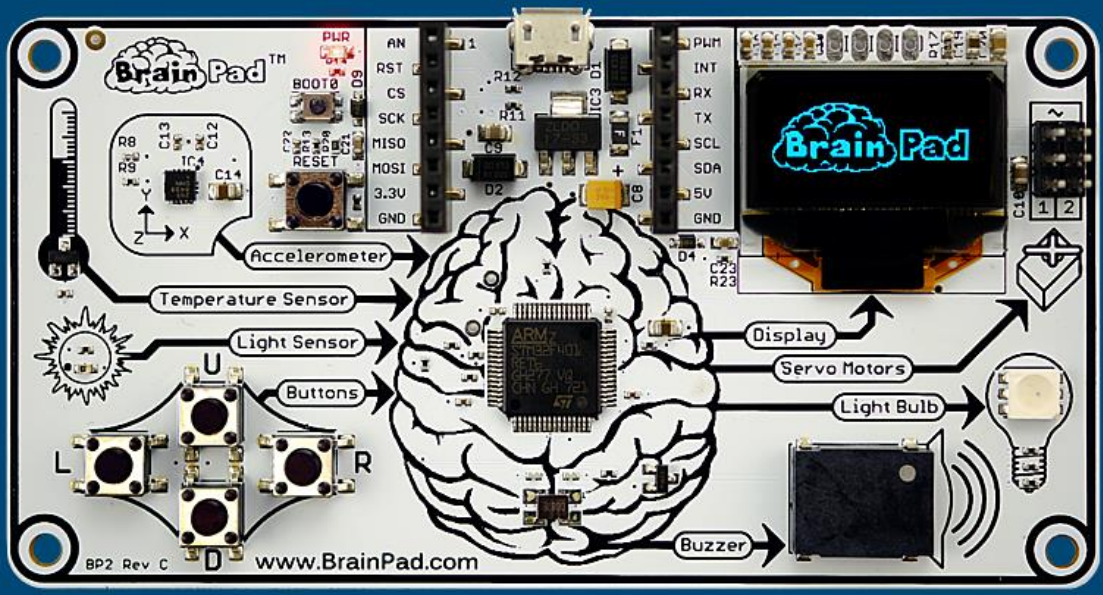
GHI electronics

BrainPad™

BEGINNERS' GUIDE



INHALT

EINFÜHRUNG

Das BrainPad ist ein leistungsfähiges pädagogisches MINT-Tool, mit dem jeder unterrichtet werden kann, von Kindern bis zu Studenten und Profis.

Dieses kostenlose E-Book bietet eine Einführung in die Welt des BrainPads. Weitere, überwiegend in Englisch gehaltene, Materialien findest du auf http://www.brainpad.com.

DER AUTOR

Gus Issa ist Gründer und CEO von GHI Electronics. Er fing als Teenager an zu programmieren und an Elektronik zu basteln. Schon früh kämpfte er damit sich ohne Internetzugang und mit beschränken Ressourcen Dinge selbst beizubringen. Dieser Kampf hat einen tiefgreifenden Einfluss auf sein Leben und ihn dazu gebracht andere leidenschaftlich zu unterrichten. Er hat unzählige Stunden damit verbracht die Ressourcen seines Unternehmens zu nutzen um einen erschwinglichen Weg zu schaffen sein Wissen zu teilen was zum BrainPad führte.

GLOBALES DENKEN

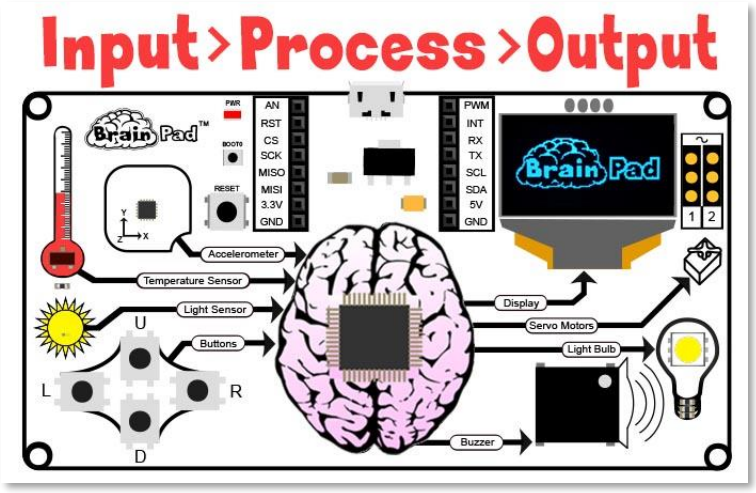
Wir arbeiten intensiv daran, das BrainPad für jede Kultur und Sprache verfügbar zu machen. Wenn du dich der Herausforderung gewachsen fühlst und dieses Buch in weitere Sprachen zu übersetzen, lassen es uns wissen http://www.brainpad.com/contact-us .

Alle aktuell verfügbaren Übersetzungen und das englische Original findest du hier https://www.brainpad.com/start

LIZENZ

DAS KONZEPT

Das BrainPad ist selbsterklärend. Es zeigt logisch wie Computer arbeiten. Es gibt vier Eingänge, die zum Gehirn geleitet werden. Das Gehirn (der Prozessor) denkt dann darüber nach, was passiert, um die vier Ausgänge zu steuern.



MINT

"MINT", bzw. STEM im Englischen, steht für Mathematik, Informatik, Naturwissenschaft und Technik. Das MINT-Akronym wurde 2001 von der U.S. National Science Foundation eingeführt. MINT ist heute eines der am meisten diskutierten Themen in der Bildung. MINT-Bildung ist jedoch mehr als nur diese vier Studienfächer. Der MINT-Bildungsansatz zielt darauf ab, das Lernen im Klassenzimmer mit der realen Welt zu verbinden indem Kommunikation, Kollaboration, kritisches Denken und Kreativität beim Unterrichten miteinbezogen werden.

DIE PHILOSOPHIE

MINT-Bildung erfordert eine sich entwickelnde Plattform. Wie du entwickelt sich auch das BrainPad. Viele Spielzeuge werden als MINT-Werkzeuge vermarktet, aber den meisten fehlt die Vielseitigkeit um die Schüler für längere Zeit zu beschäftigen. Das BrainPad vereint die Programmierung von Elektronik, Maschinenbau, Robotik und mehr zu einer kostengünstigen Plattform. Die von Anfängern bis zum Profi genutzt werden kann. Es stützt sich auf unsere 15-jährige Erfahrung in der Konstruktion von professionellen Steuerungen für die Industrie. Dieselben Tools mit denen kommerzielle Kunden unsere industriellen Steuerungen programmieren können auch für die Programmierung des BrainPad verwendet werden. Keine andere MINT-Plattform kann Studenten dazu bringen, die Blockprogrammierung per Drag-and-Drop in professionelle Programmierung und Engineering wie das BrainPad zu verwandeln.

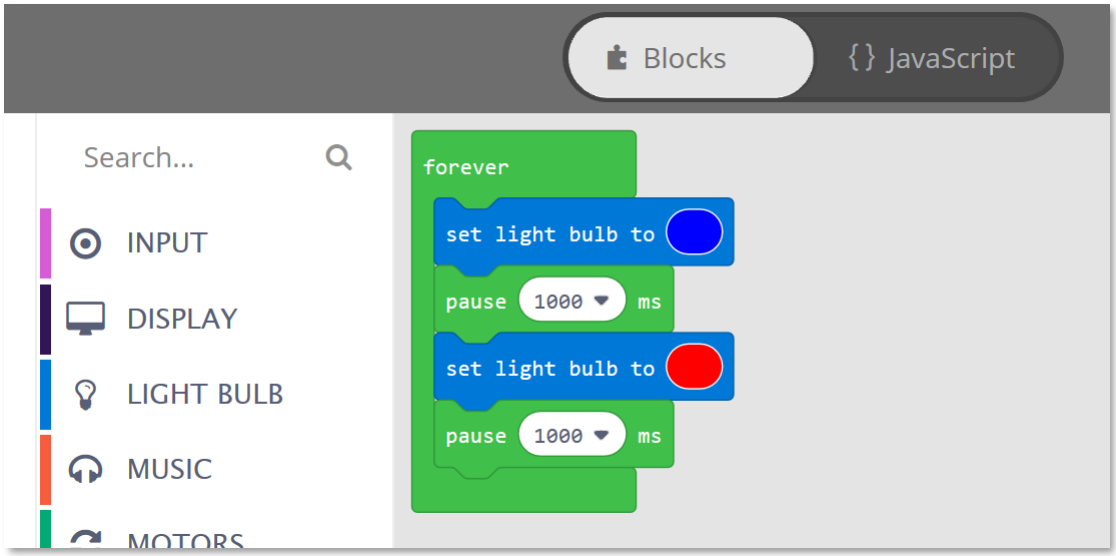
ZWEI WEGE

Das BrainPad bietet zwei Lernpfade "**Fange an**" („**Start Making**“)und „**Gehe weiter**“ („**Go Beyond**“). Wie du vielleicht schon erraten hast, macht die erste Option das Starten von Projekten sehr einfach und die zweite Option führt duch darüber hinaus und ermöglicht es dir, die gleichen Programmierwerkzeuge zu verwenden die auch von Profis verwendet werden.

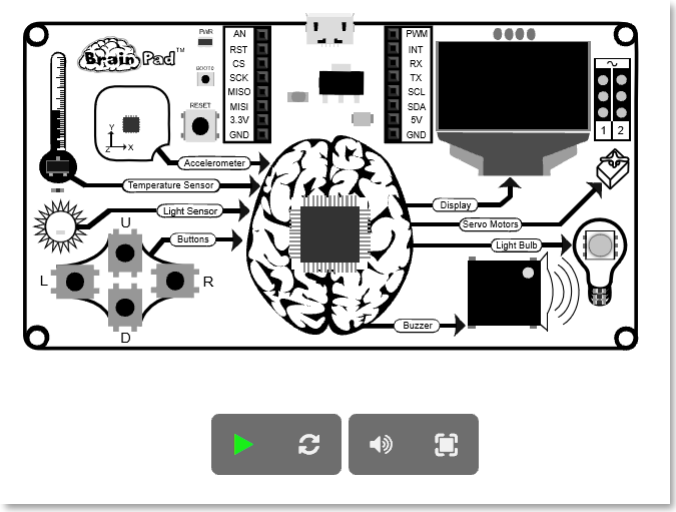
FANGE AN (START MAKING)

Bei „Start Making“ musst du nichts auf deinem Computer zu installieren! Alles funktioniert über den Internet-Browser mit Microsoft MakeCode. Die „Go Beyond“ Option erfordert einige Setup- und Softwareinstallationen wie zum Beispiel Microsoft Visual Studio.

Mit Microsoft MakeCode (Start Making) kann man programmieren, indem man einfach Blöcke anordnet und optional JavaScript-Code direkt in den Webbrowser eingibt. Im folgenden Bild siehst du einen Beispiel Code der eine Lampe abwechselnd eine Sekunde blau und dann rot leuchten lässt. Das Programm wird dann endlos wiederholt.



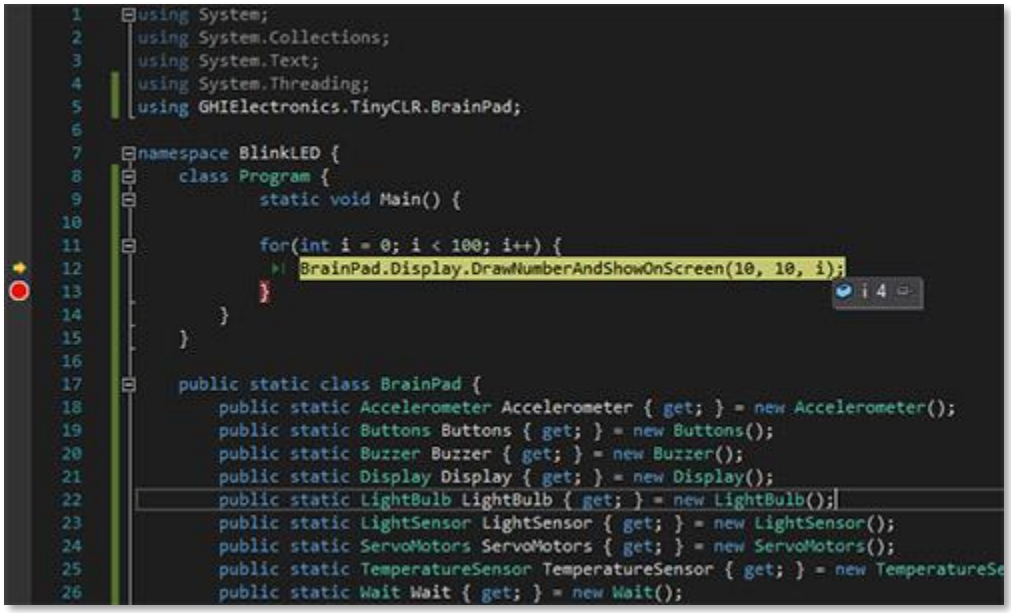
Der mitgelieferte Simulator (siehe unten) ist im Grunde genommen ein virtuelles BrainPad das deine Programme direkt auf deinem Computerbildschirm ausführt. Du kannst es auf der linken Seite des Microsoft MakeCode-Fensters sehen. Mit dem Simulator kannst du deine Programme direkt in deinem Browser ausprobieren, ohne es vorher auf das BrainPad zu laden. Dies ist auch eine hervorragende Option, wenn du noch kein BrainPad hast.



GEHE WEITER (GO BEYOND)

Bei "Go Beyond" werden die Dinge ernster. Du lernst wie du mit Microsoft Visual Studio auf deinem Rechner installierst und einrichtest und dein Brainpad programmierst und den Code debuggst.

Dank GHI Electronics TinyCLR OS™ kann das BrainPad in C# und Visual Basic programmiert werden. Vollständige Debugging-Funktionen sind ebenso enthalten, wie das Schrittweise durchlaufen von Code, Breakpoints und die Überprüfung von Variablen zur Laufzeit. Du programmierst das BrainPad genauso wie Profis einen PC oder eine Smartphone-App programmieren.



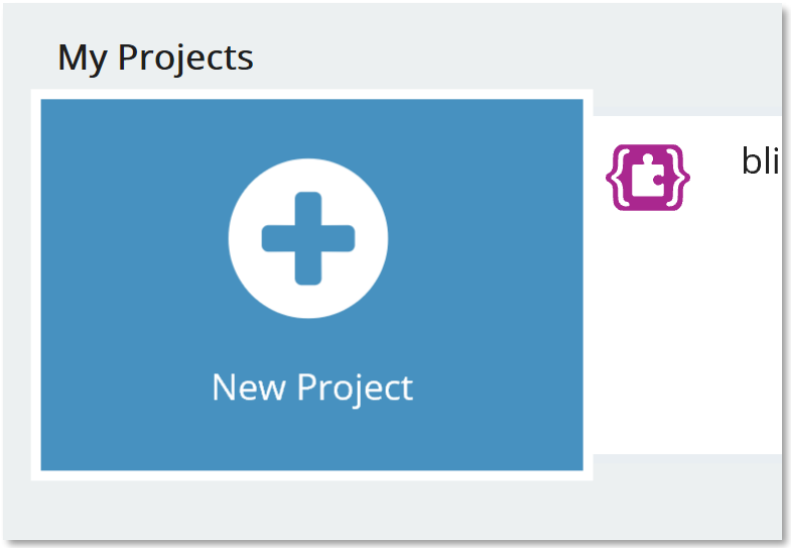
```
using System;
using System.Collections;
using System.Text;
using System.Threading;
using GHIElectronics.TinyCLR.BrainPad;

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();
```

GEHE NOCH WEITER (GOING BEYOND THE BEYOND)

Das BrainPad kann auch mit anderen Tools programmiert werden, die in diesem Buch nicht dokumentiert sind. Zum Beispiel die Arduino-Tools https://www.arduino.cc ,MicroPython https://www.micropython.org und Arm Mbed https://www.mbed.com.

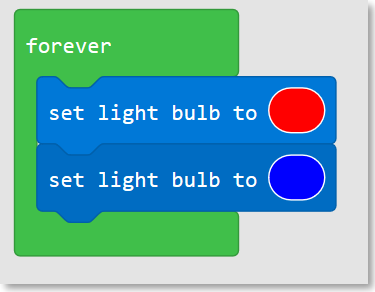
GANGE AN (START MAKING)

Genug geredet - lass uns anfangen Spaß zu haben und ein paar Projekte zu machen! Wir werden Microsoft MakeCode verwenden, also gehe auf https://makecode.brainpad.com/ Die Website enthält eine Menge fertiger Materialien und Projekte, aber wir werden uns einfach daranmachen, ein eigenes Projekt zu starten. Dazu klicke auf “New Project” (*Neues Projekt*).

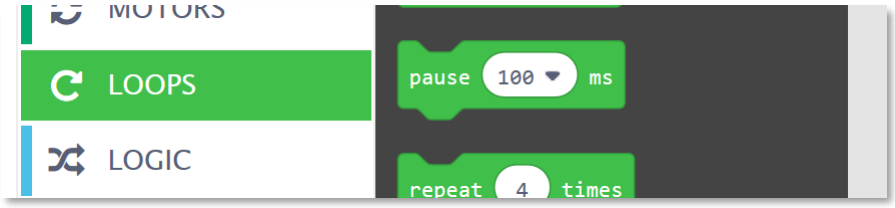


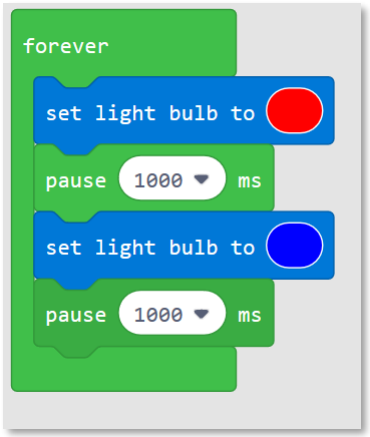
Wählen nun unter der Option “LIGHT BULB” (*„Glühbirne“*) den Menüpunkt "set light bulb to …" und ziehe ihn in den Block "forever" (*„für immer“*). Der forever-Block wird für Code verwendet den das Brainpad ständig ausführen soll. Die Anweisungen innerhalb des forever-Blocks werden immer ausgeführt solange das BrainPad über Strom verfügt. Sobald alle Anweisungen ausgeführt wurden, führt das BrainPad den Code erneut aus, beginnend mit der ersten Anweisung im Block forever. Dies ist als Endlosschleife bekannt und wird häufig in der Computerprogrammierung verwendet.

Wiederhole den gleichen Schritt, aber klicken im zweiten Block auf das Farb-Oval, um die Farbe auf blau zu ändern. Das solltest du bisher haben:



Das Programm, das wir erstellt haben, wird die Glühbirne zuerst auf Rot stellen und dann auf Blau ändern, und das immer wieder! Wenn du jedoch den Simulator betrachtest funktioniert die Glühbirne nicht wie erwartet. Wenn du das Programm auf dem BrainPad lädst funktioniert es auch nicht richtig, aber warum? Die Lösung ist ganz einfach! Die Farben ändern sich viel zu schnell und man kann sie nicht als getrennte Farben erkennen. Wir sollten also nach jedem Farbwechsel eine kurze Pause machen. Ziehe dazu aus dem LOOPS (“Schleifen”) Menü den Block "Pause" an die Position nach jedem „set light bulb to …“.



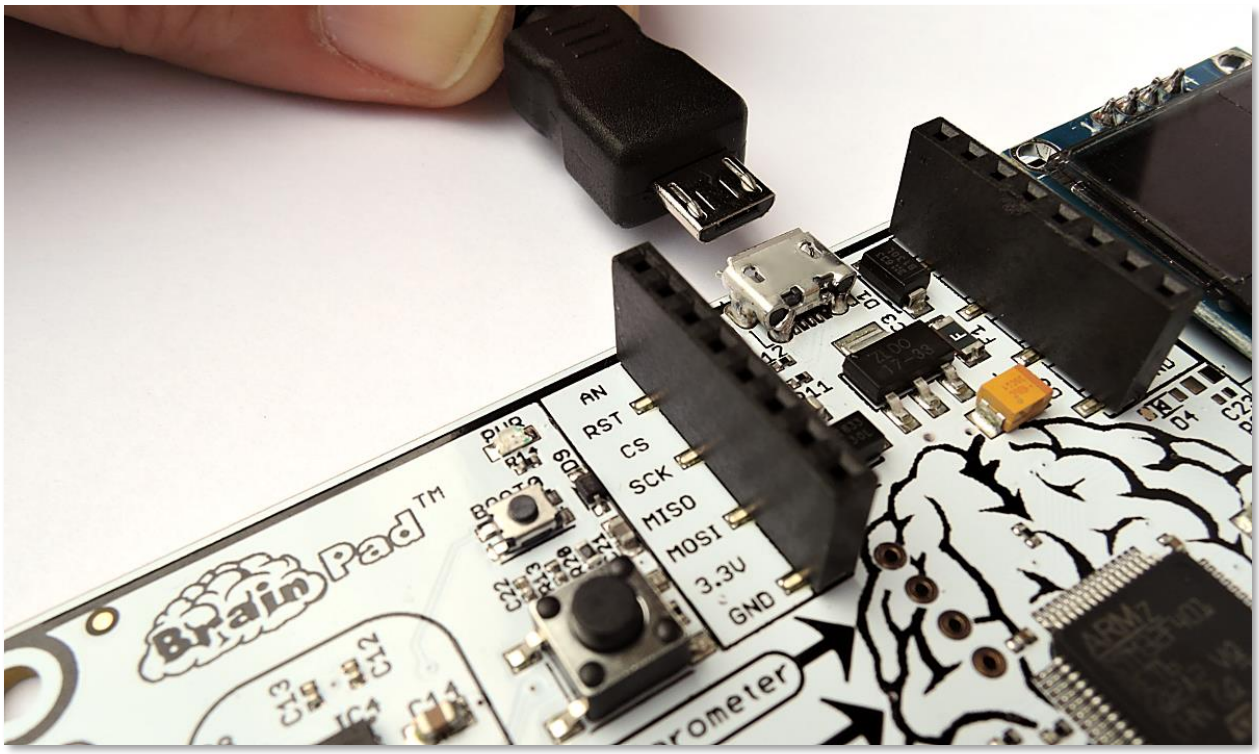


Die Änderung kannst du sofort im Simulator sehen. Nicht schlecht, oder? Wie wäre es, wenn wir dieses Programm jetzt auf das BrainPad laden?

PROGRAMME AUF DAS BRANPAD LADEN

Das Kopieren eines Programms auf das BrainPad ist prinzipiell recht einfach, kann aber beim ersten Mal schwierig sein. Zuerst musst du das BrainPad über ein Micro-USB-Kabel mit deinem Computer, egal ob Windows, Mac oder Chromebook, verbinden.

Ein Micro-USB-Kabel sollte bei deinem BrainPad enthalten sein. Wenn nicht kannst du auch das Micro-USB Datenkabel deines Smartphones verwenden.

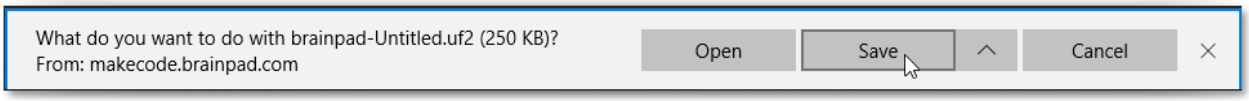


Das kleinere Ende des USB-Kabels wird an das BrainPad angeschlossen.

Das größere Ende des Kabels wird an deinem Computer angeschlossen.

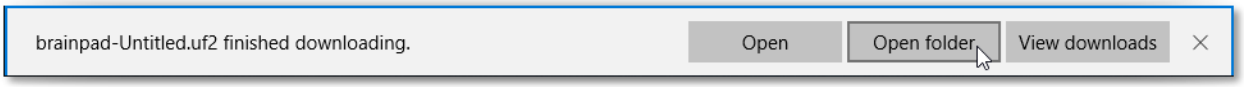
Wenn das BrainPad an deinem Computer angeschlossen ist, sollte die rote Power-LED (PWR) leuchten..

