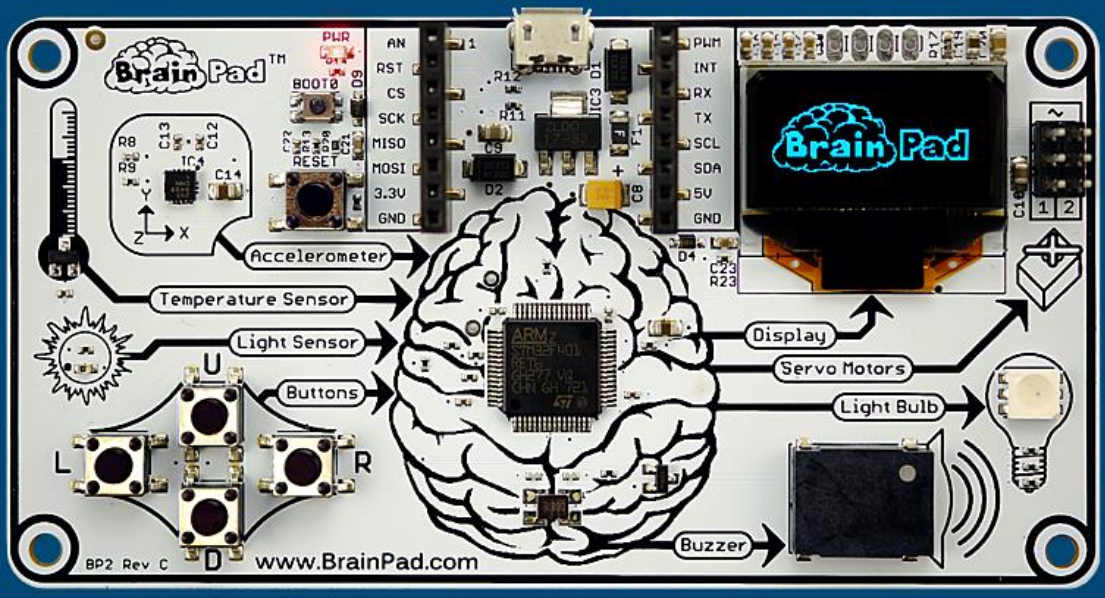
GHI electronics

BrainPad™

BEGINNERS' GUIDE



CONTENTS

INTRODUCCIÓN

BrainPad fue diseñado como una poderosa herramienta educacional STEM (ciencia, tecnología, ingeniería y matemáticas por sus siglas en inglés), que puede ser usada para enseñar a cualquiera, desde niños a estudiantes y profesionales.

Este libro electrónico gratuito se entrega como una guía para principiantes al mundo del BrainPad. Para más información visite el sitio http://www.brainpad.com

EL AUTOR

Gus Issa es fundador y ejecutivo principal (CEO) de GHI Electronics. Dio sus primeros pasos en programación y electrónica cuando era solo un adolescente. Desde sus inicios se esforzó por autoeducarse, con recursos limitados y sin acceso a Internet. Esta lucha, junto con el profundo impacto que la educación tuvo en su vida, gestaron su pasión por enseñar a otros. Ha dedicado incontables horas de su propio tiempo junto con recursos de su empresa para crear una forma de compartir sus conocimientos, que se tradujo en el BrainPad.

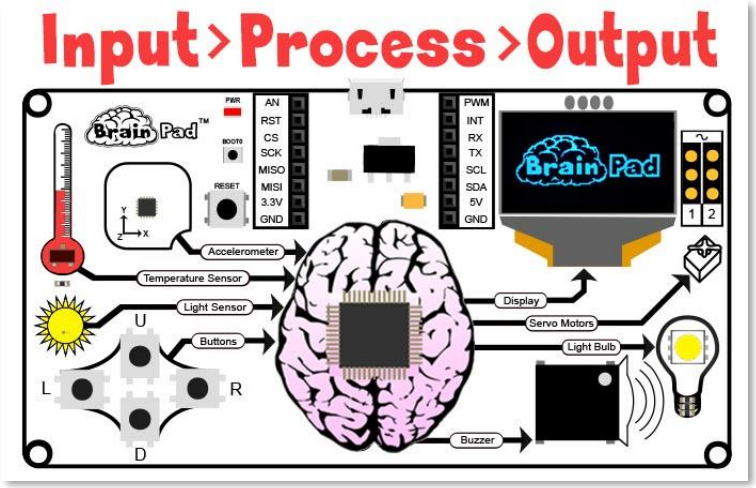
VISIÓN GLOBAL

Nos esforzamos por hacer el BrainPad accesible a toda cultura e idioma. Si te atrae el desafío y quieres traducir este libro a tu idioma, por favor contáctanos en el enlace http://www.brainpad.com/Company/ContactUs.

LICENCIA

EL CONCEPTO

El BrainPad es autoexplicativo. Muestra de forma lógica la forma en que trabajan los computadores. Hay 4 entradas que están conectadas al cerebro. El cerebro (procesador) piensa sobre lo que está ocurriendo en esas entradas para decidir cómo controlar las cuatro salidas, en un esquema lógico del tipo Entrada->Proceso-> Salida.



LA FILOSOFIA

La educación STEM requiere una plataforma evolutiva, no un juguete. BrainPad evoluciona a medida que tú lo haces. Muchos juguetes son comercializados como herramientas STEM, pero la mayoría no tiene la versatilidad para mantener interesados a los estudiantes por mucho tiempo. El BrainPad combina programación (de principiante a profesional), electrónica, ingeniería, robótica y mucho más en una plataforma de bajo costo que puede ser usada desde la enseñanza básica hasta la Universidad. Está respaldado por nuestros 15 años de experiencia profesional en ingeniería. Las mismas herramientas usadas para programar nuestros controladores industriales pueden ser usadas para programar el BrainPad. Ninguna otra plataforma STEM puede acompañar a los estudiantes desde la programación 'drag and drop' hasta la programación profesional y la ingeniería como lo hace el BrainPad.

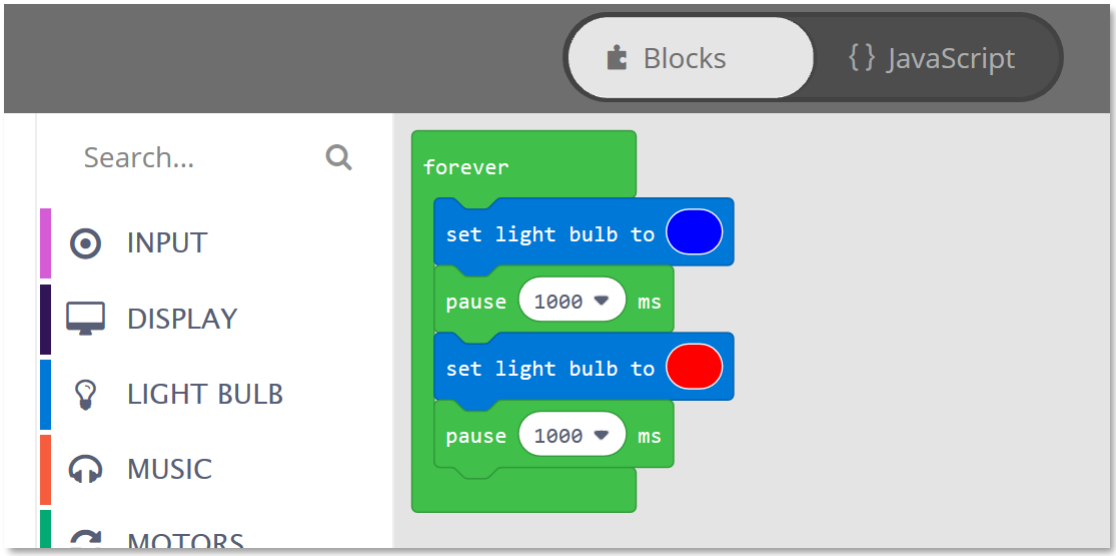
DOS CAMINOS

El BrainPad ofrece dos caminos para aprender a programar, llamados '**Start Making**' (comenzar haciendo) y '**Go Beyond**' (yendo más allá). Como ya habrás imaginado, la primera opción facilita comenzar haciendo proyectos mientras que la segunda te lleva más allá permitiéndote usar las mismas herramientas de programación de los profesionales.

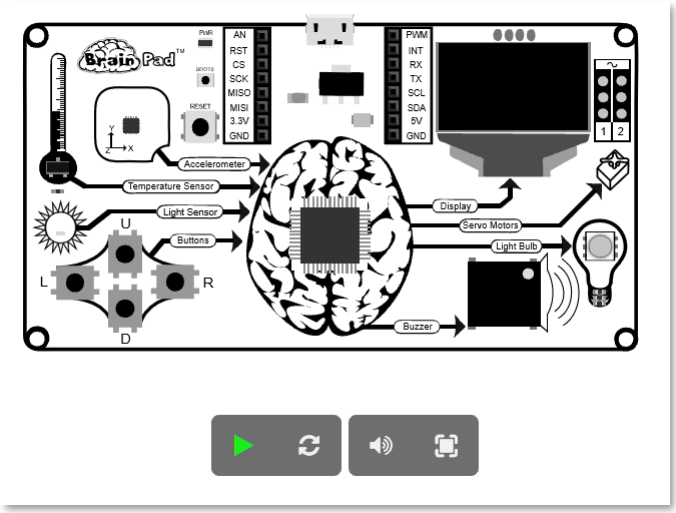
LA OPCIÓN START MAKING

Con la opción Start Making no se requiere instalar software en tu computador – todo funciona mediante tu navegador Internet usando MakeCode de Microsoft. La opción Go Beyond requiere un poco de configuración e instalación de software. La opción más popular es utilizar Microsoft Visual Studio.

Con Microsoft MakeCode (opción Start Making) puedes programar solo con ordenar bloques y opcionalmente puedes escribir Código JavaScript directamente en tu navegador. Aquí vemos un programa que enciende la ampolleta de color azul y luego roja cada un segundo, de forma repetitiva, "para siempre".



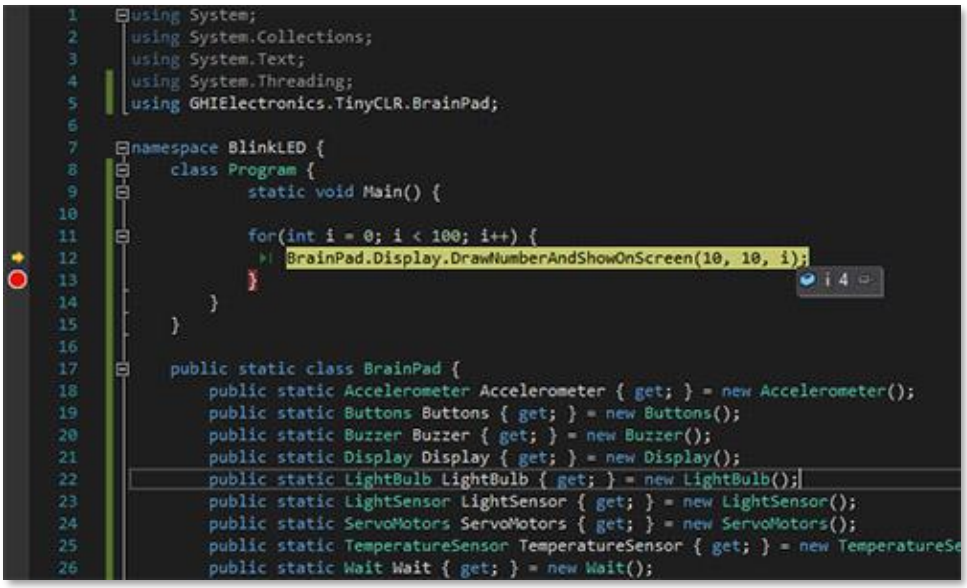
Se incluye un simulador (mostrado abajo), que es básicamente un BrainPad virtual que ejecuta tu programa en la pantalla de tu computador. Se despliega al lado izquierdo de la ventana de Microsoft MakeCode. El simulador te permite probar programas sin necesidad de cargarlos en el BrainPad. Esto es también una excelente opción en caso que aún no hayas recibido tu BrainPad.



LA OPCIÓN GO BEYOND

Las cosas se ponen más avanzadas con la opción Go Beyond. En este caso habilitarás un computador con Visual Studio de Microsoft para codificar y depurar o hacer 'debug' de los programas que se ejecutan en el BrainPad.

Gracias al sistema operativo TinyCLR de GHI Electronics, el BrainPad puede ser programado en C# y Visual Basic. Se incluyen capacidades de debug, tales como ejecutar el código paso a paso, establecer puntos de detención y revisar los valores de variables durante la ejecución del programa. En Go Beyond estarás programando el BrainPad de la misma forma que un profesional programa un computador o una app de smartphone. Este libro te muestra los dos caminos.



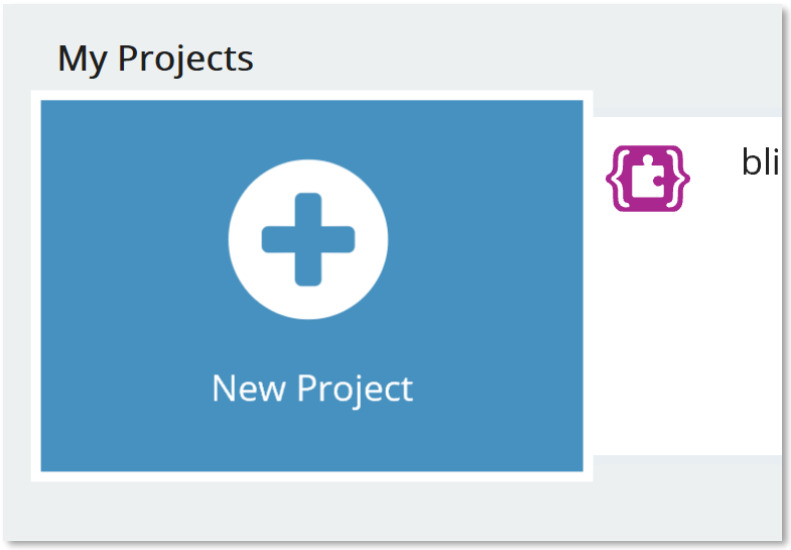
```
using System;
using System.Collections;
using System.Text;
using System.Threading;
using GHIElectronics.TinyCLR.BrainPad;

namespace BlinkLED {
    class Program {
        static void Main() {

            for(int i = 0; i < 100; i++) {
                BrainPad.Display.DrawNumberAndShowOnScreen(10, 10, i);
            }
        }
    }

    public static class BrainPad {
        public static Accelerometer Accelerometer { get; } = new Accelerometer();
        public static Buttons Buttons { get; } = new Buttons();
        public static Buzzer Buzzer { get; } = new Buzzer();
        public static Display Display { get; } = new Display();
        public static LightBulb LightBulb { get; } = new LightBulb();
        public static LightSensor LightSensor { get; } = new LightSensor();
        public static ServoMotors ServoMotors { get; } = new ServoMotors();
        public static TemperatureSensor TemperatureSensor { get; } = new TemperatureSe
        public static Wait Wait { get; } = new Wait();
```

YENDO AÚN MÁS ALLÁ (BEYOND THE BEYOND!)

El BrainPad también puede ser programado con otras opciones, no detalladas en este libro. Por ejemplo, con las herramientas Arduino (https://www.arduino.cc), MicroPython (https://www.micropython.org) y Arm Mbed (https://www.mbed.com).

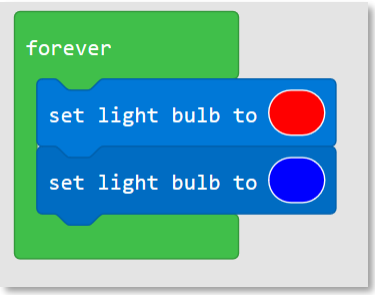
COMENCEMOS A HACER (START MAKING)

Ahora, ¡basta de charla y comencemos a entretenernos con algunos proyectos! Para eso usaremos MakeCode así que ingresa al sitio https://makecode.brainpad.com/. Este sitio incluye mucho material y proyectos, pero nosotros iremos directo a comenzar nuestro propio proyecto. Así que has click en el botón 'New Project'.



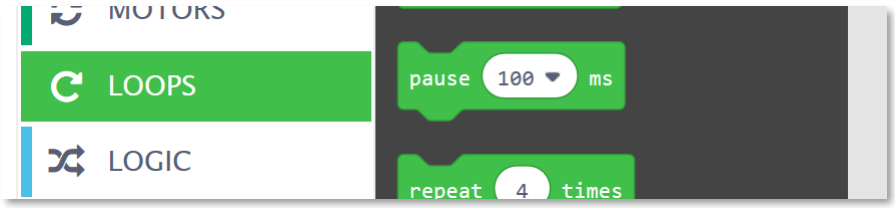
Ahora, arrastra con el mouse el bloque "set light bulb to", ubicado bajo la opción LIGHT BULB (ampolleta), y ponlo dentro del bloque "forever". Este bloque "forever" es utilizado para poner el código que deseas que se ejecute permanentemente. Las instrucciones dentro del bloque "forever" se ejecutarán en secuencia todo el tiempo que el BrainPad esté energizado. Cuando todas las instrucciones se han ejecutado, el BrainPad vuelve a ejecutarlas, comenzando por la primera del bloque "forever". Esto se conoce como un bucle infinito (o "loop" infinito) y es usado muy frecuentemente en la programación de computadores.

Repite el paso anterior y agrega un nuevo bloque “set light bulb to”. Haz click en el óvalo de color del segundo bloque para cambiar su color. Esto es lo que deberías tener:

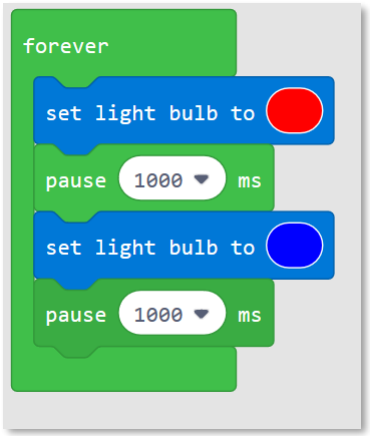


EL programa que hemos creado encenderá la ampolleta en color rojo y luego la cambiará a azul, repetidamente, para siempre. Sin embargo, puede que la ampolleta no se comporte como esperas en el simulador. Incluso si cargas el programa en el BrainPad puede que no funcione bien. ¿Por qué? Simplemente porque no le hemos dicho cuanto tiempo debe permanecer en un color antes de cambiar. Debido a eso el color cambia demasiado rápido y no podemos ver los colores individuales.

Para resolver esto, arrastra y agrega un bloque “pause” (pausa) debajo de cada uno de los bloques “set light bulb to”. Los bloques “pause” se encuentran bajo el menú LOOPS (ciclos).



Necesitamos agregar una pausa después que la ampolleta se ponga roja y lo mismo después que se ponga azul. Fijaremos las pausas en 1 segundo. Como las pausas están definidas en milisegundos, fijaremos su duración en 1000 (1000 milisegundos = 1 segundo).



Observa el simulador para verificar que la ampolleta está funcionando como deseamos. ¿Nada mal, cierto? ¿Qué te parece si ahora cargamos este programa en el BrainPad?

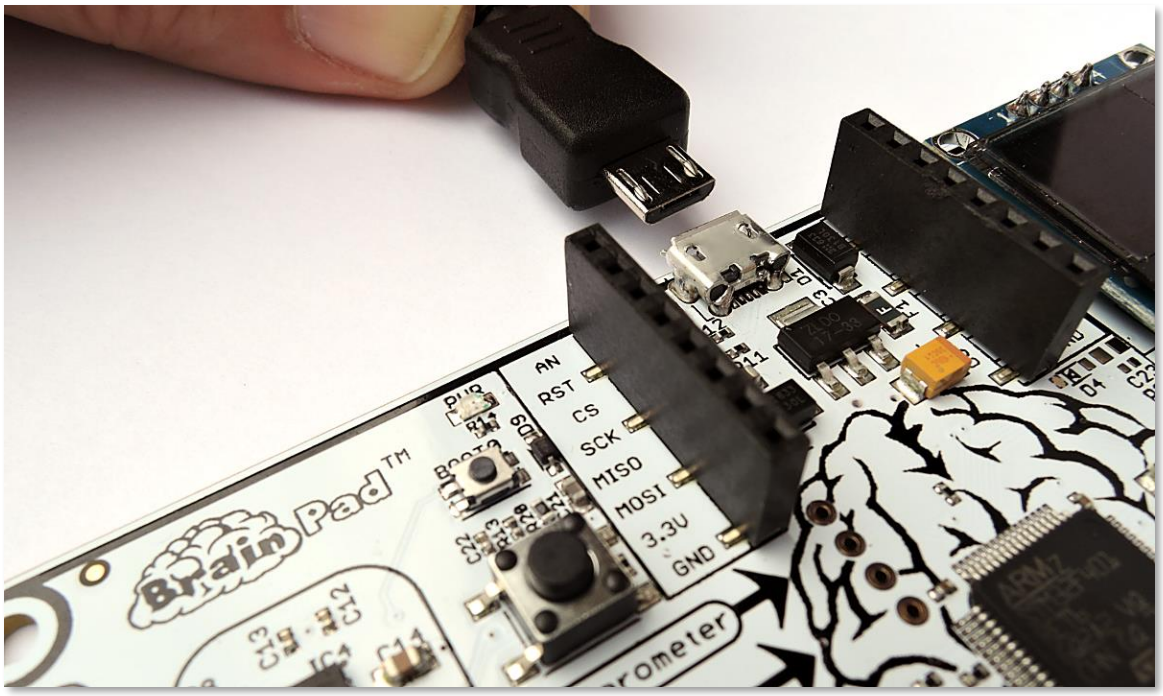
CARGANDO EL PROGRAMA EN EL BRAINPAD

El proceso de copiar (cargar) un programa en el BrainPad es fácil, pero puede que te cueste un poco la primera vez. Primero, debes conectar el BrainPad a tu computador usando un cable micro-USB. Entre paréntesis, para hacer esto puedes usar Windows, un Chromebook, un Mac o casi cualquier equipo con un navegador Internet moderno y un puerto USB.

Junto con tu BrainPad te debe haber llegado un cable micro-USB y es muy probable que tú tengas uno en tu casa. Si no tienes uno no te preocupes. Son fáciles de conseguir en el supermercado o una tienda de electrónica.

El conector más pequeño del cable se conecta al BrainPad.

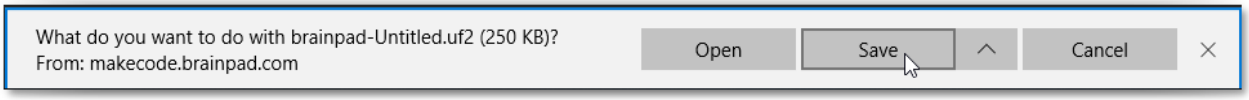
El conector más grande se conecta a tu computador.



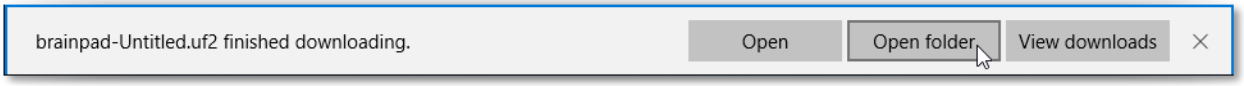
Cuando conectas el cable se debe encender la luz roja PWR en el BrainPad.

Vuelve al navegador con el programa que preparaste. Haz click en el botón “Download” (descarga) y guarda el archivo con tu programa en tu computador.

