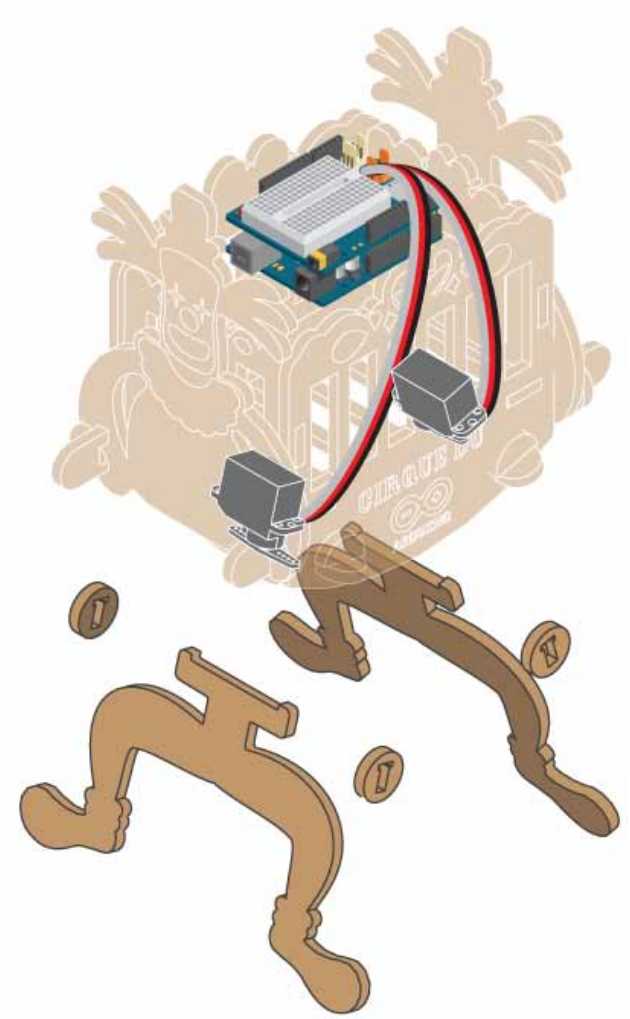


4.

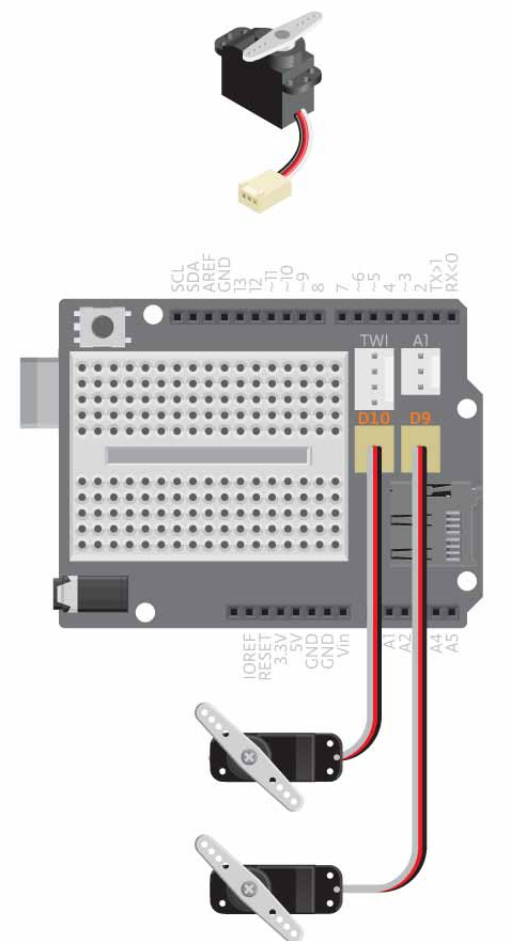
5. Gira el servo a mano hasta que llegue al final. A continuación, separa y vuelve a colocar el brazo del servo para que tenga la longitud del servo. Haz esto a ambos servos, delanteros como traseros.



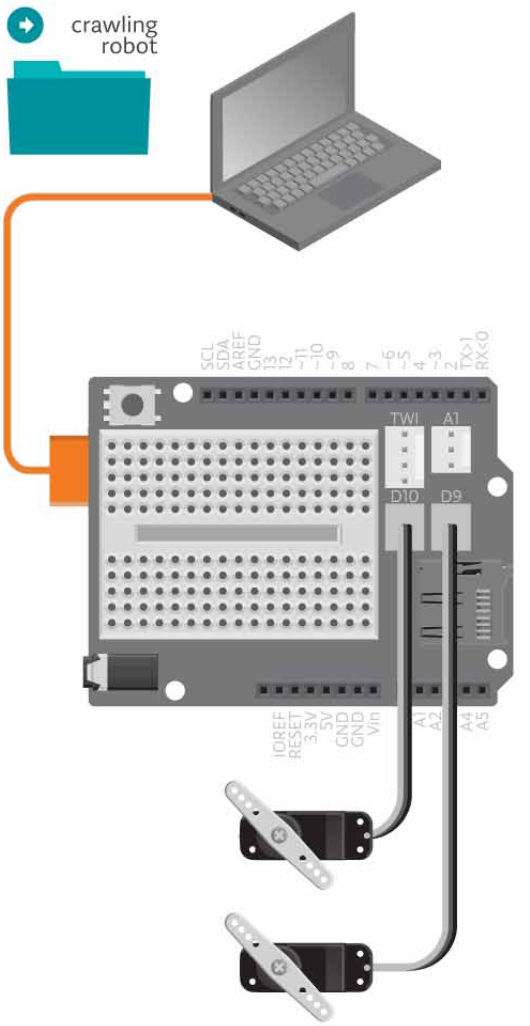
6. Monta un motor en cada agujero de servo y sujeta las piernas a los ejes de los servos.



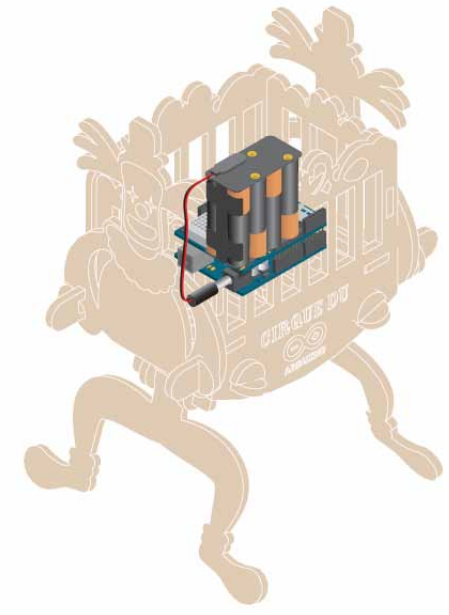
7. Conecta el servo delantero al pin D9. Conecta el servo trasero al pin D10.



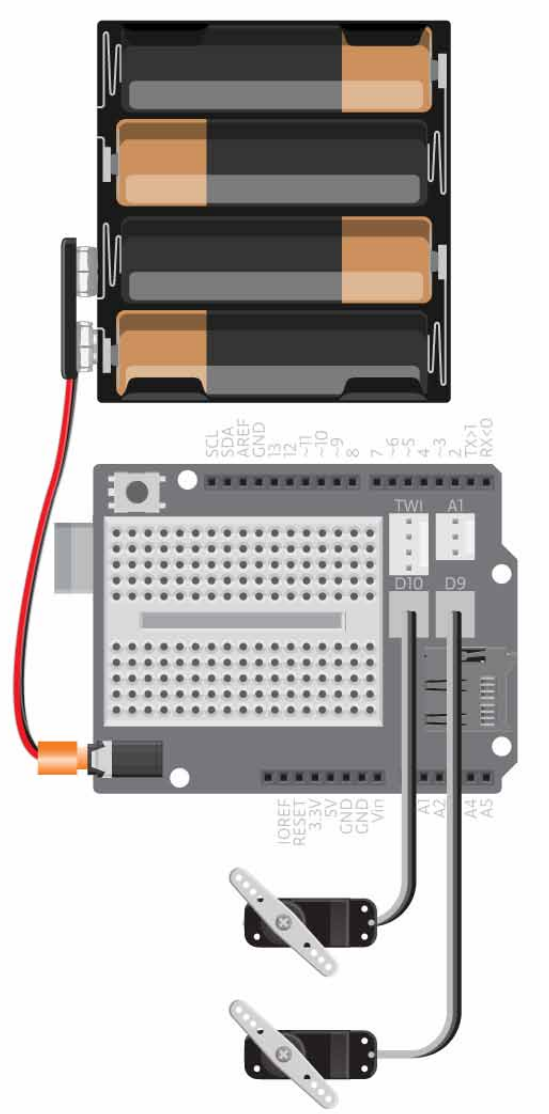
8. Carga el ejemplo CrawlingRobot.



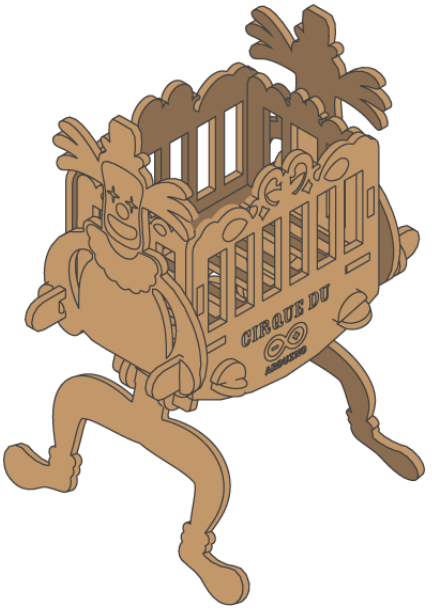
9. Desconecta tu Arduino del ordenador y colócalo en el "cuerpo" del robot gateador



10. Conecta la pila de 9V al portapilas y conéctalo al jack de alimentación de Arduino.



11.



Código

Puedes encontrar el código en: Archivo -> Ejemplos -> BasicEducationShield -> Robots -> CrawlingRobot

```
/*
  Crawling Robot

  This small robot can crawl. It's all it can do, and it isn't
  very good at it. But it is super cute and great fun.

  (c) 2013 Arduino Verkstad
*/

#include <BasicEducationShield.h>
#include <Servo.h>

Servo front, back;

void setup(){
  //servos are initialized
  front.attach(9);
  back.attach(10);
}

void loop(){
  //Make the robot crawl by setting the servos to opposite angles.
  //90 is the middle angle

  //First step of each pair of legs
  front.write(110);
  delay(200);
  back.write(70);
  delay(200);

  //Second step of each pair of legs
  front.write(70);
  delay(200);
  back.write(110);
  delay(200);
}
```

Cómo funciona

La posición de los servos frontales está ajustada a 120 grados. Esperamos a 200 milisegundos y luego se cambia la posición de los servos de atrás a 60 grados. Es decir, el ángulo opuesto a la parte frontal. Tras otros 200 milisegundos establecemos la posición del servo frontal a 60 grados. De nuevo 200 milisegundos más tarde, ajustamos el servo a la posición opuesta.

¿No funciona?

1. ¿No están funcionando los servos? ¿Funcionan cuando Arduino está conectado al ordenador pero no al utilizar la pila? Si es así, intenta cambiar la pila.

2. ¿Sigue sin funcionar? Asegúrate de que has conectado los servos correctamente. Prueba a establecer diferentes ángulos para cada uno de ellos de uno en uno con el ejemplo Archivo -> Ejemplos -> BasicEducationShield -> Help -> StandardServo.

¡Sigue experimentando!

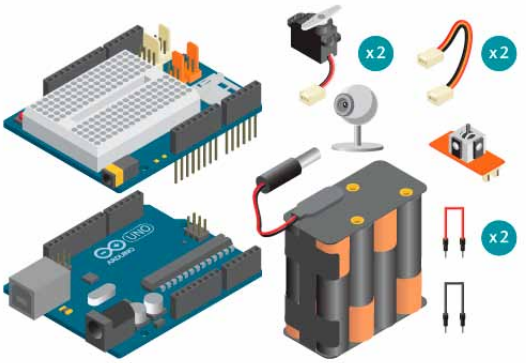
- ¿Quieres que el robot de pasos más grandes o más pequeños? Intenta cambiar los ángulos.
- ¿Quieres que el robot gatee más rápido o más lento? Intenta cambiar el tiempo de delay.
- Prueba a ver de cuántas maneras diferentes puedes hacer que el robot se arrastre y cuál es el más eficiente.

Cámara robótica

¡Espía a tu gato con esta cámara robótica! Monta la cámara espía y contrólala con un pequeño joystick. Puedes dirigir todo desde el ordenador y cuando veas algo sospechoso, ¡hazle una foto!

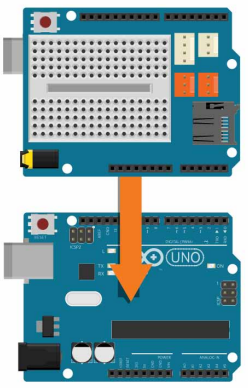
Materiales

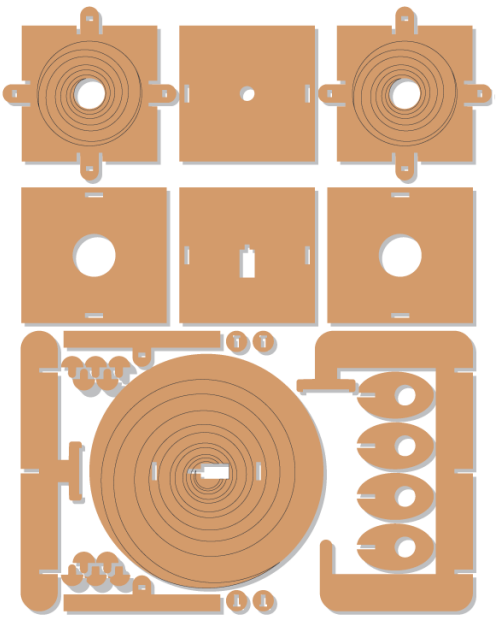
- 1 placa Arduino Uno
- 1 Shield Básica Educativa
- 2 servomotores estándar
- 1 joystick TinkerKit
- 3 cables
- 2 conectores TinkerKit
- 1 cámara web
- 1 kit de cámara robótica
- 8 pilas AA
- 1 portapilas
- 1 conector a corriente



Instrucciones

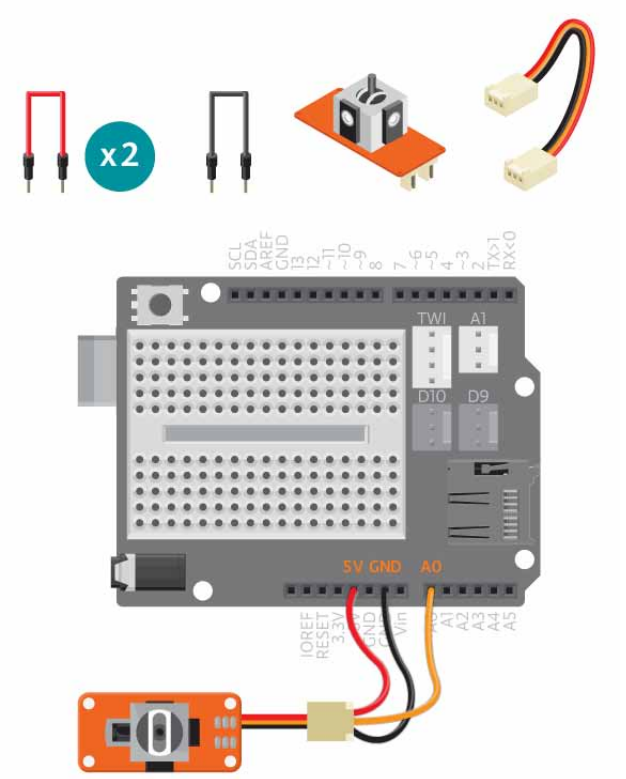
1. Conecta la shield a la parte superior de tu placa Arduino.

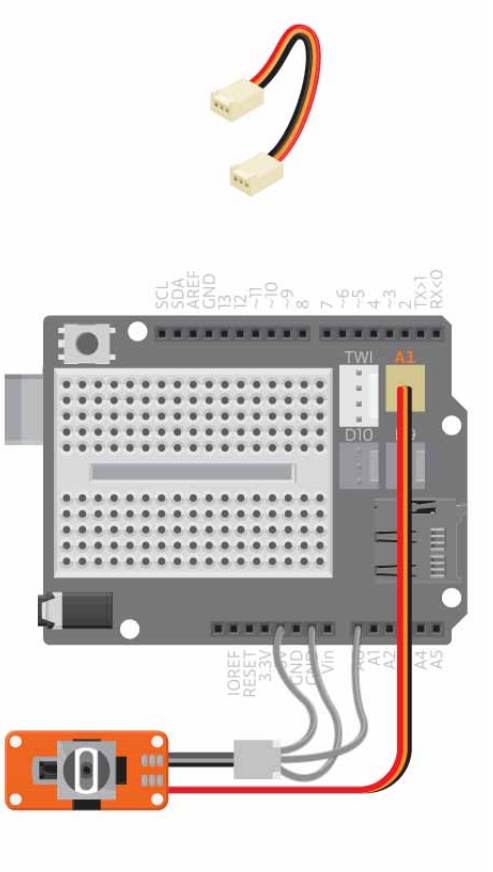




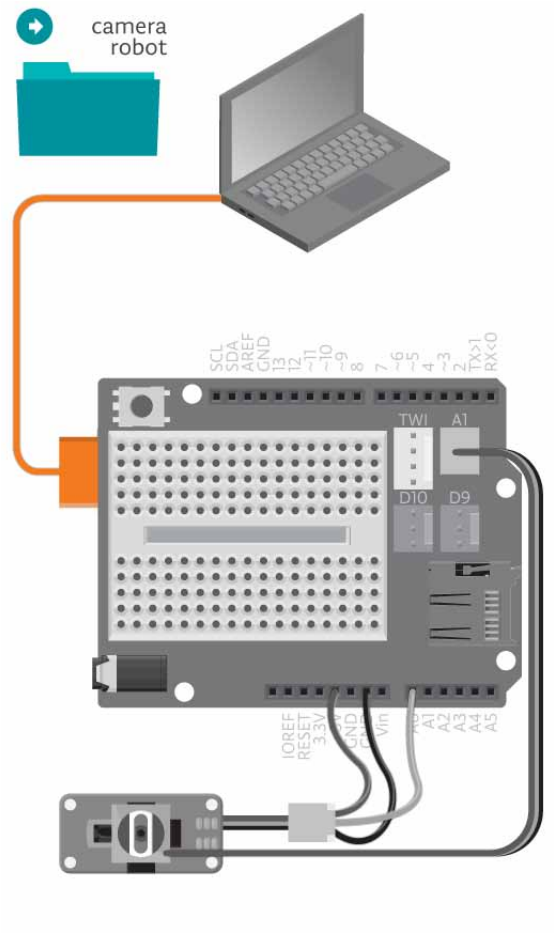
2. Conecta uno de los cables al joystick TinkerKit en el puerto marcado con Y. Conecta un cable rojo al extremo opuesto del cable TinkerKit asegurándote de que vaya a 5V (es decir, al cable rojo del Tinkerkit). Conecta un cable negro entre el conector negro TinkerKit y a tierra (es decir, a 0V). Conecta el último cable al TinkerKit y al Pin analógico A0.

3. Conecta el segundo cable TinkerKit al joystick en el puerto X y al conector analógico TinkerKit.

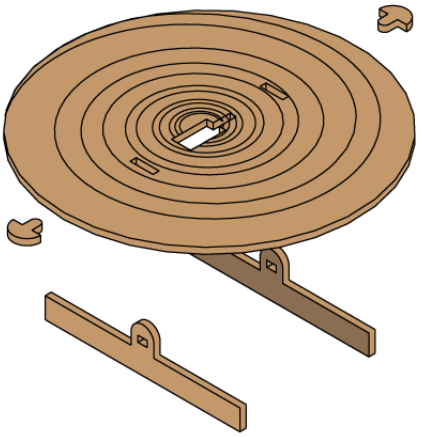




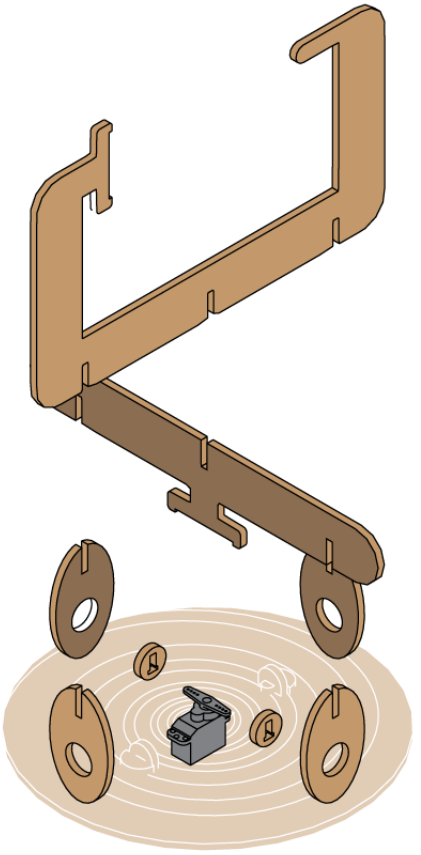
4. Carga el ejemplo Archivo -> Ejemplos -> BasicEducationShield -> Robots -> CameraRobot.



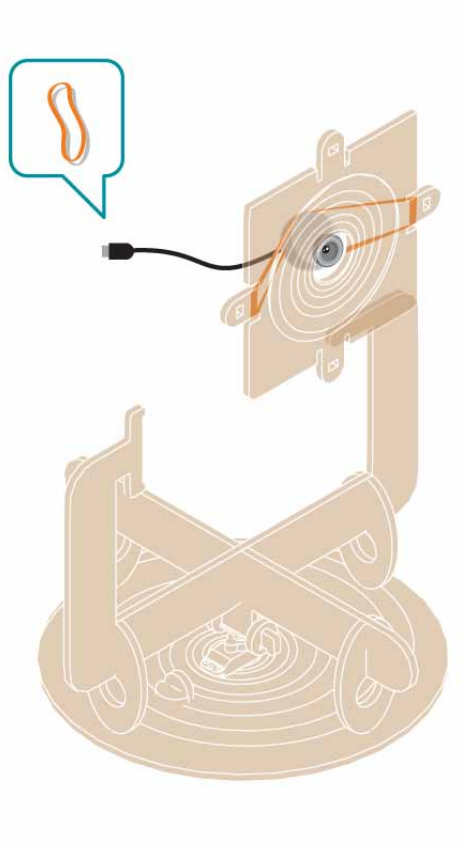
5. Construye el kit, ¡no olvides meter la webcam dentro!



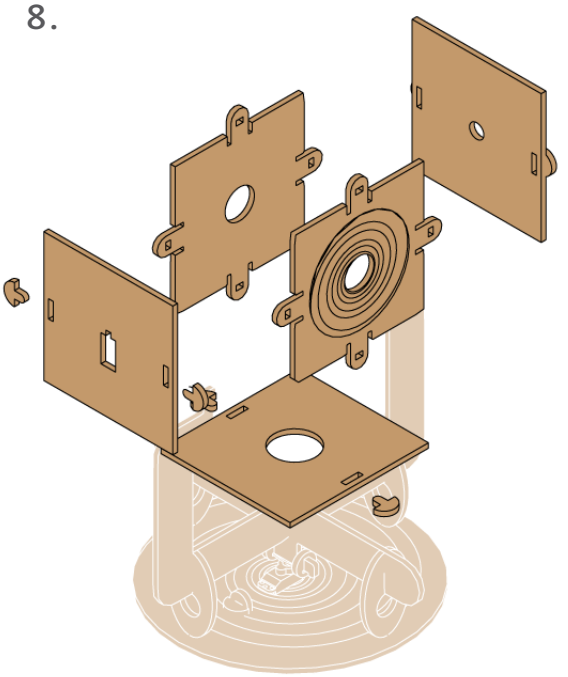
6.



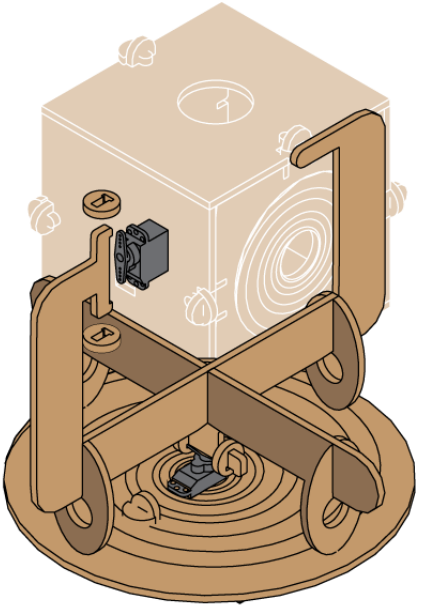
7.



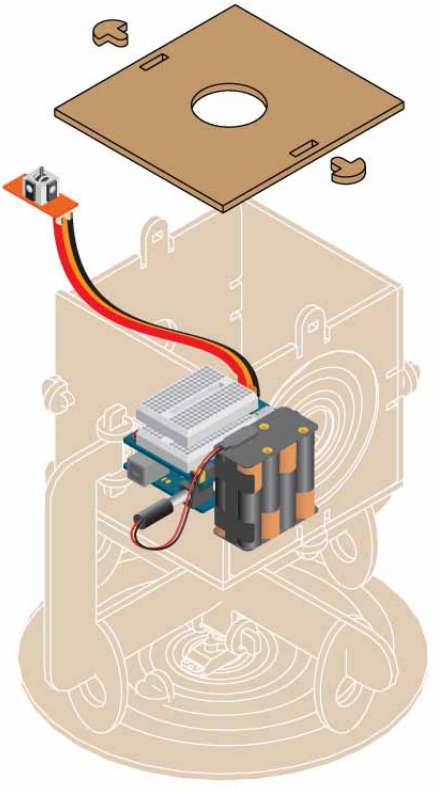
8.