Zurück im Browser klicke dann links unten auf die Schaltfläche Download („Herunterladen“)

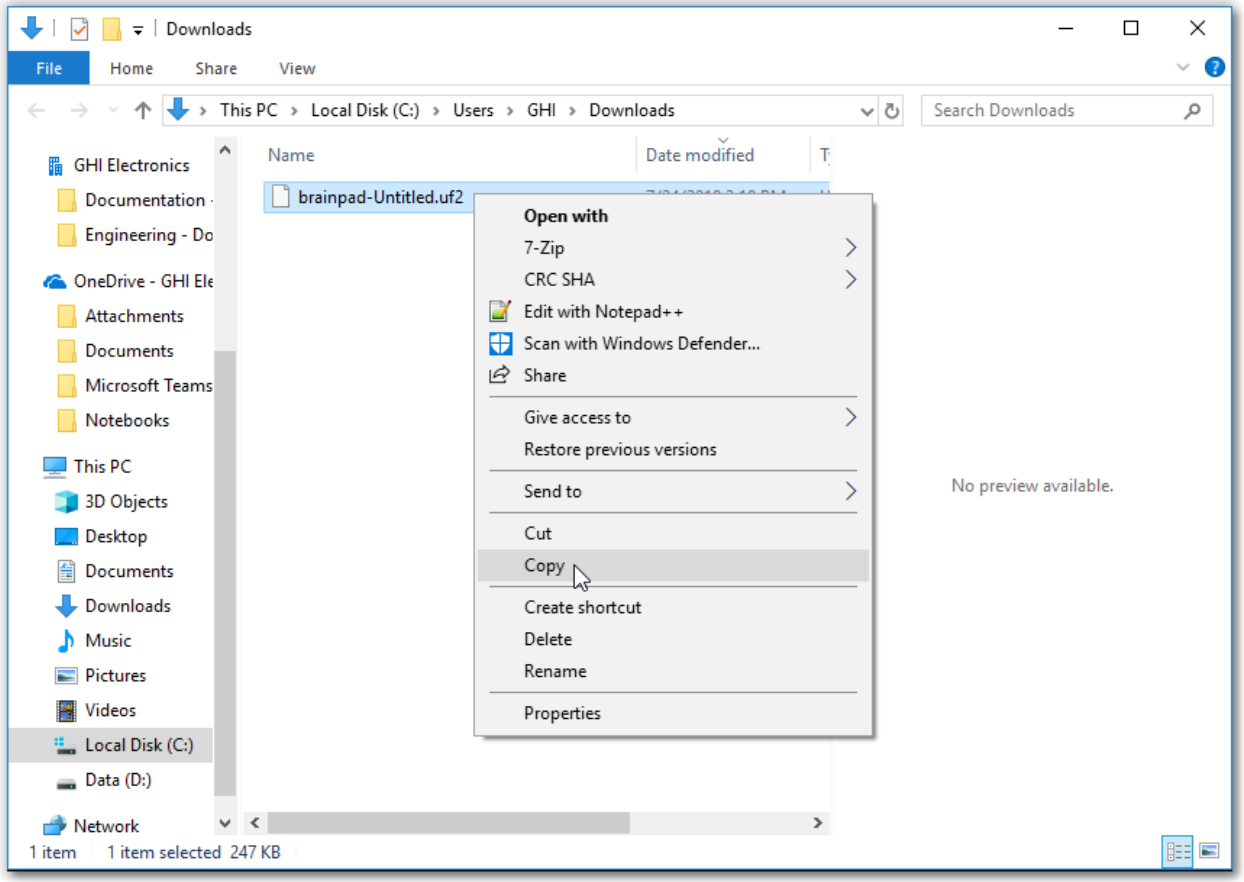


Wenn du Microsoft Edge verwenden, solltest du ein Dialogfeld wie das unten gezeigte sehen:

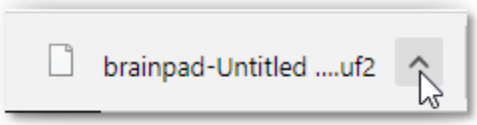
Klicke auf die Schaltfläche Speichern und dann auf die Schaltfläche "Ordner öffnen" im nächsten Dialogfeld, um die



heruntergeladene Datei anzuzeigen.

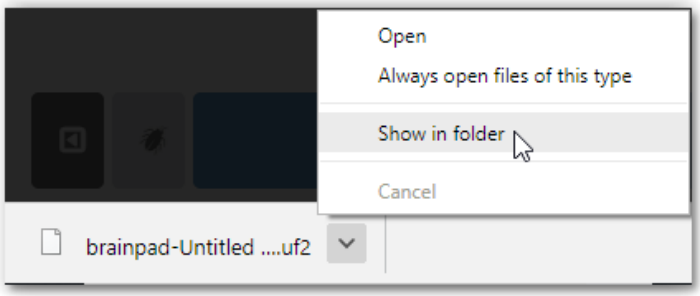


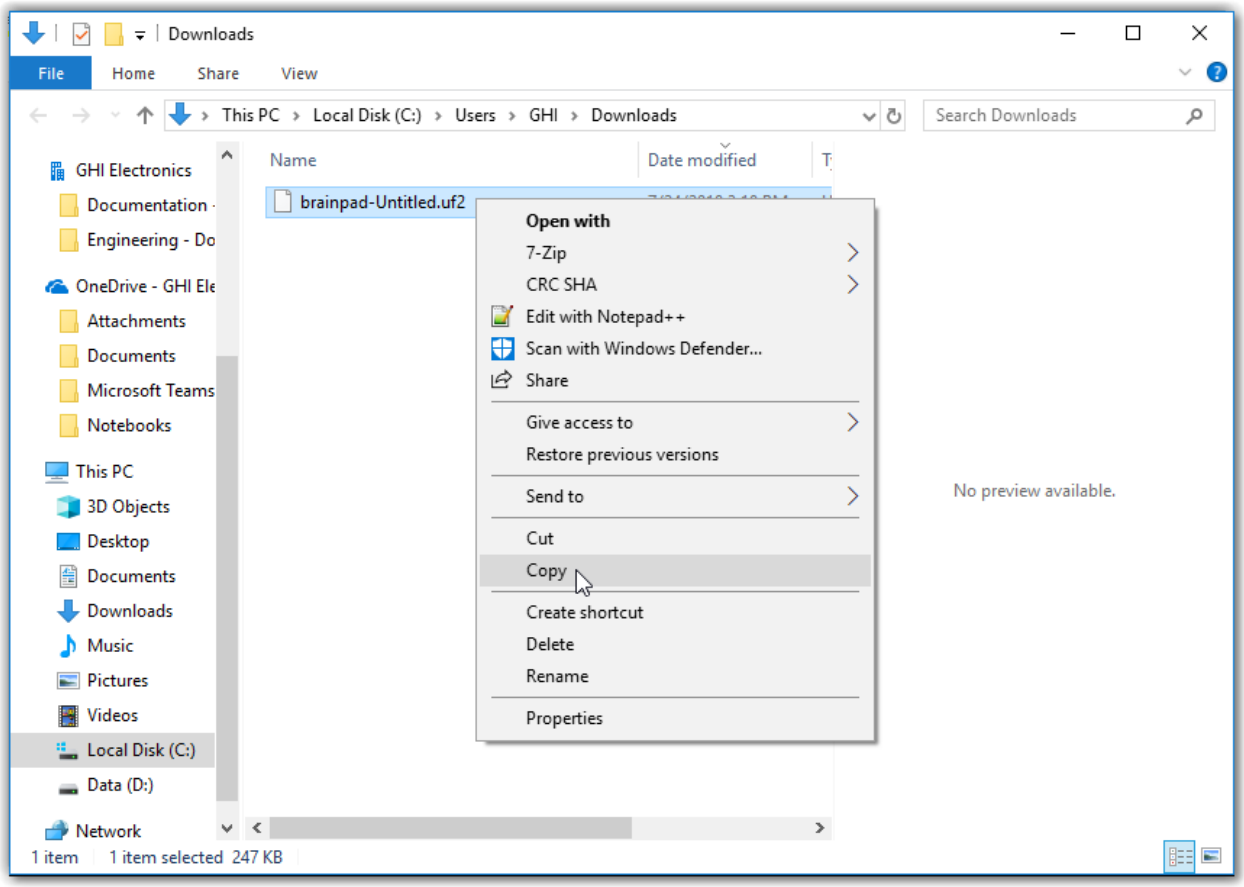
Die Datei wird markiert. Klicke sie mit der rechten Maustaste an und wähle kopieren.



Wenn du den Chrome-Browser verwenden, wird die Datei in der unteren linken Ecke des Bildschirms angezeigt:

Klicke auf den kleinen Pfeil nach oben und wähle "Im Ordner anzeigen"





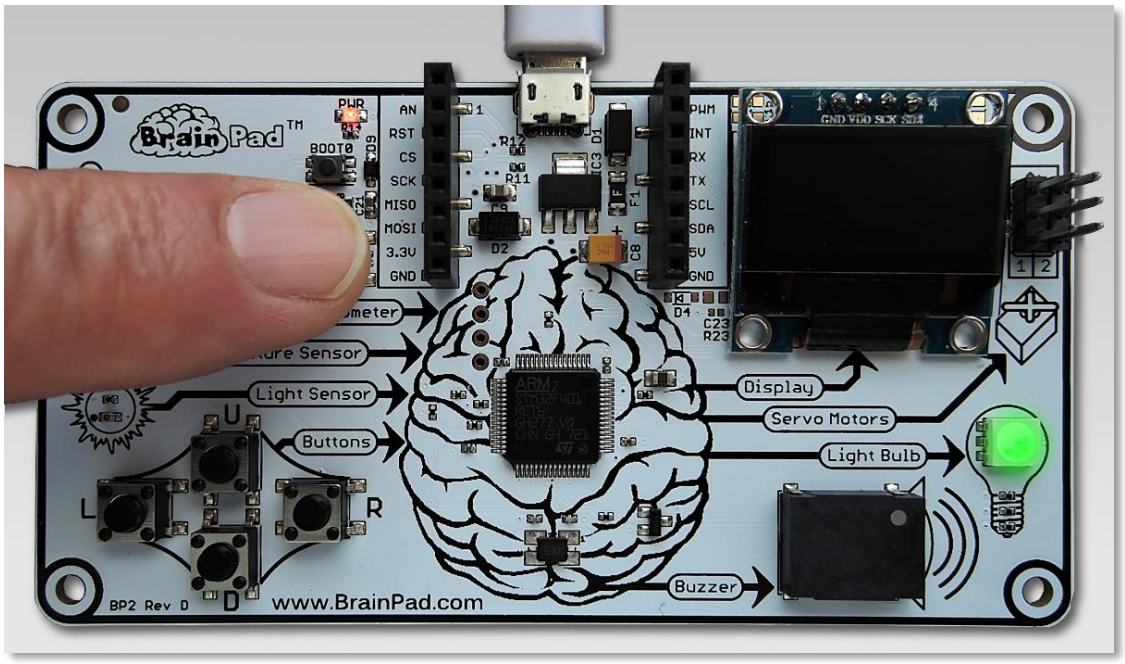
Die heruntergeladene Datei wird hervorgehoben.

Mit beiden Browsern löscht Microsoft MakeCode deine älteren Dateien nicht. Wenn du dasselbe Programm mehrmals speicherst, wird eine Nummer zum Dateinamen hinzugefügt (zum Beispiel "brainpad-Untitled (1) .uf2").

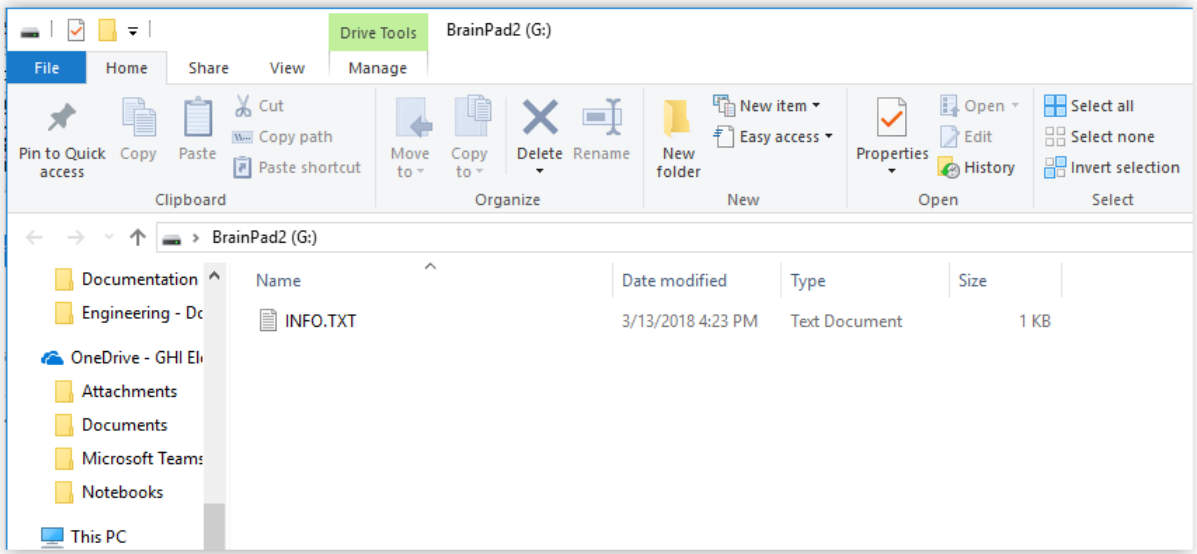
Die aktuellste Datei ist nach dem Herunterladen markiert und hat üblicherweise die höchste Nummer bzw. das späteste Erstellungsdatum.

In anderen Browsern funktioniert der Vorgang ähnlich.

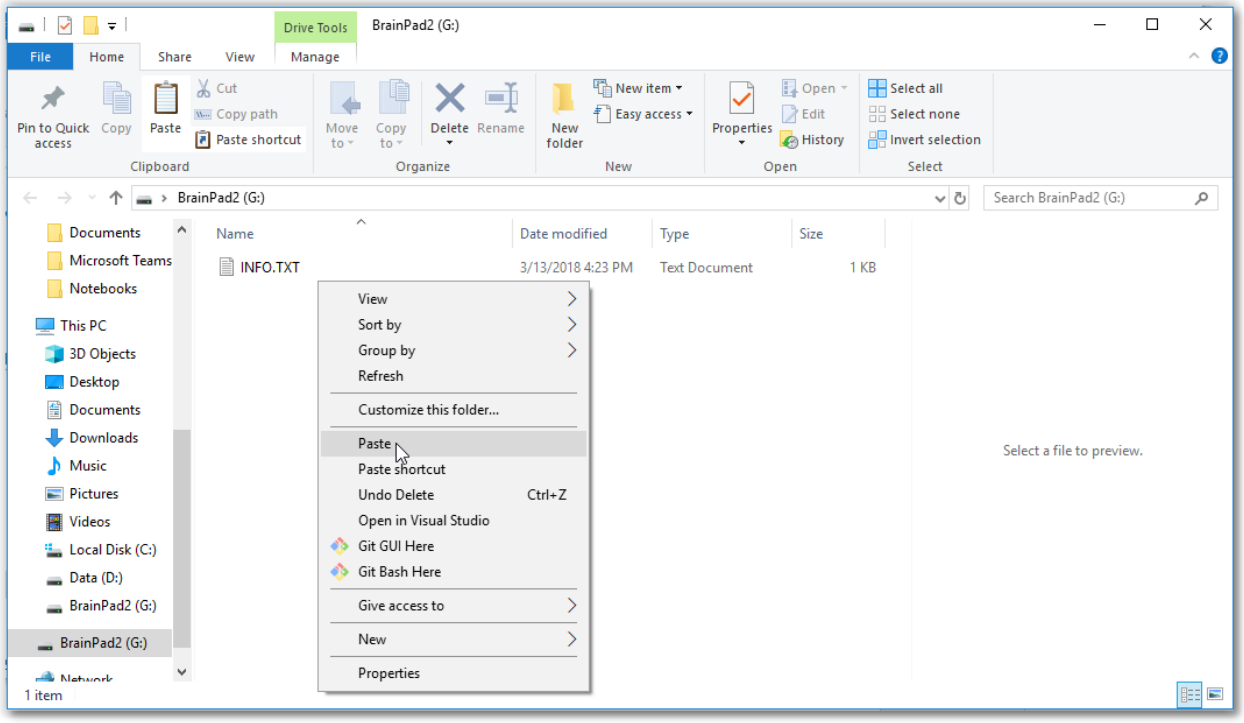
Das BrainPad sollte an deinem Computer angeschlossen sein und die rote Betriebsanzeige sollte leuchten. Halte die Reset-Taste etwa drei Sekunden lang gedrückt. Die Glühbirne wird grün.



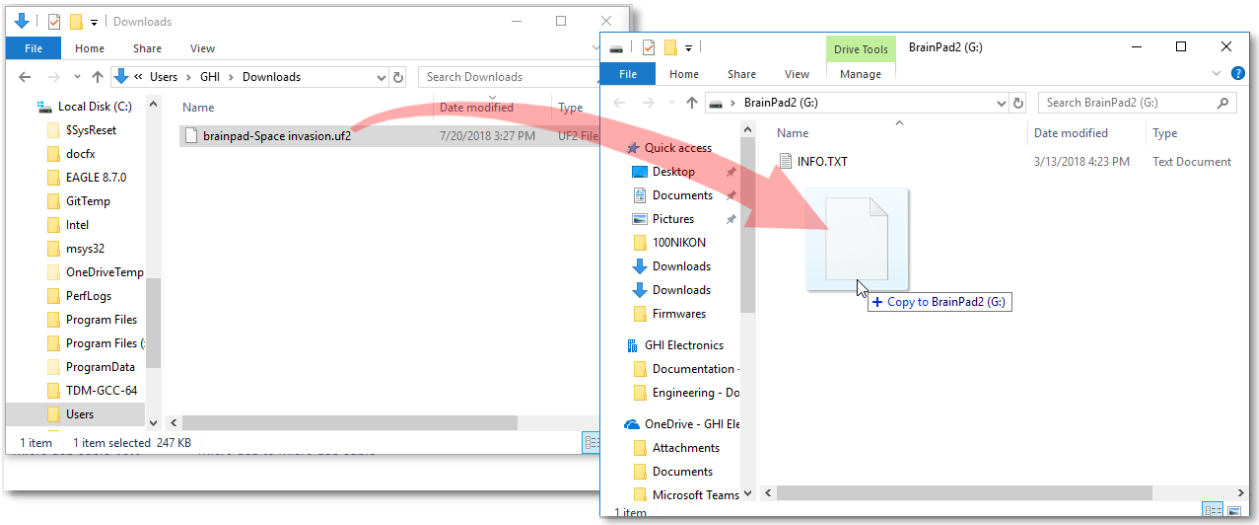
Dein Computer erkennt nun ein USB-Speichergerät. Es kann eine Minute dauern bis das neue Laufwerk mit dem Namen Brainpad erscheint. In den meisten Fällen öffnet sich auch gleich ein Fenster mit dem Inhalt des neuen Laufwerks. Du kennst diesen Vorgang vielleicht vom anstecken eines USB Sticks oder einer Speicherkarte.

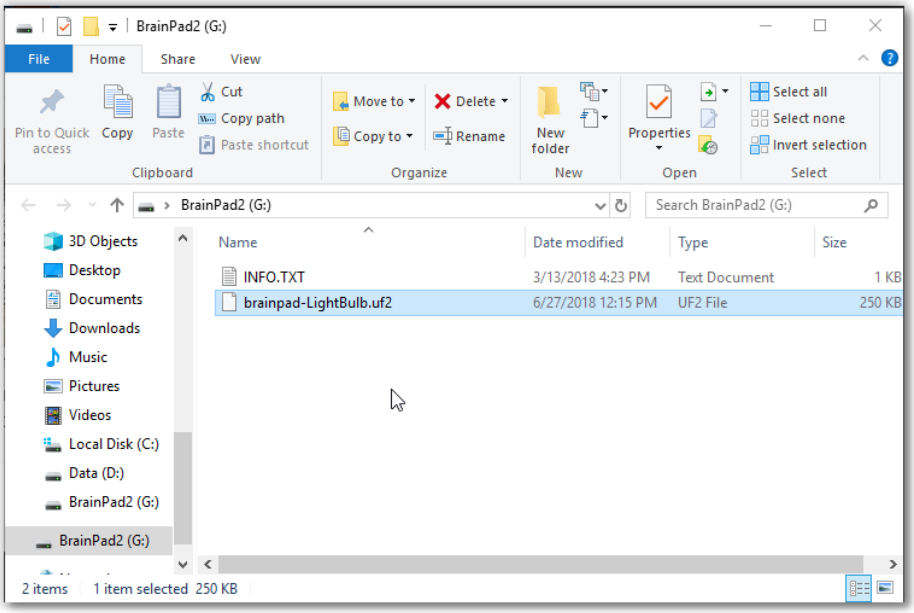


Dieses Fenster zeigt den Inhalt des Speichers im BrainPad. Klicken nun mit der rechten Maustaste in das Fenster und wähle Einfügen um die Datei in das BrainPad zu kopieren.



Du kannst die heruntergeladene Datei auch per Drag & Drop in das BrainPad-Fenster ziehen.

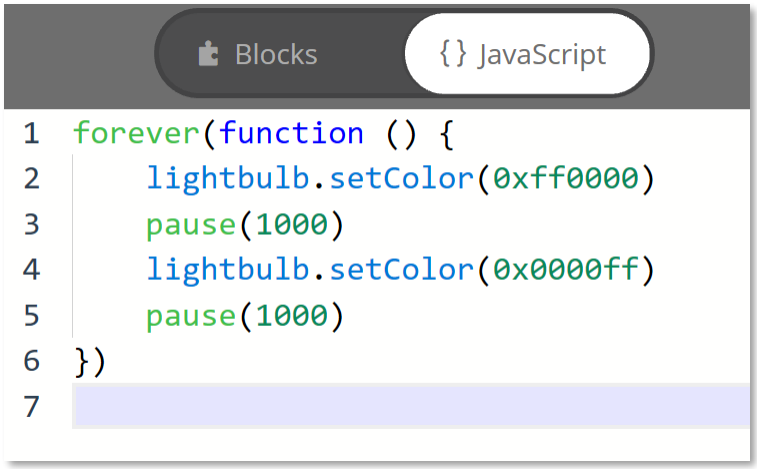




Nach dem Kopieren flackert die Glühbirne für einen Moment und dann wird das geladene Programm vom BrainPad ausgeführt. Jetzt werden die Dinge interessanter!

JAVASCRIPT

Blöcke sind am Anfang lustig, aber sobald deine Programme komplexer werden musst du wahrscheinlich Code schreiben. Aber noch nicht jetzt! Eine der erstaunlichen Funktionen von Microsoft MakeCode ist die Umwandlung von Blöcken in JavaScript-Text-basierten Code und Text-basierten Code zurück in Blöcke. Du kannst also jederzeit ansehen wie deine Blöcke in Javascript Code aussehen und umgekehrt. Kehre zu dem Programm zurück, das wir zuvor erstellt haben, und klicken Sie auf die Registerkarte JavaScript.



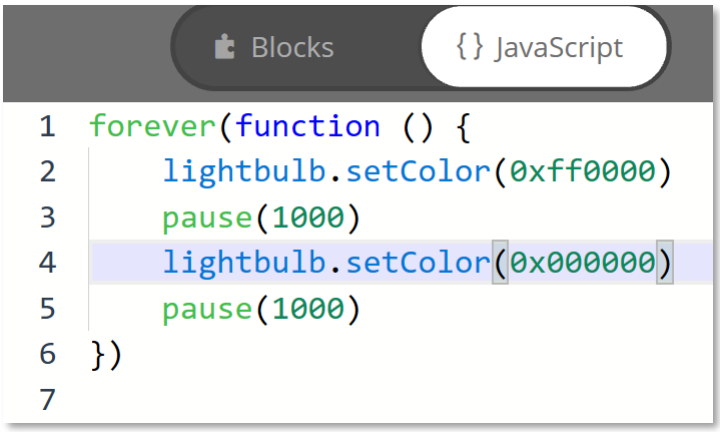
Sieh dir den Code an! Siehst du die Verbindung zwischen Blöcken und Code? Die Farben werden als hexadezimalen Wert festgelegt der Rest des Codes sollte aber verständlich sein.

Das Hexadezimale Zahlensystem ist ein Zahlensystem das auf sechzehn Werten pro Stelle anstelle der üblichen 10 basiert. Jede Ziffer in einer hexadezimalen Zahl kann von 0 bis F anstelle von 0 bis 9 für Dezimalzahlen (die Zahlen, die wir jeden Tag verwenden) gehen. Ein hexadezimaler Wert von 0 bis 16 würde wie folgt aussehen: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10.

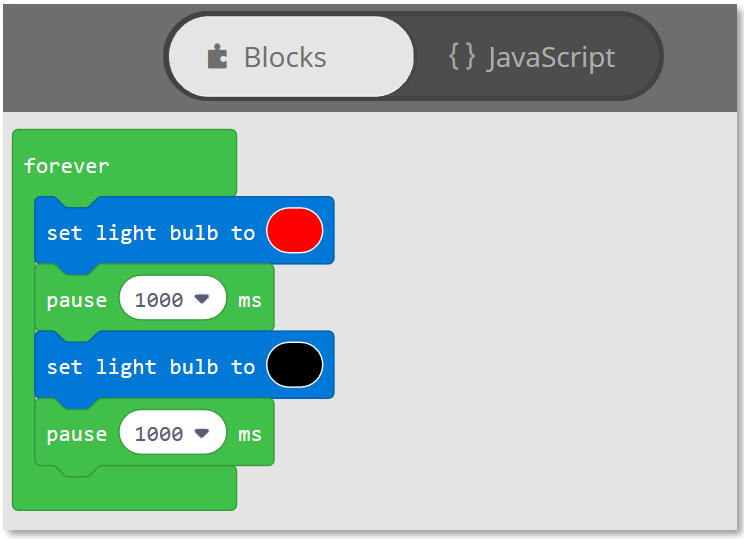
Um Verwechslungen zu vermeiden setzt man ein "0x" am Anfang der Hexadezimalen Zahl. Hexadezimale Zahlen verwenden als Basis 16. Ein weiteres Zahlensystem das beim Programmieren wichtig ist, ist das Binäre System das die Basis 2 hat. Also nur die Werte 0 und 1 verwendet. Das Zahlensystem das du sicher kennen ist das Dezimalsystem mit der Basis 10 also den Werten 0 bis 9 Es braucht sicher einige Zeit, um sich mit verschiedenen Zahlensystemen vertraut zu machen.