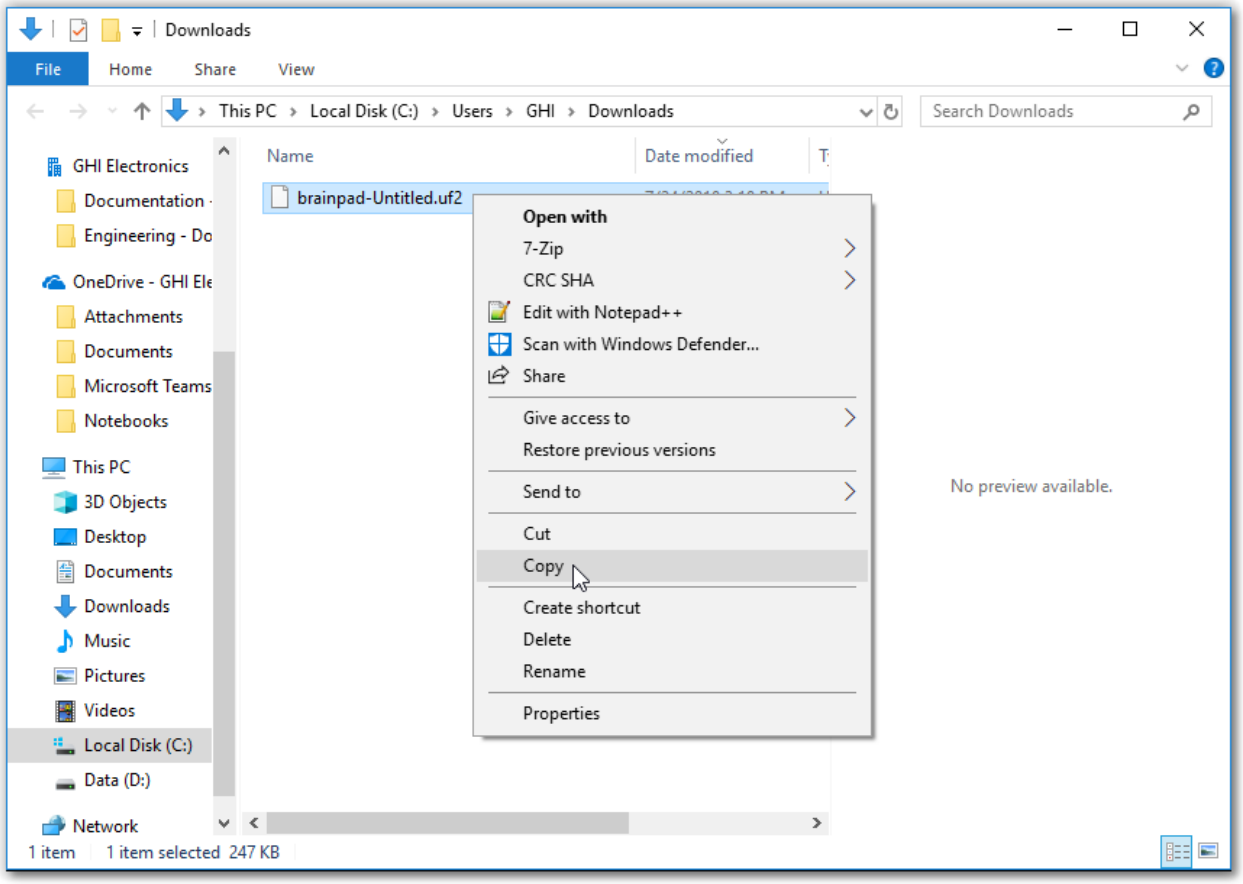
Si estás usando el navegador Microsoft Edge, deberías ver una ventana como la que se muestra abajo:



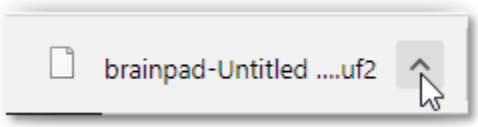
Haz click en el botón “Save” y luego en “Open folder” en la siguiente ventana que aparecerá, para mostrar el archivo descargado.



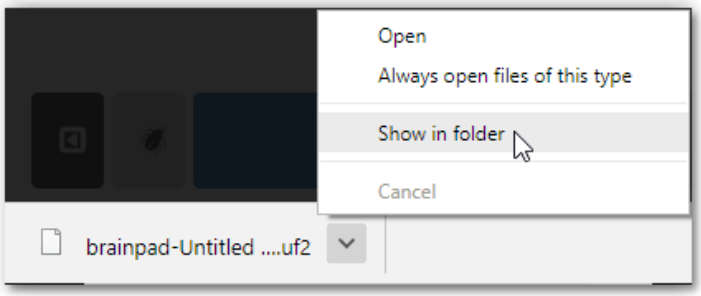
El archivo debe aparecer destacado. Haz click en “Copy” con el botón derecho del mouse para copiarlo.



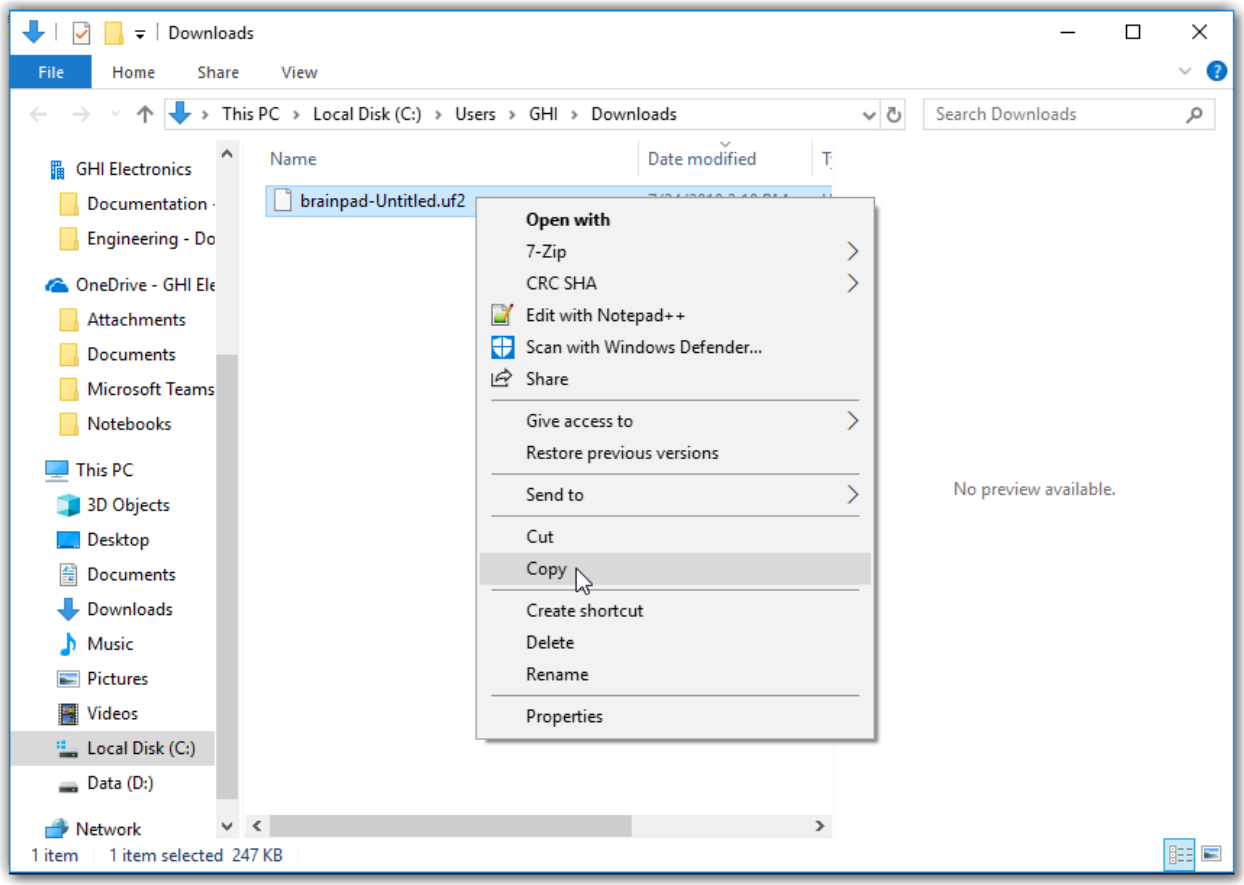
Si estás usando el navegador Chrome, cuando descargues el archivo éste aparecerá en la parte inferior izquierda de la pantalla:



Haz click en la pequeña flecha que apunta hacia arriba y luego selecciona “Show in folder” (mostrar en carpeta).



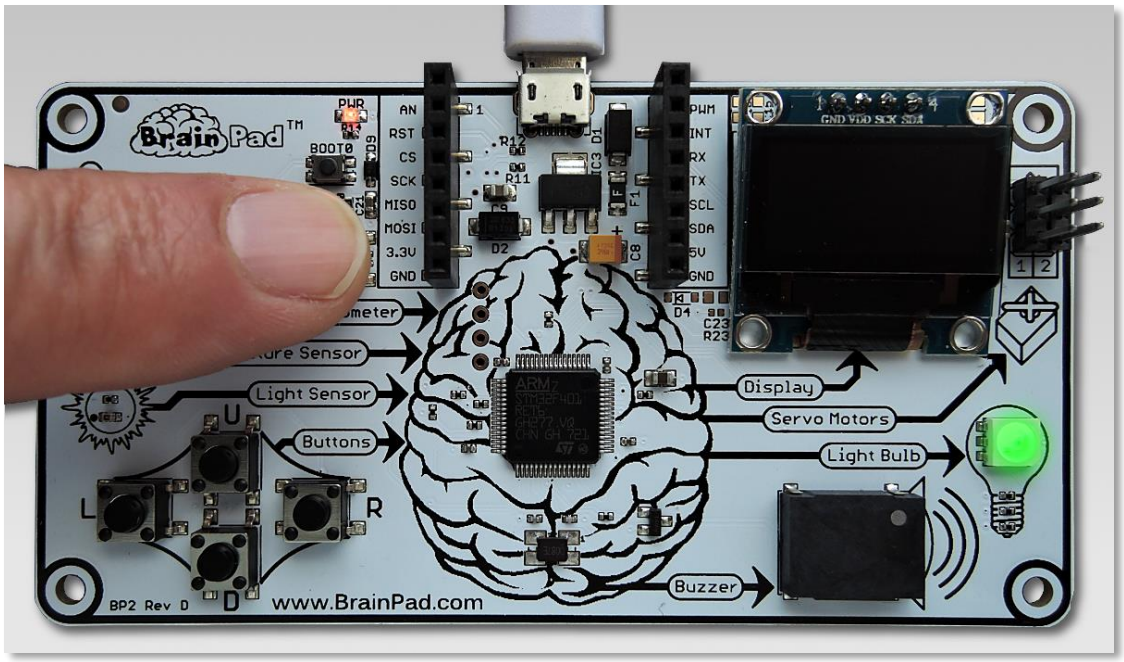
El archivo descargado aparecerá destacado. Haz click en “Copy” con el botón derecho del mouse para copiarlo.



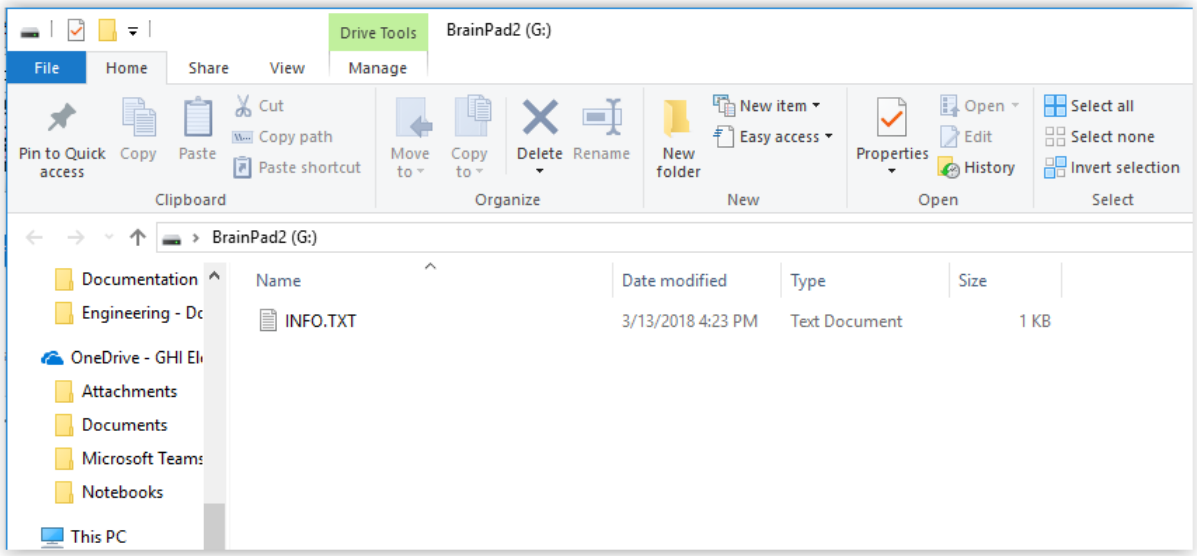
Cualquiera que sea el navegador que uses, MakeCode no borra tus archivos anteriores. Si descargas el mismo programa muchas veces, se agregará un número al nombre (por ejemplo, “brainpad-Untitled(1).uf2”). Asegúrate de copiar el más reciente. El más reciente será el que está destacado y también tendrá el número más alto entre paréntesis y la fecha y hora más reciente.

Si usas otros navegadores el comportamiento debería ser similar a lo que acabamos de describir.

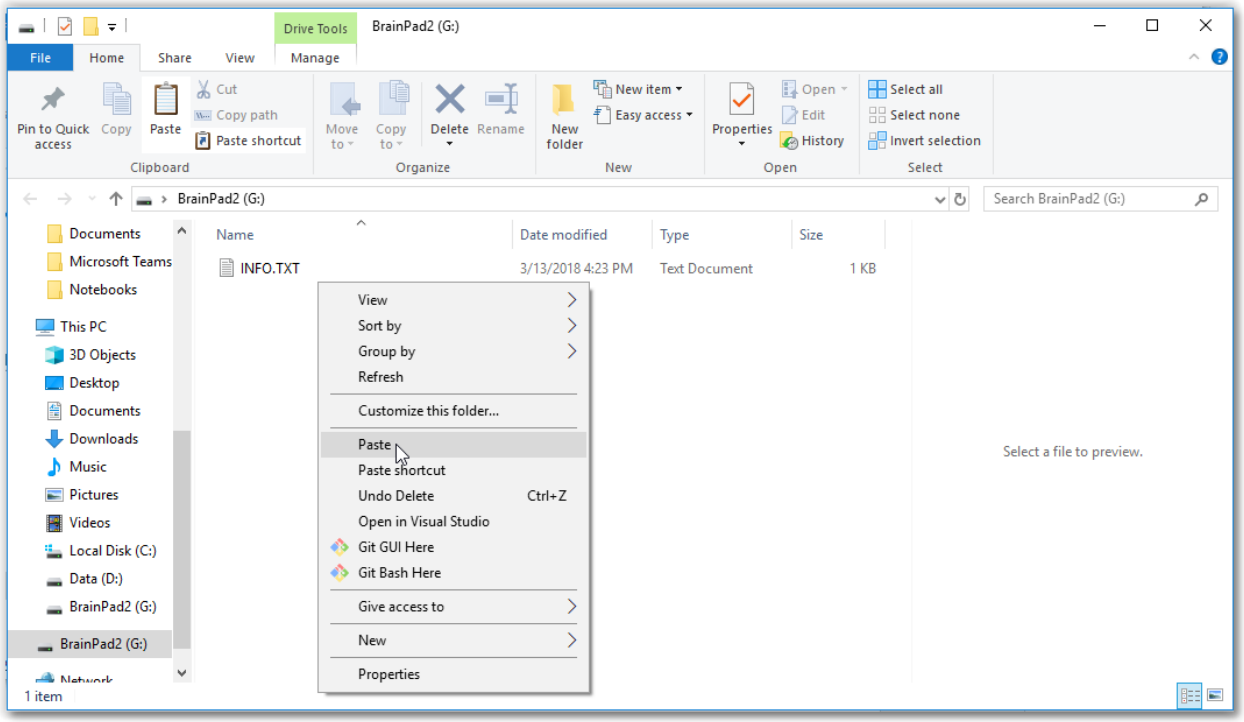
Para continuar, el BrainPad debe estar conectado a tu computador y la luz roja PWR debe estar encendida. Presiona y mantén presionado el botón “reset” por unos 3 segundos. La ampolleta (ubicada a la derecha) se encenderá de color verde.



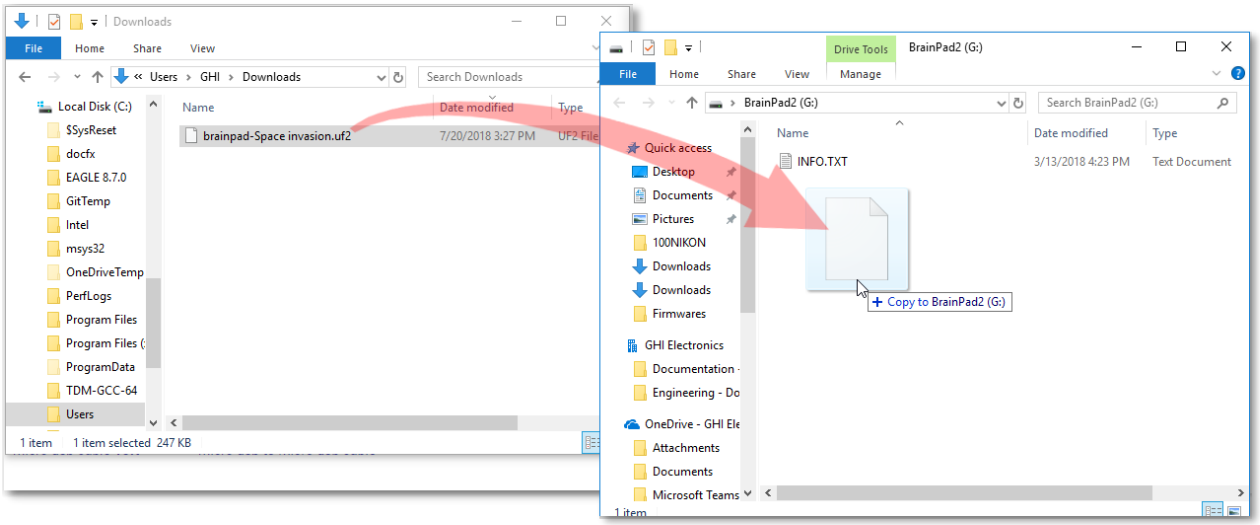
Tu computador ahora detectará un dispositivo de almacenamiento USB. Puede tomar hasta un minuto hasta que aparezca una ventana llamada “BrainPad” en la pantalla de tu computador.

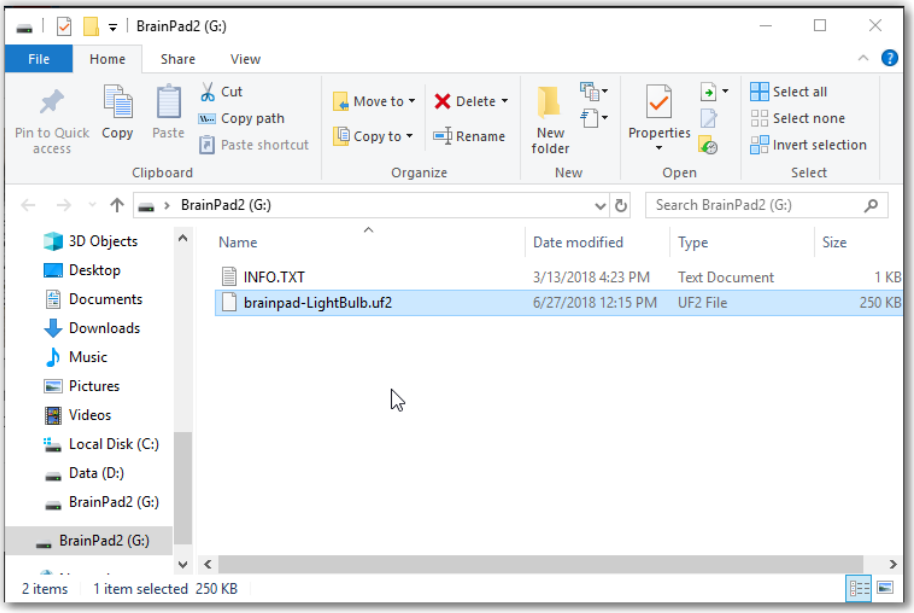


Esta ventana muestra el contenido de la memoria dentro del BrainPad. Ahora haz click en “Paste” para copiar el archivo al BrainPad.



Como método alternativo, también puedes arrastrar el archivo desde la carpeta de tu computador a la carpeta del BrainPad, como muestra la imagen siguiente:

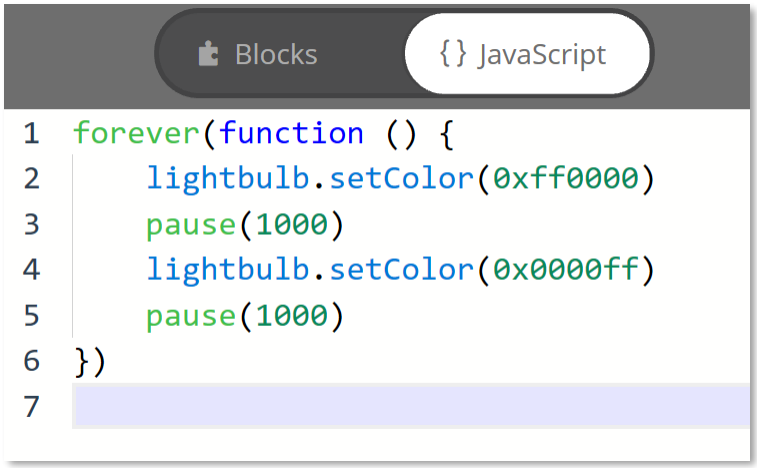




Una vez copiado el archivo, la ampolleta parpadeará por un momento y luego el BrainPad comenzará a ejecutar el programa cargado. ¡Ahora las cosas comienzan a ponerse más interesantes!

JAVASCRIPT

Los bloques son divertidos, pero a medida que tus programas crecen deberás atreverte y comenzar a escribir código. No te preocupes, puedes hacerlo cuando te sientas preparado. Una de las características maravillosas de MakeCode es la forma en que traduce los bloques a código JavaScript en formato de texto y viceversa. Vuelve al programa que hiciste y haz click en la pestaña JavaScript.



Echale una mirada al código. ¿Ves la correlación entre los bloques y el código? Algo que no se ve con claridad es el color ya que el código usa un valor hexadecimal para representarlo. ¿Pero entiendes el resto del código, cierto?

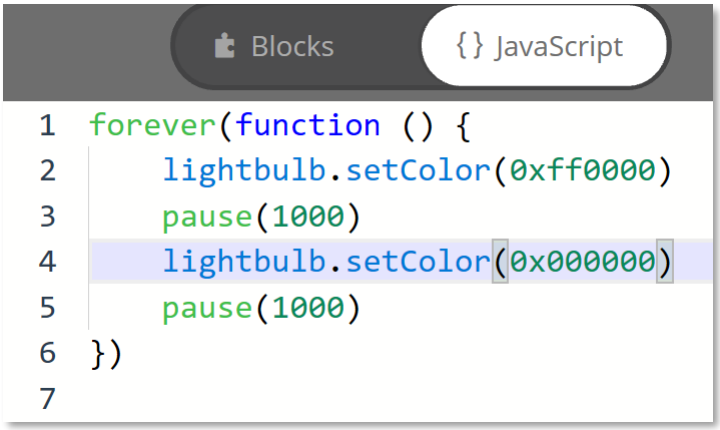
El sistema hexadecimal es un sistema numérico basado en dieciséis valores por dígito en lugar de 10 como estamos habituados. Cada dígito en un número hexadecimal va desde 0 a F, en lugar de 0 a 9 como en el sistema decimal

(los números que usamos día a día). Contando de 0 a 16 en hexadecimal se ve como sigue: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10.

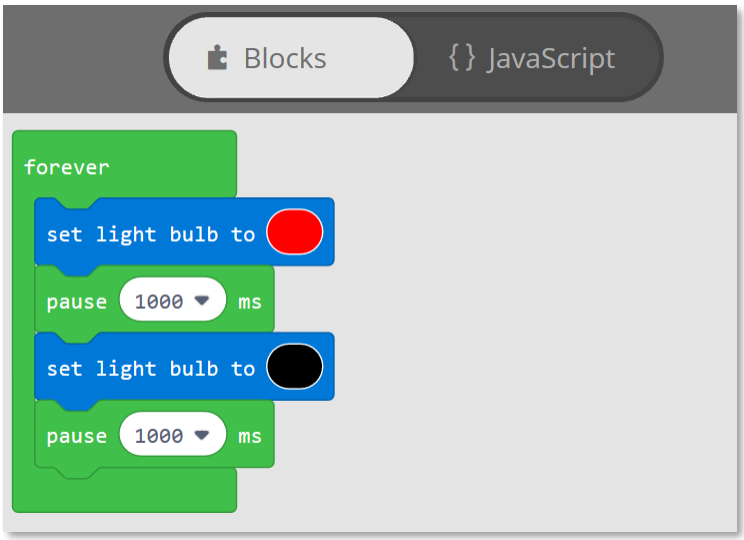
EL “0x” al comienzo de los números hexadecimales en el código se usa para hacer notar que el número está escrito en hexadecimal y así evitar confusión. Los números hexadecimales son también llamados “hex” y también son mencionados como “base 16”. Los números binarios son llamados “base 2” y los números decimales son llamados “base 10”. Si esto te resulta confuso, no te preocupes. Toma algo de tiempo familiarizarse con los distintos sistemas numéricos.