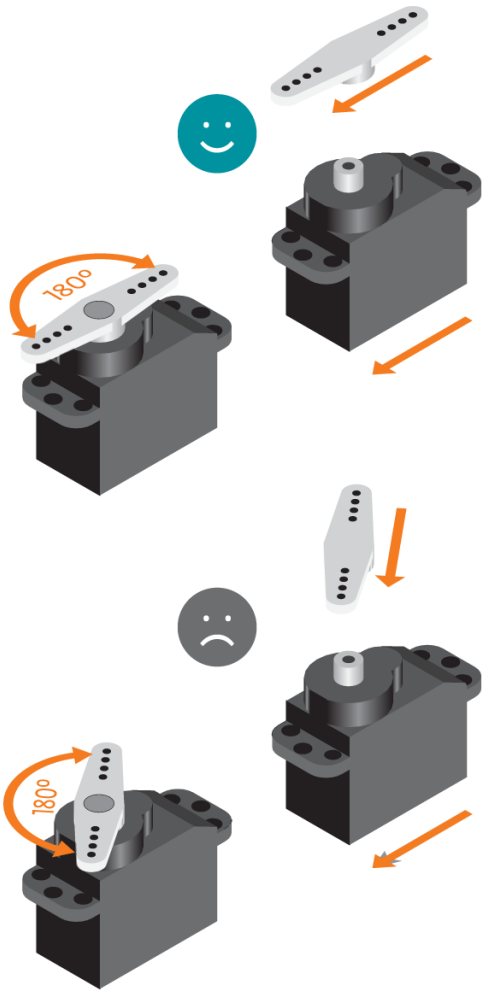
9.



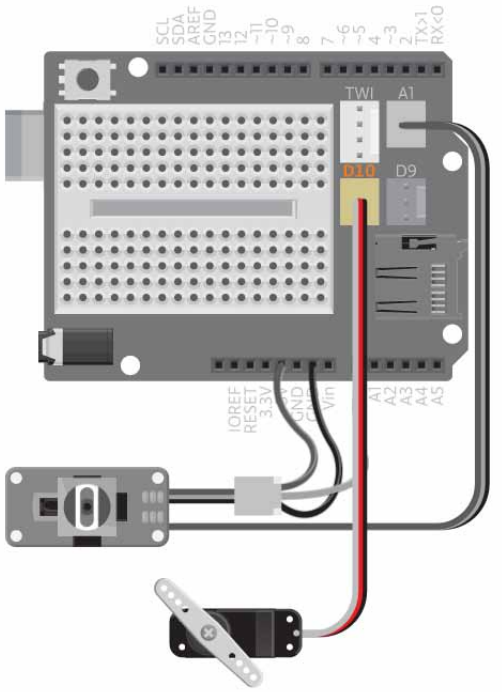
10.



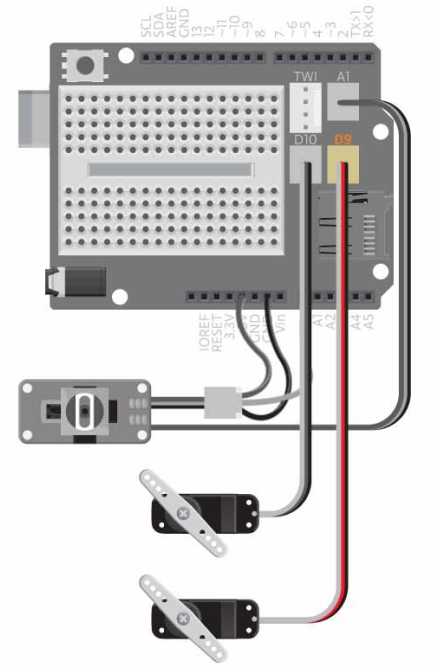
11. Gira el servo a mano hasta llegar al final de su recorrido. Gíralo a mano hasta la posición de 90 grados. Fija el brazo con forma de barra a lo largo del servo. Haz lo mismo con el otro servo.



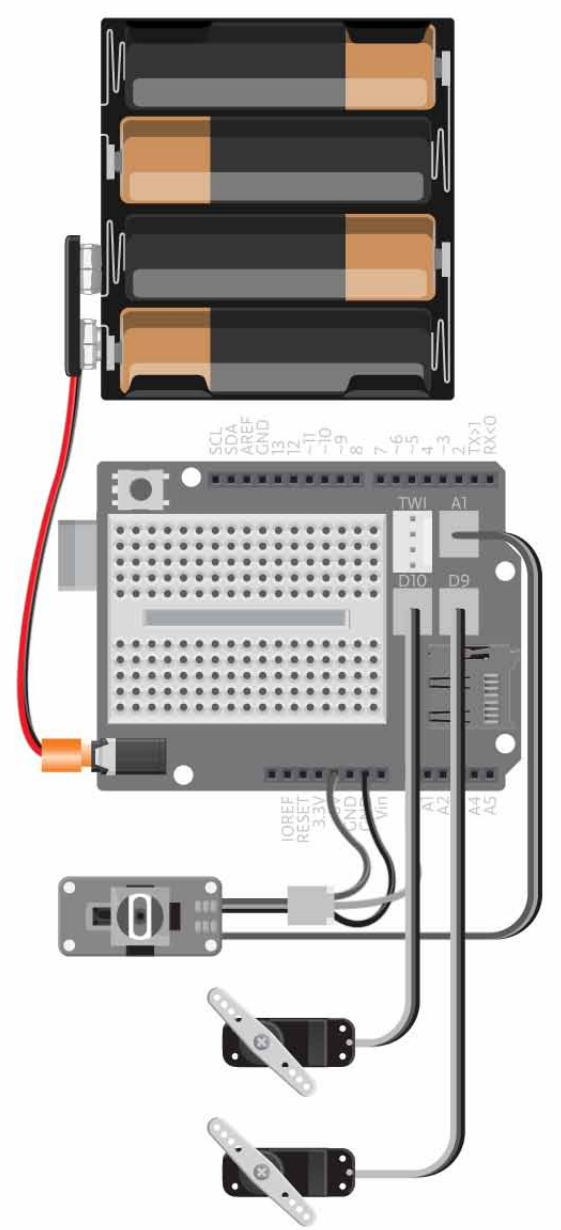
12. Conecta el servo lateral al pin D10.



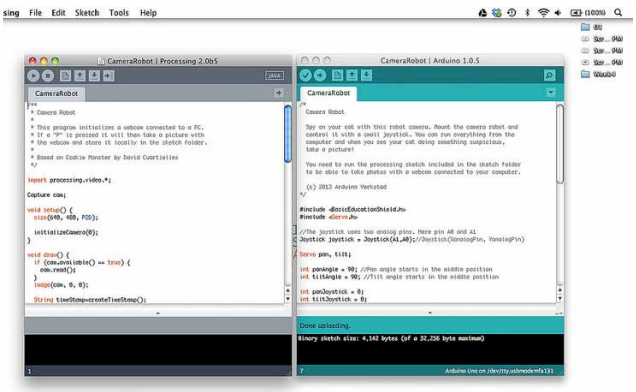
13. Conecta el servo inferior encargado de hacer el barrido panorámico al pin D9.



14. Conecta la batería al jack de alimentación.



15. Conecta la cámara web al ordenador. Abre el programa de Processing llamado CameraRobot y ejecútalo.



Código

Puedes encontrar el código en: Archivo -> Ejemplos -> BasicEducationShield -> Robots -> CameraRobot

```
/*
  Camera Robot

  Spy on your cat with this robot camera. Mount the camera robot and
  control it with a small joystick. You can run everything from the
  computer and when you see your cat doing something suspicious,
  take a picture!

  You need to run the processing sketch included in the sketch folder
  to be able to take photos with a webcam connected to your computer.

  (c) 2013 Arduino Verkstad
*/

#include <BasicEducationShield.h>
#include <Servo.h>

//The joystick uses two analog pins. Here pin A0 and A1
Joystick joystick = Joystick(A1,A0);//Joystick(XanalogPin, YanalogPin)

Servo pan, tilt;

int panAngle = 90; //Pan angle starts in the middle position
int tiltAngle = 90; //Tilt angle starts in the middle position

int panJoystick = 0;
int tiltJoystick = 0;

void setup(){
  //servos are initialized
  pan.attach(9);
  tilt.attach(10);
}
void loop(){
  panJoystick = joystick.getX(); //Get X value from joystick
  tiltJoystick = joystick.getY(); //Get Y value from joystick

  //If the joysticks X value isn't 0 we will pan
  if(panJoystick!=0){
    //If the X value from the joystick equals 1 and
    //panAngle is less than 180 degrees, increase panAngle with 1
    if(panJoystick==1 && panAngle<180) panAngle++;
    //If the X value from the joystick equals -1 and
    //panAngle is more than 0 degrees, decrease panAngle with 1
    else if(panJoystick==-1 && panAngle>0) panAngle--;
  }
  //If the joysYicks Y value is 0 we will tilt
```

```
48   if(tiltJoystick!=0){
49     //If the Y value from the joystick equals 1 and
50     //tiltAngle is less than 180 degrees, increase tiltAngle with 1
51     if(tiltJoystick==1 && tiltAngle<180) tiltAngle++;
52     //If the Y value from the joystick equals -1 and
53     //tiltAngle is more than 0 degrees, decrease tiltAngle with 1
54     else if(tiltJoystick==-1 && tiltAngle>0) tiltAngle--;
55   }
56   pan.write(panAngle); //Set position of the pan servo
57   tilt.write(tiltAngle); //Set position of the tilt servo
58   delay(5);
59 }
```

Cómo funciona

Leemos los valores del joystick. Estos valores serán o bien 0, -1 o 1 para cada eje. 0 es si el joystick está en el medio, -1 es uno de los lados y 1 es el otro lado.

Ya que sólo podemos mover un servo a la vez, tenemos que comprobar a cuál vamos a escribir. Si el valor X del joystick no es 0, desconectamos por software con `dettach` el servo que controla la inclinación (tilt en inglés) para controlar el servo de giro (pan en inglés). A continuación, comprobamos a qué lado se está moviendo el joystick y luego agregamos o quitamos una unidad del ángulo de giro.

Si de contrario, el valor X del joystick es 0, significa que no está siendo movido en el eje X. Así que en su lugar desconectamos el servo de giro y conectamos el servo de inclinación. A continuación, comprobamos el movimiento del joystick en el eje Y para escribir el valor del ángulo al servo correspondiente.

¿No funciona?

1. Asegúrate de que has conectado los servos correctamente. Mira la referencia para asegurarte de que los servos funcionan.

2. Asegúrate de que el joystick esté conectado correctamente. Mira la referencia para asegurarte de que funciona.

¡Sigue experimentando!

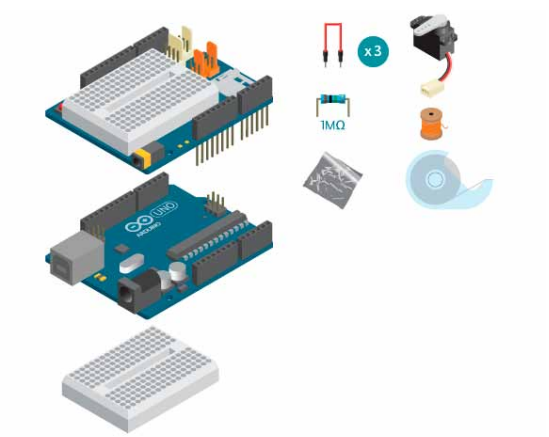
- ¿Se están moviendo los servos demasiado lento o demasiado rápido? Intenta cambiar el delay al final del loop o intenta cambiar el aumento y la disminución de los ángulos.
- ¿Los servos se mueven demasiado a los lados, o demasiado hacia arriba y abajo? Intenta limitar los ángulos permitidos.

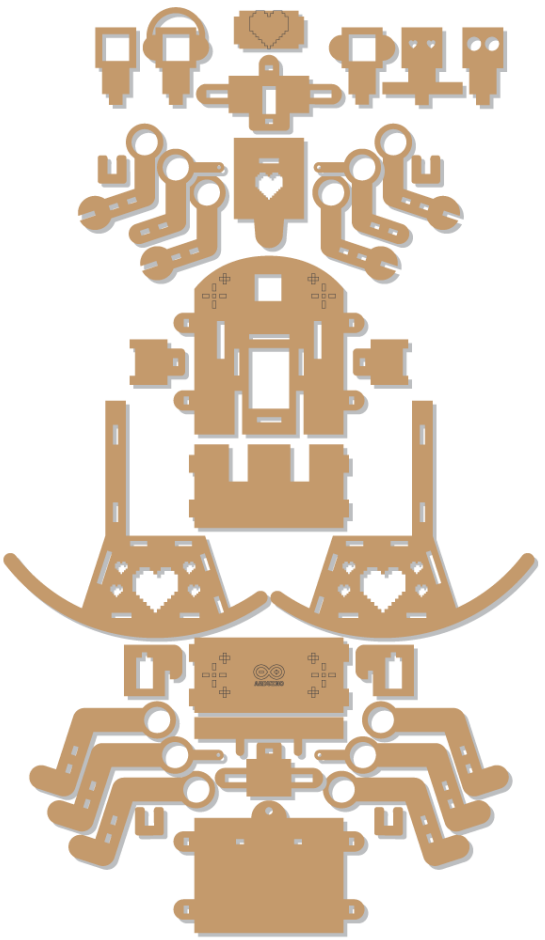
Tickle robot

Este es un robot normal con cosquillas en una mecedora. Si lo piensas bien, no suena nada normal. Hazle cosquillas en su corazón y empezará a menearse como loco.

Materiales

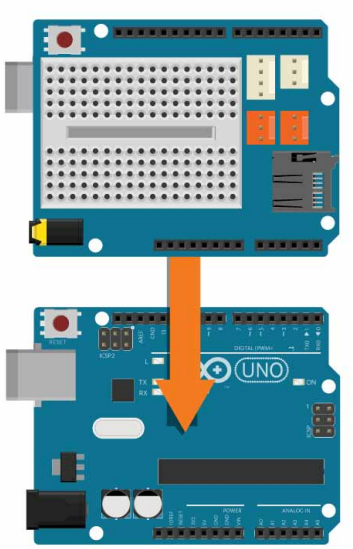
- 1 placa Arduino Uno
- 1 Shield Básica Educativa
- 1 servo estándar
- 1 resistencia de 1MOhm
- 3 cables
- papel de aluminio
- cinta adhesiva
- cuerda
- 1 kit TickleRobot
- 1 breadboard



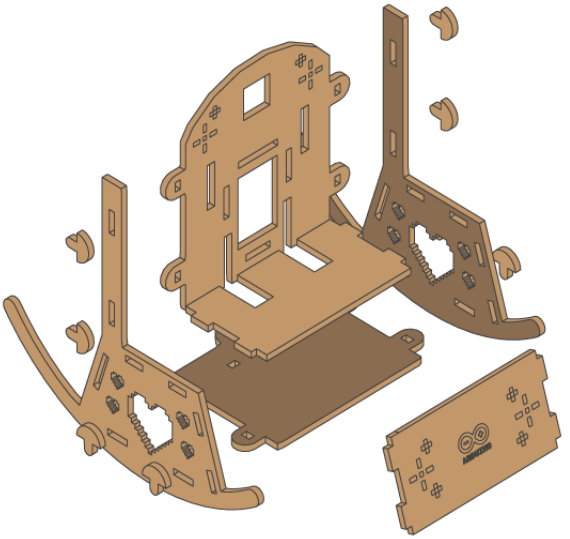


Instrucciones

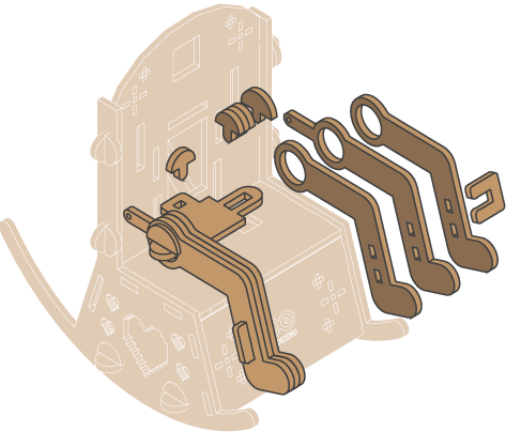
1. Conecta la shield a la parte superior de tu placa Arduino.



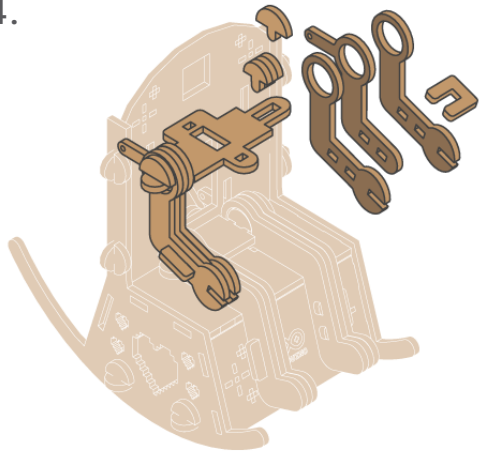
2. Construye el TickleRobot.



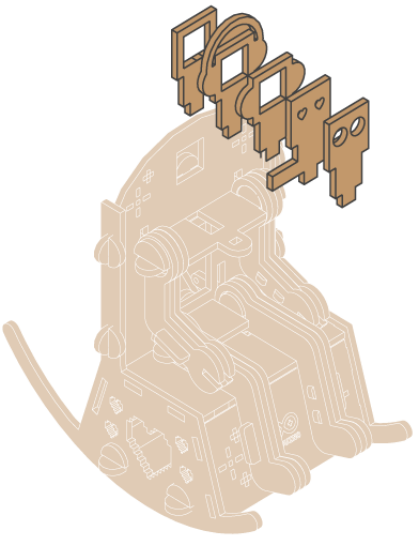
3.



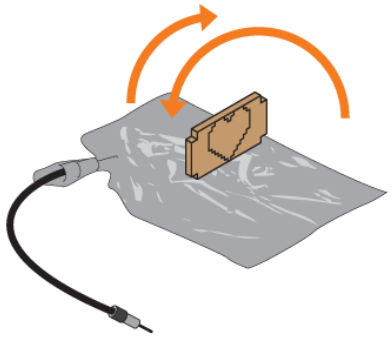
4.



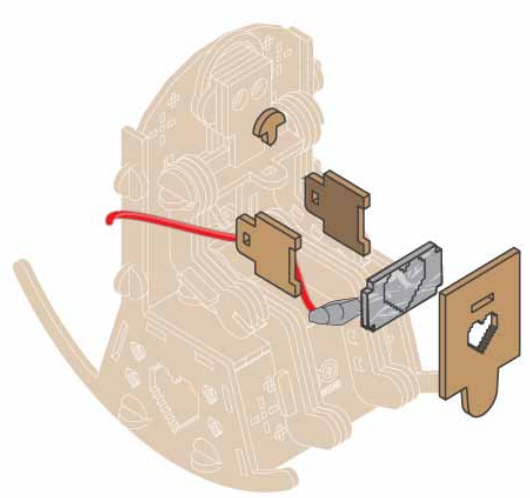
5.



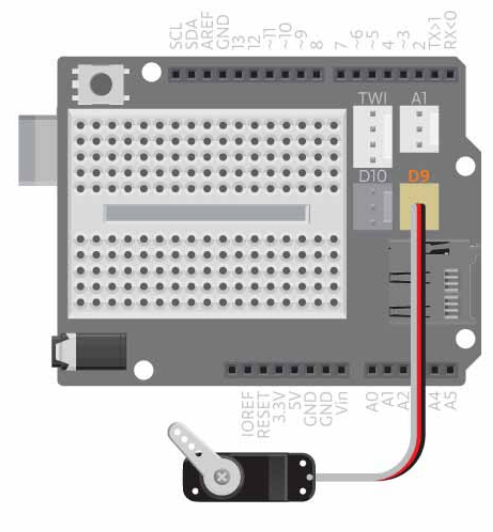
6. Haz un sensor de contacto cortando una tira de papel de aluminio. Envuelve el papel alrededor del corazón de la placa de TickleRobot, utiliza cinta adhesiva alrededor de los bordes para que se mantenga. Desde la parte posterior de la placa del corazón, envuelve el papel de aluminio a un cable suelto -el cable metálico debe estar en contacto con el papel de aluminio-. Pega la conexión firmemente.



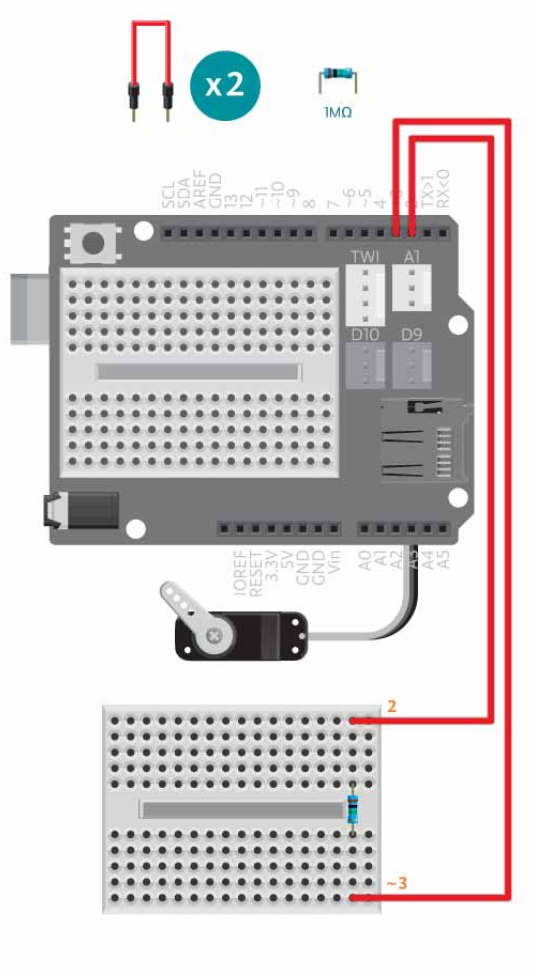
7.



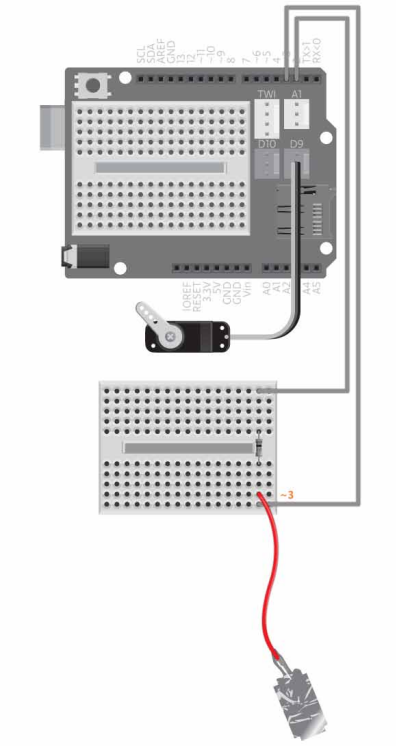
8. Conecta el cable del servo al pin D9.



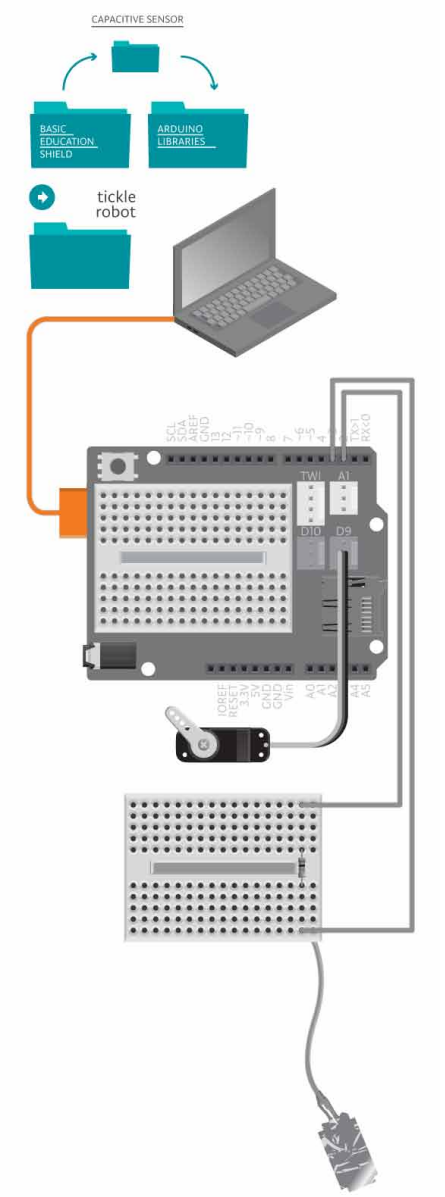
9. Conecta una resistencia de 1Mohm entre el Pin digital 2 y el Pin digital 3.



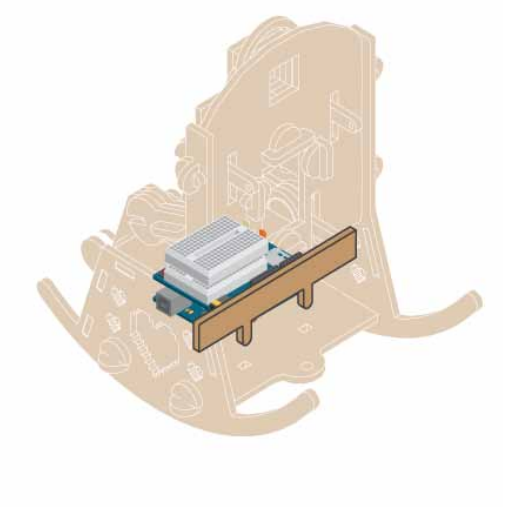
10. Conecta el cable suelto del sensor capacitivo al Pin digital 3.



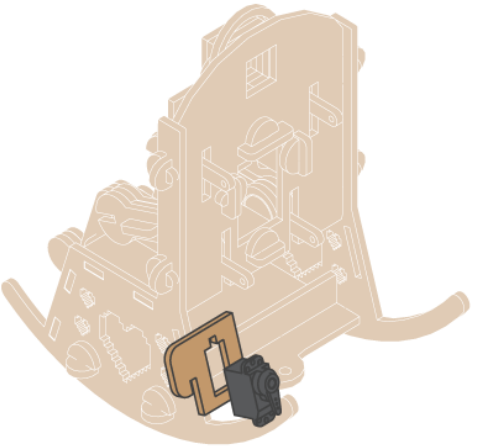
11. Conecta tu Arduino al ordenador. Asegúrate de que la librería CapacitiveSensor está instalada dentro de las librerías de tu carpeta sketchbook/ libraries. Carga el ejemplo Archivo -> Ejemplos -> BasicEducationShield -> Robots -> TickleRobot. Asegúrate de que el servo se mueve cuando toques el sensor capacitivo.



