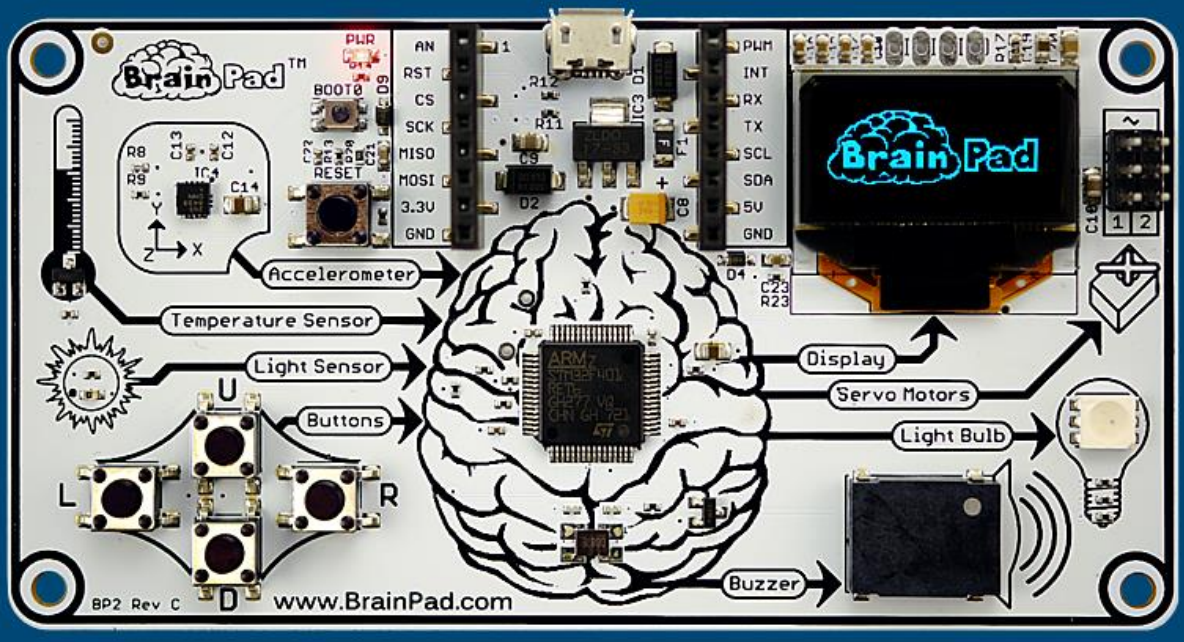
GHI electronics

# BrainPad™

## BEGINNERS' GUIDE




GHI electronics

## CONTENTS

# ВСТУПЛЕНИЕ

BrainPad разработан как мощный STEM-инструмент для обучения людей всех возрастов от детей младшего возраста до студентов и профессионалов.

Это бесплатная электронная книга является руководством для новичков в мире BrainPad. Больше материалов вы можете найти на https://www.brainpad.com.

## АВТОР

Гас Исса – основатель и CEO компании GHI Electronics. Он начал увлекаться программированием и электроникой практически с детства. На первых порах было очень сложно самообучаться из-за ограниченных ресурсов и отсутствия интернета. Эта постоянная борьба за знания привела его на путь обучения других людей. Он потратил бесчисленное количество времени, задействовав ресурсы своей компании, чтобы создать доступный способ поделиться своими знаниями. Результат в BrainPad.

## ГЛОБАЛЬНОЕ МЫШЛЕНИЕ

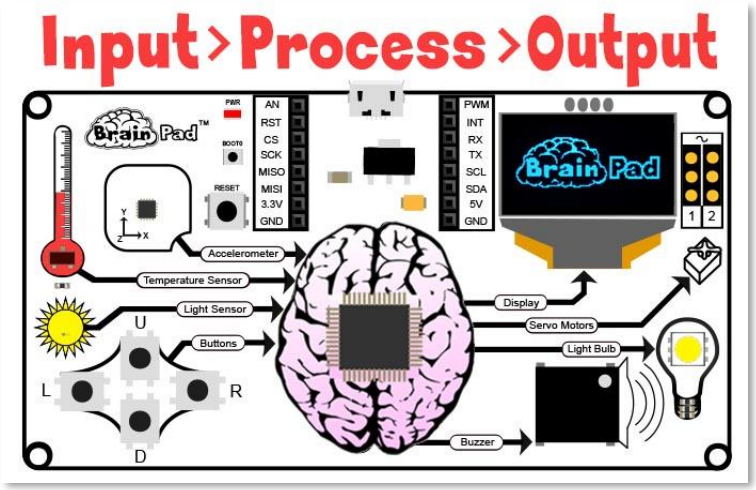
Мы много работаем над тем, чтобы BrainPad стал доступен для каждой культуры и языка. Если вы испытываете трудности и хотите перевести эту книгу на свой язык, пожалуйста, сообщите нам об этом, перейдя по ссылке https://www.brainpad.com/contact-us.

## ЛИЦЕНЗИЯ



## КОНЦЕПЦИЯ

BrainPad не нуждается в длительных инструкциях. Он логически объясняет, как работает компьютер. В нем четыре входа (inputs), которые связаны с мозгом. Мозг (процессор) обдумывает входящую информацию и отображает результат на четырех выходах (outputs).



## STEM

«STEM» — это аббревиатура от Science (наука), Technology (техника), Engineering (инженерия) и Mathematics (математика). Впервые понятие STEM было представлено в 2001 году Национальным научным фондом США. STEM сейчас является одной из самых обсуждаемых тем в образовательной системе Соединённых Штатов Америки. Однако преподавание STEM — это не только эти четыре отрасли в образовании. Образовательный подход STEM направлен на то, чтобы связать обучение в классе с реальным миром, подчеркивая коммуникацию, сотрудничество, критическое мышление и творческий подход в процессе преподавании инженерного проектирования. Термин «STEAM» является тем же, что и STEM с дополнительным акцентом на искусство.

## ФИЛОСОФИЯ

Образовательная система STEM требует развивающих платформ, а не игрушек. Если вы развиваетесь, значит и BrainPad с вами. Существует много игрушек под эгидой инструментов STEM, но большинству из них не хватает многогранности, чтобы поддерживать увлеченность студентов на долгое время. BrainPad заключает в себе программирование (от новичка до профессионала), электронику, инженерное дело, робототехнику и многое другое на недорогой платформе, которую можно использовать со старшей школы до колледжа. Это подтверждается нашим 15-летним профессиональным опытом. Те же инструменты, что используют коммерческие клиенты для работы с нашими промышленными контроллерами, также могут быть использованы для работы с BrainPad. Никакая другая платформа STEM не может привести студентов к программированию от построения программ перетаскиванием блоков к профессиональному программированию и инженерии, как BrainPad.

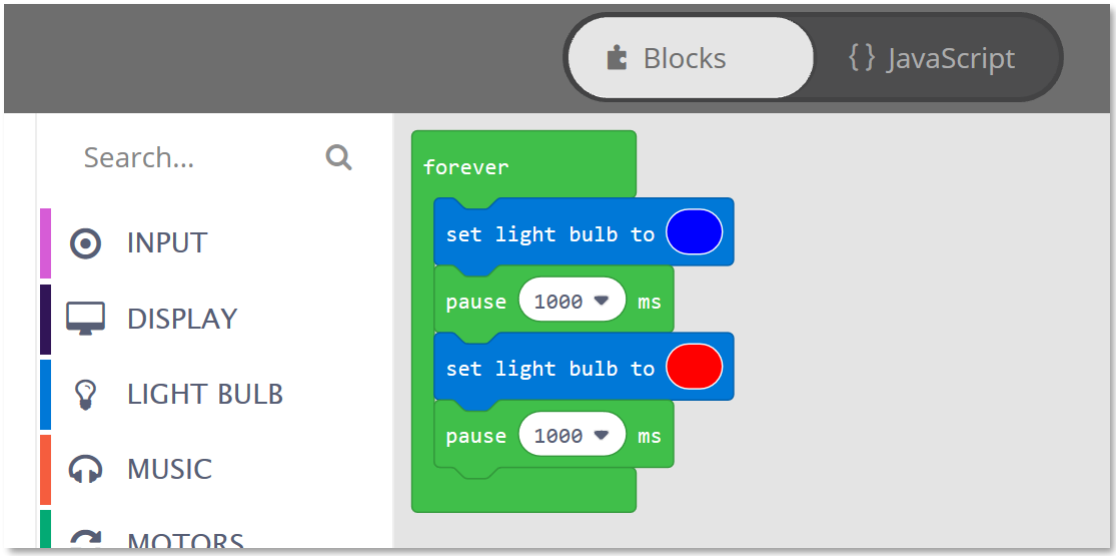
## ДВА ПУТИ

BrainPad предлагает два пути изучения программирования: **Start Making (Начни программировать) и Go Beyond (Выходи за грани).** И как вы догадываетесь, первый путь очень простой, с низким порогом вхождения, а второй предлагает выйти за пределы и начать использовать те же инструменты, что и профессионалы.

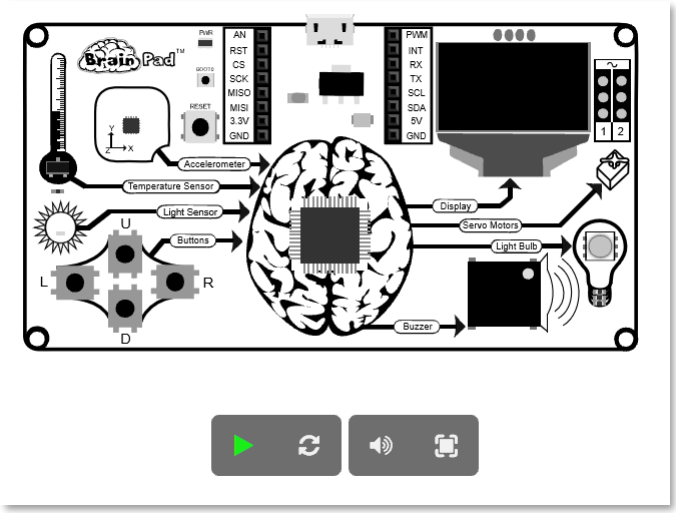
### START MAKING (НАЧНИ ПРОГРАММИРОВАТЬ)

Начиная работу с **Start Making** вам ничего не нужно устанавливать на компьютер – все инструменты уже установлены и готовы к работе через интернет-браузер, используя Microsoft MakeCode. Путь **Go Beyond** требует установки и настройки программного обеспечения. Мы работаем в среде Microsoft Visual Studio.

С Microsoft MakeCode (Start Making) вы можете программировать, просто складывая блоки, или использовать язык программирования JavaScript прямо в вашем интернет-браузере. На рисунке ниже пример программы, зажигающей сначала синий светодиод, а потом красный каждую секунду в бесконечном цикле.



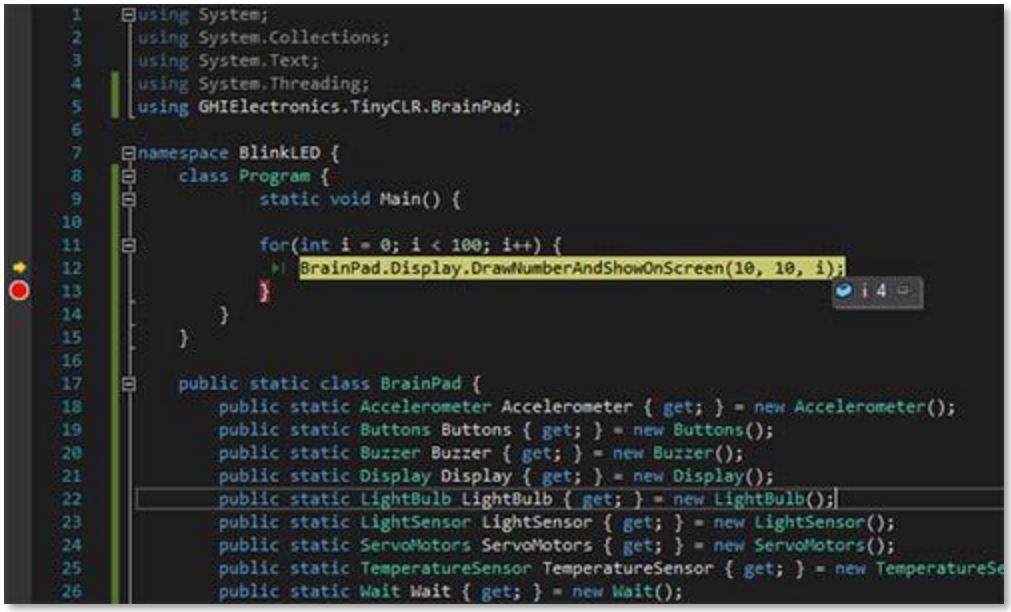
На этом рисунке скриншот симулятора, то есть виртуальный BrainPad. Он запускает программу прямо на дисплее вашего монитора. Находится он в левой стороне экрана в окне Microsoft MakeCode. Симулятор позволяет вам тестировать программы прямо в вашем браузере без загрузки в BrainPad. Это отличная возможность начать работу прямо сейчас, даже если вы еще не получили ваш BrainPad.




## GO BEYOND (ВЫХОДИ ЗА ГРАНИ)

Идя по пути Go Beyond, возможности у вас гораздо серьезней. Вам нужно установить на компьютере Microsoft Visual Studio, чтобы программировать и отлаживать программы, запущенные на BrainPad.

Благодаря операционной системе TinyCLR OS™ от GHI Electronics у вас есть возможность программировать BrainPad в среде .NET на языках программирования C# и Visual Basic. Доступна возможность полной отладки программы, такой как пошаговый запуск кода, контрольной точки и инспекция переменных во время запуска. Вы будете писать программы для BrainPad точно так же, как это делают профессионалы, к примеру, для ПК или мобильных устройств. Эта книга опишет оба путь.



```
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();
```

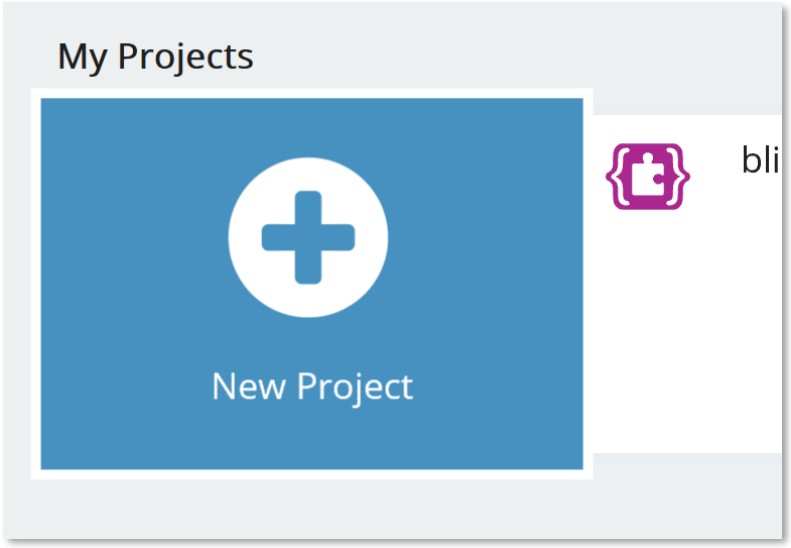
## ИДЕМ ЕЩЕ ДАЛЬШЕ!

Для BrainPad можно так же писать программы, используя другие языки программирования и инструменты, которые не описаны в этой книге. Например, среда разработки Arduino.

https://www.arduino.cc, MicroPython https://www.micropython.org и Arm Mbed https://www.mbed.com.

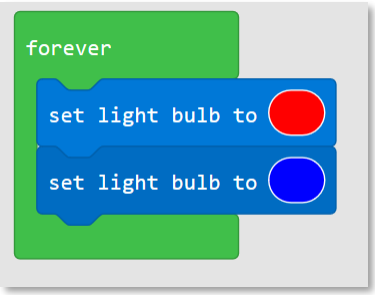
## START MAKING (НАЧНИ ПРОГРАММИРОВАТЬ)

Хватит болтать – начнем работать и получать удовольствие! Мы будем использовать Microsoft MakeCode, так что перейдем по ссылке https://makecode.brainpad.com/. На этом сайте вы найдете много интересного материала и проектов, но мы начнем сразу с нового проекта. Просто нажмите на кнопку New Project.



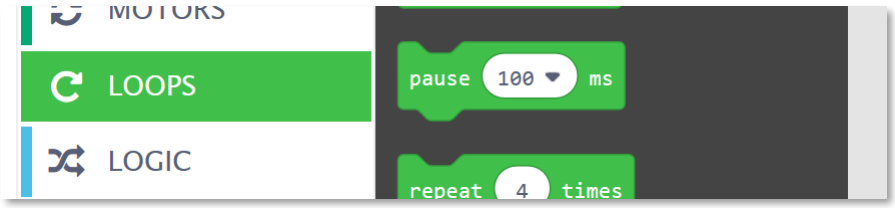

Из вкладки LIGHT BULB, что в главном меню, перетащите блок "set light bulb to" и поместите его в блок "forever". Блок "forever" используется, когда вы хотите, чтобы ваша программа исполнялась все время. Все команды внутри этого блока будут исполнятся BrainPadom, пока он включен. И когда очередь инструкций будет выполнена, BrainPad запустит код снова, начиная с первой инструкции в блоке "forever". Это называется бесконечным циклом и часто используется в программировании.

Повторите этот шаг, но измените цвет во втором блоке, нажав на цветной овал. Вот что у вас должно получиться:

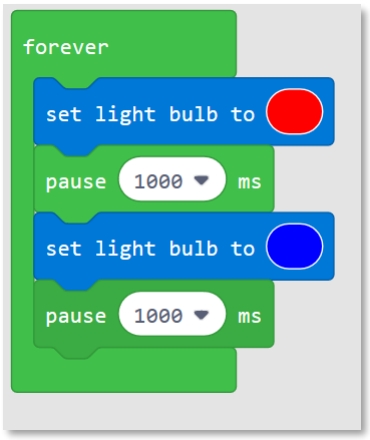



Программа, которую мы создали, должна менять цвет светодиода на красный, а потом опять на синий мгновенно и бесконечно. Если вы посмотрите на симулятор, то заметите, что светодиод будет работать не так, как мы ожидаем. И когда вы загрузите программу на BrainPad, то удостоверитесь, что программа работает неправильно. Но почему? Дело в том, что мы не указали как долго светодиоду нужно подождать перед тем, как изменить свой цвет. Сейчас цвет меняется, но на столько быстро, что глаз не может этого уловить.

Из меню LOOPS перетяните блок "pause".




Нам нужно поставить паузу после включения красного светодиода и после синего. Также давайте изменим время паузы на одну секунду.



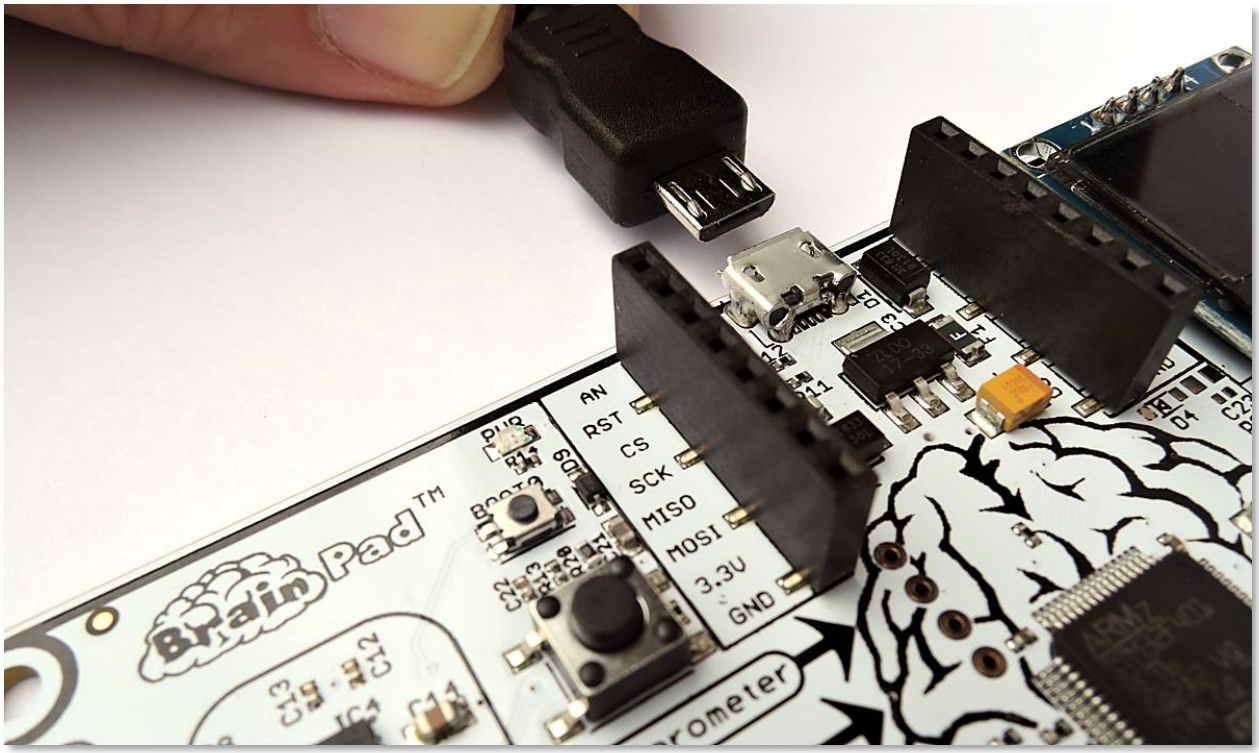

Посмотрите на симулятор и убедитесь, что светодиоды работают как мы и ожидали. Не плохо, правда? Что, если сейчас мы загрузим нашу программу на BrainPad?

## СКОПИРОВАТЬ НА BRAINPAD

Процесс копирования программы на BrainPad довольно прост, но в начале может показаться немного сложным. Для начала вам нужно подсоединить BrainPad к вашему компьютеру, используя Micro-USB кабель. Кстати, вы можете использовать Windows, Chromebook, Mac или практически любое современное устройство, в котором есть интернет браузер и USB порт.

Micro-USB кабель есть в комплекте вашего BrainPad. Хотя, скорее всего у вас уже есть такой дома. Если же вдруг нету, то этот тип кабеля очень распространен и его можно купить практически где угодно по очень доступной цене.