Cuando se usa un número hexadecimal para indicar un color, los primeros 2 dígitos representan la cantidad de rojo en el color. El segundo par de dígitos indica la cantidad de verde y los dos últimos representan la cantidad de azul. “00” indica que el color tiene intensidad cero, es decir, está apagado. “FF” indica la máxima intensidad del color. Por ejemplo, rojo brillante es 0xFF0000. Blanco es 0xFFFFFF.

¿Qué pasa si cambias uno de los métodos “setColor” a 0x000000? Pruébalo.



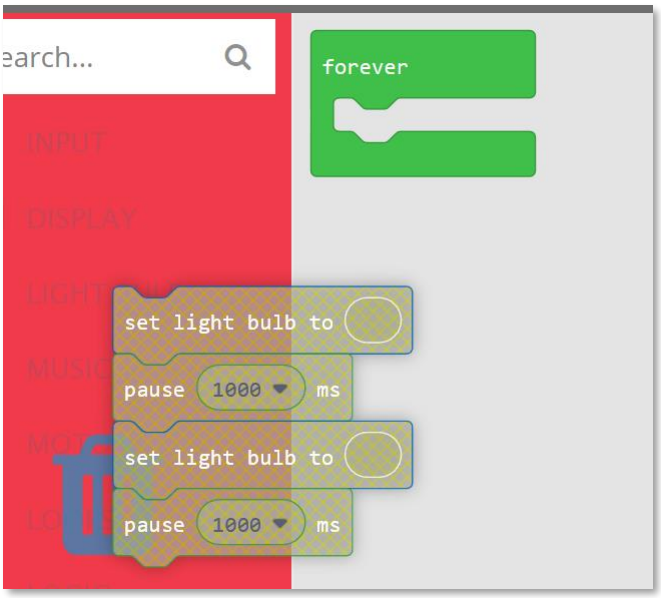
Ahora regresa a la vista de bloques para ver qué pasó.



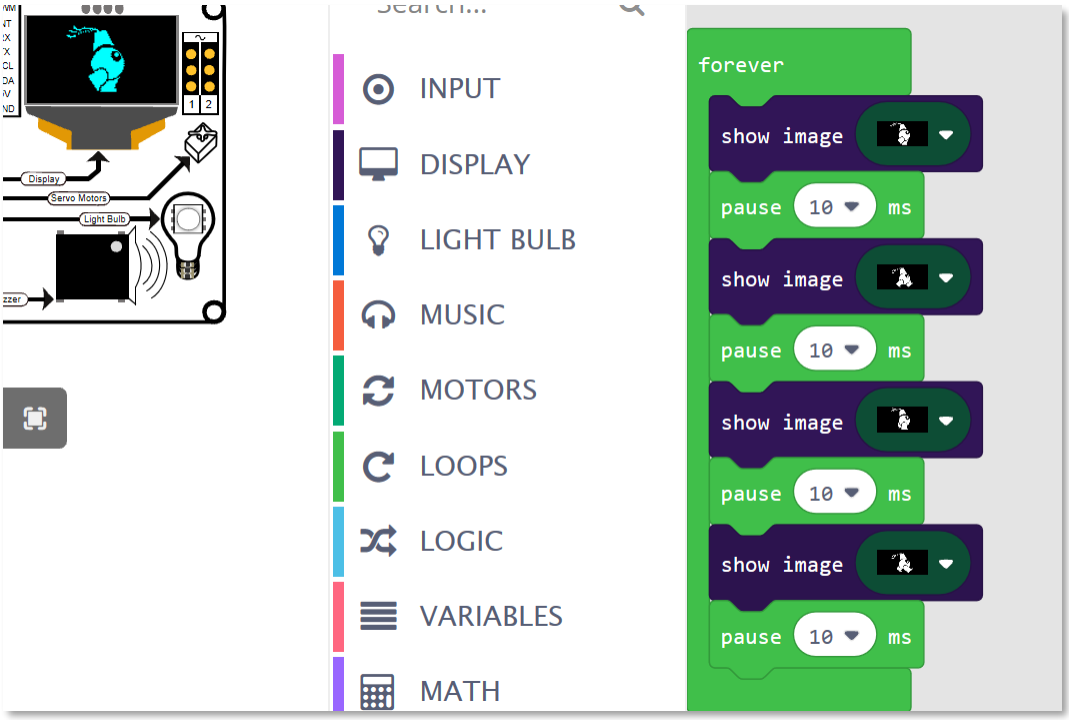
Se ve que el color azul cambió a negro. Seguro adivinaste que poner la ampolleta de color negro es lo mismo que apagarla. Puedes verlo en el simulador. La ampolleta debería ponerse roja por un segundo y luego apagarse por otro segundo.

ENTRETENCIÓN CON LA PANTALLA.

¡Si la ampolleta te pareció entretenida, espera a que probemos la pantalla! Si aún tienes los bloques del programa anterior, arrástralos de vuelta al menú para borrarlos.



Si borraste accidentalmente el bloque ‘forever’, lo puedes volver a poner arrastrándolo desde el menú LOOPS. Desde el menú DISPLAY, arrastra el bloque “show image” (desplegar imagen) y selecciona la primera de las cuatro imágenes de “walking man” (hombre caminando). Agrega las otras tres imágenes y agrega un bloque “pause” (espera) de 10 ms entre ellos. El bloque de “pause” no tiene la opción 10 ms, pero la puedes escribir en él.

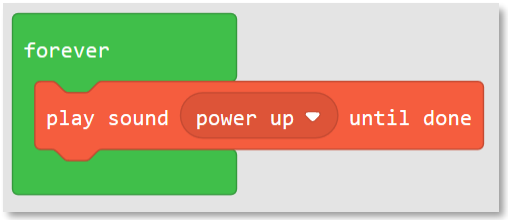


Nuestro hombre debería estar caminando en la pantalla del simulador. Ahora transfiere el programa a la placa BrainPad.

TOCANDO NOTAS MUSICALES

Bajo el menú MUSIC encontrarás las notas musicales, en notación inglesa (A, B, C, etc)…

… y también sonidos.

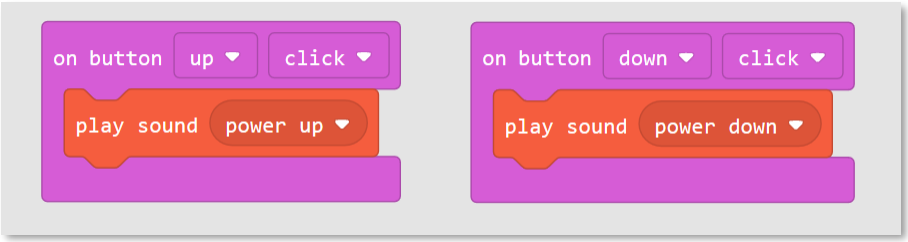


Fíjate que estamos usando el bloque “play sound until done” (tocar el sonido hasta que termine). Este bloque detiene el programa hasta que termina de sonar el sonido. En cambio, el bloque “play sound” (sin “until done”) comienza a ejecutar el siguiente bloque aunque el sonido anterior no haya terminado. Si pruebas los dos bloques te darás cuenta de la diferencia.

USANDO LOS BOTONES

Hay dos formas de leer los botones. Una de ellas es simplemente revisar si un botón está presionado y ejecutar una acción cuando lo está. El problema de este método es que debes revisar muy seguido el botón para estar seguro de que no se te pase una presión de botón. Otra forma es pedirle al sistema que revise el botón y si lo encuentra presionado (o sea, detecta un ‘evento’ de presión del botón) que ejecute alguna acción.

Ahora trataremos de leer un botón usando un ‘evento’. Queremos tocar el sonido de “power up” (encendido) cuando se presiones el botón “up” (arriba) y tocar el sonido de “power down” (apagado) cuando se presiona el botón “down” (abajo).



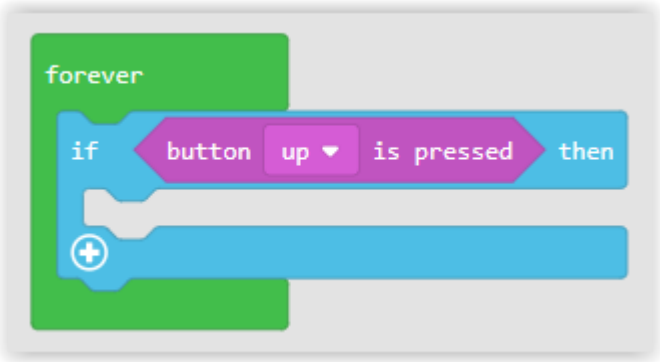
¿Te fijas que no hay bloque “forever” en este ejemplo? Aunque puedes tener un bloqe “forever”, no es necesario en este ejemplo. Esto se debe a que el programa manejará el ‘evento’ automáticamente cuando ocurra (o sea, cuando el botón sea presionado).

Este es el código JavaScript que equivale a los bloques del programa:

```
input.buttonU.onEvent(ButtonEvent.Click, function () {
    music.playSound(music.sounds(Sounds.PowerUp))
})

input.buttonD.onEvent(ButtonEvent.Click, function () {
    music.playSound(music.sounds(Sounds.PowerDown))
})
```

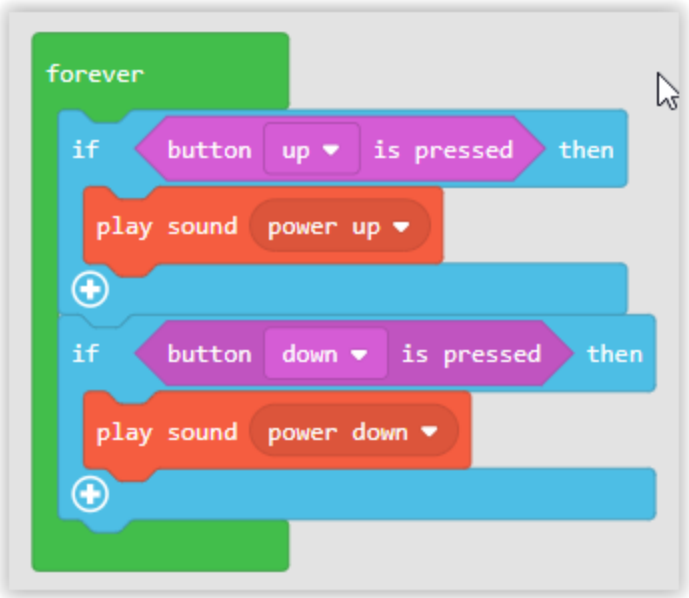
Esto funcona bien, pero volvamos a probar nuestro confiable bloque “forever”. Crearemos un programa que haga lo mismo, pero esta vez sin ‘eventos’. Para revisar si un botón fue presionado necesitarás una sentencia “if” (que significa ‘si’ en condicional). Encontrarás el “if” en el menú LOGIC. Arrástralo dentro del bloque “forever”.

El bloque “if” revisa el estado de algo que puede ser “true” (verdadero) o “false” (falso), pero nosotros queremos revisar el estado del botón. Así que arrastra el bloque “button… is pressed” (‘el botón … fue presionado’) desde el menú INPUT al bloque “if … then” tal como se muestra en la imagen siguiente. Fíjate que este no es el bloque “on button” que usamos anteriormente.



Estos bloques revisarán el botón “up” una y otra vez, permanentemente. Si el botón está presionado en el momento en que se revisa, el programa ejecutará los bloques que hay dentro del bloque “if”.

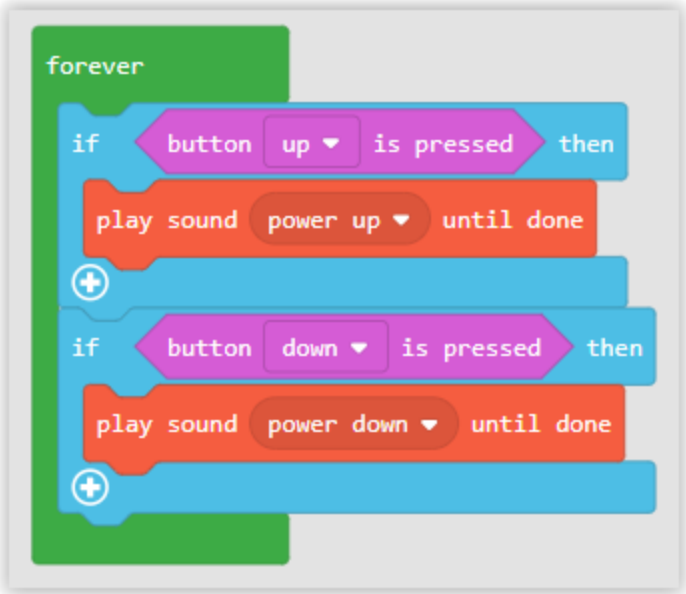
Ahora agreguemos más bloques para construir el programa que se muestra a continuación:



Cuendo ejecutes este programa te darás cuenta que el sonido no suena correctamentehasta que sueltas el botón. La razón de esto es que cada vez que el Brain Pad detecta que se está presionando un botón comienza a tocar un nuevo sonido, sin importar si el anterior terminó. El BrainPad comenzará a tocar un nuevo sonido varias veces por segundo mientras el botón se mantenga presionado.

Hay dos formas de resolver esto. Una manera es reemplazar los bloques “button … is pressed” (‘el botón … está presionado’) con bloques “button … was pressed” (‘el botón … fue presionado’ o ‘estuvo presionado’). El bloque “button … was pressed” será verdadero si el botón fue presionado al menos una vez desde que se revisó la última vez. Incluso si el botón fue presionado muchas veces, se tomará como si solo se presionó una vez. Si se presiona el botón “up” o el botón “down” mientras se está tocando un sonido lo va a interrumpir, pero el sonido no se repetira´si se mantiene el botón presionado.

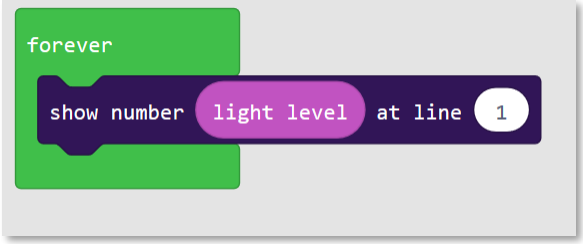
La otra forma es reemplazar los bloques “play sound…” (‘tocar el sonido…’) por bloques “play sound … until done” (‘tocar el sonido … hasta el final’). Al hacer esto el programa se detiene mientras está tocando un sonido, por lo que deja de revisar si se presionan los botones. En otras palabras, usar los bloques “play sound… until done” hará que el sonido no pueda ser interrumpido. Los bloques se muestran a continuación.



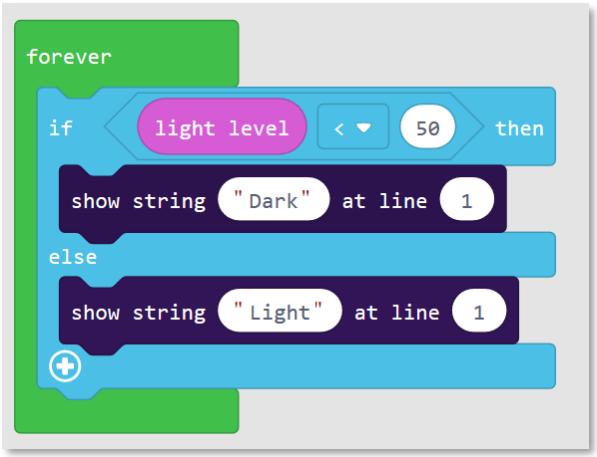
¿ESTÁ OSCURO AQUÍ?

Así como podemos leer si un botón está presionado también podemos leer la intensidad de la luz para hacer algo, o podemos agregar un evento que se active cuando hay mucha luz o está oscuro. Sin embargo, leer la luz es un poco diferente de leer un botón.

Un botón solo puede estar presionado o no presionado. Así que cuando preguntamos si el botón está presionado la respuesta solo puede ser verdadero o falso. En el caso del sensor de luz, éste entrega el nivel de luz como un valor. En programación esto se conoce como un valor análogo. Vamos a leer la intensidad de luz y desplegarla en la pantalla. Queremos hacer esto en un ciclo sin fin (“forever”), de modo de leer constantemente el nivel de luz y actualizar la pantalla.



Como ya eres un experto en usar la sentencia “if”, ¿puedes escribir “dark” (‘oscuro’) o “bright” (‘iluminado’) en la pantalla dependiendo de la luz detectada por el sensor? En realidad, hacerlo es un poco complejo, pero te lo voy a explicar. El bloque “if” necesita saber si algo es verdadero o falso, por ejemplo: ¿el botón está presionado? En cambio, el nivel de luz no es verdadero o falso, sino que puede tener diferentes valores. En otras palabras, no tendría sentido una sentencia “if ligth…”. Lo que tenemos que hacer es preguntar si la luz es más oscura o intensa que algún nivel. Así que vamos al menú LOGIC y agreguemos una comparación. Tu sentencia “if” tendrá ahora la forma que se muestra a continuación, con una sentencia “else” agregada. La sentencia “else” indica lo que debe hacer el programa cuando no se cumple la condición del “if”.

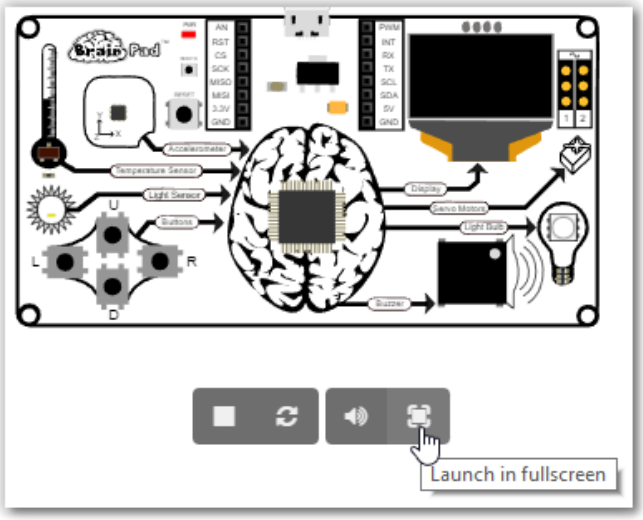


¿Qué significa esto? Si el nivel de luz es menos de 50, muestra la palabra “Dark” (‘oscuro’) en la línea 1. Y si no se cumple esa condición, o sea, si el nivel de luz no es menor que 50, muestra la palbra “Light” (‘luz’) en la línea 1.

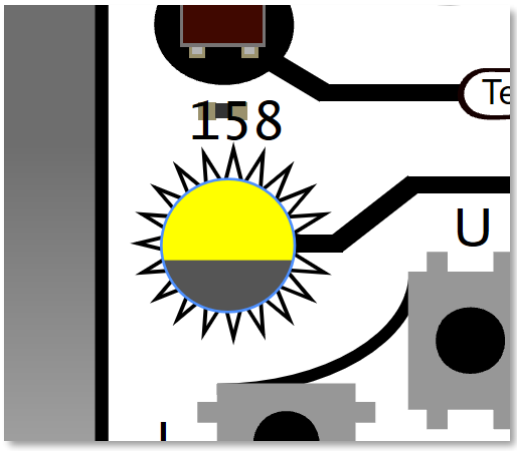
Echale una mirada a la pestaña JavaScript para ver cómo queda el código.

```
forever(function () {
    if (input.lightLevel() < 50) {
        display.showString("Dark", 1)
    } else {
        display.showString("Light", 1)
    }
})
```

El simulador también funciona con el sensor de luz. Cuando usas el sensor en tu programa, el simulador muestra una opción para modificar el nivel de luz. Para probar esto, primero agrandemos el simulador haciendo click en el botón de pantalla completa ubicado debajo del simulador.



El nivel del sensor de luz se puede cambiar moviendo la línea de luz con el mouse (click y mover el mouse). Esto es como si taparas el sensor de luz con tu mano. Cuando muevas el nivel arriba y abajo la pantalla debería mostrar “dark” y “light”.



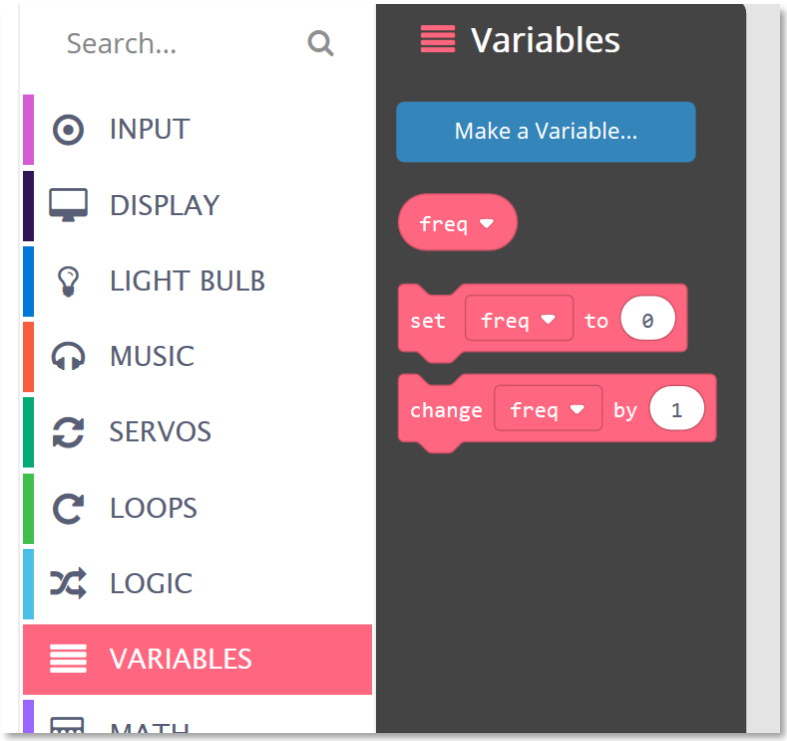
PRUEBA DE AUDICIÓN

El oído humano puede escuchar sonidos de frecuencias tan bajas como 20 Hertz (ciclos por segundo) y tan altas como 20.000 Hertz. Los perros pueden oír frecuencias más altas que los humanos, llegando hasta los 44.000 Hertz. Por eso no podemos oír los silbatos de alta frecuencia para perros, pero los perros los oyen perfectamente.

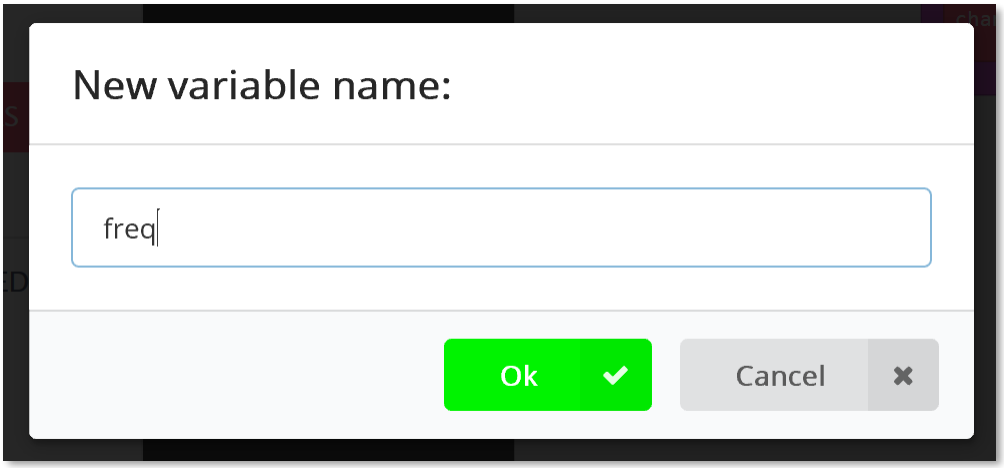
A medida que envejecemos vamos perdiendo la capacidad de escuchar frecuencias altas. Y la pérdida de audición causada por el ruido afecta más la capacidad de escuchar las altas frecuencias que las bajas. ¿Quieres saber cuál es la frecuencia más alta que puedes escuchar? ¡Averiguémoslo!

Para eso tenemos que escribir un programa que usa una “variable”. Una variable es simplemente una posición de memoria que almacena información. En este caso nuestra variable almacenará un número que representa la frecuencia que vamos ha hacer sonar en el Buzzer.