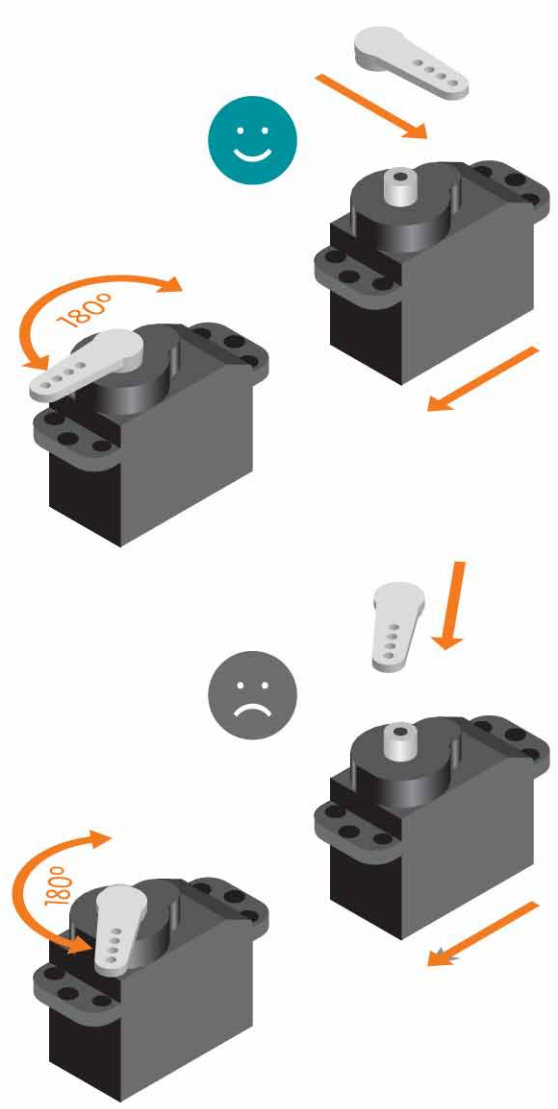
12. Coloca la electrónica en la parte inferior del tickle robot y encaja el servo en la parte izquierda.



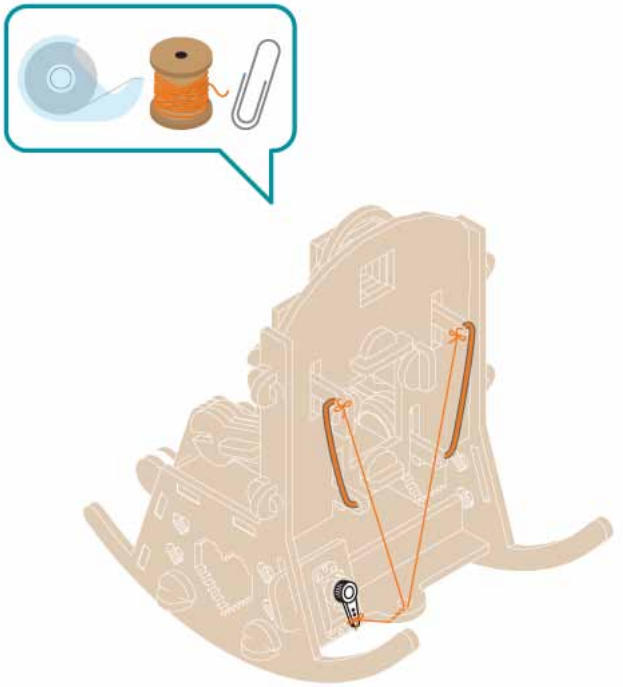
13.



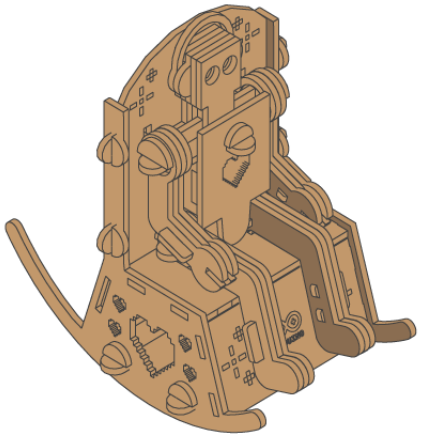
14. Gira el servo a mano hasta llegar al final de su recorrido. Gíralo a mano hasta la posición de 90 grados. Fija el brazo con forma de barra a lo largo del servo.



15. Ata una hilo al brazo y pierna izquierda, a través del agujero extremo del brazo del servo. Haz lo mismo con el brazo y pierna derecha y átalos al mismo sitio.



16.



Código

Puedes encontrar el código en: Archivo -> Ejemplos -> BasicEducationShield -> Robots -> TickleRobot

```
/*
    Tickle Robot

    This is just a regular ticklish robot in a rocking chair.
    Come to think of it, that doesn't sound very regular at all.
    Tickle the robot on its heart and it will start wiggle like crazy.

    (c) 2013 Arduino Verkstad
*/

#include <BasicEducationShield.h>

//Necessary to include Servo.h when using Servo
#include <Servo.h>

//Necessary to include CapacitiveSensor.h when using capacitive sensor
#include <CapacitiveSensor.h>

//Declare the servo for controlling the string robot
Servo pull;

//Declare the capacitive sensor
CapacitiveSwitch sensor=CapacitiveSwitch(2,3);

void setup(){
  //initialize the capacitive sensor. Threshold is 400
  //See the example CapacitiveSwitchTest in the Help folder
  //to find the right thresshold
  sensor.config(400);

  //initialize the servo motor
  pull.attach(9);
}
void loop(){
  if(sensor.getState()){
    //If the capacitive sensor is touched, pull the strings
    pull.write(0);
  }else{
    //Otherwise, loosen the strings
    pull.write(90);
  }
  delay(30);

}
```

Cómo funciona

Cuando el sensor de contacto detecta ser tocado, el servomotor se mueve hacia un determinado ángulo para que el muñeco levante sus brazos y piernas. Cuando el sensor no detecta el toque, el servo se mueve hacia otro ángulo que hace que el muñeco baje brazos y piernas.

¿No funciona?

- ¿No reacciona a las cosquillas o el servo no se mueve hacia el ángulo correcto? Primero, mira la referencia del Servo Estándar para corregir errores del mismo. Si todavía no funciona, mira la referencia del Sensor Capacitivo para corregir el sensor de tacto.
- ¿No puedes cargar el código? Asegúrate de que la librería CapacitiveSensor está situada dentro de la carpeta de libraries en tu sketchbook.

¡Sigue experimentando!

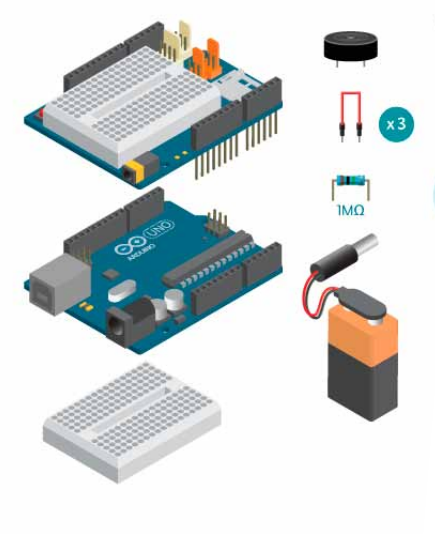
1. ¿Puedes hacer que el TickleRobot baile de una manera predeterminada? Hazle bailar justo después de tocarlo.
2. ¡Intenta añadir otro servo para que pueda hacer movimientos de baile más avanzados!
3. ¡Haz que el robot mueva la silla por si mismo! utiliza el sensor capacitivo para activarlo y desactivarlo.

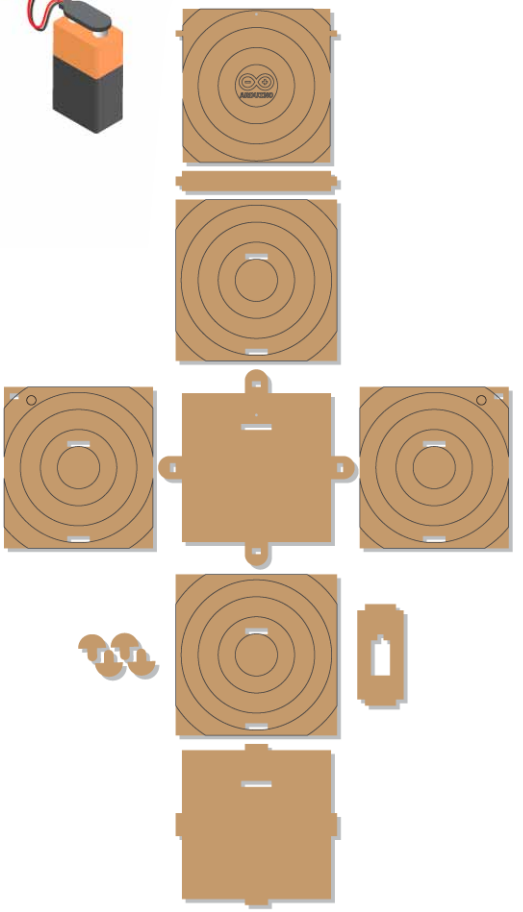
Abre la caja

Esta es una caja 'open source' (o de código abierto) que contiene electrónica libre. Ah, y se abre automáticamente cuando la llamas golpeándola.

Materiales

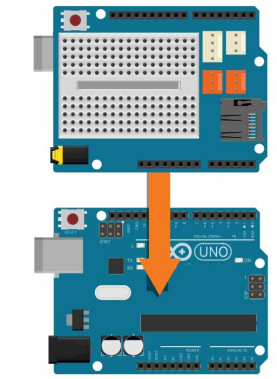
- 1 placa Arduino Uno board
- 1 Shield Básica Educativa
- 1 servo estándar
- 1 piezo
- 1 resistencia 1Mohm
- 3 cables
- 1 pilas 9V
- 1 portapilas 9V
- 1 conector a corriente
- 1 kit OpenBox
- cinta adhesiva
- hilo
- 1 breadboard



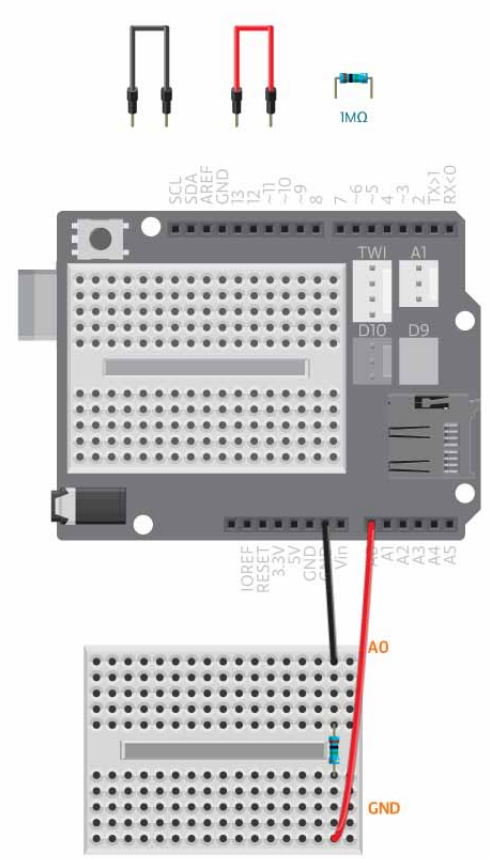


Instrucciones

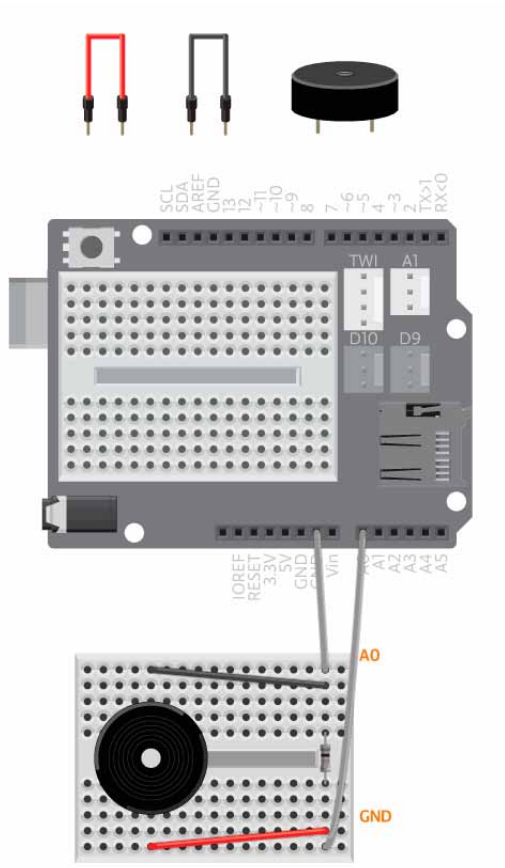
1. Conecta la shield a la parte superior de tu placa Arduino.



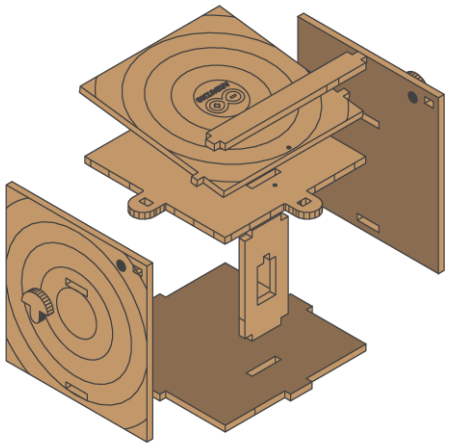
2. Conecta la resistencia de 1Mohm al Pin analógico A0 y a GND a través de placa.



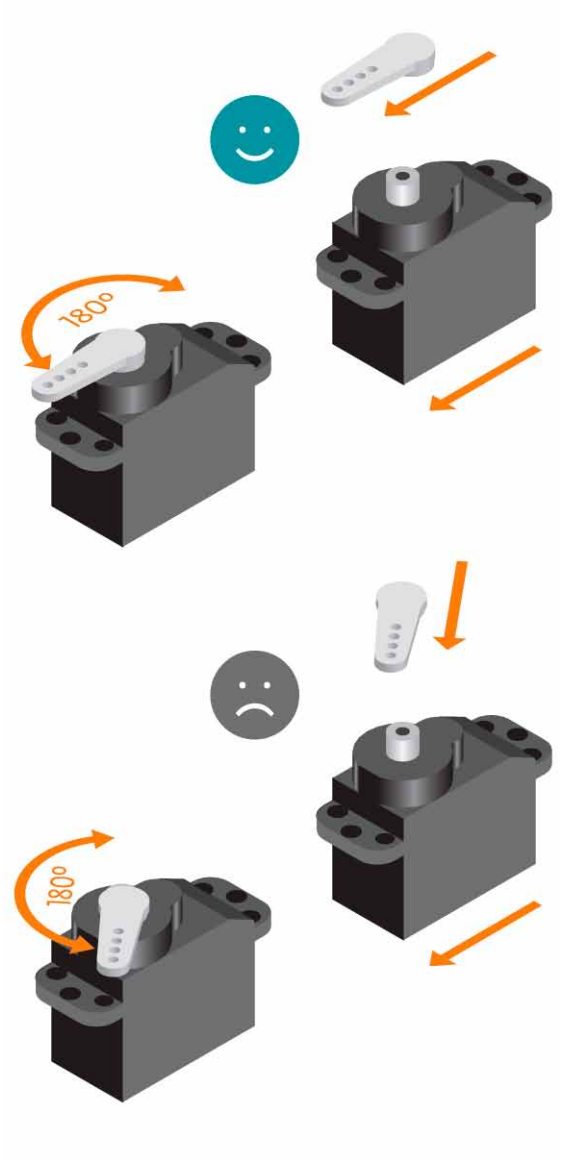
3. Conecta una pata del piezo al Pin analógico A0, y la otra a GND en paralelo con la resistencia.



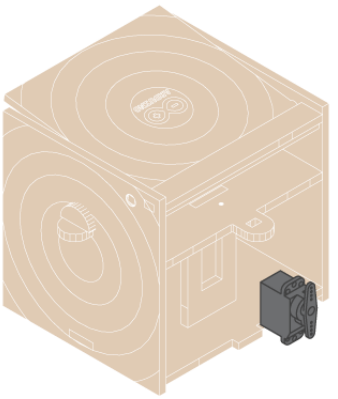
4. Construye el OpenBox kit pero sin montar la tapa trasera.



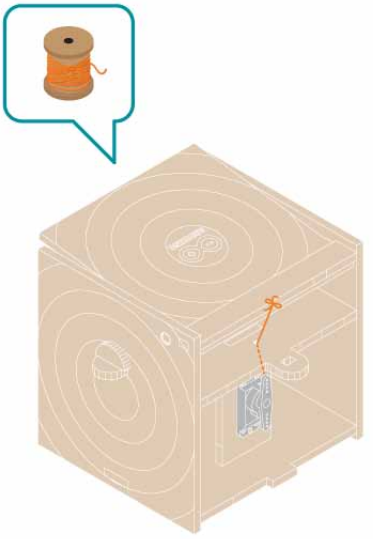
5. Gira el servo a mano hasta llegar al final de su recorrido. Gíralo a mano hasta la posición de 90 grados. Fija el brazo con forma de barra a lo largo del servo.



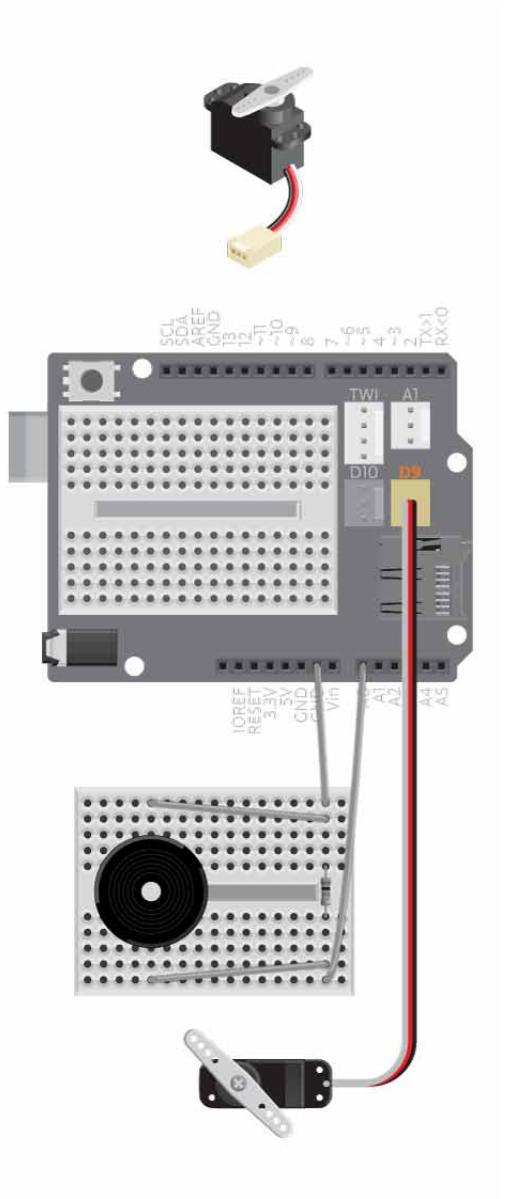
6.



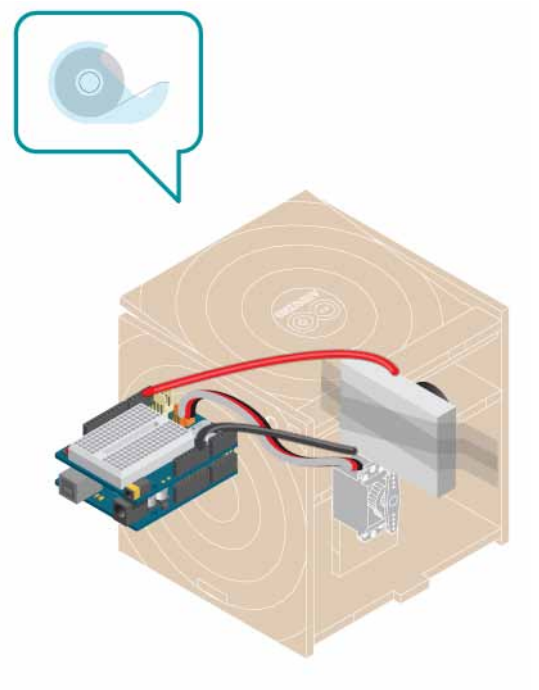
7. Ata la cuerda al servo. Pásala por el agujero de en medio de la placa de madera y ata el otro lado a la tapa.



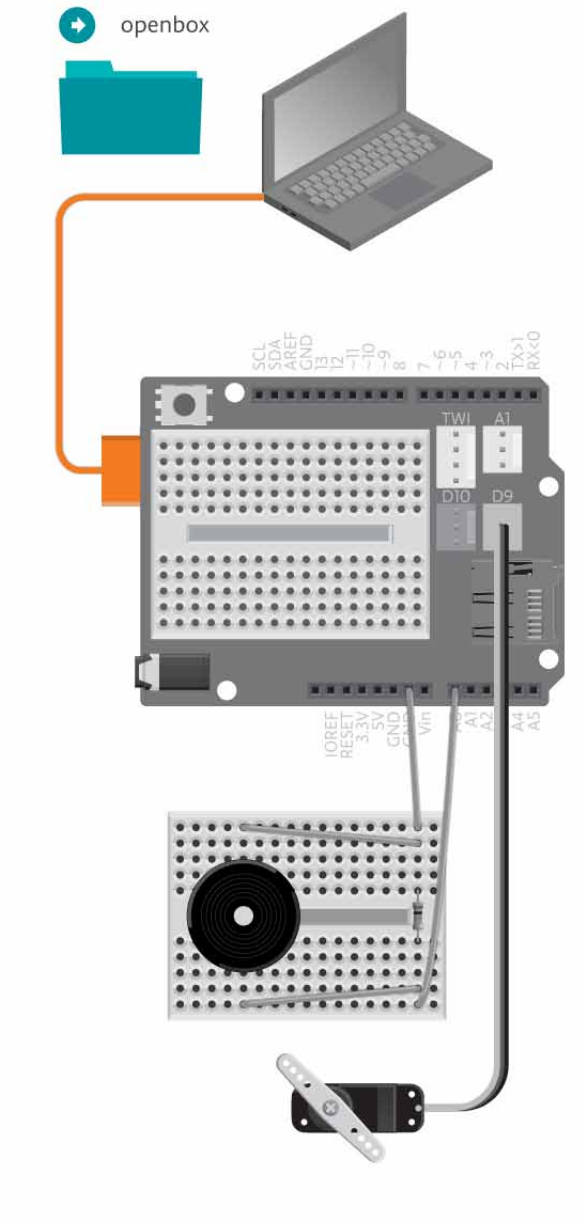
8. Conecta el servo al conector D9.



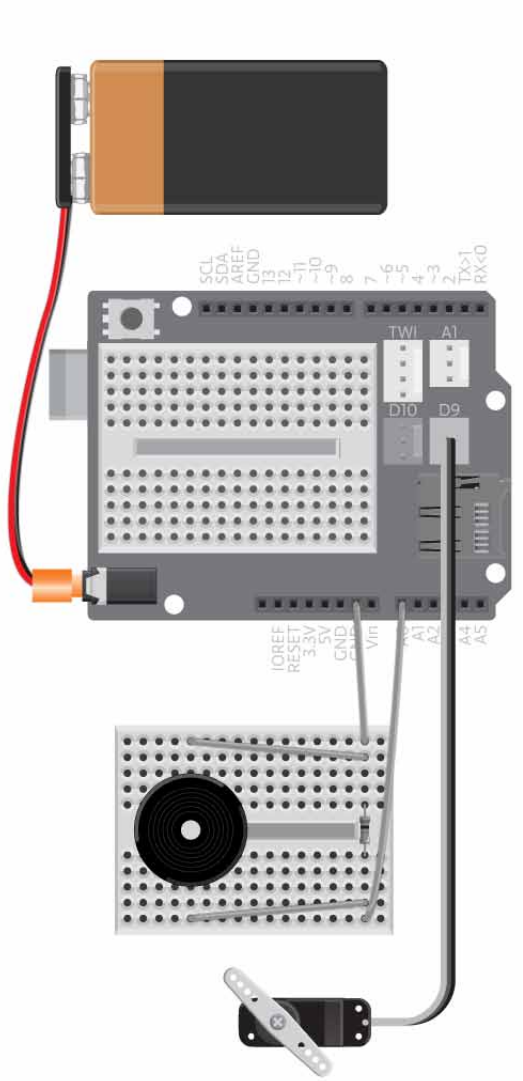
9. Coloca el Arduino dentro de la caja. Pega el piezo a la pared interior de la caja.



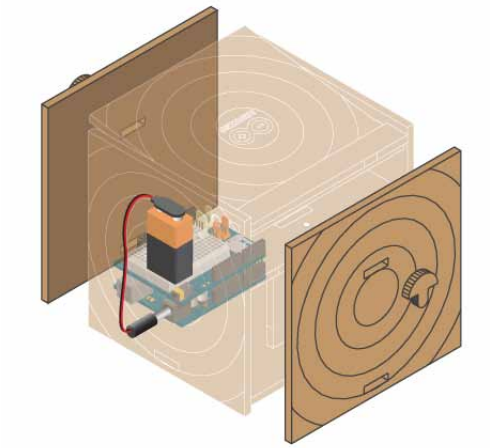
10. Conecta tu Arduino al ordenador y carga ejemplo Archivo -> Ejemplos -> BasicEducationShield -> Robots -> OpenBox.



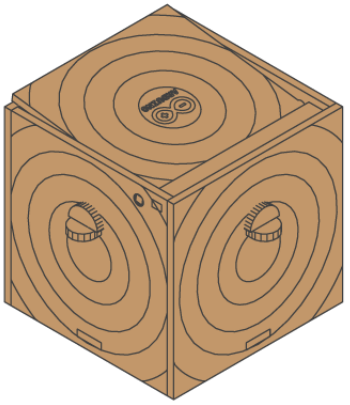
11. Conecta la pila de 9V al Arduino antes de montar la última pared.



12.



13.