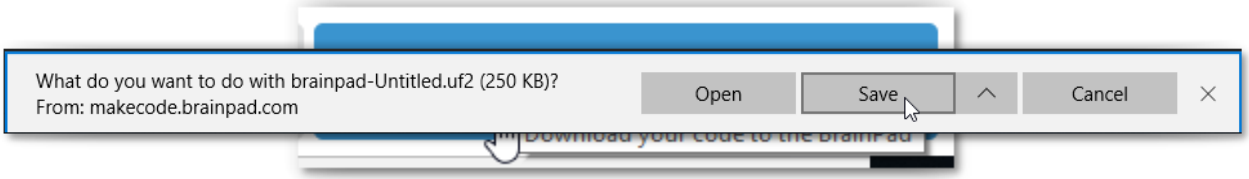
Подключите маленький коннектор к BrainPad.



Большой коннектор к компьютеру.

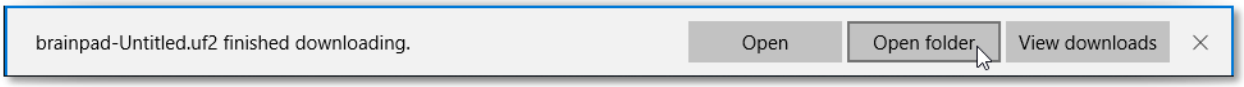

Когда BrainPad подключен к вашему компьютеру, на нем должен загореться красный светодиод.

Вернемся к браузеру с программой, которую мы сделали раньше. Нажмите на кнопку сохранить (save) и сохраните файл на компьютер.

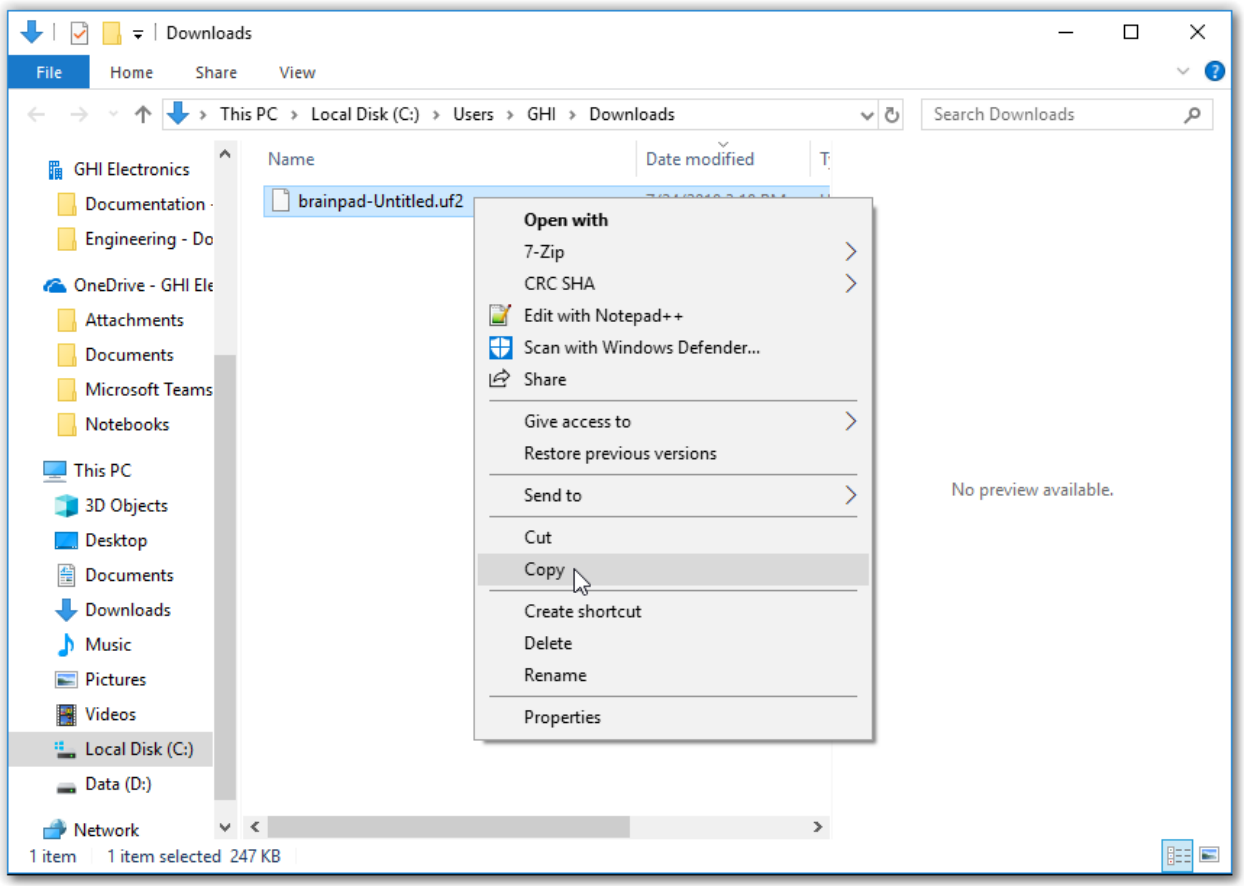



Если вы используете Microsoft Edge, то увидите диалоговое окно как на рисунке ниже:

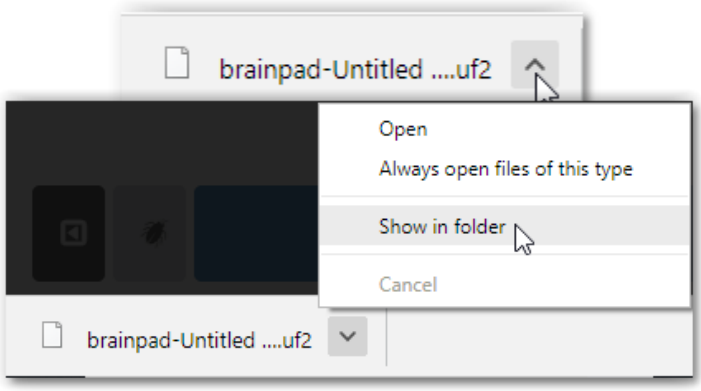
Нажмите на кнопку «сохранить», а за тем "открыть папку" (Open folder) в следующем окне, чтобы увидеть сохраненный файл.




Файл будет подсвечен. Вы можете нажать правую кнопку мышки и выбрать "копировать" (Copy)

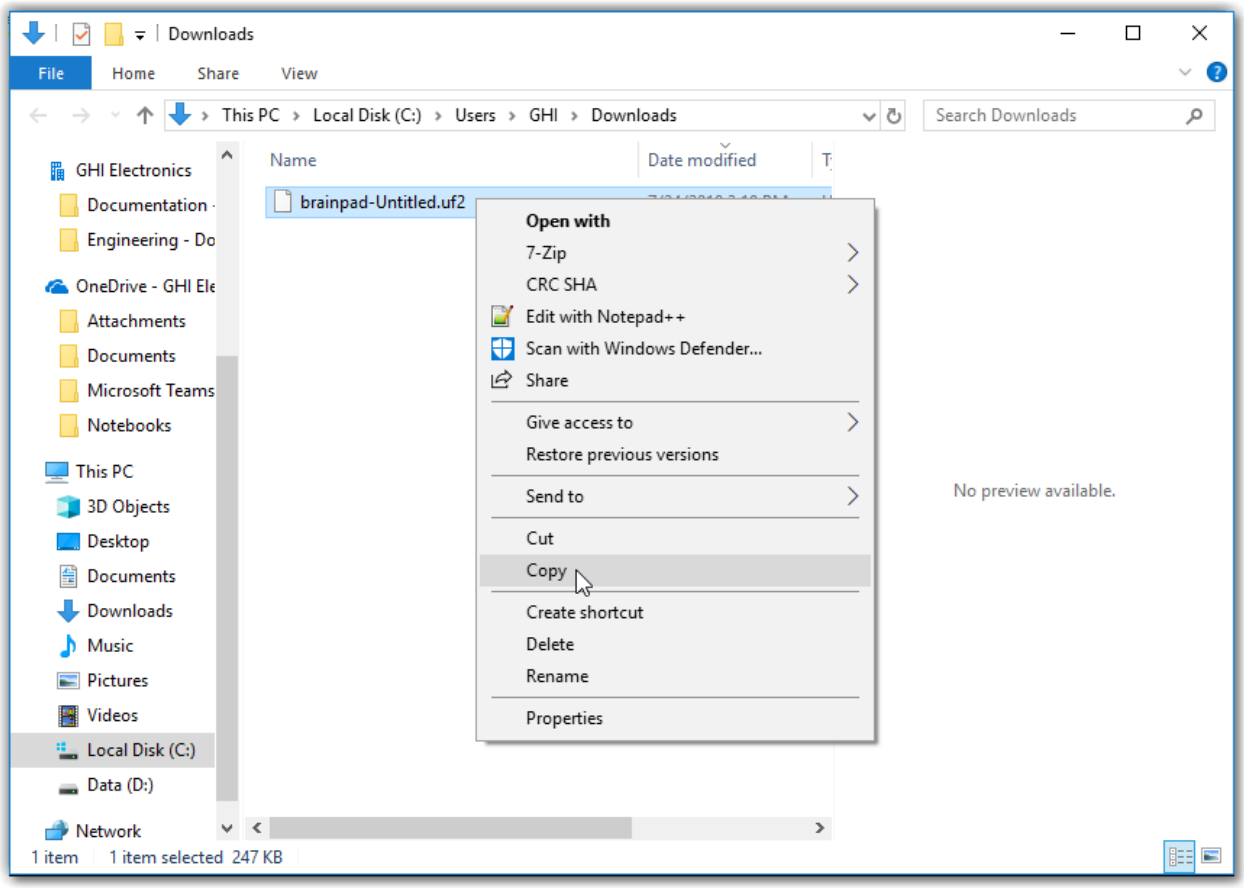


Если вы используете Chrome, файл будет доступен в нижнем левом углу экрана:



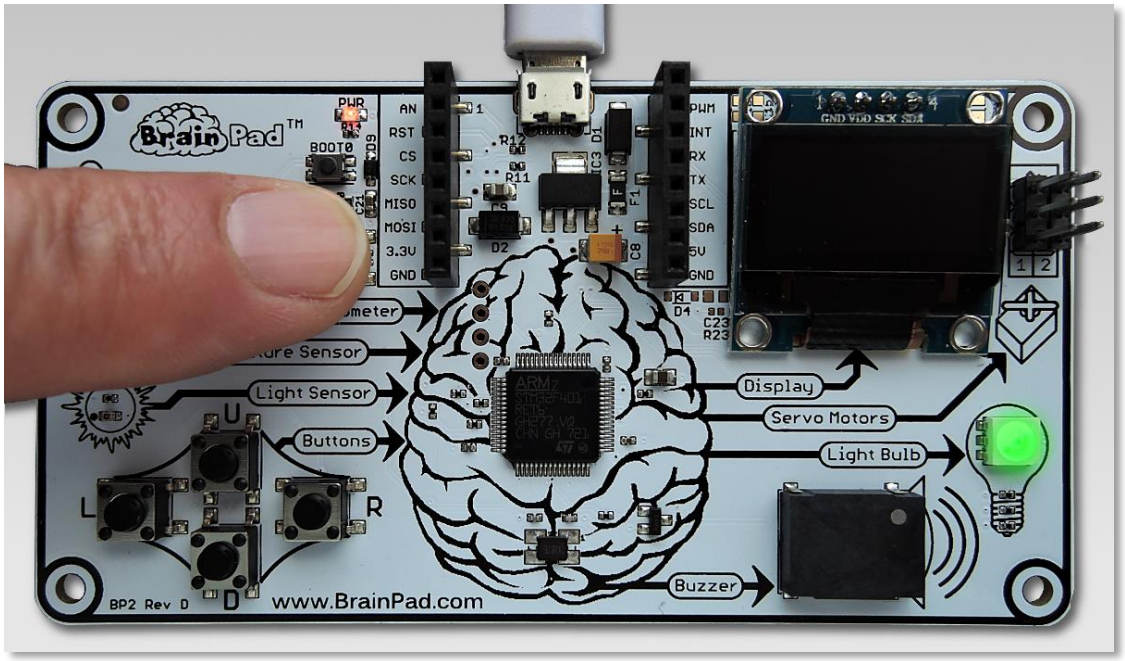

Нажмите на маленькую стрелку «вверх» и выберите "Показать в папке" (Show in folder).

Загруженный файл будет подсвечен, и вы сможете нажать правую кнопку мышки и выбрать "копировать" (Copy)

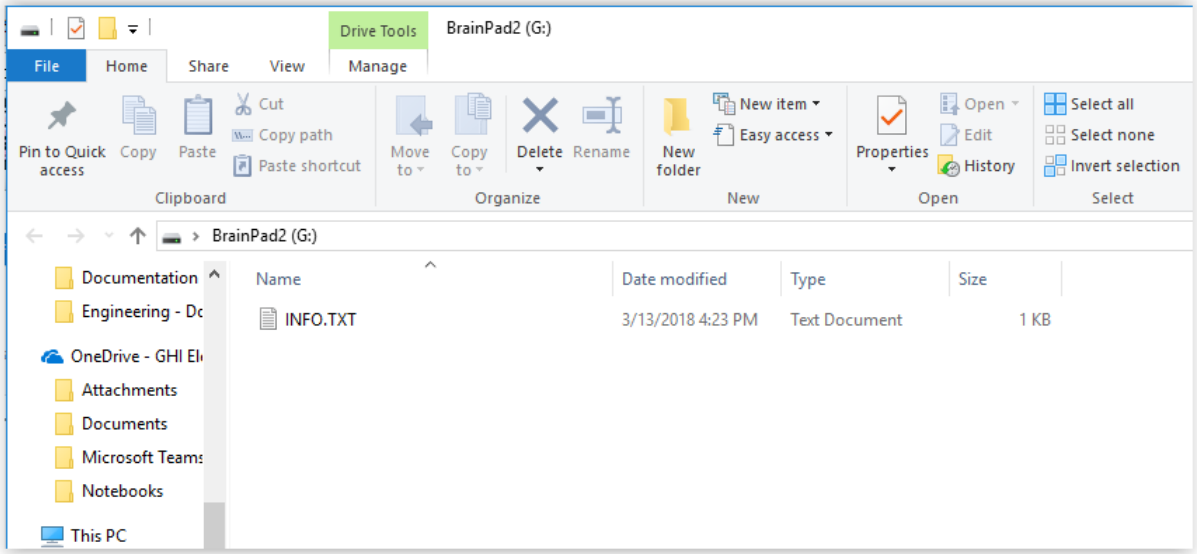



Работая в любом браузере, Microsoft MakeCode не удаляет ваши старые файла. Если вы загрузили одну и ту же программу несколько раз, к названию файла будет добавлен номер (на пример, "brainpad-Untitled (1).uf2") . Убедитесь, что вы скопировали последний файл. Последний файл будет подсвечен. В его названии так же будет самый большой номер в скобках и последняя дата и время.

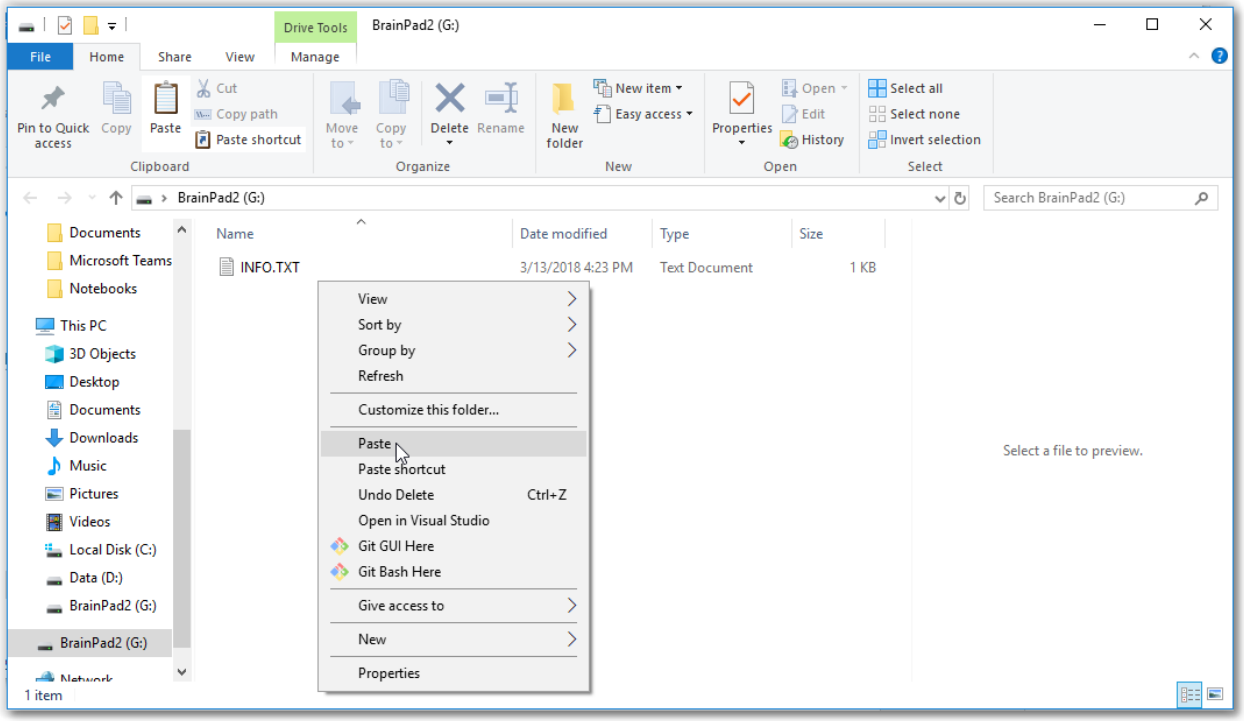
Теперь, когда BrainPad подключен к вашему компьютеру, у него должен гореть красный светодиод. Нажмите и удерживайте кнопку «reset» около трех секунд. Должен загореться зеленый светодиод.



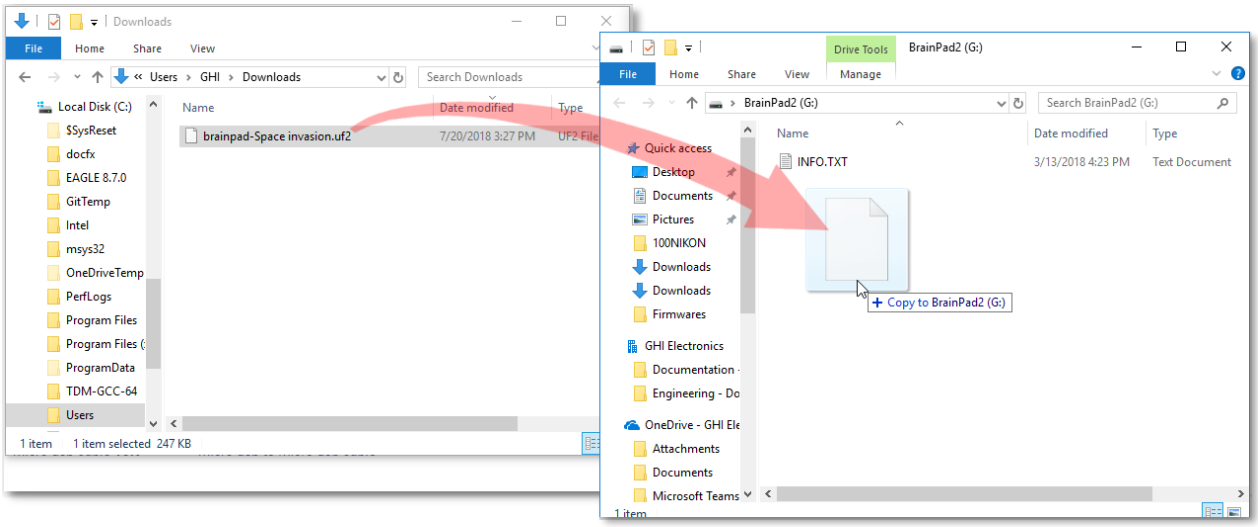

Ваш компьютер определит USB накопитель. Это может занять до минуты, и на вашем экране появится окно с название "BrainPad".

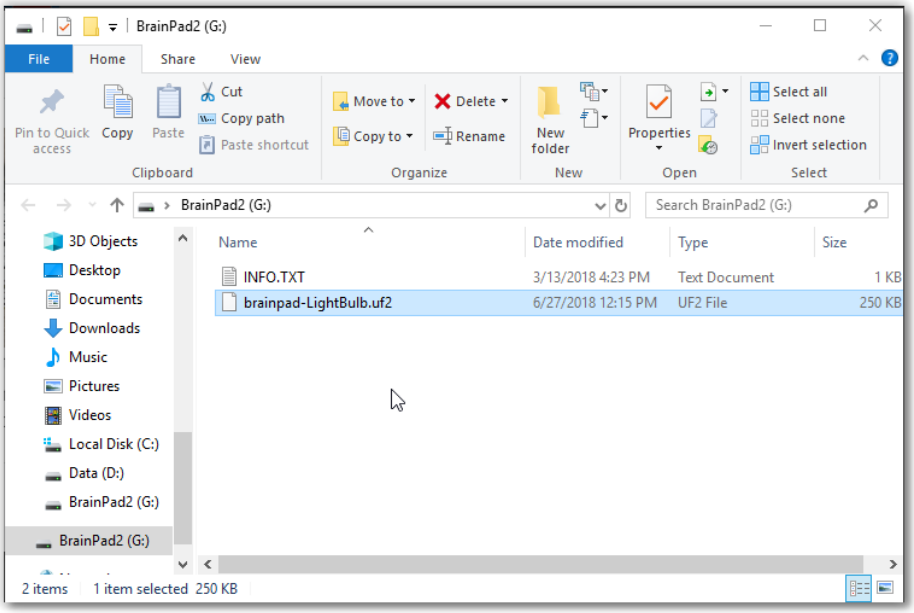


В этом окне отображается то, что находится в памяти BrainPad. Сейчас нажмите правую кнопку мыши и выберите "вставить" (paste) чтобы скопировать файл на BrainPad.




Так же вы можете перетащить загруженный файл в окно BrainPad.

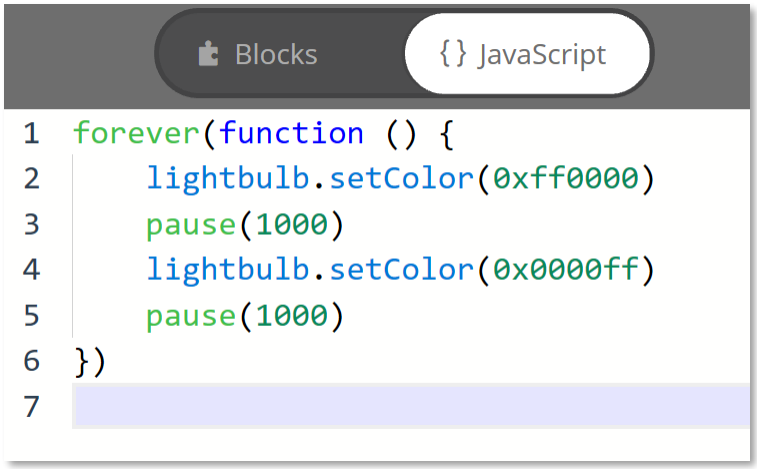





Как только вы это сделаете, светодиод начнет мигать, и после этого BrainPad запустит загруженную программу.