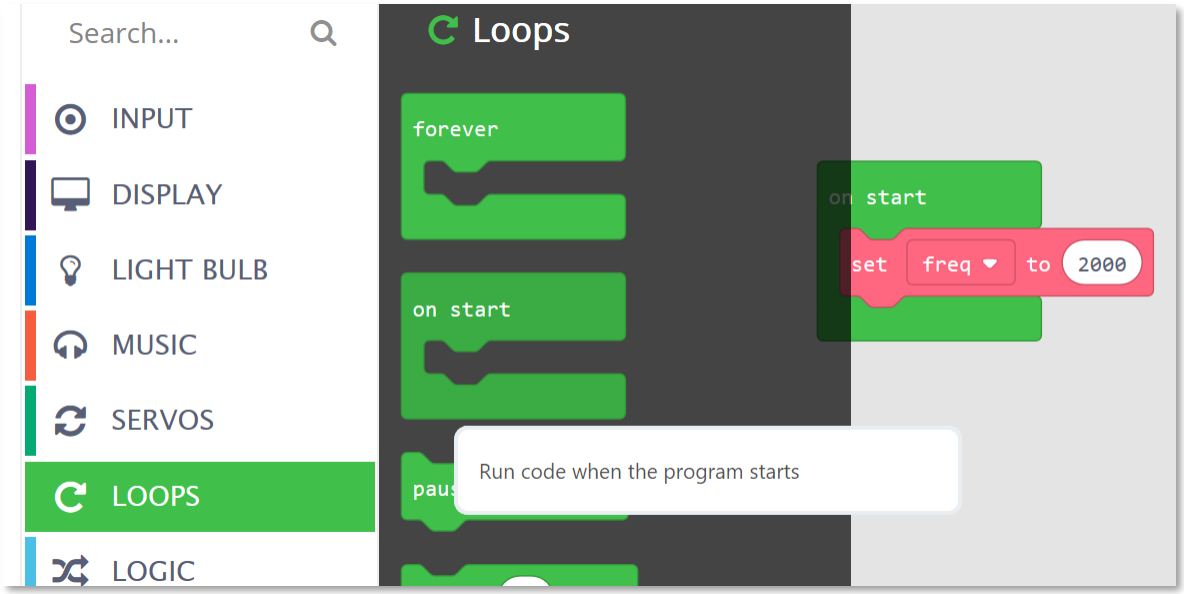
Anda al menú VARIABLES y haz click en el botón “Make a Variable…”.



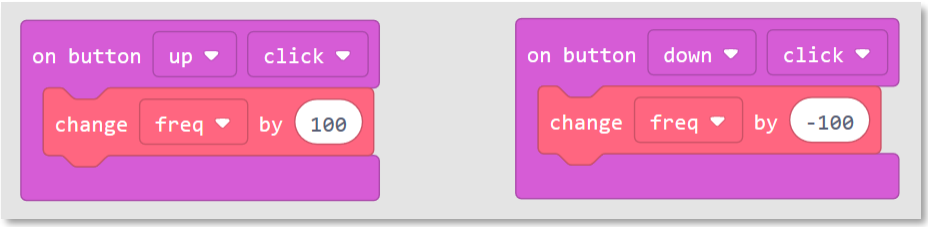
Ahora dale un nombre a tu variable. Yo elegí el nombre “freq” (abreviatura de ‘frecuencia’ en inglés). Esta variable almacenará la frecuencia que quiero tocar en el Buzzer.



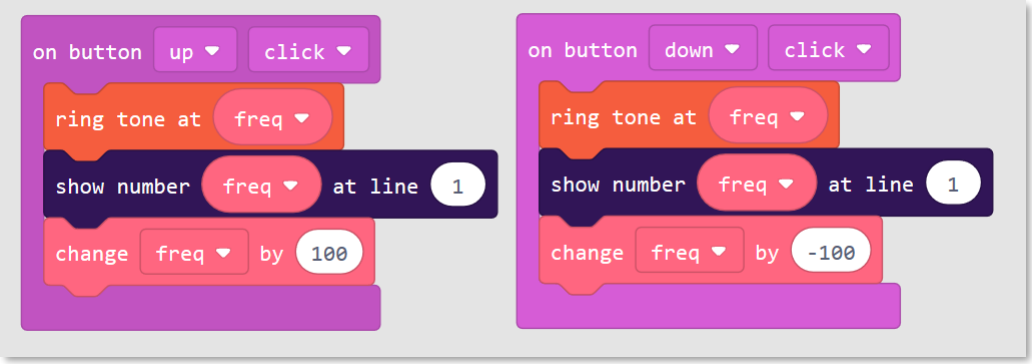
Ahora anda a LOOPS y agrega un bloque “on start”. Todo lo que agregues a este bloque se ejecutará cuando el programa se comience a ejecutar. Aquí queremos asignar el valor 2000 a la variable. Como es una frecuencia que todos pueden escuchar, será un buen punto de partida. El bloque “set” (‘asignar’) está en el menú VARIABLES.



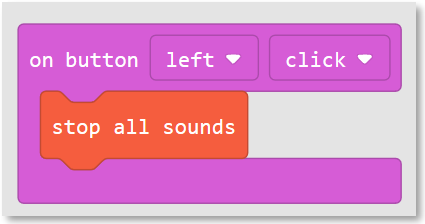
Ahora necesitamos generar un sonido a esa frecuencia y también vamos a hacer que la frecuencia aumente cuando presionemos el botón “up” y disminuya cuando presionemos el botón “down”. Para eso necesitamos un bloque “change” (‘cambiar’) ubicado en el menú VARIABLES. Ponlo para que la frecuencia cambie en 100 cuando se presiona el botón “up” y cambie en -100 cuando se presione el botón “down”.



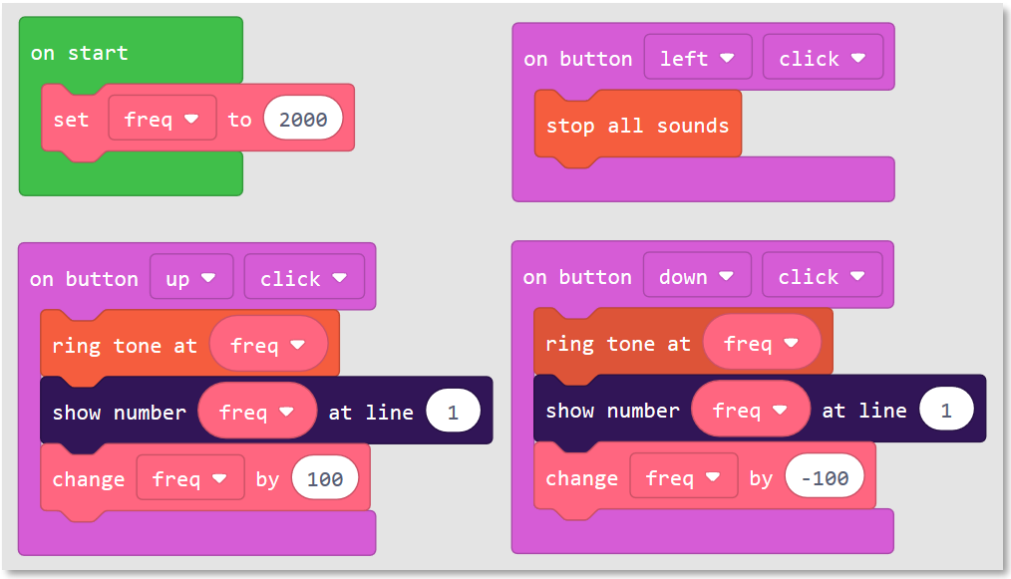
Ahora que tenemos la variable cambiando a nuestro gusto, podemos generar el sonido con el bloque “ring tone at” (‘sonido a …’) y también mostraremos la frecuencia en la pantalla con el bloque “show number”.



¡Importante! El sonido puede ser muy molesto así que agregaremos código para detenerlo cuando se presione el botón “left” (‘hacia la izquierda’).



Aquí está el programa complete:



MOTORES SERVO

Un motor servo es básicamente un motor que tiene un pequeño circuito interno que permite que lo controles usando pulsos eléctricos.

Lo servos tiene un conector de 3 pines que se conecta directamente al BrainPad. Desafortunadamente los distintos servos usan diferentes colores para los cables. Para identificar la forma correcta de conectarlo, ignora el cable del medio y fíjate en los cables de los lados. El cable de color más claro es el que debe ir cerca del símbolo ~ impreso en el Brain Pad. Este cable se llama ‘cable de señal’.

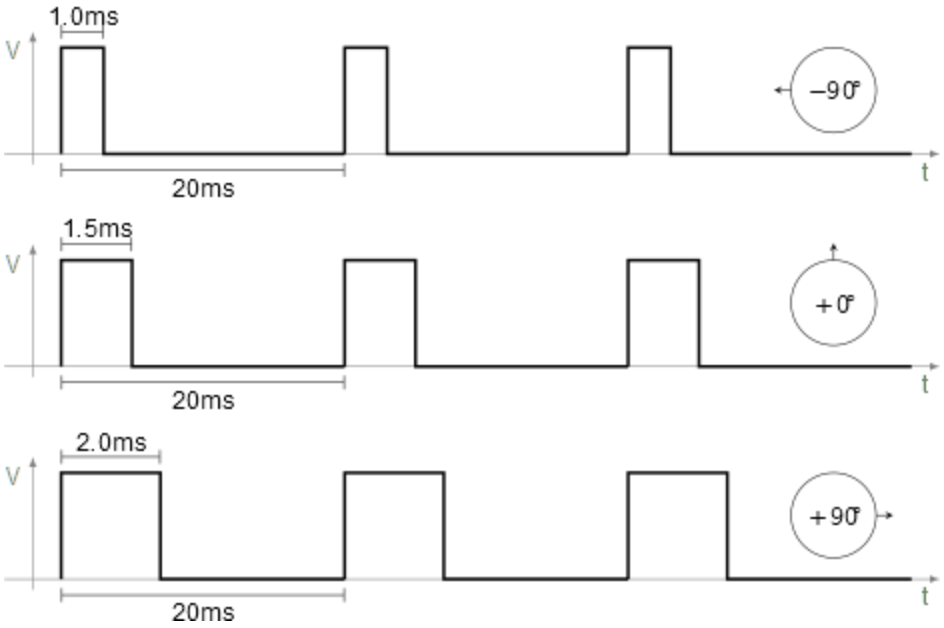
Los motores servo (o simplemente servos) se utilizan a menudo en modelos radio controlados y se pueden encontrar en Internet o en tiendas de hobby. Estos servos reciben energía desde el BrainPad, el que a su vez saca energía desde tu PC o de una batería. Por lo tanto te recomendamos usar los servos más pequeños, llamados micro servos. Algunos servos grandes consumen demasiada energía y no trabajan adecuadamente (a menos que se les provea energía adicional externa).

Existen servos de tipo contínuo o posicional. Aunque se ven iguales, un servo posicional solo gira a una posición dada y permanece allí hasta que le indiques que se mueva a otra posición. Un servo contínuo girará permanentemente en una dirección hasta que le indiques que gire en la dirección contraria o que se detenga.

Si giras a mano un servo posicional no energizado, se moverá aproximadamente media vuelta en cualquier dirección y se detendrá. Si giras un servo contínuo no se detendrá. Debido al mecanismo interno de los servo debes aplicar poca fuerza para hacerlos girar.

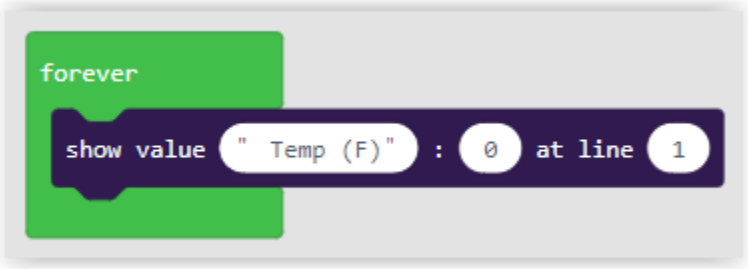
MOTORES SERVO POSICIONALES

Como ya se mencionó, un motor servo posicional gira hasta la posición (o ángulo) que le indiques y permanece en esa posición hasta que le indiques que gire hasta otra. Si tratas de mover un motor servo a mano y se detiene repentinamente, entonces es un servo posicional. Un pulso enviado por el BrainPad le indicará a qué ángulo o posición moverse.

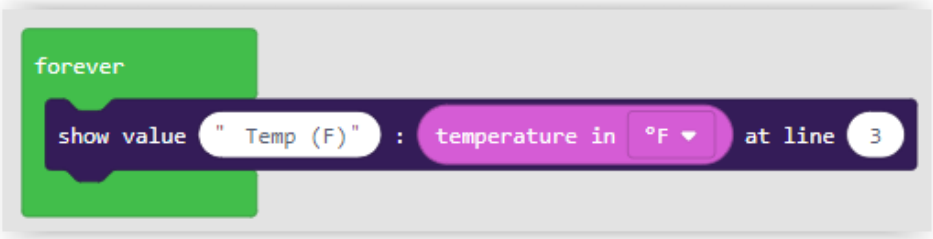


Hay muchas formas de usar un servo posicional. En este ejemplo crearemos un termómetro de aguja. El código que usaremos es simple y ni siquiera usa una variable.

En un bloque “forever” agrega un bloque “show value” (‘mostrar valor’) disponible dentro de la sección DISPLAY. En el primer óvalo del bloque “show value” escribe “Temp (F)” tal como se muestra a continuación:



Del menú INPUT selecciona el bloque “temperatura in °C” y arrástralo en el segundo óvalo. Nuestro termómetro mostrará la temperatura en Fahrenheit así que cambia el “°C” a “°F”. Ahora cambia el tercer óvalo a “3”. Esto mostrará la temperatura en la tercera línea de la pantalla del BrainPad. Tus bloques deberían verse tal como se muestra a continuación:

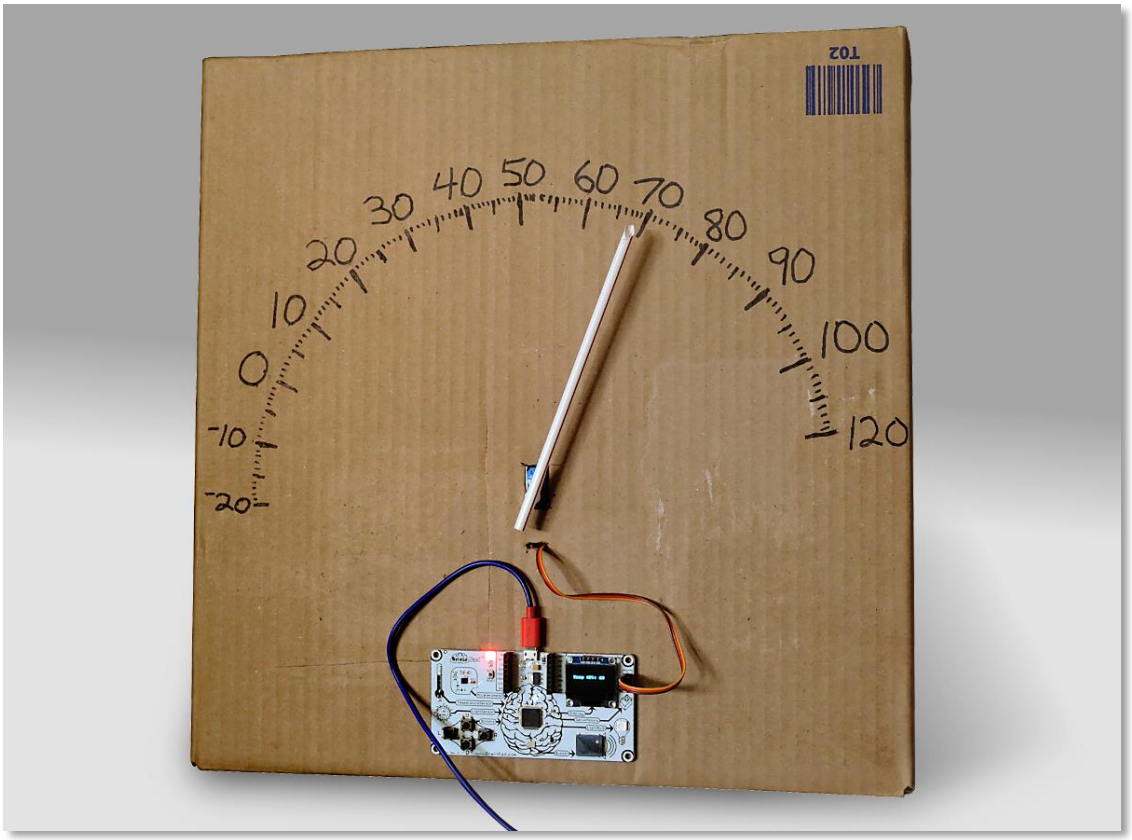


Hasta aquí hemos creado un termómetro que muestra la temperatura en la pantalla del BrainPad. Ahora debemos convertir la temperatura en un ángulo y enviarla al motor servo para mover la aguja en nuestro termómetro. Hagamos que nuestro termómetro se mueva entre -20° Fahrenheit y 120° Fahrenheit.

Debemos convertir la temperatura (-20 a 120) en un ángulo entre 0 y 180. Si no eres un matemático no te preocupes. El software MakeCode tiene un bloque “map” que hará las matemáticas por ti. Lo encontrarás en el menú MATH. Completemos nuestro programa agregando una línea más:

Notarás que el ángulo va desde 180 en el punto más bajo a 0 en el punto más alto de nuestro bloque “map”. ¿Nos habrá quedado al revés? La razón es que cuando el motor servo es fijado en un ángulo de 0 grados, gira hacia la derecha y cuando es fijado en un ángulo de 180 grados gira hacia la izquierda. Si hubiéramos asociado la temperatura de -20° Fahrenheit al ángulo de 0 grados el termómetro se habría leído al revés.

Aquí se ve el termómetro terminado, hecho con una caja y una aguja de plástico pegada al motor servo.

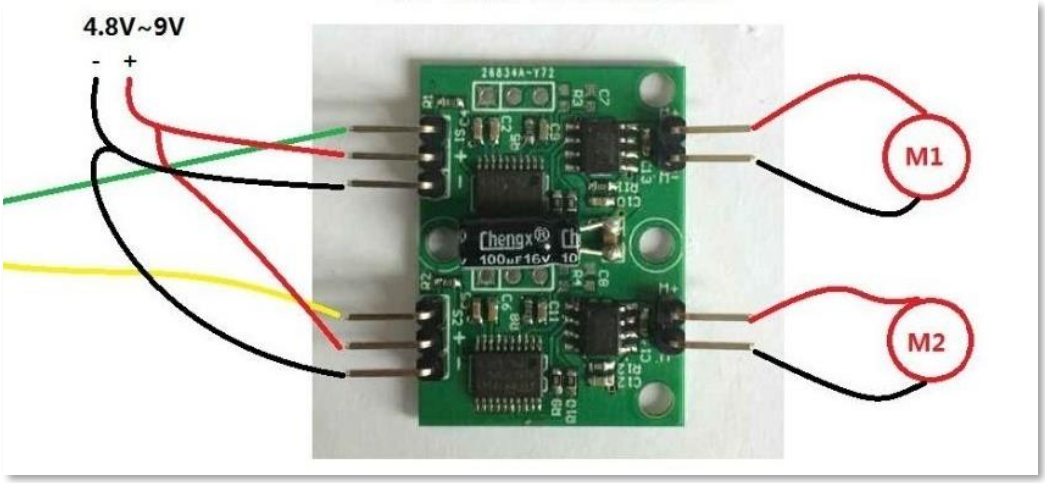


MOTORES SERVO CONTÍNUOS

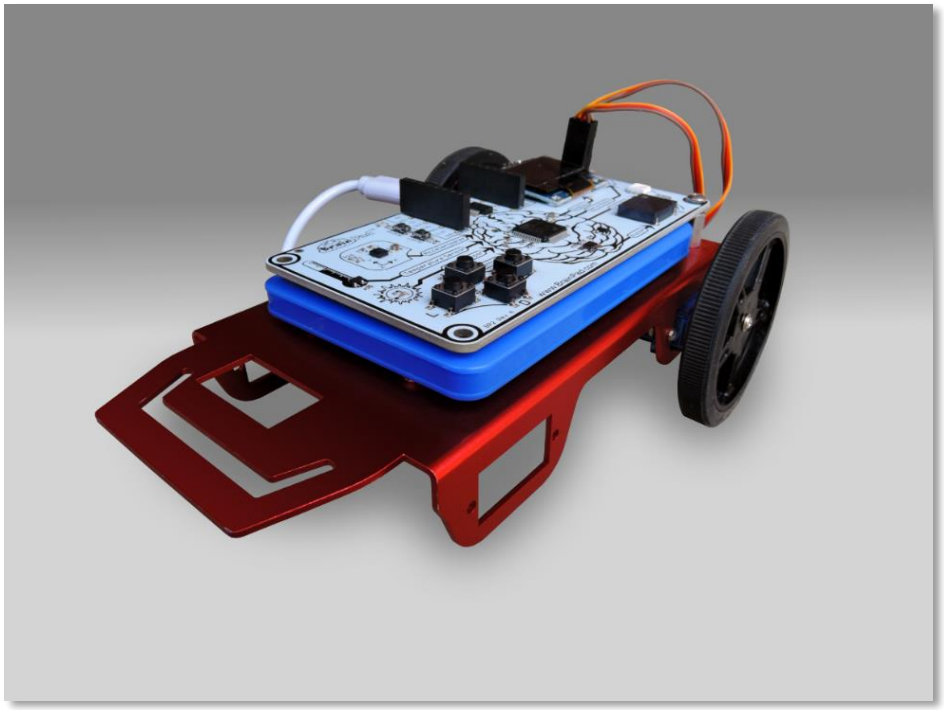
El otro tipo de motor servo es el contínuo, que, como su nombre lo indica, puede girar permanentemente. EN este caso el pulso fija la velocidad y dirección de giro del motor. El BrainPad tiene conexiones para 2 motores. Controlando la dirección y velocidad de dos servos rotacionales puedes fácilmente mover un robot. Yo busqué en Amazon “micro servos contínuos con ruedas” y encontré exactamente lo que necesitaba.

Podría haber montado el mecanismo en una caja de cartón, pero encontré este chasis fabricado por FEETECH.

Aquí las cosas se me complicaron porque el robot venía con ruedas y motores, que parecían motores servo pero eran motores normales, no servo. Lo supe inmediatamente porque tenían 2 cables en lugar de 3. El kit traía también un pequeño circuito que descubrí que sirve para convertir los motores normales en motores servo.

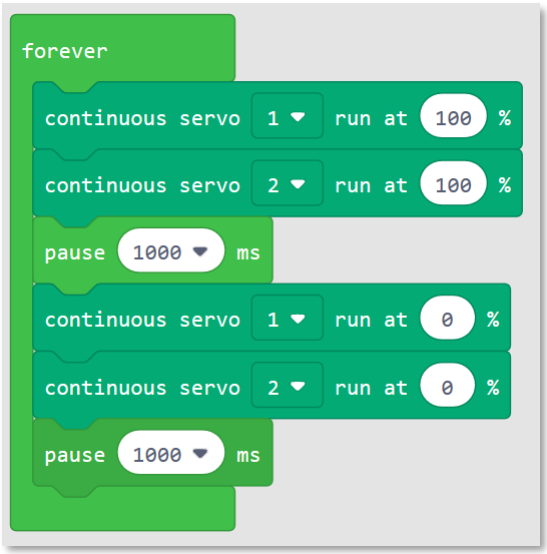


Eso serviría, pero es muy complicado así que dejé de lado el circuito y los motores normales y decidí seguir adelante con motores servo contínuos. Mi BrainPad calza perfectamente en el chasis, como si hubiera sido hecho para eso. Después encontré una batería plana para cargar celulares que calza justo debajo del BrainPad.