Código

Puedes encontrar el código en: Archivo -> Ejemplos -> BasicEducationShield -> Robots -> OpenBox

```
/*
  OpenBox

  OpenBox is an open source box that contains open source
  electronics. Oh, and it automatically opens when you
  knock on it.

  (c) 2013 Arduino Verkstad
*/

#include <BasicEducationShield.h>

//It's necessary to include Servo.h if servo is used
#include <Servo.h>

//Declare the pezio knock sensor. Connected to A0
PiezoKnockSensor sensor=PiezoKnockSensor(A0);

//Declare the servo motor for opening the lid
Servo lidOpener;

void setup(){
  //define the threshold and debounce time of the knock
  //sensor. Threshold defines how hard you need to knock,
  //debounce time prevents the sensor from detecting
  //false knocks, but also limits how rapid you can knock.
  //See the PiezoKnockSensor sketch in the help folder
  //to make sure your values are correct
  sensor.config(40,80);

```

```
  //initialize the servo
  lidOpener.attach(9);
  //Rotate servo to close lid
  lidOpener.write(60);
}
void loop(){

  if(sensor.knocked()){
    //rotate the servo motor to open the lid
    lidOpener.write(0);
    delay(3000); //Wait for 3 seconds
    //close the lid
    lidOpener.write(60);
    delay(200);
  }

}
```

Cómo funciona

Cuando el sensor piezoeléctrico detecta un golpe, el servo se posiciona en un ángulo determinado para que la tapa se levante. Tras esperar un tiempo, el servo hace que la tapa se cierre. Luego el programa se reinicia.

¿No funciona?

1. ¿La tapa no se abre / cierra por completo? Ajusta los valores en `write()` para arreglarlo. Pero primero, ajusta el ángulo del brazo del servo.

2. ¿La caja no hace nada cuando le das un toque? Mira la referencia para corregir errores del sensor piezoeléctrico de vibración. Si aún no funciona, consulta la referencia para corregir errores del servo estándar.

¡Sigue experimentando!

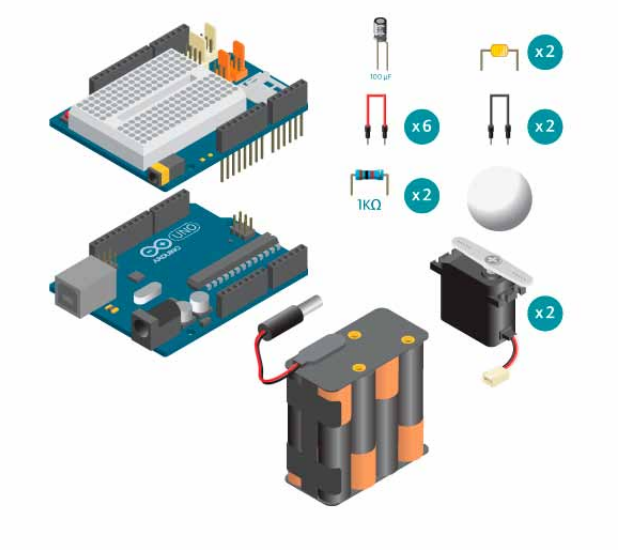
- ¡Haz una caja de seguridad! Para que así tengas que abrirlo con un patrón de toques.
- Modifícalo de tal manera que tocando la caja se abra o se cierre, en lugar de abrirse y luego cerrarse automáticamente como hace ahora.

Cazador de luz

Este pequeño vehículo tiene la eterna e imposible misión de atrapar la luz. Puedes hacer que te siga enfocándole una luz, la de tu teléfono móvil por ejemplo. Siempre girará a donde esté la luz.

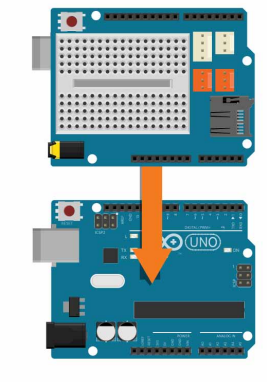
Materiales

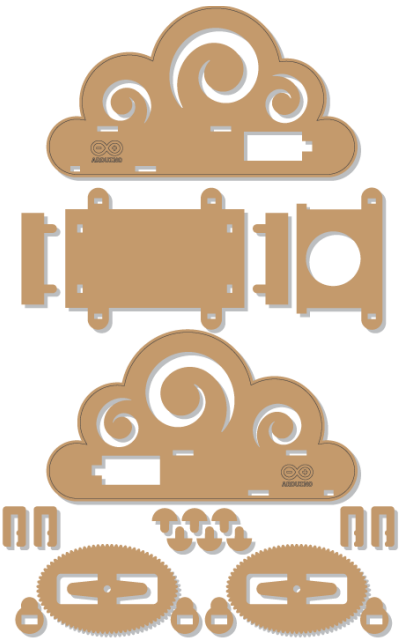
- 1 placa Arduino Uno
- 1 shield Básica Educativa
- 2 servomotores de giro continuo
- 1 condensador de 100 µF
- 2 sensores LDR
- 2 resistencias 1kohm
- 2 cables negros
- 6 cables de colores
- 8 pilas AA
- 1 conector a corriente
- 1 portapilas
- 1 kit Vehículo
- 1 pelota de ping pong



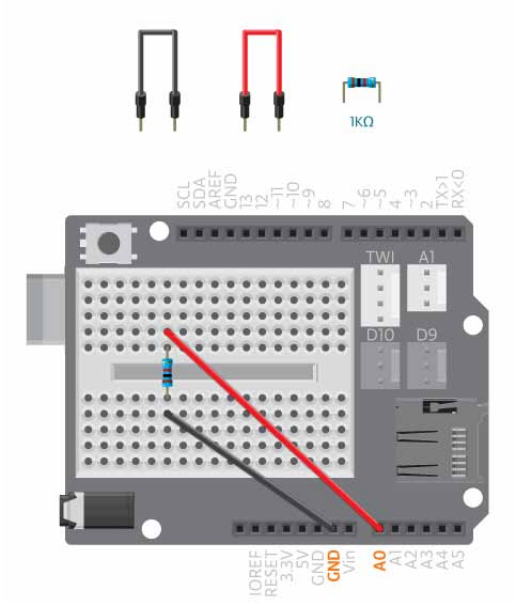
Instrucciones

1. Conecta la shield a la parte superior de tu placa Arduino.

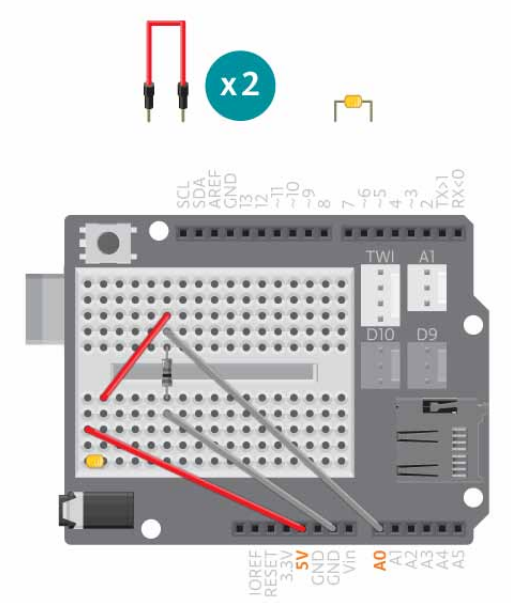




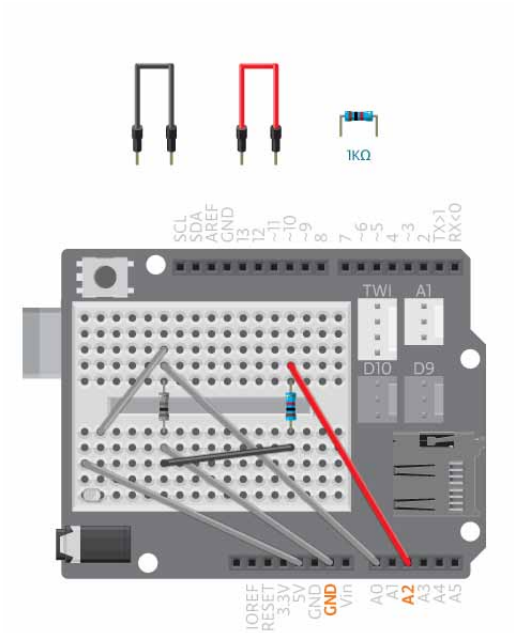
2. Conecta una resistencia de 1kohm entre GND y el pin analógico A0, a través del puente de la breadboard.



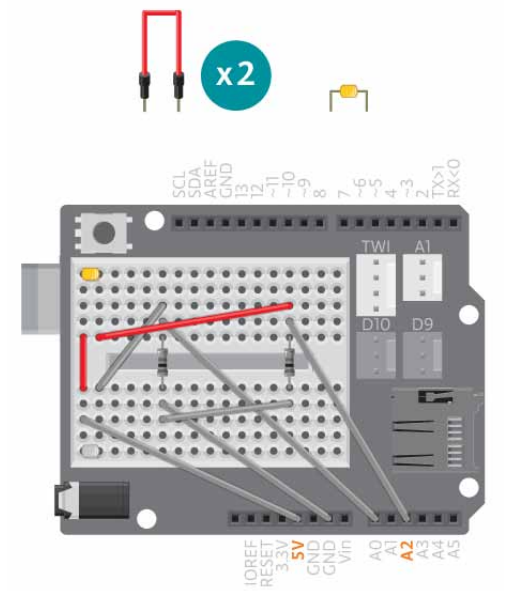
3. Conecta un LDR entre Pin analógico A0 y el alimentador. Colócalo en el lado izquierdo del vehículo.



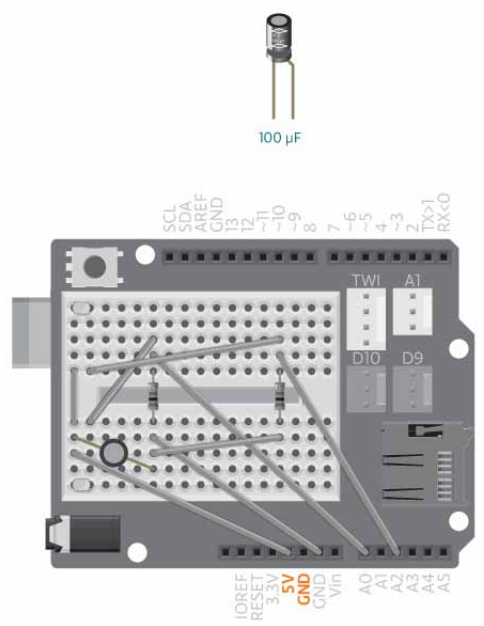
4. Conecta otro LDR como se muestra en las siguientes imágenes. Utiliza el Pin analógico A2.



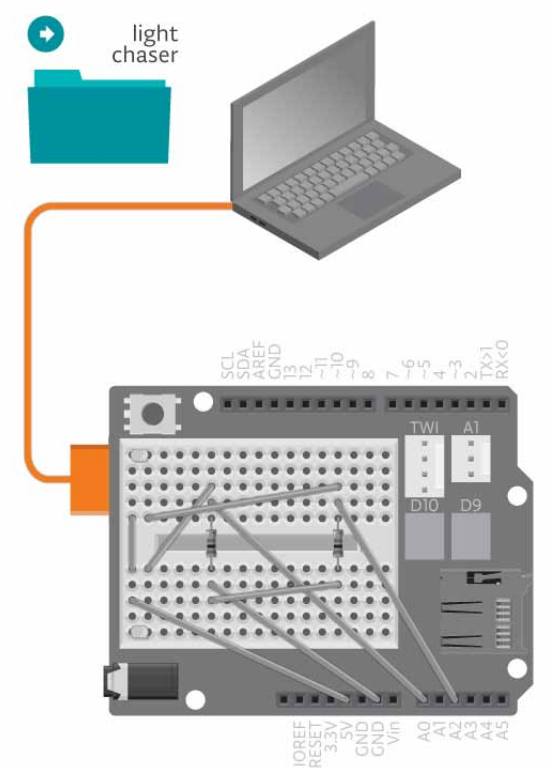
5.



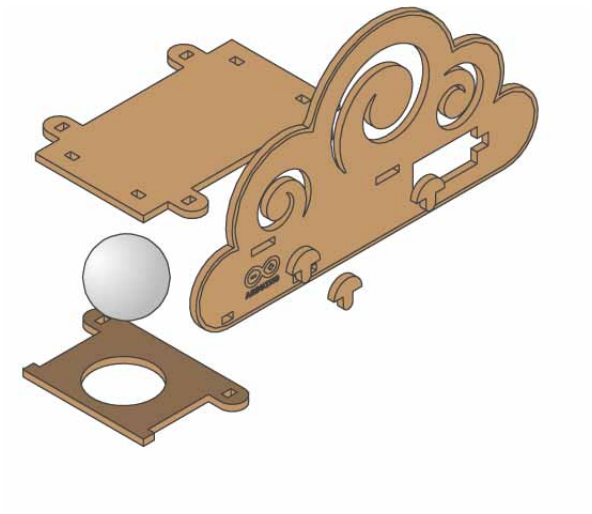
6. Conecta el condensador de 100 µF entre GND y 5V. La pata corta a GND y la larga a 5V.



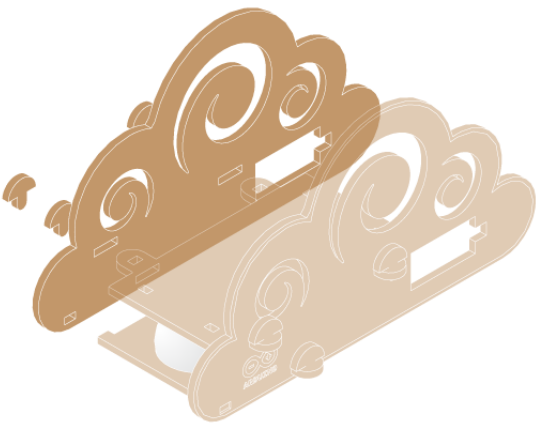
7. Conecta tu Arduino al ordenador y carga el ejemplo Archivo -> Ejemplos -> BasicEducationShield -> Robots -> Vehicle.



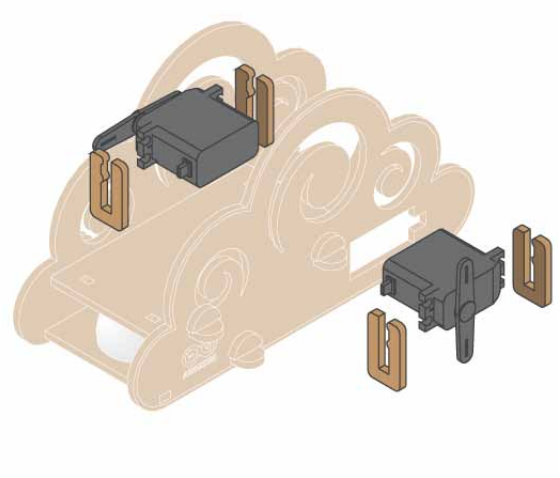
8. Construye el kit Vehículo.



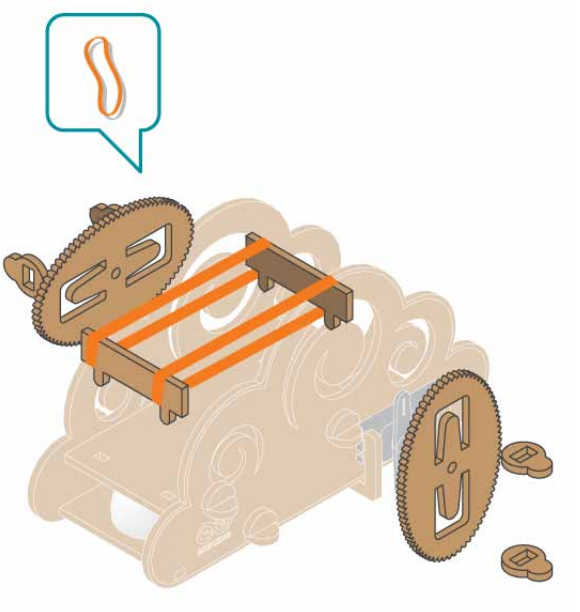
9.



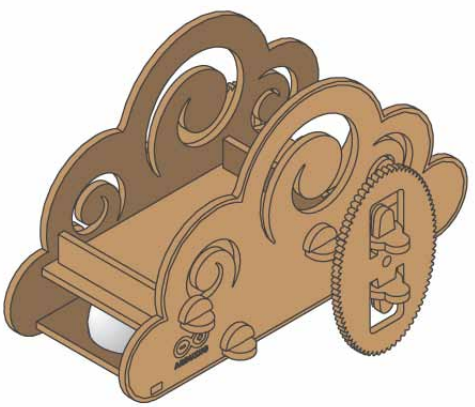
10.



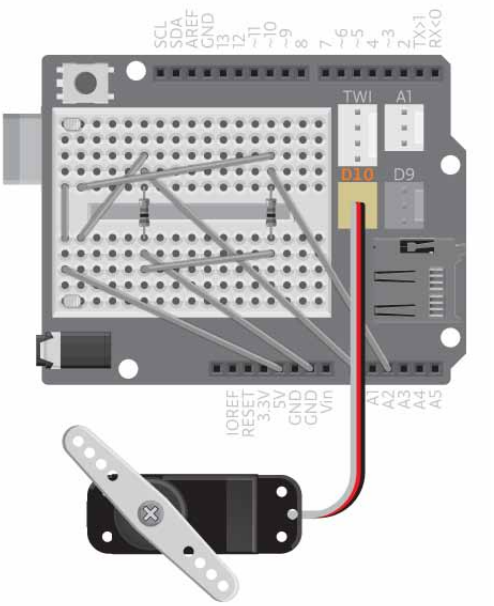
11.



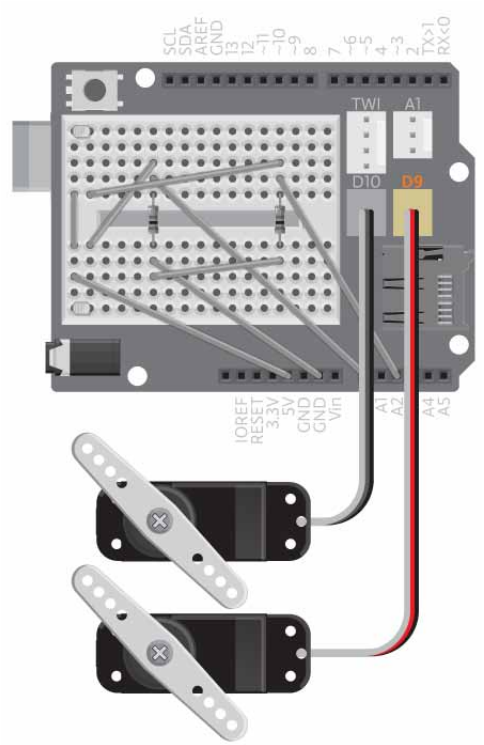
12.



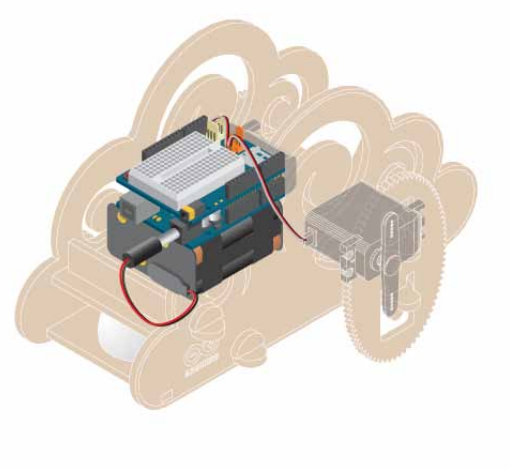
13. Conecta el servomotor izquierdo a D10.



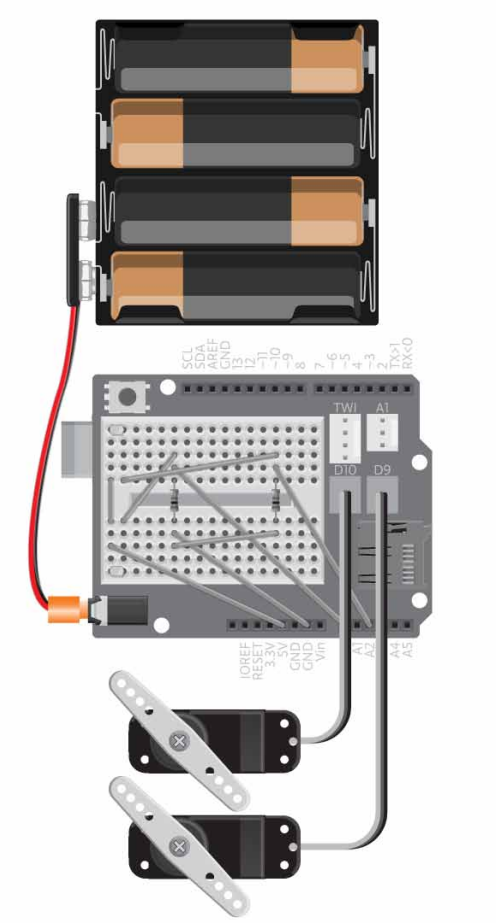
14. Conecta el servomotor derecho a D9.



15. Coloca la electrónica en el vehículo. Haz que los LDR apunten hacia afuera del vehículo.



16. Conecta el portapilas al jack de alimentación.



Código

Puedes encontrar el código en: Archivo -> Ejemplos -> BasicEducationShield -> Robots -> LightChaser

```
/*
  Light Chaser

  This little vehicle is on an impossible, never ending
  mission to catch the light. You can make it follw you
  by pointing a flashligt at it. (The one on your mobile
  phone eg.) It will always turn towards the light.

  (c) 2013 Arduino Verkstad
*/

#include <BasicEducationShield.h>

//Servo.h is necessary to be included here
#include <Servo.h>

//Declare the two wheels of robot, left wheel to D10 and
// right wheel to D9
Wheels wheels=Wheels(10, 9);

//Declare the two LDR sensors
LDR sensorLeft=LDR(A0);
LDR sensorRight=LDR(A2);

void setup(){
  //initialize the LDR sensors
  sensorLeft.config(600,800);
  sensorRight.config(600,800);

  //initialize the servo motors
  wheels.begin();
}
void loop(){
  if(sensorLeft.getState()){
    //Left ldr detects strong light, the vechile turns left
    wheels.turnLeft();
  }else if(sensorRight.getState()){
    //Right ldr detects strong light, the vechile turns right
    wheels.turnRight();
  }else{
    //No strong light detected, the vechile goes straight
    wheels.goForward();
  }
}
```

Cómo funciona

El programa comprueba si el sensor LDR izquierdo está estimulado por encima del umbral. Si es así, gira la rueda izquierda hacia atrás y la rueda derecha hacia delante para que el vehículo gire a la izquierda. Si no es así, comprueba si el sensor derecho LDR está por encima del umbral. Si es así, se gira el vehículo hacia la derecha. Si ninguno de los LDR está pasando del umbral, deja que el vehículo continúe recto.

¿No funciona?

1. ¿El vehículo no gira hacia la luz? Mira la referencia LDR sobre cómo corregir errores de LDR.
2. ¿Los motores no están funcionando? ¿el vehículo no va recto? Mira la referencia sobre cómo utilizar los servomotores de giro continuo.

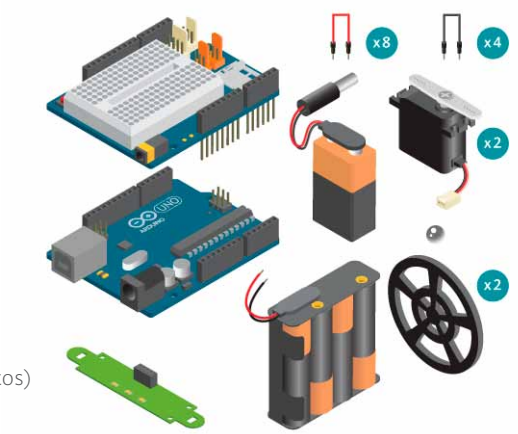
¡Sigue experimentando!

- ¿Puedes convertirlo en un "Vehículo Vampiro" que se esconda de la luz?
- Modifica el ejemplo para que el vehículo sólo se mueva cuando haya luz apuntándole.

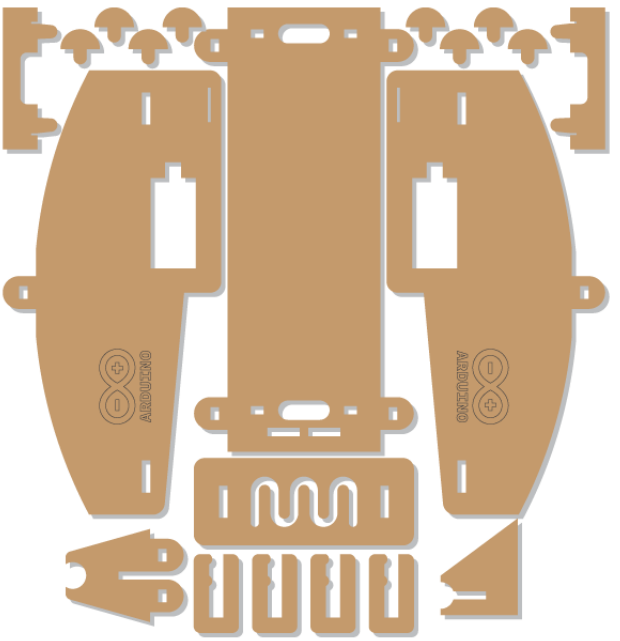
Sigue líneas

El Sigue Líneas hace exactamente lo que su nombre sugiere: sigue una línea. Hazle ir por donde quieras que vaya enseñándole el camino con una línea negra de 3 cm de ancho.