## JAVASCRIPT

Блоки — это увлекательно как для начала, но ваши программы будут становиться все больше, и вам придётся набраться смелости и начать писать код. Не беспокойтесь, вы можете это сделать, когда будете готовы. Одним из замечательных свойств Microsoft MakeCode является перевод блоков на язык программирования JavaScript и обратно с JavaScript в блоки. Вернемся к программе, которую мы написали и нажмем на кнопку JavaScript.



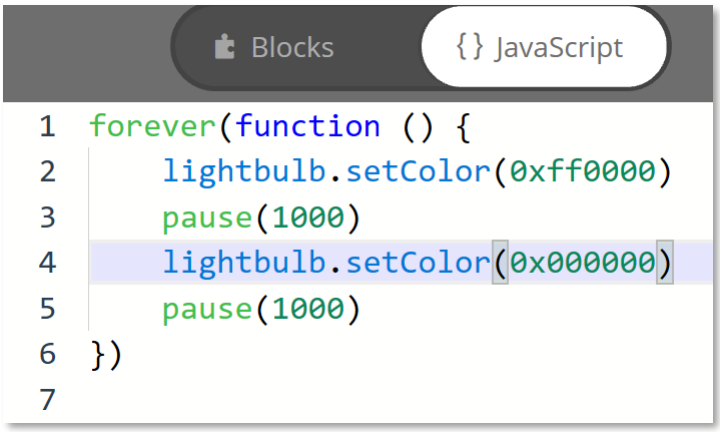

Гляньте на этот код. Вы находите сходство с блоками и кодом? Один момент, который будет не совсем для вас понятен – это цвета, для определения которых JavaScript использует число в HEX формате. Остальной код понятен?

Представление чисел в HEX формате основано на шестнадцатеричной системе чисел вместо привычной нам десятичной. Цифры в шестнадцатеричной системе представлены от 0 до F, в отличии от десятичной, в которой от 0 до 9 (числа которые мы используем в жизни). Если мы попробуем посчитать от 0 до 16 в шестнадцатеричной системе, то это будет выглядеть так: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10.

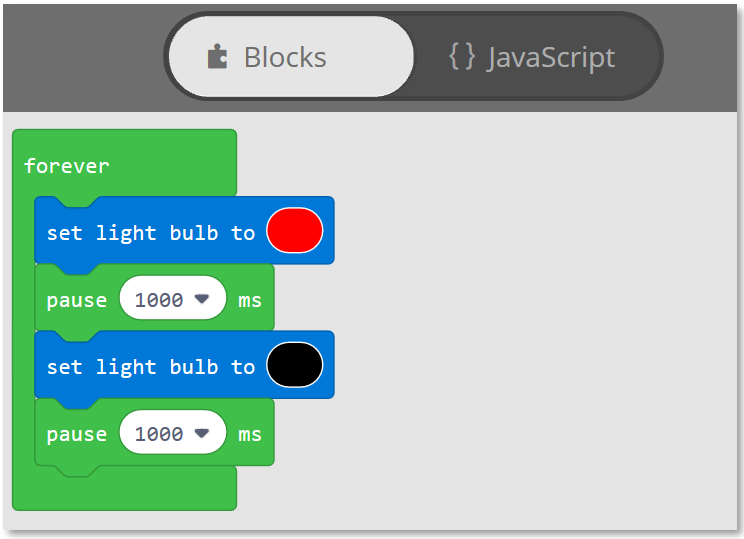
"0x" означает начало шестнадцатеричного числа и дает нам понять, что мы действительно имеем дело с шестнадцатеричным числом. Часто используют сокращение HEX для обозначения шестнадцатеричных чисел. Обязательно нужно упомянуть о бинарной системе чисел, или двоичной как ее еще называют, где используется только 0 и 1. Итак, теперь мы знаем, что числа могут быть представлены в двоичной, десятичной и шестнадцатеричной системе исчисления. Это немного запутывает, но не беспокойтесь. Через некоторое время вы будете легко в этом разбираться.

Когда используют шестнадцатеричное число для обозначения цвета, первые две цифре обозначают наличие красного в цвете, вторые две – наличие зеленого, и следующие три – синего. "00" означает, что интенсивность цвета равна нулю, или цвет отключен. "FF" означает полную интенсивность цвета. Яркий красный обозначается как 0xFF0000. Белый - 0xFFFFFF.

Что произойдет если мы изменим один из блоков "setColor" на 0x000000? Давайте попробуем.



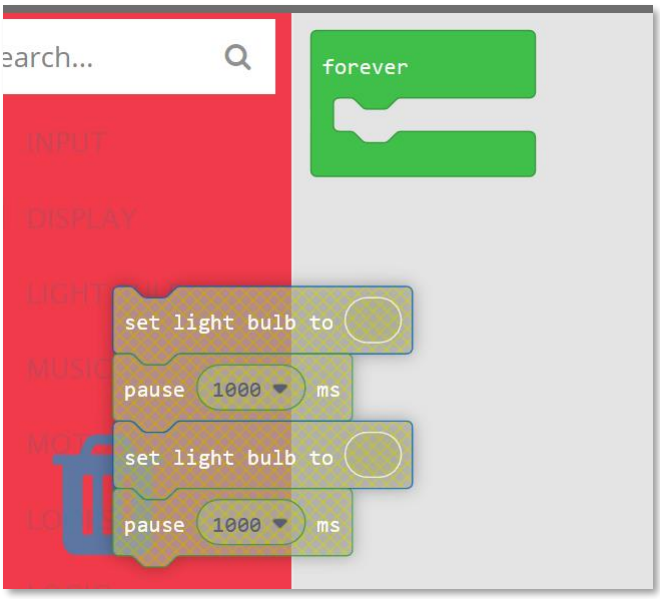

Переключимся обратно на блоки и посмотрим, что изменилось.



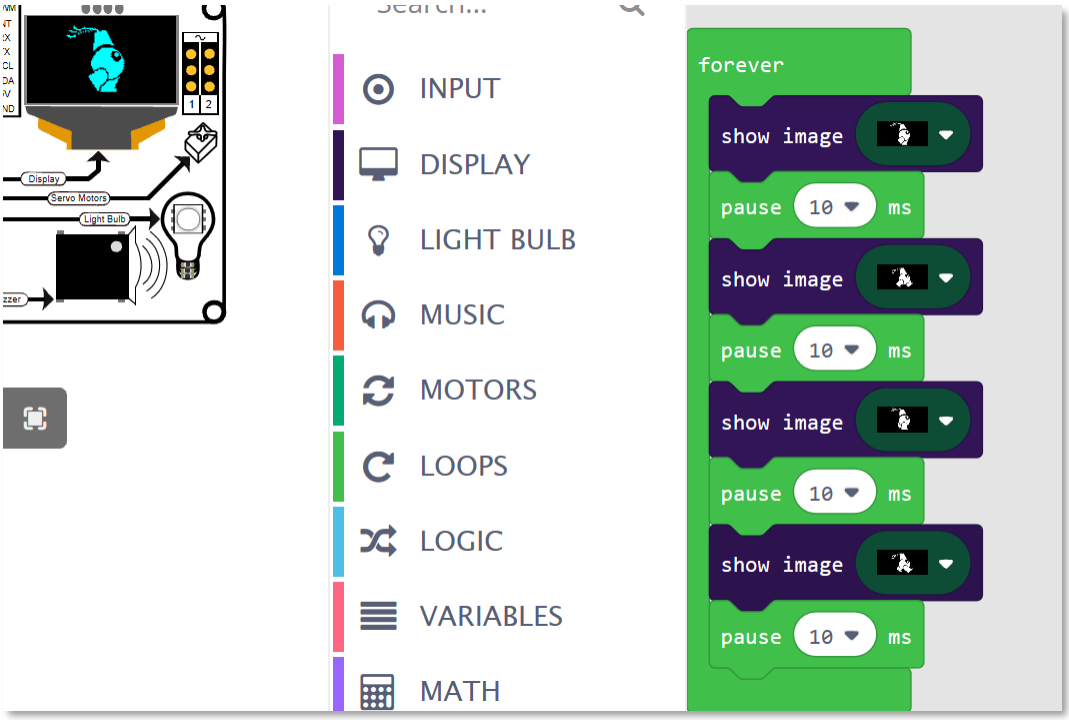

Похоже, что цвет изменился с синего на черный. Вы, наверное, догадываетесь, что, установив на светодиоде черный цвет, мы его тем самым выключили. Вы это можете увидеть на симуляторе. Светодиод должен загореться красным на секунду, а затем выключиться на секунду.

## УВЛЕКАТЕЛЬНЕЙ С ДИСПЛЕЕМ

Если вам кажется, что мигать светодиодом – это увлекательно, то скорее начинайте использовать дисплей! Если вы все еще используете блоки из предыдущего примера, перетяните их обратно в меню, чтобы удалить.




Если вы случайно удалили блок «forever», вы можете его легко вернуть из меню LOOPS. Из меню DISPLAY перетяните блок “show image” и выберите первый из четырех рисунков "идущий человек". Добавьте остальные три изображения и не забудьте про паузу в 10 миллисекунд между ними. В блоке "пауза" нету опции 10 миллисекунд, но вы может сделать это просто изменив число в овале на 10.

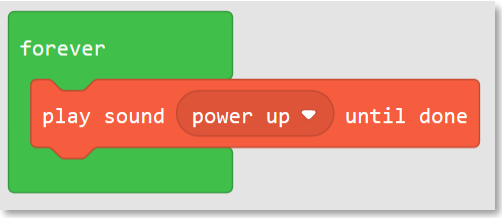



Теперь наш человечек должен ходить в симуляторе на дисплее вашего компьютера. Двигаемся дальше и загрузим это в наш BrainPad.

## ИГРАЕМ ПО НОТАМ

В меню MUSIC вы можете найти отдельные ноты…


. . .или мелодии

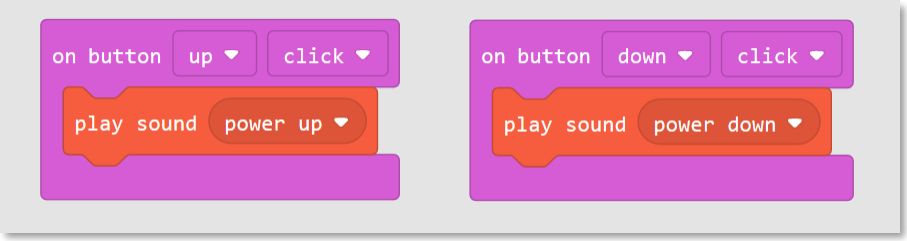



Обратите внимание как мы используем "play sound until done" блок. Этот блок останавливает программу, пока не закончится проигрывание мелодии. Блок "play sound" (без "until done") начинает исполнение следующего блока в то время, как мелодия все еще проигрывается. Если вы попробуете оба блока, то увидите, что результат очень отличается.

## ИСПОЛЬЗУЕМ КНОПКИ

Есть два пути использования кнопки. Первый – просто проверять, нажата ли кнопка и выбрать подходящее действие, если это так. Недостаток этого метода заключается в том, что вам нужно постоянно проверять, не пропустили ли вы нажатие кнопки. Второй путь – это позволить системе делать это за вас и в случае «события» нажатой кнопки автоматически сделать что-то.

Сначала давайте попробуем "читать" состояние кнопки, используя события. Я хотел бы, чтоб играла мелодия «power up», когда нажата кнопка «вверх» и мелодия «power down», когда нажата кнопка «вниз».



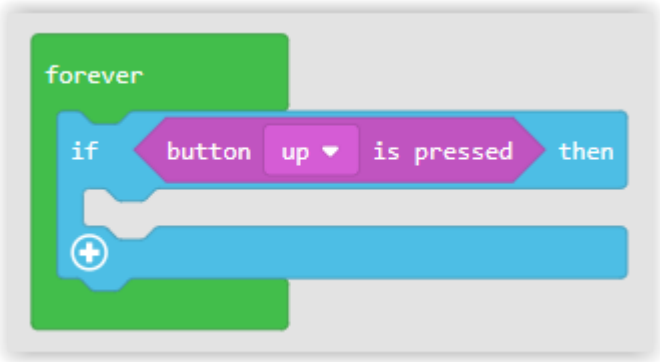

Вы обратили внимание, что мы не используем блок “forever” в этом примере? У вас все еще есть этот блок в вашем коде, но он не нужен в этом примере. Это потому, что система обрабатывает "события" автоматически, когда кнопка нажата.

Эти же блоки в представлении JavaScript кода:

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

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

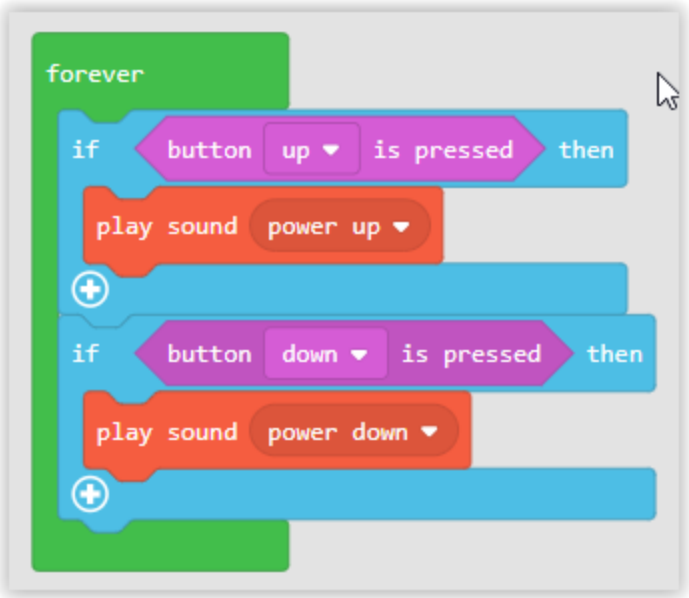
Мы напишем программу, которая будет делать то же самое, но без использования «событий». Чтобы проверить, нажата ли кнопка, мы будем использовать блок «if». Слово «if» в переводе с английского означает "если". То есть "если" кнопка нажата, "то" мы выполняем какое-то действие. Если вы знакомы с английским языком, то должны были заметить, что практически все имена инструкции в блоках и JavaScript-коде соответствуют тем действиям, которые мы хотим выполнить. "if" вы можете найти в меню LOGIC. Перетяните его в наш вечный цикл.


"if" проверяет состояние чего-то и это может быть "да" или "нет", а точнее "true" или "false". По умолчанию стоит "true", но мы хотим заменить это условие на проверку состояния кнопки. Перетяните "button. . . is pressed" из меню input в "if. . . then" как показано на рисунке ниже. Заметьте, что это не событие “on button” которое мы использовали раньше.




В примере выше блоки будут проверять кнопку "всегда". Если кнопка "вверх" будет нажата, то наше условие выполнится, и программа выполнит все команды, которые находятся внутри блока "if".

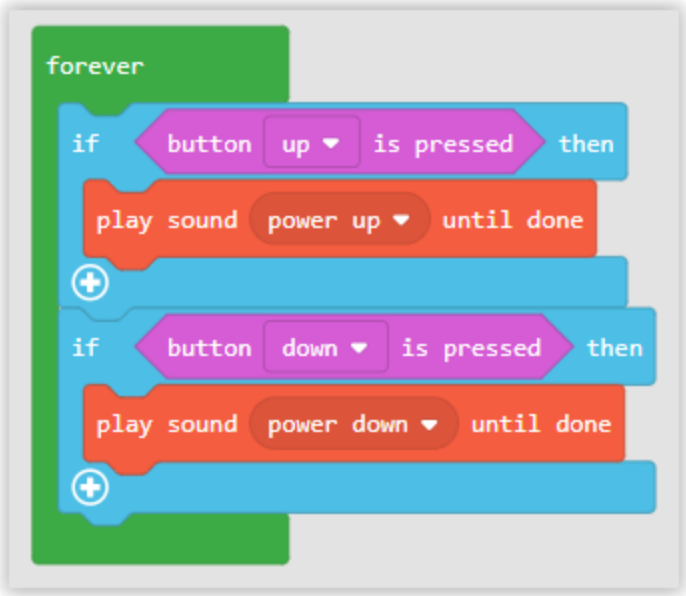
Добавим больше блоков, чтобы сделать нашу программу такой, как показано ниже:




Когда вы запустите этот код, то заметите, что мелодия не проигрывается правильно, пока вы не отпустите кнопку. Причина в том, что каждый раз, когда BrainPad определяет, что кнопка "вверх" или "вниз" нажаты, он начинает проигрывать новую мелодию, даже если предыдущая мелодия не закончилась. BrainPad начнет проигрывать мелодию много раз каждую секунду, пока кнопка нажата.

Есть два пути исправить это. Первый – это заменить "button. . . is pressed" блок на "button. . . was pressed" блок. Условие "button. . . was pressed" установит значение "true", если копка была нажата хотя бы один раз с момента последней проверки кнопки. Даже если кнопка была нажата много раз, все равно будет зарегистрировано только одно нажатие.  Нажатие кнопки "вверх" или "вниз" во время проигрывания мелодии прервет ее, но новая мелодия не будет повторяться сначал,а если кнопка зажата.

Другой способ — это заменить блок "play sound. . ."  на "play sound. . . until done».  Этот блок остановит исполнение остальной части программы, пока мелодия не будет проиграна. То есть программа останавливает проверку нажатия кнопок или проигрывание остальных мелодий, пока текущая мелодия не будет воспроизведена. Этот блок не даст программе возможность прервать мелодию.  Блоки показаны ниже.

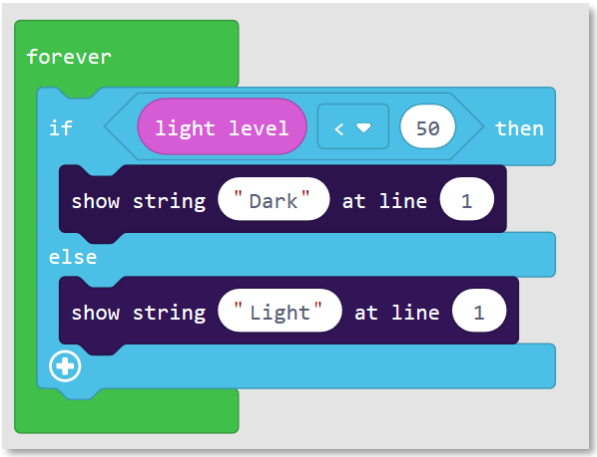



## ЗДЕСЬ ТЕМНО?

Так же, как и кнопки вы можете "читать" уровень освещенности и использовать это как то, или же воспользоваться "событием" и запускать какое-то действие, когда темно или светло. Однако, чтение кнопки и уровня освещенности немного отличаются.

Кнопка может быть нажата или не нажата. Если спрошу: "кнопка нажата"? Вы ответите "да", если это так, или "нет" если это не так. Датчик света дает уровень освещенности как значение. Программисты это называют аналоговым значением. Давайте "прочитаем" уровень освещенности и выведем его на дисплей. Это нужно делать в цикле "forever", то есть, постоянно читать освещенность и обновлять дисплей.


Так как вы теперь эксперты в использовании оператора "if", сможете ли вы вывести на дисплей "темно" или "светло" в зависимости от значения датчика освещенности? Это может быть немного сложно, но я объясню, как это сделать. Оператор "if" требует чего-то, что дает "true" или "false", как например, кнопка. Уровень освещенности - это не «true» или «false» — это может быть одно из многих значений. Если подумать над этим, то нет смысла в том, чтобы сказать: "если свет – то сделай что-то". Нам следует сказать: "если светлее или темнее, чем какое-то значение, тогда сделай что-то". Откройте меню LOGIC и добавьте блок сравнения, то есть блок "if" плюс блок "else". Посмотрите на это:

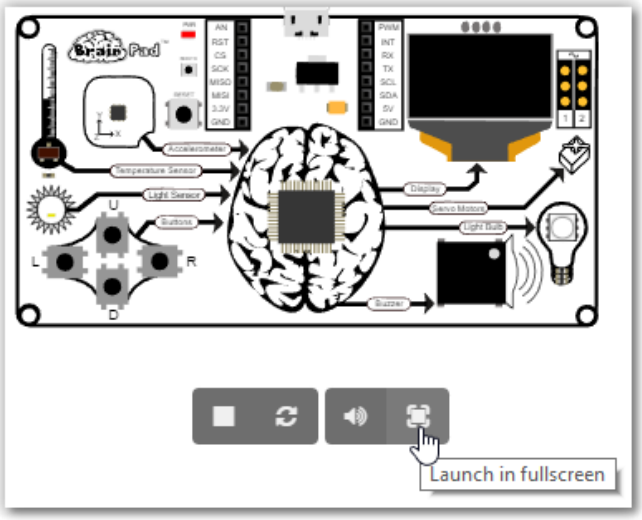



Что это значит? Если уровень освещенности меньше, чем 50, тогда напечатать "Темно", а если нет, то напечатать "Светло" в одной строке.

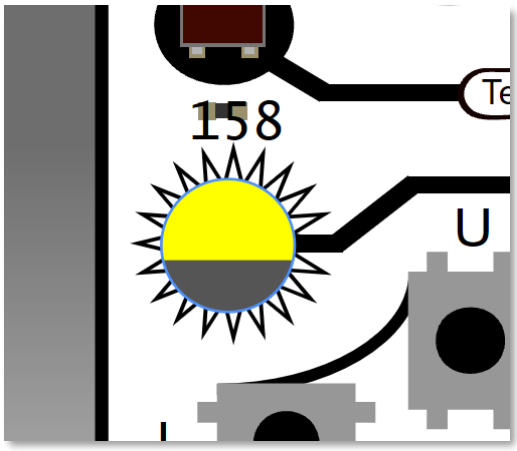
Не бойтесь открывать JavaScript-секцию чаще и посматривать как будет выглядеть код.

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

В симуляторе так же есть уровень освещенности. Как только Light Sensor будет добавлен в проект, симулятор начнет отображать опцию для установки уровня освещенности.



Light Sensor показывает уровень освещенности, который можно менять, перетягивая линию вверх или вниз. Это тоже самое, если бы вы закрыли датчик рукой на BrainPad. Когда вы меняете уровень освещенности вверх или вниз, дисплей так же отображает «Темно» или «Светло».