Wenn eine hexadezimale Zahl verwendet wird um auf eine Farbe zu verweisen, stellen die ersten beiden Ziffern die Menge an Rot in der Farbe dar, die zweiten zwei Ziffern die Menge an Grün und die dritten zwei Ziffern die Menge an Blau. "00" bedeutet, dass die Farbe eine Intensität von 0 oder ausgeschaltet ist. "FF" bedeutet, dass die Farbe in voller Intensität ist. Hellrot wäre 0xFF0000 Weiß ist 0xFFFFFF

Was passiert, wenn du eine der "setColor" -Methoden im Javascript auf 0x000000 änderst? Lassen es uns einfach versuchen.



Wechsle jetzt wieder in den Blöcke Modus…

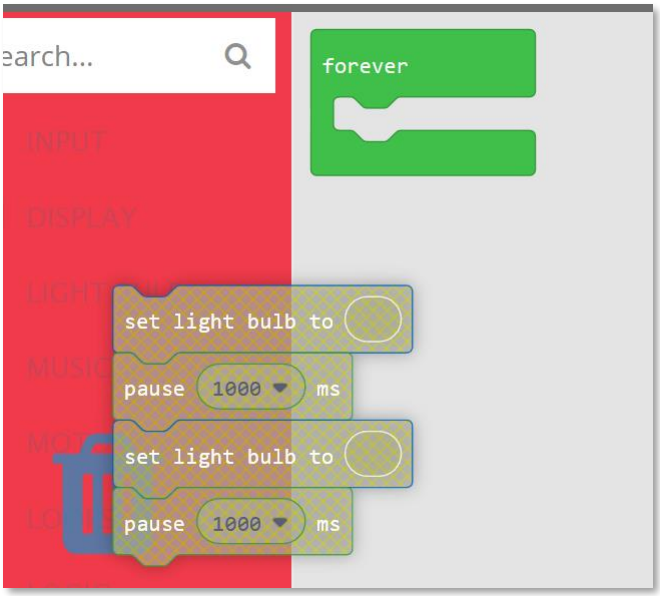


Die Farbe in der zweiten Anweisung hat sich von Blau zu Schwarz geändert. Wie du vielleicht richtig vermutet hast bedeutet Schwarz das die Lampe ausgeschaltet ist. Du kannst es auch im Simulator sehen. Die Glühbirne sollte für eine Sekunde rot und dann für eine Sekunde ausgeschaltet werden.

DISPLAY FUN

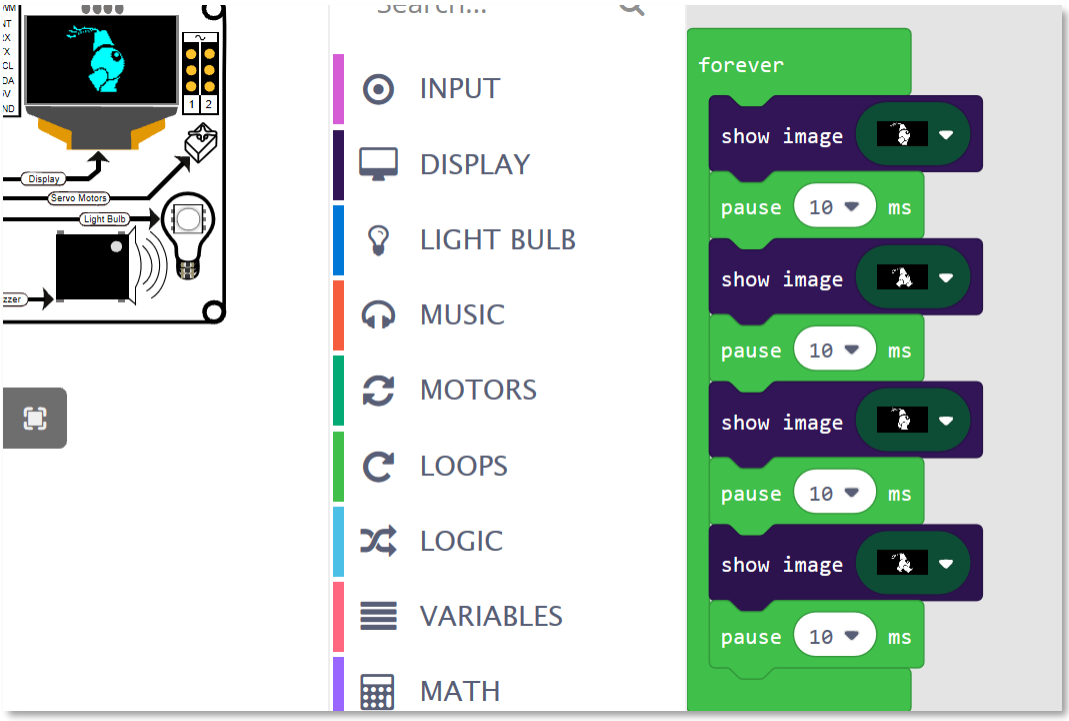
Wenn du glaubst, dass die Glühbirne aufregend war warte bis du das Display ausprobiert hast!

Wenn du immer noch die Blöcke aus dem vorherigen Beispiel hast ziehe sie zurück in das Menü um sie zu löschen.



Wenn du versehentlich den Block „forever“ gelöscht hast kannst du ihn aus dem LOOPS Menü zurückholen.

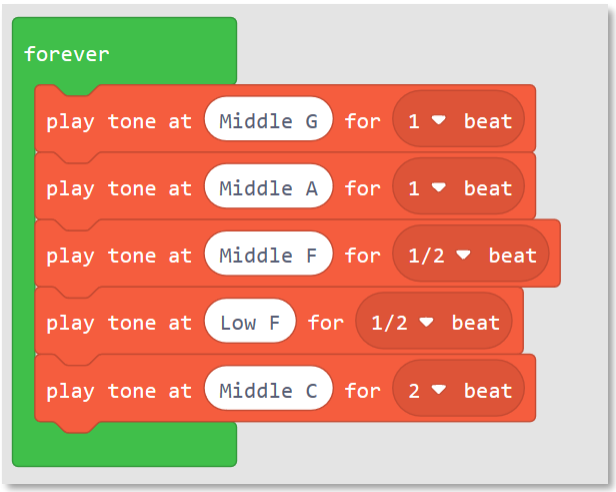
Ziehe den Block "Show Image" („Bild anzeigen“) aus dem DISPLAY-Menü in die Schleife und wähle den ersten der vier "Walking Man" - Bilder aus. Fügen danach die anderen drei Bilder hinzu und dazwischen eine Verzögerung von 10 ms ein. Es gibt keine 10ms-Option im Auswahlmenü aber du kannst den Wert über die Tastatur eingeben.



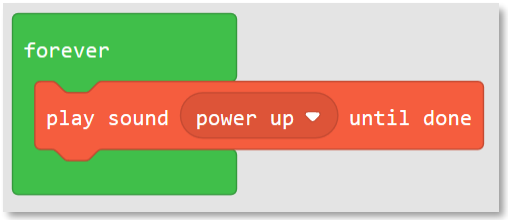
Unser Typ sollte jetzt auf dem Bildschirm des Simulators laufen. Du kannst das Programm wie zuvor gezeigt herunterladen und auf das BrainPad kopieren

MUSIK UND NOTEN ABSPIELEN

Im Menü MUSIC findest du Noten ….



. . .und fertige Sounds

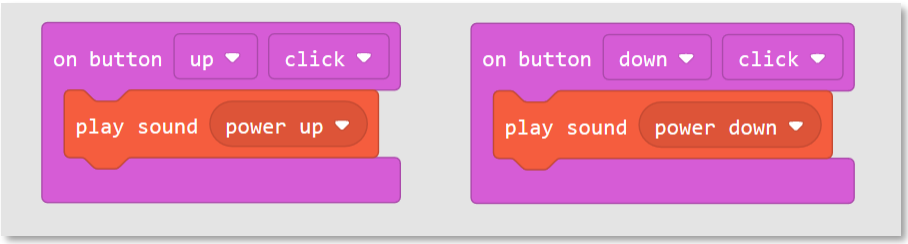


Beachte das wir den Block "play sound **until done**" (*„Sound abspielen bis fertig“*) verwendet haben. Dieser Block stoppt den Rest des Programms bis der Sound fertig abgespielt ist. Der "play sound" Block (ohne "until done") startet den nächsten Block, während der Sound noch läuft. Wenn du beide Blöcke ausprobierst wirst du den Unterschied schnell merken.

SCHALTFLÄCHEN BZW TASTEN (BUTTONS)

Es gibt zwei Möglichkeiten den Zustand der Tasten zu lesen. Eine besteht darin, einfach zu prüfen, ob die Taste gerade jetzt gedrückt ist und wenn ja dass eine entsprechende Aktion ausgeführt wird. Das Problem mit dieser Methode besteht darin, dass wir die Taste permanent überprüfen müssen um sicherzustellen dass wir keinen Knopfdruck verpassen. Eine andere Möglichkeit besteht darin, das System dies für uns tun zu lassen und im Anlassfall darüber benachrichtigt zu werden. In der Software Entwicklung spricht man hier von einem „event“ bzw. auf Deutsch von einem „Ereignis“

Am besten lässt sich das mit einem Beispiel zeigen. Wir fügen jetzt zwei „on button … click“ events aus dem Block INPUT ein und lassen verschiedene Töne abspielen.



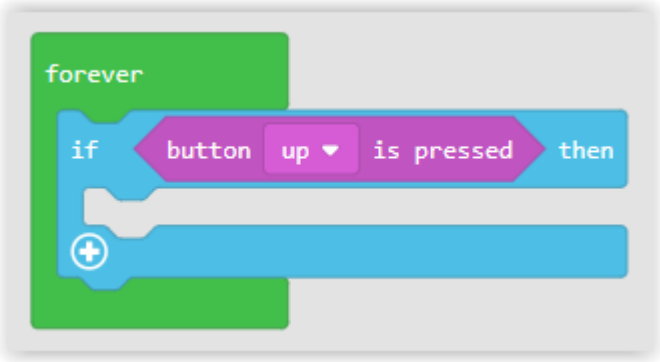
Hast du bemerkt, dass es in diesem Beispiel keinen Block „forever“ gibt? Du kannst immer noch einen Block „forever“ in deinem Code haben, aber für dieses Beispiel wird er nicht benötigt. Dies liegt daran, dass das System das "event" automatisch behandelt, wenn die Taste gedrückt wird.

Hier ist der JavaScript-Code der die oben genannten Blöcke repräsentiert:

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

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

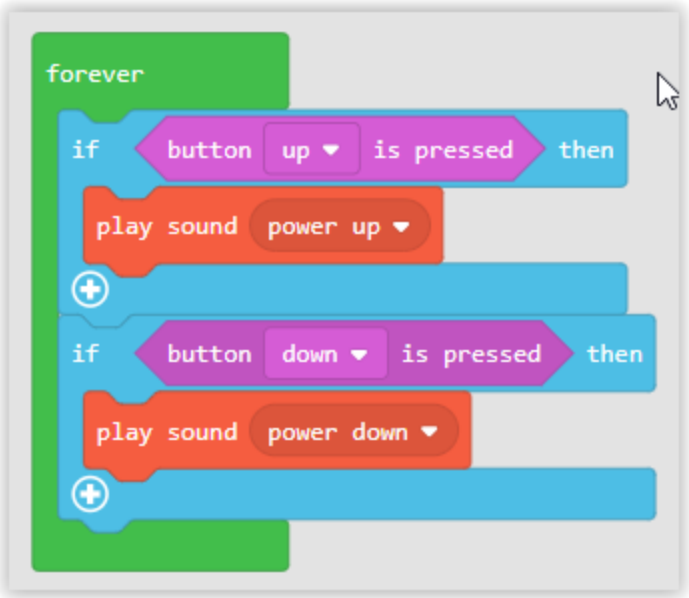
Das hat gut funktioniert aber lass uns auch noch die andere Variante anschauen. Wir brauchen dazu eine „forever“ Schleife und einen Block der „Wenn die Taste gedrückt ist dann ….“ repräsentiert. Da makecode.brainpad.com derzeit nur in Englisch verfügbar ist heißt es dann if … then ….. Diesen Block findest du im „LOGIC“ Menü.

Die „if“ Anweisung prüft den Status von etwas, das wahr oder falsch sein kann. Standardmäßig wird "true" angezeigt, aber wir möchten dies durch den Status der Taste ersetzen. Ziehe den "button .. is pressed" („Taste ist gedrückt“) Block vom INPUT Menü zum "if.. Then" -Block, wie unten gezeigt. Beachte dass dies nicht das event "on button" ist, das wir zuvor verwendet haben.



Die oben genannten Blöcke überprüfen die Hoch-Taste immer wieder. Wenn die Schaltfläche zum Hochdrücken gedrückt wird, wird das Programm alle Blöcke innerhalb der If-Anweisung ausführen.

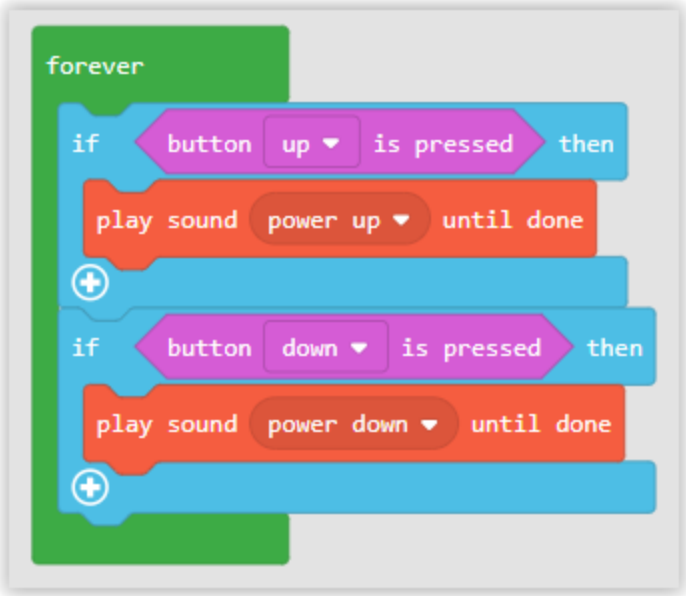
Füge nun weitere Blöcke hinzu um ein Programm wie unten gezeigt zu erstellen:



Wenn du dieses Programm ausführst wirst du bemerken dass der Sound nicht richtig wiedergegeben wird, bis du die Taste loslässt. Der Grund dafür ist, dass jedes Mal, wenn das BrainPad den gedrückten Auf- oder Ab-Knopf erkennt, ein neuer Ton abgespielt wird selbst wenn es in der Mitte des vorherigen Tones ist. Das BrainPad beginnt den Ton also mehrmals pro Sekunde zu spielen während die Taste gedrückt wird.

Es gibt zwei Möglichkeiten das zu beheben. Eine Möglichkeit ist das Ersetzen der "button... **is** pressed" („Taste… ist gerückt“) Blöcke mit „Button… **was** pressed“ (Taste... wurde gedrückt"). Der „Taste wurde gedrückt“ Block ist wahr, wenn der Knopf mindestens einmal seit dem letzten Knopfdruck gedrückt wurde. Auch wenn die Taste mehrmals gedrückt wurde wird sie nur als einzelner Tastendruck registriert. Durch Drücken der Aufwärts- oder Abwärtstaste während der Wiedergabe wird der Ton unterbrochen aber der Ton wird nicht wiederholt solange die Taste gedrückt gehalten wird..

Die andere Möglichkeit besteht darin, den "play sound. . ." durch "play sound. . . until done" zu ersetzen. Der „play sound. . . until done“ Block verhindert, dass das Programm etwas anderes tut bis der Sound fertig abgespielt ist. Dies verhindert, dass das Programm nach dem Drücken einer Taste sucht oder andere Sounds spielt, bis der aktuelle Sound vollständig abgespielt wurde.



IST ES DUNKEL?

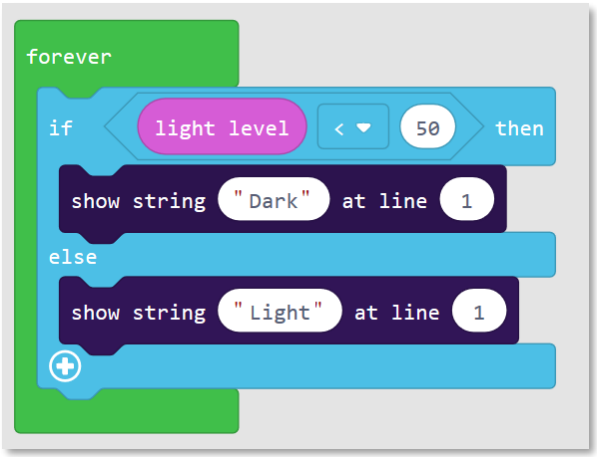
Genauso wie die Tasten kannst du die Lichtstärke auslesen und etwas damit machen oder du kannst ein Ereignis auslösen das anzeigt ob es gerade hell oder dunkel ist. Das Lesen einer Taste unterscheidet sich jedoch geringfügig von Auslesen des Lichtwertes.

Eine Schaltfläche ist entweder gedrückt oder nicht gedrückt. Also wenn ich frage „Ist der Knopf gedrückt?“ Kann die Antwort nur Ja bzw. Wahr (true) oder Nein bzw. Falsch (false) sein. Bei dem Lichtsensor gibt der Sensor das Lichtniveau als Wert an. Lass uns die Lichtintensität ablesen und auf dem Bildschirm zeigen. Wir wollen dies in einer „forever“ Schleife machen die ständig das Lichtniveau ausliest und auf dem Bildschirm ausgibt.

Da du ein Experte in der Verwendung der if-Anweisung bist kannst du ja schon auf dem Bildschirm je nach Lichtsensor-Anzeige „hell“ und „dunkel“ schreiben? Wenn nicht kannst du es nach dem nächsten Beispiel.

Die Schwierigkeit hier ist das die „If“ Anweisung etwas möchte das Wahr (true) oder Falsch (false) ist Das gemessene Lichtniveau ist aber nicht Wahr oder Falsch sondern einer von vielen Werten Du musst also überprüfen ob der Wert heller oder dunkler als ein bestimmter von dir festgelegter Wert ist. Gehe zum LOGIC-Menü und füge einen Vergleich hinzu. Deine „If“-Anweisung wird nun auch eine Sonst (else) Anweisung benötigen.

Um Text auf dem Display auszugeben verwendet man „show string xxx at line …“ („Zeige Text xxx in Zeile …“)

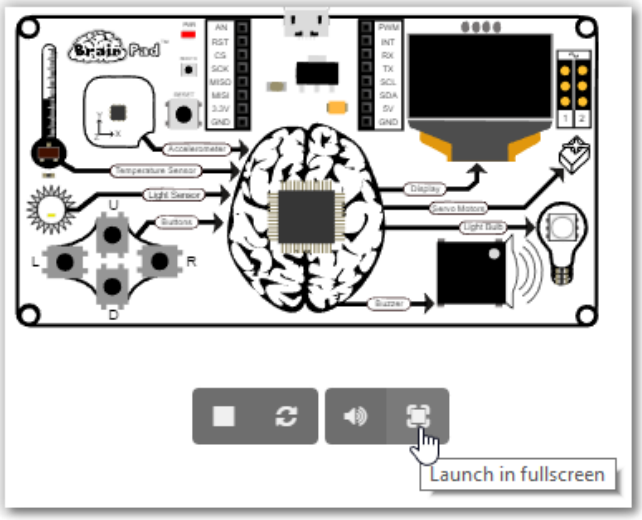


Wenn die Lichtstärke unter 50 ist dann schreibt er “Dark” auf das Display. Bei 50 oder höher sollte „Light“ erscheinen.

Die Blöcke sehen im Javascript Code so aus:

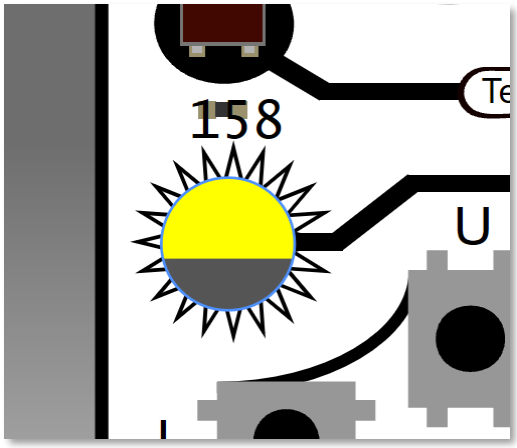
```
forever(function () {
    if (input.lightLevel() < 50) {
        display.showString("Dark", 1)
    } else {
        display.showString("Light", 1)
    }
})
```

Der Simulator funktioniert auch mit dem Lichtsensor. Sobald der Lichtsensor einem Projekt hinzugefügt wurde, zeigt der Simulator eine Option zum Einstellen der Lichtstärke an. Beginne mit dem Vergrößern des Simulators



indem du auf den Vollbild-Button („ Launch in fullscreen“) klickst.

Der Lichtsensor zeigt die Lichtstärke an, durch Ziehen der Linie nach oben und unten kann diese geändert werden. Dies ist vergleichbar mit dem Abdecken des eigentlichen Lichtsensors am BrainPad mit deiber Hand. Wenn du das Licht auf und ab bewegst sollte der Bildschirm abwechselnd "Hell" und "Dunkel" anzeigen.



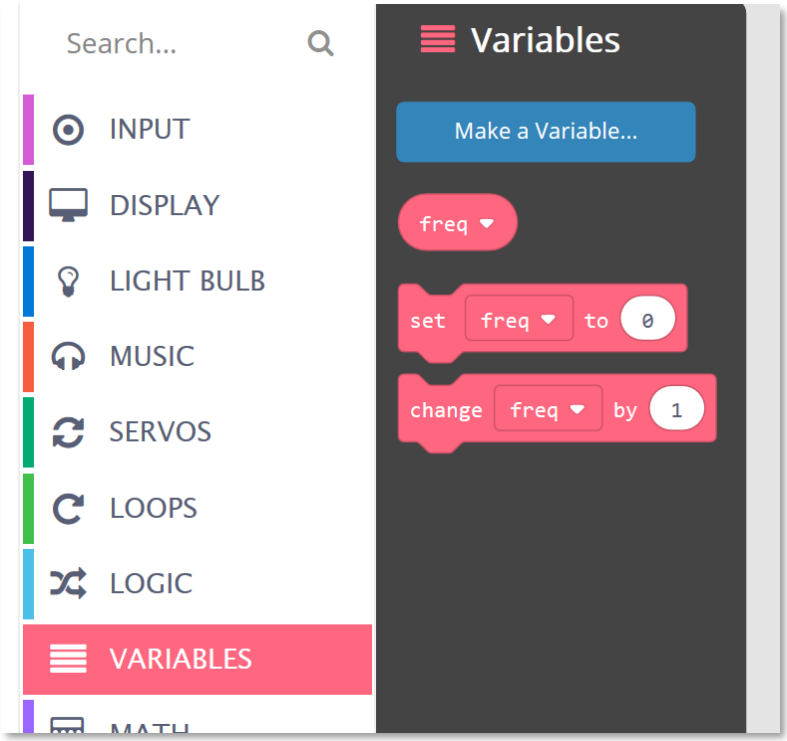
HÖRTEST

Das menschliche Gehör kann Frequenzen zwischen ungefähr 20 Hertz (oder Zyklen pro Sekunde) bis zu ungefähr 20.000 Hertz hören. Hunde können höhere Frequenzen bis 44.000 Hz hören. Deshalb hören wir keine Hundepfeifen hören, Hunde und andere Tiere aber schon! Mit dem Alter nimmt die Fähigkeit ab hohe Frequenzen zu hören. Junge Menschen hören zwar nicht so gut wie Hunde können aber höhere Frequenzen hören als ältere Menschen.

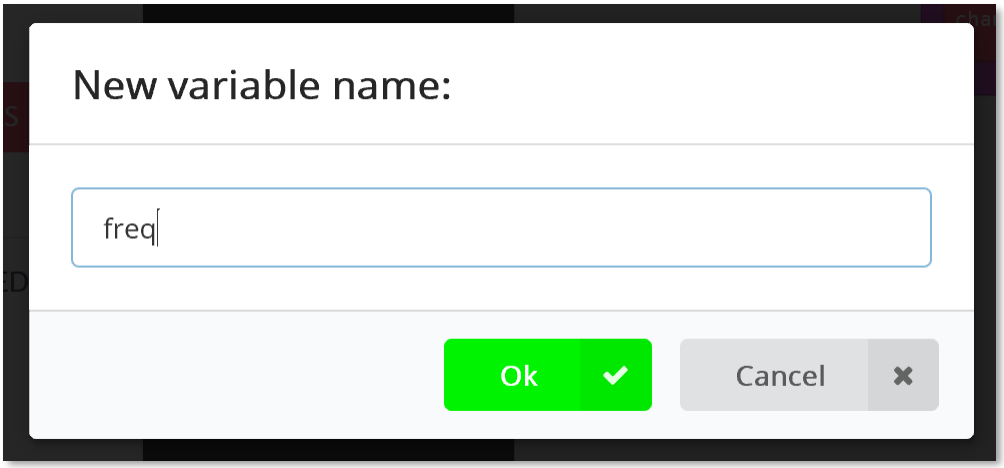
Wir möchten also ein Programm schreiben um unsere Hörfähigkeit zu Testen also wie hoch wir Frequenzen noch wahrnehmen können.

In diesem Beispiel werden wir das erste mal Variablen benutzen. Variablen benötigt man in der Programmierung um sich Werte zu merken. Wie der Name schon sagt sind die Werte „variabel“ können also vom Programm geändert werden.