Materiales

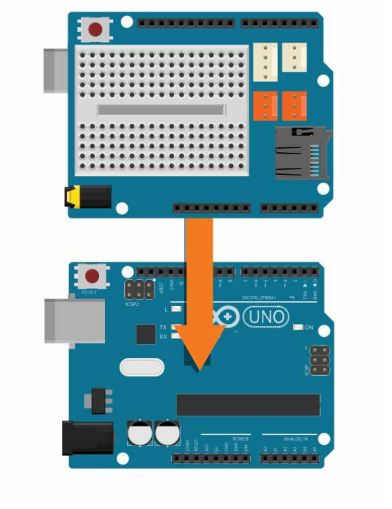


- 1 placa Arduino Uno
- 1 shield Básica Educativa
- 1 IR Array
- 2 servos de rotación continua
- 4 cables negros
- 8 cables de colores
- 2 conectores de pilas
  (uno con jack conector y otro con dos cables sueltos)
- 1 portapilas 4 AA
- 1 pila de 9V
- 4 pilas AA
- 1 kit Sigue líneas
- 1 bola metálica

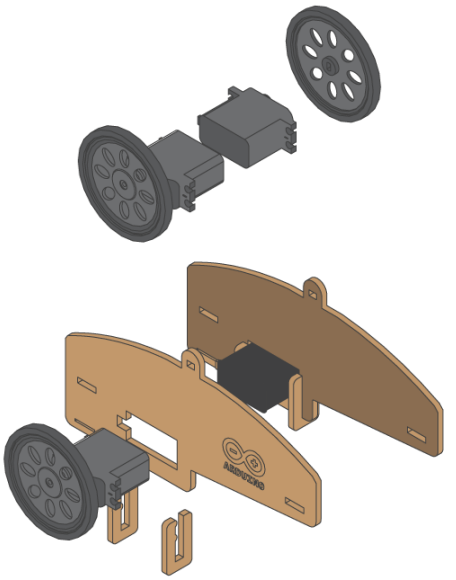


Instrucciones

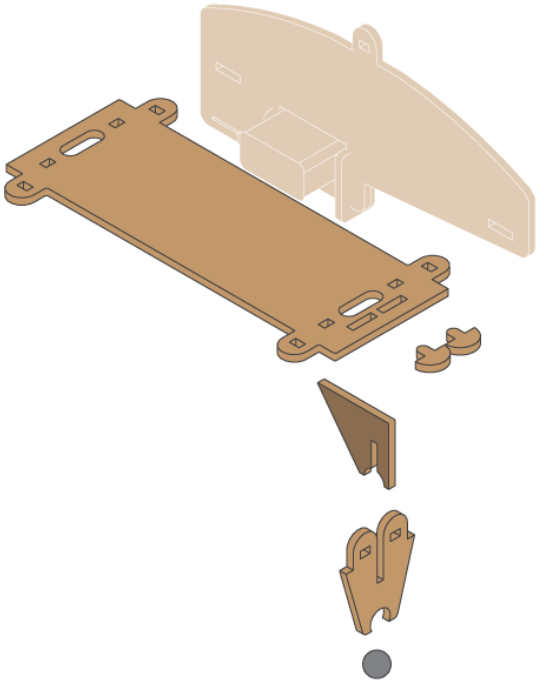
1. Conecta la shield a la parte superior de Arduino.



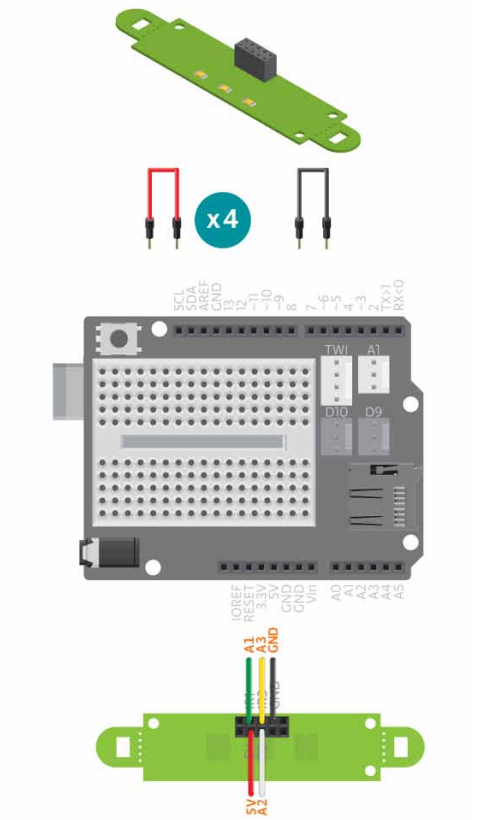
2. Monta un servo a cada lado del Sigue Líneas.



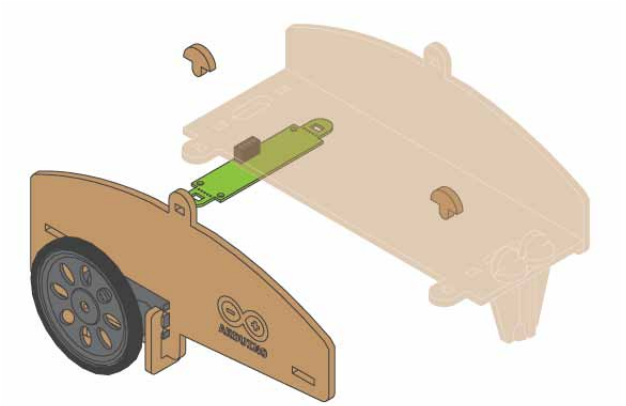
3. Monta las piezas de la bola en la parte media del Sigue Líneas.



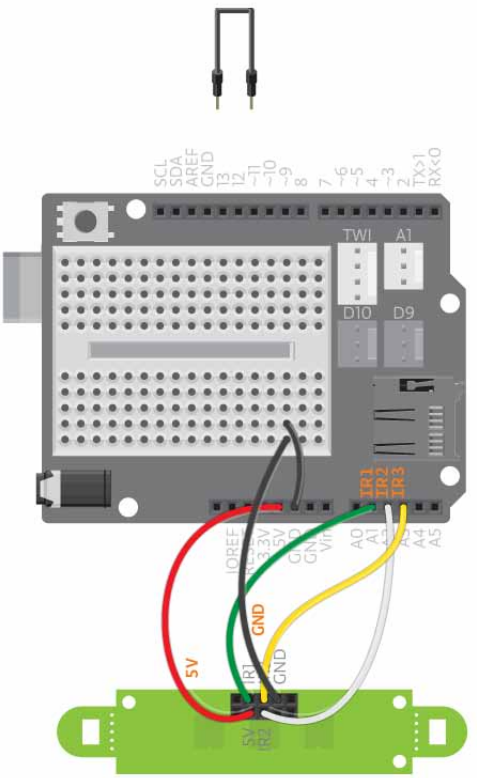
4. Conecta cinco cables a 5V, GND, IR1, IR2, y IR3 en el IRArray.



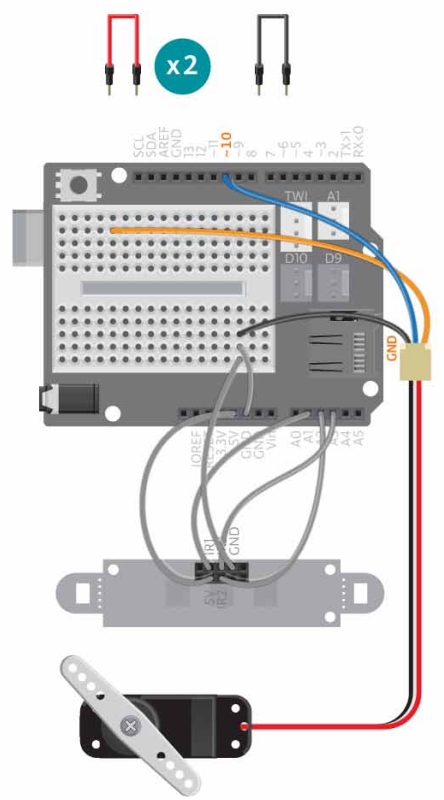
5. Pasa los cables por el agujero en frente del medio de la base y termina de montar el cuerpo principal.



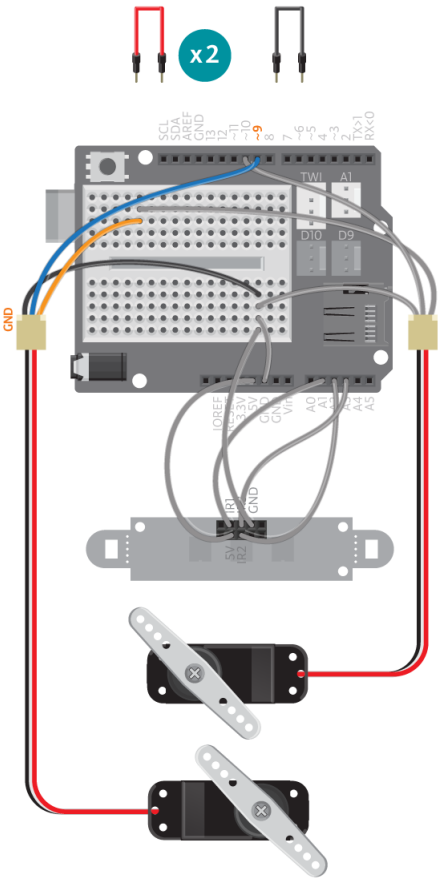
6. Conecta los cables sueltos del IRArray. 5V a 5V, GND a GND, IR1 a A1, IR2 a A2 y IR3 a A3.



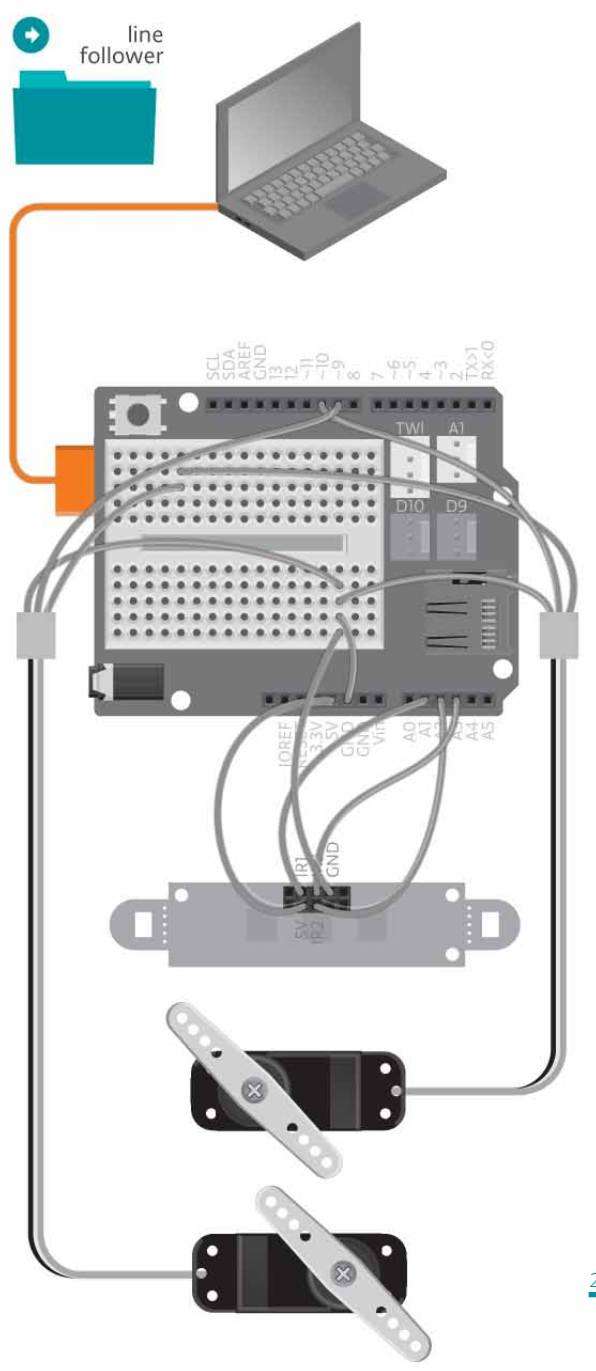
7. Utilizando tres cables, conecta el servo izquierdo. El cable negro a GND, el cable blanco al pin digital 10 y el cable rojo a la breadboard.



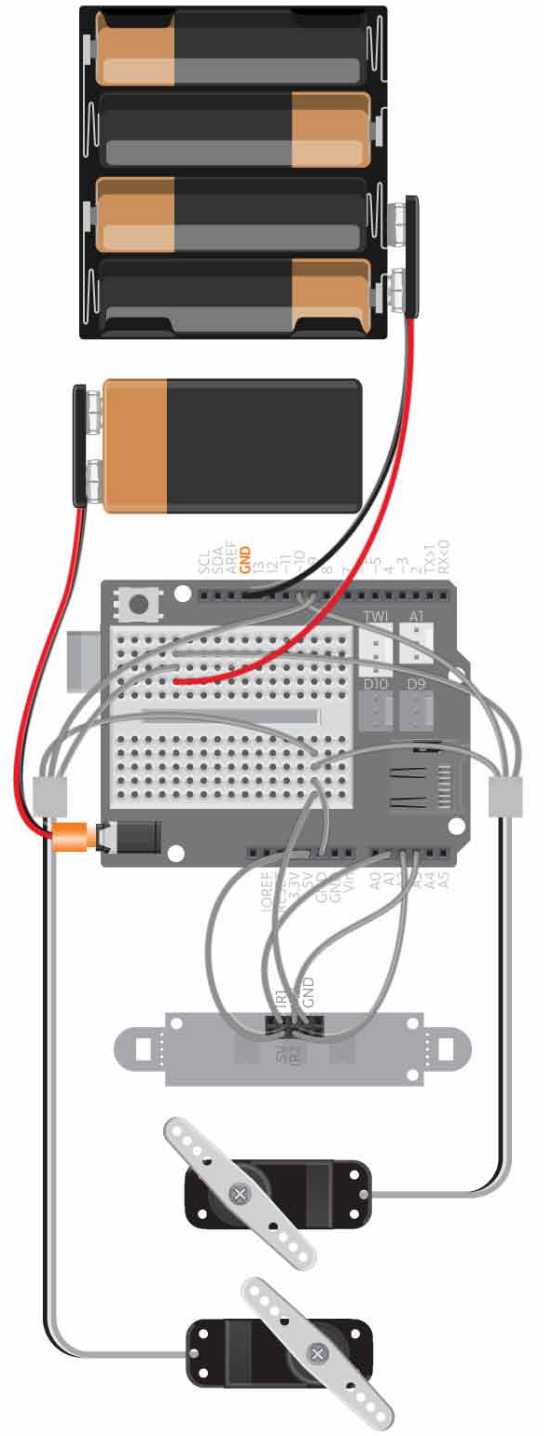
8. Utilizando tres cables, conecta el servo derecho. El cable negro a GND, el cable blanco al pin digital 9 y el cable rojo a la breadboard.



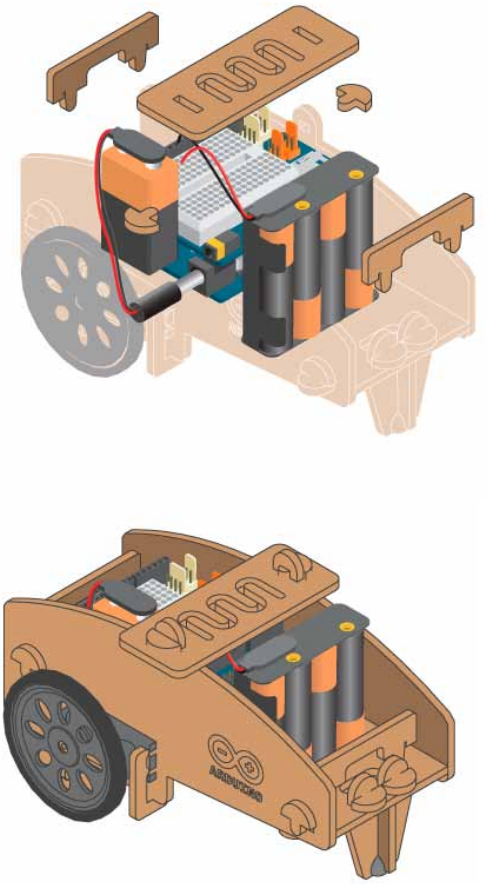
9. Conecta Arduino al ordenador y carga el ejemplo LineFollower.



10. Conecta el portapilas 4 AA. El cable negro a GND y el rojo a los dos cables rojos de los servos en la breadboard.



11. Coloca el Arduino y la shield en la parte superior del Sigue Líneas y conecta la pila de 9V a Arduino.



Código

Puedes encontrar el código en Archivos -> Ejemplos -> BasicEducationSheild-> Robots-> LineFollower

```
/*
  LineFollower
  The Line Follower does exactly what the name suggests,
  it follows a line. Make it go where ever you want by
  showing the way with a 3 cm wide, black line.

  (c) 2014 Arduino Verkstad
*/


#include <BasicEducationShield.h>
//Servo.h is necessary to be included here
#include <Servo.h>

//IRArray(IR1, IR2, IR3)
IRArray ir = IRArray(A1, A2, A3);
//(Wheels(left, right)
Wheels wheels=Wheels(10, 9);

void setup(){
  wheels.begin();
  delay(1000);
}

void loop(){
  int dir = ir.readLine();
  wheels.follow(dir);
}
```

¿Cómo funciona?

Los sensores IR en el IRArray pueden detectar blanco y negro. Utilizando el comando `readLine()` obtenemos un valor entre -100 y 100 representando la dirección en que la línea se dirige. Este valor es luego usado para decir a las ruedas en qué dirección ir escribiendo `wheels.follow(direccion)`.

¿No funciona?

1. Revisa las ilustraciones y comprueba tus conexiones. Asegúrate de que todos los cables están firmemente conectados.
2. ¿Los motores no funcionan? Mira la referencia para corregir los servos usandos como ruedas.
3. Corrige el IRArray, mira la referencia para corregir el IRArray.

¡Sigue experimentando!

- Imprime una pista y construye obstáculos para el robot. Si los obstáculos son demasiado difíciles, piensa si hay alguna forma para mejorar el funcionamiento del robot modificando las ruedas.
- Haz que el robot reaccione de una forma especial cuando "ve" sólo blanco. Es decir, cuando haya perdido la línea negra.

REFERENCIA

El siguiente contenido está diseñado para ayudarte a aprender de forma independiente cómo funcionan algunos componentes o conceptos desarrollados durante el curso. También puedes usar esta referencia cómo material auxiliar en caso de que alguno de tus proyectos o ejercicios no funcione.

Instala Arduino

1. Entra en la página de Arduino http://arduino.cc

2. Haz clic en el enlace de descarga en la barra de navegación superior.

3. Dependiendo de la versión de tu sistema operativo, haz clic en uno de los enlaces de la sección "Descargas", y comienza la descarga.

4. Cuando termine, descomprime el paquete en el lugar deseado del disco duro. ¡Ahora ya tienes Arduino instalado!

5. Ejecuta el programa una vez, esto creará una carpeta donde se almacenarán tus programas. Para saber dónde está esa carpeta, ve a Archivo -> Preferencias y mira el campo de texto Ubicación de Sketchbook. Esa dirección en tu ordenador es donde se almacenarán tus programas al dar a "Guardar" en el menú.

Instala la librería de la Shield Básica Educativa

1. Entra en https://github.com/arduinoverkstad/BasicEducationShield

2. Haz clic en el botón "ZIP" junto a "Clonar en Windows", y descarga el paquete.

3. Ve a la carpeta donde Arduino guarda tus programas. Si no existe, crea una carpeta llamada libraries, que es donde se almacenarán tus librerías. Dentro de esa crea una carpeta llamada BasicEducationShield.

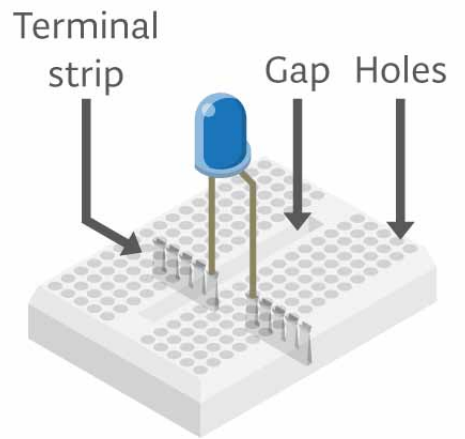
4. Descomprime el fichero y mueve todo el contenido de dentro de la carpeta "BasicEducationShield-master" a "BasicEducationShield".

5. Mueve la carpeta CapacitiveSensor a la carpeta libraries. ¡Ahora todo está hecho!

Breadboard

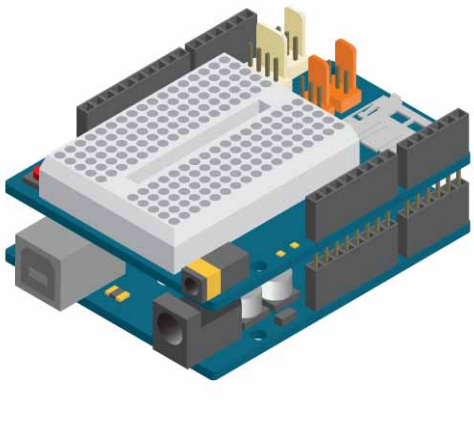
La breadboard o placa de entrenamiento es una pieza de plástico que va sujeta a la parte superior de la Shield Básica Educativa. En algunos ejemplos, vendrá separada. Se parece (un poco) al pan (bread en inglés) y se utiliza para poder hacer circuitos de manera sencilla.

Como puedes observar en la ilustración, cada línea de agujeros está conectada entre sí. Esta característica la puedes utilizar para conectar diferentes componentes, mediante la conexión a las diversas líneas de la placa de entrenamiento y a los pines que deberían estar conectados en la misma línea. También puedes utilizar cables a modo de puentes para conectar dos líneas entre sí.



Shield Básica Educativa

En el mundo Arduino, las shields son placas que pueden ser conectadas sobre Arduino, extendiendo así sus capacidades. La Shield Básica Educativa, está especialmente diseñada para este proyecto. Tiene un conjunto de características de fácil manejo, para que sólo tengas que conectar los componentes y comenzar a experimentar.



1. Pines digitales de entrada y salida. Están conectados directamente a los pines digitales de Arduino. Se utilizan estos pines para sensores o actuadores digitales.

2. Pines PWM. Están directamente conectados a los pines PWM de Arduino. Utiliza estos pines cuando quieras hacer salidas analógicas.

3. Pines de entrada analógica. Estos pines están conectados a los pines analógicos de Arduino. Utilízalos para sensores analógicos.

4. D9 y D10. Estos son los puertos digitales de TinkerKit. Están conectados a los pines digitales 9 y 10, por lo que si tienes algo conectado a ellos, no podrás conectar ninguna otra cosa a los pines digitales 9 y 10.

5. El conector de 4 pines se utiliza para programar Arduino. No te preocupes si no sabes muy bien cómo funciona.

6. El puerto para la tarjeta SD se utiliza para meter una tarjeta microSD. Empuja la tarjeta hasta que haga un "Click" y se encaje. Si quieres sacarla, empuja la tarjeta primero y hará un "Click" al soltarse, luego puedes tirar de ella.

7. Conector para altavoz. Puedes utilizarlo para conectar un altavoz. Está conectado al pin digital 11, por lo que no debes utilizarlo para otras cosas en el caso de que tengas conectado un altavoz.

No te olvides:

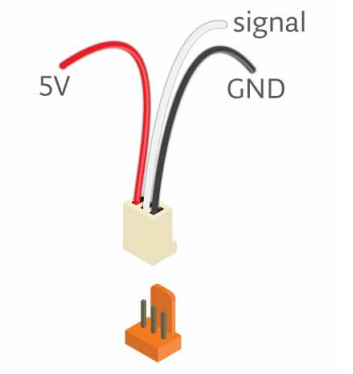
- Intenta evitar el uso de los Pins analógicos A4 y A5. Estos están conectados al lector de la tarjeta SD y por lo tanto pueden comportarse de forma extraña ocasionalmente.
- Los Pins digitales 9 a 13 no se pueden utilizar para sensores capacitivos.
- Los Pins digitales 9 y 10, así como el pin analógico A1 se conectan a los puertos TinkerKit, así que si estás utilizando los puertos TinkerKit, no utilices dichos pines.
- El pin digital 11 está conectado al conector de altavoz. Al utilizar un altavoz, no uses dicho pin digital.

Sobre los puertos de conexión TinkerKit

Los puertos TinkerKit se utilizan para todos los componentes TinkerKit, así como algunos componentes de 3 Pins (servos en nuestro caso). Los componentes TinkerKit se conectan sin dificultad, ya que los soportes de plástico limitan la forma en la que se pueden conectar.

Si vas a utilizar servos u otros componentes de 3 pines que no usen el conector del TinkerKit, ten en cuenta que la dirección del conector importa. Si estás sujetando el shield con la breadboard hacia arriba, los 3 pines del puerto TinkerKit irán al alimentador, a la señal y a GND respectivamente. El color del cable TinkerKit te puede ayudar a recordarlo: rojo significa alimentación, naranja significa señal y negro significa GND.

Técnicamente se pueden conectar componentes TinkerKit sin puertos TinkerKit, siempre y cuando conectes los cables a los pines correctos.



Botón

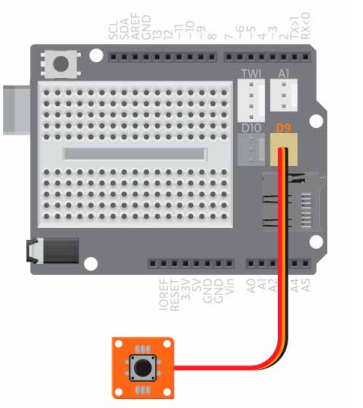
Un botón es un `INPUT` (entrada) digital. Esto quiere decir, que tiene dos estados, o bien `HIGH` (5V) o `LOW` (0V). Cuando conectas un botón a tu placa Arduino puede leer estos valores como 1 ó 0.

Materiales

- 1 placa Arduino Uno
- 1 shield Básica Educativa
- 1 botón TinkerKit
- 1 cable TinkerKit

Instrucciones

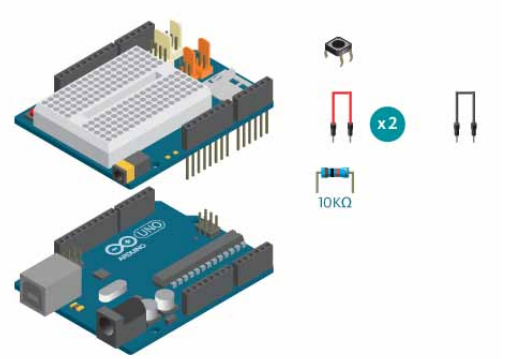
1. Conecta el botón TinkerKit al cable TinkerKit. Conecta el otro extremo del cable a D9 o D10.



o también puedes usar:

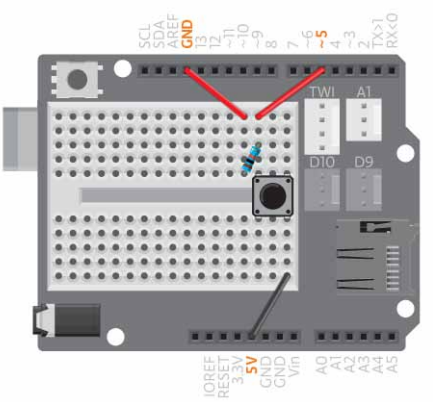
Materiales

- 1 botón
- 1 resistencia 10KOhm
- 3 cables



Instrucciones

1. Conecta el botón a través del puente de la breadboard. Conecta un Pin del botón a cualquier Pin digital. Aquí usamos el Pin digital. 4. Conecta la misma clavija a una resistencia de 10KOhm y la resistencia a GND. Conecta el otro extremo del botón a 5V.