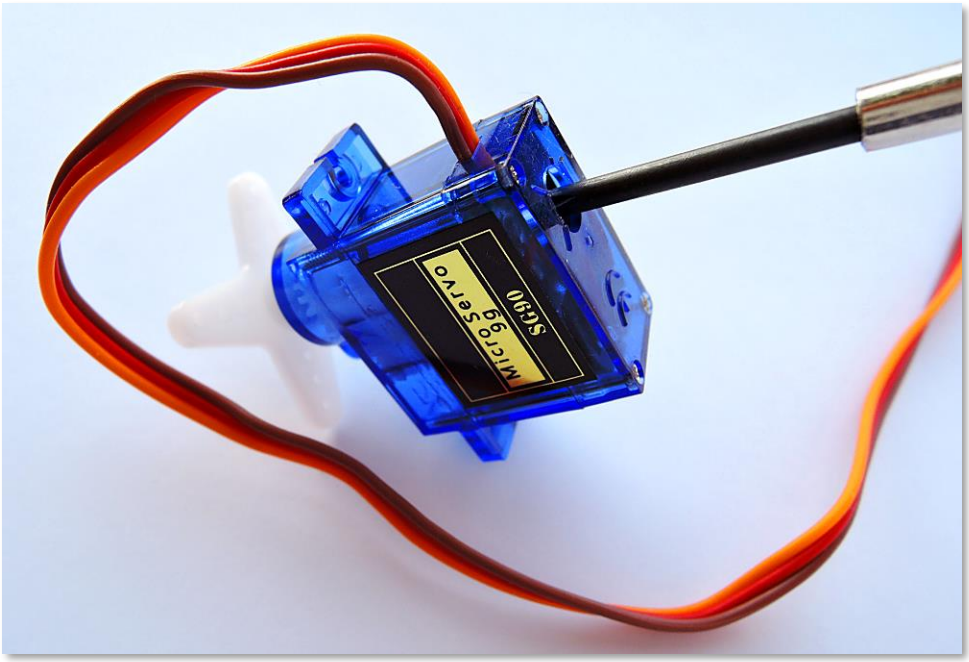


Este robot está listo para bailar. Para que avance, ambos motores tienen que girar hacia adelante. Pero como uno de los motores apunta a la derecha y el otro a la izquierda, van a tener que girar de forma inversa para que las

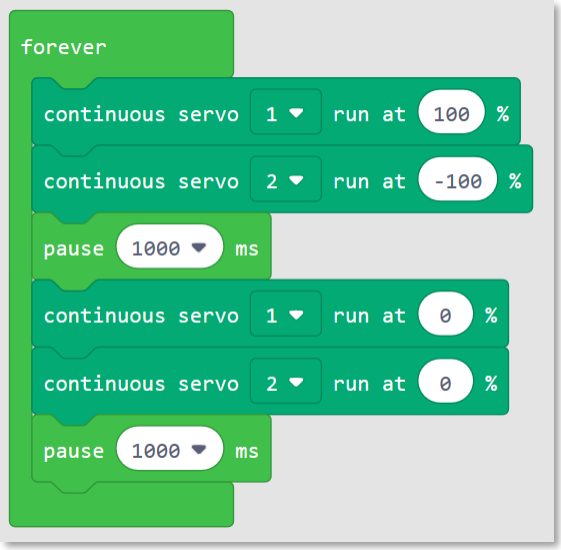
ruedas avancen en la misma dirección. Esto puede parecer confuso, pero cuando lo pruebes vas a entender cómo funciona. El siguiente código moverá ambos servos hacia adelante a máxima velocidad por 1 segundo, detendrá los motores por un segundo y luego repetirá el ciclo.



Si los motores siguen girando cuando la velocidad se fija en cero tendremos que calibrarlos. La mayoría de los motores servo tienen un ajuste en su interior- Para ajustarlos vas a necesitar un pequeño desatornillador.



Volvamos al programa. ¿Ves que uno de los motores se está moviendo en la dirección contraria? Cambia uno de los servos para que vaya en reversa a máxima velocidad.



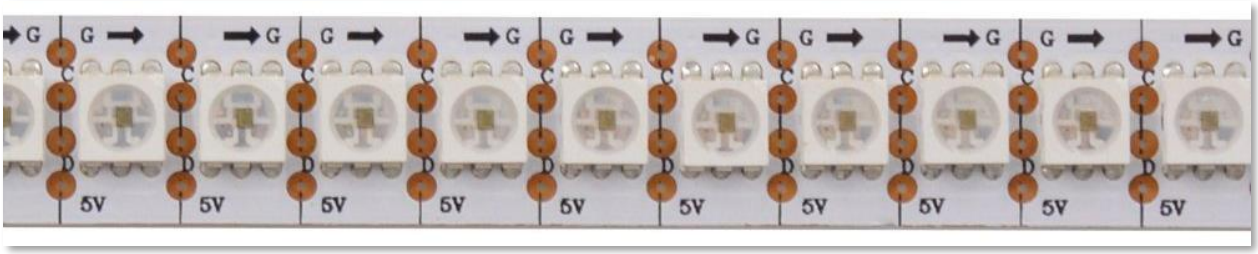
Este trabajo fue el comienzo de nuestro proyecto “Sun seeker” (‘buscador de sol’). Puedes encontrar el proyecto aquí: https://docs.brainpad.com/go-beyond/projects/sun-seeker.html Asegúrate de ver el video. El robot usa el sensor de luz para detectar si está al sol o a la sombra. Si está a la sombra, avanzará hasta encontrar un lugar soleado donde detenerse.

LUCES DE NAVIDAD

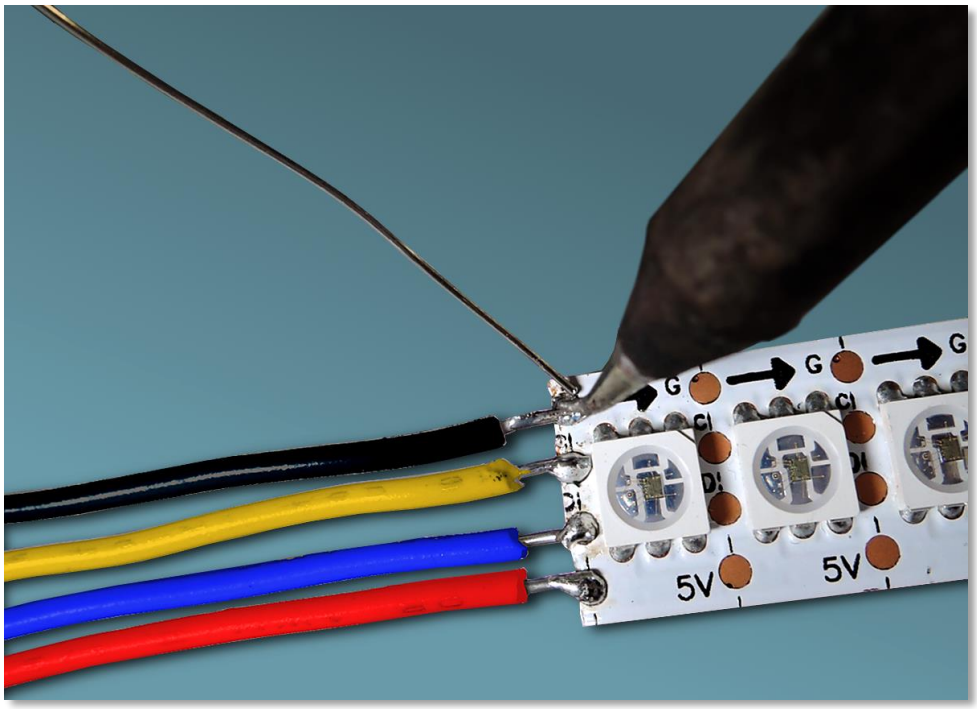
Hay muchas cosas que puedes hacer con los dispositivos incluídos en el BrainPad, pero en algún momento vas a querer conectar el BrainPad a componentes externos. Aunque eso está fuera de los objetivos de este libro, aquí hay una rápida introducción que te dará una idea de lo que se puede hacer.

Para este proyecto usaremos cintas de LED (Light Emitting Diodes o Diodos Emisores de Luz). Esas contas consisten de una tira de luces que pueden ser controladas individualmente. Se pueden fijar con prácticamente cualquier combinación de color e intensidad usando solamente 2 cables. Esos cables envían pulsos a los LED para indicar lo que queremos hacer.

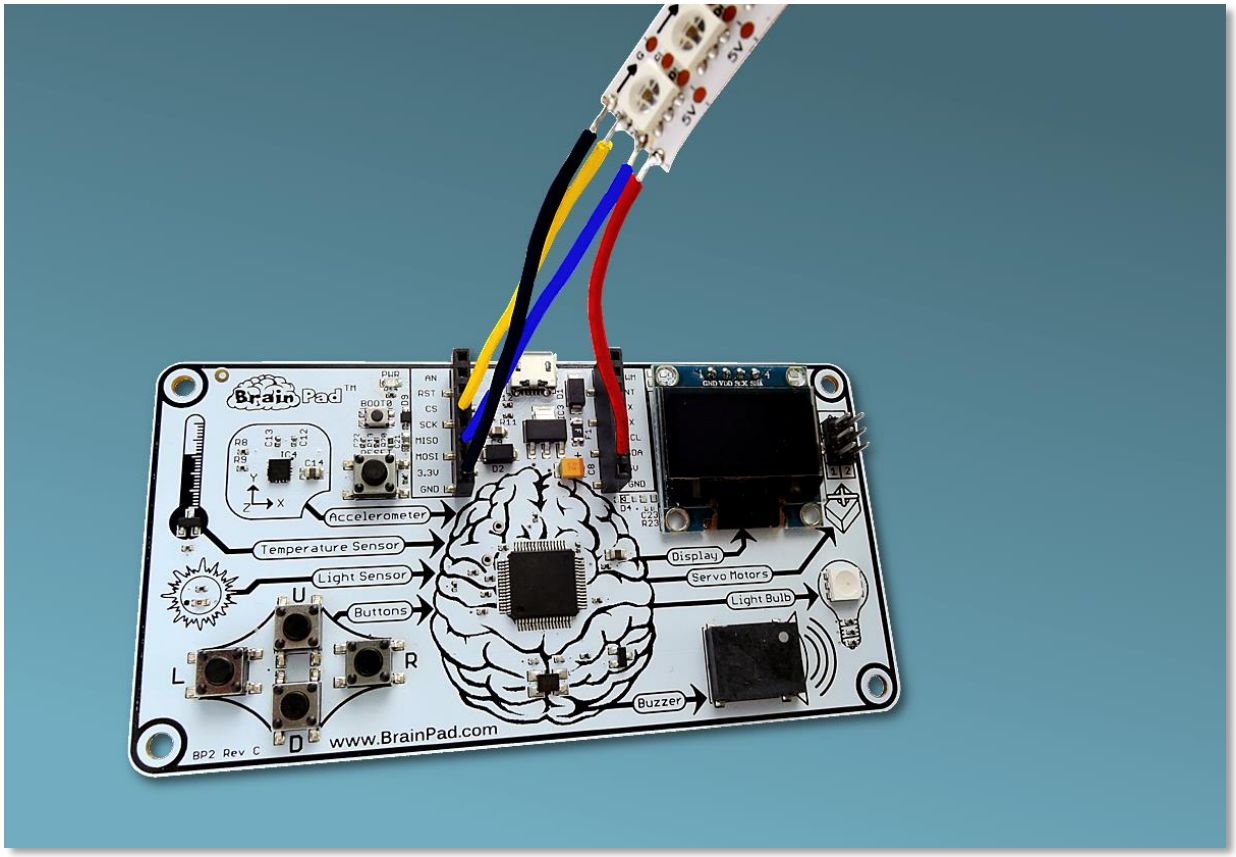
Necesitarás cintas con el controlador APA102. El controlador usa un bus SPI, que está disponible en el BrainPad. Hay cintas que se energizan con diferentes voltajes. Nosotros necesitaremos cintas de 5V (5 volts) ya que tenemos acceso a 5 V.



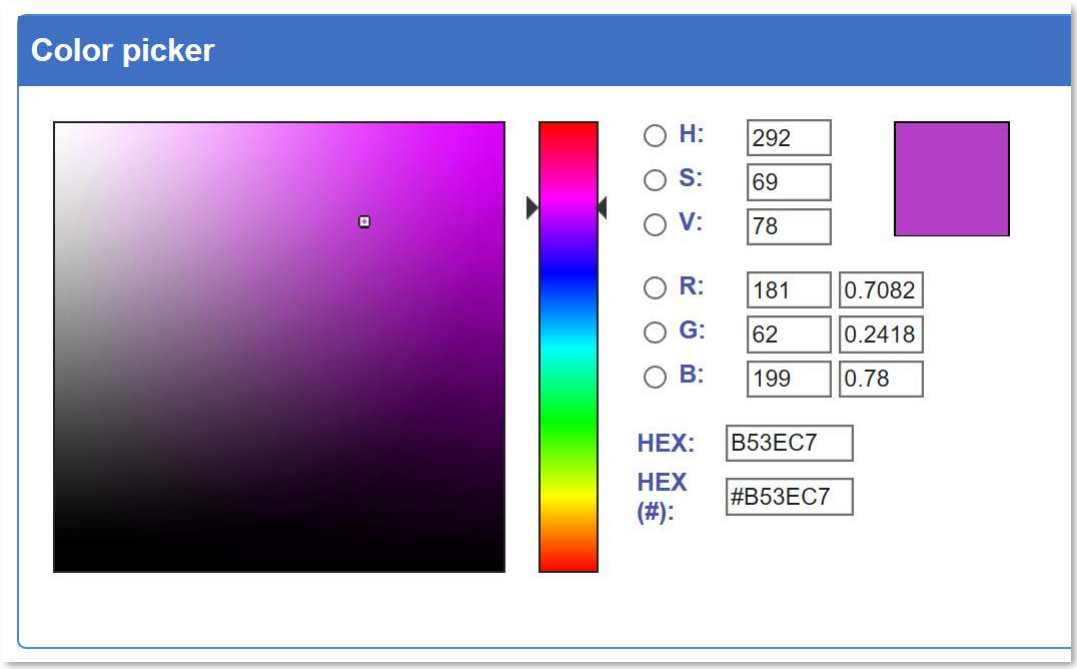
Las cintas se pueden cortar de cualquier tamaño que se desee. Para evitar consumir mucha energía cortaré para tener 25 LED. Ahora tenemos que soldar algunos cables. La flecha en la cinta apunta al lado en que tienes que soldar. Los cuartro cables son: G-Ground (tierra), C-Clock (reloj), D-Data (datos) y 5V.


G
G
G
5V
5V

Ground y 5V están claramente marcados en el conector de expansión del BrainPad. El cable C-Clock va conectado a SCK en el BrainPad. D-Data va conectado a MOSI en el BrainPad.



Cada LED tiene 3 LED internos, de color rojo, verde y azul. Controlando los niveles de esos colores básicos puedes crear cualquier color que quieras. Hay varias herramientas en Internet para ayudar a elegir colores. Un buen ejemplo se encuentra en http://www.rgbtool.com. Tienes que fijarte en los valores de R (red-rojo), G (green-verde) y B (blue-azul).



Los comandos son enviados a los LEDs usando el bus SPI del BrainPad. Para comenzar a escribir una secuencia de colores se debe enviar una trama de inicio primero. La trama de inicio está formada por 4 ceros.

Después de la trama de inisio, cada LED requiere 4 números para controlarlo. El primer número fija el brillo de toda la cadena de LEDs. No queremos complicarnos, así que siempre enviaremos el primer número con el valor 255, que corresponde al brillo máximo.

Después del primer número, los otros tres tienen valores que van de 0 a 255 cada uno, para controlar la intensidad de azul, verde y rojo en cada LED. Para poner un led de color azul puro, envías 255,0,0. Para apagar un LED, tienes que enviar 0,0,0. Para hacer que un LED se ponga blanco brillante, envías 255,255,255.

Los LED en la cadena van en orden. El primer color es enviado al primer LED, el segundo colos al segundo LED y así sucesivamente. Los LEDs se irán iluminando en secuencia hasta que envíes una nueva trama de inicio (cuatro ceros). Después de una trama de inicio la secuencia comienza de nuevo con el primer LED.

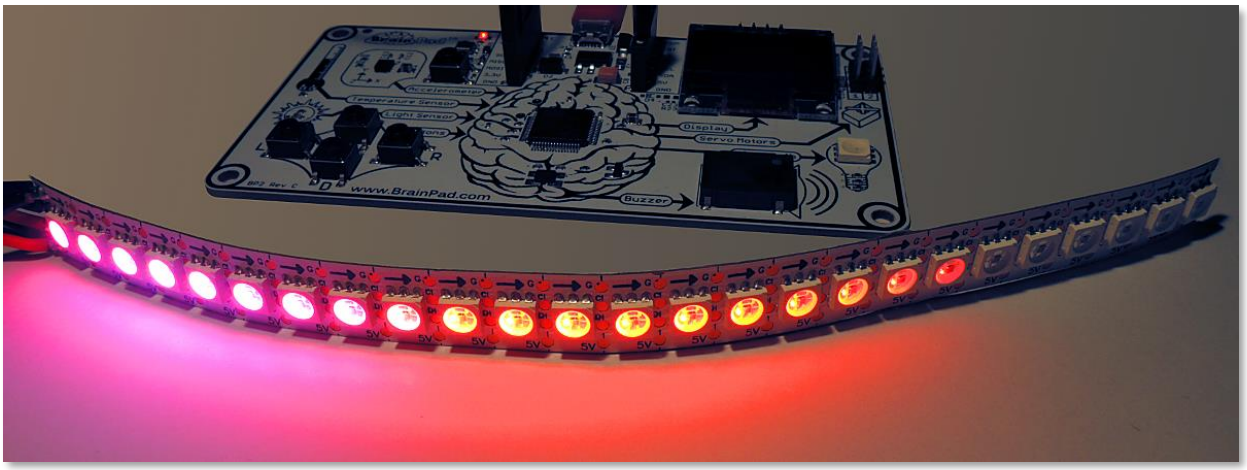
Debido a la cantidad de bloques que necesitaremos, es más fácil cambiarnos a la ventana de JavaScript y copiar y pegar el código. Copia y pega el siguiente código en la ventana de edición de JavaScript en Microsoft MakeCode:

```
let center = 12
let startRed = 0
let startGreen = 0
let startBlue = 0
let red = 0
let green = 0
```

```
let blue = 0
let data: Buffer = pins.createBuffer(4)
let dummy: Buffer = pins.createBuffer(4)

pins.spiFrequency(1000000)

function SendStart() {
    data[0] = 0
    data[1] = 0
    data[2] = 0
    data[3] = 0

    pins.spiTransfer(data, dummy)
}

function SendRGB(red: number, green: number, blue: number) {
    data[0] = 255
    data[1] = blue
    data[2] = green
    data[3] = red

    pins.spiTransfer(data, dummy)
}

forever(function () {
    startRed = Math.constrain(startRed + Math.randomRange(-20, 20), 0, 255)
    startGreen = Math.constrain(startGreen + Math.randomRange(-20, 20), 0, 255)
    startBlue = Math.constrain(startBlue + Math.randomRange(-20, 20), 0, 255)

    SendStart()

    for (let index = 0; index <= 24; index = index + 1) {
        red = Math.constrain(startRed - Math.abs(index - center) * 16, 0, 255)
```

```
        green = Math.constrain(startGreen - Math.abs(index - center) * 16, 0, 255)
        blue = Math.constrain(startBlue - Math.abs(index - center) * 16, 0, 255)

        SendRGB(red, green, blue)
    }
})
```

Si hiciste todo correctamente deberías ver un espectáculo de brillantes colores al azar. Prueba a cambiar parte del código para ver cómo afecta las luces.



CONCLUSIÓN

Microsoft MakeCode y el BrainPad forman un gran combo para aprender a programar. El simulador te permite probar fácilmente el código sin necesidad de tener un BrainPad. Luego puedes cargar el programa en el BrainPad simplemente copiando el programa descargado a la ventana BrainPad.

En los capítulos siguientes iremos apun más allá (‘Go Beyond’) y codificaremos usando C# o VisualBasic en Microsoft Visual Studio. Ir más allá (‘Going Beyond’) es opcional pero altamente recomendable una vez que te sientas cómodo programando en MakeCode. En muchos casos, MakeCode será todo lo que necesites.

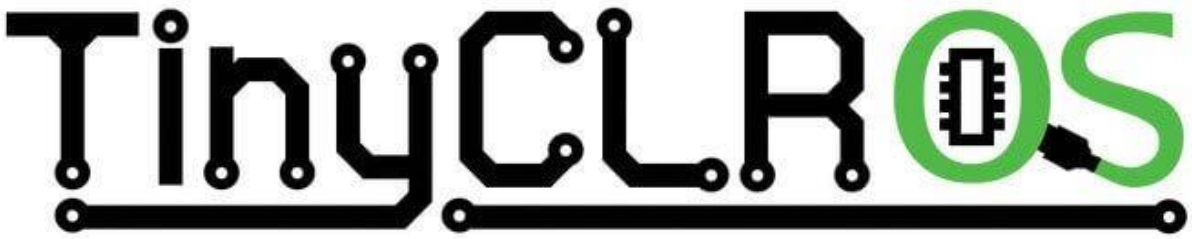
IR MÁS ALLA - GO BEYOND

Si estás leyendo esto es porque quieres aprender a programar como un profesional. Usarás Microsoft Visual Studio para programar en C# o Visual Basic. Tendrás acceso a capacidades de depuración completas junto con un rico ambiente de desarrollo.

Haber usado Microsoft Making Code no es un prerequisito, pero es altamente recomendable. Después de todo, lo llamamos Comenzar Haciendo (‘Start Making’) por una buena razón. De cualquier modo, ajusta tu cinturón de seguridad ya que vamos a comenzar un recorrido serio por la programación profesional, que puede transformarse en tu futura profesión!

TINYCLR OS™

Para que el BrainPad trabaje con .NET en C# y Visual Basic, necesitará TinyCLR OS. Este pequeño sistema operativo hace toda la magia. Interpreta códigos compilados -NET y contiene miles de servicios tales como ‘threading’ (manejo de múltiples tareas), manejo de memoria y depuración. Este es el mismo .NET que se usa para desarrollar software para smartphones y computadores. TinyCLR OS sin embargo está hecho para ejecutarse en pequeños computadores, como BrainPad, así que es un subconjunto de .NET estándar.



VISUAL STUDIO

En teoría, cualquier compilador puede ser usado con TinyCLR OS, sin embargo, TinyCLR OS solo es soportado en Microsoft Visual Studio. La edición ‘community’ de Visual Studio es un ambiente de programación completo, considerado uno de los mejores (si no el mejor) actualmente disponibles, ¡y lo mejor de todo es que es gratis! Para instalar Visual Studio necesitarás un computador con Windows, preferentemente Windows 10.

Fíjate que podrás usar el mismo Microsoft Visual Studio y las misma habilidades que aprenderás programando el BrainPad para luego desarrollar apps para muchos otros dispositivos, como smartphones, PCs e incluso consolas de juegos Microsoft Xbox.

PREPARACIÓN DEL SISTEMA.

Microsoft Visual Studio Community es gratis, pero no es un juguete y no es pequeño. Necesitarás algún tiempo para preparar el sistema. Las instrucciones completas las puedes encontrar en el sitio web de BrainPad en https://docs.brainpad.com/go-beyond/system-setup.html.

En breves palabras, necesitas descargar e instalar:

1. Microsoft Visual Studio Community editio. No olvides seleccionar la opción ".NET desktop development" durante la configuración.
2. El sistema de proyectos GHI Electronics TinyCLR OS.

Si encuentras dificultades, puedes visitar nuestro foro para obtener ayuda:
https://forums.ghielectronics.com/c/brainpad

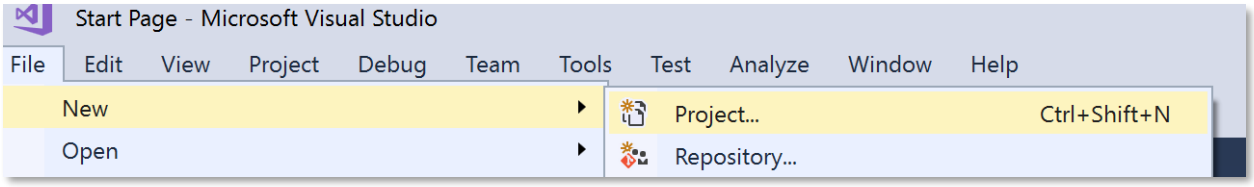
Una vez que tienes preparado el computador, necesitas cargar TinyCLR en el BrainPad. Descarga el archivo con el firmware TinyCLR OS desde https://docs.brainpad.com/resources/downloads.html y luego cárgalo en el BrainPad tal como cargarías cualquier archivo creado con Microsoft MaekCode:

1. Presiona el botón de reset por tres segundos. La luz se encenderá de color verde.
2. El PC detectará un dispositivo de almacenamiento externo y abrirá una ventana para él.
3. Graba el firmware TinyCLR OS en el dispositivo de almacenamiento. Puedes copiar y pegar el archivo en la ventana del BrainPad.