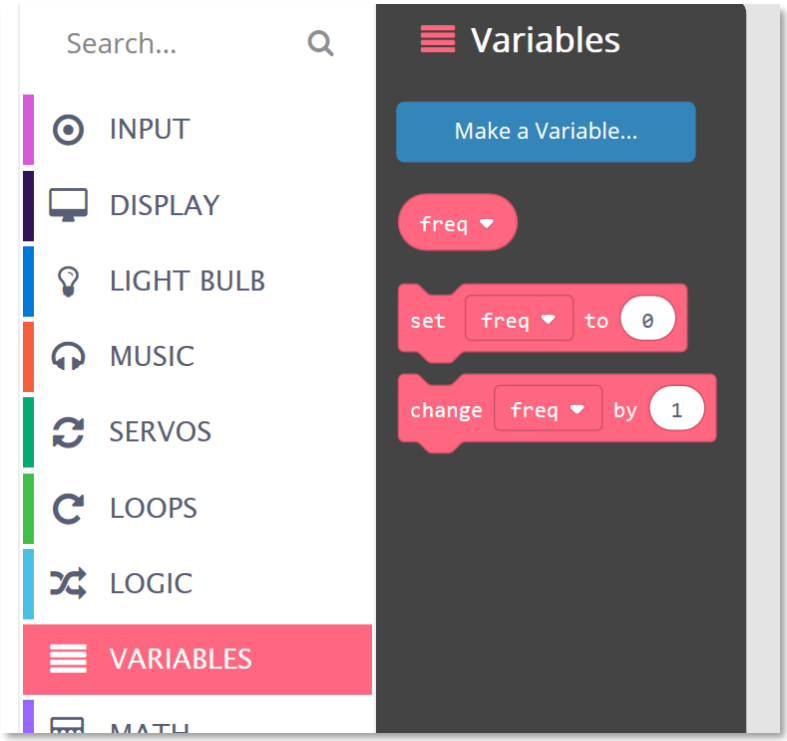

## ПРОВЕРКА СЛУХА

Человеческое ухо может слышать частоты от низких около 20 Гц до высоких около 20,000 Гц. Собаки могут слышать более высокие частоты чем люди - до 44,000 Гц. Вот почему мы не слышим высокочастотные свистки для собак, но собаки их слышат хорошо.

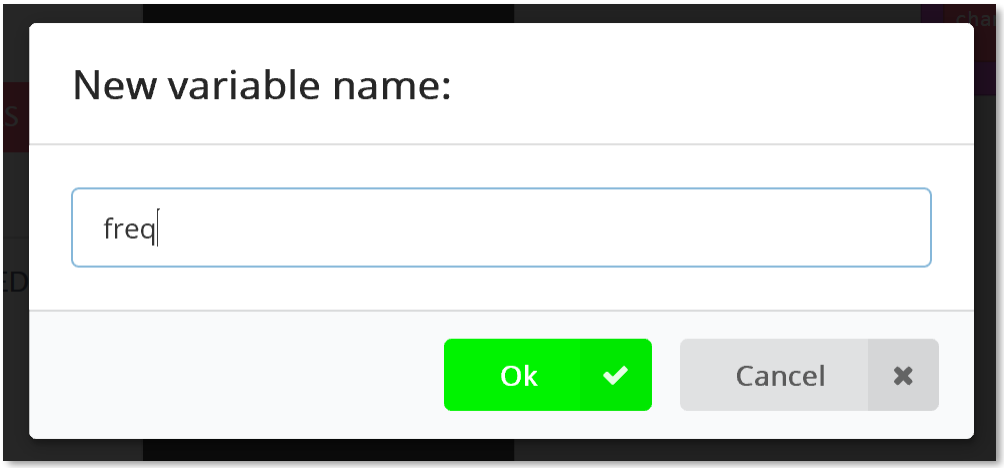
С возрастом мы теряем способность слышать очень высокие частоты. Так же потери слуха больше способствует высокочастотный шум нежили низкочастотный. Хотите проверить, насколько высокие частоты вы способны слышать? Давайте разберемся!

Нам нужно написать программу, которая будет использовать "переменные". Представим переменные как ящики, у которых есть свои имена. В такие ящики мы будем складывать какие-нибудь значения. В нашем случае, в "ящиках" будут храниться частоты, которые мы хотим проигрывать на нашем динамике.

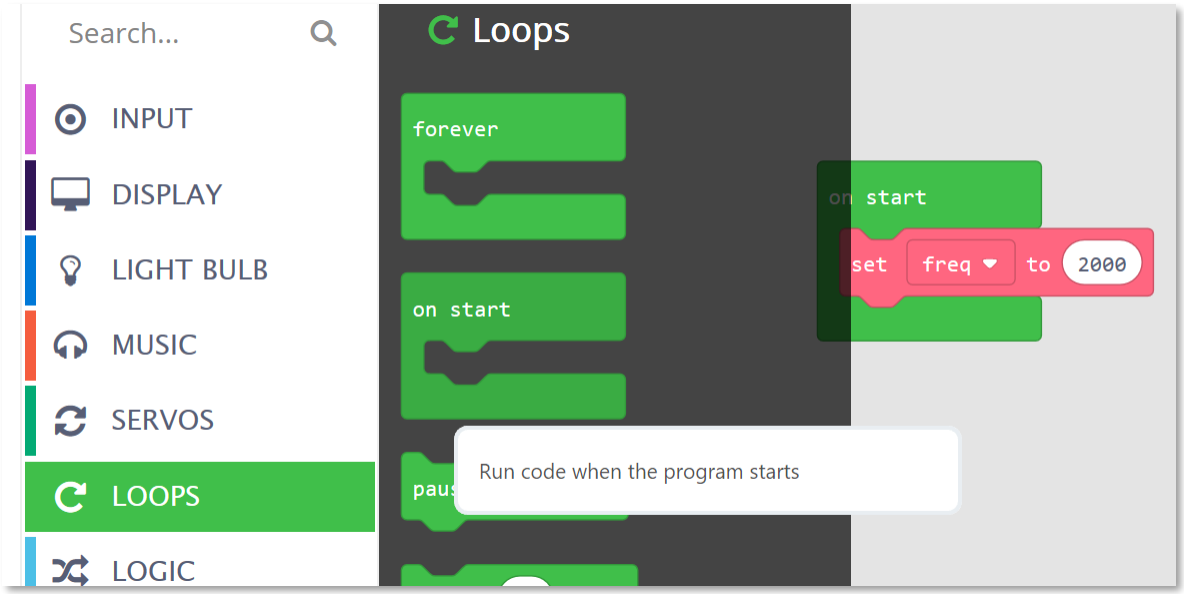
Перейдем в меню VARIABLES и выберем блок “Make a Variable. . .”



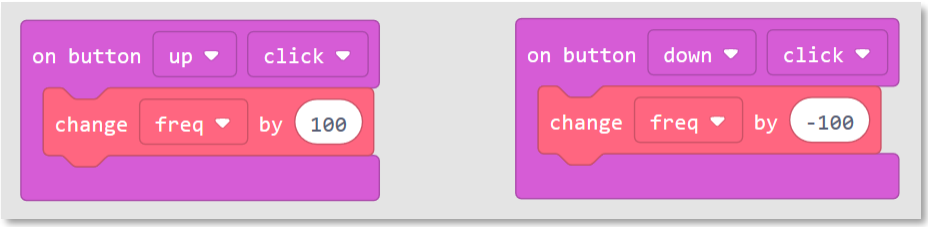

Дадим нашей переменной имя. Я буду использовать имя "freq". Эта переменная будет хранить частоту, которую я буду проигрывать на динамике.



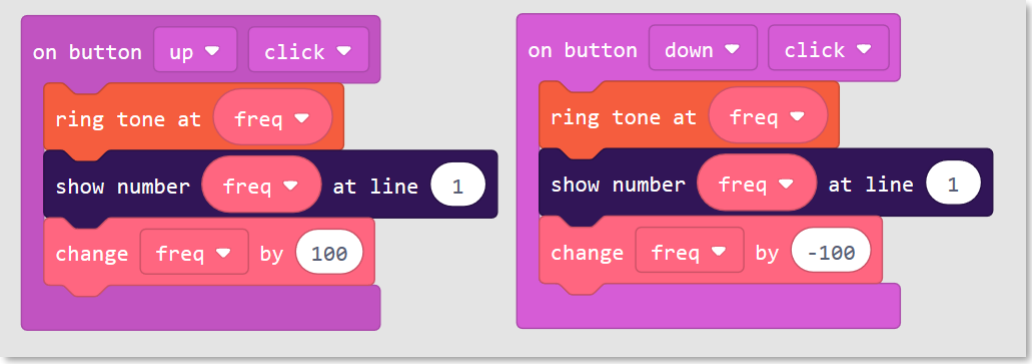

Теперь добавим цикл "on start", который вы найдете в меню LOOPS. Что бы вы не добавили в этот блок, это произойдет, когда программа будет запущена. Установим сюда переменную со значением 2000. Большинство людей может слышать частоту 2000 Гц, так что это хорошее начало. Блок "set" находится в меню VARIABLES.



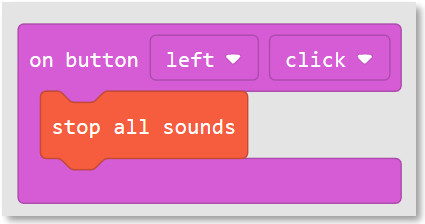

Сейчас нам нужно воспроизвести звук с этой частотой, но также нам нужно повысить частоту, когда кнопка «вверх» нажата и понизить, когда кнопка «вниз» нажата. Нам нужно изменить блок с переменной. Увеличить на 100, то есть прибавить, когда нажимаем «вверх» и уменьшить на 100, то есть, отнять или изменить на -100, когда нажимаем кнопку «вниз».



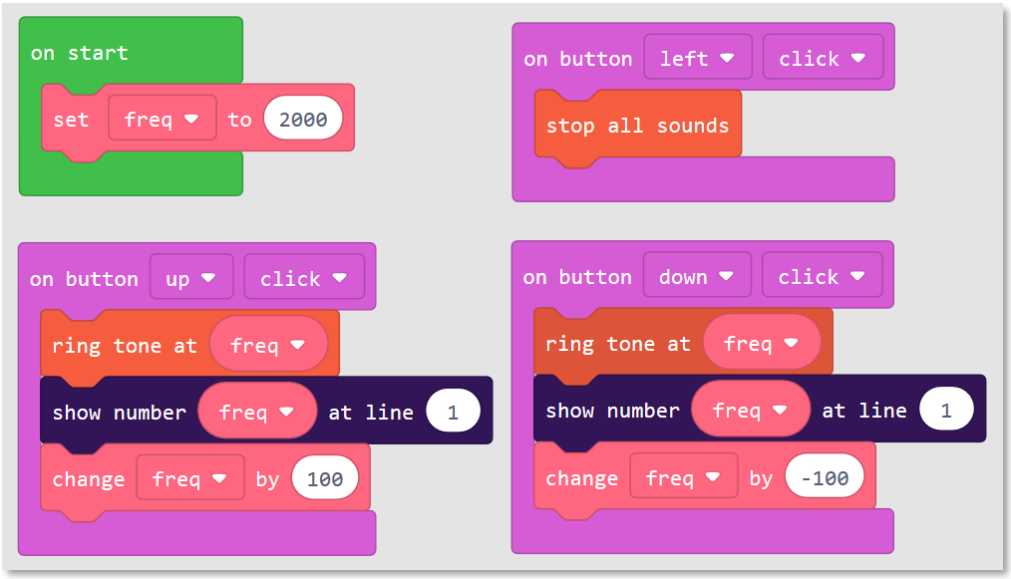

У нас есть переменная, которую мы можем менять как нам нужно, и мы можем генерировать звук (ring tone) и выводить частоту на дисплей (show number).




И последнее! Шум может сильно раздражать вас, так что давайте остановим звук нажатием кнопки "влево".




Это вся программа.




## СЕРВОМОТОРЫ

Сервомотор — это мотор, у которого есть внутренняя электроцепь, позволяющая устанавливать нужное положение ротора электрическими пульсами.

У сервомотора есть три контакта, которые подключаются прямо к BrainPad. К сожалению, у разных моторов часто провода разных цветов. Чтобы определить, как правильно подключить мотор, не обращайте внимание на центральный провод, для нас важны провода, которые на краях. Светлый провод должен быть подключен к контакту, над которым напечатан символ ~ на BrainPad. Это сигнальный провод, кстати.

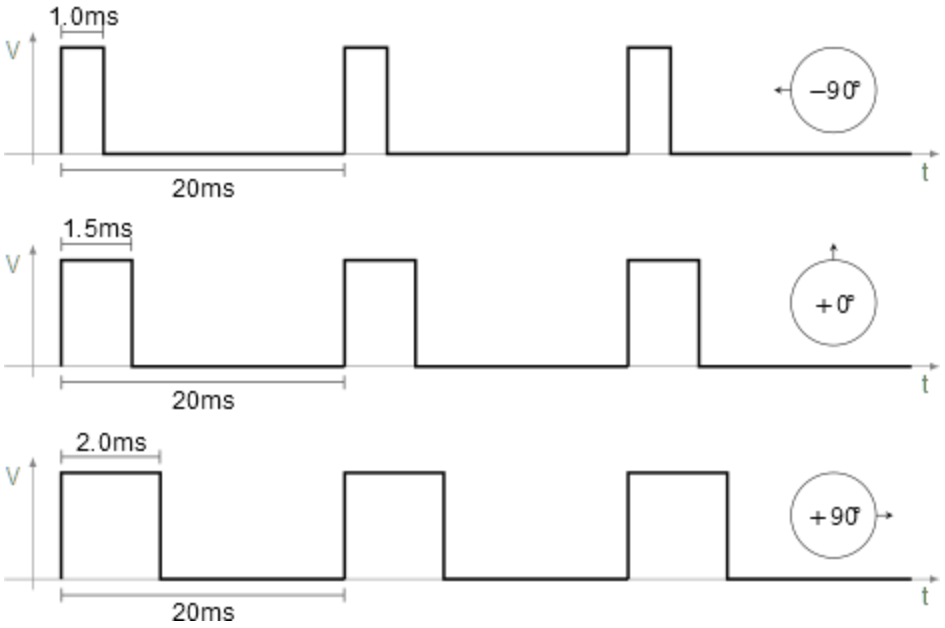

Сервомоторы часто используют в радиоуправляемых моделях и их можно найти в интернет-магазинах или в радиолюбительских магазинах неподалеку от вас. Сервомотор питается электротоком от BrainPad, который берет электроэнергию от вашего ПК или аккумуляторной батареи (АКБ). Следовательно, мы рекомендуем использовать маленькие сервомоторы или микросерво. Некоторые сервомоторы потребляю слишком много электроэнергии для корректной работы без дополнительного источника питания.

Сервомоторы также бывают непрерывными или позиционными. Выглядят они, как правило, одинаково, но отличаются функционально. В позиционных сервомоторах можно устанавливать ротор в определенную позицию и удерживать его там, пока вы не измените эту позицию. Непрерывные же вращаются в одном направлении, пока вы не пошлете сигнал остановиться или вращаться в другом направлении.

Если вы повернете ротор не подключенного мотора рукой, то заметите, что он поворачивается только на пол оборота в каждом направлении и останавливается. Тогда как, непрерывный мотор можно вращать в любом направлении без ограничений. Так как внутри каждого сервомотора есть механизм, вам придётся приложить некоторые усилия, чтоб повернуть его.

## ПОЗИЦИОННЫЕ СЕРВОМОТОРЫ

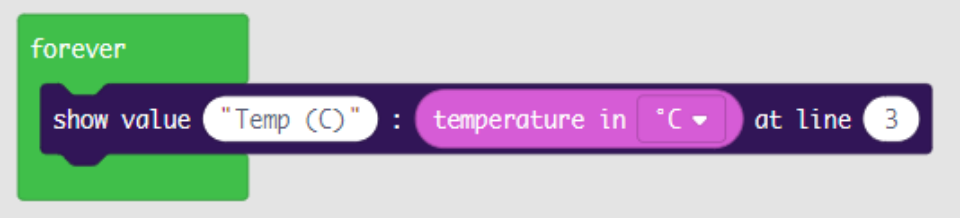
Как я упомянул выше, позиционные моторы можно поворачивать в определенную позицию (или угол), после чего он в этой позиции и останется, пока вы не измените этот угол. Если вы попробуете повернуть сервомотор рукой и в определенной позиции он остановится, значит это позиционный сервомотор. Пульсирующий сигнал, который посылает BrainPad, говорит мотору, на какой угол или позицию ему повернуться.




Есть много способов использовать позиционные сервомоторы. К примеру, мы сконструировали большой термометр с циферблатом. Код, который мы для этого использовали очень простой и для него даже не нужны переменные!

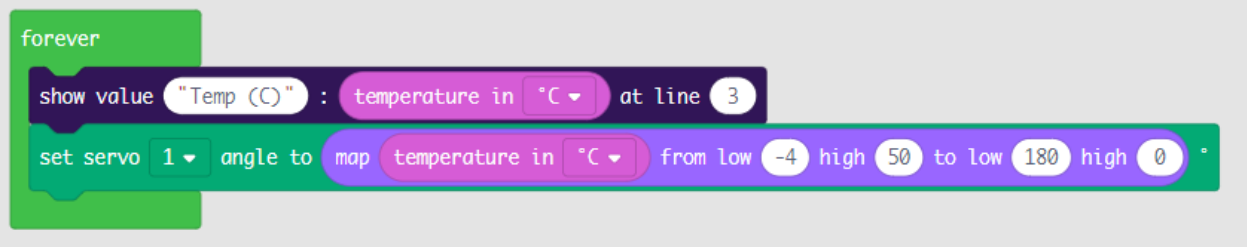
В вечном цикле добавьте блок "show value" (который вы найдете в меню в секции дисплей). В первом овале наберите " Temp C)" в скобках как показано ниже.


Из категории «input» выберете блок «temperature in °C» и перетяните его во второй овал. И в третьем овале измените значение и поставьте 3. Теперь значение температуры будет отображаться на дисплее BrainPad в третьей строчке. Ваши блоки должны выглядеть так:



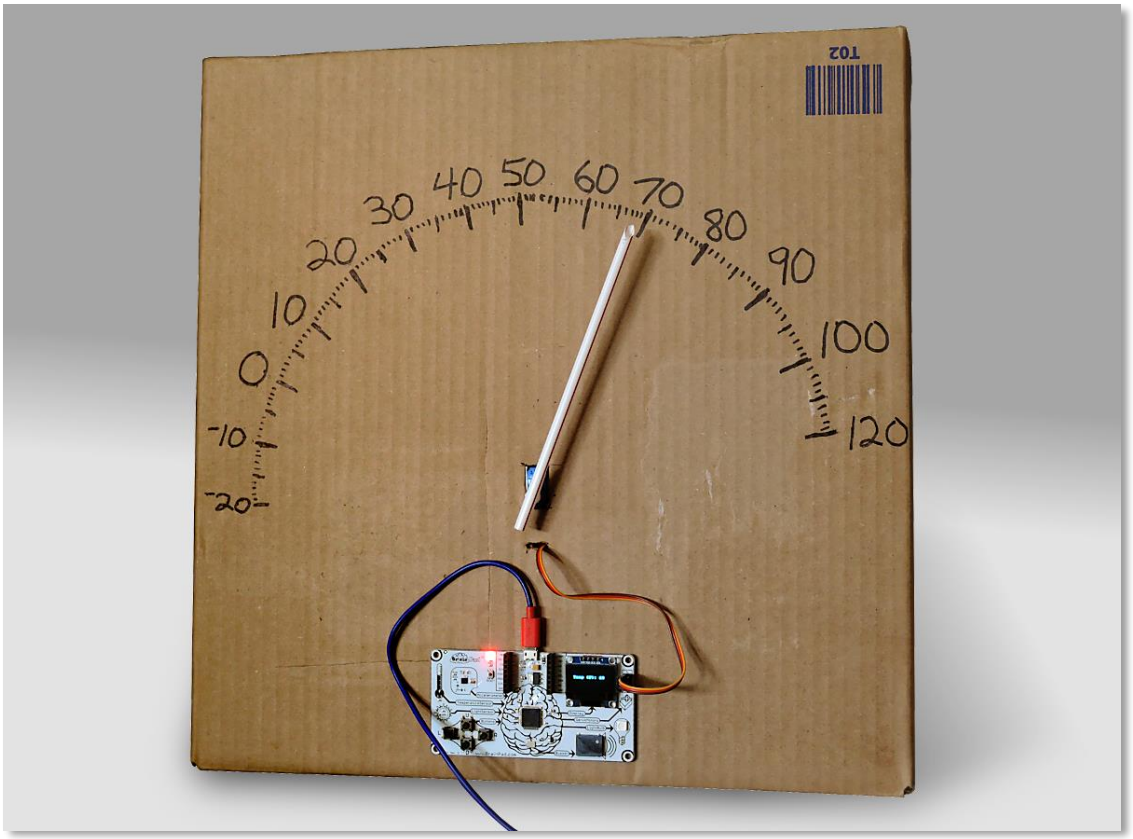

Довольно просто мы сделали термометр, который показывает температуру на дисплее BrainPad. Теперь нам нужно преобразовать температуру в угол и послать это значении сервомотору, чтобы повернуть его на циферблате нашего термометра. Сделаем термометр так, чтобы он измерял температуру в пределах от -4° до 50° по Цельсию.

Нам нужно преобразовать температуру (от -4 до 50) в угол (от 0 до 180). Если вы не очень дружите с математикой, то не беспокойтесь! В MakeCode есть блок "map", который решит все математические задачи за вас. Найти его можно в меню "MATH". Мы можем закончить нашу программу, добавив всего одну строку:




Вы, наверное, обратите внимание на то, что низкой температуре соответствует 180 градусов, а высокой 0 градусов в нашем "map" блоке. Все наоборот? Причина в том, что угол в 0 градусов соответствует повороту сервомотора вправо, а 180 градусов - влево. Если бы мы это не учли, то наш термометр мерял температура на оборот.