Código

```
/*
Button
*/#include <BasicEducationShield.h>

//Declare the button component.
Button me=Button(9,HIGH);

void setup(){
Serial.begin(9600);
me.begin();
}
void loop(){
Serial.println("Please press...");
//Wait until the button is pressed.
Serial.println(me.pressed());

Serial.println("Please release...");
//Wait untill the button is released.
Serial.println(me.released());

Serial.println("Please double press...");

//Wait untill the button is double clicked. If only clicked
//once or action is too slow, it will return 0.
Serial.println(me.doublePressed());

}
```

Para que el botón funcione, tienes que asegurarte de que tu programa está leyendo desde el Pin digital donde conectaste el botón. Si está utilizando un botón TinkerKit, es o bien 9 o 10. Tienes que cambiar el número indicado en la línea de `Button me = Button(9, HIGH)` al número de Pin donde está conectado el botón. Abre el monitor serial y sigue las instrucciones mostradas para ver si funciona. Cada vez que ejecutes una instrucción, el monitor serial imprimirá "1'.

¿No funciona?

1. En primer lugar, asegúrate de que las conexiones entre botones, cable y placa son correctas.

2. Si estás usando un botón normal, generalmente viene con 4 Pins. Están separados en dos grupos, cada uno de los dos Pins que están enfrentados, están conectados entre sí. Así que asegúrate de que los pins conectados juntos estén en cada uno de los diferentes lados de la breadboard separados por el puente central.

3. Si estás utilizando un botón TinkerKit, hay un pequeño LED en la parte posterior del botón que se ilumina cuando el botón está conectado. Si estás seguro de que está conectado correctamente y el LED todavía no se enciende, el botón probablemente esté roto.

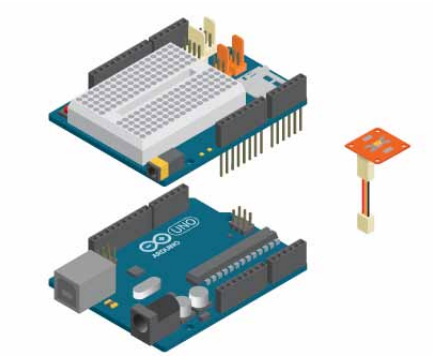
4. Asegúrate de que el programa esté leyendo desde el Pin digital derecho. Intenta cambiar el número en la línea de `Button me = Button(9, HIGH)`.

LDR

LDR es un sensor que lee la intensidad de la luz y ofrece una lectura analógica. Cuando se conecta a los Pins analógicos de Arduino, su valor oscila entre 0 y 1023 dependiendo de la cantidad de luz recibida por él.

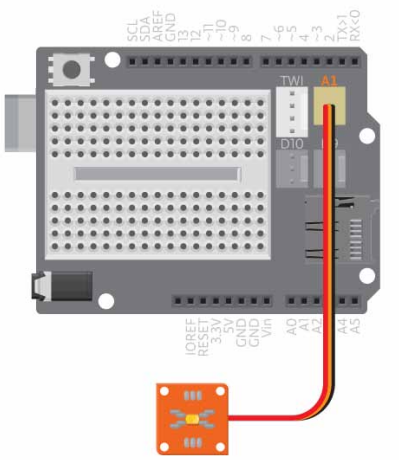
Materiales

- 1 LDR TinkerKit
- 1 cable TinkerKit



Instrucciones

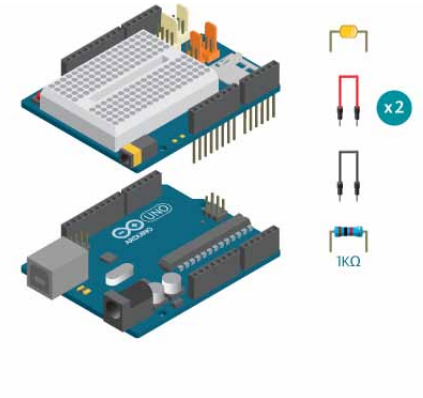
1. Conecta el LDR al cable TinkerKit LDR.
2. Conecta el otro extremo del cable al puerto analógico TinkerKit A1 en el shield.



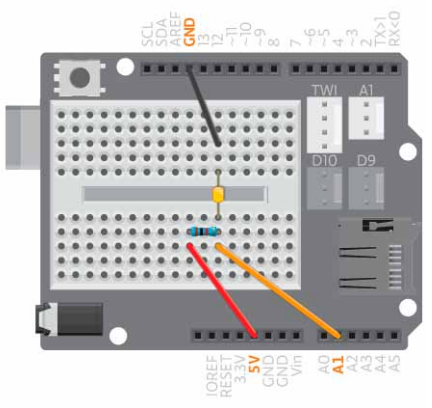
o también puedes...

Materiales

- 1 placa Arduino Uno
- 1 shield Básica Educativa
- 1 sensor LDR
- 1 resistencia 10KOhm
- 3 cables



Instrucciones



1. Conecta la resistencia de 10KOhm entre GND y un Pin analógico (A1 en este ejemplo) de la breadboard.

2. Conecta el LDR entre el Pin analógico y 5V.

Leer los valores

Para probar LDR, abre Archivo -> Ejemplos -> BasicEducationShield -> Help -> LDRTest.

```
1  /*
2  LDR test
3  */
4  #include <BasicEducationShield.h>
5
6  //Tinkerkit LDR is connected to analog 1.
7  LDR sensor = LDR(A1);
8
9  void setup(){
10 Serial.begin(9600);
11 }
12
13 void loop(){
14 //test() prints data to Serial port.
15 sensor.test();
16 delay(100);
17 }
```

Carga el programa y abre el monitor serial, deberías poder ver las lecturas del sensor LDR. Si cubres el LDR, el valor de lectura debería ser menor. Si proyectas luz sobre el LDR, la lectura debería ser mayor.

¿No funciona?

1. Primero, asegúrate de que las conexiones entre los cables, sensores y la placa son correctas.

2. Asegúrate de que el sensor está conectado al mismo Pin analógico que has indicado en tu código.

3. Si no se enciende el LED de la parte trasera del TinkerKit LDR, es porque o bien el componente está roto, o a Arduino no le llega alimentación.

Nota: Recuerda no utilizar A4 ni A5.

Interruptor

Debido a que la lectura del LDR está fuertemente influenciada por el entorno, deberías calibrarla antes de usarla. Si quieres usar el LDR como botón, lo podrás hacer con la función `GetState()`.

Para esto necesitas obtener dos lecturas del LDR: `baseValue` o la lectura cuando nada cubre el sensor; y el umbral. Para obtener el umbral, primero debes poner el objeto deseado / luz sobre el LDR, que te dará el `topValue` (que significa valor máximo en inglés). Y el umbral es uno que escojas entre `baseValue` y `topValue`.

Cuanto más cerca que esté el umbral de `baseValue`, más sensible será tu interruptor LDR pero también es más probable obtener más clicks falsos.

Abre Archivo -> Ejemplos -> BasicEducationShield -> Help -> LDR.

Código

```
1  /*
2  LDR
3  */
4
5  #include <BasicEducationShield.h>
6
7  LDR sensor = LDR(A1);
8
9  void setup(){
10 Serial.begin(9600);
11 sensor.config(700,900);
12 }
13
14 void loop(){
15 Serial.println(“Please press...”);
16 //Wait until the LDR gets cover-uncovered.
17 Serial.println(sensor.pressed());
18
19 Serial.println(“Please press...”);
20 //Wait until the LDR gets uncover-covered.
21 Serial.println(sensor.pressed());
22
23 While(true){
24 //Continuously output whether the sensor has
25 //passed threshold.
26 Serial.println(sensor.getState());
27 delay(30);
28 }
29
30 }
```

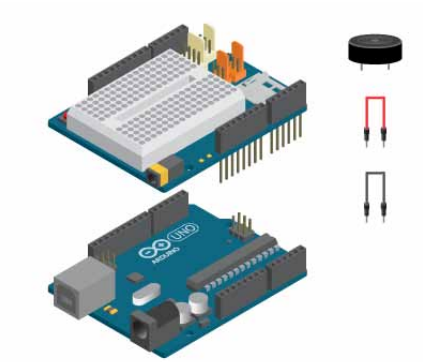
En este ejemplo el LDR se utiliza como un botón. Así que reemplaza los argumentos en `sensor.config (700.900)` mediante la variable `baseValue` y el umbral medido previamente. Carga el programa y abre el monitor Serial. Si proyectas luz sobre el sensor, deberías ver que el programa lo entiende como "presionado".

Melodía

Puedes utilizar el piezoeléctrico para hacer un sonido básico con Arduino. Aquí utilizamos la librería Melody para reproducir una pieza musical, de una lista de notas y duraciones.

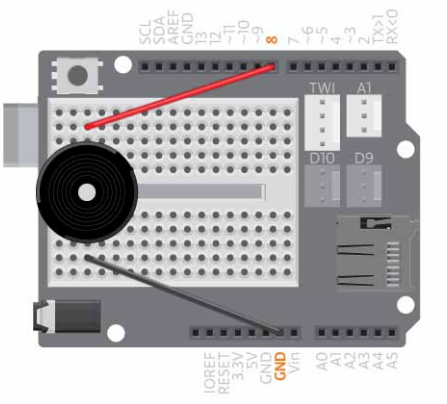
Materiales

- 1 placa Arduino Uno
- 1 shield Básica Educativa
- 1 piezo
- 2 cables (rojo y negro)



Instrucciones

1. Conecta el cable rojo de tu piezo al Pin digital 8 y el cable n



Código

```
1  /*
2  Melody
3  */
4  #include <BasicEducationShield.h>
5
6  #include "pitches.h"
7
8  //8 is the pin the piezo is connected to.
9  Melody me=Melody(8);
10
11 void setup(){
12 }
13
14 void loop(){
```

```
15 //Defining the notes used in the music. No more than 30.
16 int notes[] = {
17 NOTE_C4, NOTE_G3,NOTE_G3, NOTE_A3, NOTE_G3,0, NOTE_B3, NOTE_C4};
18
19 //Duration of each note. should be corresponding to the notes above.
20 int noteDurations[] = {
21 4, 8, 8, 4,4,4,4,4 };
22
23 //Play the notes defined above
24 me.play(8,notes,noteDurations,1.4);
25
26 delay(3000);
27
28 //Make a beep sound
29 me.beep();
30
31 delay(3000);
32 }
```

Carga el código. Ahora el piezo debería tocar una pequeña melodía, quedarse en silencio durante 3 segundos, y luego emitir un pitido corto.

Para cambiar la melodía, tienes que cambiar las notas y la duración de las notas. La línea que define las notas utilizadas en el código de Melody es: `int notes [] = {NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4}`

Para cambiar una nota, reemplázala con una nota establecida en el fichero pitches.h que acompaña al programa (verás que el programa tiene dos tabs, uno es el programa en sí mismo, el otro es el fichero mencionado).

Puedes añadir hasta 30 notas por melodía. No te olvides de colocar una coma entre cada nota. Si deseas que el piezo se quede en silencio en una notas, solo tienes que escribir "0".

Al mismo tiempo, cada nota tiene una duración. Estas duraciones se definen en la línea: `int noteDurations [] = {4, 8, 8, 4,4,4,4,4}`. 4 es la duración de una nota negra, 8 una corchea, 16 semi-corchea, 2 una blanca y 1 una redonda.

Cada número corresponde a una nota. NOTE_C4 es una negra (4 unidades de tiempo de duración), NOTE_G3 es una corchea, etc. Tiene que haber tantas duraciones como notas. Es decir, los dos arrays tienen que ser de la misma longitud.

La melodía se toca con esta línea de código: `me.play (8, notes, noteDurations, 1,4)`, `8`: Es el número de notas. Si agregas notas a la melodía, este número debe ser cambiado para que coincida con el número de notas, `notes` son las notas de la melodía, `noteDuration` son las duraciones de las notas, `1,4` es la velocidad. Prueba a cambiar el último número para ver lo que ocurre con la melodía.

Si quieres escribir tu propio programa y utilizar las notas, asegúrete de incluir un tab con pitches.h en tu programa (equivale a copiar el fichero pitches.h dentro de la carpeta donde esté tu programa), y de poner `# include "pitches.h"` al comienzo de tu programa.

¿No funciona?

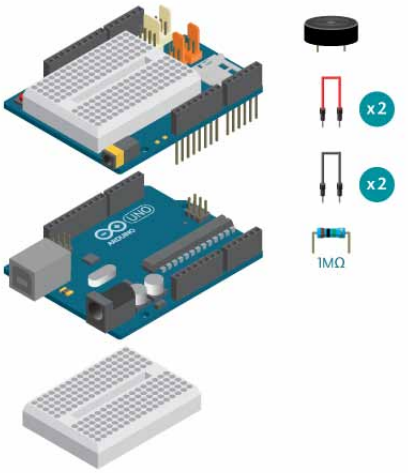
1. Asegúrate de que está bien conectado al pin digital y a tierra (GND).
2. Asegúrate de que hayas conectado el piezo al mismo pin que dice en el código.

Sensor de knock

En este ejemplo explicamos cómo utilizar un piezo como sensor de vibración. Puedes darle un toque directamente; o pegarlo con cinta adhesiva a una superficie y darle un toque desde la superficie.

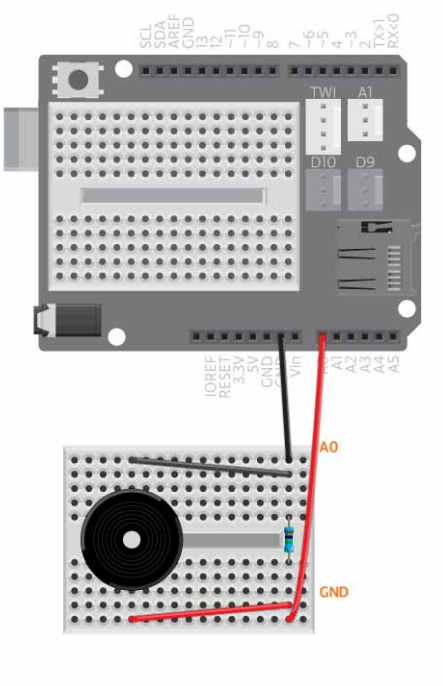
Materiales

- 1 placa Arduino Uno
- 1 shield Básica Educativa
- 1 piezo
- 1 resistencia 1Mohm
- 4 cables



Instrucciones

1. Conecta la resistencia a A0 y a GND.
2. Conecta el piezo a A0 y a GND a través de la resistencia.
3. Abre Archivo -> Ejemplos -> BasicEducationShield -> Help -> PiezoKnockSensor



Código

```
/*
Piezo Knock Sensor
*/#include <BasicEducationShield.h>

PiezoKnockSensor sensor=PiezoKnockSensor(A0);

void setup(){
Serial.begin(9600);
sensor.config(40,80);

}
void loop(){
Serial.println(“Please knock...”);
//Wait until the button is knocked.
Serial.println(sensor.knocked());
}
```

Carga el código en la placa. Abre la monitor serial y dale un toque al piezo para ver si funciona. Debería imprimir un "1" en la ventana del monitor serial al detectar la vibración.

Para cambiar la sensibilidad del sensor capacitivo, cambia la línea en la configuración `sensor.config (40,80)`. El umbral de detección del toque es 40 y el tiempo de rebote es 80. El umbral define la intensidad con la que debes llamar, el tiempo de rebote previene que el sensor retorne varios golpes de un solo toque pero que también limita lo rápido que puedes tocar.

La mayoría de las veces, es más fácil para mejorar la construcción que encontrar un buen valor de umbral / rebote. Si pegas el piezo a una superficie, es mejor darle un toque lo más cerca posible del sensor. Quitar la carcasa de plástico del piezo también ayuda a aumentar la sensibilidad.

¿No funciona?

1. Asegúrate de que todo esté conectado correctamente.
2. Asegúrate de que has conectado el piezo al mismo Pin que has indicado indicado en tu código.
3. Si utilizas piezos tanto para la melodía como para el sensor de vibración, asegúrate de que no intentas leer el que usas como altavoz.

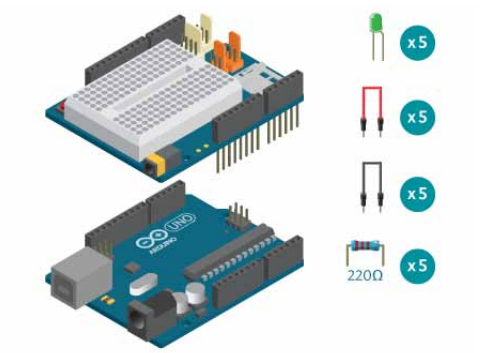
Nota: Recuerda que no debes utilizar A4 ni A5

VU-Meter de LED

Un VU–Metro es una línea que agrupa varios LED. Este ejemplo te explicará cómo conectarlos entre sí y cómo controlarlos desde la librería BasicEducationShield. Puedes utilizar cuantos LED quieras, siempre y cuando Arduino tenga suficientes Pins. En este ejemplo utilizaremos 5 LED.

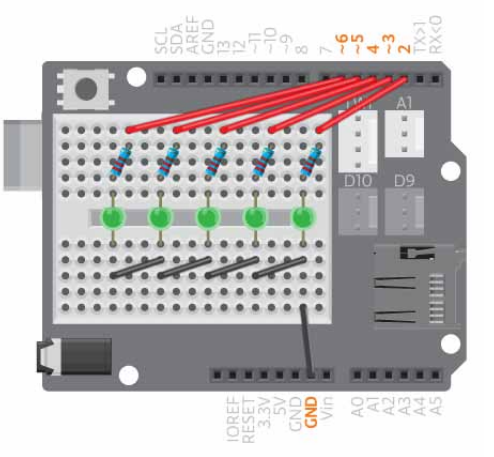
Materiales

- 1 placa Arduino Uno
- 1 shield Básica Educativa
- 5 LEDs
- 5 resistencia 220 ohm
- 10 cables



Instrucciones

1. Conecta cinco LEDs a través del puente central de la breadboard.
2. Conecta una resistencia de 220 ohmios al Pin digital 2.
3. Conecta la resistencia a la pata larga del primer LED.
4. Conecta cada uno de los Pins digitales 3 hasta 6 a un LED correspondiente siguiendo el mismo método.
5. Conecta la pata corta de los LEDs a un Pin de Arduino GND utilizando cables negros.



La pata corta del LED siempre va a tierra (GND) y la pata larga a un Pin digital a través de una resistencia de 220 ohmios. Cuando conectamos varios LEDs, puesto que todas las patas cortas deben estar conectadas a tierra, todos ellos están conectados entre sí. Para poder controlar cada LED tenemos que conectar las patas largas por separado a un Pin digital.

Dado que hay muchos componentes conectados a la breadboard, revísalos con cuidado para que las dos patas de un LED o de la resistencia no estén en la misma línea (ya que provocarían un corto circuito y no funcionaría).

Abre Archivo -> Ejemplos -> BasicEducationShield -> Help -> vuMeter

Código

```
1  /*
2  VU-Meter
3  */
4
5  #include <BasicEducationShield.h>
6
7  //Declare the VUMeter
8  VUMeter me;
9  int pins[]={2,3,4,5,6};
10 int pinCount=5;
11
12 void setup(){
13 //Configure the VU meter using parameters defined previously.
14 me.config(pinCount,pins);
15
16 //initialize the component. Must be called.
17 me.begin();
18
19 }
20 void loop(){
21 //Fill 5 LEDs
22 me.fill(5);
23 delay(3000);
24
25 //turn all the LEDs off
26 me.clear();
27 delay(1000);
28
29 //Turn the 3rd LED on
30 me.on(2);
31 delay(1000);
32
33 //Turn the 3rd LED off
34 me.off(2);
35 delay(1000);
36
37 //One LED light up at a time, scroll from left to right
38 me.scrollRight(700);
```

```
39
40 //And then scroll back from the 2nd on the right
41 me.scrollLeft(700,1);
42
43 //the 3rd led will be blinking for 10 times, each time
44 //with 100 milliseconds on and 100 milliseconds off
45 me.blink(2,100,10);
46
47 //All LEDs will be blinking 10 times
48 me.blinkAll(100,10);
49
50 //The 2nd to 4th LED will be light up
51 me.fillFrom(1,3);
52 delay(2000);
53 }
```

Si utilizas un número mayor o menor de LEDs que en el ejemplo, recuerda cambiar `int pinCount = 5` e indica el número correspondiente a los LEDs conectados.

Cuando has cargado el código, lo primero que ocurre es que todos los LEDs se encienden durante 3 segundos. El programa sigue la siguientes funciones:

`clear()`: Apaga todos los LEDs.

`on(LEDindex)`: Enciende un LED.

`off(LEDindex)`: Apaga un LED.

`scrollRight(speed, startIndex)`: Los LEDs se van encendiendo de uno en uno de derecha a izquierda.

`scrollLeft(speed, startIndex)`: Los LEDs se van encendiendo de uno en uno de izquierda a derecha.

`blink(LEDindex,speed, times)`: Un LED parpadea.

`blinkAll(speed, times)`: Todos los LEDs parpadean.

`fillFrom(startIndex, stopIndex)`: Pasa los LEDs de `startIndex` a `stopIndex`

`fill(numberOfLEDs)`: Pasa los LEDs desde el primero a `numberOfLEDs` (número de LEDs).

¿No funciona?

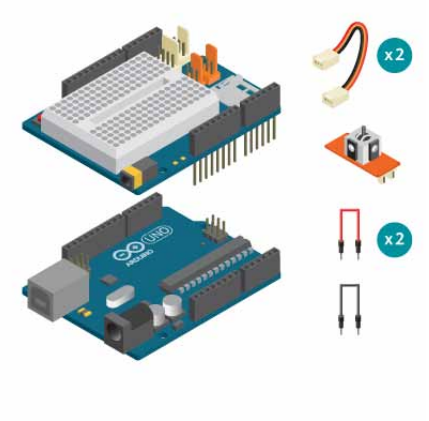
1. Si no se iluminan todos los LEDs, asegúrate de que los cables, los LEDs y las resistencias estén conectados correctamente a la placa de entrenamiento. Mira la referencia de la breadboard si no estás seguro de cómo funciona. Si conectas los componentes que supuestamente no deben estar conectados a la misma fila en la breadboard, hará corto circuito.
2. Asegúrate de que has conectado los LEDs a los mismos Pins que has indicado en el código.
3. Asegúrate de que las patas cortas de los LEDs están conectados a GND y no al revés.

Joystick

Es un solo componente pero utiliza dos sensores analógicos y te da dos valores. Un valor X y un valor Y. El valor para cada dirección es 0 si está en la mitad, -1 si está a una lado y 1 si está al otro lado.

Materiales

- 1 placa Arduino Uno
- 1 shield Básica Educativa
- 1 joystick Tinkerkit
- 2 cables Tinkerkit
- 3 cables

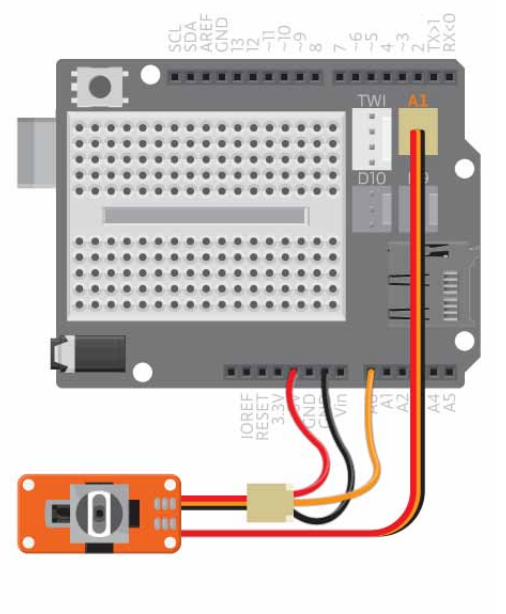


Instrucciones

1. Conecta uno de los cables al joystick TinkerKit donde dice Y.
2. Conecta un cable puente rojo al cable rojo en el cable TinkerKit y a 5V.
3. Conecta un cable puente negro al cable negro en el cable TinkerKit y a GND.
4. Conecta el último cable puente al cable TinkerKit y al Pin analógico A0.
5. Conecta el segundo cable TinkerKit al joystick y puerto analógico TinkerKit A1 en la shield.

Abre Archivo -> Ejemplos -> BasicEducationShield -> Help -> Joystick

```
/*  Joystick
*/

#include <BasicEducationShield.h>

//The joystick uses two analog pins. One for
X and one for Y.
Joystick me=Joystick(A1,A0);

void setup(){
Serial.begin(9600);
}
void loop(){
Serial.print("X: ");
Serial.print(me.getX()); //Get the X value
and print it.
Serial.print("  Y: ");
Serial.println(me.getY()); //Get the Y value
and print it.
delay(200);
}
```



Abre el monitor serial para ver si el joystick está funcionando en ambas direcciones. Imprimirá "0"para cada valor si está en el medio, -1 en un lado y 1 en el otro.

¿No funciona?

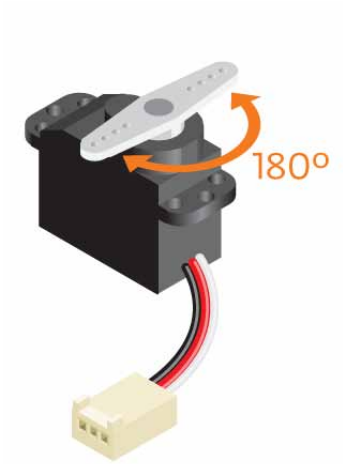
1. Primero, asegúrate de que las conexiones entre el joystick, cables y placa son correctos.

2. Asegúrate de que has conectado el joystick a los mismos Pins que has indicado en tu código.

3. Hay tres pequeños LEDs en la parte posterior del joystick que se encienden cuando el botón está conectado (estas luces se encienden aunque sólo hayas conectado uno de los conectores). Uno muestra que le está llegando alimentación, otro muestra X y el tercero, muestra Y. Si estás seguro de que está conectado correctamente y el LED todavía no se enciende, probablemente el joystick esté roto.

Nota: Si estás utilizando la shield Básica Educativa, recuerda que no debes utilizar el Pin analógico A4 o A5.

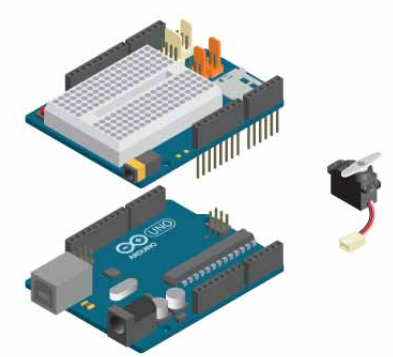
Servo Estándar



El servo estándar es un tipo de motor que tiene un ángulo limitado de rotación (generalmente 180º) y que puedes controlar exactamente en qué ángulo se para.