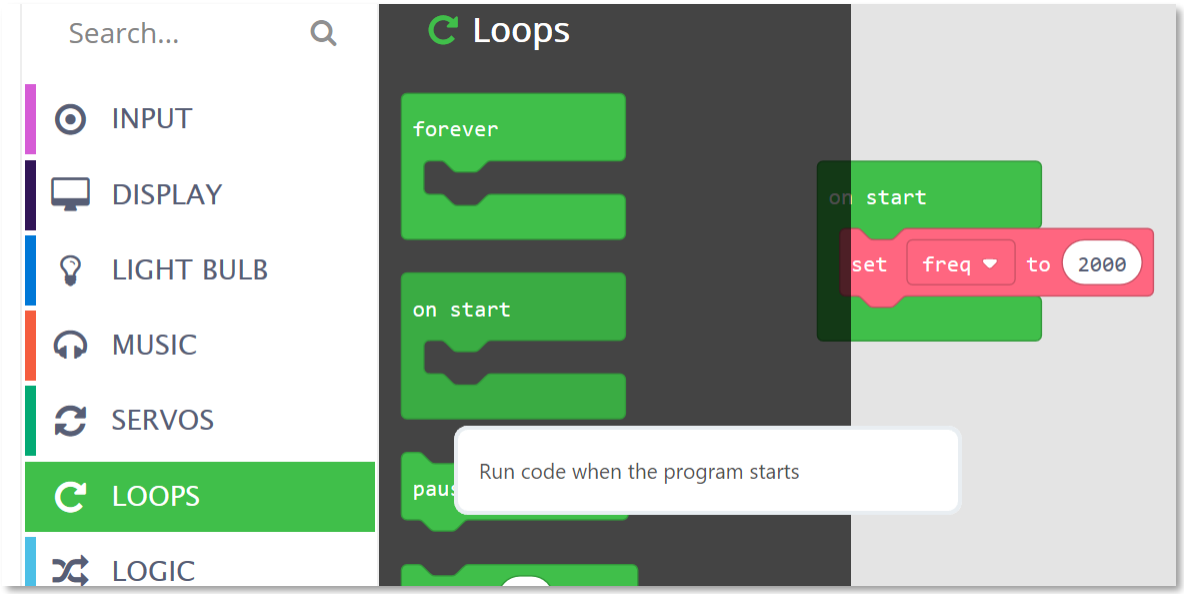
Im Menü VARIABLES kann man beliebig viele Variablen für sein Projekt anlegen. Dazu klickt man auf die Schaltfläche „Make a Variable“



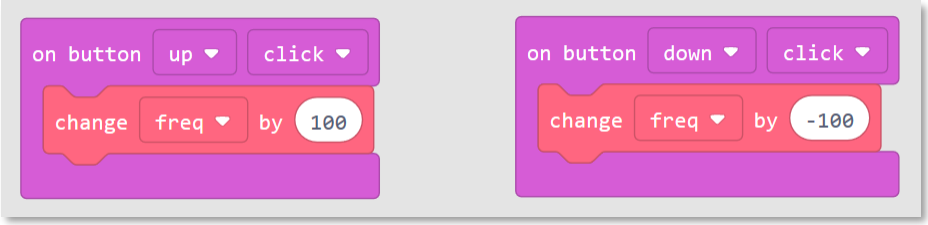
Die Variable braucht einen Namen. Wir möchten die Frequenz die auf dem Buzzer (kleiner Lautsprecher) abgespielt wird speichern also nennen wir sie „freq“.



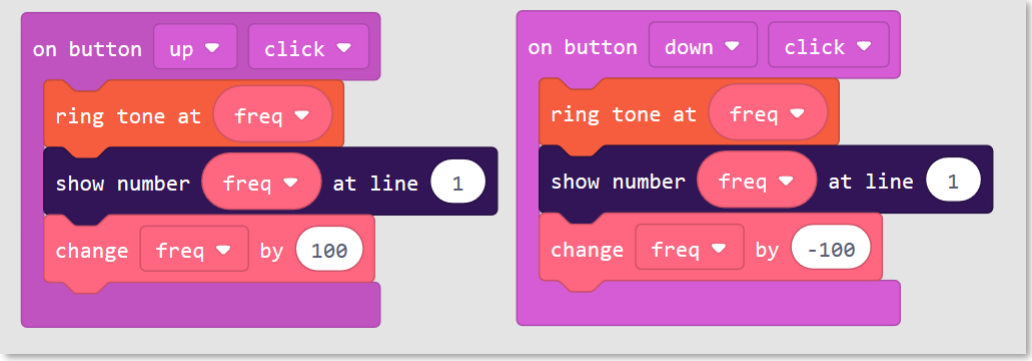
Gehe nun zu LOOPS und füge einen "on start" -Block hinzu. Was auch immer du in diesem Block hinzufügst wird ausgeführt wenn das Programm startet. Hier wollen wir die Variable auf den Wert 2000 setzen. Wir alle können eine Frequenz von 2000 Hertz hören, also ist es ein guter Ausgangspunkt. Der Block zum setzen des Wertes findet man unter VARIABLES "set freq to …" („Setze die Variable freq auf dem wert …“).



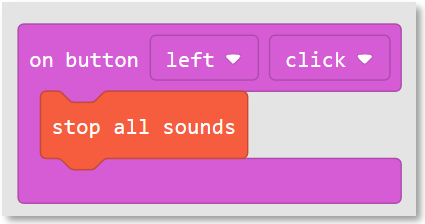
Nun müssen wir mit dieser Frequenz einen Ton erzeugen, aber wir möchten auch die Frequenz erhöhen, wenn die Aufwärts-Taste gedrückt wird, und die Frequenz verringern, wenn die Abwärts-Taste gedrückt wird. Du benötigst dafür den Block "change freq by …" („Ändere die Variable freq um …“) aus dem Menü VARIABLEN. Wir werden den Wert um 100 erhöhen, wenn „Up“ und um 100 Verringern, also -100, wenn „Down“ gedrückt wird.



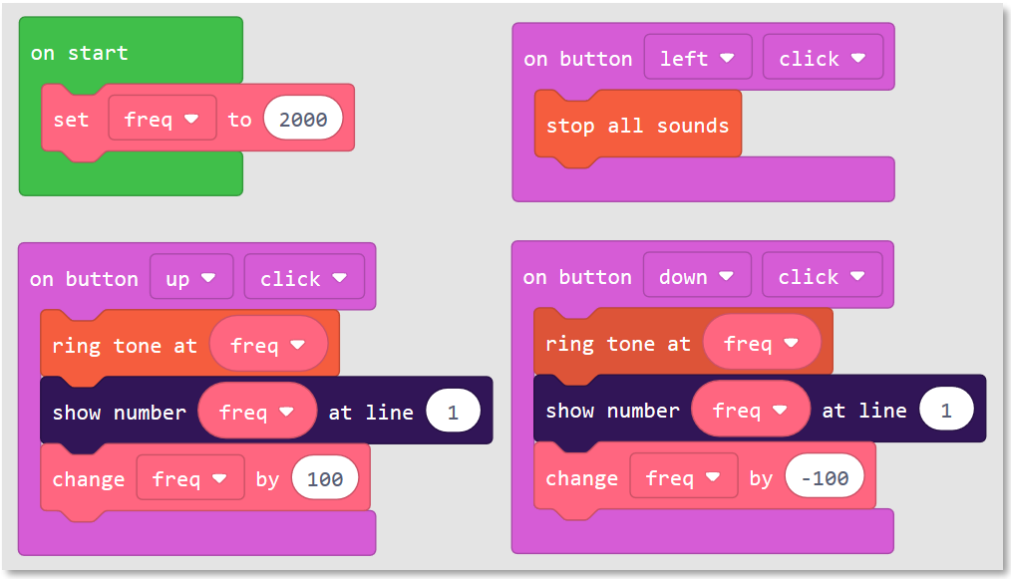
Jetzt, da wir die Variablen ändern können wollen wir mit „ring tone at“ einen Ton abspielen und den Wert mit „show number“ auf dem display anzeigen.



Als letztes werden wir noch einbauen, dass der Ton abgeschaltet wird, wenn man die Taste Left („Links“) drückt. Wir wollen ja nicht dass Nachbarn und Hunde belästigt werden.



Hier ist das komplette Programm.



SERVO MOTOREN

Ein Servomotor ist im Grunde genommen ein Motor, der über eine kleine interne Schaltung verfügt, mit der du den Servo mit einfachen elektrischen Impulsen steuern kannst.

Servos haben einen dreipoligen Stecker, der direkt in das BrainPad eingesteckt wird. Leider verwenden verschiedene Servos oft verschiedenfarbige Drähte. Um den korrekten Anschluss zu erkennen, ignoriere immer den mittleren Draht und schauen dir die Drähte auf jeder Seite an. Der Draht mit der helleren Farbe ist derjenige, der neben dem ~ -Symbol auf dem BrainPad sein muss. Diese nennt man Signal-Leitung.

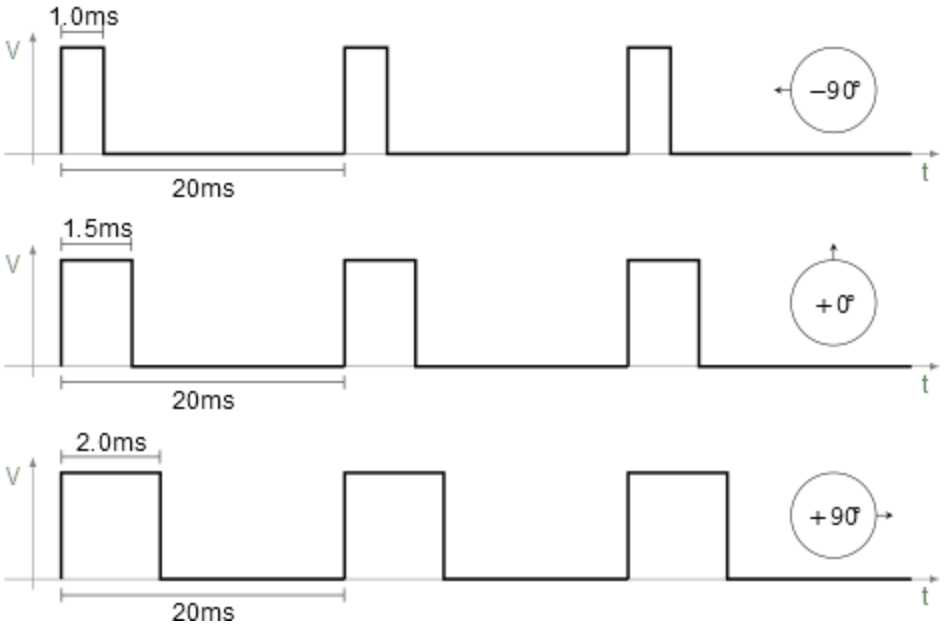
Servomotoren (oder Servos) werden häufig in ferngesteuerten Autos und Robotern verwendet und sind online und in lokalen Hobbyläden zu kaufen. Diese Servos werden vom BrainPad gespeist, das wiederum Strom von deinem PC oder einer Batterie bezieht. Daher empfehlen wir die Verwendung der kleineren Servos, die Mikroservos genannt werden. Einige der größeren Servomotoren verbrauchen zu viel Leistung und funktionieren ohne eine zusätzliche Stromversorgung gar nicht.

Servomotoren sind als entweder kontinuierliche- oder Positions-Servo erhältlich. Während sie identisch aussehen, dreht sich ein Positionsservo nur zu einer bestimmten Position und hält diese Position dann so lange bis du ihm eine andere Position nennst zu der er sich bewegen soll. Ein kontinuierlicher Servomotor dreht sich kontinuierlich in einer Richtung, bis er angewiesen wird, entweder zu stoppen ,die Richtung umzukehren oder die Geschwindigkeit zu verändern.

Wenn du einen nicht angeschlossenen Positionsservo von Hand drehen, bewegt es sich nur etwa eine halbe Umdrehung in beide Richtungen, bevor er anhält. Einen kontinuierliches Servo kannst du beliebig oft in die gleiche Richtung drehen. Wegen des internen Getriebes des Servos musst du eventuell etwas Kraft anwenden, um einen Servo überhaupt zu drehen.

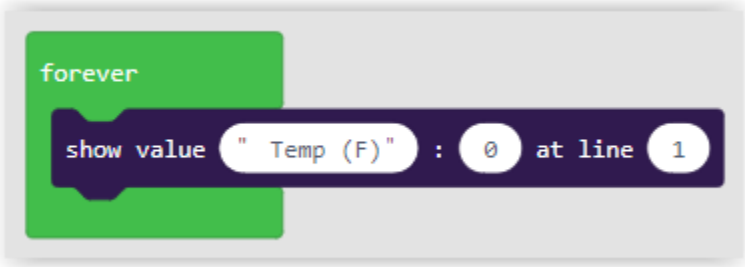
POSITIONS SERVO

Wie oben erwähnt bewegt sich ein Positionsservomotor in die Position (oder Winkel) die man ihm sagen, und bleibt dort, bis man ihn in eine andere Position bewegt. Ein vom BrainPad gesendeter Impuls teilt dem Servo mit, in welchen Winkel oder in welche Position er sich bewegen soll

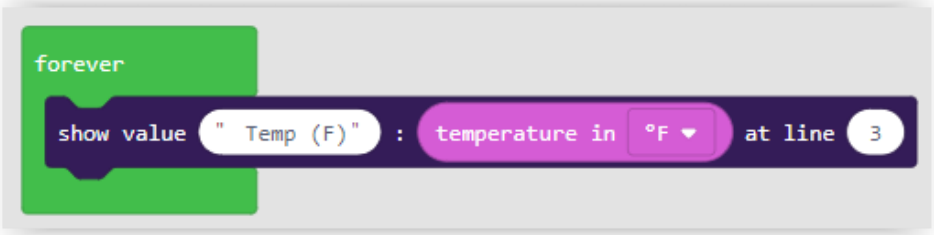


Es gibt viele Möglichkeiten, ein Positionsservo zu verwenden. In diesem Beispiel erstellen wir ein großes Thermometer mit einem Zeiger. Der von uns verwendete Code ist sehr einfach und verwendet nicht einmal eine Variable

Füge in einer forever-Schleife einen "show value" -Block (im Anzeigebereich gefunden) hinzu. Im ersten Oval des Show-Value-Blocktyps "Temp (F)" innerhalb der Klammern wie unten gezeigt



Wähle aus den INPUT Menü den Block " temperature in °C " und ziehe ihn in das zweite Oval. Unser Thermometer zeigt die Temperatur in Fahrenheit an, also ändere "° C" in "° F". Ändere nun das dritte Oval auf "3". Dadurch wird die Temperatur in der dritten Zeile des BrainPad-Displays ausgegeben. Deine Blöcke sollten wie der Screenshot unten aussehen

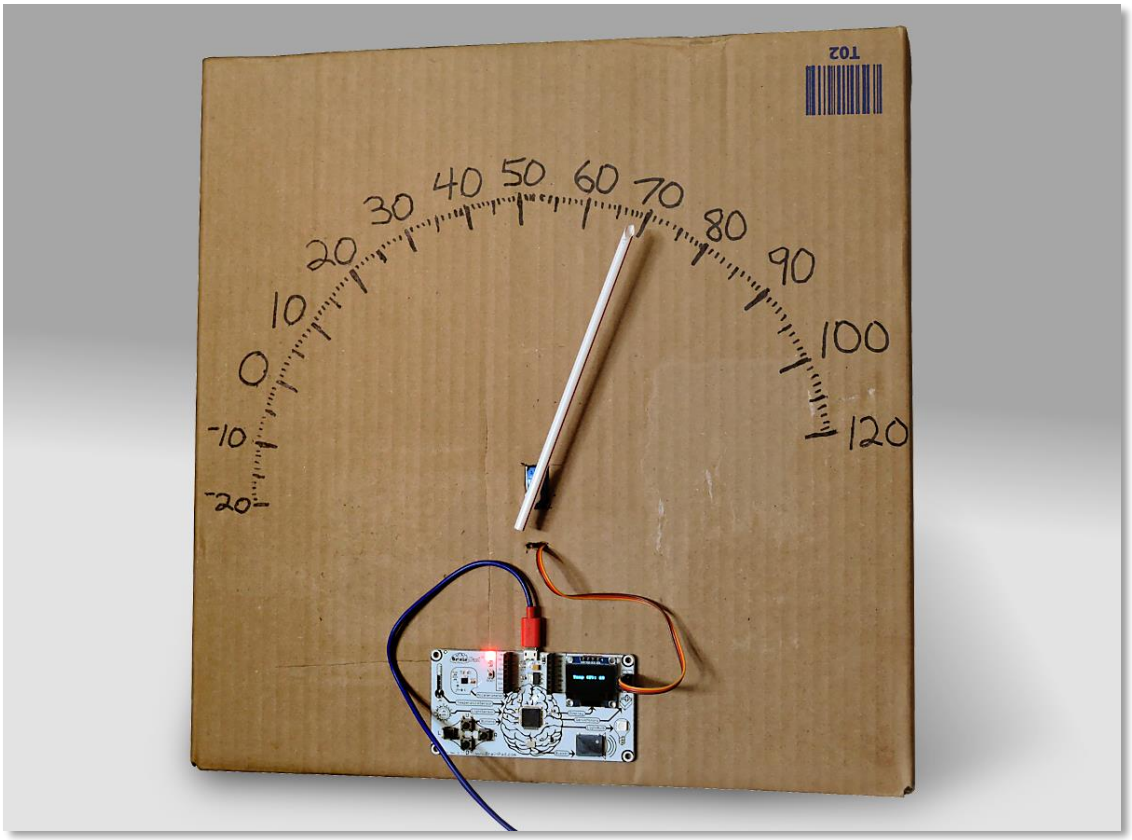


Bisher haben wir ein Thermometer hergestellt, das die Temperatur auf dem Display des BrainPads anzeigt. Jetzt müssen wir die Temperatur in einen Winkel umwandeln und es an einen Servomotor senden. Der Servo bewegt dann das Ziffernblatt unseres Thermometers. Unser Thermometer soll Werte von -20 ° Fahrenheit (-29°C) bis 120 ° Fahrenheit (49°C) anzeigen.

Daher müssen wir die Temperaturen -20 bis 120) in einen Winkel von 0 bis 180 umrechnen. Mathematikkenntnisse sind dafür nicht notwendig da MakeCode dafür eine eigene Funktion anbietet. Wir nutzen dafür den Block „map“ aus dem MATH Menü

Du wirst feststellen dass der Winkel in unserem map Block von einem Tief von 180 zu einem Hoch von 0 geht. Ist das nicht verkehrt? Nein! Wenn der Servomotor auf 0 Grad eingestellt ist, wird der Motor nach rechts gedreht, und wenn der Servo auf 180 Grad eingestellt ist, wird er nach links gedreht. Auf unser Skala ist -20 °F Links also bei einem Winkel von 180 Grad und 120 °F rechts also bei einem Winkel von 0 Grad

Hier ist fertig Thermometer aus einer Kiste und einem Strohhalm der auf einen Servomotor geklebt wurde.

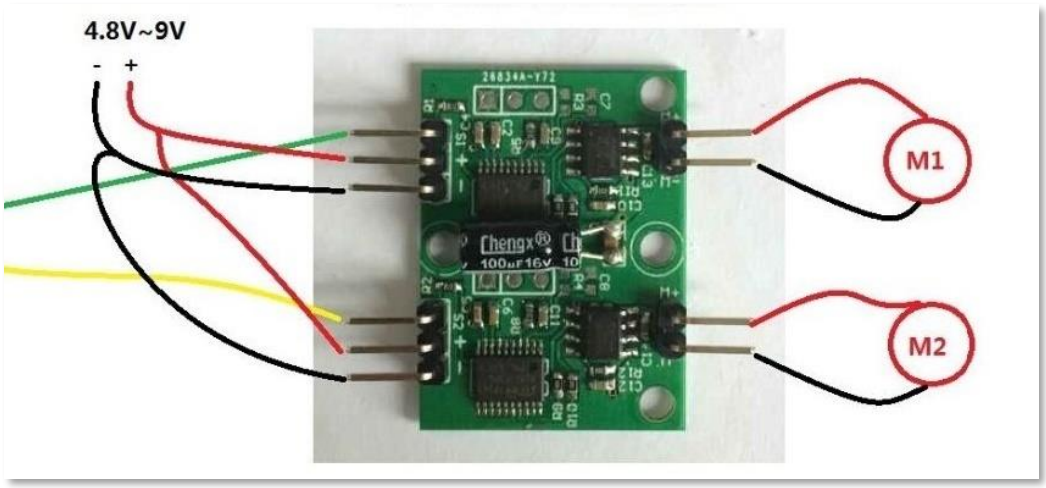


KONTINUIRLICHER SERVO MOTOR (ROTATIONSSERVO)

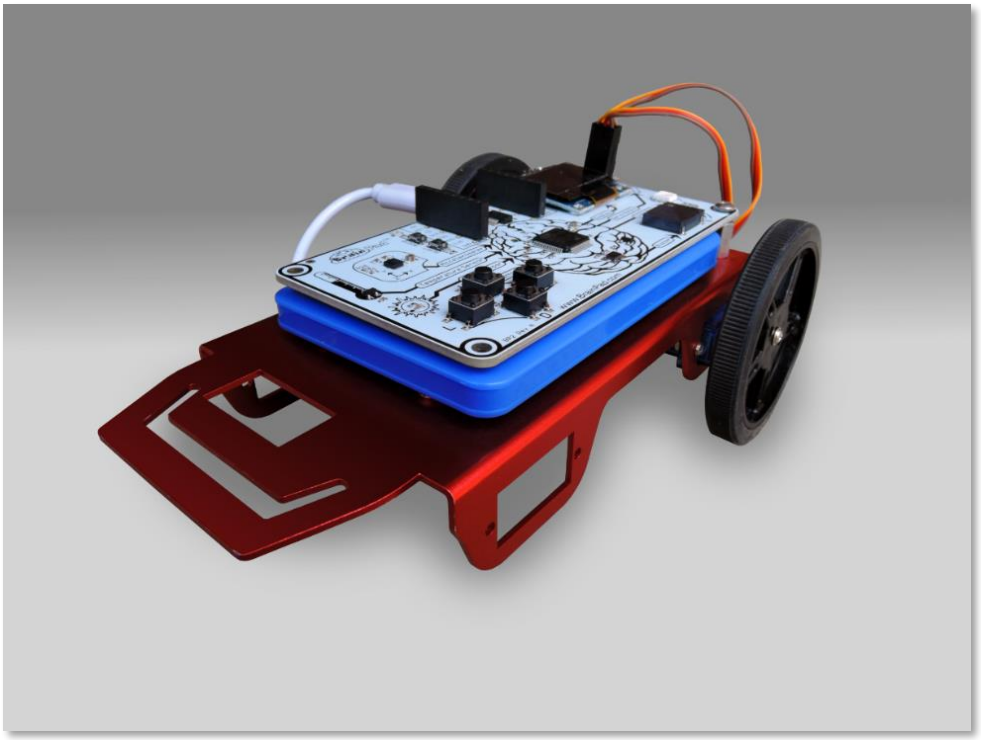
Die andere Art von Servomotor ist der Kontinuierliche Servo Motor oder auch Rotationsservo genannt,wie der Name schon sagt kann dieser Motor rotieren. Der Impuls gibt nicht Position an sondern die Geschwindigkeit und die Richtung in die sich der Motor drehen soll. Das BrainPad verfügt über Anschlüsse für zwei Motoren. Wenn du die Richtung und die Geschwindigkeit von zwei Rotationsservos steuern kannst kannst du auch einen Roboter leicht bewegen. Rotationsservos findet man übrigens auch bei diversen Onlinehändlern und Elektronikgeschäften.

Wir könnten den Roboter jetzt aus einer Pappschachtel basteln aber dann fand ich dieses Chassis von FEETECH.

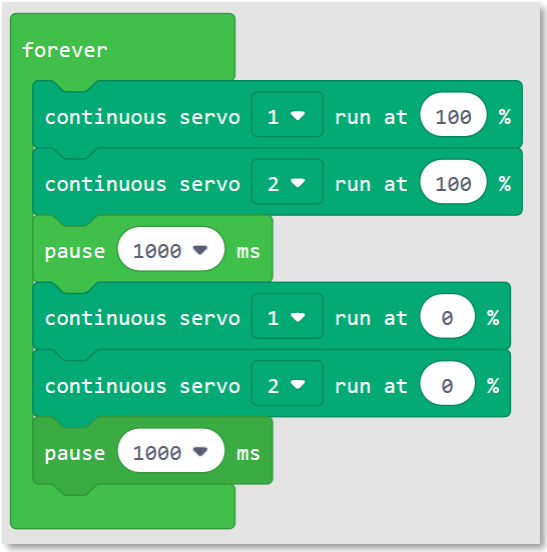
Hier wurde es etwas knifflig! Der Roboter kommt mit Rädern und Motoren. Die Motoren sehen aus wie Servomotoren aber sie erwiesen sich nicht als solche sondern als ganz normale Motoren. Das erkennt man sehr leicht da sie nur 2 Anschlußkabel haben. Der Bausatz kam auch mit einer kleinen Schaltung die die normalen Motoren in Servomotoren umwandelt.