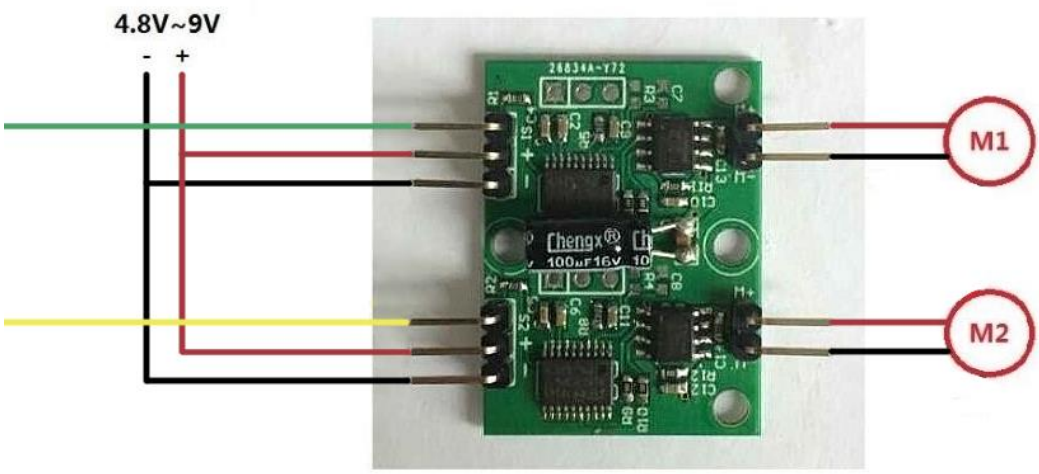
Вот законченный термометр, сделанный из картонной коробки и соломки, приклеенной к сервомотору (в США температуру измеряют в градусах по Фаренгейту. Шкала температуры на термометре, что на картинке, отличается от того, что должно получится у вас. Вам нужно будет отобразить градусы по Цельсию, где вместо -20 будет -4, а вместо 120 будет 50).



## НЕПРЕРЫВНЫЕ СЕРВОМОТОРЫ

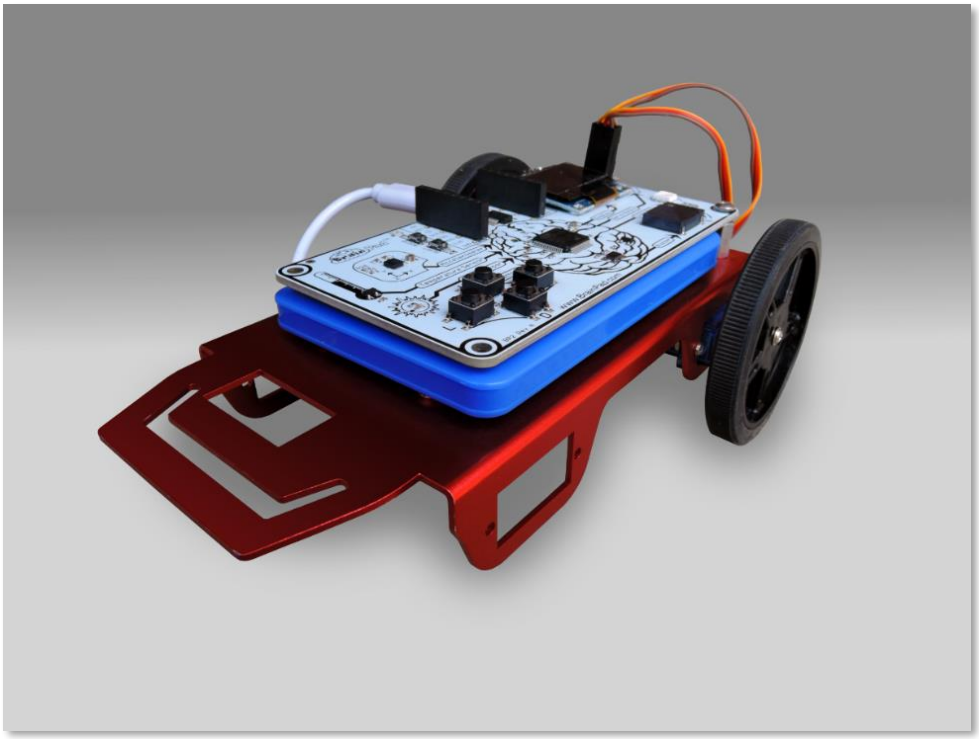
Другой тип сервомоторов - непрерывные, которые, как понятно из названия, могут вращаться непрерывно. В данном случае, пульсы устанавливают скорость и направление движения мотора. К BrainPad можно подключить два мотора. Управляя направлением и скоростью, вы можете легко сконструировать робота. В интернет-магазине можно быстро и не дорого купить то, что нам нужно.

Этот набор можно легко установить на картонную коробку, но потом я нашел вот это шасси от компании FEETECH.



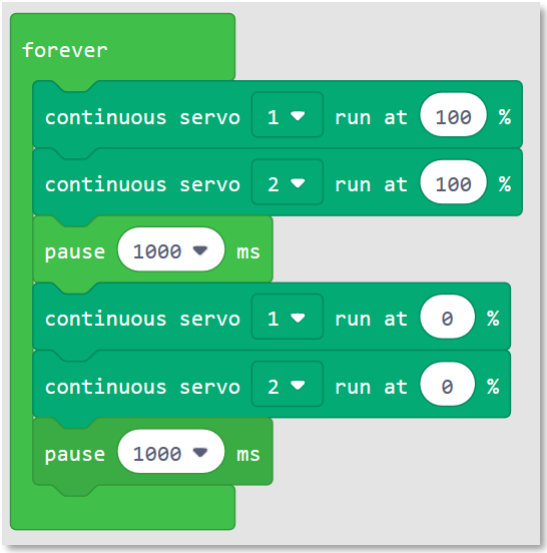

Тут есть небольшие сложности! В комплекте этого робота есть колеса и моторы. Моторы выглядят как серво, но подключаются они как обычные моторы. Я это понял из того, что у них только по два провода, а не по три, как у наших сервомоторов.  В наборе я так же нашел печатную плату, которая, как я потом догадался, преобразует простые моторы в серво!

Можно работать и с этим, но для занятий в классе это довольно сложно. Поэтому я отложил этот преобразователь и моторы и решил установить на нашего робота непрерывные сервомоторы, о которых мы говорили ранее. Мой BrainPad отлично подходит к этому шасси, как будто они были созданы друг для друга.

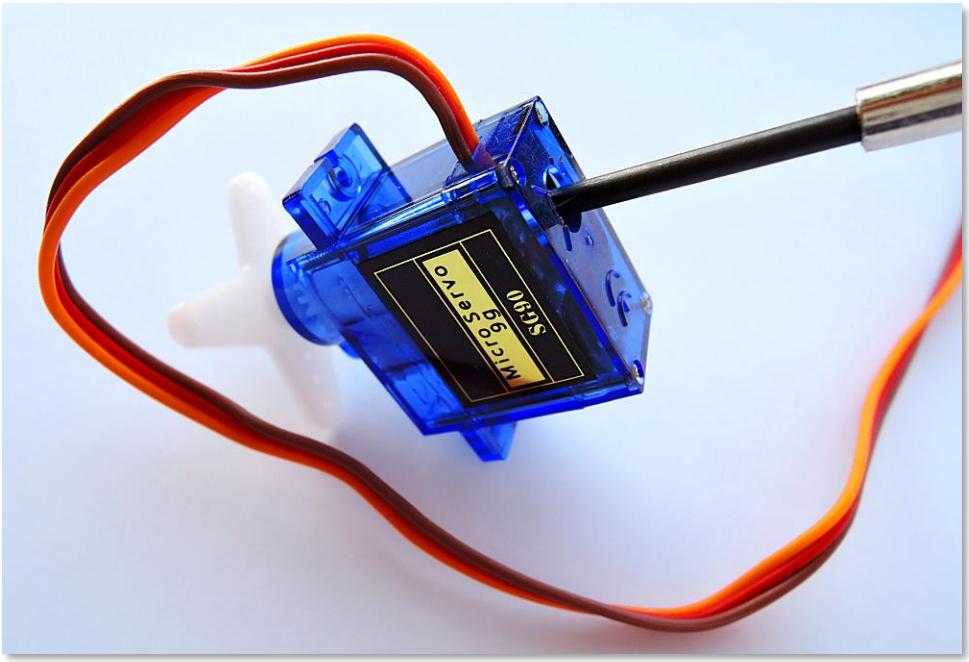


Так же для этого проекта я подобрал внешнюю АКБ для мобильного телефона (power bank), которая тоже хорошо подошла по размеру к BrainPad.

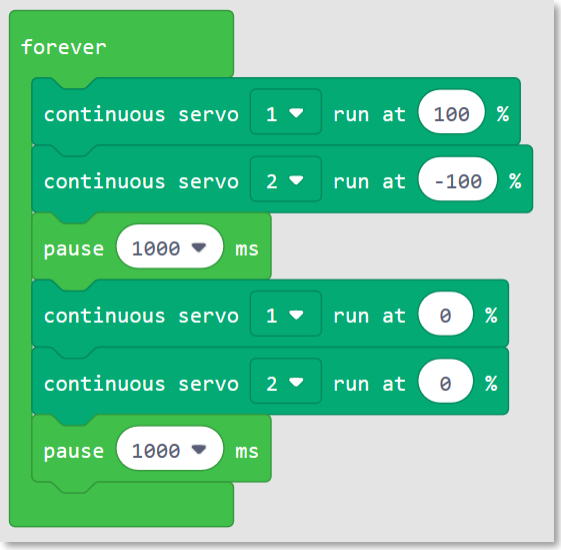
Этот робот наконец готов к запуску! Чтобы он двигался вперед, оба мотора должны вращаться в перед. Поскольку, один из моторов развернут "лицом" справа, а второй слева, то один из них должен двигаться в противоположную сторону. На словах это звучит запутанно, но как только вы попробуете все собрать и запустить, вам все станет понятно. Этот код двигает оба мотора вперед на полной скорости одну секунду, останавливает их на одну секунду и потом опять повторяет этот процесс.




В случае, если моторы продолжают вращаться при выставленной скорости 0, то они нуждаются в калибровке. Большинство сервомоторов можно настроить. Вам понадобится маленькая отвертка для этого.




Вернемся к нашему роботу. Вы обратили внимание на то, что один из моторов двигается в неправильном направлении? Измените направление его движения на обратное.



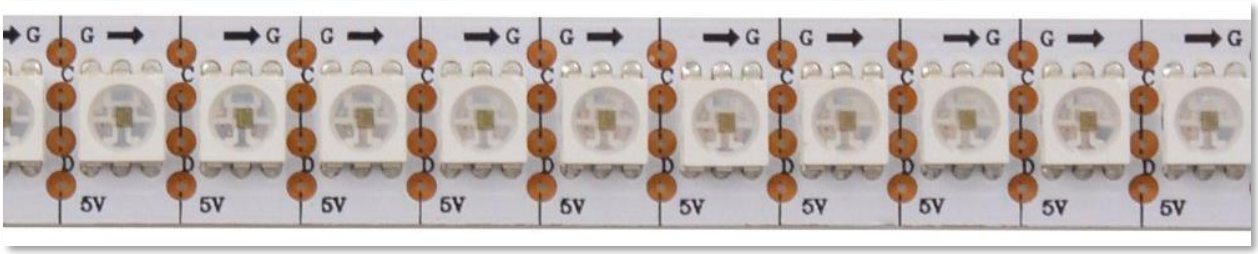

Этот код часть нашего проекта “Sun Seeker” ("искатель Солнца"). Вы можете найти его тут https://docs.brainpad.com/projects/sun-seeker.html. Обязательно посмотрите видео. Робот использует датчик света, чтобы определить, находится он под солнцем или в тени. Если в тени, то серводвигатели будут его разворачивать, пока он не найдет солнце.

## РОЖДЕСТВЕНСКИЕ ОГНИ

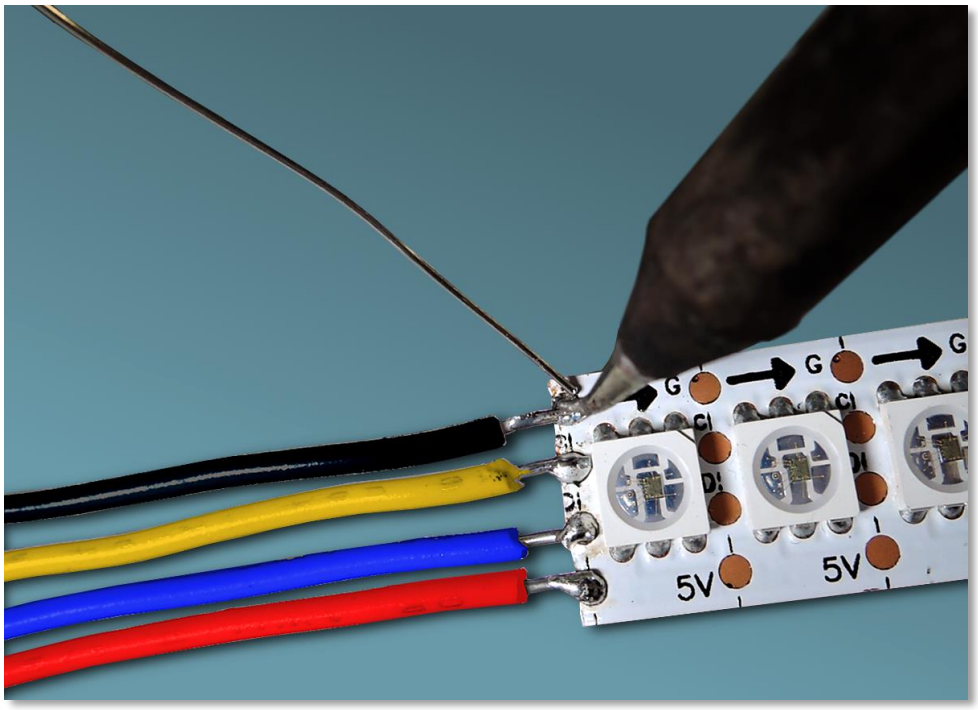
Вы можете сделать много проектов, используя встроенные в BrainPad опции, но скоро вы захотите подключить к нему внешние устройства. Так как это выходит за рамки нашей книги, мы для вас подготовили короткую инструкцию, чтобы показать, какие проекты можно реализовывать.

В этом проекте мы будем использовать ленту светодиодов, к каждому из которых можно обращаться по определенному адресу. То есть любым светодиодом можно управлять отдельно. Каждый светодиод можно зажечь определенным цветом и яркостью, используя только два провода. По этим проводам мы посылаем данные, зашифрованные пульсирующими сигналами светодиодам.

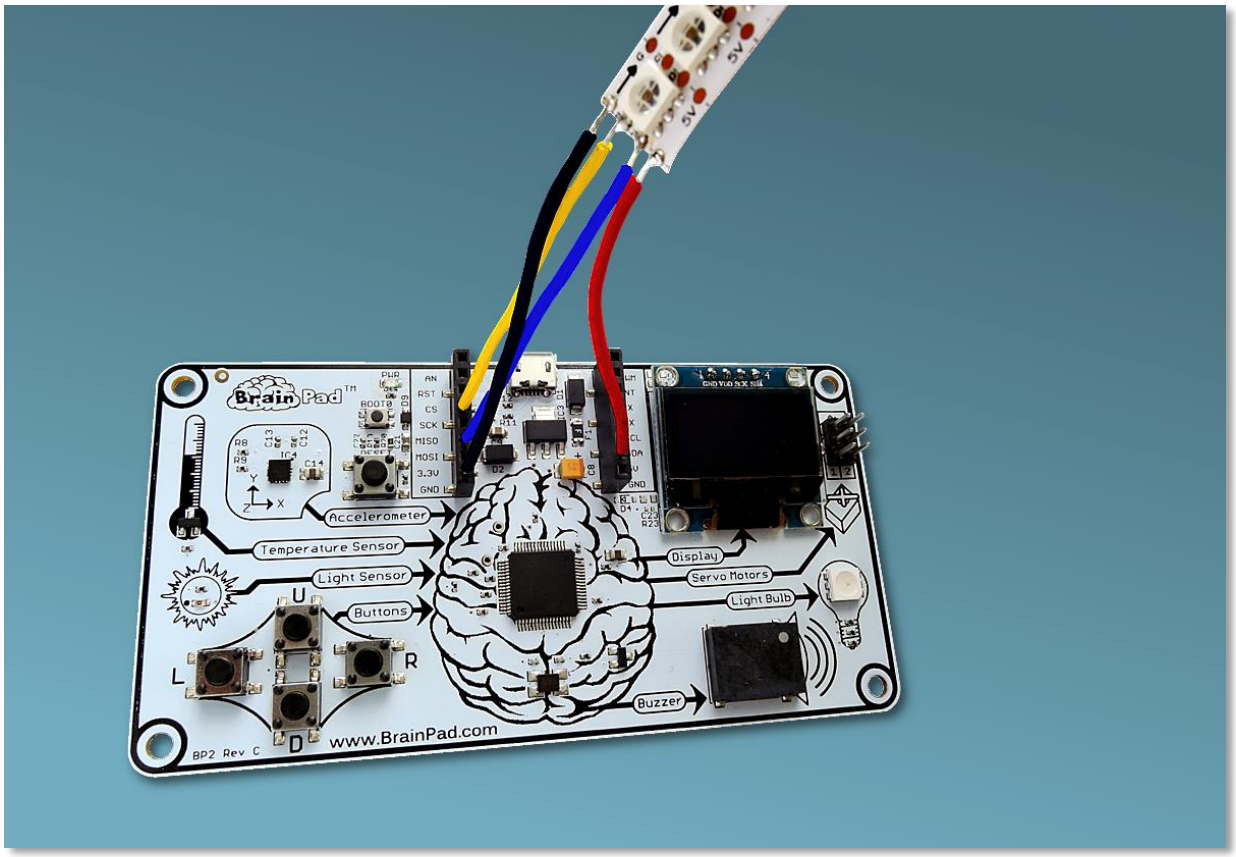
Вам необходимо найти светодиодную ленту с контроллером APA102. Этот контроллер использует SPI протокол, который доступен на BrainPad. Эти ленты подключаются к источникам питания с разным вольтажом. Нам нужны те, что работают от 5В, так как у нас есть доступ к 5В.



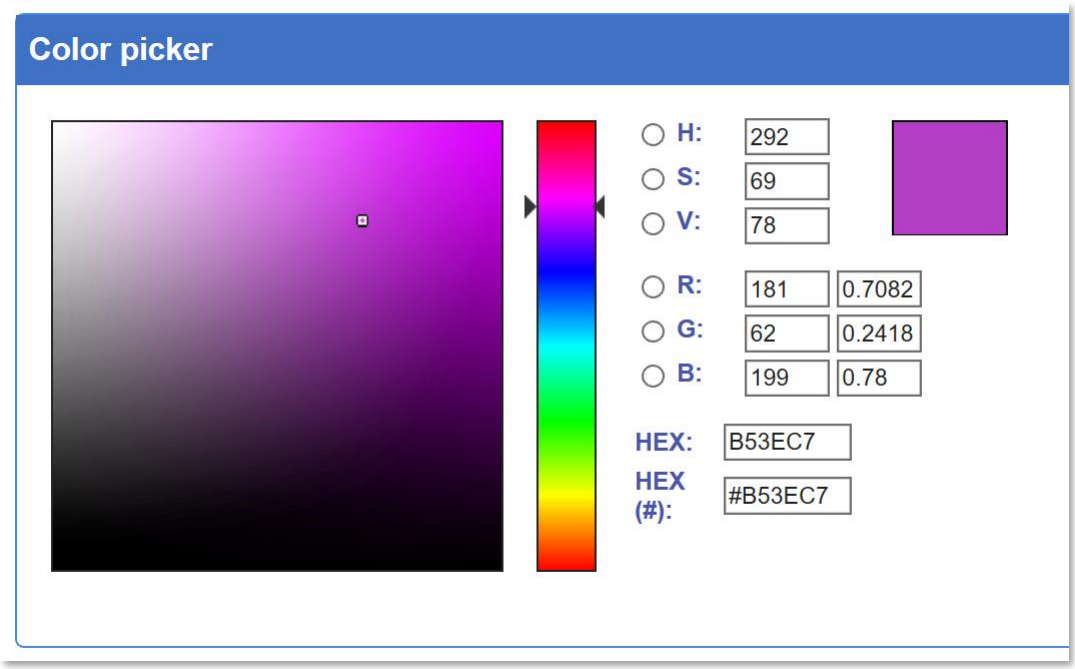

Эту ленту можно разрезать в любом месте. Чтобы она не потребляла слишком много энергии, я оставлю 25 светодиодов. Теперь нужно припаять несколько проводов. Стрелка на ленте указывает, к какой стороне нужно припаивать провода. И так четыре провода: G-Ground (земля), C-Clock (тактирующий сигнал), D-Data (данные) и 5V (5В)




GND и 5V понятно обозначены на дополнительных коннекторах BrainPad'a. C-Clock должен быть подключен к SCK на BrainPad. D-Data нужно подключить к MOSI BrainPad.



В каждом из светодиодов есть три маленьких светодиода. Красного, синего и зеленого. Управляя интенсивностью свечения каждого из них можно создать цвет, который вы хотите. Есть несколько онлайн инструментов, которые помогут вам выбрать цвет. Хороший пример вы можете найти тут http://www.rgbtool.com. Вам нужно установить значение R (красный), G (зеленый), и B (синий)




Команды, что мы отправляем светодиодам, используют шину SPI, которая доступна в BrainPad. Чтобы начать последовательно отправлять команды светодиодам, нужно отправить блок стартового кода. В данном случаем это просто четыре нуля.

После стартового блока каждый светодиод требует четыре числа, чтобы им управлять. Первое число устанавливает общую яркостью всех светодиодов на ленте. Мне не хотелось бы грузнуть в деталях, так что просто установим первое число равным 255, которые выставляет полную яркость.

После первого числа отправляем три остальных в диапазоне от 0 до 255, так мы управляем яркостью синего, зеленого и красного в каждом светодиоде. Чтобы зажечь светодиод синим цветом, вам нужно отправить 255,0,0. Чтобы отключить его, вам нужно отправить 0,0,0. Чтобы светодиод горел ярким белым светом, нужно отправить 255,255,255.

Светодиоды в ленте просто подключены последовательно друг за другом. Первый цвет отправляется первому светодиоду, второй второму и т.д. Светодиоды будут непрерывно светиться, пока не получат новый стартовый блок (четыре нуля), после чего последовательно повторяется процесс.