El que te ha sido proporcionado para este proyecto es más parecido a un microservo. Actúan igual que un servo normal pero son más pequeños y menos potentes al girar.

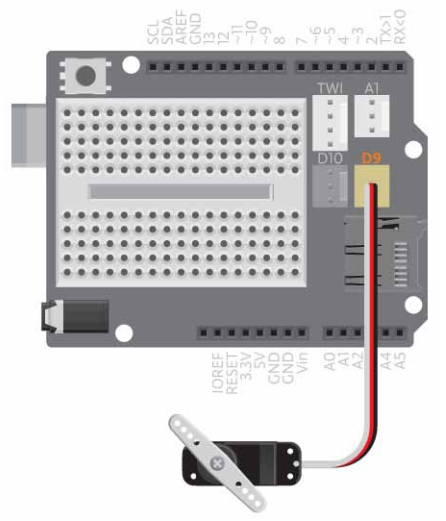
Materiales



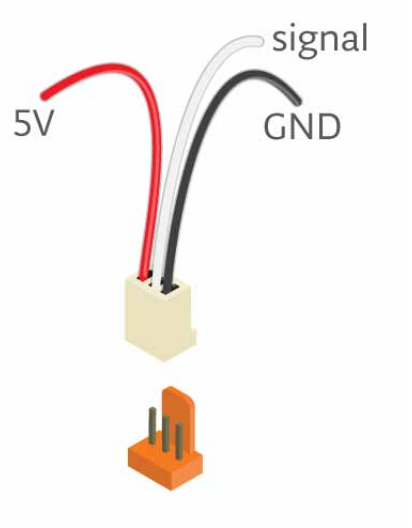
- 1 placa Arduino Uno
- 1 shield Básica Educativa
- 1 servo estándar

Instrucciones

1. Conecta el servo al puerto digital 9 del TinkerKit.



2. Asegúrate de que la dirección de la conexión está bien: si estás sosteniendo el shield con la breadboard hacia ti, el cable negro del conector debe estar en el lado derecho.



Abre desde Arduino Archivo -> Ejemplos -> BasicEducationShield -> Help -> ServoEstándar.

```
1  /*
2  StandardServo
3  */
4  #include <BasicEducationShield.h>
5  #include <Servo.h>
6
7  Servo me;
8
9  void setup(){
10 //Servo is initialized,
11 me.attach(9);
12 }
13
14 void loop(){
15 //Make the servo rotate to 76 degrees.
16 me.write(76);
17
18 }
```

Ejecuta el programa, y tendrás el servo apuntando a 76 grados. Cambia los grados en el código, cárgalo de nuevo y verás cómo se mueve a dicha posición.

Nota: Debido al diseño mecánico de tus proyectos, algunas veces el ángulo de giro permitido para el servo es mucho menor a 180 grados. Si ves que el brazo del servo empuja o tira con fuerza hacia otras partes de tu proyecto, intenta bajar el brazo del servo y ponlo de nuevo en un ángulo más apropiado. Puede requerir varios intentos hasta que consigas ponerlo bien.

¿No funciona?

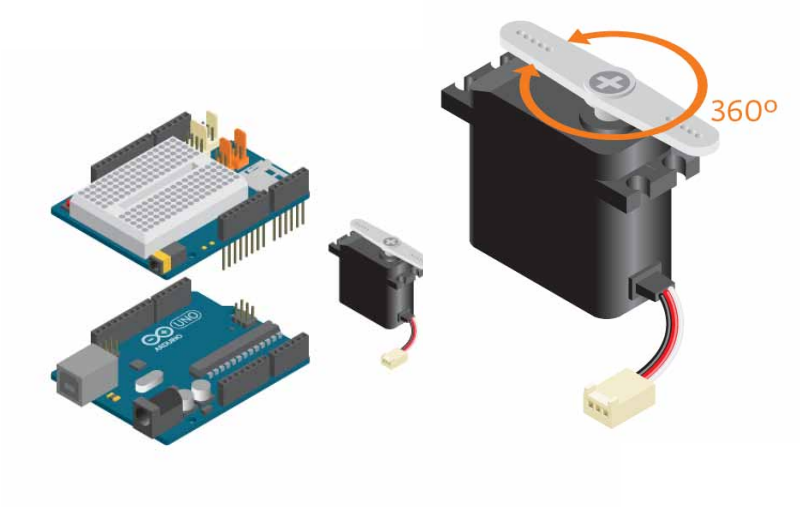
1. Asegúrate que has conectado el servo al mismo pin que el utilizado en el código.

Servo de giro continuo

El servo continuo tiene el mismo aspecto que un servo estándar, pero es capaz de girar continuamente como un motor normal. No puedes controlar qué ángulo está señalando pero puedes especificar la rapidez con la que quieres que gire y en qué dirección debe girar.

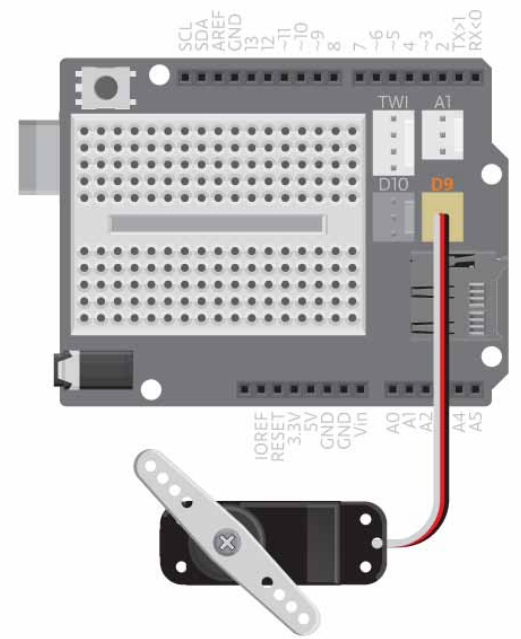
Materiales

- 1 placa Arduino Uno
- 1 shield Básica Educativa
- 1 servo continuo



Instrucciones

1. Conecta el servo al puerto D9.



Abre Archivo -> Ejemplos -> BasicEducationShield -> Help -> ServoContinuo en el Arduino IDE.

```
1  /*  ContinuousServo
2  */
3
4  #include <BasicEducationShield.h>
5  #include <Servo.h>
6  Servo me;
7
8  void setup(){
9  me.attach(9);
10 }
11
12 void loop(){
13 //Make the servo rotate in speed 120.
14 me.write(120);
15
16 }
```

Este programa hará que el servo gire a una velocidad de 120. Puedes utilizar una velocidad entre 0 y 180. 180 es la velocidad más rápida en una dirección y 0 la más rápida también pero en la dirección opuesta. 90 debería dejarlo parado. Este valor no es exacto y puede variar de un servo a otro, por lo que deberás calibrarlo.

¿No funciona?

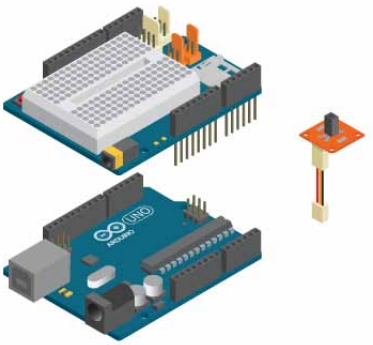
1. Asegúrate que has conectado el servo al mismo pin que el utilizado en el código.

Sensor Tilt

El interruptor tilt es un componente que detecta si está vertical o inclinado.

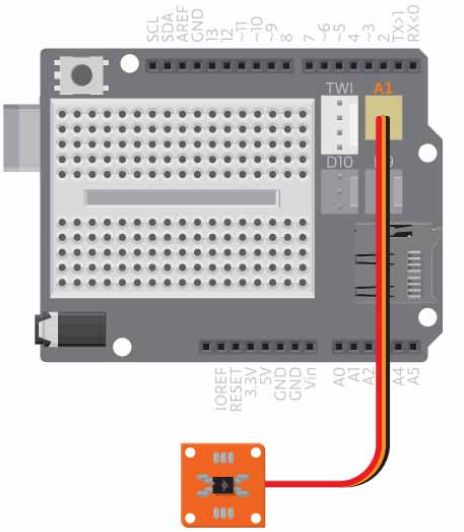
Materiales

- 1 placa Arduino Uno
- 1 shield Básica Educativa
- 1 interruptor Tilt TinkerKit
- 1 cable TinkerKit



Instrucciones

1. Conecta el interruptor tilt al cable TinkerKit.
2. Conecta el otro extremo del cable al puerto D10 en el shield.



Abre en el IDE de Arduino el programa Archivo -> Ejemplos -> BasicEducationShield -> Help -> TiltSwitch

Code

```
1  /*
2  TiltSwitch
3  */#include <BasicEducationShield.h>TiltSwitch me=TiltSwitch(10);void setup(){
4  Serial.begin(9600);//Initialize the component. Must be called.
5  me.begin();
6  }
7  void loop(){
8  Serial.println("Please tilt...");//Wait until the tilt switch is tilted.
9  Serial.println(me.pressed());
10 }
```

Carga el código, abre el monitor serial, y coloca el sensor de manera vertical. Inclínalo cuando veas "Por favor, inclínalo..." y deberías ver que aparece "1"

Nota: El interruptor tilt solo tiene 2 estados, vertical u horizontal. No detecta si está al revés, ni mide el ángulo de inclinación. Así que asegúrate de que esté adaptado a tu proyecto.

¿No funciona?

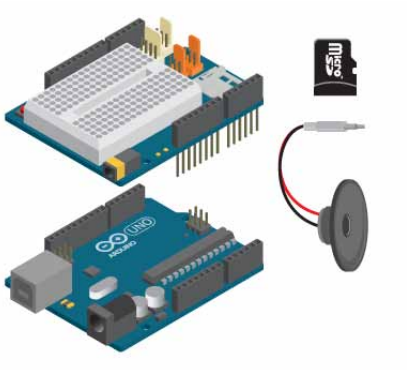
1. Asegúrate de haber conectado el sensor al mismo pin que el utilizado en el código.

Player

Player utiliza la librería SD de Arduino para leer ficheros wav de la tarjeta micro SD y reproducirlos.

Materiales

- 1 placa Arduino Uno
- 1 Basic Education Shield
- 1 tarjeta microSD
- 1 altavoz de papel (no piezo eléctrico)



Instrucciones

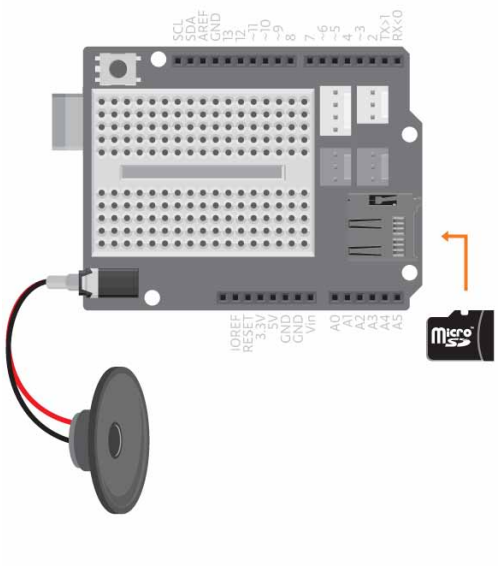
1. Graba un archivo de sonido y guárdalo como "0.wav". Mira la referencia Preparar sonido wav o utiliza la muestra pregrabada en la carpeta del programa.

2. Guárdalo en el directorio raíz de la tarjeta SD.

1. Inserta la tarjeta SD en la ranura SD del shield.

2. Conecta el altavoz a la salida de audio del shield.

3. Abre en el IDE de Arduino Archivo -> Ejemplos -> BasicEducationShield-> Help -> Player.



Código

```
/*
Player
*/

#include <BasicEducationShield.h>
#include <SD.h>

Player player=Player();

void setup(){
//Initialize the sound player
player.begin();
}

void loop(){

//Play the file named “0.wav” on SD card
player.play(“0.wav”);
delay(1000);
}
```

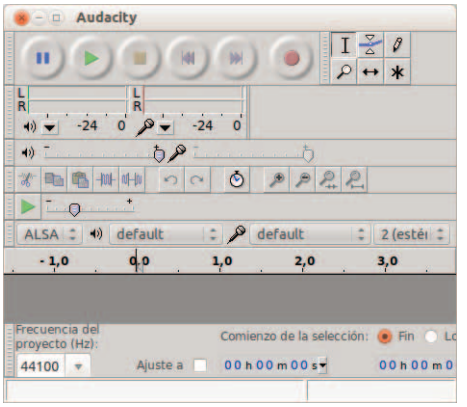
¿No funciona?

1. Asegúrate de que el altavoz esté conectado del todo.

2. Abre el monitor serial. Si el archivo no se puede abrir, formatea la tarjeta SD con formato FAT y prueba de nuevo. Si estás utilizando tu propia grabación, asegúrate que esté en el formato adecuado.

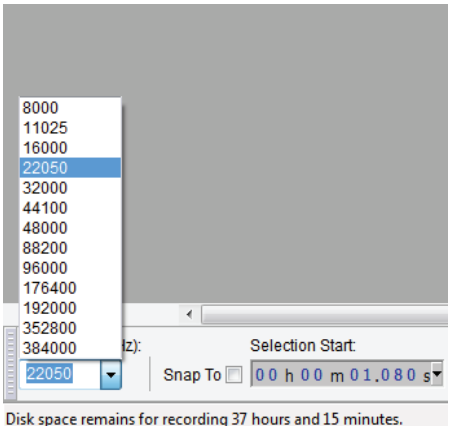
Prepara Sonidos Wav

Audacity, herramienta open source para audio



El Basic Education Shield lleva una tarjeta SD que puede ser utilizada para la reproducción de sonido. El sonido necesita tener un formato específico. Debe tratarse de un fichero mono, wav sin comprimir, de 8 bits sin signo. Los dos tutoriales que siguen explican cómo crear o transformar sonidos al formato apropiado.

Puedes utilizar Audacity para grabar un sonido que quieras usar con Arduino.

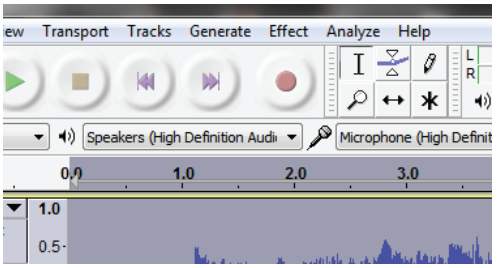


1. Descarga la última versión de Audacity.

2. Crea una nueva pista de sonido con el menú Pistas -> Añadir nueva -> Pista de audio. Nota que deberá ser una única pista de audio mono.

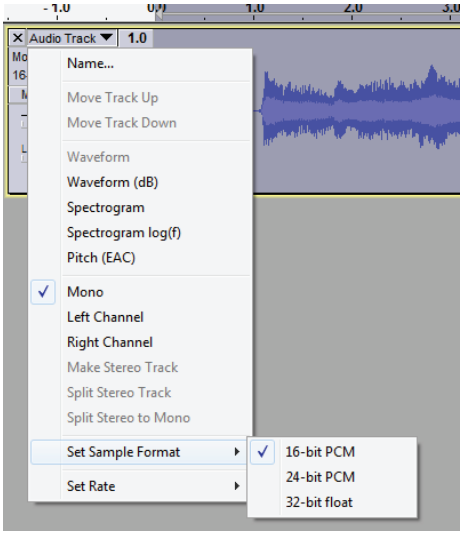
3. En el botón izquierdo de la interface hay un menú desplegable "Project Rate", elige 22050.

4. Haz clic en el botón rojo de grabación para iniciar la grabación de sonido.



5. Cuando termine, haz clic en el botón amarillo para pararlo.

6. Haz clic en el título de la pista en el menú desplegable, selecciona "Establecer Formato de Muestra" para 16-bit PCM.

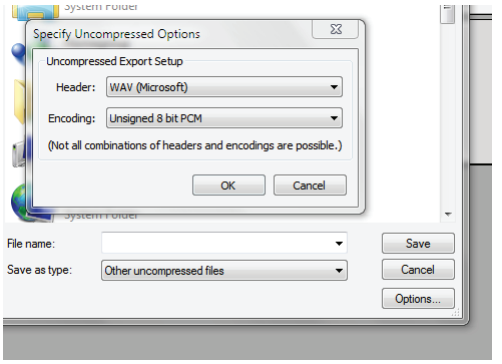


7. En el menú archivo, selecciona "Exportar ...".

8. Selecciona "Otros archivos sin comprimir".

9. Haz clic en el botón "Opciones".

10. Selecciona "WAV (Microsoft)" como tipo, "Unsigned 8 bit PCM" como codificación.

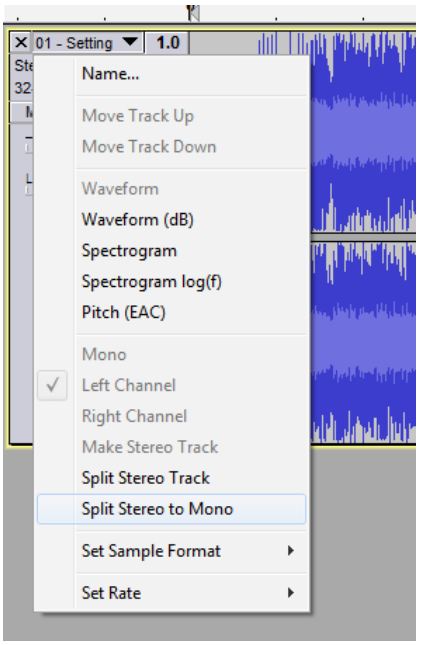


11. Asígnale un nombre al archivo (intenta que sea de 8 caracteres de largo), y pulsa el botón "Guardar" para guardarlo.

12. No rellenes ningún campo en la ventana "Editar Metadatos".

Convierte un Fichero Existente

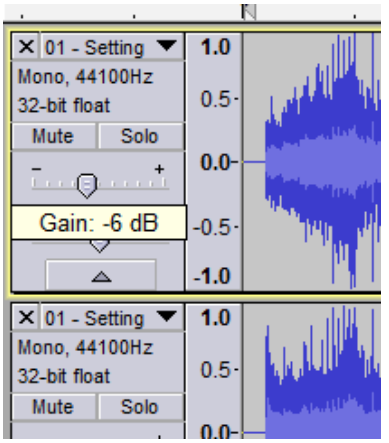
Si quieres convertir una pista de sonido existente:



1. Abre el fichero de sonido con Audacity.

2. Si es en estéreo, haz clic y despliega el título de la pista, luego elige la opción "Dividir pista estéreo a mono".

3. Reduce la ganancia a -6 dB en cada canal.



4. Haz clic en el menú "pistas", y selecciona "Mezclar y generar". Obtendrás un sonido de un solo canal (mono).

5. Selecciona toda la pista o la parte que te guste, y cópiala.

6. Haz clic en Files -> New.

7. Cambia la velocidad de proyecto a 22050.

8. Haz clic en Tracks -> Add New -> Audio Track.

9. Pega la pista aquí.

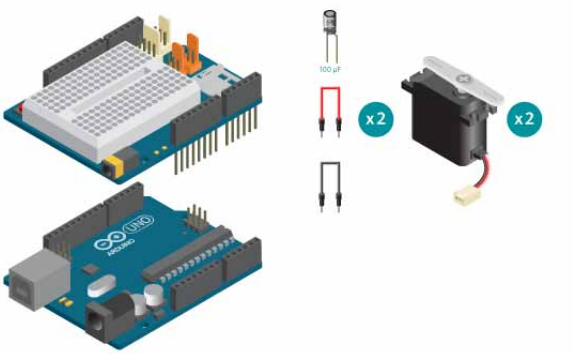
10. En las instrucciones "grabar sonido", sigue los pasos 5 hasta el final.

Ruedas

Construir robots que se puedan mover sobre dos ruedas es bastante común. Para ello, necesitas servos de giro continuo. Esto puede ser problemático ya que cuando hacen giros rápidos demandan más corriente de la que Arduino puede suministrarles. Por lo tanto, necesitas utilizar un condensador y que cuando las ruedas cambien el sentido sea suavemente. Hemos incluido esta segunda parte en la librería, por lo que no has de preocuparte.

Materiales

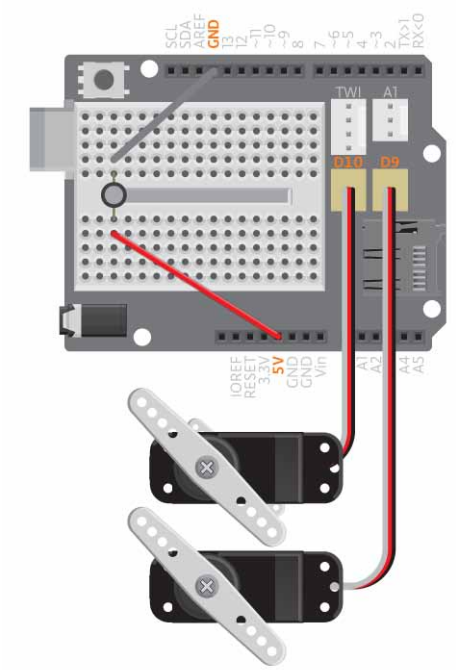
- 1 placa Arduino Uno
- 1 shield Básica Educativa
- 2 Servos de rotación continua
- 1 condensador de 100 microF
- 2 cables



Instrucciones

1. Conecta el servo de la rueda izquierda a D10 y el servo de la rueda derecha a D9.

2. Conecta el condensador entre GND y 5V, la pata corta a GND y la larga a 5V.



3. Carga el código: Archivo -> Ejemplos -> BasicEducationShield -> Ayuda -> Wheels

```
/*
Wheels
*/

#include <BasicEducationShield.h>
#include <Servo.h>

//wheels(left, right)
Wheels wheels=Wheels(10, 9);

void setup(){
//Initialize the servo wheels
wheels.begin();
}

void loop(){
//Makes the wheel spin forward
wheels.goForward();
delay(2000);

//Makes the wheels spin backwards
wheels.goBackwards();
delay(2000);

//Makes the wheels spin in opposite direction so that
//the vehicle will spin left
wheels.turnLeft();
delay(2000);

//Makes the wheels spin in opposite direction so that
//the vehicle will spin right
wheels.turnRight();
delay(2000);

//Makes the wheels stop
wheels.standStill();
delay(2000);
}
```

Cuando el código esté cargado, las ruedas deberían empezar a girar hacia delante durante 2 segundos, luego hacia atrás otros 2 segundos. `turnLeft()` hace que las ruedas giren en dirección opuesta una de la otra, de forma que el vehículo gira en sentido contrario de las agujas del reloj (antihorario), `turnRight()` lo hace girar en el sentido opuesto de forma que el vehículo gira en sentido horario. `standStill()` hace que ambos servos se paren.

¿No funciona?

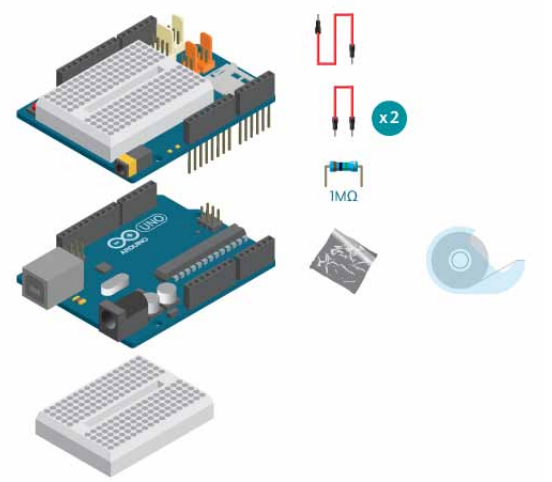
1. Asegúrate de que los servos están conectados en los mismos pines que los utilizados en el código. Asegúrate de que el primer pin digital en `begin(left, right)` es donde la rueda izquierda está conectada y el segundo donde lo está la derecha. Si tienes esto mal, el robot irá hacia atrás cuando se supone que tiene que ir hacia delante, etc.

Sensor Capacitivo

Un sensor capacitivo se compone de una resistencia de valor alto (en este ejemplo 1MOhm) y una pieza de material conductor (papel de aluminio). Puede ser utilizado como un sensor de contacto, bien como un sensor analógico que lea los valores del mismo o como un interruptor digital. El sensor necesita estar conectado a través de una breadboard separada para que la electrónica del shield no interfiera la lectura.

Materiales

- 1 placa Arduino Uno
- 1 shield Básica Educativa
- 1 Breadboard
- 1 resistencia 1MOhm
- 3 cables
- papel de aluminio
- cinta adhesiva

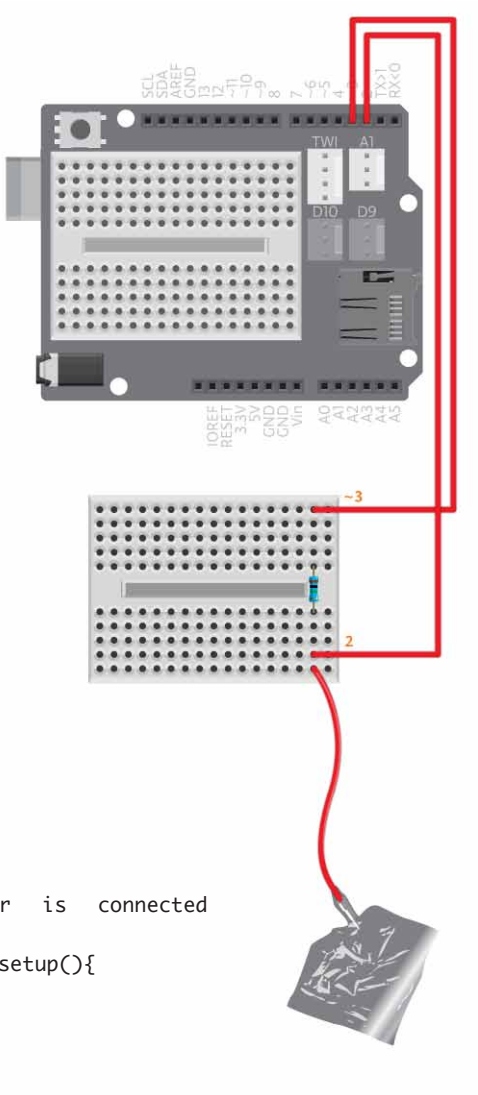


Instrucciones

1. Conecta un cable al Pin 2 y un cable al Pin 3.
2. Conecta una resistencia de 1MOhm entre el Pin 2 y Pin 3.

3. Conecta el Pin 3 con un cable suelto.

4. Corta un trozo de papel de aluminio, conéctalo al cable suelto con cinta adhesiva (asegúrate de que las piezas de metal tengan buena conexión).