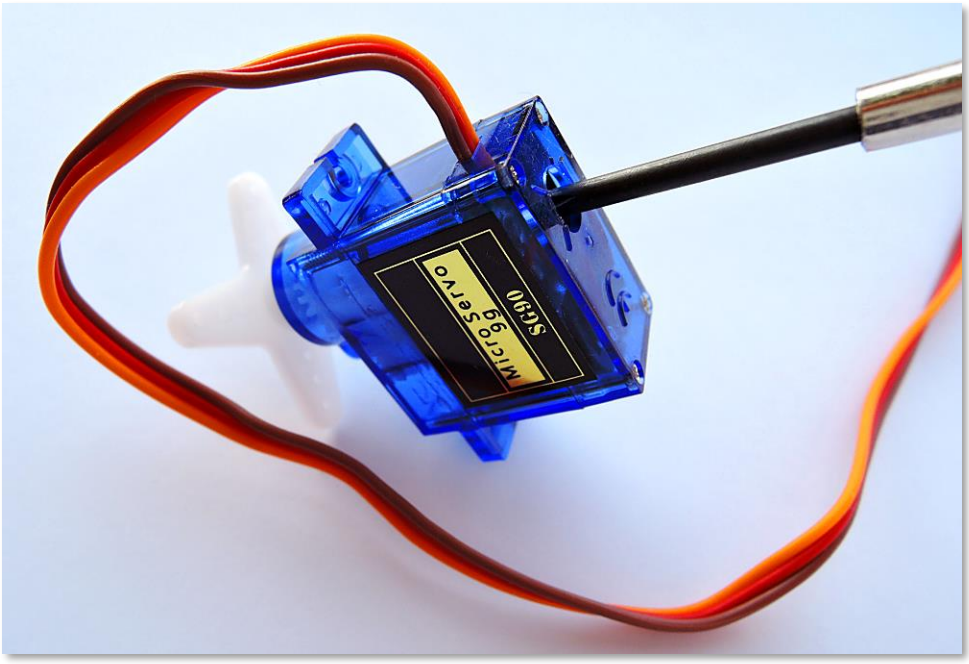
Dies ist zwar machbar aber für Anfänger zu kompliziert, deshalb habe ich die Schaltung und die darin enthaltenen Motoren beiseitegelegt und beschlossen, mit geeigneten kontinuierlichen Servomotoren zu arbeiten. Mein BrainPad passt perfekt auf dieses Chassis, als wäre es dafür gemacht! Ich fand dann auch eine Powerpack fürs Handy das sehr gut unter das BrainPad passte und dieses mit Strom versorgt.



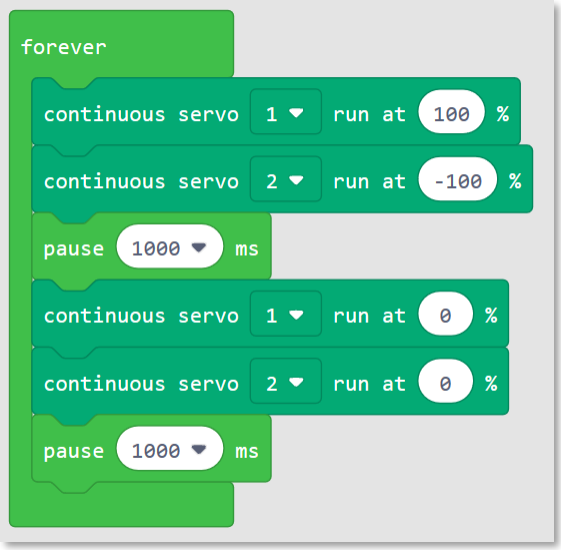
Dieser Roboter ist endlich bereit zu tanzen! Um vorwärts zu gehen, müssen beide Motoren vorwärts drehen. Da jedoch einer der Motoren nach rechts und der andere nach links zeigt muss, wenn man gerade aus fahren will, einer sich anders als der anderen drehen. Das mag verwirrend klingen, aber wenn man es einmal ausprobiert hat sieht man gleich wie das funktioniert. Dieser Code bewegt beide Servos eine Sekunde lang mit voller Geschwindigkeit vorwärts, stoppt die Motoren für eine Sekunde und wiederholt dann.



Wenn die Motoren weiterdrehen, wenn die Drehzahl auf 0% eingestellt ist, müssen sie kalibriert werden. Die meisten Servomotoren haben diese Möglichkeit du benötigst dafür einen kleinen Schraubendreher. Im Zweifelsfall schau in das Handbuch oder Datenblatt des Servos nach.



Zurück zum Bewegen des Roboters nach vorne. Siehst du jetzt was ich gemeint habe ? Ein Motor dreht sich in die falsche Richtung? Ändere bei einen der Servos die Richtung also sage ihn er soll mit voller Geschwindigkeit rückwärts zu fahren.



Jetzt fährt der Roboter gerade aus oder rückwärts aber er dreht sich nicht mehr im kreis.

Diese Arbeit war der Beginn unseres Roboterprojekts "Sun Seeker". Du kannst es hier https://docs.brainpad.com/projects/sun-seeker.html finden. Am Besten du siehst dir das Video an . Der Roboter erkennt mit dem Lichtsensor, ob er sich in der Sonne oder im Schatten befindet. Wenn es im Schatten ist wird sich der Roboter drehen bis er einen sonnigen Platz findet, in dem er sich bewegen kann.

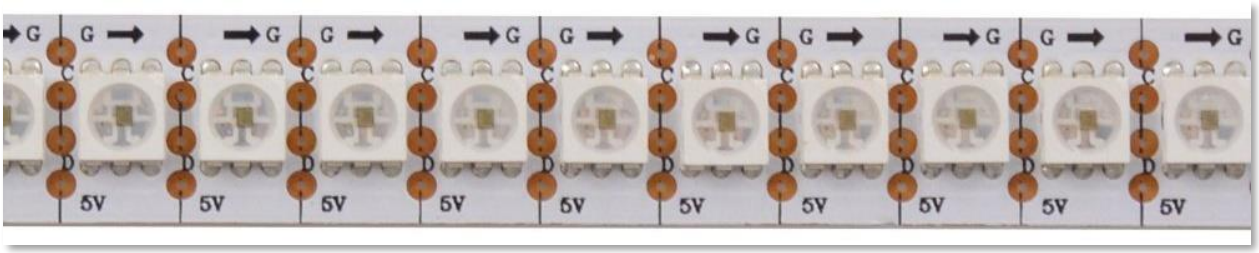
WEIHNACHTSBELEUCHTUNG

Es gibt viele Dinge die du mit den im BrainPad eingebauten Geräten tun können, aber irgendwann möchtest du vielleicht das BrainPad mit externen Komponenten verbinden. Obwohl dies den Rahmen dieses Buches sprengen würde, möchten wir Ihnen hier eine kurze Einführung geben, um dir einen Eindruck davon zu geben, was alles möglich ist.

Für dieses Projekt werden wir adressierbare LED-Streifen (Light Emitting Diodes) verwenden. Diese LED-Streifen bestehen aus einer Lichterkette, die individuell gesteuert werden kann. Sie können mit nur zwei Drähten auf

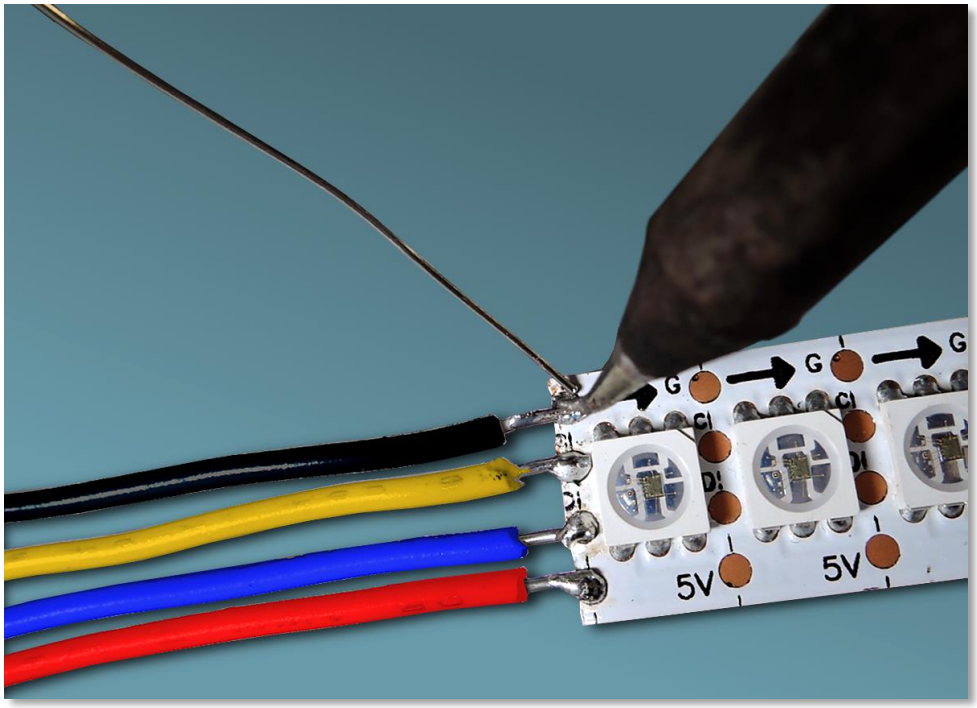
nahezu jede Farb- und Helligkeitskombination eingestellt werden.

Für dieses Projekt benötigst du LED Streifen mit dem APA102 Controller. Dieser Controller verwendet einen SPI-Bus, der auf dem BrainPad verfügbar ist. Diese Streifen werden durch unterschiedliche Spannungen versorgt. Wir benötigen eine Variante mit 5V da wir diese direkt vom BrainPad abgreifen können.

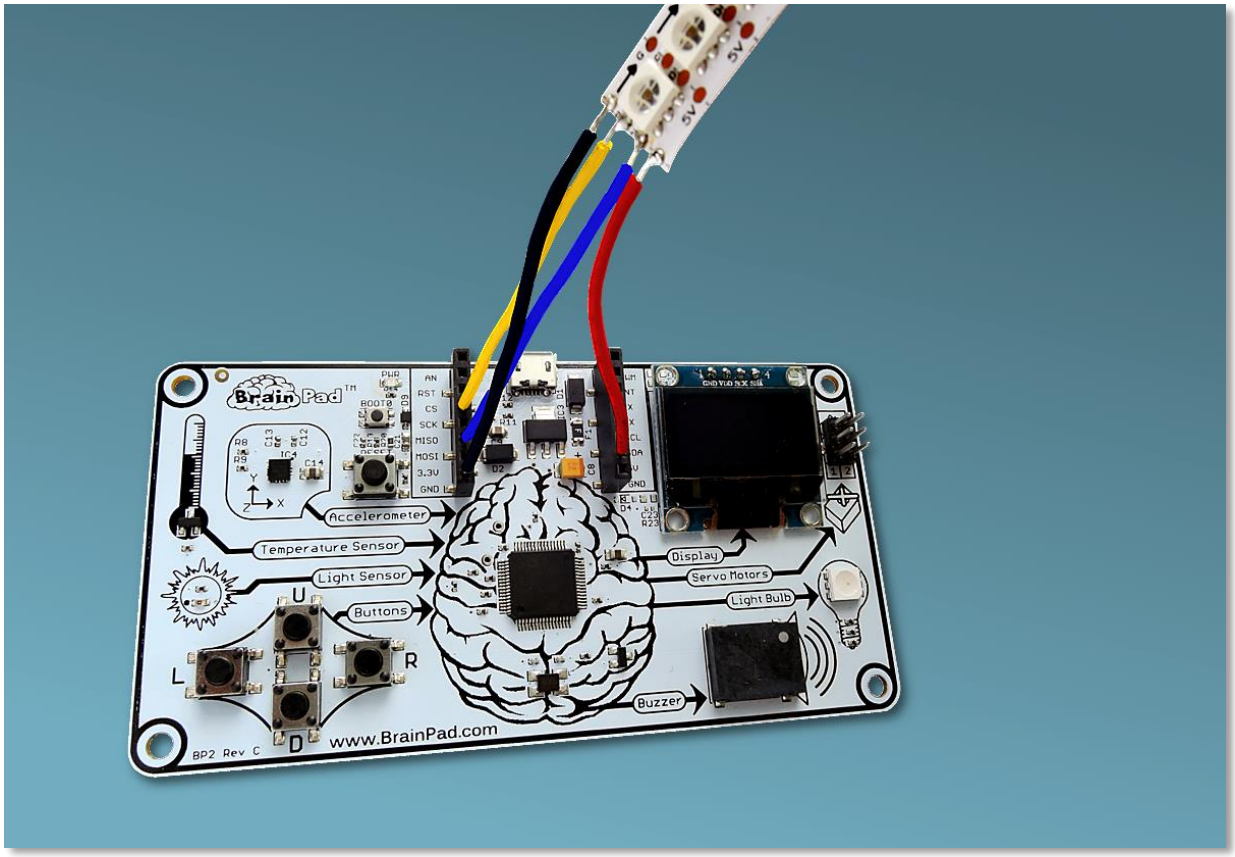


Der Streifen kann auf jede benötigte Größe zugeschnitten werden. Um zu verhindern, dass das Setup zu viel Strom verbraucht, werde ich es auf 25 LEDs reduzieren. Für längere Streifen empfiehlt sich dann eine externe Stromversorgung. Sollten deine Streifen keine Anschlusskabel haben müssen wir welche anlöten. Dazu benötigt man eine Lötausrüstung und ein bisschen Übung. Diese Arbeit ist nicht ganz ungefährlich da man sich leicht verbrennen kann also sollte unbedingt ein Erwachsener dabei sein!

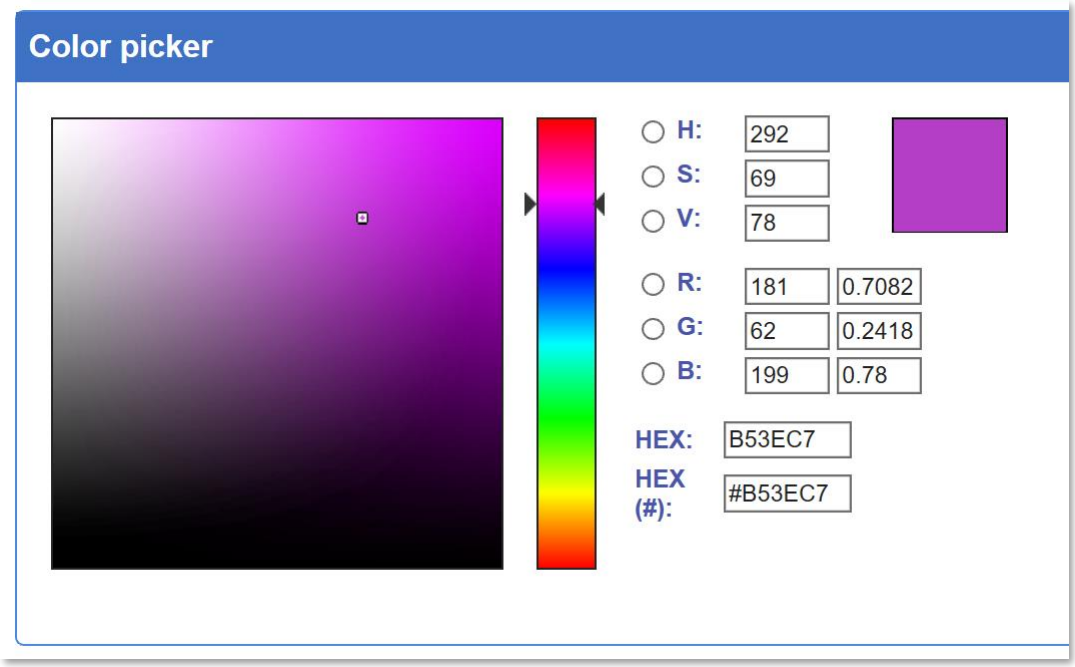
Die Drähte müssen an der Seite angelötet werden von denen der Pfeil weg zeigt also in Bild auf der linken Seite. Die 4 Drähte sind G-Masse, C-Clock, D-Data und 5V.



Masse (GND) und 5 V sind deutlich auf dem BrainPad-Erweiterungs-Header markiert. C-Clock muss mit SCK auf dem BrainPad verbunden werden. D-Data mit dem Anschluss der mit MOSI Bezeichnet ist



Jede LED hat drei interne LEDs. Sie sind rot, blau und grün. Mit diesen Grundfarben kann man jede beliebige Farbe erstellen. Es gibt mehrere Online-Tools die dir bei der Farbauswahl helfen. Ein gutes Beispiel findest du unter http://www.rgbtool.com .Uns interessieren die Werte die mit R, G, bzw. B beschriftet sind.



Die Befehle werden über den SPI-Bus des BrainPads an die LEDs gesendet. Um mit dem Schreiben einer Folge von Farben auf den LED-Streifen zu beginnen, muss zuerst ein Startsignal gesendet werden. Das Startsignal ist einfach vier Nullen.

Nach dem Startsignal benötigt jede LED vier Zahlen, um sie zu steuern. Die erste Zahl legt die globale Helligkeit für die gesamte LED-Reihe fest. Wir wollen uns nicht im Detail verzetteln, also werden wir immer die höchst mögliche Zahl 255 senden um die volle Helligkeit hat zu bekommen.

Nach der ersten Nummer für die Helligkeit kommen jeweils drei Zahlen zwischen 0 bis 255 die die die Helligkeit von Blau, Grün und Rot in jeder LED steuern. Um eine LED hell blau zu machen senden wir 255, 0, 0 senden. Um eine LED auszuschalten sendet man 0,0,0 senden. Und um eine LED hellweiß zu machen 255, 255, 255. Eigentlich ganz einfach.

Die LEDs im Streifen sind einfach miteinander verkettet. Die erste Farbe wird an die erste LED, die zweite an die zweite LED und so weiter, gesendet. Die LEDs leuchten nacheinander auf, bis ein neues Startsignal gesendet wird (vier Nullen). Nach dem Startsignal beginnt die Sequenz mit der ersten LED.

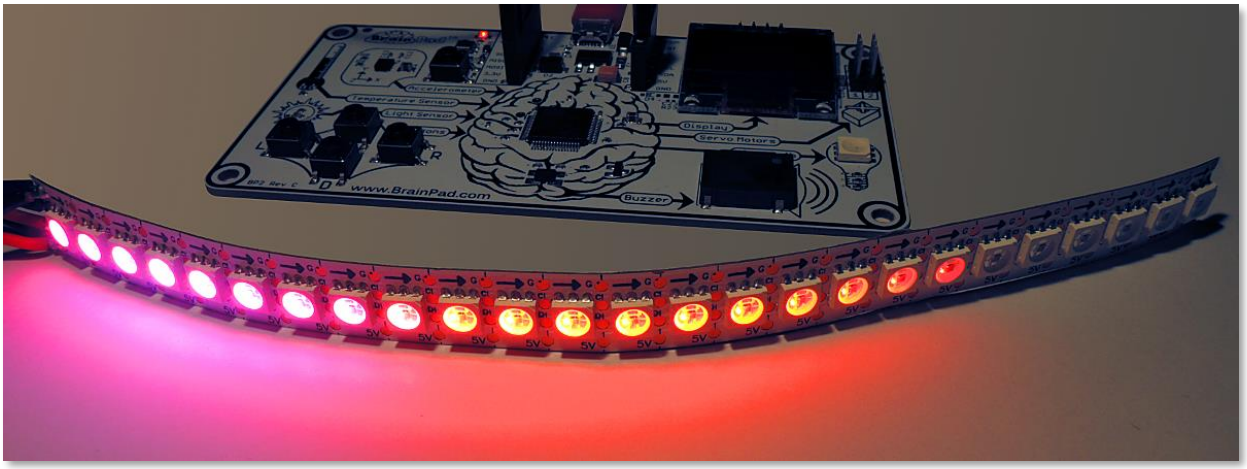
Aufgrund der benötigten Anzahl von Blöcken ist es einfacher in den JavaScript Modus zu wechseln und den unten aufgelisteten Code kopieren und einzufügen. Also kopiere den folgenden Code und ihn in den Microsoft MakeCode-JavaScript-Editor ein:

```
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)
    }
})
```

Wenn du alles richtiggemacht hast und das Programm auf das Brainpad kopiert hast solltest du eine helle zufällige Lichtshow sehen. Versuche einen Teil des obigen Codes zu ändern um zu sehen wie sich das auf die Beleuchtung auswirkt.



FAZIT

Microsoft MakeCode und das BrainPad sind eine großartige Kombination um das Programmieren zu lernen. Der Simulator hilft dir dass du den Code ohne ein BrainPad einfach testen kannst. Dann kannst du das Programm auf das BrainPad laden, indem du einfach die heruntergeladene Programmdatei in das BrainPad-Fenster kopierst.

In den nächsten Kapiteln werden wir weitergehen und mit C # oder Visual Basic in Microsoft Visual Studio programmieren. In vielen Fällen ist die Verwendung von MakeCode alles was du brauchst wenn du aber mehr über das Programmieren lernen willst solltest du dir die nächsten Kapitel auch ansehen.

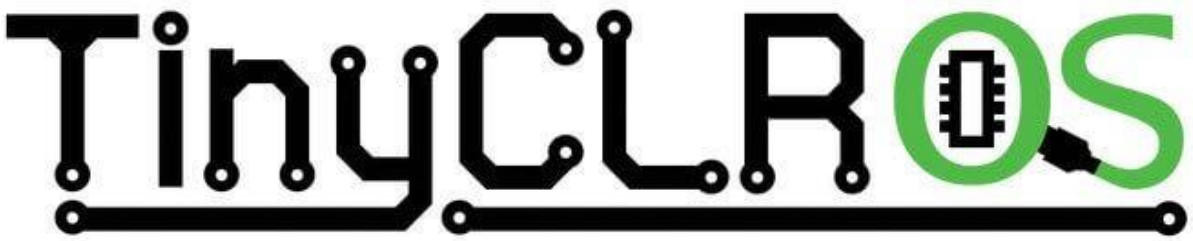
GO BEYOND – GEHE WEITER

Wenn du hier angekommen bist, willst du wie ein richtiger Profi programmieren. Du wirst hier die professionelle Programmierumgebung *Microsoft Visual Studio* kennenlernen, um in *C#* (C Sharp) oder *Visual Basic* zu programmieren. Du wirst lernen, das komplette Arsenal zur Fehlersuche (Debugging) und die reichhaltigen Funktionen dieser Programmierumgebungen zu verwenden.

Microsoft **MakeCode** zu kennen, ist für das Folgende keine Voraussetzung, wird aber sehr empfohlen. Nicht umsonst haben wir das **MakeCode** Kapitel **Start Making** (Fange an) genannt.
Wie auch immer, schnallt euch an, und los geht‘s mit einem coolen Ausflug in die Welt der professionellen Programmierung, vielleicht der Beginn deiner zukünftigen Karriere als Programmierer/in.

TINYCLR OS™

Um mit dem BrainPad in der .NET Umgebung (.NET ist eine gemeinsame Basis zur Verwendung von u.a. C# und Visual Basic) mit C# oder Visual Basic zu arbeiten, braucht ihr auf dem BrainPad TinyCLR OS (OS = Operating System = Betriebssystem). Dieses kleine Betriebssystem macht all die ‚magischen‘ Sachen im Hintergrund. Es führt kompilierte .NET Assemblies (.NET Programmcode-Zusammenstellungen) aus und stellt tausende verschiedener Funktionen/Dienste unter anderem für Threading, Speicherverwaltung und Debugging zur Verfügung. Das sind die gleichen .Net Funktionen, die auch benutzt werden, um Software (Programme/Apps) für Handys oder Computer zu schreiben. TinyCLR wurde allerdings speziell für kleine Computer wie das BrainPad gemacht. Es stellt deshalb nicht alle Funktionen des .NET Standard zur Verfügung.



VISUAL STUDIO

TinyCLR wird aktuell nur von der Programmierumgebung Microsoft Visual Studio unterstützt. Theoretisch könnten jedoch auch andere .NET Compiler einer Unterstützung bereitstellen. Die ‚Community Edition‘ von Visual Studio ist eine voll ausgestattete Programmierumgebung, die als eine der besten (vielleicht als die beste überhaupt) angesehen wird -- und das Beste von allem, sie ist **kostenlos**. Um Visual Studio zu installieren, braucht ihr einen modernen PC mit Windows Betriebssystem. Windows 10 wird empfohlen.

Nicht zu vergessen -- ihr könnt das gleiche Microsoft Visual Studio und die Erfahrungen, die ihr bei der Programmierung des BrainPad erwerbt, nutzen, um für andere Geräte wie Handys, PCs und sogar die Microsoft Xbox Spielkonsole Apps zu schreiben.

SYSTEM SETUP – SYSTEM VORBEREITUNG

Die Visual Studio Community Edition ist kostenlos, aber sie ist kein Spielzeug und sie ist nicht klein. Du wirst etwas Zeit brauchen, um das System aufzusetzen. Genaue Anweisungen kannst du auf der BrainPad Dokumentations-Website finden.

https://docs.brainpad.com/go-beyond/system-setup.html.

Zusammenfassend musst du die folgenden Programme herunterladen und installieren:

1. Microsoft Visual Studio Community Edition. Wichtig ist, dass du bei der Installation die ".NET desktop development" option auswählst.
2. GHI Electronics TinyCLR OS Project system

falls du Schwierigkeiten hast, kannst du jederzeit Hilfe in unserem Forum finden
https://forums.ghielectronics.com/c/brainpad

Nachdem der PC vorbereitet ist, musst du das TinyCLR OS Betriebssystem auf das BrainPad laden. Lade die TinyCLR OS Firmware Datei von hier https://docs.brainpad.com/resources/downloads.html herunter und lade sie dann auf das BrainPad, genau so, wie du jede Datei lädst, die du mit Microsoft MakeCode erzeugt hast.