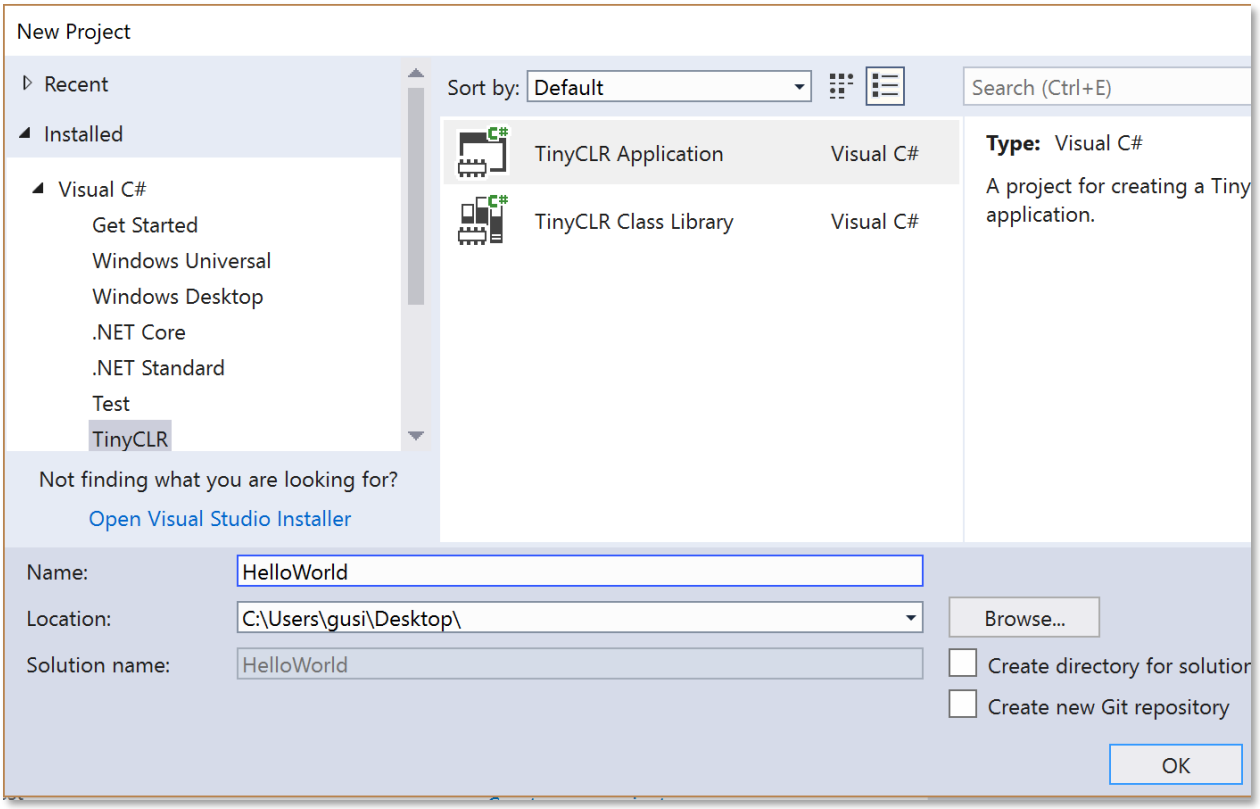
HOLA, MUNDO.

Es habitual cuando se programa un dispositivo nuevo, o cuando se usan nuevas herramientas de programación, comenzar con un programa muy simple para verificar que todo está funcionando correctamente. Este programa es normalmente llamado "hola mundo" ('hello world'). Hagamos precisamente eso ahora.

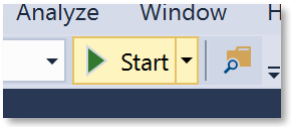
Abre Microsoft Visual Studio y comienza un nuevo proyecto. En el menú "File" selecciona "New" y luego "Project" (File > New > Project).



Debería haber una opción TinyCLR bajo Visual C#. Si no la hay, quiere decir que no has instalado el sistema de proyectos TinyCLR OS. Aquí muestro cómo he llamado "HelloWorld" a mi proyecto y lo he colocado en mi escritorio.



Puedes cargar ahora este programa vacío en tu BrainPad. No hará mucho, pero ayudará a verificar que todo está trabajando bien. Conecta el BrainPad y haz click en Start.

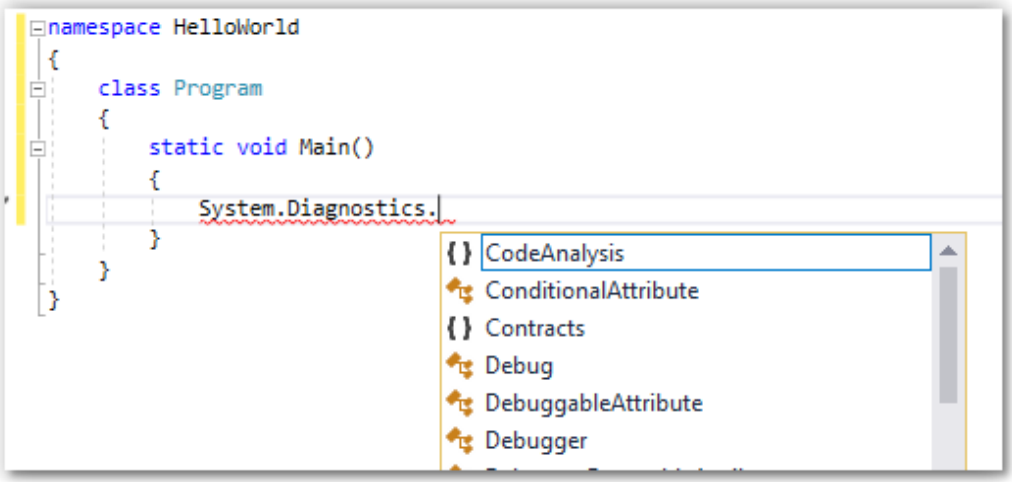


Fíjate en la actividad que se muestra en la venta de salida (‘Output’). Si no ves esa venta, presiona Alt+Ctrl+O en tu teclado o selecciona “Output” en el menú “View” (View > Output).

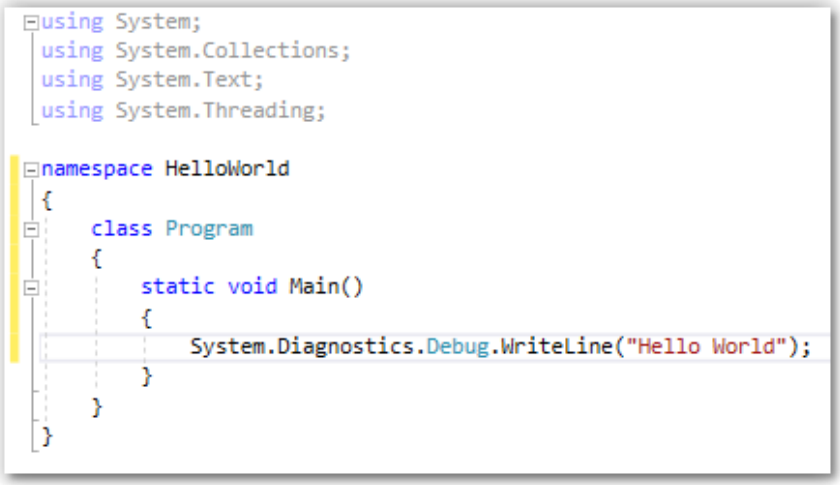
Ahora estamos listos para hacer que BrainPad diga “Hola Mundo”. Primero vamos desplegar “Hola Mundo” en la ventana de salida. Después lo mostraremos en el display del BrainPad. Escribe esta línea en el programa, justo debajo del paréntesis de llave izquierdo ({) a continuación de Main().

```
System.Diagnostics.Debug.WriteLine("Hello World");
```

Fíjate cómo Visual Studio automáticamente te da sugerencias de palabras.



Cuando estés listo, el código debería verse así:



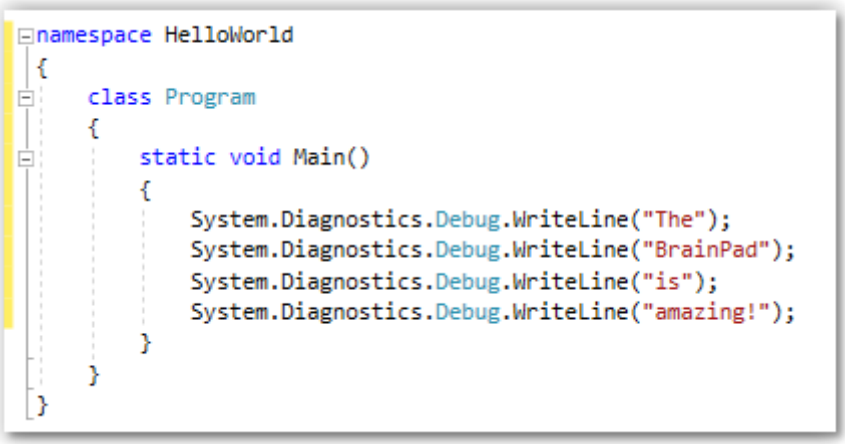
Haz click en Start y observa la ventana de output nuevamente.

```
The debugging target runtime is loading the application assemblies and starti
Ready.

Done.

Waiting for debug commands...

'GHIElectronics.TinyCLR.VisualStudio.dll' (Managed): Loaded 'C:\Users\gusi\De
The thread '<No Name>' (0x2) has exited with code 0 (0x0).
Hello World
The thread '<No Name>' (0x1) has exited with code 0 (0x0).
The program '[3] TinyCLR application: Managed' has exited with code 0 (0x0).
```

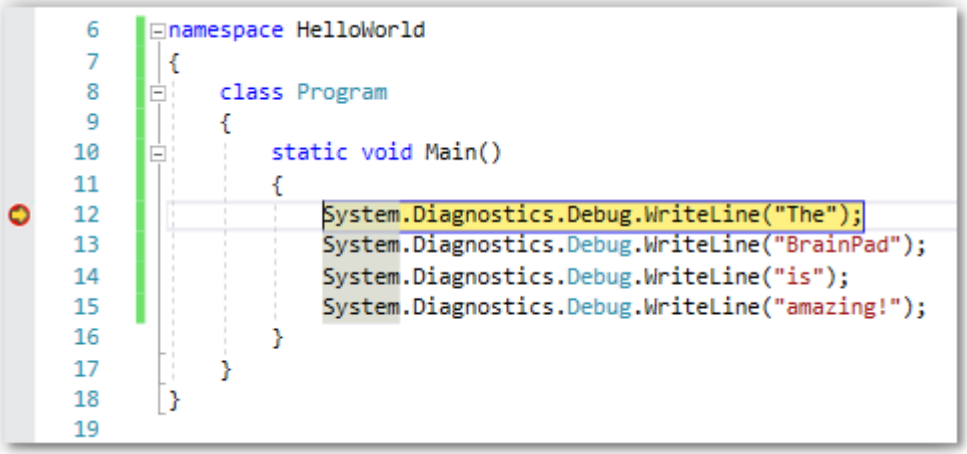
¿Viste dónde dice “Hello World”? Este es un buen comienzo, pero no muy emocionante. Intentemos con múltiples líneas. Reemplaza la línea que agregaste por estas cuatro líneas.

```
System.Diagnostics.Debug.WriteLine("The");
System.Diagnostics.Debug.WriteLine("BrainPad");
System.Diagnostics.Debug.WriteLine("is");
System.Diagnostics.Debug.WriteLine("amazing!");
```

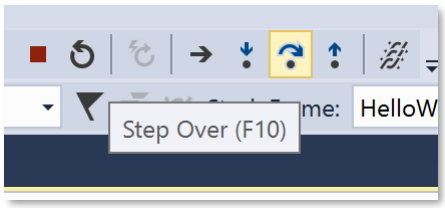
Pon el cursor en la primera línea y presiona F9. Eso agregará un punto de detención (“breakpoint”). Puedes hacer lo mismo en el menú “Debug”.



Inicia al programa igual que antes. En cuanto la ejecución llegue al punto de detención, el programa quedará en pausa en esa línea.



Detendrá justo antes de ejecutar la línea. Presiona F10 o haz click en el botón “step over” (avanzar un paso).



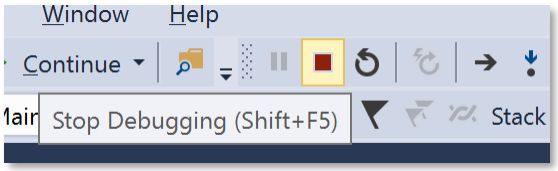
Esto hará que el programa ejecute una línea de código, o un paso. Ya que la línea en que estamos imprime “The”, verás que ese texto aparece en la ventana de salida. Sigue avanzando con F10 mientras observas la ventana de salida. Este proceso se llama ejecución paso a paso y es muy útil para depurar programas cuando algo no está funcionando como debiera.

PROGRAMAS SIN FIN.

En un PC o smartphone tiene sentido que los programas se cierren después de usarlos. Sin embargo, en dispositivos pequeños, llamados dispositivos embebidos (‘embedded’), los programas normalmente no terminan. Piensa en un microondas por ejemplo. El microondas tiene un pequeño ‘cerebro’ en su interior que está siempre monitoreando las teclas en caso que sean presionadas, y también mantiene actualizado el reloj en la pantalla. A menos que desenchufes el microondas el programa correrá permanentemente. Esto se llama un loop infinito y es usado a menudo en programación.

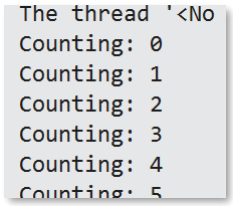
Vamos a crear un contador que corra por siempre (‘forever’) en el BrainPad. Basicamente, el BrainPad contará una vez por segundo y nunca se detendrá. Más abajo se muestra cómo sería el código.

Fíjate que tienes que detener el programa si está corriendo antes de poder modificarlo. Para hacerlo presiona el botón de ‘stop’.

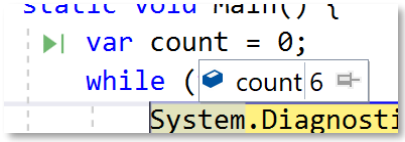



```
namespace HelloWorld {
    class Program {
        static void Main()
        {
            var count = 0;

            while (true)
            {
                System.Diagnostics.Debug.WriteLine("Counting: " + count);
                Thread.Sleep(1000); // Wait one second

                count = count + 1;
            }
        }
    }
}
```

La ventana de salida mostrará el contador:

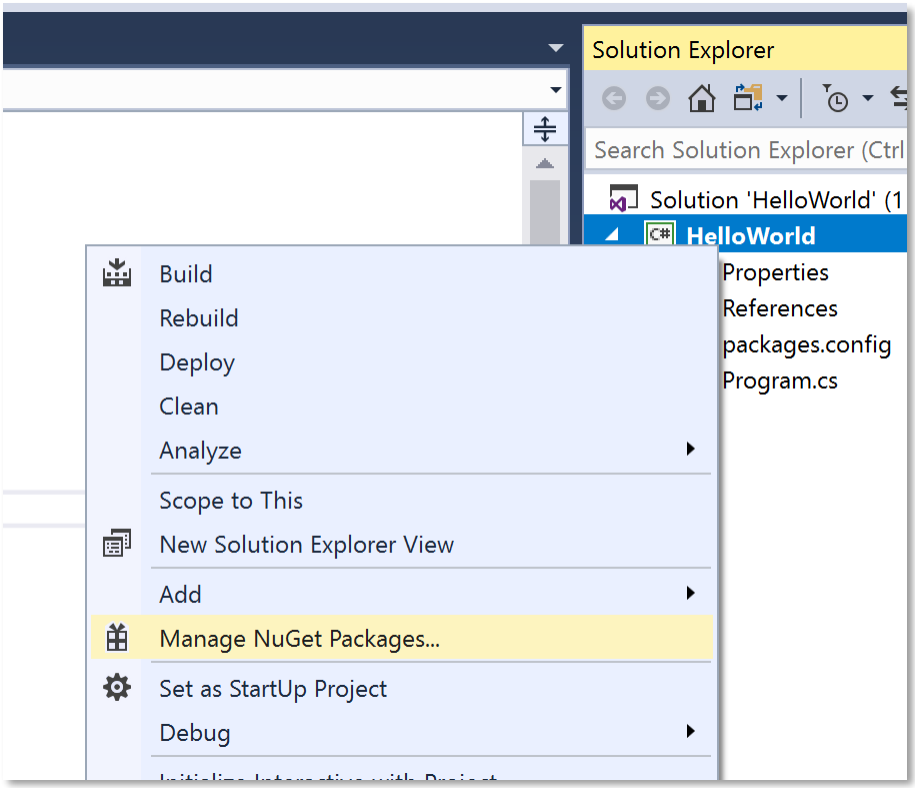


Mientras el programa está corriendo inserta un punto de detención (‘breakpoint’) en la línea del contador. Así podrás avanzar paso a paso en el programa, tal como hicimos antes. Ahora pon el mouse sobre la variable ‘count’, Visual Studio te mostrará el valor actual de la variable. Cuando yo lo hice, el valor esa 6. A propósito, inspeccionar variables es otra característica útil para depurar programas.

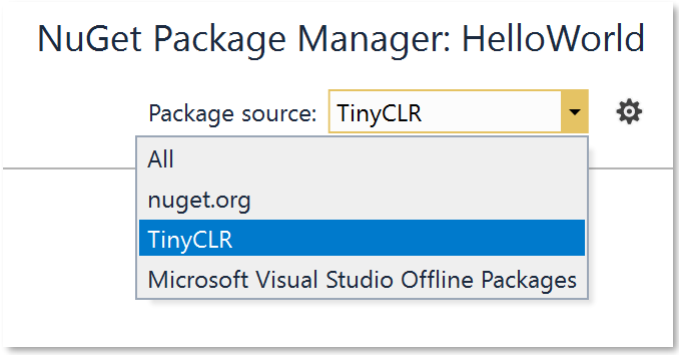


LIBRERIAS DE BRAINPAD.

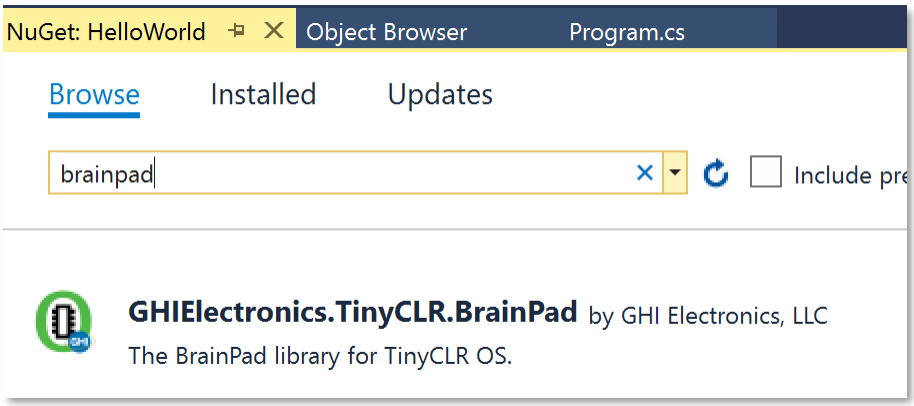
Hasta ahora hemos usado TinyCLR puro para correr programas simples. El BrainPad se distribuye con librerías para ayudar a usar todas las entradas y salidas disponibles. Haz click con el botón derecho en la ventana Solution Explorer y selecciona ‘Manage NuGet Packages’…



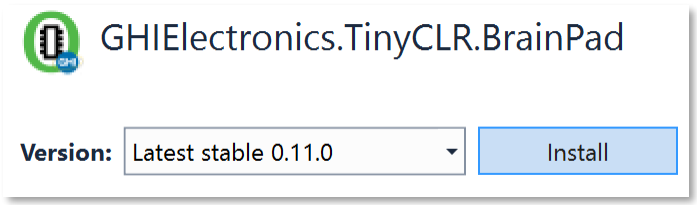
NuGet es un servicio en línea para almacenar librerías. También puedes descargar las librerías y almacenarlas localmente en tu PC. Selecciona dónde quieres que queden las librerías. En mi caso, por ejemplo, yo guardo las librerías localmente en un directorio llamado TinyCLR.



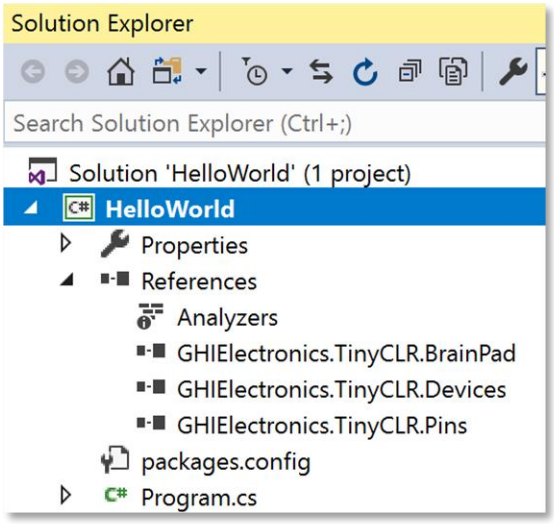
Bajo 'Browse', ingresa 'brainpad' en la barra de búsqueda.



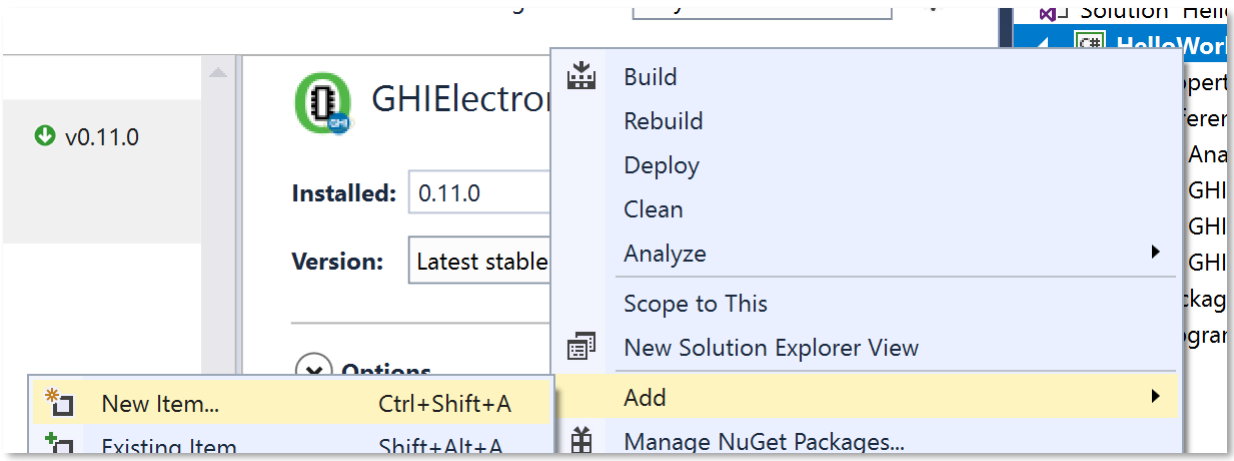
Esto mostrará la librería `GHIElectronics.TinyCLR.BrainPad`. Selecciónala y haz click en 'Install'.



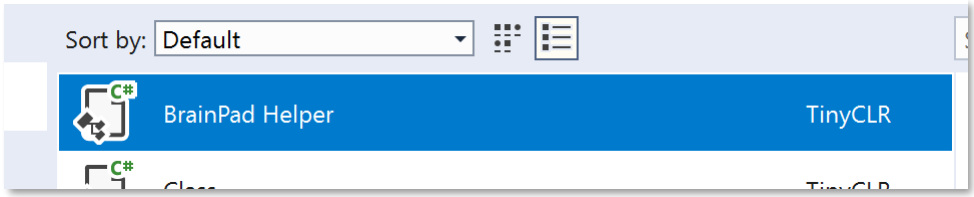
Este paso agregará las librerías necesarias en la 'referencias' en el Solution Explorer.



Para facilitar el agregar librerías a tu código, se provee una clase auxiliar. Haz click con el botón derecho en el proyecto y selecciona Add > New Item … También puedes usar Ctrl+Shift+A para eso.



De las opciones disponibles, selecciona BrainPad Helper y haz click en el botón Add.