Para leer los valores del sensor

Carga el ejemplo Archivo -> Ejemplos -> BasicEducationShield -> Help -> CapacitiveSwitchTest:

```
1  /*
2  CapacitiveSwitch Test
3  */#include <CapacitiveSensor.h>
4  #include  <BasicEducationShield.h>//The  sensor  is  connected
between 2 and 3 by default
5  CapacitiveSwitch me=CapacitiveSwitch(2,3);void setup(){
6  Serial.begin(9600);
7  }
8  void loop(){
9  //Print out the value of the sensor
10 me.test();
11 delay(30);
12 }
```

Abre el monitor serial para asegurarte de que el sensor funciona. Lo que verás ahora son los valores leídos desde el sensor capacitivo. Si nada está tocando el sensor, el valor debe ser inferior a 10 la mayor parte del tiempo; y sube hasta 1000 o más si lo pulsas con más fuerza.

¿No funciona?

1. Si la lectura no es normal, vuelve a comprobar la conexión. Tienes que tener especial cuidado con la conexión entre el papel de aluminio y el cable suelto, pueden perder contacto fácilmente.

2. Asegúrate de que has conectado el sensor a los mismos terminales digitales indicadas en el código. En CapacitiveSwitch (2,3) es el Pin 2 quien envía y el Pin 3 quien recibe. Esto significa que el extremo que está suelto debe estar conectado al Pin 3.

Nota: Asegúrate de que los dos Pins digitales que utilizas están entre 2 y 8. Al usar la shield Básica Educativa, los Pins digitales, 9, 10, 11, 12 o 13 no sirven para los sensores capacitivos.

Sensor como interruptor

Si quieres utilizar el sensor capacitivo como un interruptor, necesitas medir un valor de umbral. Anota el valor obtenido cuando nada está en contacto con el papel de aluminio y el valor cuando está en contacto con un objeto deseado.

Ahora abre el ejemplo Interruptor capacitivo. Carga el ejemplo Archivo -> Ejemplos -> BasicEducationShield -> Help -> CapacitiveSwitchTest:

```
1  /*  CapacitiveSwitch
2  */#include <CapacitiveSensor.h>
3  #include <BasicEducationShield.h>//The sensor is connected between 2 and 3 by default
4  CapacitiveSwitch me=CapacitiveSwitch(2,3);void setup(){
5  Serial.begin(9600);
6  me.config(400);
7  }
8  void loop(){
9  Serial.println(“Please press...”);
10 //Wait until the capacitive sensor is pressed.
11 Serial.println(me.pressed());delay(1000);Serial.println(“Please release...”);
12 //Wait untill the capacitive sensor is released.
13 Serial.println(me.released());do{   //Print values bigger than 10. Print 0 otherwise.
14 Serial.println(me.getValue(20));
15 }while(true);
16 }
```

En la configuración de la `línea me.config (umbral)`, sustituye `umbral` por un número entre los dos valores que anotaste anteriormente. El valor por defecto es 400, en la mayoría de los casos, no tendría que cambiarse a menos que necesites utilizar algo que no fuera el dedo para tocar el sensor.

Abre el monitor serial y sigue las instrucciones mostradas para ver si funciona. Cada vez que ejecutes una instrucción, el monitor serial imprimirá "1".

Puedes utilizar varios sensores capacitivos para el mismo Arduino. En este caso, el Pin emisor (2 en este caso) puede ser compartido.

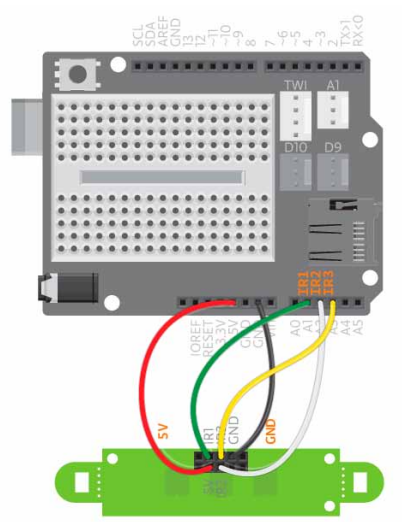
IR Array

El IR Array consiste en 3 sensores IR (Infra Rojo). Si te fijas, podrás ver que cada sensor tiene dos "puntos", uno azul y otro negro. El azul es el LED IR y el negro es el verdadero sensor IR. Cuando se conecta a alimentación el LED IR, emite luz infrarroja. No puedes verla directamente, pero si tienes una cámara digital (como la del teléfono móvil) y la enfocas al LED IR, podrás verla claramente.

Cuando acercas los sensores a una superficie blanca, la superficie reflejará gran parte de la luz que emite el LED IR y que será detectada por el sensor IR. Si por el contrario lo acercas a una superficie negra, ésta absorberá la luz y muy poca será reflejada y por tanto, detectada. De este modo, se puede distinguir entre el blanco y el negro.

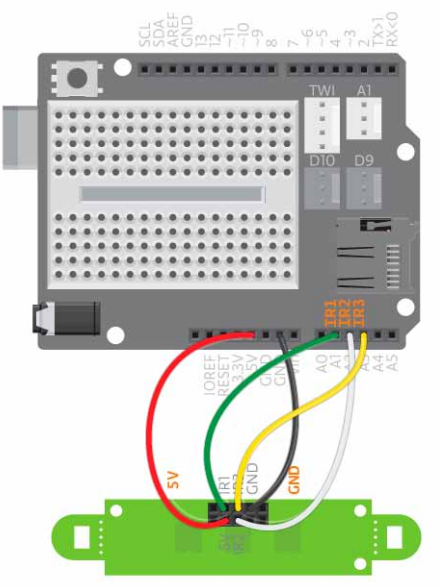
Materiales

- 1 IR Array
- 5 cables



Instrucciones

1. Conecta IR1 a A1, IR2 a A2 y IR3 a A3. Conecta el IRArray a GND y 5V según la ilustración.



Código

Puedes encontrar el código en Archivo -> Ejemplos -> BasicEducationShield-> Ayuda-> IRArray

```
#include “BasicEducationShield.h”
#if ARDUINO >= 100
#include “Arduino.h”
#else
#include “WProgram.h”
#endif

IRArray::IRArray(int IR1, int IR2, int IR3){
    this->IR1=IR1;
    this->IR2=IR2;
    this->IR3=IR3;
    sensPins[0] = IR1;
    sensPins[1] = IR2;
    sensPins[2] = IR3;
    for(int i=0; i<3; i++){
      sensorVal[i]=0;
    }
}

int IRArray::readBinary(){
    for(int i=0; i<3; i++){
        toBinary[i]=0;
    }

    bool check = true;
    while(check){
        for(int i=0; i<3; i++){
            sensorVal[i] = constrain(map(analogRead(sensPins[i]), 350, 400, 1, 0), 0,
1);
        }

        for(int i=0; i<3; i++){
            if(sensorVal[i]) toBinary[i]=1;
           else if(!sensorVal[i]&&!sensorVal[i+1]&&!sensorVal[i+2]) check=false; //if
all three are 0
        }
    }
    return translateBinary();
}
```

```
38
39 int IRArray::translateBinary(){
40   byte o = (toBinary[2]?1:0)|(toBinary[1]?2:0)|(toBinary[0]?4:0);
41   return o;
42 }
43
44 int IRArray::readLine(){
45     int sum=0;
46     for(int i=0; i<3; i++){
47         int reading = constrain(analogRead(sensPins[i]), 60, 400);
48         //int reading = constrain(analogRead(sensPins[i]), low[i], high[i]);
49
50         sensorVal[i]=map(reading,60,400,0,127);
51         sum+=sensorVal[i];
52         //Serial.println(sum);
53     }
54     int vel = 0;
55     if(sum>LINE_THRESSHOLD) vel = calculateVelocity(sum);
56
57     return vel;
58 }
59
60 int IRArray::calculateVelocity(int s){
61     s/=3;
62
63     int diff = ((sensorVal[0]<<5)-(sensorVal[2]<<5))/s;
64     diff = constrain(diff,-100,100);
65
66     int velocity = (diff * KP)/10 + (diff-last_diff)*KD;
67      velocity = constrain(velocity, -100, 100);
68     //Serial.println(velocity);
69     last_diff = diff;
70
71     delay(INTEGRATION_TIME);
72
73     return velocity;
74 }
75
76 void IRArray::test(){
77     for(int i=0; i<3; i++) sensorVal[i] = constrain(analogRead(sensPins[i]), 60, 400);
78
79     Serial.print("IR1: ");
80     Serial.print(sensorVal[0]);
81     Serial.print("  IR2: ");
82     Serial.print(sensorVal[1]);
83     Serial.print("  IR3: ");
84     Serial.println(sensorVal[2]);
85 }
```

Carga el código y abre el monitor Serial. El valor de cada sensor se mostrará ahí. Cuando enfocas el IR Array a una superficie blanca todos los sensores deberán dar un valor de 400. Cuando lo enfoques sobre una superficie negra, debe de bajar hasta alrededor de 330. Asegúrate de que has conectado todos los pins del sensor de forma correcta alternando entre blanco y negro en frente de cada sensor. Por ejemplo, si mueves el IR1 de blanco a negro, pero sólo el valor de IR2 cambia, necesitas o bien cambiar los pins analógicos del código o bien cambiar las conexiones a los pins analógicos en Arduino.

Resistencias

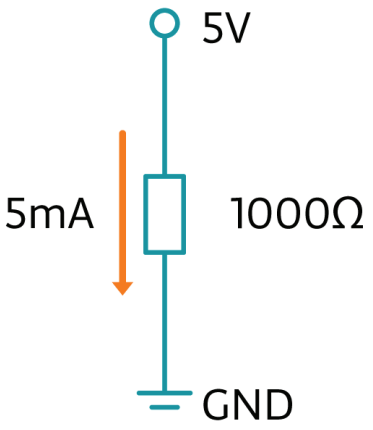
Las resistencias son un tipo de componente electrónico que pueden ser usadas para limitar la corriente de un circuito. También permite dividir el voltaje de la fuente de alimentación. Son conocimientos básicos tanto para ser inventor de cohetes o si simplemente quieres aprender algo de electrónica.

Podemos pensar en un circuito como si fuese un jardín, que se parecen bastante. El voltaje es como el grifo, puedes girar el grifo para ajustar la presión del agua. La corriente es como la cantidad de agua que sale por la tubería. Puedes ajustarla apretando la tubería, cuanto más la estrujes, menos agua saldrá. Con esta analogía, la resistencia es equivalente a cuánto la estrujas.

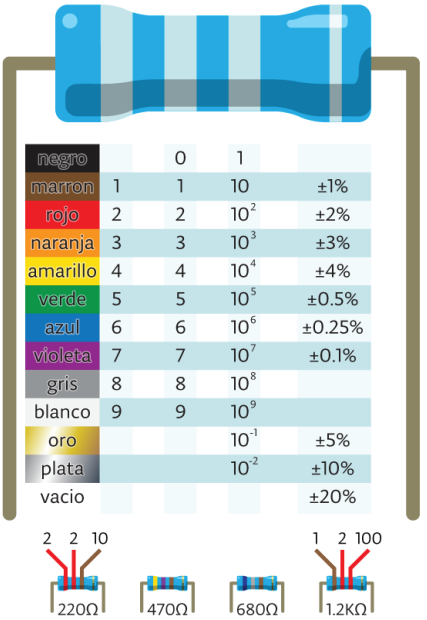
Las unidades de voltaje, corriente y resistencia son, respectivamente, Voltios (V), Amperios (A) y Ohmios (ohm). Puedes calcular la corriente de un circuito mediante la ley de Ohm:

$$I = V/R$$

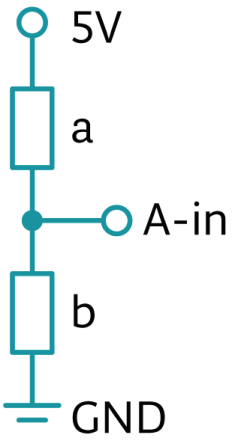
Donde I es la corriente, V es el voltaje total y R es la resistencia total. Así, si el circuito tiene 5v, 1000 ohm (1 KOhm) de resistencia total, la corriente será de 0.005A (5mA).



¿Cómo se puede saber el valor de una resistencia? La posición y el color de las líneas en la resistencia te dicen su valor. Mira esta tabla para entender cómo funciona el código:



Cuando conectamos LEDs a Arduino, suele haber una resistencia de 220 Ohm conectada en serie. La resistencia limita la corriente, de forma que no haya demasiada a través del LED y lo rompa. A su vez, también limita el brillo del LED.

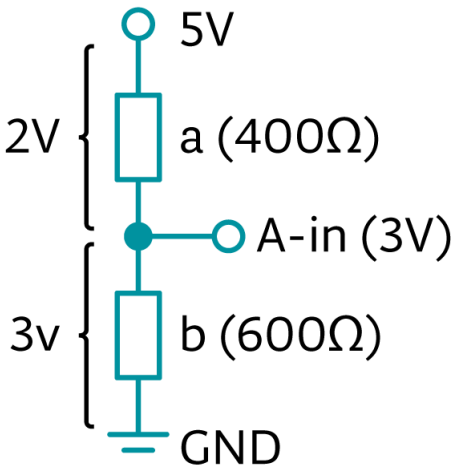


Cuando hay varias resistencia en un circuito, ellas comparten los voltajes. ¿Recuerdas cómo obteniamos señales analógicas? Mira el ejemplo del potenciómetro. Un extremo está conectado a 5V, el otro a GND y el pin intermedio está conectado al pin analógico. El potenciómetro es como dos resistencias variables. En este caso, ambas comparten 5V, y cómo los reparten depende de su resistencia. Puedes usar esta fórmula para calcular los voltajes:

$$Va = V/(Ra + Rb) * Ra$$

Donde Va es el voltaje compratido por 'a' y V es el voltaje total, Ra y Rb son las resistencias 'a' y 'b' respectivamente.

Por ejemplo, si tenemos dos resistencias de 400 ohm y 600 ohm respectivamente, con 5 voltios en la fuente de alimentación, por la fórmula sabemos que tomarán un voltaje de 2v y 3v respectivamente.



Como sabemos, los pins analógicos toman valores entre 0 y 1023 al medir el voltaje en un cierto punto. ¿Cómo obtener la lectura de ese voltaje? Continuemos con el ejemplo anterior. La fuente de alimentación tiene 5V, la resistencia 'a' 2V, por lo que en la entrada de A habrán los restantes 3V. La resistencia 'b' tiene 3V por la entrada de A, dejando 0V en GND, como debe ser. Sabemos que 5V corresponden a 1023 en valor analógico, por lo que aquí deberemos obtener 614 en la lectura.

¿Qué aplicaciones tiene? Mira el proyecto "Secuenciador". En este proyecto intentamos obtener diferentes lecturas analógicas pequeñas, por lo que las resistencias son seleccionadas para que la diferencia sea máxima. Intenta hacer algunos cálculos a partir de los diagramas de abajo y los valores proporcionados de las resistencias. Compara tus resultados con los valores en el sketch, ¿ves la relación? ¿Cómo añadirías más secuencias? Pista: tienes que añadir resistencias de diferente valor, y siempre es bueno calcularlas de antemano.

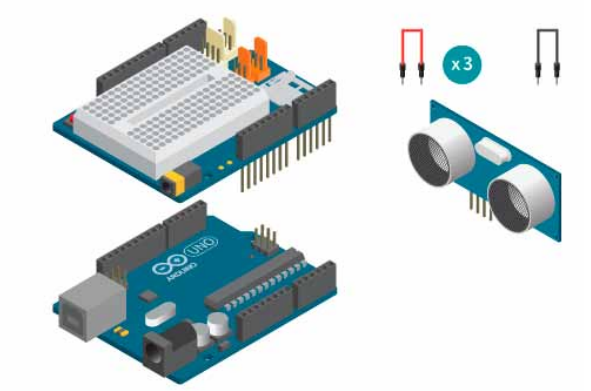
Nota: los valores calculados y los obtenidos en la lectura analógica no son exactamente los mismos, porque siempre existen errores en la realidad. El valor de las resistencias no es exacto, el circuito posee resistencias internas, etc. Este es el motivo por el que usamos un cierto rango para comparar las resistencias en el sketch. Si quieres añadir más secuencias, puedes reducir los rangos originales para que quepan los nuevos.

Sensor ultrasónico

Un sensor ultrasónico o detector de proximidad se utiliza para detectar la distancia al objeto más cercano en frente del sensor. Se utiliza el ultrasonido, que es sonido con una frecuencia más alta que la que los humanos pueden oír. Como probablemente sabes, el sonido viaja en el espacio y puede reflejarse en las superficies creando un eco. Este eco es lo que utiliza el sensor para calcular la distancia. Los sensores envían una ráfaga de ultrasonido y luego esperan al eco. Al saber la velocidad del sonido y el tiempo entre la explosión y la detección se puede calcular la distancia.
Entre menos tiempo tome detectar el eco, más cerca estará el objeto.

Materiales

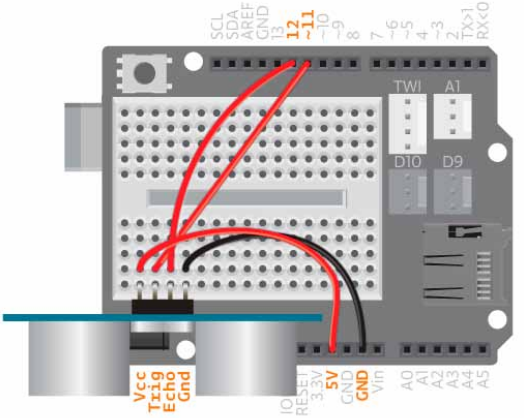
- 1 placa Arduino Uno
- 1 Basic education shield
- 1 Ultrasonic sensor
- 4 jumper wires



Instrucciones

1. Conecta VCC a 5V, Trig al Pin digital 11, Echo al pin digital 12 y GND to "ground" cómo la indica la siguiente ilustración.

2. Abrir Archivo>Ejemplo>Basic EducationShield>Help>UltrasoniсSensor.



Multímetro

Un multímetro es un instrumento utilizado para medir la corriente, voltaje y resistencia. Puedes leer más sobre la relación entre éstos en la referencia de las resistencias. Ahora veremos tres ejemplos de cuándo y cómo utilizar el multímetro.

El multímetro se compone de una pantalla de datos, un indicador para seleccionar el tipo de medición, y dos cables con terminales en punta metálica que se utilizan para tocar las partes que se desea verificar. El cable negro, por regla, siempre debe estar enchufado en la ranura marcada como "COM", en inglés "common" (común). En los tres ejemplos a continuación, el cable rojo debe ser conectado a la ranura marcada como "V" (voltaje).

Continuidad

Cuando ponemos a prueba la continuidad, es decir, si la corriente fluye a través de un circuito, lo que queremos es asegurarnos de que el cable que vamos a utilizar esté intacto o que no haya corto circuitos.

Materiales

- 1 Multímetro
- 1 cable

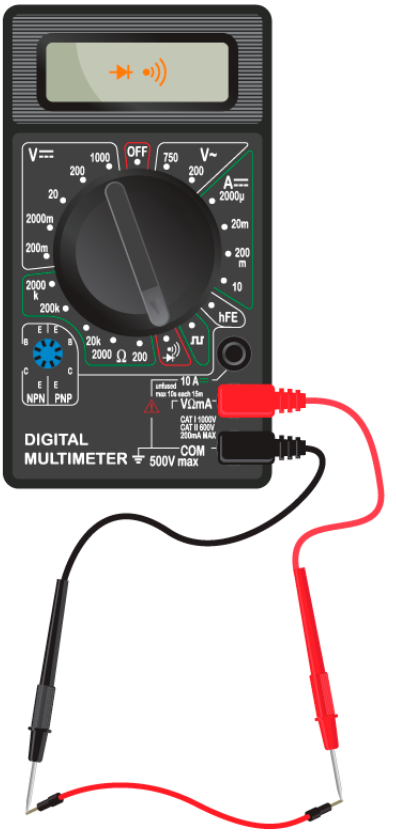
Instrucciones

1. Gira el indicador hasta el icono de sonido.

2. Toma las dos terminales y haz que sus puntas se toquen. Oirás un pitido que significa que hay un circuito cerrado.

3. Toma un cable y las dos terminales del multímetro. Coloca las puntas de las terminales en cada uno de los extremos del cable. Lo más probable es que el multímetro emita un pitido, pero si no es así, eso significa que el cable está roto.

Si has soldado un circuito pero no está funcionando por alguna razón, este es un buen comienzo para hacer pruebas de funcionamiento. Al revisar todas las soldaduras puedes asegurarte de que el circuito está cerrado como es debido y que por lo tanto no hay conexiones donde no debe haberlas.



Resistencia

Si no recuerdas el código de colores de las resistencias de memoria, puedes medir la resistencias con el multímetro.

Materiales

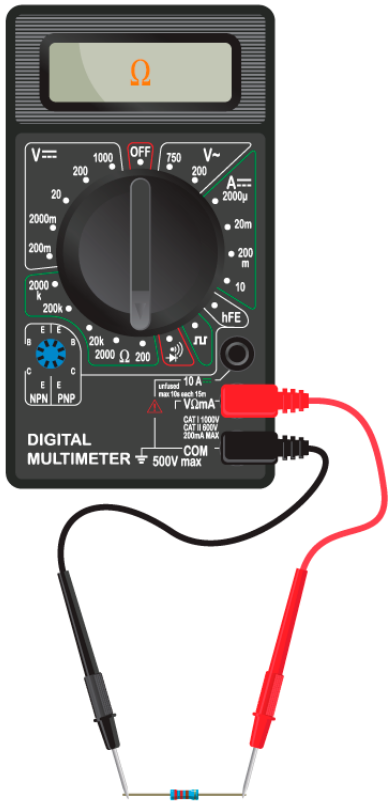
- 1 Multímetro
- 1 Resistencia de cualquier valor

Instrucciones

1. Gira el indicador del multímetro hacia la ranura marcada con el signo de Ohm. Puedes empezar por girarlo en el valor más bajo, que es de 200 ohmios. Ese valor significa que podemos medir una resistencia de hasta 200 ohmios.

2. Toca los extremos de la resistencia con las puntas de las terminales del multímetro. Si la pantalla muestra "0.L", eso significa que la resistencia es mayor de lo que ha establecido el multímetro para comprobar.

3. Gira el indicador al siguiente nivel, hasta que puedas leer un número en la pantalla. Si el indicador está en "20k" y en la pantalla dice "10.0" esto significa que la resistencia es de 10k ohms.

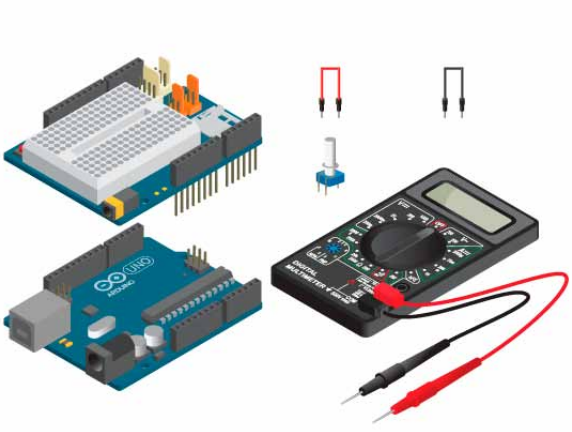


Voltaje

En este ejemplo te mostraremos cómo medir el voltaje a través de un a utilizar tiene un rango de 0V a 5V.

Materiales

- 1 Multímetro
- 1 Arduino Uno
- 1 Shield Básica Educativa
- 1 Potenciómetro
- 2 cables



Instrucciones

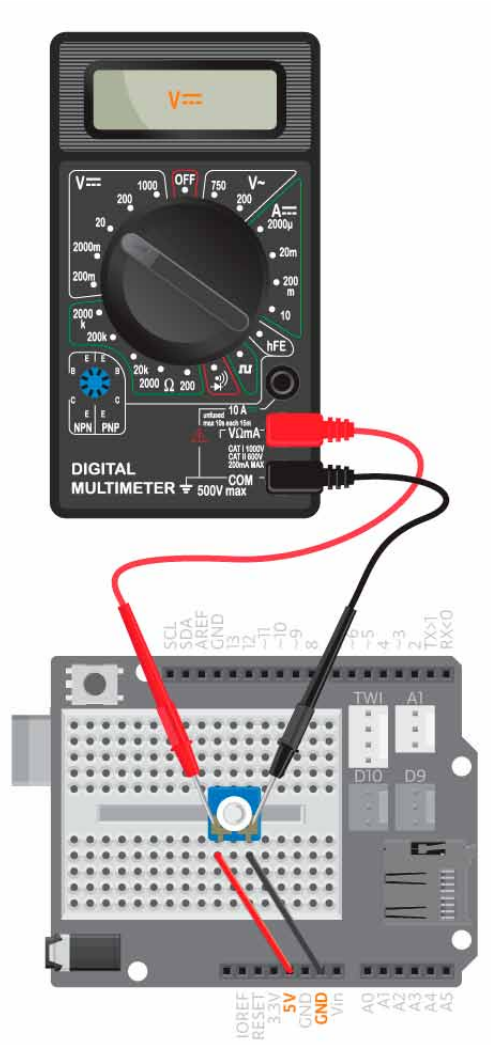
1. Conecta el potenciómetro a GND y a 5V en la breadboard de la shield educativa.

2. Conecta Arduino a la computadora.

3. Gira el indicador del multímetro a "20" en el lado "V" DC. Esto significa que podemos medir el voltaje de hasta 20 V.

4. Toca el pin de GND del potenciómetro con el cable negro del multímetro y el pin medio del potenciómetro con el cable rojo.

5. Gira el potenciómetro para ver los cambios de voltaje.



Créditos

El equipo responsable de la creación de estos ejercicios y contenidos es:

Xun Yang
Diseñador Interactivo Senior

Clara Leivas
Diseñador Interactivo Junior

Laura Balboa
Directora de arte, diseño gráfico y editorial

Kristoffer Engdahl
Especialista en modelado e impresión en 3D

Tien Pham
Diseñador industrial

David Cuartielles
Coordinador

Arduino Verkstad 2014