1. Drücke den Reset Button ungefähr drei Sekunden lang. Die Light Bulb LED leuchtet nun grün.
2. Der PC wird das BrainPad nun als Speichermedium (Storage Device) erkennen und ein Fenster für es öffnen.
3. Speichere die TinyCLR OS Firmware Datei auf dieses BrainPad Speichermedium. Hierfür kannst die die Datei durch Ziehen mit der Maus oder per Copy and Paste (Kopieren und Einfügen) in das BrainPad Fenster überführen.

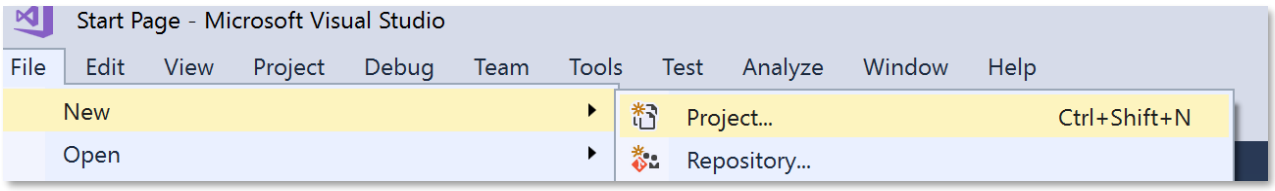
HELLO WORLD – HALLO WELT

Es ist üblich, wenn man ein noch unbekanntes Gerät programmieren will oder wenn man neue Programmierwerkzeuge verwenden will, mit einem sehr einfachen Programm anzufangen. Hierdurch stellt man

sicher, dass das Gerät funktioniert und alles richtig zusammenarbeitet. Solch ein einfaches Programm wird oft ‚Hello Word' Programm genannt.

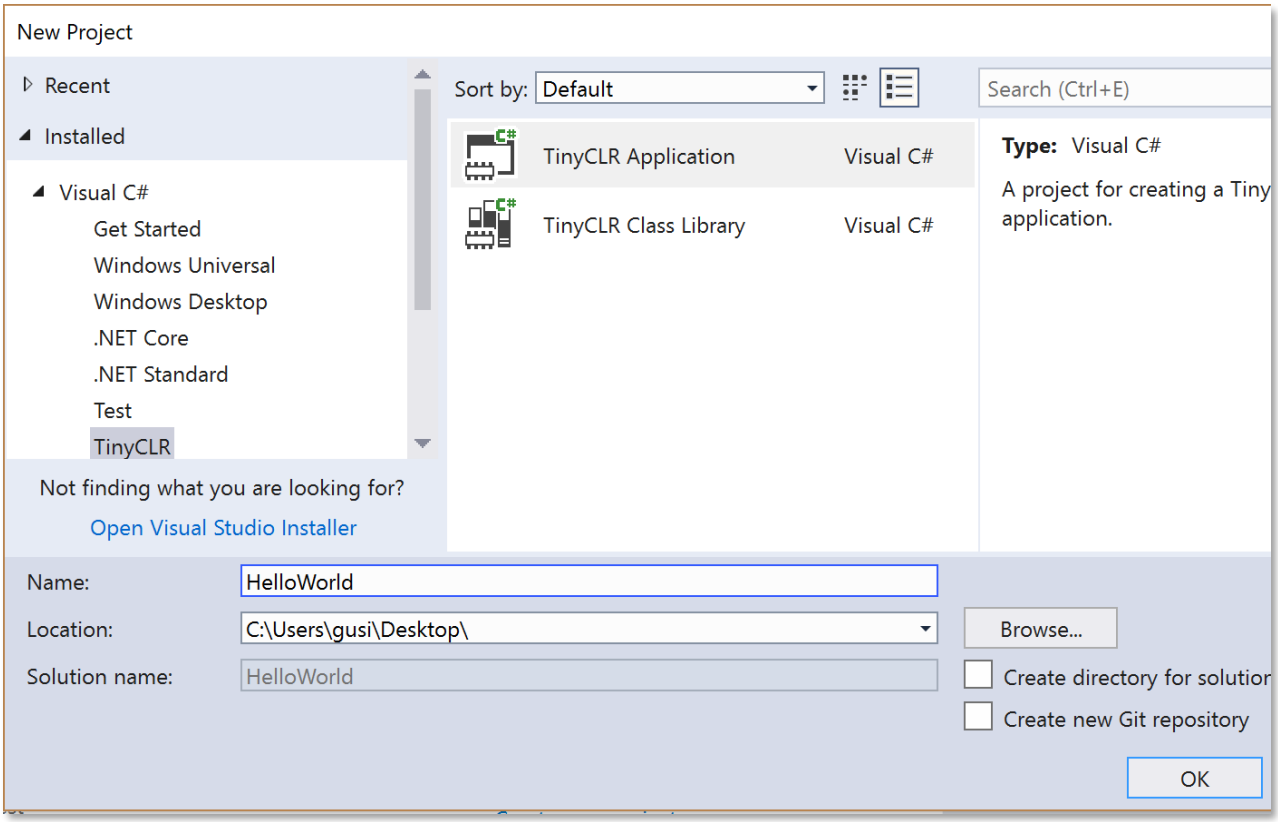
Lasst uns damit anfangen.

Öffne Microsoft Visual Studio und starte ein neues Projekt. Im „Datei" Menü wähle „Neu" und dann „Projekt" (Datei > Neu > Projekt).

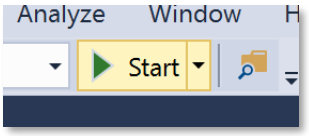


Unter Visual C# sollte als Option TinyCLR erscheinen. Wenn nicht, hast du wahrscheinlich das TinyCLR OS Projekt-System nicht installiert.

Im Beispiel auf dem Bild habe ich das Projekt "HelloWorld" genannt und es auf meinem Desktop plaziert.



Du kannst dieses leere Projekt jetzt auf dein BrainPad laden. Das Projekt wird nicht viel tun, aber das Laden auf das BrainPad wird zeigen ob alles wie erwartet funktioniert. Also los, verbinde das BrainPad mit dem USB Kabel und klicke auf Start.

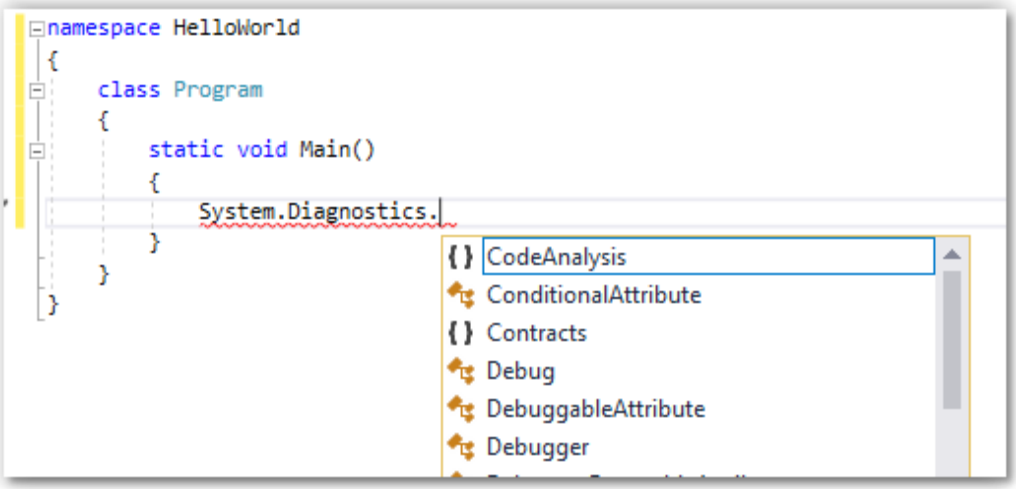


Beobachte die Meldungen im Ausgabe Fenster von Visual Studio. Wenn dieses Fenster nicht sichtbar ist, drücke die Tastenkombination Alt+Ctrl+O (Alt+Strg-O) auf der Tastatur oder klicke auf "Ausgabe" im "Ansicht" Menü (Ansicht > Ausgabe).

Jetzt sind wir soweit, dass uns das BrainPad mit "Hello World." begrüßen kann. Zuerst werden wir "Hello World." im Visual Studio Ausgabe Fenster anzeigen. Später werden wir den Text auf dem BrainPad Display anzeigen. Füge die folgende Textzeile im Programm genau unter der linken geschweiften Klammer ({) nach Main() hinzu.

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

Beim Schreiben wirst du sehen, wie Visual Studio dir automatisch Wortvorschläge gibt.



Der Programm Code sollte nun so aussehen:

```
using System;
using System.Collections;
using System.Text;
using System.Threading;

namespace HelloWorld
{
    class Program
    {
        static void Main()
        {
            System.Diagnostics.Debug.WriteLine("Hello World");
        }
    }
}
```

Starte das Programm und beobachte wieder das Ausgabe Fenster

```
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).
```

Hast du "Hello World." gefunden? Das war ein guter Start aber nicht sehr aufregend. Versuchen wir jetzt, mehrere Zeilen auszugeben. Ersetze die Zeile, die wird gerade hinzugefügt haben, durch die folgenden vier Zeilen.

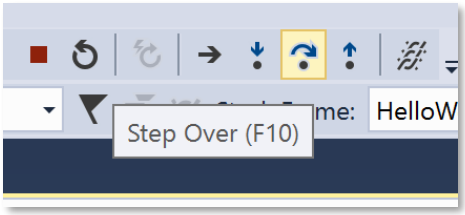
```
System.Diagnostics.Debug.WriteLine("The");
System.Diagnostics.Debug.WriteLine("BrainPad");
System.Diagnostics.Debug.WriteLine("is");
System.Diagnostics.Debug.WriteLine("amazing!");
```

Setze den Cursor auf die erste Zeile und drücke die F9 Funktionstaste. Das wird einen Breakpoint (Haltepunkt) im Programm erzeugen. Du kannst auch Breakpoints erzeugen, wenn du dich im Debuggen Menü befindest.

```
namespace HelloWorld
{
    class Program
    {
        static void Main()
        {
            System.Diagnostics.Debug.WriteLine("The");
            System.Diagnostics.Debug.WriteLine("BrainPad");
            System.Diagnostics.Debug.WriteLine("is");
            System.Diagnostics.Debug.WriteLine("amazing!");
        }
    }
}
```

Starte das Programm genau wie gehabt. Sobald die Programmausführung auf den Breakpoint trifft, wird das Programm in dieser Zeile angehalten.

Es wird angehalten, gerade bevor der Code in dieser Zeile ausgeführt wird. Drücke nun die F10 Funktionstaste oder klicke auf den Step over (Prozedurschritt) Button.



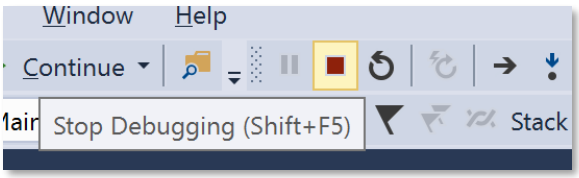
Das Programm wird hierauf eine Codezeile oder einen Programmschritt ausführen. Weil die Zeile, in der wir uns befinden, “The” im Ausgabe Fenster ausgibt, wirst du sehen, dass dieser Text nun dort erscheint. Gehe weiter schrittweise durch den Code (F10) und beobachte das Ausgabe Fenster. Dieses Vorgehen wird "Stepping through Code" (Schrittweise Programmausführung) genannt und ist sehr hilfreich beim "Debuggen" (Fehlersuche) deiner Programme, um also herauszufinden, warum etwas nicht wie erwartet läuft.

ENDLOS LAUFENDE PROROGRAMME

Auf einem PC oder einem Smartphone ist es oft sinnvoll, dass ausgeführte Programme enden. Auf manchen kleineren Geräten jedoch, "Embedded Devices" (eingebettete Geräte) genannt, enden die ausgeführten Programme üblicherweise nicht. Denke zum Beispiel an eine Mikrowelle in der Küche. Die Mikrowelle hat ein kleines "Gehirn", das dauernd die Eingabetasten überwacht und die Uhr auf dem Bildschirm aktualisiert. Solange du den Stecker nicht ziehst, läuft dieses Programm ununterbrochen. Dieses Verhalten wird "unendliche Schleife" genannt, eine Funktion, die beim Computer Programmieren häufig verwendet wird.

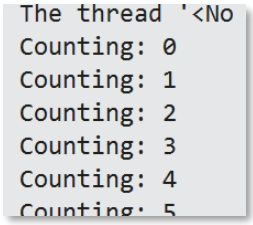
Lasst uns jetzt einen Zähler mit dem BrainPad machen, der "unendlich" läuft. Das BrainPad wird jede Sekunde um eins weiter zählen und damit "unendlich" fortfahren. Unten seht ihr, wie der Code hierfür aussieht.

Denke daran, dass du jedes Programm beenden musst, bevor du den Code ändern kannst. Der "Stop" Button (Debuggen beenden) dient zum Beenden des Programms.

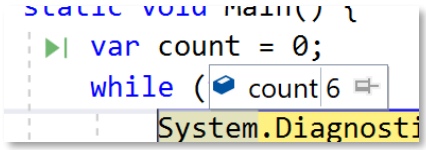



```
namespace HelloWorld {
    class Program {
        static void Main()
        {
            var count = 0;

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

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

Das Ausgabe Fenster zeigt den Zähler.



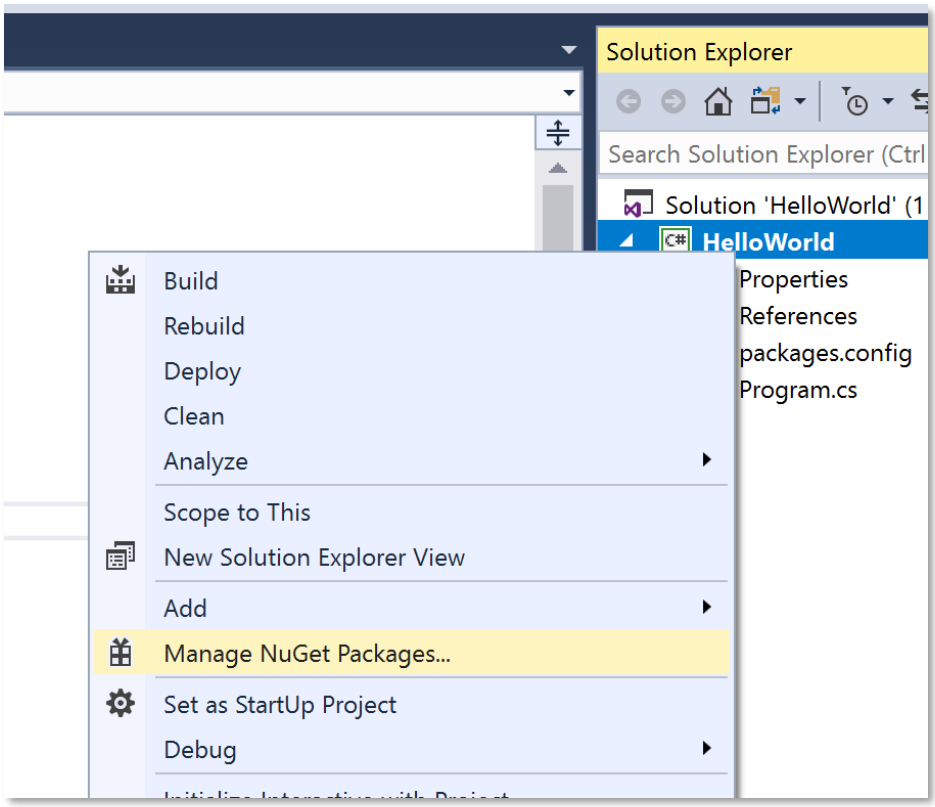
Erstelle einen Breakpoint in der Zähler-Programmzeile wärend das Programm läuft. Von hier aus kannst du, wie zuvor beschrieben, schrittweise den Code abarbeiten lassen. Bringe jetzt den Mauszeiger über die Variable “count” – Visual Studio zeigt dir jetzt den aktuellen Wert der Variablen an. Als ich das Programm anhielt, war die Variable gerade 6. Nebenbei: Sich auf diese Weise Variablen anzusehen, ist eine weitere sehr nützliche Funktion zum Debuggen von Programmen.



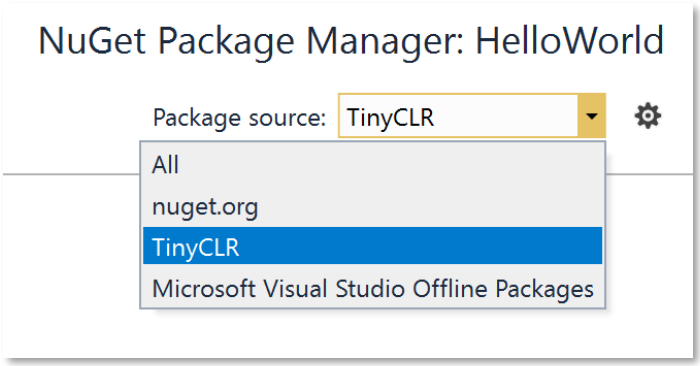
BRAINPAD LIBRARIES -- BRAINPAD BIBLIOTHEKEN

Bisher haben wir das reine TinyCLR OS zur Ausführung einfacher Programme verwendet. Das BrainPad wird jedoch zusammen mit sog. Libraries (Programmbibliotheken) ausgeliefert, um die Benutzung der vorhandenen Ein-und

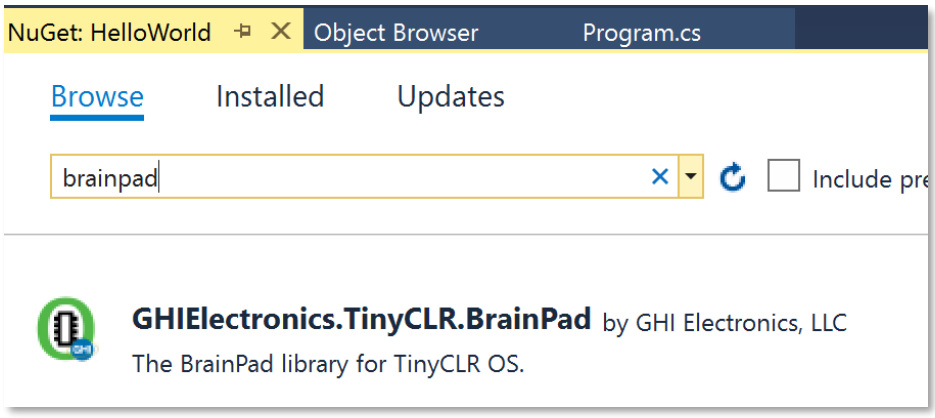
Ausgabemöglichkeiten einfacher zu gestalten. Mache einen Rechts-Klick auf das Projekt im Projektmappen – Explorer und wähle dann NuGet-Pakete verwalten… aus.



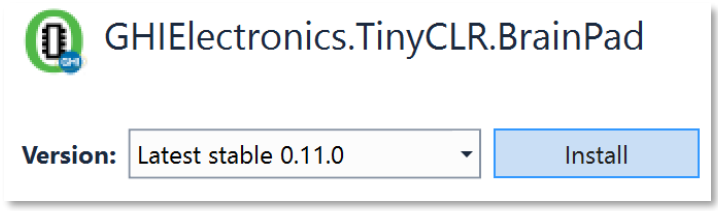
NuGet ist ein Onlinedienst zur Bereitstellung von Bibliotheken. Man kann die Libraries auch herunterladen und auf dem eigenen PC bereitstellen. Ich persönlich habe zum Beispiel die Libraries lokal auf meinem PC in einem Verzeichnis mit dem Namen TinyCLR bereitgestellt.



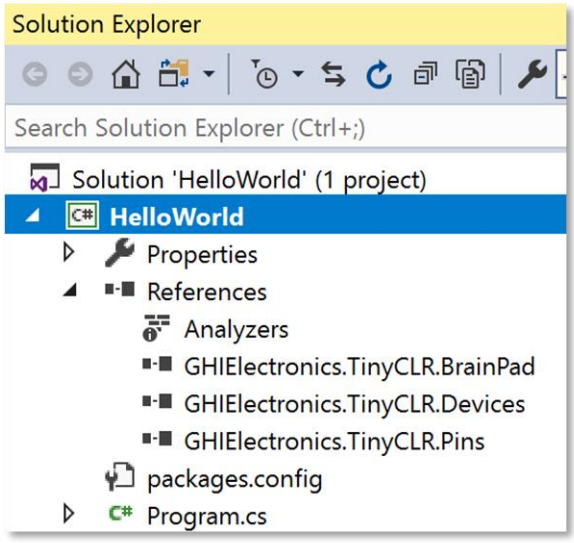
Nach Auswahl der Registerkarte "Durchsuchen", gebe "brainpad" in die Such-Maske ein.



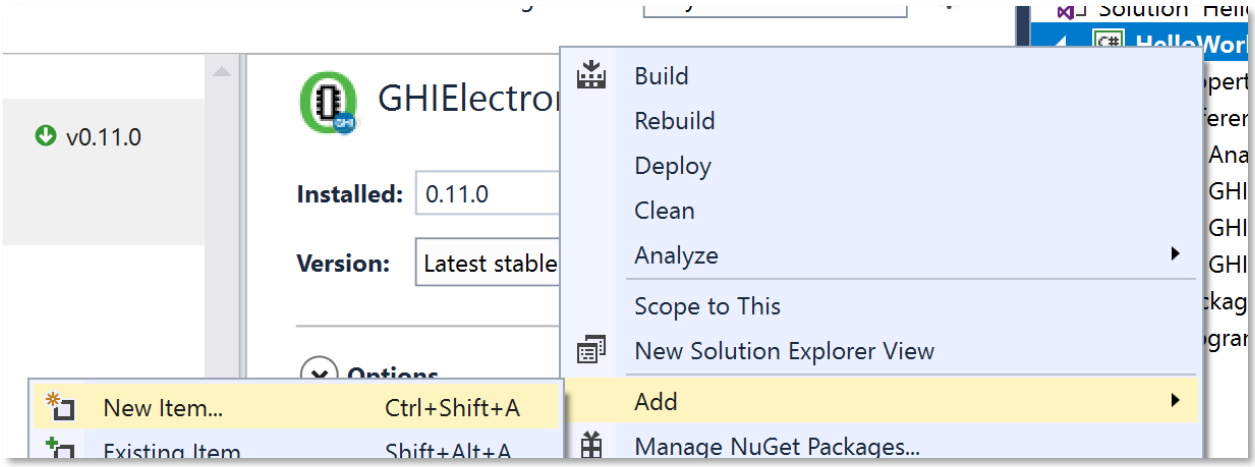
Die `GHIElectronics.TinyCLR.BrainPad` Library wird angezeigt. Wähle sie aus und klicke auf installieren.



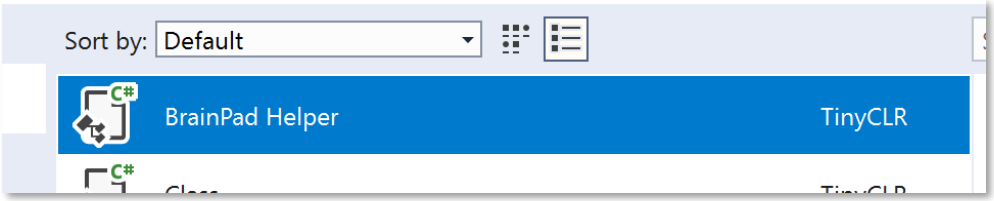
Hierdurch werden die benötigten Libraries den Verweisen (References) im Projektmappen-Explorer hinzugefügt.



Um allgemein das Hinzufügen von Libraries zum Code zu erleichtern, ist eine Hilfsfunktion vorhanden. Rechtsklicke auf das Projekt, anschließen wähle Hinzufügen > Neues Element … Du kannst auch die Tastenkombination Ctrl+Shift+A (Strg+Umschalt+A) verwenden.



Nach dem Klicken auf Neu (oder Ctrl+Shift-A) öffnet sich ein Auswahlmenü. Wähle durch Anklicken BrainPadHelper und klicke dann rechts unten auf Hinzufügen.



Du wirst sehen, jetzt fängt es an, richtig Spaß zu machen! Gehe zurück in unser Projekt und öffne Program.cs durch Doppelklick. Wir ändern jetzt den Code, so dass der Zähler auf dem Display des BrainPad angezeigt wird.