¡Aquí comienza la diversión! Vuelve a Program.cs, donde agregamos el código del contador anteriormente. Modifica el código para mostrar el contador en el display del BrainPad.

```
namespace HelloWorld
{
```

```
    class Program
    {
        static void Main()
        {
            var count = 0;

            while (true)
            {
                BrainPad.Display.DrawSmallText(0, 0, "Count: " + count);
                BrainPad.Display.RefreshScreen();
                BrainPad.Wait.Seconds(1);

                count = count + 1;
            }
        }
    }
}
```

La línea que despliega el contador es similar al comando Debug.WriteLine que usamos anteriormente, excepto que esta vez despliega sobre la pantalla del BrainPad. La segunda línea es necesaria para refrescar (actualizar) la pantalla. Los display son mucho más lentos que los procesadores. Si refrescáramos la pantalla cada vez que desplegamos, el display correría muy lento. En vez de eso, el procesador 'dibuja' la pantalla en su memoria. Solo cuando se llama el comando RefreshScreen el dibujo es traspasado al display. Piensa en un videojuego donde tienes que dibujar una nave, obstáculos, enemigos, etc. Haces todo este dibujo en memoria solamente, lo que es muy rápido, y después refrescas el display cuando es necesario.

La última linea es para pedir al BrainPad que espere un segundo.

PELOTA SALTARINA.

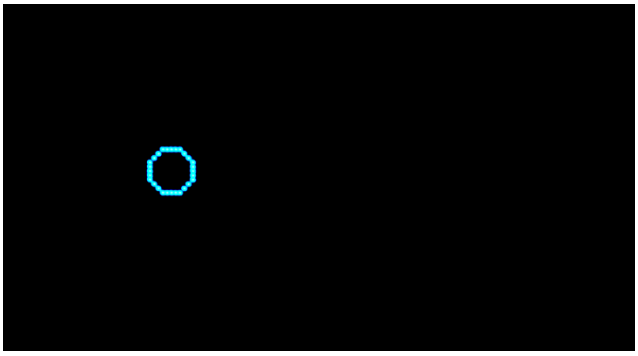
Las librerías del BrainPad hacen que todo parezca más idioma inglés que código. Agrega alguna lógica y podrás crear programas espectaculares. Este libro no cubre la enseñanza dde C# o Visual Basic, pero eso no significa que no podamos tener algo de diversión.

Volvamos al programa original y dibujemos un círculo de 5 pixeles de diámetro. Lo ubicaremos en la posición 30,30 del display. Las posiciones en los display comienzan en la parte superior izquierda, que es la ubicación 0,0. Para mover imágenes a la derecha hay que incrementar el valor x, que es el primer número. Por ejemplo, 30,0 indica un punto ubicado 30 pixeles a la derecha de la esquina superior de la pantalla, y pegado el borde superior.

```
namespace HelloWorld
```

```
{
    class Program
    {
        static void Main()
        {
            var x = 30;
            var y = 30;

            while (true)
            {
                BrainPad.Display.DrawCircle(x, y, 5);
                BrainPad.Display.RefreshScreen();
                BrainPad.Wait.Seconds(1);
            }
        }
    }
}
```

¡Ahora tenemos un círculo!



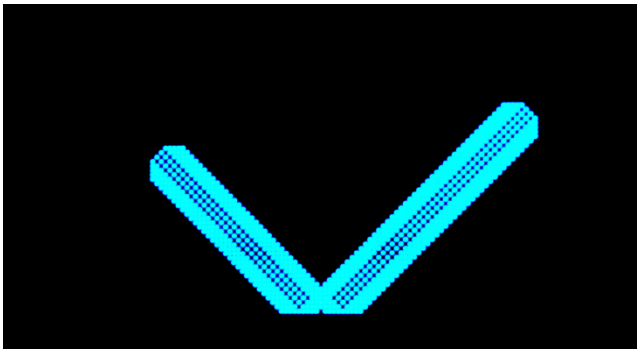
En nuestro loop sin fin incrementaremos las variables x e y en cada ciclo. Ya que comenzamos en 30, la variable x cambiará a 31, 32, 33, 34, etc. ¿Se pone más entretenido esto?

El círculo se moverá lentamente debido a la espera de un segundo. Lo cambiaremos a 0.01. También agregaremos alguna lógica para rebotar la pelota de vuelta. Para devolver la pelota necesitamos agregar nuevas variables, dx y dy. Estas variables registrarán la dirección en la que se está moviendo la pelota. Luego revisaremos si la pelota llegó a un borde. Si llegó, entonces invertiremos la dirección.

```
namespace HelloWorld
{
    class Program
    {
        static void Main()
        {
            var x = 30;
            var y = 30;
            var dx = 1;
            var dy = 1;

            while (true)
            {
                x = x + dx;
```

```
                y = y + dy;

                if (x < 0 || x > BrainPad.Display.Width)
                {
                    dx = dx * -1;
                }

                if (y < 0 || y > BrainPad.Display.Height)
                {
                    dy = dy * -1;
                }

                BrainPad.Display.DrawCircle(x, y, 5);
                BrainPad.Display.RefreshScreen();
                BrainPad.Wait.Seconds(0.01);
            }
        }
    }
}
```

El círculo se verá así mientras se mueve por la pantalla.



Pero nosotros queremos una pelota que rebote así que limpiaremos la pantalla antes de dibujar el círculo. Agrega esta línea justo antes de DrawCircle.

```
BrainPad.Display.Clear();
```

¿Quieres que se mueva más rápido? El delay que tenemos, de 0.01, es muy pequeño así que eso no es problema. Estamos desplazando la pelota solo un pixel en cada ciclo. Si lo movemos 5 pixeles irá 5 veces más rápido. Esto se logra cambiando el valor inicial dx y dy a 5.

```
var dx = 5;
var dy = 5;
```

Pero ¿qué pelota rebota sin hacer ruido?. Hagamos que el BrainPad suene cada vez que la pelota rebota. Agrega esta línea dentro de cada sentencia 'if'.

```
BrainPad.Buzzer.Beep();
```

¡Ahá! Ahora tenemos una pelota que rebota. Pero esto es muy ruidoso, especialmente si estás en una sala de clases con 30 BrainPads. Agrega una sentencia para hacer sonido solo si el botón 'down' está presionado.

```
if (BrainPad.Buttons.IsDownPressed())
{
    BrainPad.Buzzer.Beep();
}
```

¿Puedes dar un efecto bacán a la pelota agrandándola y achicándola mientras se mueve? Fíjate cómo cambiamos la posición de la pelota dentro de la frontera de los bordes. Haremos exactamente lo mismo. Necesitarás una variable r para el radio y una variable dr para la dirección del radio, creciendo o achicándose.

```
var r = 3;
var dr = 1;
```

Cambiaremos el valor del tamaño en el rango de 1 a 8.

```
r = r + dr;

if (r < 1 || r > 8)
{
    dr = dr * -1;
}
```

Por supuesto, tenemos que cambiar la línea DrawCircle para que use la variable r en lugar del valor fijo 5.

```
BrainPad.Display.DrawCircle(x, y, r);
```

Aquí está el código completo. Pero antes de copiar y pegar el código, si le has puesto al proyecto un nombre distinto de HelloWorld (sin espacios, con mayúsculas y minúsculas) entonces tienes un espacio de nombres (‘namespace’) distinto. En ese caso tienes que mantener el espacio de nombres de tu propio código y no copiar el que está en el código incluido más abajo. Por ejemplo, si tu código muestra

```
namespace SomethingElse
```

Entonces mantén la línea ‘namespace’ y copia el resto del código, todo comenzando con `class Program` y terminando con el penúltimo paréntesis de llave (}). El último paréntesis de llave es parte del espacio de nombres (‘namespace’).

```
namespace HelloWorld
{
    class Program
    {
        static void Main()
        {
            var x = 30;
            var y = 30;
            var dx = 5;
            var dy = 5;
            var r = 3;
            var dr = 1;

            while (true)
            {
                x = x + dx;
                y = y + dy;
```

```
                if (x < 0 || x > BrainPad.Display.Width)
                {
                    if (BrainPad.Buttons.IsDownPressed())
                    {
                        BrainPad.Buzzer.Beep();
                    }

                    dx = dx * -1;
                }

                if (y < 0 || y > BrainPad.Display.Height)
                {
                    if (BrainPad.Buttons.IsDownPressed())
                    {
                        BrainPad.Buzzer.Beep();
                    }

                    dy = dy * -1;
                }

                r = r + dr;

                if (r < 1 || r > 8)
                {
                    dr = dr * -1;
                }

                BrainPad.Display.Clear();
                BrainPad.Display.DrawCircle(x, y, r);
                BrainPad.Display.RefreshScreen();
                BrainPad.Wait.Seconds(0.01);
            }
        }
    }
}
```

LUZ DE NAVIDAD.

Has leído correctamente. Vamos a crear una luz de Navidad, como una única ampolleta. Para esto vamos a usar una de las miles de funciones disponibles en TinyCLR OS. Esta función nos entrega un número al azar. A diferencia de la vida real, en un computador nada es verdaderamente al azar. Eso dificulta la generación de números aleatorios. Afortunadamente hay una función de números aleatorios preconstruída, que hace todo más fácil. Solo tenemos que crear un objeto 'random' (aleatorio) y pedirle el próximo número aleatorio.

Usaremos esos números aleatorios para fijar los colores primarios de rojo, verde y azul de la ampolleta ('Light Bulb') del BrainPad. En caso que no lo sepas, con esos tres colores puedes crear cualquier color que desees.

```
static void Main()
{
    var rnd = new Random();

    while (true)
    {
        BrainPad.LightBulb.TurnColor(rnd.Next(100), rnd.Next(100), rnd.Next(100));
        BrainPad.Wait.Seconds(0.1);
    }
```

```
}
```

VIENDO LA LUZ.

Queremos poner una planta en alguna parte alinterior de la casa y nos preguntamos cuánta luz recibirá ese rincón. Queremos tener un resumen del nivel de luz a lo largo del día en forma de gráfico.

El gráfico es muy simple. Básicamente dibujaremos una línea vertical que parte donde está el nivel de luz y termina en el fondo de la pantalla. La línea se desplazará un pixel a la derecha en cada ciclo.

```
static void Main()
{
    var x = 300;

    while (true)
    {
        x++;

        if (x > BrainPad.Display.Width)
        {
            x = 0;

            BrainPad.Display.Clear();
            BrainPad.Display.DrawSmallText(30, 0, "Light Level");
        }

        var light = BrainPad.LightSensor.ReadLightLevel() / 2;
        var y = BrainPad.Display.Height - light;

        BrainPad.Display.DrawLine(x, y, x, BrainPad.Display.Height);
        BrainPad.Display.RefreshScreen();
        BrainPad.Wait.Seconds(0.1);
    }
}
```

Tu probablemente quieres que el programa haga una pausa cuando llegue al extremo derecho, pero en este ejemplo lo que haremos es limpiar la pantalla y volver a comenzar. Hagamos que el loop se ejecute rápidamente para que podamos ver resultados inmediatos. Si quieres graficar 2 horas y la pantalla tiene 128 pixeles de ancho, necesitarás un minuto de espera en cada ciclo. Esto grabará 128 minutos a lo ancho de la pantalla.

¡También puedes dejar el programa tal como está y cubrir el sensor con tu mano para hacer dibujos geniales!

Ya que a estas alturas somos programadores expertos, ¿puedes responder estas preguntas?

1. ¿Por qué estamos dividiendo el nivel de luz por 2?
2. ¿Por qué ponemos la línea inicial en `BrainPad.Display.Height - light`?
3. ¿Por qué x es inicializado con el valor 300? ¿Qué pasaría si usáramos 0?

Aquí están las respuestas, pero prométeme que intentarás responderlas por ti mismo primero. El nivel de luz tiene valores entre 0 y 100. La pantalla tiene 64 pixeles de alto. SI dividimos el nivel de luz por la mitad, el máximo valor será 100/2, o sea, 50. Con una pantalla de 64 pixeles de alto, nos sobrarán 14 pixeles en la parte superior. Allí aprovecharemos de escribir "Light Level." ('nivel de luz').

La segunda respuesta se relaciona a la forma en que queremos mostrar los niveles de luz. Las coordenadas y más pequeñas están cerca de la parte superior del display, pero queremos que los niveles más bajos de luz se muestren cerca del fondo de la pantalla. Al restar el nivel de señal sobre la altura del display, estamos invirtiendo el gráfico.

La última respuesta es un truco para ayudar cuando el programa comienza a correr. Solo imprimimos "Light Level" cuando limpiamos la pantalla. Así, al poner x en un valor alto, automáticamente estaremos fuera del rango que pusimos y forzaremos que aparezca "Light Level" en la pantalla. Prueba a poner x en 0 y verás que "Light Level" no aparecerá en la pantalla la primera vez que se muestra el gráfico. Solo aparecerá cuando la pantalla sea refrescada después de mostrar el primer gráfico. También podríamos imprimir el texto en cada ciclo, pero eso haría más lento el programa. Siempre tenemos que pensar con astucia cuando programamos y hacer solo lo que es necesario.

MULTITAREA. (MULTITASKING)

Pedirle a un sistema que haga múltiples cosas a la vez es algo complejo internamente. Típicamente no es algo amigable en pequeños sistemas. Las buenas noticias son que TinyCLR OS permite hacer multitarea ('multitasking') y es fácil de usar. Trabaja exactamente como lo haría en cualquier sistema mayor programado en .NET. Multitarea ('multitasking' en inglés) es llamado "threading", donde cada "thread" es una tarea individual ejecutando una parte del programa.

¿No se te ocurre para qué necesitarías threading? ¿Cómo podrías contar en el display al mismo tiempo que parpadeas la luz? Por supuesto, si quieres incrementar el contador cada vez que parpadea la luz sería fácil.

```
static void Main()
{
    var count = 0;

    while (true)
    {
        BrainPad.LightBulb.TurnGreen();
        BrainPad.Wait.Seconds(0.1);
        BrainPad.LightBulb.TurnOff();
        BrainPad.Wait.Seconds(0.9);

        BrainPad.Display.DrawSmallText(0, 0, "Count: " + count);
        BrainPad.Display.RefreshScreen();

        count = count + 1;
    }
}
```

Ahora, cambia el programa para mantener el parpadeo una vez por segundo, pero contando lo más rápido posible. No podrás hacerlo sin usar threading. Comencemos moviendo el código de parpadeo a su propio método, que

llamaremos 'Blink'. Un método es una porción de código que realiza una tarea particular. Para realizar esa tarea solo necesitas llamar a ese método. Por ejemplo, la luz de BrainPad tiene un método llamado TurnGreen ('ponerse verde') para que se encienda verde.

```
static void Blink()
{
    while (true)
    {
        BrainPad.LightBulb.TurnGreen();
        BrainPad.Wait.Seconds(0.1);
        BrainPad.LightBulb.TurnOff();
        BrainPad.Wait.Seconds(0.9);
    }
}
```

Si llamas este método, parpadeará la luz una vez por segundo; sin embargo, nunca terminará debido a que tiene un loop de tipo while. Esto está bien porque llamaremos a este método desde un thread separado. ¿Estás preparado para el complejo código necesario para crear un Thread? Aquí está:

```
new Thread(Blink).Start();
```

Simplemente le decimos al sistema que cree un nuevo Thread para el método Blink y luego le pedimos que lo eche a correr.

¡Momento! Antes de ejecutar el código necesitamos un último cambio. Cada thread en el sistema tiene que incluir una espera ('sleep'). Esto ayuda a lsistema a procesar sus tareas internas y permite que los otros threads se ejecuten sin problemas. Tú puedes decir que solo tenemos un thread. En realidad, tenemos dos. El sistema crea automáticamente un thread que ejecuta el 'Main()', que es el método principal. Si quieres que se ejecute lo más rápido posible, solo agrega la espera mínima.

```
BrainPad.Wait.Minimum();
```

Aquí están los dos métodos, Main y Blink.

```
static void Blink()
{
    while (true)
    {
        BrainPad.LightBulb.TurnGreen();
        BrainPad.Wait.Seconds(0.1);
        BrainPad.LightBulb.TurnOff();
        BrainPad.Wait.Seconds(0.9);
    }
}

static void Main()
{
    new Thread(Blink).Start();

    var count = 0;

    while (true)
    {
        BrainPad.Display.DrawSmallText(0, 0, "Count: " + count);
        BrainPad.Display.RefreshScreen();
```

```
        BrainPad.Wait.Minimum();

        count = count + 1;
    }
}
```

Ahora hay dos loops infinitos y ambos están ejecutándose simultáneamente. Aunque este es un ejemplo muy simple, te muestra las posibilidades. Como ejercicio, agrega otro thread para hacer sonar el 'beep' cada 3 segundos.

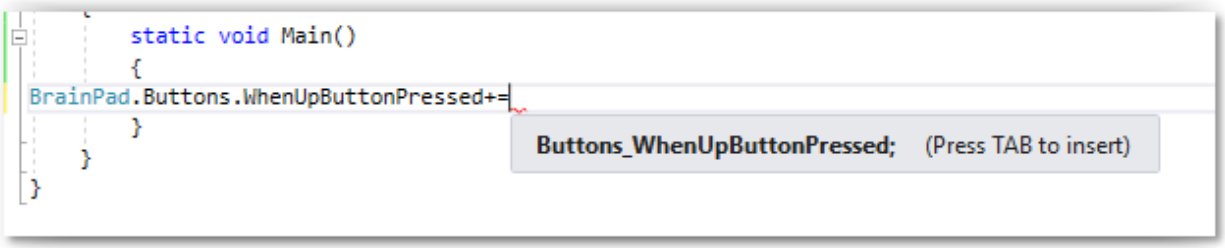
LLAMAME.

Ya hemos usado los botones del BrainPad, así que ya sabes cómo trabajan. Sin embargo, siempre los hemos usado en un loop donde revisamos si el botón fue presionado. El sistema puede revisar si un botón fue presionado milones de veces antes de que lo presionen. Esto es malo para sistemas operados con baterías. Un buen diseño se irá a dormir cuando no hay tareas que ejecutar, para ahorrar consumo de batería. Si tu teléfono celular no durmiera no duraría una hora sin tener que recargarlo.

Este tema puede llegar a ser complejo así que solo cubriremos los eventos de botones. En vez de estar permanentemente revisando los botones, le pediremos al sistema que nos llame cuando se presione un botón. Queremos hacer un sonido ('beep') cada vez que se presione el botón 'Up'. También haremos parpadear la luz.

```
static void Main()
{
    while (true)
    {
        BrainPad.LightBulb.TurnGreen();
        BrainPad.Wait.Seconds(0.1);
        BrainPad.LightBulb.TurnOff();
        BrainPad.Wait.Seconds(0.9);

        if (BrainPad.Buttons.IsUpPressed())
        {
            BrainPad.Buzzer.Beep();
        }
    }
}
```

Presiona el botón Up y nada ocurrirá. Mantenlo presionado y sonará una vez por segundo. Esto es natural porque el loop tiene esperas y el botón es revisado una vez por segundo. Puedes disminuír la espera, pero eso no resuelve el problema. En cambio, usaremos eventos. Visual Studio puede generar automáticamente el código necesario. Comienza escribiendo "`BrainPad.Buttons.WhenDownButtonPressed`" y luego agrega "+=". Ahora puedes presionar la tecla 'Tab' para generar el método del evento.



El método del evento generado se ve así:

```
private static void Buttons_WhenUpButtonPressed()
{
    throw new NotImplementedException();
}
```

El código generado tiene una línea para generar un error (‘exception’ o ‘excepción’). Eso está allí solo para que no lo olvidemos. Reemplaza esa línea con un Beep.

```
private static void Buttons_WhenUpButtonPressed()
{
    BrainPad.Buzzer.Beep();
}
```

No olvides eliminar el código que revisa el botón en el loop principal (‘Main’).

```
static void Main()
{
    BrainPad.Buttons.WhenUpButtonPressed += Buttons_WhenUpButtonPressed;

    while (true)
    {
        BrainPad.LightBulb.TurnGreen();
        BrainPad.Wait.Seconds(0.1);
        BrainPad.LightBulb.TurnOff();
        BrainPad.Wait.Seconds(0.9);
    }
}

private static void Buttons_WhenUpButtonPressed()
{
    BrainPad.Buzzer.Beep();
}
```

A propósito, múltiples eventos pueden llamas al mismo manejador de eventos. Aquí está el código para que suene un beep cuando cualquiera de los cuatro botones es presionado.

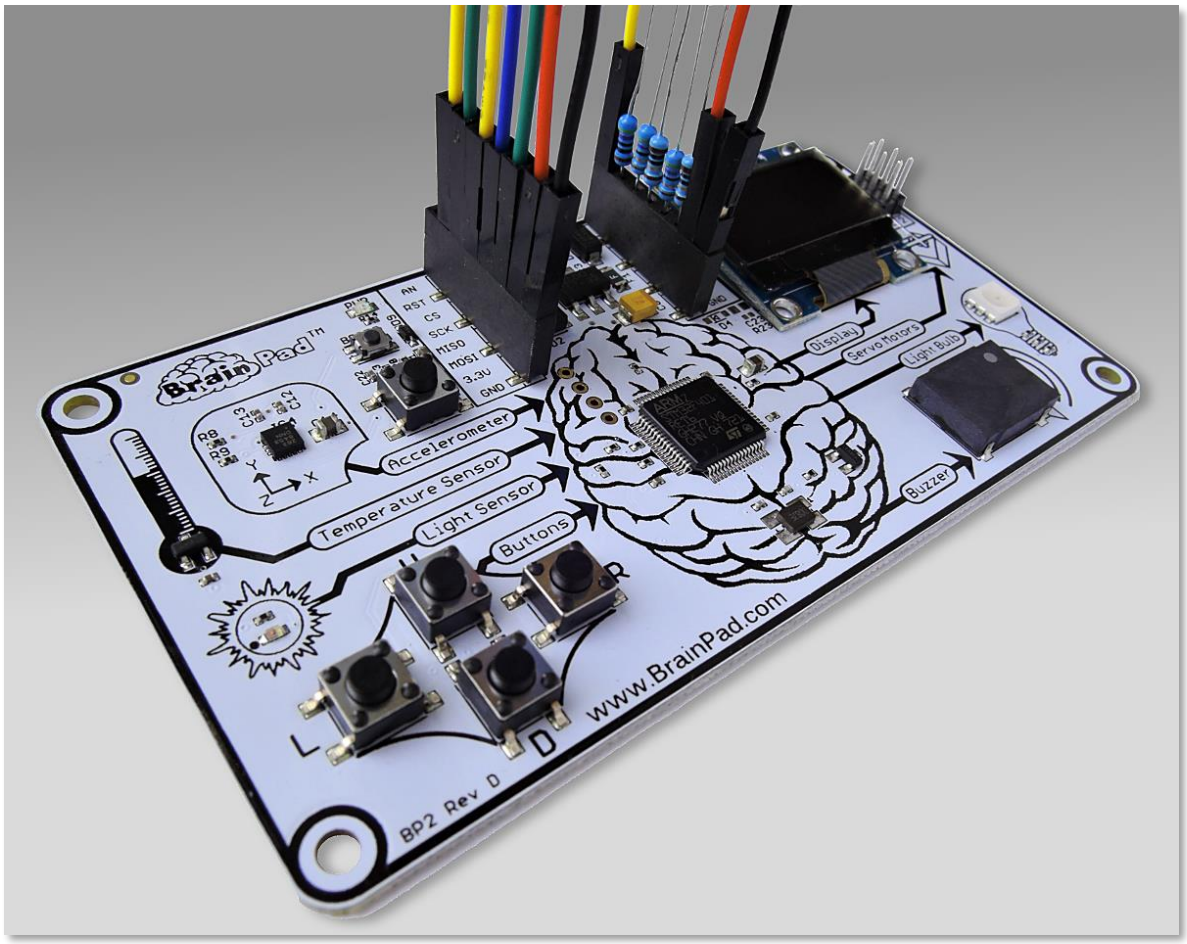
```
static void Main()
{
    BrainPad.Buttons.WhenUpButtonPressed += Beeper;
    BrainPad.Buttons.WhenDownButtonPressed += Beeper;
    BrainPad.Buttons.WhenLeftButtonPressed += Beeper;
    BrainPad.Buttons.WhenRightButtonPressed += Beeper;

    while (true)
    {
        BrainPad.LightBulb.TurnGreen();
        BrainPad.Wait.Seconds(0.1);
        BrainPad.LightBulb.TurnOff();
        BrainPad.Wait.Seconds(0.9);
    }
}
```

```
private static void Beeper()
{
    BrainPad.Buzzer.Beep();
}
```

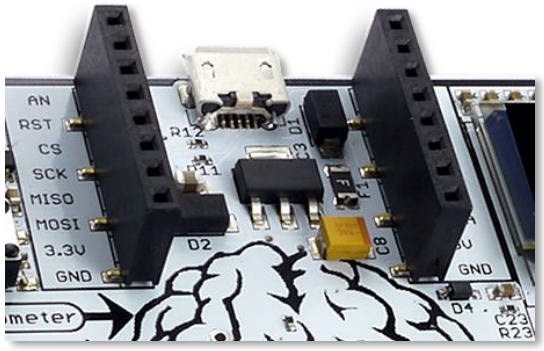
CABLES QUE ASUSTAN.