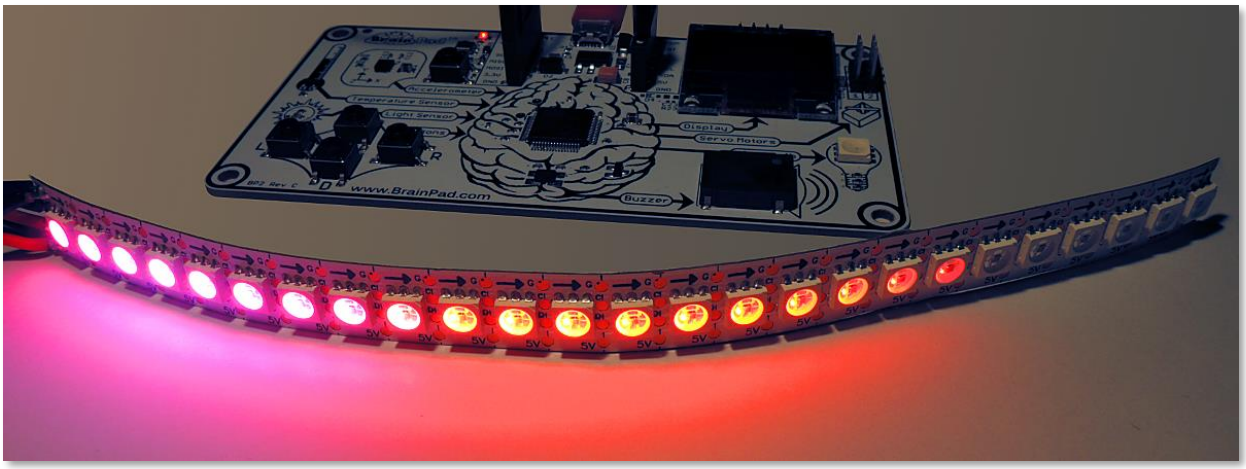
В этом примере довольно много блоков, поэтому более легким путем будет переключиться на JavaScript, скопировать и вставить этот код JavaScript редактор в MakeCode:

```
let center = 12
let startRed = 0
```

```
let startGreen = 0
let startBlue = 0
let red = 0
let green = 0
let blue = 0
let data: Buffer = pins.createBuffer(4)
let dummy: Buffer = pins.createBuffer(4)

pins.spiFrequency(1000000)

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

  pins.spiTransfer(data, dummy)
}

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

  pins.spiTransfer(data, dummy)
}

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

```
    SendStart()


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

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

Если вы сделали все правильно, то должны увидеть разноцветные огни. Попробуйте сделать некоторые изменения в коде и посмотреть, как это повлияет на результат.




## ВЫВОДЫ

Microsoft MakeCode и BrainPad это отличные инструменты для изучения программирования, которые превосходно дополняют друг друга. Благодаря симулятору вы можете тестировать код даже если у вас еще нет BrainPad. Далее вы можете загрузить программу на BrainPad, просто скопировав загруженный файл в окно BrainPad.

В следующих главах мы выйдем за рамки и будем программировать на C# или Visual Basic в Microsoft Visual Studio. "Выходим за грани" — это дополнительный, но очень рекомендованный шаг, к которому рекомендуем перейти, когда вы начнете уверенно чувствовать себя в MakeCode. В большинстве случаев MakeCode – это все, что вам нужно.

# GO BEYOND - ВЫХОДИМ ЗА ГРАНИ

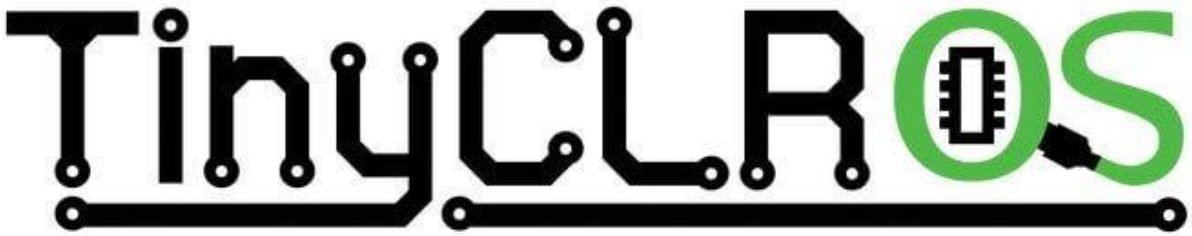
Вы здесь потому, что хотите научится программировать как профессионалы. Мы будем пользоваться Microsoft Visual Studio, чтобы программировать на C# или Visual Basic. У вас будут полные возможности отладки, а также многофункциональная среда разработки.

Предварительно рекомендуем ознакомиться с Microsoft MakeCode.

Ну что ж, пристегните ремни безопасности, так как мы отправляемся в серьезное путешествие по стране профессионального программирования, которое может превратиться в будущую карьеру!

## TINYCLR OS™

Чтобы BrainPad работал с .NET на C # и Visual Basic, вам понадобится TinyCLR OS. Эта миниатюрная операционная система создает всю магию. Она интерпретирует скомпилированный код .NET и содержит тысячи таких сервисов, как потоки, управление памятью и отладка. Это тот же .NET, который используется для разработки программного обеспечения для телефонов или компьютеров, с той лишь разницей, что TinyCLR OS работает на миникомпьютерах, таких как BrainPad. Так что TinyCLR OS – это часть стандарта .NET.




## VISUAL STUDIO

Теоретически любой компилятор .NET может использоваться с TinyCLR OS; однако TinyCLR поддерживается только Microsoft Visual Studio. Visual Studio версия Comunity представляет собой полнофункциональную среду программирования, которая считается одной из лучших (если не лучшей), доступной - и, самое главное, она бесплатна! Для установки Visual Studio вам понадобится современный ПК с Windows. Рекомендуется использовать Windows версии 10.

Обратите внимание, что вы можете использовать ту же Microsoft Visual Studio и те же навыки, которые вы изучаете, программируя BrainPad, чтобы начать писать приложения для многих других устройств, таких как смартфоны, ПК и даже игровые консоли Microsoft Xbox!

## СИСТЕМНЫЕ НАСТРОЙКИ

Microsoft Visual Studio версия Community бесплатна, но это не игрушка, и она не маленькая. Вам потребуется некоторое время для настройки системы. Полные инструкции приведены на веб-сайте документации BrainPad по адресу https://docs.brainpad.com/go-beyond/system-setup.html.

В итоге вам нужно будет скачать и установить:

1. Microsoft Visual Studio версия Community. Не забудьте выбрать опцию «.NET desktop development» во время настройки.

2. GHI Electronics TinyCLR OS project system

Если вы запутались, то всегда сможете зайти на наш форум и получить ответы на ваши вопросы https://forums.ghielectronics.com/c/brainpad

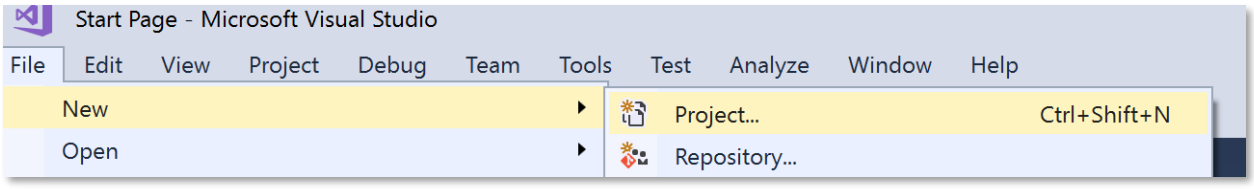
Теперь, когда компьютер настроен, вам нужно загрузить TinyCLR OS на BrainPad. Скачайте файл прошивки TinyCLR OS здесь https://docs.brainpad.com/resources/downloads.html, а затем загрузите его на BrainPad так же, как вы это делали с Microsoft MakeCode:

1. Удерживайте кнопку reset в течение, примерно, трех секунд. Лампочка станет зеленой.
2. ПК обнаружит запоминающие устройство и откроет новое окно.
3. Сохраните прошивку TinyCLR OS на BrainPad. Вы также можете перетащить файл или скопировать и вставить файл в окно BrainPad.

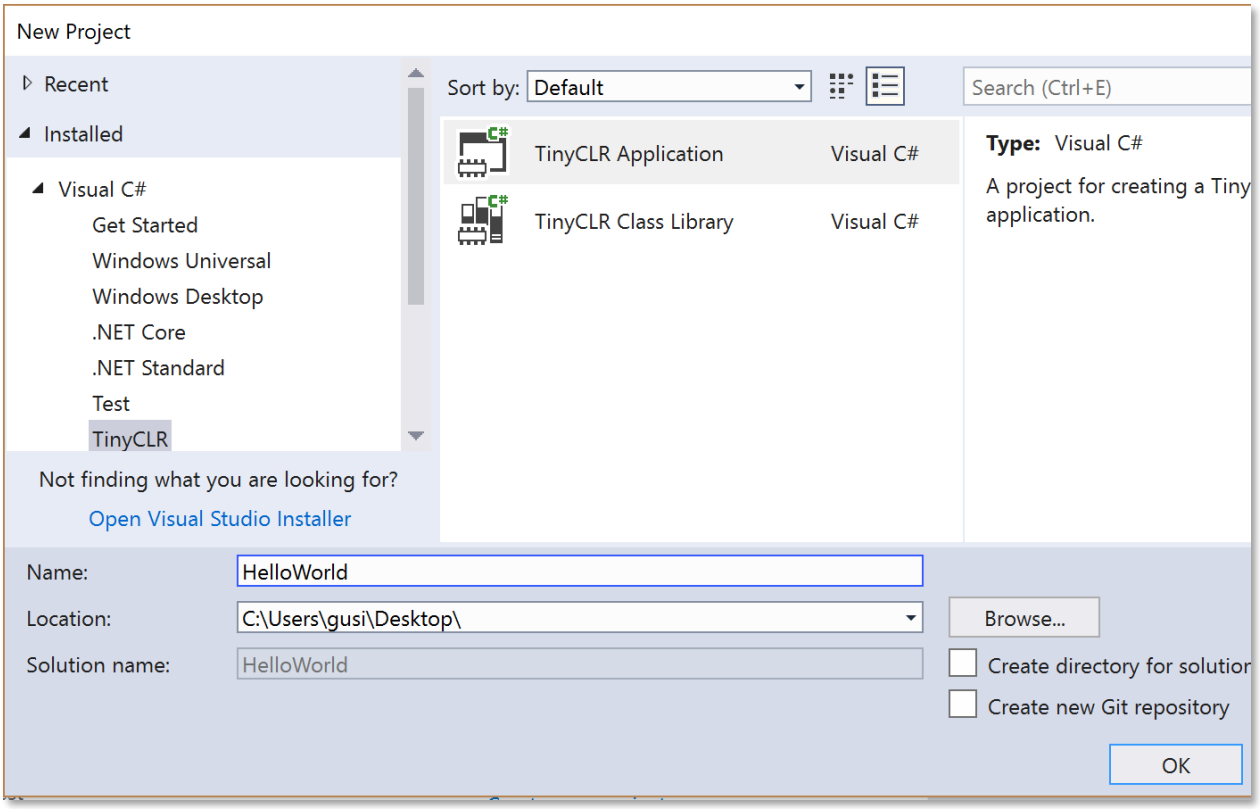
## HELLO WORLD

Обычно, когда программируют новое устройство или начинают использовать новые инструменты для программирования, чтобы убедиться, что все работает правильно, пишут очень простую программу. Эта программа обычно называется программой «привет мир» или «Hello world». Давайте сделаем это сейчас.

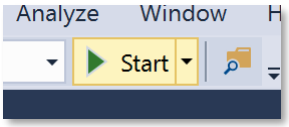
Откройте Microsoft Visual Studio и начните новый проект. В меню "File" выберете "New" и ""Project" (File > New > Project).




В Visual C # должна быть опция TinyCLR. Если нет, то вы не установили TinyCLR OS project system. Этот проект я назвал "HelloWorld" и разместил его на своем рабочем столе.




Теперь вы можете запустить эту пустую программу на своем BrainPad. Она не сделает ничего полезного, но система проверит, что все в порядке и все работает как мы ожидаем. Подключите BrainPad и нажмите "Start".

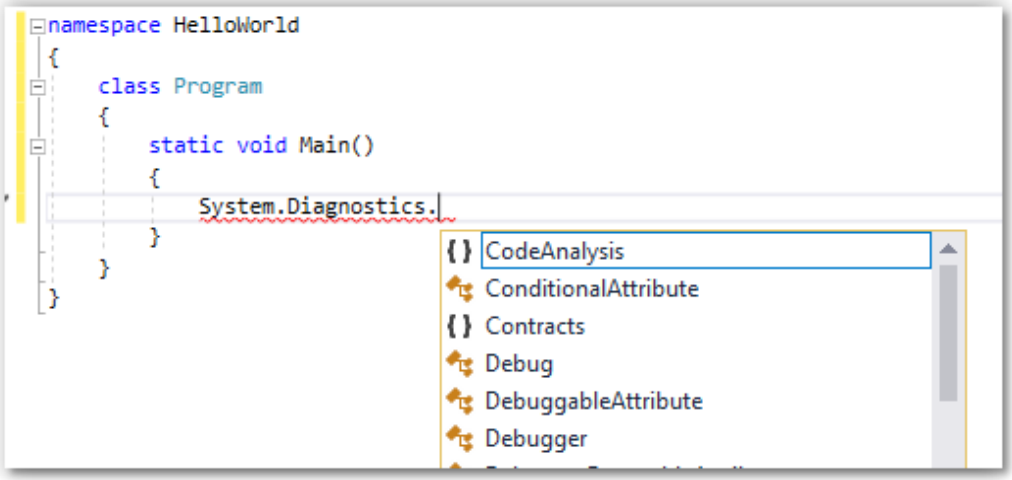



Наблюдайте за процессами, которые отображаются в окне вывода. Если этого окна не видно, нажмите Alt + Ctrl + O на клавиатуре или выберите "Output" в меню "View" (View > Output).

Теперь мы готовы сказать BrainPad'у «Hello World». Сначала мы выведем на печать «Hello World» в окне вывода Visual Studio. Позже мы выведем его на дисплее BrainPad. Добавьте эту строку в программу прямо под левой скобкой ({), следующей за Main ().

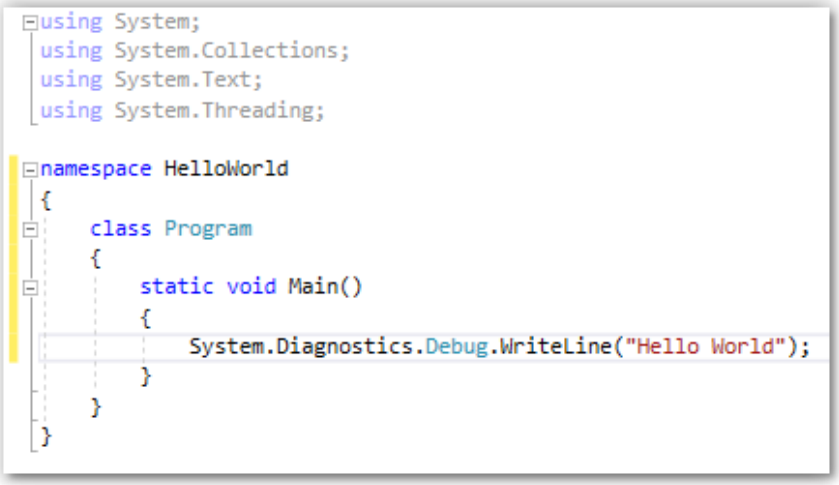
```
System.Diagnostics.Debug.WriteLine("Hello World");
```

Обратите внимание, что Visual Studio автоматически "догадывается" что вы хотите набрать на клавиатуре.




Когда будет готово, код должен выглядеть так:

Нажмите на "Start" и наблюдайте за окном вывода.

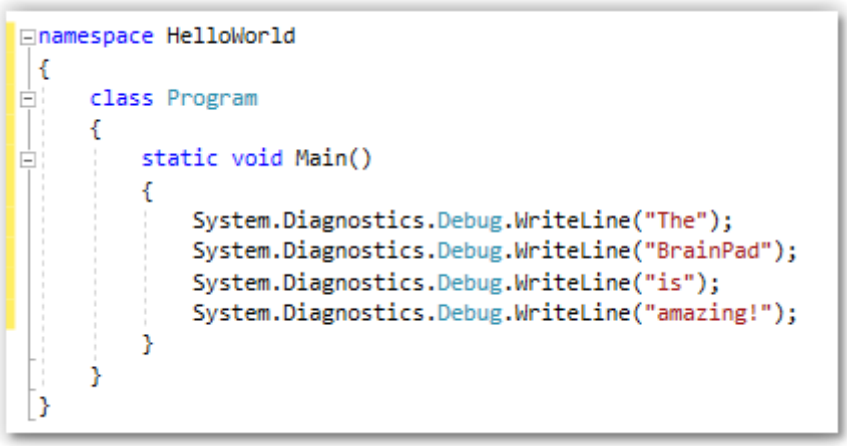



```
The debugging target runtime is loading the application assemblies and starti
Ready.

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).
```

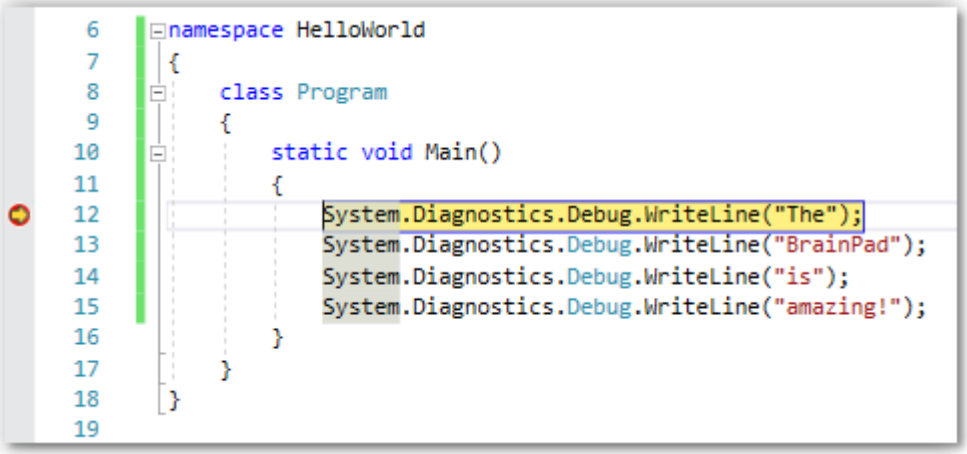
Вы нашли «Hello World»? Это было хорошее начало, но не очень захватывающее. Попробуем вывести на экран несколько строк. Замените ранее добавленную строку этими четырьмя.

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

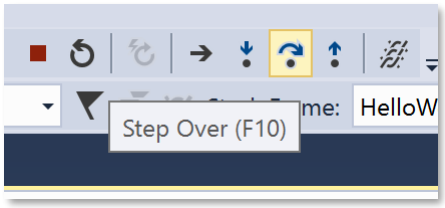
Поместите курсор в первую строку и нажмите функциональную клавишу F9. Это добавит точку инспекции кода или проще "breakpoint". Вы можете сделать то же самое из меню "Debug".




Запустите программу как вы делали ранее. Как только исполнение кода дойдет до breakpoint, программа остановиться в той же линии.



Программа остановится прямо перед тем, как эта строка должна быть выполнена. Нажмите клавишу F10 или нажмите кнопку пошаговой инспекции кода.

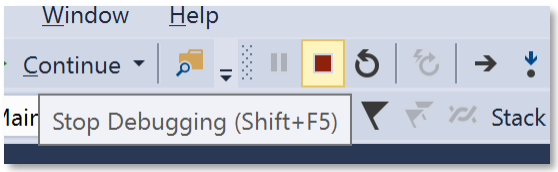



Это заставит программу выполнить одну строку кода за один шаг. Нажмите (F10), и наблюдайте за окном вывода. Это называется пошаговая инспекция кода. Эти действия чрезвычайно полезны, когда мы отлаживаем нашу программу, и дают нам возможность понять, почему что-то не работает должным образом.

## НЕСКОНЧАЕММЫЕ ПРОГРАММЫ

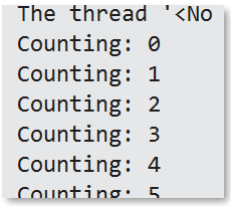
На телефоне или ПК есть смысл закрывать или заканчивать программы. Однако на небольших устройствах, работающих под управлением микроконтроллеров, программы обычно не заканчиваются. Например, микроволновая печь. У нее есть небольшой «мозг» внутри, который всегда контролирует клавиши для запуска процессов и запускает часы на экране. Если вы не отключаете микроволновую печь, эта программа всегда работает. Это называется бесконечным циклом и часто используется в программировании.

Давайте создадим программу-счетчик на BrainPad, которая запускается и работает всегда. В общем, BrainPad будет считать один раз в секунду и никогда не останавливаться. Обратите внимание, что перед тем, как вы хотите что-то изменить, вам необходимо остановить любую запущенную программу. Вы можете сделать это, нажав кнопку "Stop". Ниже приведен код, который должен выглядеть так.