```
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;
            }
        }
    }
}
```

Die Codezeile, die den Zähler ausgibt, ist ähnlich wie der Debug.WriteLine Befehl, den wir bisher benutzt haben. Allerdings erfolgt die Ausgabe jetzt auf das BrainPad Display. Die zweite Codezeile (...RefreshScreen) ist nötig, um den neuen Inhalt auf das Display zu übertragen. Displays arbeiten viel langsamer als Mikroprozessoren . Wenn wir die Anzeige auf dem Display jedes Mal aktualisieren würden, wenn wir den Inhalt (z.B: Text) ändern, könnte die Ausgabe auf das Display nur sehr langsam erfolgen. Deshalb schreibt der Prozessor den auf dem Dispaly auszugebenden Inhalt zunächst in seinen Speicher. Erst wenn der Befehl RefreshScreen ausgeführt wird, werden die Änderungen aus dem Speicher auf das eigentliche Display übertragen (Refresh). Denke an ein Video Spiel, in dem du z.B. ein Schiff, Hindernisse, Feinde u.a. zeichnen musst. Du wirst hierbei alle Zeichenaufgaben zunächst nur im Speicher des Computers durchführen, zu dem der Prozessor einen schnellen Schreibzugriff hat. Nur wenn nötig wirst du das eigentliche Display aktualisieren (Refresh).

Die letzte Codezeile bewirkt, dass das BrainPad im Programmablauf eine Sekunde pausiert.

BOUNCY BALL – ABPRALLENDER BALL

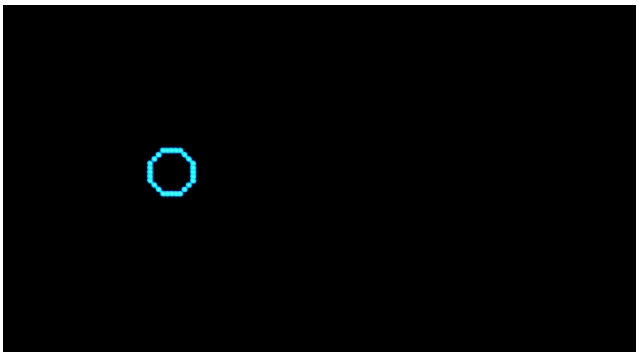
Durch die BrainPad Libraries sieht alles mehr "Sprachlich" aus, als wenn reiner Code verwendet würde. Denke dir ein Paar logische Funktionen zur Änderung des Codes aus und du wirst faszinierende Programme schreiben können. Das eingehende Vermitteln der Programmiersprachen C#/Visual Basic ist nicht Gegenstand dieses Buches, das bedeutet aber nicht, dass wir mit dem hier Gezeigten nicht eine Menge Spaß haben können!

Gehen wir zurück zu unserem Original Programm und zeichnen einen Kreis mi 5-pixel Durchmesser. Wir zeichnen den Kreis an die Location (Lokalisation) 30, 30. Computerdisplays starten mit den Koordinaten an der linken oberen Ecke, Location 0, 0. Von hier aus etwas weiter rechts darzustellen bedeutet, die x-Koordinate zu erhöhen (die erste Zahl in der Koordinatenangabe. Die Angabe 30,0 beschreibt also einen Punkt, 30 Pixel von der linken oberen Ecke entfernt, in der obersten Punktzeile des Bildschirms.

```
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);
            }
        }
    }
}
```

Und… wir haben einen Kreis!



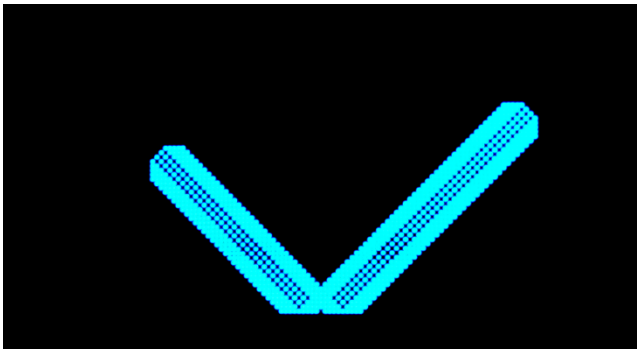
In unserer Endlosschleife erhöhen wir nun die x- und y-Variable bei jedem Durchlauf. Da wir mit 30 angefangen haben, wird der Wert 31, 32, 33, 34 … usw. sein. Wird es jetzt spannender?

Der Kreis bewegt sich sehr langsam wegen der ganzen Sekunde Verzögerung (Wait.Seconds(1)). Wir ändern die Verzögerung jetzt auf 0.01. Außerdem fügen wir etwas Code hinzu, um den Kreis/Ball abprallen zu lassen. Zum Abprallen benötigen wir zwei neue Variablen, dx und dy. Diese enthalten die Richtung, in die der Ball sich bewegt. Wir prüfen dann, ob der Ball an einer Begrenzung angekommen ist. Ist dies der Fall, kehren wir die Richtung um.

```
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);
            }
        }
    }
}
```

Der Kreis, immer neu gezeichnet, wie er sich über den Schirm bewegt:



Wir wollen aber einen abprallenden Ball sehen, also, löschen wir den Bildschirminhalt, bevor wir einen Kreis zeichnen. Füge diese Zeile genau vor DrawCircle ein.

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

Soll der Ball etwas schneller sein? Die verwendete Verzögerung von 0.01 ist sehr gering, kann somit nicht das Problem sein. Wir bewegen den Kreis nur um ein Pixel in jedem Durchlauf. Nehmen wir stattdessen 5, wird der Ball 5 x schneller sein. Das bewerkstelligen wir, indem den initialen Wert für dx und dy auf 5 setzen.

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

Welcher Ball bewegt sich schon ohne Lärm zu machen? Lasse das BrainPad bei jedem Anprall ”Beep” machen. Füge diese Zeile in jeden “if “ Ausdruck ein.

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

Aha! Jetzt ist es ein richtiger abprallender Ball! Aber er ist zu laut, vor allem in einem Klassenraum mit 30 BrainPads! Füge eine Bedingun, nur zu ”beepen”, wenn die Down-(D)Taste gedrückt wird, hinzu.

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

Kannst du dem Ball den coolen Effekt geben, beim sich Bewegen größer und kleiner zu werden? Denke darüber nach, wie wir die Position des Balles in festgelegten Grenzen halten können? Genau das werden wir jetzt machen. Du brauchst eine Variable r für den Radius und eine Variable dr für die Richtung des Radius, wachsend oder schrumpfend.

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

Wir lassen dann die Größe ansteigen mit einer Begrenzung von 1 und 8.

```
r = r + dr;

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

Natürlich müssen wir auch die `DrawCircle` ändern und die Variable `r` anstatt 5 verwenden.

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

Hier ist der vollständige Code. Aber achte auf das Folgende, bevor du den Code per Copy and Paste einfügst. Falls du dein Projekt anders als HelloWorld (keine Leerschritte zwischen Worten, Berücksichtigung von Groß- und Kleinschreibung) genannt hast, wirst du einen anderen vorgegebenen NameSpace (Namensraum) haben. Du solltest dann den vorgegebenen NameSpace deines Programmcodes beibehalten und nicht den NameSpace aus dem unten stehenden Code kopieren. Wenn zum Beispiel dein Programmcode den folgenden NameSpace hat:

```
namespace SomethingElse
```

Dann behalte die namespace Zeile bei und kopiere nur den Rest des Codes, also alles von der Zeile class Program bis einschließlich der zweitletzten schließenden geschweiften Klammer (}). Die letzte geschweifte Klammer ist Teil des beibehaltenen 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);
            }
        }
    }
}
```

EIN WEIHNACHTSLICHT

Du hast richtig gelesen. Wir werden ein Weihnachtslicht erzeugen, mit einer einzigen Light-Bulb (Glühbirne) LED. Hierzu benutzen wir eine der tausend in TinyCLR enthaltenen Funktionen. Die betreffende Funktion erzeugt Zufallszahlen. Anders als im wirklichen Leben, ist in einem Computer nichts wirklich zufällig. Hierdurch kann es schwierig sein, eine Zufallszahl zu erzeugen. Erfreulicherweise gibt es in TinyCLR einen eingebauten Zufallszahlengenerator, das macht es uns leicht. Wir erzeugen ein Zufalls- Objekt und rufen von diesem Objekt immer den nächsten Zufallswert ab.

Wir benutzen diese Zufallszahlen, um die Farben der Light-Bulb primär auf Rot, Grün und Blau zu setzen. Falls du es bisher nicht wusstest, aus diesen drei Farben kann man jede gewünschte Farbe zusammensetzen.

```
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);
    }
```

```
}
```

BEOBACHTE DAS LICHT

Angenommen, wir wollen in einem Raum eine Pflanze aufstellen und wir wissen nicht genau, wieviel Licht in der betr. Ecke des Raumes ankommt. Um einen Eindruck zu bekommen, wollen wir daher die Lichtintensität über den Tagesverlauf in Form einer Grafik darstellen.

Die Grafik ist sehr einfach. Im Prinzip zeichnen wir eine vertikale Linie, die bei der gemessenen Lichtintensität startet und nach unten zur unteren Begrenzung des Displays führt. In jedem Durchlauf der Schleife wird die Linie ein Pixel weiter rechts neu gezeichnet.

```
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);
    }
}
```

Wahrscheinlich willst du, dass das Programm stoppt, wenn es die rechte Displaybegrenzung erreicht hat. In diesem Beispiel jedoch löschen wir den Bildschirm und starten ganz links von neuem. Lassen wir die Schleife schnell durchlaufen, so dass wir schnell ein vorläufiges Ergebnis sehen. Wenn du den Verlauf über 2 Stunden darstellen

willst, und das Display in der Breite 128 Pixel aufweist, brauchst du eine Verzögerung von einer Minute für jeden Schritt. Es werden so 128 Minuten über die Breite des Bildschirms aufgezeichnet.

Du kannst das Programm auch so lassen wie es ist und den Lichtsensor mit der Hand bedecken, um coole Darstellungen zu bekommen!

Weil wir jetzt schon Experten als Coder sind, … kannst du die folgenden Fragen beantworten?

1. Kannst du sagen, warum wir den Wert der Lichtintensität durch 2 teilen?
2. Kannst du sagen, warum wir als Start der Linie nehmen: `BrainPad.Display.Height – light` ?
3. Kannst du sagen, warum x mit 300 initialisiert wird? Was passiert, wenn wir statt dessen 0 nehmen?

Hier sind die Antworten, aber versprich mir, dass du es zuerst selbst versuchst. Die Werte der Lichtintensität liegen irgendwo zwischen 0 und 100. Das Display hat eine Höhe von 64 Pixel. Wenn wir die Werte durch 2 teilen, wird der maximale Wert 100 / 2, also 50 sein. Bei einem 64 Pixel hohen Display haben wir 14 Pixel am oberen Rand übrig. Hierhin schreiben wir “Light Level” (Lichtintensität).

Die zweite Frage bezieht sich darauf, wie wir stärkere Lichtintensitäten darstellen wollen. Kleinere y-Koordinaten werden am oberen Ende des Displays dargestellt. Wir wollen jedoch kleine Lichtintensitäten am unteren Ende des Displays dargestellt haben. In dem wir den Wert der Lichtintensität von der Höhen-Pixelanzahl des Displays subtrahieren, invertieren wir die Grafik.

Das dritte ist ein cooler Trick, um sich zu helfen, wenn der Code das erste Mal läuft. Die Beschriftung “Light Level” wird erst geschrieben, bevor wir den Bildschirm aktualisieren (Refresh Screen). So erreichen wir, indem wir X auf einen großen Wert außerhalb der Begrenzung des Displays setzen, dass die Beschriftung sofort dargestellt wird. Wenn du den Wert für die Initialisierung versuchsweise auf 0 setzt, wird die Beschriftung nicht erscheinen, solange der Graph zum ersten Mal erstellt wird. Die Beschriftung wird erst erscheinen, wenn der Bildschirm aktualisiert (refreshed) wird, nachdem die erste Grafik erstellt ist. Du könntest die Beschriftung auch in jedem Schleifendurchlauf ausgeben, das würde allerdings dein Programm verlangsamen. Wir sollten beim Programmieren immer “smart” denken und nicht mehr Code-Funktionen ausführen, als wirklich nötig sind.

MULTITASKING -- NEBENLÄUFIGKEIT

Ein System zu veranlassen, mehrere Dinge gleichzeitig zu tun, ist intern eine sehr komplexe Aufgabe. Meistens sind diese Funktionen auf kleinen Systemen nicht benutzerfreundlich gelöst. Die gute Nachricht ist: TinyCLR OS erlaubt Multitasking,… und es ist einfach zu benutzen. Es wird genauso gehandhabt wie auf größeren Systemen, die in .NET programmiert werden. Multitasking wird als “Threading” bezeichnet, wobei die “Threads” (Thread = Faden) individuelle Befehlsfolgen darstellt, die vom Betriebssystem in schnell abwechselnder Folge ausgeführt werden.

Keine Ahnung wozu man Threading braucht? Wie würdest du zum Beispiel auf dem Display zählen und gleichzeitig die Light-Bulb blinken lassen? Klar, wenn du den Zähler nur mit jedem Blinken erhöhen wolltest,… kein Problem.

```
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;
    }
}
```

Jetzt ändere das Programm so, dass es nur einmal pro Sekunde blinkt, aber so schnell wie möglich zählt. Das geht kaum ohne Threading. Überführe zunächst den Blinking Code in eine eigene Methode (Unterprogramm). Eine method ist in C# ein Stück Code, das eine spezifische Aufgabe ausführt. Du rufst die method auf, um diese Aufgabe auszuführen. Zum Beispiel hat die Light-Bulb u.a. eine method, die mit TurnGreen bezeichnet ist. Das Aufrufen der method bewirkt, dass die Light-bulb grün leuchtet.

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

Das Aufrufen der im obigen Kasten dargestellten method Blink bewirkt, dass die Light-Bulb einmal pro Sekunde blinkt. Wegen der Endlosschleife wird die method jedoch nie beendet werden und nie zu dem Code zurückkehren (return), aus dem sie aufgerufen wurde. Das ist auch so in Ordnung, weil wir diese method als in einem separaten Thread laufend aufrufen werden. Bist du soweit, den komplizierten Code zu schreiben, der hierfür nötig ist? Hier ist er!

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

Wir haben einfach das Betriebssystem angewiesen, einen neuen Thread für die method Blink zu erzeugen und dann die method Blink in diesem Thread zu starten.

HALT! Bevor wir den Code laufen lassen, fehlt noch eine letzte Änderung. Jeder Thread im System muss eine kurze Sleep bzw. Wait Anweisung enthalten. Das ist notwendig, damit das Betriebssystem (System) seine internen verborgenen Aufgaben erledigen kann und all die anderen Threads möglichst unbeeinflusst von diesem einen Thread ablaufen können. Du wirst vielleicht sagen: … aber wir haben doch nur diesen einen Thread. Nein, tatsächlich haben wir zwei Threads. Das System erzeugt automatisch im Hintergrund einen Thread auf dem Main() abläuft, die Hauptmethode des Programms. Wenn das Programm so schnell wie möglich ablaufen soll, verwende für Wait den kleinstmöglichen Verzögerungswert (Minimum).

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

Hier siehst du beide methods, Main und 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;
    }
}
```

Wir haben zwei Endlosschleifen, die beide quasi simultan ablaufen. Dies ist ein sehr einfaches Beispiel, aber es zeigt die Möglichkeiten. Zur Übung füge dem Programm einen dritten Thread hinzu, der alle drei Sekunden einen Beep Ton erzeugt.

CALL ME – RUF MICH

Wir haben bereits die Buttons (Druckschalter) benutzt, du weißt also wie sie funktionieren. Allerdings haben wir die Buttons in einer Schleife benutzt, in der wir immer abgefragt haben, ob der Schalter gerade gedrückt wird. Es kann sein, dass sein, dass das Programm den Schalter millionenmal abfragen muss, bevor er irgendwann einmal gedrückt wird. Dieses Vorgehen ist sehr ungünstig für batteriebetriebene Geräte, da dieses Vorgehen viel Strom verbraucht. In einem guten Programmdesign wird das System zur Schonung der Batterie so häufig wie möglich in einen stromsparenden Sleep Modus gebracht, sobald also das Programm gerade keine wichtigen Aufgaben auszuführen hat. Wenn dein Smartphone nicht solche Sleep Funktionen nutzen würde, wäre der Akku bestimmt schon nach einer Stunde leer.

Dieses Thema kann sehr vielschichtig sein, daher wollen wir uns hier auf die Behandlung von Button Events beschränken. Anstatt fortlaufend den Schalter abzufragen, wollen wir das System veranlassen, uns eine Nachricht zu schicken, sobald der Knopf gedrückt wurde. Bei jedem Drücken des Knopfes soll ein Beep Ton erzeugt werden, außerdem soll die Light-Bulb blinken.

```
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();
        }
    }
}
```

Drücke kurz den Button, … nichts wird passieren. Halte den Button gedrückt, … ein Beep Ton pro Sekunde wird erzeugt. Dieses Verhalten ist zu erwarten weil die Schleife Delays (Verzögerungen) enthält und der Button nur einmal pro Sekunde abgefragt wird. Auch wenn du die Delay Zeit verminderst, wird dies das Problem nicht lösen. Als Alternative werden wir jetzt sog. Events (Ereignisse) verwenden. Visual Studio kann automatisch den erforderlichen Code erzeugen. Beginne mit dem Eintippen von "`BrainPad.Buttons.WhenDownButtonPressed`", dann tippe (evtl. nach Leerschritt) "+=". Jetzt kannst du die Tab (Tabulator) –Taste drücken und Visual Studio wird automatisch den Code für die Event method für dich erzeugen.

Die erzeugte Event method sieht wie folgt aus:

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

Der erzeugte Code enthält eine Zeile zum Aufruf einer Fehler Ausnahmenbehandlungsfunktion (error exception). Das ist nur ein Platzhalter, der bewirken soll, dass wir nicht vergessen, hier den auszuführenden Code einzusetzen. Ersetze die Zeile durch die Beep Funktion.

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

Vergesse nicht, den Code in der Main Schleife, der den Button abfragt, zu entfernen.

```
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();
}
```

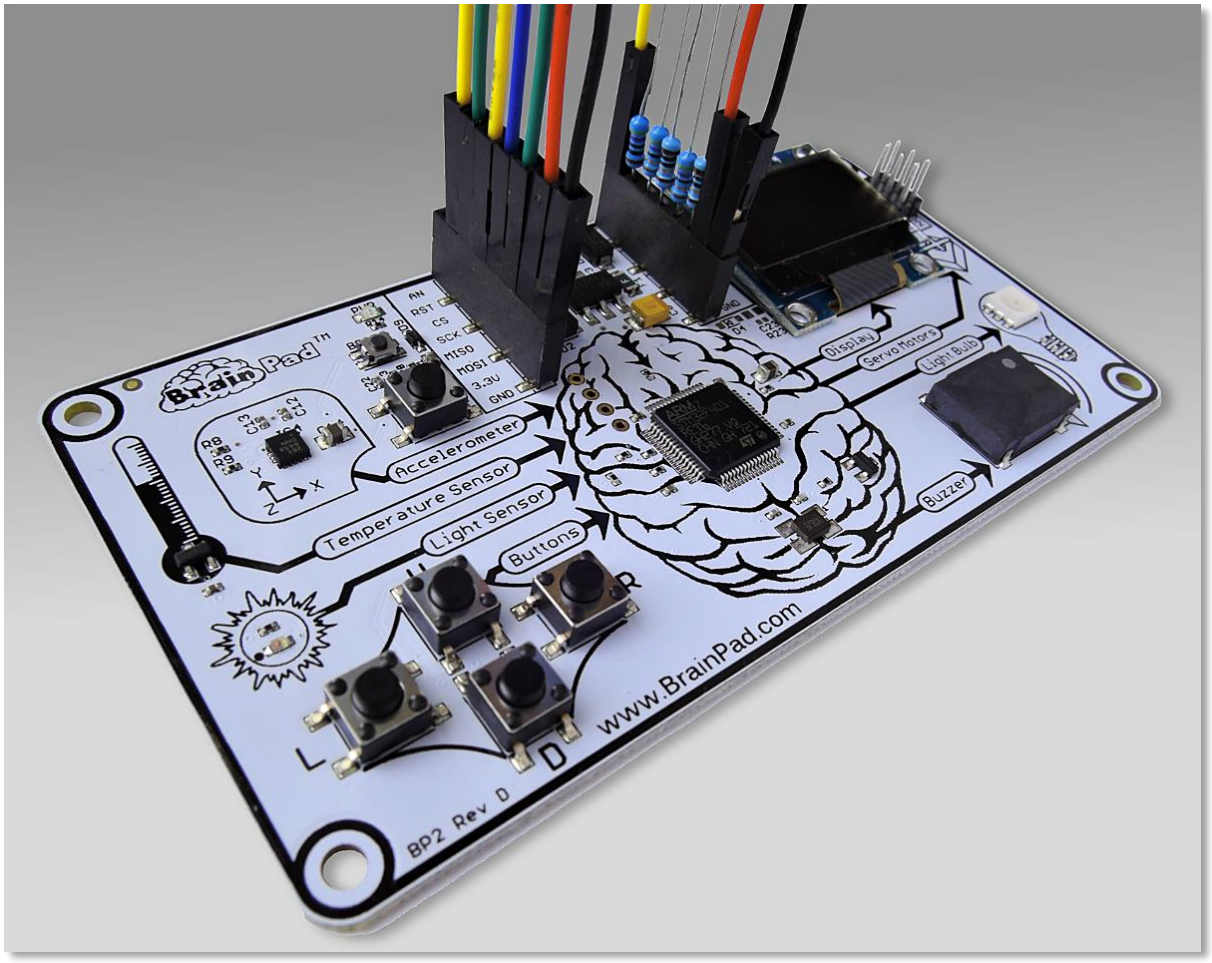
Nebenbei bemerkt, mehrere Events können den gleichen event handler aufrufen. Hier ist der Code für “beepen“ wenn irgendeine der vier Tasten gedrückt wird.

```
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();
}
```

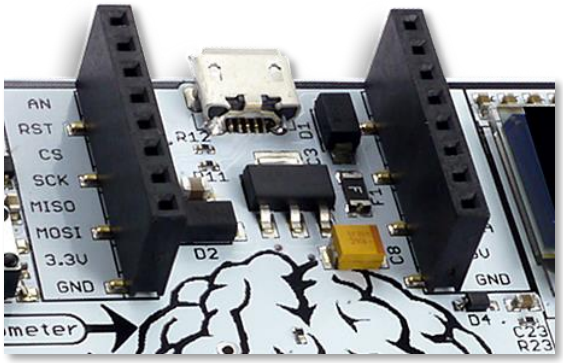
SCARY WIRES – UNHEIMLICHE KABEL

Es ist ganz normal , dass Leute zu Beginn der Beschäftigung mit Elektronik bei der Vorstellung, etwas verkabeln zu müssen, eingeschüchtert und ängstlich sind. Was ist, wenn ich einen Schlag bekomme? Was ist, wenn ich die Schaltkreise zerstöre?

Bei dem BrainPad ist es weniger unheimlich. Vorweg, … das BrainPad läuft mit einer sehr niedrigen Spannung von 5 Volt (5V). Es kann normalerweise nichts passieren, wenn man Teile mit 5 V Spannung berührt. Außerdem hat das BrainPad eingebaute Schutzvorrichtungen und wurde so konstruiert, dass es sehr unwahrscheinlich ist, es durch eine falsche Verkabelung zu beschädigen. Trotzdem solltest du immer verstehen, was du gerade machst. Es gibt viele Kurse und Online Artikel, die dir helfen können, elektronische Schaltungen zu verstehen.

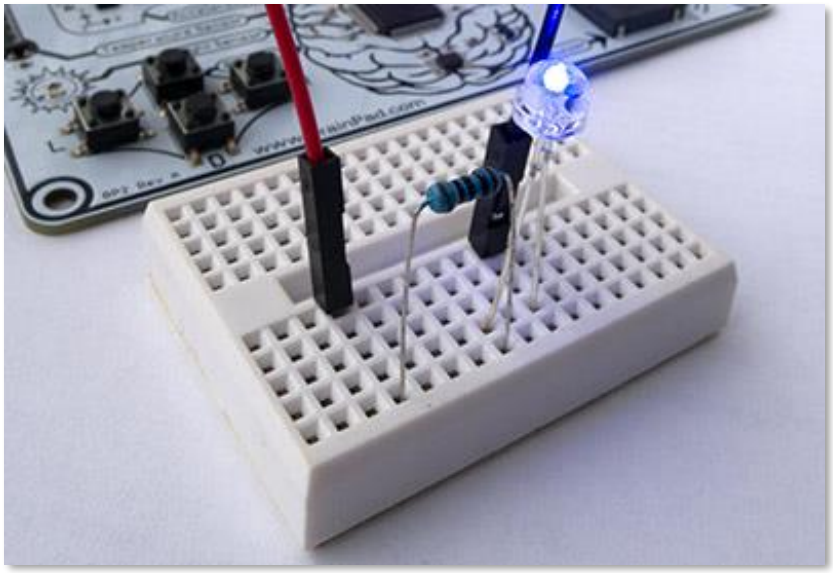


Wie bereits im ersten Kapitel dieses Buches erwähnt, gibt es verschieden Wege, das BrainPad zu erweitern. Hierzu dienen die Buchsenleisten (female headers) in der Nähe des USB Anschlusses.