Es normal que las personas nuevas en electrónica se asusten ante la idea de conectar cables. ¿Qué pasa si me electrocuto? ¿Qué pasa si daño el circuito?

El BrainPad hace que esto sea menos terrible. Primero, el BrainPad usa un voltaje muy bajo, de 5 volts (5V). Ese voltaje no puede causar daño. Además, el BrainPad tiene protección incluída y está diseñado para que sea muy difícil dañarlo si se conecta algo de forma incorrecta. De todas formas, siempre debes entender lo que estás haciendo. Hay muchos cursos y artículos en línea que te pueden enseñar a entender circuitos.

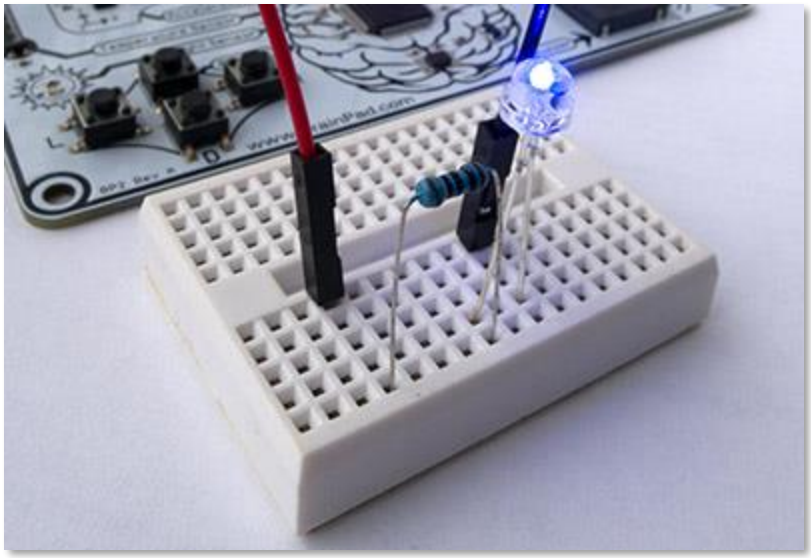


Como se dijo en el primer capítulo de este libro, hay muchas formas de expandir el BrainPad, gracias a los conectores hembra ubicados cerca del conector USB.

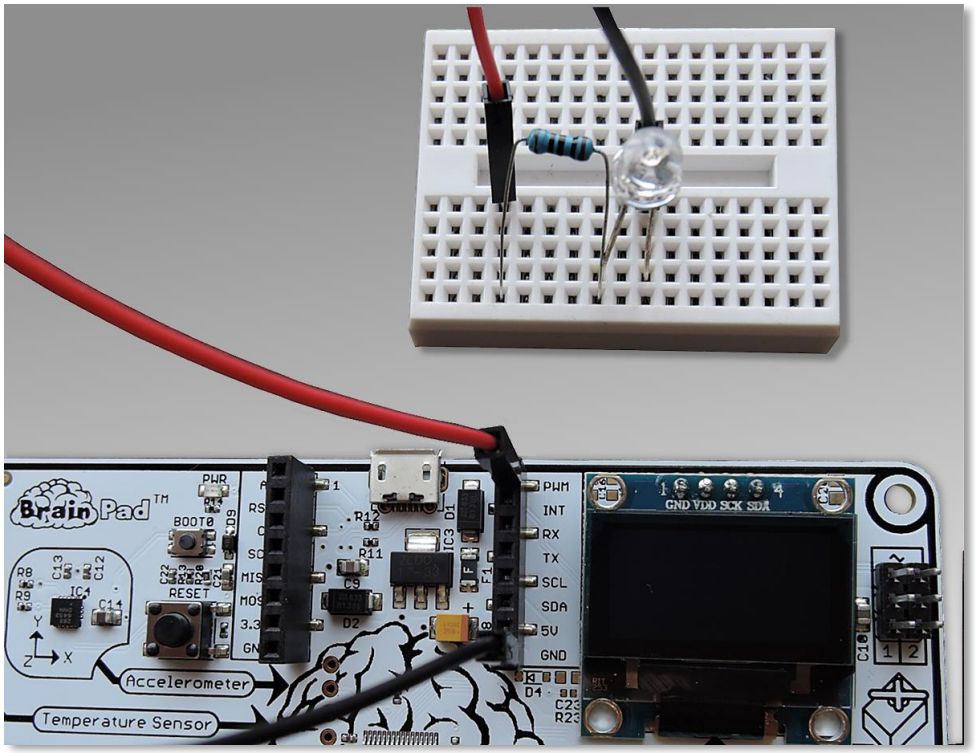


Esos pines tienes muchas funciones y necesitaríamos varios libros para explicarlas completamente. Rasparemos solo la superficie aquí, conectando un simple LED (diodo emisor de luz).

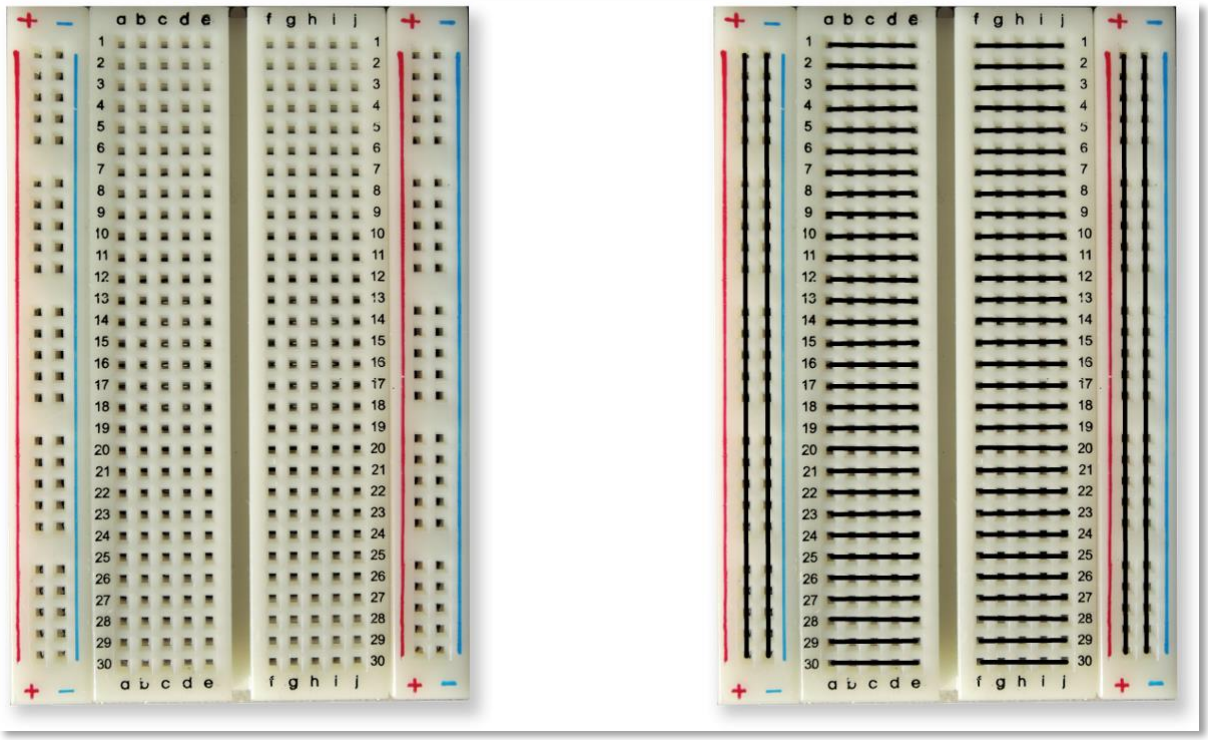
Los LEDs son muy baratos y vienen en diferentes colores. Tienen 2 patitas una de las cuales es más larga que la otra. Algunos LEDs tienen más de un color (como el de la luz del BrainPad). No trabajaremos con ellos ahora. Para controlar un LED se necesita una resistencia limitadora de corriente. Algunos valores comunes son 220, 330, o 470 ohms. También necesitarás algunos cables y un 'breadboard' (tablero de conexiones).



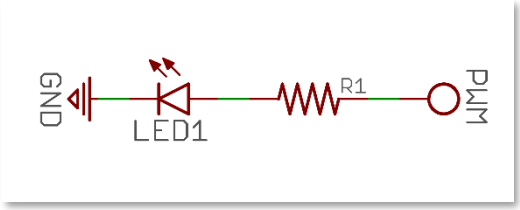
Podemos usar uno de los pines GPIO en el conector de expansión para controlar el LED. GPIO significa Entrada Salida de Propósito General en sus siglas en inglés. Usaremos el pin etiquetado como PWM.



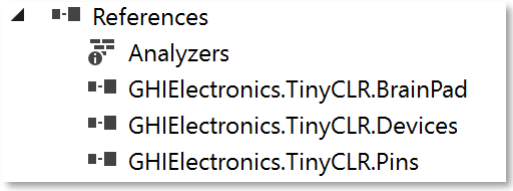
Los 'breadboard' tienen conexiones internas donde cada hilera de hoyos está conectada eléctricamente. En la imagen más abajo el 'breadboard' al lado derecho tiene líneas negras mostrando las conexiones internas. Las cosas conectada en esos hoyos estarán eléctricamente conectadas.



Un alambre irá desde el pin PWM al breadboard donde está conectada la patita más larga del LED. La patita más corta del LED irá conectada a una patita de la resistencia, y la otra patita de la resistencia irá conectada al pin GND (‘ground’ o ‘tierra’). Esto forma un círculo (o circuito) por donde circularán los electrones entre PWM y GND. Cuando el pin PWM es activado, el LED se encenderá.



Para controlar el pin PWM, que es un GPIO, necesitaremos incluir las librerías GHIElectronics.TinyCLR.Devices y GHIElectronics.TinyCLR.Pins de NuGet. Esas librerías las necesita la librería GHIElectronics.TinyCLR.BrainPad, que siempre agregamos, así que no necesitamos hacer nada ya que son automáticamente agregadas.



Necesitaremos acceso a las librerías GPIO en nuestro código. Esto se hace con una línea de código.

```
using GHIElectronics.TinyCLR.Devices.Gpio;
```

Necesitamos acceso al pin a ravés del controlador de GPIO, que está dentro del procesador.

```
var controller = GpioController.GetDefault();
var pwmPin = controller.OpenPin(BrainPad.Expansion.GpioPin.Pwm);
```

Los pines pueden ser entradas o salidas, tal como las entradas y salidas del BrainPad. Las entradas van hacia el 'cerebro' y las salidas vienen desde el 'cerebro'. Un ejemplo de entrada es un botón, como los botones del BrainPad. Un ejemplo de salida es un LED, como la luz del BrainPad. Así que el PWM debe ser una salida.

```
pwmPin.SetDriveMode(GpioPinDriveMode.Output);
```

Ahora podemos escribir ('write') en este pin, para ponerlo 'High' (activarlo) o 'Low' (desactivarlo). Pongamos eso en un loop con algunas esperas y tendremos un LED parpadeando.

```
while (true)
{
    pwmPin.Write(GpioPinValue.High);
    BrainPad.Wait.Seconds(0.1);
    pwmPin.Write(GpioPinValue.Low);
    BrainPad.Wait.Seconds(0.5);
}
```

El método completo debería verse así:

```
static void Main()
{
    var controller = GpioController.GetDefault();
    var pwmPin = controller.OpenPin(BrainPad.Expansion.GpioPin.Pwm);

    pwmPin.SetDriveMode(GpioPinDriveMode.Output);


    while (true)
    {
        pwmPin.Write(GpioPinValue.High);
        BrainPad.Wait.Seconds(0.1);
        pwmPin.Write(GpioPinValue.Low);
        BrainPad.Wait.Seconds(0.5);
    }
}
```

Conectar un botón es algo similar, excepto que el pin debe ser una entrada en vez de salida. Sin entrar en detalles, el pin debe ser una entrada con 'pull up'. Esto mantiene el pin en alto hasta que el botón sea presionado.

```
pwmPin.SetDriveMode(GpioPinDriveMode.InputPullUp);
```

No es necesario agregar resistencias al botón. Simplemente conectar el botón entre uno de los GPIO y GND.

CONCLUSION

TinyCLR OS de GHI Electronics provee programación verdaderamente profesional al BrainPad. El conocimiento que alcanzarás te servirá para programar cualquier dispositivo, desde teléfonos hasta computadores. Esta opción se facilita gracias al sistema operativo TinyCLR y a Microsoft .NET framework.

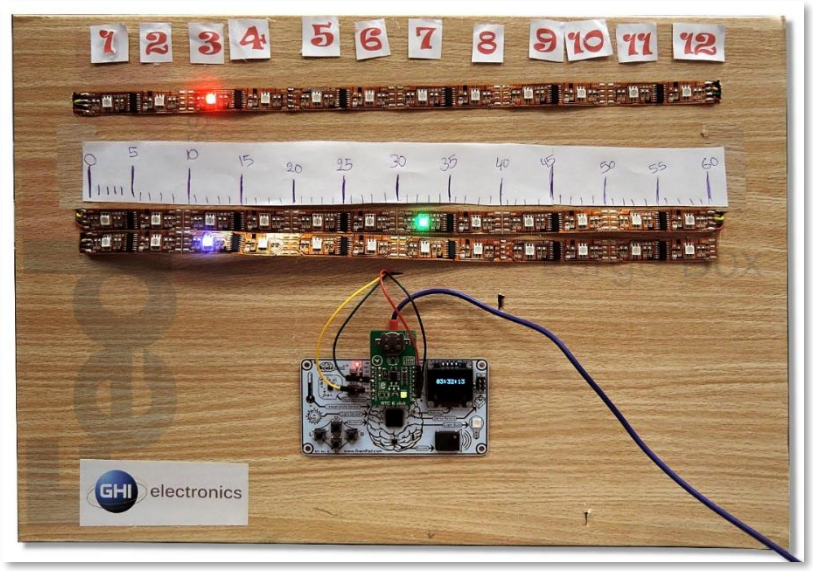
EXPANSIONES.

Los conectores de expansión abren un mundo de posibilidades. Aquí hay una lista de opciones para que pienses en los que se puede hacer.

MIKROELEKTRONIKA CLICK BOARDS™

La empresa MikroElectronika ofrece cientos de pequeños módulos que se conectan directamente al BrainPad. Hay módulos con sensores simples de temperatura y humedad y otros dispositivos sofisticados con reconocimiento de voz y módulos de electrocardiograma. También hay muchos módulos actuadores y de control para cosas como control de motores y luces digitales. Puedes encontrarlos en su sitio web https://www.mikroe.com/click.

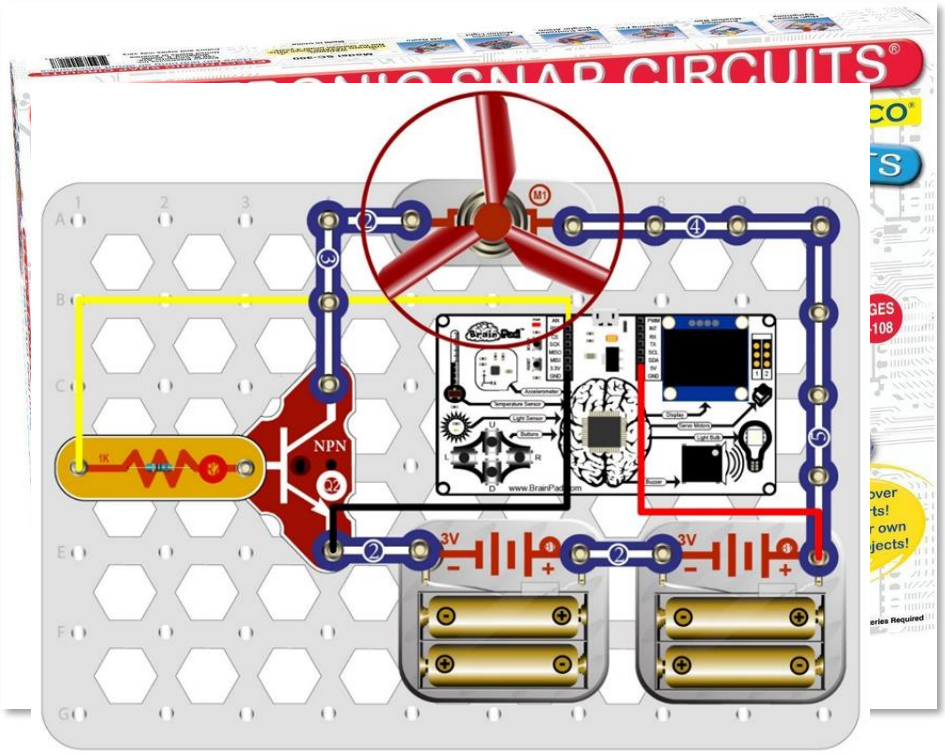
En este proyecto de ejemplo hicimos un reloj lineal que usa un módulo RTC ('real time clock' o 'reloj de tiempo real') para mantener la hora cuando el BrainPad está desenergizado (https://docs.brainpad.com/projects/linear-clock.html).



ELENCO® SNAP CIRCUITS®

Una de las plataformas más populares para aprender electrónica se llama Snap Circuits, hechos por Elenco (https://www.elenco.com). Estos populares kits se pueden encontrar en tiendas de electrónica e incluyen una amplia variedad de componentes es instrucciones para construir distintos proyectos.

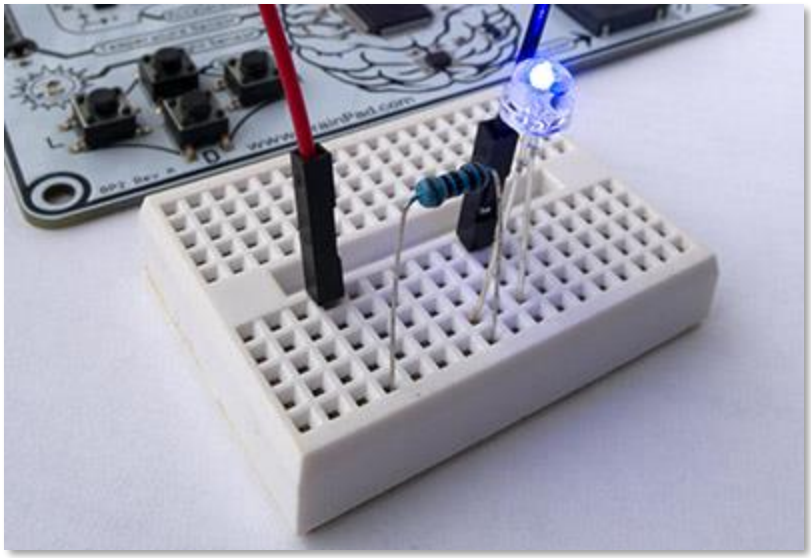
Con el BrainPad, puedes agregar inteligencia y programación a tus proyectos electrónicos. Este proyecto de ejemplo es un contador de tiempo hacia atrás que arroja una hélice. Asegúrate de ver el video. Es uno de nuestros proyectos favoritos (https://docs.brainpad.com/projects/lift-off.html).



BREADBOARDS

¿Quieres diseñar tus propios circuitos electrónicos desde cero como lo hacen los ingenieros? Hay muchas formas de crear circuitos que trabajen con el BrainPad. Mientras que el diseño de circuitos requiere ciertos conocimientos, es la forma más creativa y barata de expandir tu BrainPad. También es la más educativa.

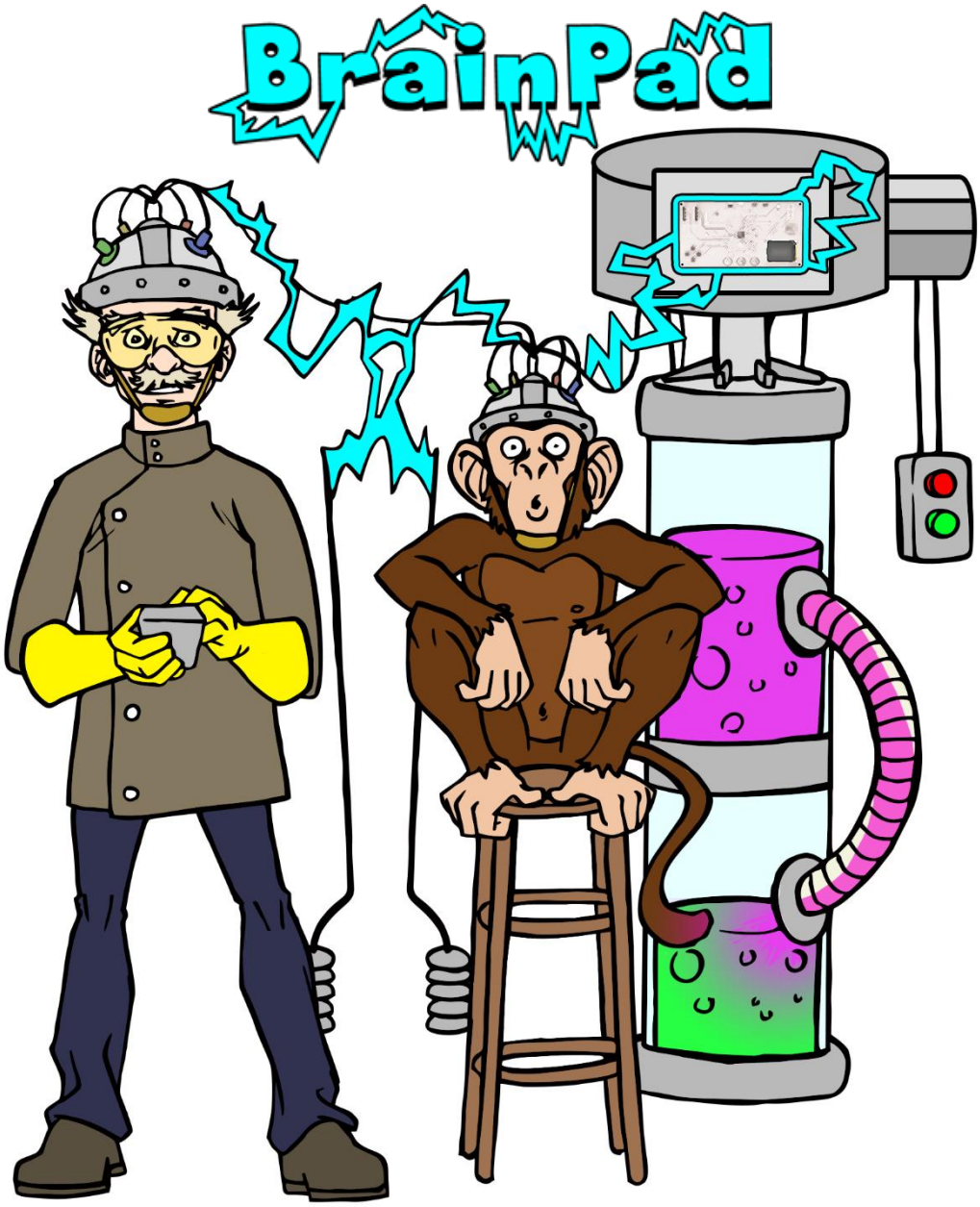
La forma más fácil de comenzar haciendo tus propios circuitos es usar breadboards, ya que no requieren soldadura. Los cables y componentes se conectan directamente en los hoyos del breadboard. El breadboard sostiene los componentes y los conecta eléctricamente. No solo es la forma más fácil de probar un nuevo circuito, sino que también es la forma más simple para corregir errores.



Si desarrollas un circuito que quieres hacer más permanente, hay varias opciones, desde usar placas de prototipo pre-hechas, hasta diseñar tus propias placas de circuito impreso. Los diseños de circuito impreso se pueden dibujar a mano o mediante herramientas de software gratuitas. Esto hace al BrainPad tan especial – es simple para comenzar, pero puedes llegar tan lejos como quieras, aprendiendo a tu propio ritmo.

Puedes también agregar uno o más sensores disponibles a tu proyecto BrainPad. Si buscas en la we “sensor kit”, encontrarás muchos arreglos de sensores de bajo costo. Aquí hay algunos ejemplos:

Se requiere algún conocimiento para usar esos módulos, sin embargo tenemos algunos proyectos fáciles de construir que te ayudará a ir en esa dirección. Una vez que te familiarizas con un tipo de sensor, es más fácil aprender a usar otros sensores.



www.BrainPad.com