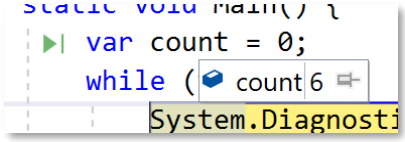



```
namespace HelloWorld {
    class Program {
        static void Main()
        {
            var count = 0;

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

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

Окно вывода покажет счетчик.

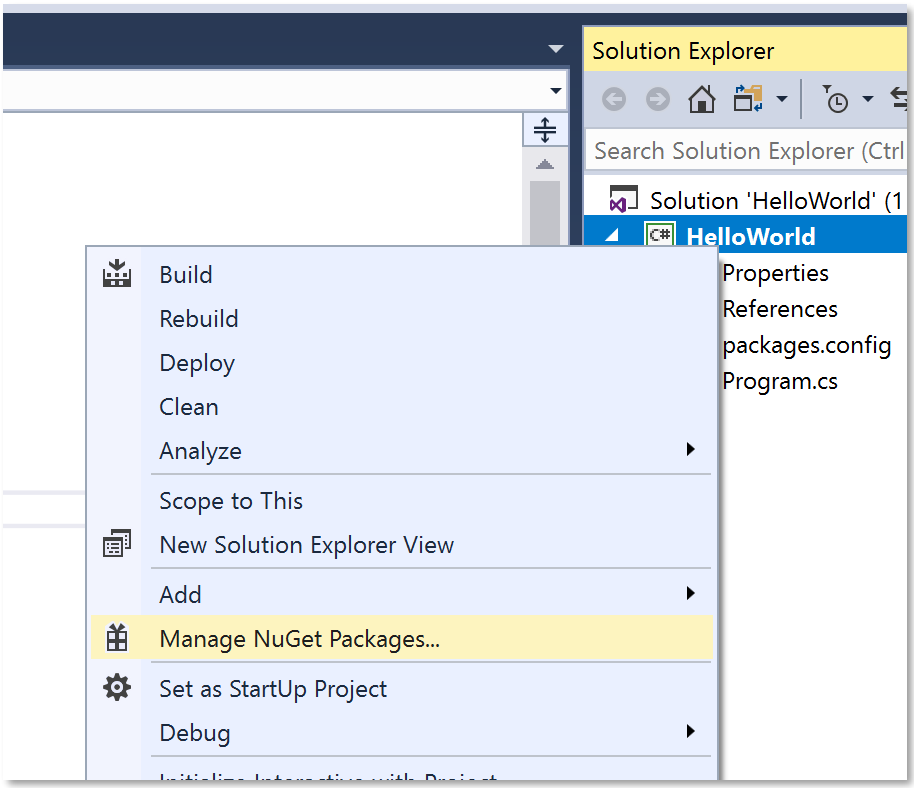



Когда программа запущена, вставьте breakpoint в строку счетчика. Оттуда вы можете "пройтись" по коду, как мы это делали раньше. Теперь наведите указатель мыши на нашу переменную "count", и Visual Studio покажет вам текущее значение переменной. Когда я остановил программу, она была равна 6. Кстати, проверка переменных - еще одна очень полезная функция для отладки программ.

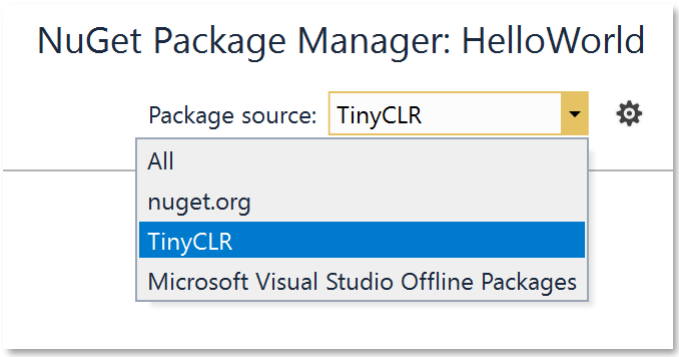



## BRAINPAD БИБЛИОТЕКИ

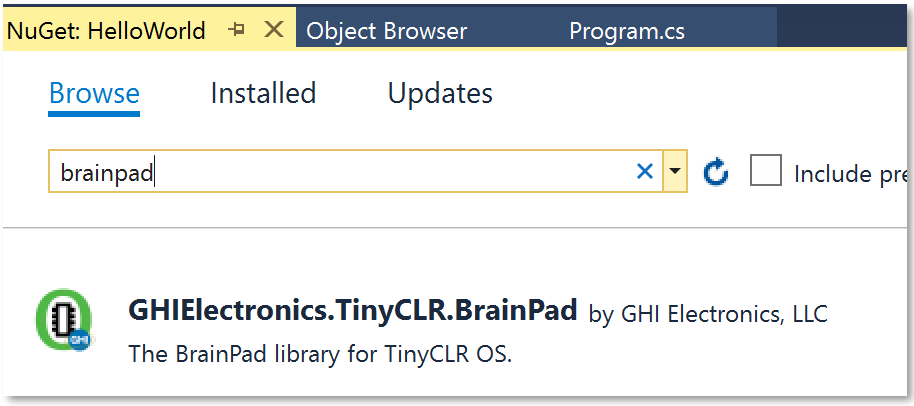
До сих пор мы использовали "чистую" TinyCLR OS для запуска простых программ. BrainPad поставляется с библиотеками, чтобы помочь в использовании всех доступных портов ввода-вывода. Щелкните правой кнопкой мыши на проект в окне Solution Explorer и выберите «Управление пакетами NuGet» ...




NuGet - это онлайн-сервис для размещения библиотек. Вы также можете загружать библиотеки и размещать их локально на своем ПК. Выберите путь к библиотеке. Например, у меня есть библиотеки, размещенные локально в каталоге с именем TinyCLR.

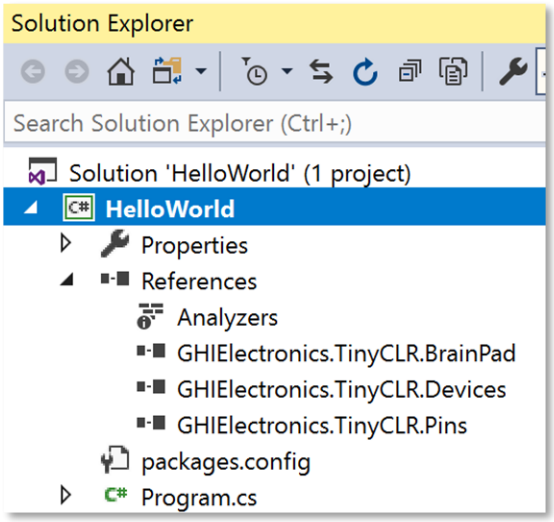



Во вкладке Browse, наберите "brainpad" в поиске.

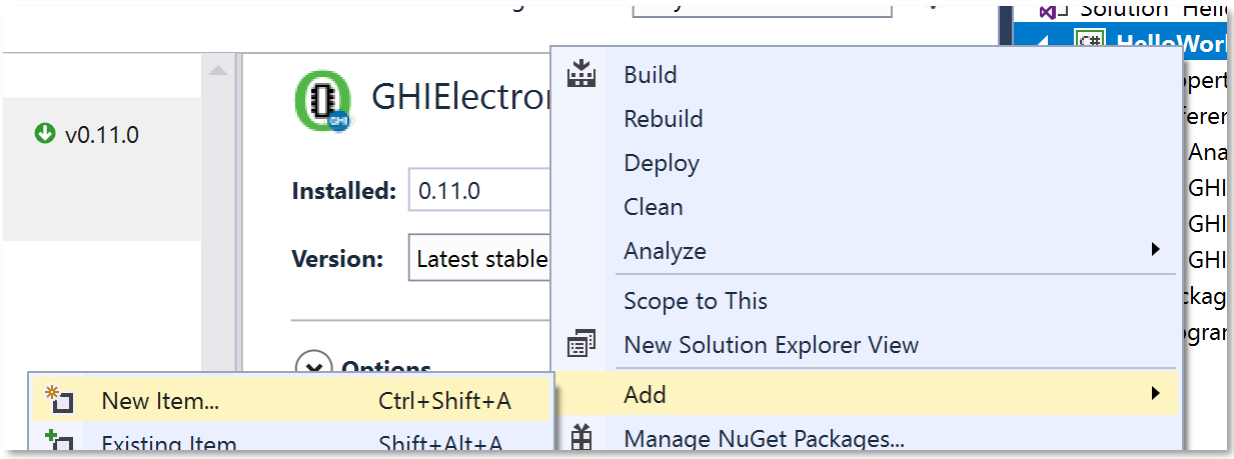



Вы увидите библиотеку GHIElectronics.TinyCLR.BrainPad. Выберите ее и нажмите "Install".

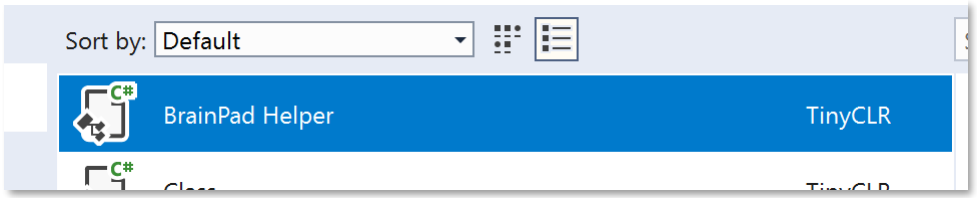

Так вы добавите необходимые библиотеки по поиску в explorer.




Добавить библиотеки в ваш код очень легко, для этого вы можете использовать вспомогательный класс. Щелкните правой кнопкой мыши по проекту, выберите "Add > New Item...". Вы также можете использовать комбинацию клавиш Ctrl + Shift + A.




Из доступных опций выберете BrainPad и нажмите кнопку "Add".




Сейчас начнется самое интересное! Перейдите в окно Program.cs, где вы написали программу для счетчика ранее. Измените код, чтобы ваш счетчик отображался на дисплее BrainPad.

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

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

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

Строка, которая отвечает за вывод счетчика, аналогична команде Debug.WriteLine, которую мы использовали ранее, за исключением того, что она выводится на дисплее BrainPad. Вторая строка необходима для обновления экрана. Дисплеи намного медленнее, чем процессоры. Если мы будем обновлять экран каждый раз, когда мы рисуем, дисплей будет работать очень медленно. Вместо этого процессор обращается к своей памяти. Только когда вызывается RefreshScreen, дисплей обновляется.

Подумайте о видеоигре, где вам нужно нарисовать корабль, препятствия, врагов... и т.д. Вы будете это рисовать только в памяти, которая работает быстро и обновлять дисплей только по мере необходимости.

Последняя строка - попросить BrainPad подождать одну секунду.


## ПОДПРЫГИВАЮЩИЙ МЯЧ

Мы уже говорили о том, что практически все языки программирования используют английский язык и, если вы его понимаете, то понять язык программирования для вас будет еще проще. Так вот, читать код, написанный с помощью библиотеки BrainPad, это как читать увлекательную книгу на английском языке. Добавьте некоторую логику, и вы можете создавать удивительные программы. Преподавание C # / Visual Basic не рассматривается в этой книге, но это не значит, что мы не можем немного повеселиться!

Вернемся к исходной программе и нарисуем круг диаметром 5 пикселей. Центр его поместим в точке с координатами 30,30. Координаты на дисплеях на компьютерных системах начинаются в верхнем левом углу, и задаются как 0,0. Перемещение фигур направо делается путем увеличения X (первого числа). 30,0 задает положение точки 30 пикселей от левого края экрана и 0 пикселей от верхнего края, то есть в самом верху дисплея.

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

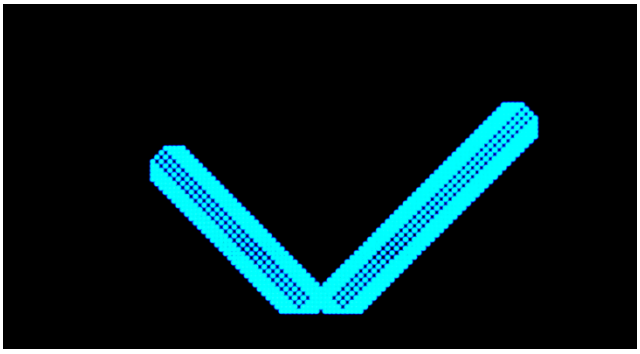
И у нас получился круг!

В нашем бесконечном цикле мы будем увеличивать переменные X и Y в каждом цикле. Поскольку мы начали с 30, переменная будет меняться так 31, 32, 33, 34 ... и т. д. Все интересней и интересней. Не так ли?

Круг двигается очень медленно из-за односекундной задержки. Мы изменим это значение на 0,01. Также добавим некоторую логику, чтобы круг / шар отскакивал обратно. Для этого нам нужны новые переменные dx и dy. Они будут хранить в себе направление, в котором двигается мяч. Затем мы проверяем, находится ли шар у края. Если это так, мы изменим направление.

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

Круг перерисовывается при движении по экрану:



Не похоже на двигающийся шар... Проблема в том, что нам нужно очищать дисплей каждый раз перед тем, как нарисовать новый круг. Добавьте эту строку прямо перед DrawCircle.

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

Вы хотите ускорить его? Задержка у нас 0,01 очень мала, поэтому проблема здесь не в этом. Мы перемещаем круг по одному пикселю в каждом цикле. Переместите его на 5 пикселей вместо этого, и он будет в 5 раз быстрее. Это достигается путем изменения начального значения dx и dy.

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

Какой шар отскакивает, не издавая шума? Используйте функцию "Beep" каждый раз, когда мяч отскакивает. Добавьте эту строку внутри каждого оператора "if".

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

Ага! Теперь это прыгающий мяч! Но это слишком громко, особенно если вы в классе с 30 BrainPada'ми! Добавьте проверку, чтобы мяч издавал звук только при нажатии кнопки вниз.

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

Можете ли вы добавить крутой эффект, делая мяч больше и больше во время движения? Подумайте, как изменить положение шара в пределах установленных границ. Мы будем делать то же самое. Нам нужна переменная R для радиуса и переменная dr для направления радиуса, растущего или сжимающегося.

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

Затем мы увеличим размер с границами от 1 до 8.

```
r = r + dr;

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

Конечно, нам нужно изменить строку DrawCircle и использовать переменную R вместо 5

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

Вот полный код. Но прежде, чем его копировать и вставлять, убедитесь, что вы назвали свой проект HelloWorld (без пробелов между словами, с учетом регистра), если же у вас другое название, тогда нужно изменить пространства имен (namespace) соответственно названию вашего проекта, а не копировать. Например:

```
namespace SomethingElse
```

Затем сохраните строку namespace, но скопируйте остальную часть кода, все, начиная с класса Program и заканчивая второй и последней правой скобкой (}). Последняя скобка на самом деле является частью пространства имен.

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

## РОЖДЕСТВЕНСКИЕ ОГНИ

Вы правильно прочитали этот заголовок. Мы будем создавать рождественский свет с одним светодиодом. Для этого мы будем использовать одну из тысяч встроенных в TinyCLR OS функций. С помощью этой функции мы получим случайное число. В отличие от реальной жизни, в компьютере нет ничего, действительно, случайного. Это может затруднить нашу задачу. К счастью, есть встроенная функция случайных чисел, которая упрощает ее. Мы просто создаем случайный объект, а затем задаем ему следующее случайное значение.

Мы будем использовать эти случайные числа, чтобы установить основные цвета светодиодов красного, зеленого и синего. Вы уже знаете, что эти три цвета могут создать любой цвет, который вы хотите.

```
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);
    }
}
```

## ВИДЕТЬ СВЕТ

Допустим, мы хотим поставить растение где-нибудь в помещении, и нам любопытно знать сколько света в этом месте. Мы хотим увидеть, как меняется уровень освещенности в течение дня в виде диаграммы.

Диаграмма очень проста. Мы будем рисовать вертикальные линии, начало которых задается уровнем освещенности, а конец в самом низу дисплея. Каждая следующая линия сдвигается на один пиксель вправо с каждой новой итерацией в цикле.

```
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);
    }
}
```

Вероятно, вы хотите, чтобы программа остановилась, когда линии дошли до самого правого края, но в этом примере мы очистим дисплей и начнем с самого начала. Давайте быстро запустим цикл, чтобы мы могли видеть результаты. Если вам нужен график освещенности за два часа, а экран имеет ширину 128 пикселей, вам потребуется одноминутная задержка на каждом шаге. Это позволит записать 128 минут по ширине дисплея.

Вы также можете сохранить программу как есть и накрыть датчик освещенности рукой, чтобы сделать несколько интересных рисунков!

Поскольку вы уже эксперты, попробуйте дать ответы на эти вопросы.

1. Почему мы делим показания уровня освещенности на 2?
2. Почему мы установили начало линии в BrainPad.Display.Height соответственно уровню освещенности?
3. Как вы думаете, почему x инициализируется со значением 300? Что произойдет, если вместо этого использовать 0?

Вот ответы, но пообещайте мне, что попробуете самостоятельно. Уровень освещенности находится в пределах от 0 до 100. Дисплей имеет высоту 64 пикселя. Если мы разделим уровень пополам, максимальное значение будет равно 100/2 или 50. При отображении на 64 пикселя мы получим 14 пикселей в верхней части экрана для отображения уровня. То есть это масштабирование.

Ответ на второй вопрос заключается в том, что мы хотим показать увеличение уровня освещенности. Меньшие координаты "y" показаны в верхней части дисплея, но мы хотим, чтобы нижние уровни

освещенности появлялись в нижней части дисплея. Вычитая уровень освещенности от высоты дисплея, мы инвертируем график.

Ответ на третий вопрос. Установить на старте большое значение "x" – это классный трюк, который поможет при первом запуске кода. Взгляните на программу внимательно. Мы говорим программе, что дисплей уже заполнен показателями уровня освещенности и его нужно очистить. То есть, с самого начала программы выполнится условие if, дисплей очистится и на нем отобразится надпись "Light Level", что в общем будет происходить после каждого заполнения экрана показателями уровня освещенности. Попробуйте изменить его до 0, и показатели датчика не будут отображаться на экране до первого заполнения дисплея. Вы можете просто очищать дисплей после каждой итерации, но это замедлит вашу программу. Мы всегда должны мыслить, как программисты, когда мы пишем программу и делать только то, что необходимо.

## МНОГОПОТОЧНОСТЬ

Просить систему сделать несколько вещей одновременно очень сложно. Как правило, это затруднительно для пользователя небольших систем. Хорошей новостью является то, что TinyCLR OS обладает функционалом многопоточности и ее легко использовать. Это работает точно так, как для обычной системы, работающих в среде .NET. Многопоточность называется threading, где каждый поток является отдельной задачей, выполняющей часть программы.

Не уверены, нужно ли вам использовать потоки? Как бы вы написали программу, которая одновременно выводит на дисплей значение счетчика и мигает светодиодом? Конечно, если вы хотите увеличивать счетчик с каждым миганием, тогда это легко.

```
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;
    }
}
```

Теперь измените программу, чтобы светодиод мигал один раз в секунду, но значение счетчика менялось как можно быстрее. Вы не обойдетесь без потоков. Начните с перемещения кода для мигающего светодиода в свой собственный метод. Метод - это фрагмент кода, который выполняет определенную задачу. Вы вызываете этот метод для обработки этой задачи. Например, LightBulb имеет метод TurnGreen, который зажигает его зеленым цветом.

```
static void Blink()
{
    while (true)
    {
```

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

Если вы вызовете этот метод, он будет мигать один раз в секунду; однако он никогда не выйдет из своего цикла while. Это нормально, потому что мы будем вызывать этот метод из отдельного потока. Готовы ли вы создать сложный код для потока? Вот он!

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

Мы просто сказали системе создать новый поток для метода Blink и затем запустить этот поток.

Стоп! Перед запуском кода нам нужно последнее изменение. В каждом потоке в системе должна быть задержка. Это помогает системе в обработке внутренних задач и позволяет другим потокам работать правильно. Но вы можете сказать, что у нас только один поток. На самом деле у нас есть два потока. Система всегда создает внутренний поток, который запускает метод Main () (основной метод запуска). Если вам нужно, чтобы программа работала как можно быстрее, просто добавьте минимально требуемую задержку.

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

Тут оба метода, Main и Blinking.

```
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;
    }
}
```

Есть два бесконечных цикла, и оба они работают одновременно. Хотя это очень простой пример, но он показывает возможности. В качестве упражнения добавьте еще один поток для подачи звукового сигнала каждые три секунды.

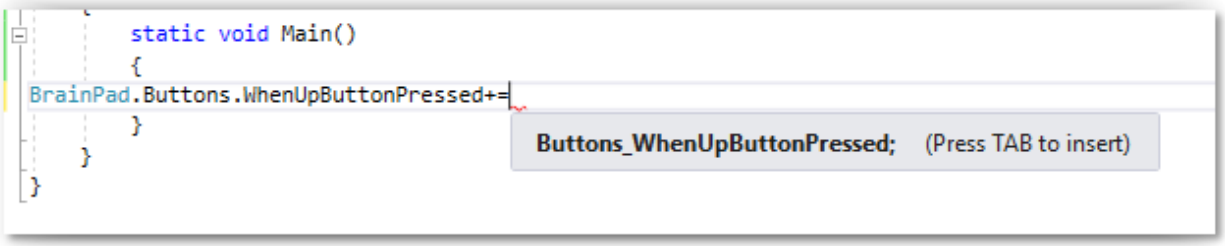
## ПОЗВОНИ МНЕ

Мы уже использовали кнопки, и вы знаете, как они работают. Однако мы всегда использовали их в цикле, где проверяли, была ли нажата кнопка. Система может проверить, нажата ли кнопка миллионы раз, прежде чем она будет нажата. Это очень плохо для устройств с питанием от батареи. Хорошо написанная программа уходит в спящий режим, когда ей не нужно обрабатывать никакие задачи. Это делается для экономии заряда батареи. Если бы ваш телефон не "засыпал", то не проработал бы и часа без подзарядки.

Эта тема довольно обширная, поэтому мы рассмотрим только события для кнопок. Вместо постоянной проверки мы попросим систему "позвонить" нам при нажатии. Мы хотим, чтобы каждый раз при нажатии кнопки вверх BrainPad издавал звуковой сигнал. А в цикле пусть мигает светодиод.

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

Нажмите кнопку "вверх", и ничего не произойдет. Удерживайте ее, и BrainPad будет подавать звуковой сигнал один раз в секунду. Это происходит, поскольку цикл имеет задержки, и кнопка проверяется один раз в секунду. Вы можете уменьшить задержку, но это все равно не решит проблему. Мы пойдем иным путем и будем использовать события. Visual Studio может автоматически генерировать необходимый код. Начните с ввода «BrainPad.Buttons.WhenUpButtonPressed», а затем добавьте «+ =» после этого. Теперь вы можете нажать клавишу табуляции, чтобы сгенерировать метод события.




Сгенерированный метод события выглядит следующим образом:

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

```
}
```

Сгенерированный код имеет строку для обработки ошибок (исключения). Можно считать, что это напоминание, чтобы вы не забыли заполнить данный метод нашим кодом. Заменим эту строку на звуковой сигнал.