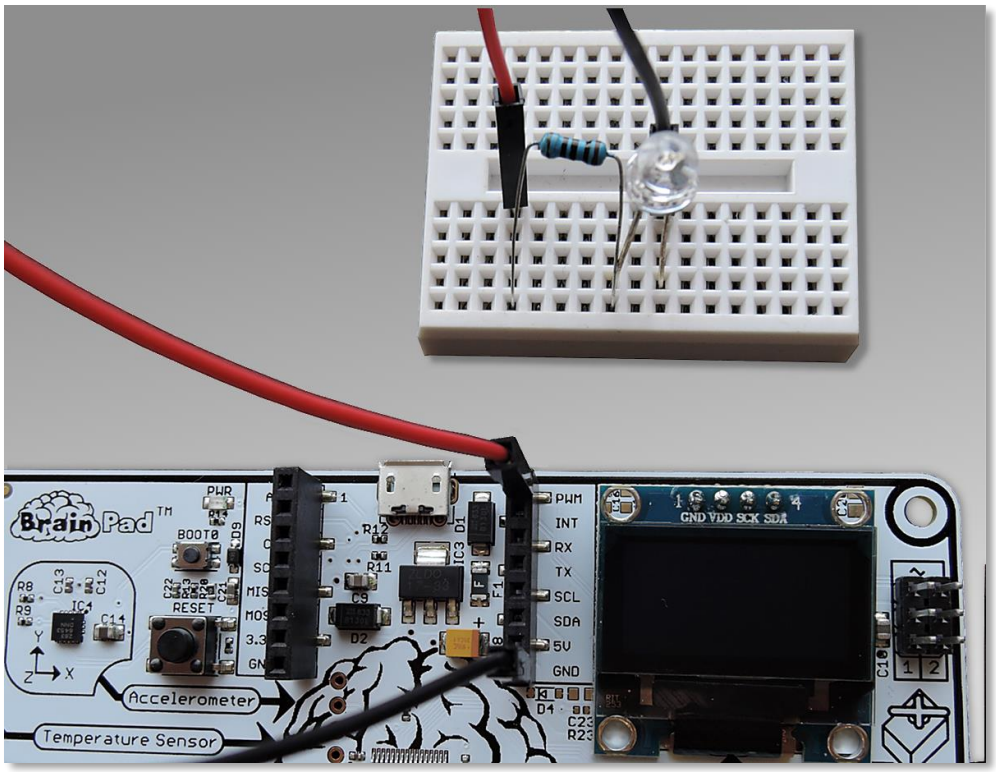


Diese Anschluss Pins haben viele Funktionen und man könnte einige Bücher darüber schreiben, um alles umfassend zu erklären. Wir werden hier nur die Oberfläche ankratzen, indem wir eine einfache LED (Light Emittierende Diode) anschließen.

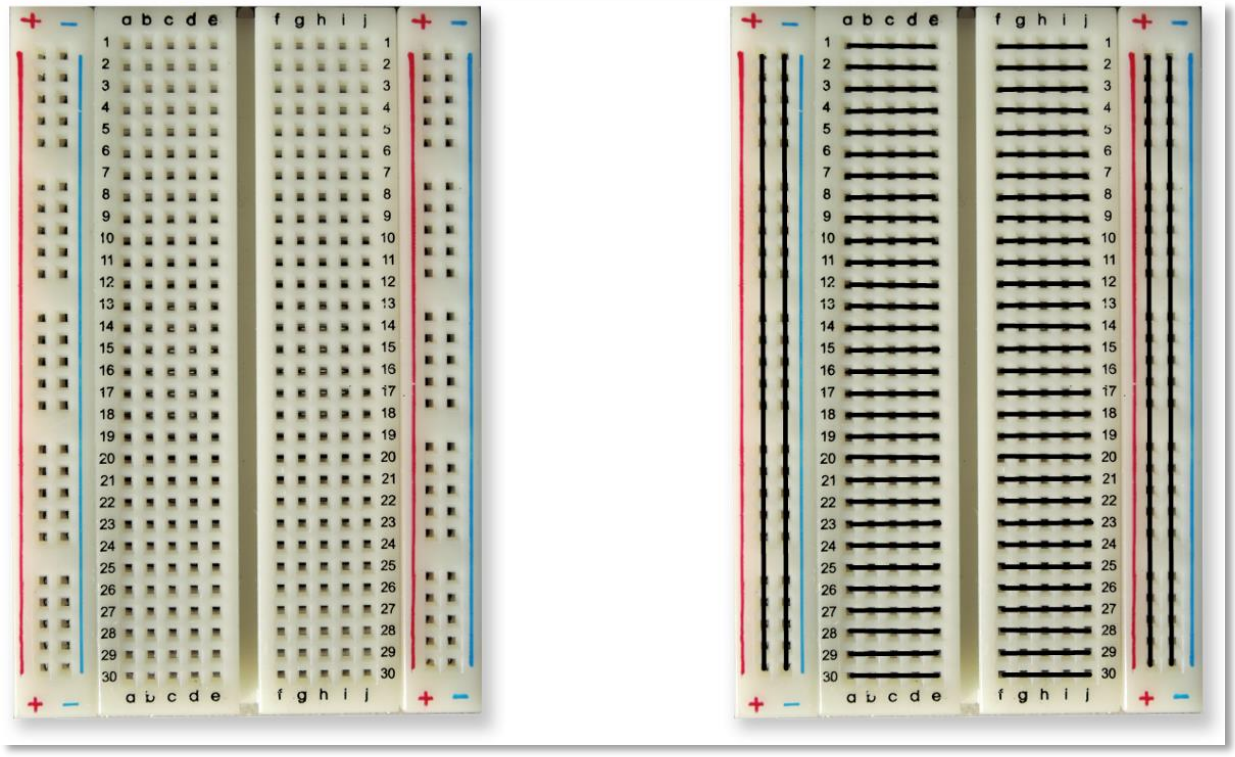
LEDs sind sehr preiswert und es gibt sie in verschiedenen Farben. Sie haben zwei Anschlussdrähte, von denen normalerweise einer etwas länger als der andere ist. Manche LEDs haben mehr als eine Farbe (z.B. die BrainPad Light-Bulb). Solche LEDs werden wir hier nicht behandeln. Um eine LED anzusteuern, braucht man zusätzlich einen den Strom begrenzenden Widerstand (Resistor). Gerbräuchliche Werte für den Widerstand eines Widerstands sind 220, 330 oder 470 Ohm. Außerdem brauchen wir noch einige Kabel und ein Breadboard (Steckplatine).



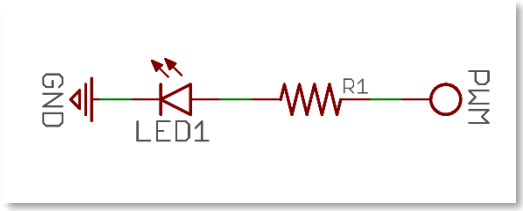
Wir können einen der GPIO pins am expansion header (Erweiterungsanschluss) verwenden, um die LED anzusteuern. GPIO pin bedeutet General Purpose Input Output pin oder auf Deutsch: Mehrzweck Eingang/Ausgang pin. Wir werden den Pin mit der Beschriftung PWM verwenden.



Breadboards verfügen über interne elektrische Verbindungen, wobei jeweils alle Löcher einer Reihe intern elektrisch miteinander verbunden sind, die Löcher einer Spalte jedoch nicht miteinander verbunden sind. Welches auf speziell deinem Breadboard die Reihen und welches die Spalten sind, musst du selbst herausfinden. Auf dem Bild unten seht ihr auf der rechten Seite ein Breadboard, bei dem durch schwarze Linien angezeigt wird, welche Löcher elektrisch verbunden sind. Alles was in die Löcher, die durch die schwarzen Linien verbunden sind, eingesteckt wird ist elektrisch miteinander verbunden.

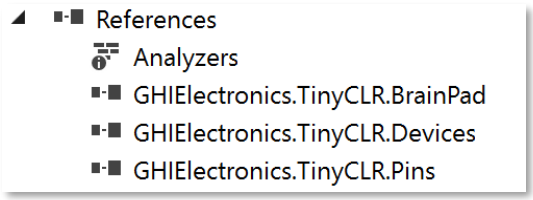


Ein Kabel wird vom PWM pin des BrainPad zu der elektrisch verbundenen Kontaktreihe des Breadboard geführt, wo auch der längere LED Draht eingesteckt ist. Der kürzere Draht der LED wird mit dem einen Anschlussdraht des Widerstands verbunden, der andere Anschlussdraht des Widerstands wird über ein zweites Kabel mit dem GND pin (Ground, deutsch auch: Masse) des BrainPad verbunden. GND entspricht hier dem Minus-Pol der Schaltung. So ist ein Stromkreis zum Fluss von Elektronen vorhanden, in dem Strom von PWM zu GND fliesen kann. Sobald der PWM pin aktiviert ist, wird die LED aufleuchten.



Um den PWM pin (GPIO pin) anzusteuern, müssen die GHIElectronics.TinyCLR.Devices und GHIElectronics.TinyCLR.Pins NuGet libraries hinzugefügt sein. Diese libraries werden von der GHIElectronics.TinyCLR.BrainPad library benötigt, die gleichfalls hinzugefügt sein muss. Wenn wir die

GHIElectronics.TinyCLR.BrainPad library hinzufügen, werden die beiden oben genannten anderen libraries automatisch mit hinzugefügt, es ist also keine weitere Aktion erforderlich.



Wir brauchen Zugriff auf die GPIO libraries in unserem Code. Das erreichen wir durch diese Code-Zeile.

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

Wir müssen über den im Prozessor befindlichen GPI Controller auf einen pin zugreifen

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

Pins können jeweils als Input (Eingang) oder als Output (Ausgang) fungieren, genau wie die Ein- und Ausgaben des BrainPads. Inputs gehen in das “brain” (Gehirn), Outputs kommen aus dem “brain” heraus. Ein Beispiel für einen Input ist ein Button, wie sie sich auch auf dem BrainPad befinden. Ein Beispiel für einen Output ist eine LED, wie die Light-Bulb. Demnach muss der PWM pin als Output erstellt werden.

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

Jetzt können wir einfach auf diesen pin schreiben (“write”), um den Ausgang auf High (aktiviert) oder Low (deaktiviert) zu schalten. Füge das zusammen mit ein paar Verzögerungen in eine Schleife ein, und schon haben wir eine blinkende LED.

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

Die fertige main method sollte wie folgt aussehen:

```
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);
    }
}
```

Einen Button anzusteuern, gestaltet sich ähnlich, mit der Ausnahme, dass der pin natürlich ein Input und nicht ein Outputs sein muss. Ohne hier auf die Details einzugehen, der Pin muss genauer gesagt ein Input mit “pull up“ sein. Durch die pull up option wird der Pin im Ruhezustand auf High gehalten, durch Drücken des Button auf Low gezogen.

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

Es ist nicht nötig, den Pin durch einen externen Widerstand auf High zu halten. Verbinde einfach einen der Anschlüsse eine Buttons mit einem der GPIOs, den anderen Anschluss mit GND.

ZUSAMMENFASSUNG

TinyCLR von GHI Electronics bringt wirklich professionelles Programmieren auf das BrainPad. Die Kenntisse, die du hier erwerben wirst, sind nützlich für die Programmierung einer Vielzahl von Geräten, vom Smartphone bis zum PC. Alles wird leicht gemacht durch unser TinyCLR Betriebssystem und Microsoft’s .NET Framework.

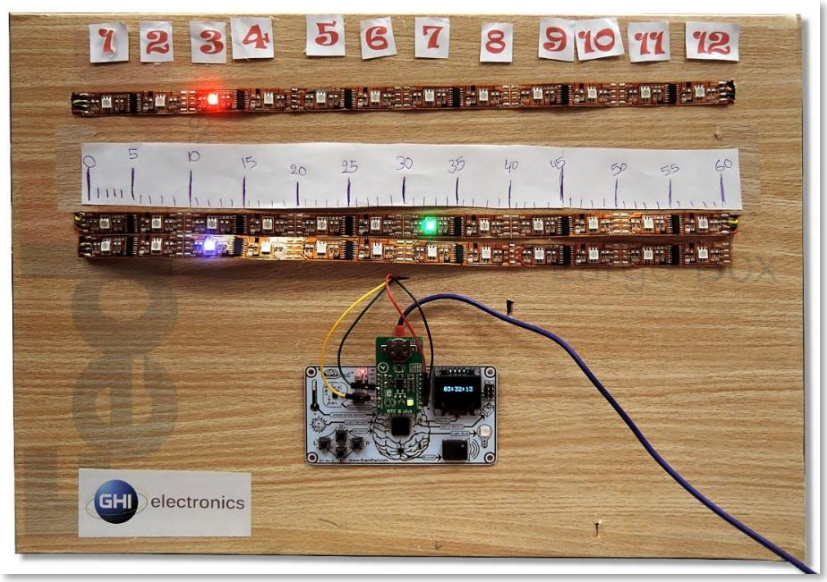
ERWEITERBARKEIT

Der Erweiterungsanschluss (expansion header) eröffnet eine ganze Welt an Möglichkeiten. Wir zeigen hier einige Optionen zur Anregung, darüber nachzudenken, was alles möglich ist.

MIKROELEKTRONIKA CLICK BOARDS™

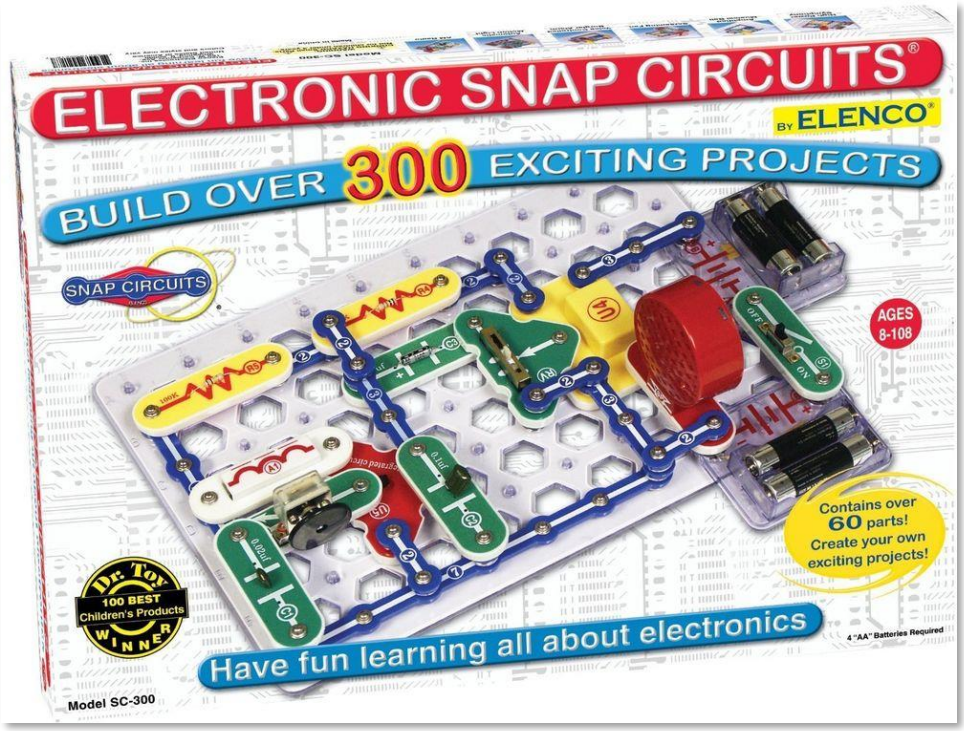
MikroElektronika bietet hunderte von kleinen Modulen an, die genau in den Erweiterungsanschluss des BrainPad passen. Die Module umfassen einfache Sensoren für Luftfeuchtigkeit und Temperatur bis hin zu komplexeren Geräten für z.B. Spracherkennung und Elektrokardiogramm (EKG). Es gibt auch zahlreiche Aktuator- und Control-Module z.B. als Motorcontroller und digitale Lichtsteuerung. Alle Module findest du auf der Website von MicroElektronica https://www.mikroe.com/click .

In diesem einfachen Projekt haben wir eine einfache Uhr erzeugt, die ein Real Time Clock (RTC) Click Modul verwendet. Durch dieses Modul wird die aktuelle Zeit erhalten, auch wenn das BrainPad nicht an die



Stromversorgung angeschlossen ist (https://docs.brainpad.com/projects/linear-clock.html).

ELENCO® SNAP CIRCUITS®

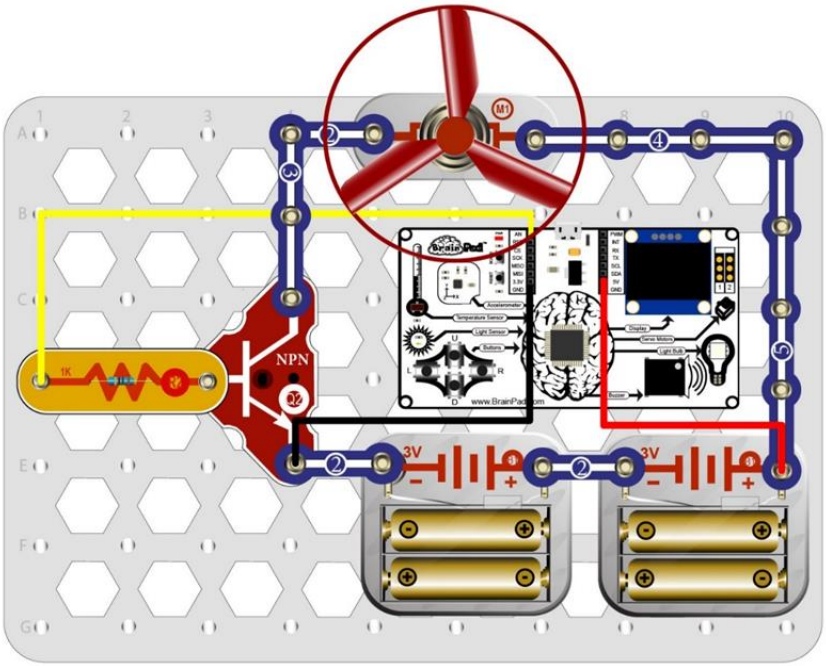


Eines der bekannteren Elektronik Lernprogramme ist Snap Circuits von Elenco (https://www.elenco.com). Die sehr populären Baukästen können im lokalen Handel oder von Online Versendern

erworben werden. Sie enthalten eine Vielzahl elektronischer Komponenten und Anleitungen, um viele verschiedene Projekte zu erstellen.

Durch das BrainPad kannst du die Projekte um Intelligenz und Programmierbarkeit erweitern. Dieses Beispiel ist ein Countdown Zähler, der bei Ablauf einen Propeller ansteuert. Schau dir auf jeden Fall das zugehörige Video an. Es ist eines unserer Lieblingsvideos.

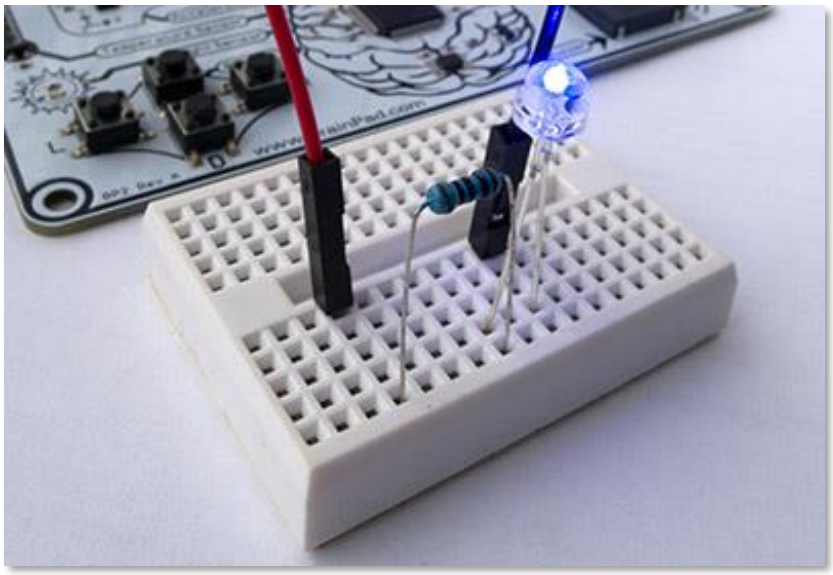
(https://docs.brainpad.com/projects/lift-off.html).



BREADBOARDS

Willst du eigene elektronische Schaltungen von Null an (from scratch) entwerfen, so wie es unsere Ingenieure machen? Es gibt viele Wege zum Entwurf von Schaltungen, die mit dem BrainPad zusammenarbeiten. Der Entwurf von Schaltungen erfordert zwar einiges an Hintergrundwissen, er ist aber die kreativste und preiswerteste Möglichkeit, dein BrainPad zu erweitern. Der eigene Entwurf hat auf jeden Fall den größten Lernwert.

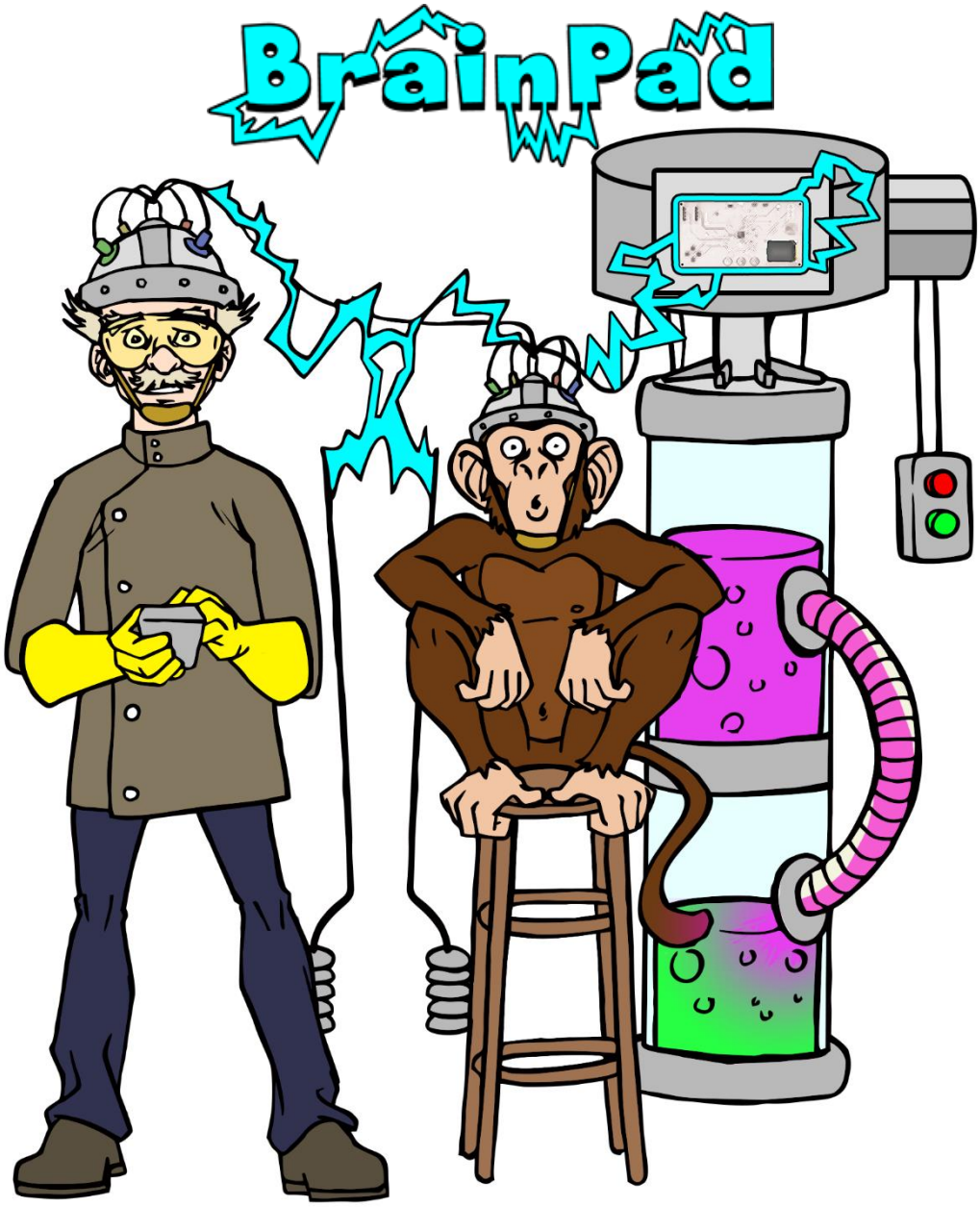
Der einfachste Weg zur Herstellung eigener Schaltungen ist die Verwendung von Breadboards, da man hierbei nicht löten muss. Kabel und die Anschlüsse von Bauteilen werden einfach in die Löcher des Breadboards gesteckt. Das Bradboard hält die Komponenten am Platz und sorgt für die elektrische Verbindung. Das ist nicht nur der schnellste Weg um eine neue Schaltung zu erstellen, auch Fehler in Schaltungen lassen sich so am einfachsten korrigieren.



Wenn du eine Schaltung als dauerhafte Anwendung haben willst, gibt es mehrere Möglichkeiten für den Aufbau. Die Palette reicht vom einfach zu handhabenden vorgefertigten Prototyping Board bis zum Design einer speziellen Platine. Das Design von Platinen kann von Hand oder mittels zum Teil kostenloser Computer Programme erfolgen. Das alles ist das Besondere am BrainPad – es ist einfach, mit dem Board loszulegen. Dann kannst du fortschreiten so weit wie du willst und so schnell wie du willst

Du kannst auch einen oder mehrere der erhältlichen Sensoren in deinem BrainPad Projekt verwenden. Wenn du im Web nach “sensor kit” suchst, wirst du viele preiswerte Sensor Sortimente finden. Hier ist eine Auswahl:

Für die Verwendung dieser Module ist einiges an Vorwissen erforderlich. Wir haben aber einige leicht nachzubauende Projekte, die dir eine Hilfe geben, wie am besten vorzugehen ist. Wenn du erst einmal mit einem Sensor umgehen kannst, wird es viel leichter, zu verstehen, wie andere Sensoren arbeiten.



www.BrainPad.com