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

Не забудьте удалить код, который проверяет кнопку в основном цикле.

```
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();
}
```

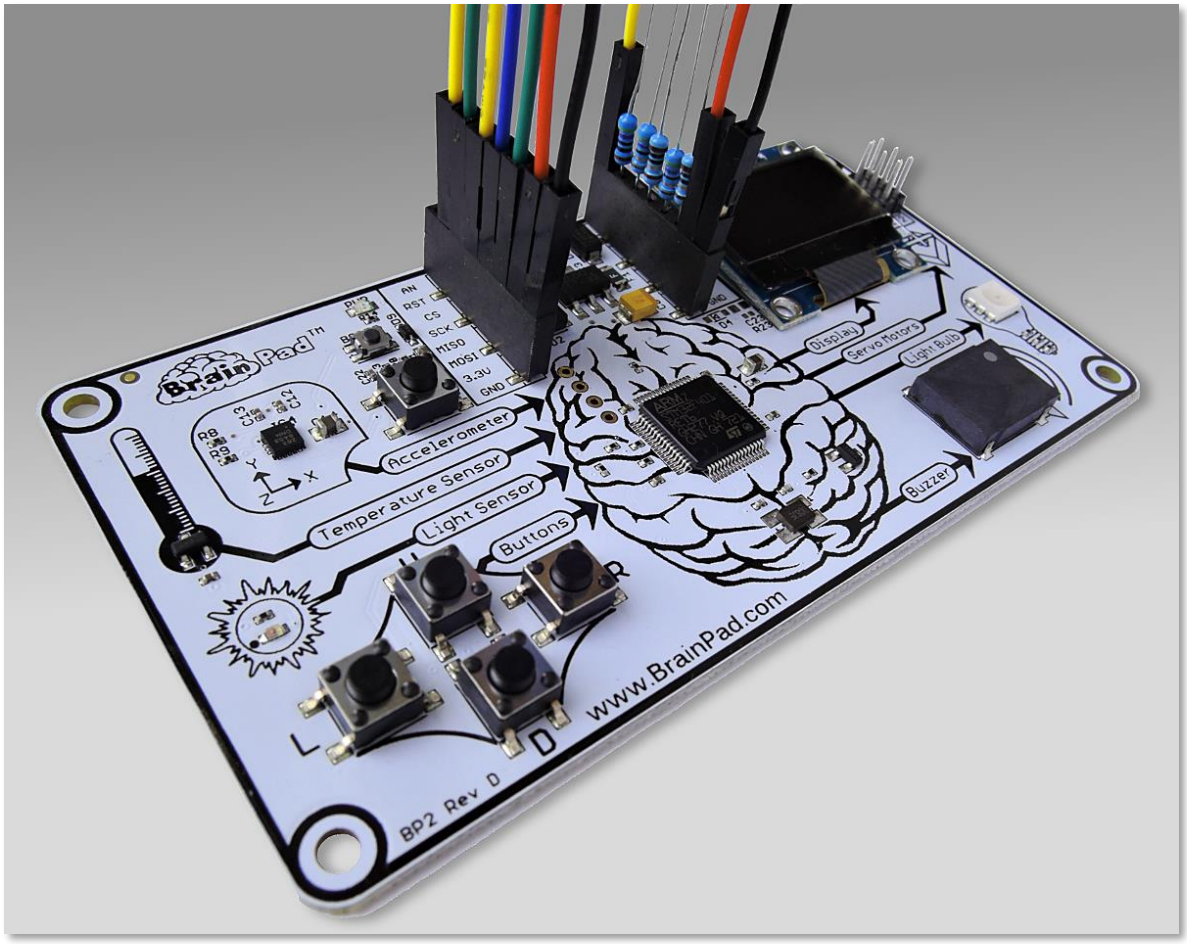
Кстати, несколько событий могут вызывать один и тот же обработчик событий. Вот код, который подает звуковой сигнал при нажатии любой из четырех кнопок.

```
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();
}
```

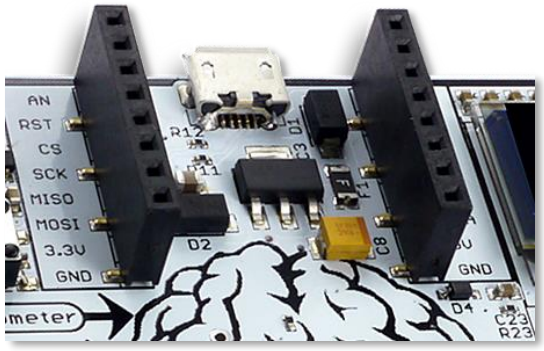
## ПУГАЮЩИЕ ПРОВОДА

Это обычное дело, когда новички в электронике боятся даже мысли, подключить что-то к чему-то проводами. Что будет, если я сделаю короткое замыкание или "сожгу" устройство?

Не переживайте по этому поводу! BrainPad готов к тяжелым испытаниям. Во-первых, он работает с очень низким напряжением 5В. Вы не можете получить травму, коснувшись 5В. Кроме того, BrainPad имеет встроенную защиту и предназначен для того, чтобы было очень сложно повредить его, подключив что-то неправильно. Тем не менее, вы должны понимать, что делаете. Есть много курсов и онлайн-статей, которые помогут вам понять схемы.

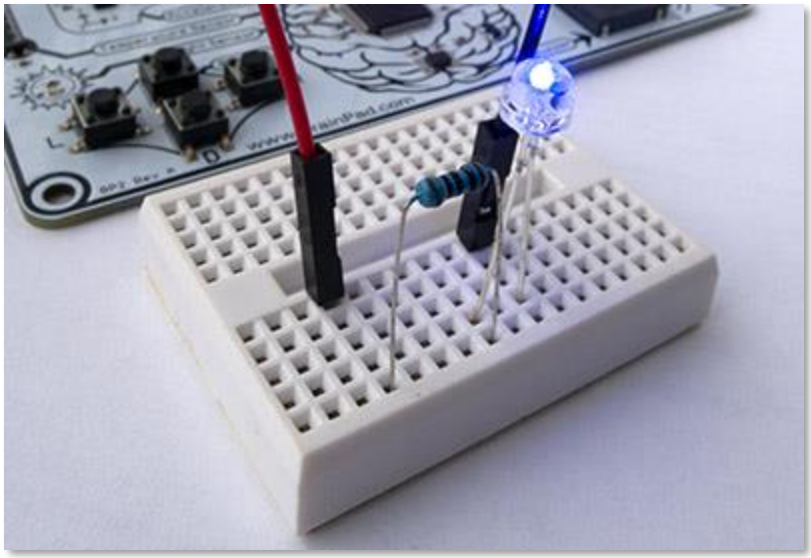



Как упоминалось в первой главе этой книги, существует несколько способов расширения возможностей BrainPad благодаря коннекторам (expansion connectors), расположенным рядом с разъемом USB.

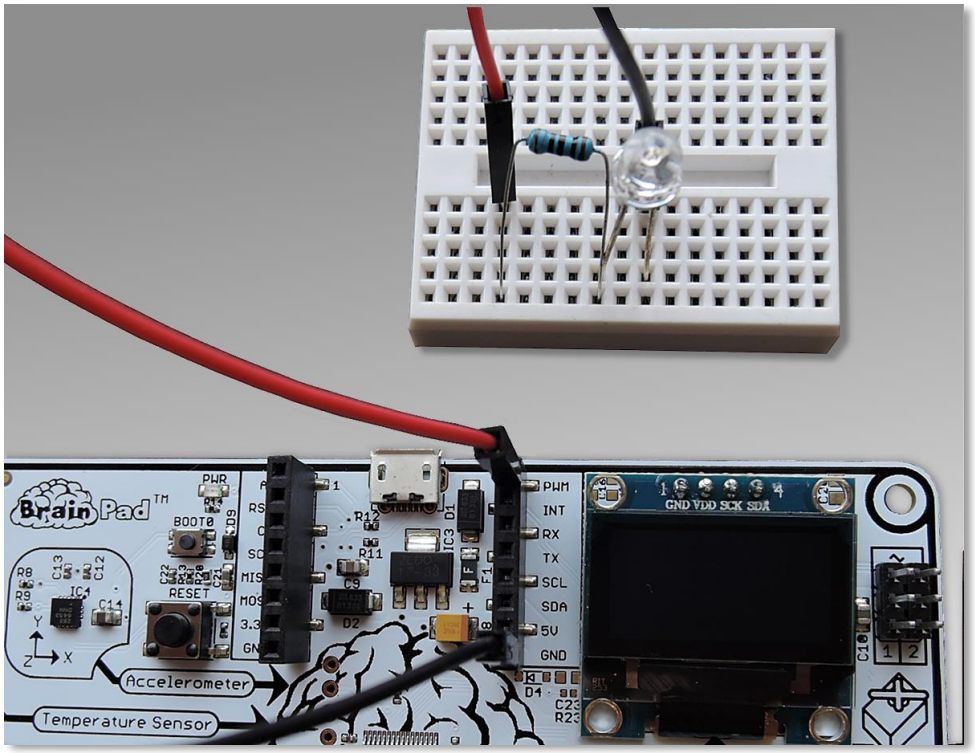



Эти разъемы имеют много функций, и нам потребуется несколько книг, чтобы полностью описать их. В этой книге мы приведем простой пример, подключив светодиод (LED).

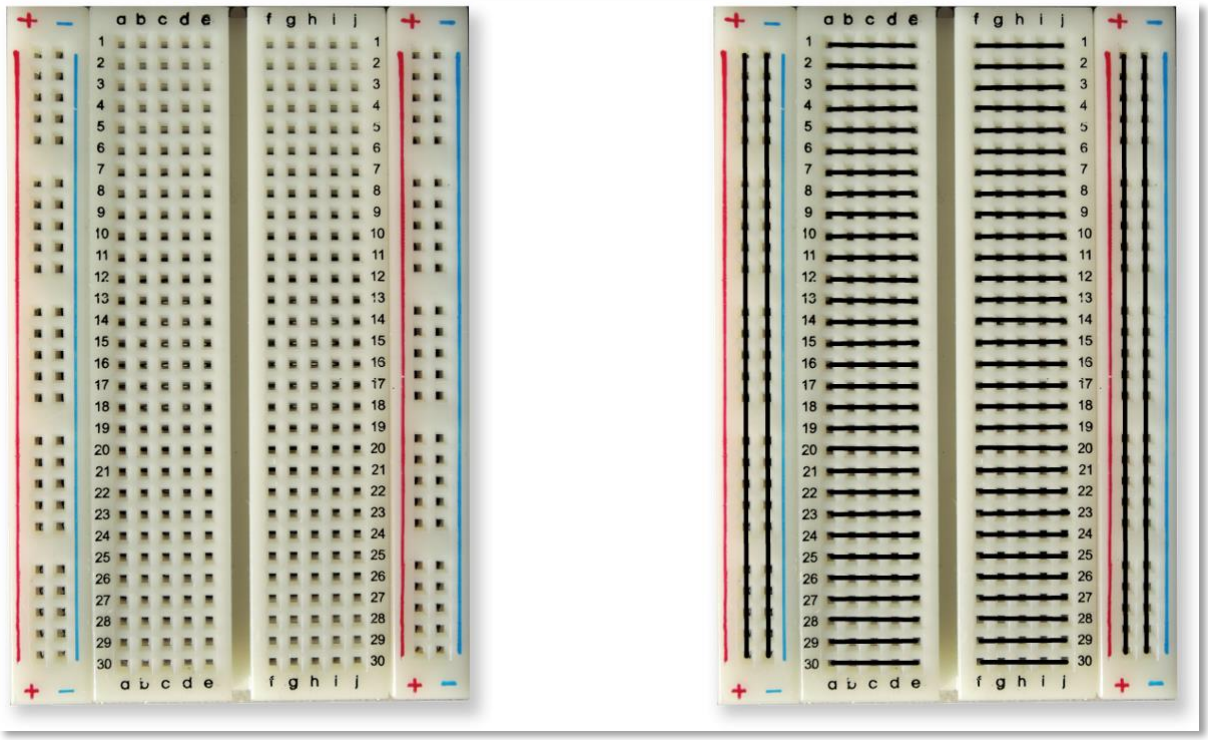
LED недорогие и имеют разные цвета. У них есть два контакта, один из которых длиннее другого. Некоторые светодиоды имеют более одного цвета (например, LED BrainPad). Для управления светодиодом необходим токоограничивающий резистор. Некоторые распространенные номиналы: 220, 330 или 470 Ом. Вам также понадобятся провода и макетная доска (bread-board).



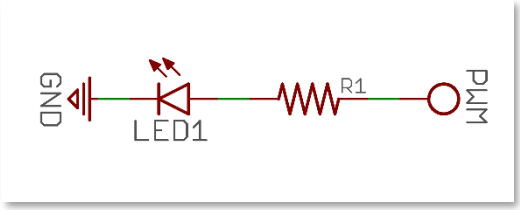
Мы можем использовать один из контактов GPIO в expansion connectors для управления светодиодом. GPIO означает выходной сигнал общего назначения. Мы будем использовать контакт с маркировкой PWM.



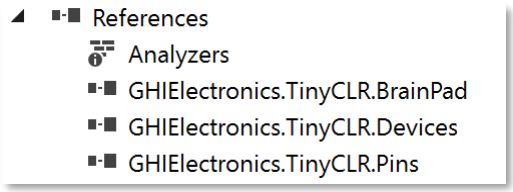

Bread-board имеет внутренние соединения, где каждый ряд отверстий замкнут. На изображении ниже bread-board справой стороны имеет черные линии, показывающие внутренние электрические соединения. Все, что подключено к отверстиям, соединенным черной линией, будет подключено между собой.



Провод будет соединять PWM контакт с контактом на bread-board, куда подключен длинный провод от светодиода. Короткий контакт светодиода должен быть подключен к резистору, а резистор - к заземляющему контакту. Это образует цепь (или схему), где электроны будут двигаться между PWM и GND. Когда PWM контакт активирован, загорится светодиод.




Для управления выводом PWM, который является GPIO, нам нужно будет добавить библиотеки GHIElectronics.TinyCLR.Devices и GHIElectronics.TinyCLR.Pins из NuGet. Эти библиотеки использует библиотека GHIElectronics.TinyCLR.BrainPad, которую мы всегда добавляем, поэтому никаких дополнительных действий не требуется, поскольку они автоматически загружаются.




Нам понадобится доступ к библиотекам GPIO в нашей программе. Это делается с помощью одной строки кода.

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

Нам нужно получить доступ к контакту по средствам контроллера GPIO, который находится внутри процессора.

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

Контакты (пины) могут иметь функцию как входа, так и выхода, точно так же, как входы и выходы на BrainPad. Входы подают сигналы в «мозг», а выходы подают сигналы из «мозга». Пример ввода - это кнопка, например, кнопки на BrainPad. Примером вывода может быть светодиод. Таким образом, контакт PWM должен быть выходным.

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

Теперь мы можем установить (write) уровень на контакте. Уровень может быть высоким (high), что означает «контакт активирован» или низкий (low) – «контакт отключен». Поместите это в цикл с некоторыми паузами. Результатом будет мигающий светодиод.

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

Полностью программа должна выглядеть так:

```
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);
    }
}
```

Подключение кнопки аналогично, за исключением того, что вывод должен быть входом вместо выхода. Не вдаваясь в подробности, хочу добавить, что контакт должен быть входом с подтягиванием. Это удерживает пин на уровне high до тех пор, пока не будет нажата кнопка.

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

Не нужно добавлять резисторы к кнопке. Просто подключите кнопку между одним из GPIO и GND контактами.

## ВЫВОДЫ

TinyCLR OS от GHI Electronics предлагает действительно профессиональный уровень программирования для BrainPad. Знания, которые вы получите, будут вам полезны для программирования любого устройства, от телефона до компьютера. И все это упрощается благодаря использованию нашей TinyCLR Operating System и Microsoft .NET Framework.

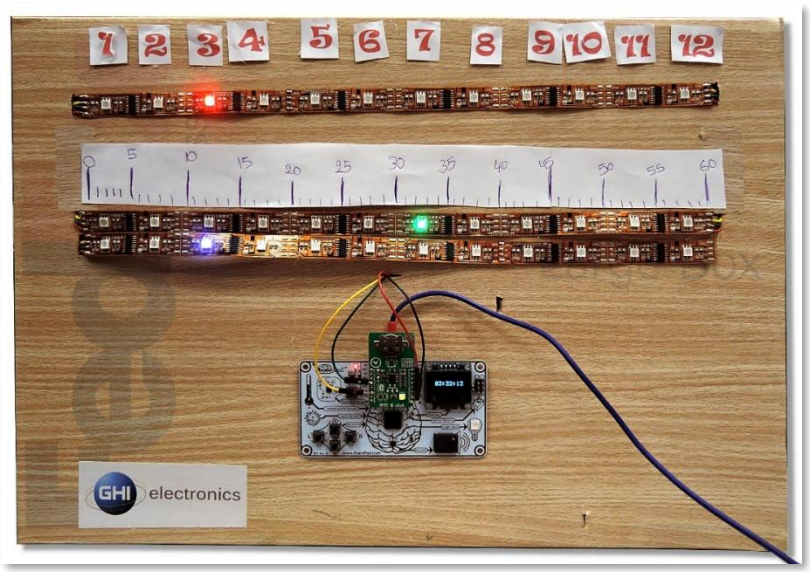
## РАСШИРЕНИЕ ВОЗМОЖНОСТЕЙ

Дополнительные контакты на BrainPad открывают целый мир возможностей. Здесь мы перечислим некоторые варианты, чтобы вы могли подумать о том, что возможно.

### MIKROELEKTRONIKA CLICK BOARDS™

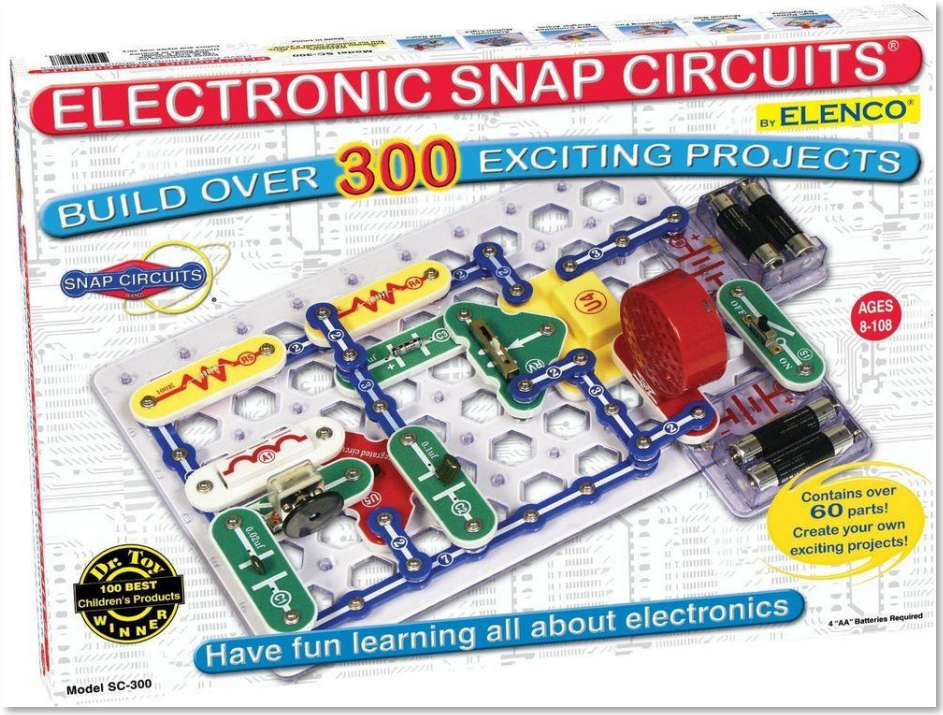
MikroElektronika предлагает сотни небольших модулей (click module), которые подключаются прямо к BrainPad. Модули включают в себя простые датчики, к примеру, для измерения влажности и температуры. Есть и более сложные устройства, такие как распознавание голоса и модули электрокардиограммы. Существует также множество управляющих модулей для управление двигателем и цифровым освещением. Вы можете найти их все на этом веб-сайте https://www.mikroe.com/click.

В этом примере проекта мы создали линейные часы, которые используют click module с функцией (RTC), чтобы отслеживать время, даже когда BrainPad не подключен



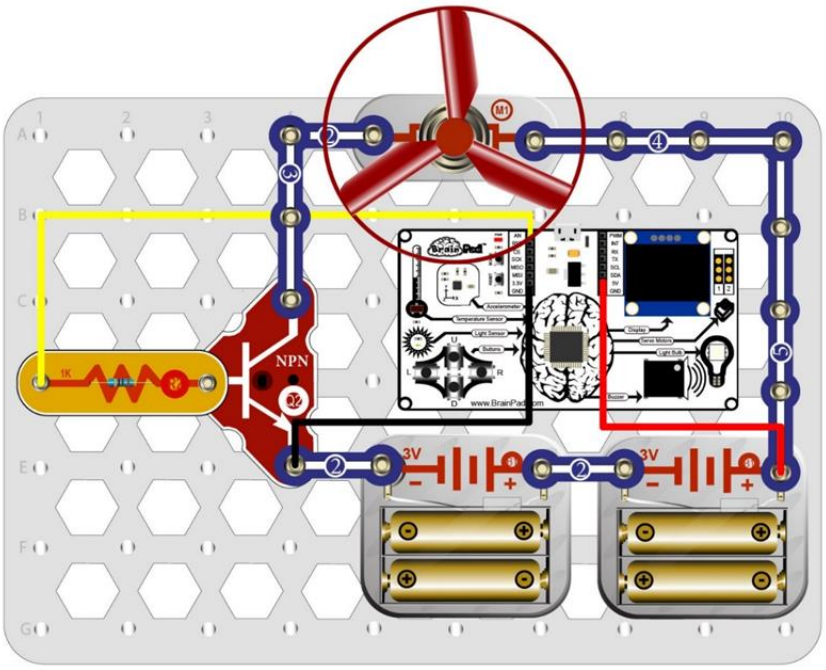

(https://docs.brainpad.com/projects/linear-clock.html).

## ELENCO® SNAP CIRCUITS®




Одна из самых популярных платформ для обучения электроники называется Snap Circuits, производства Elenco (https://www.elenco.com). Эти очень популярные наборы можно найти в интернет-магазинах и местных магазинах для радиолюбителей. Включают они в себя различные электронные компоненты и инструкции для создания множества различных проектов.

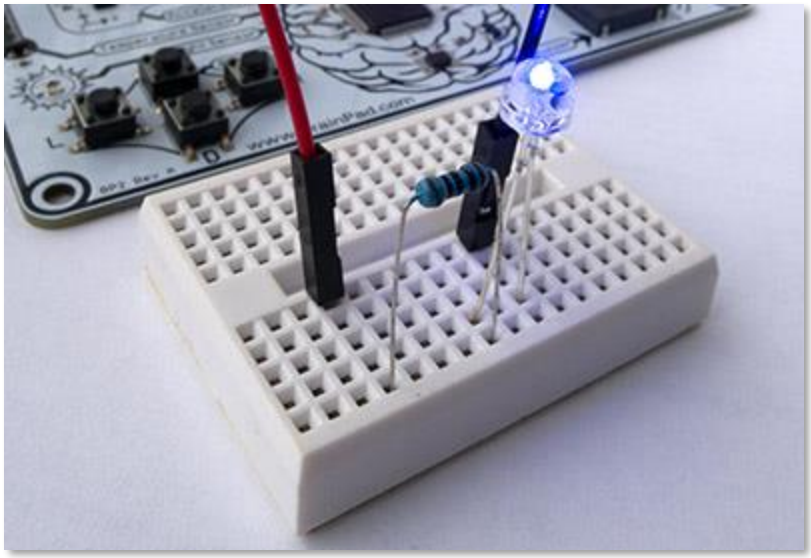
С BrainPad вы можете добавить интеллект и запрограммировать свои проекты по электроники. Этот пример проекта представляет собой таймер обратного отсчета, запускающий пропеллер. Рекомендуем посмотреть видео. Это один из наших фаворитов (https://docs.brainpad.com/projects/lift-off.html).




## BREADBOARDS

Хотите проектировать свои электронные схемы с нуля, как это делают профессиональные инженеры? Существует много способов создания схем, которые работают с BrainPad. Хотя проектирование схем требует немного больше знаний, это самый творческий и наименее дорогой способ расширить возможности вашего BrainPad. И это путь, который принесет вам больше всего знаний.

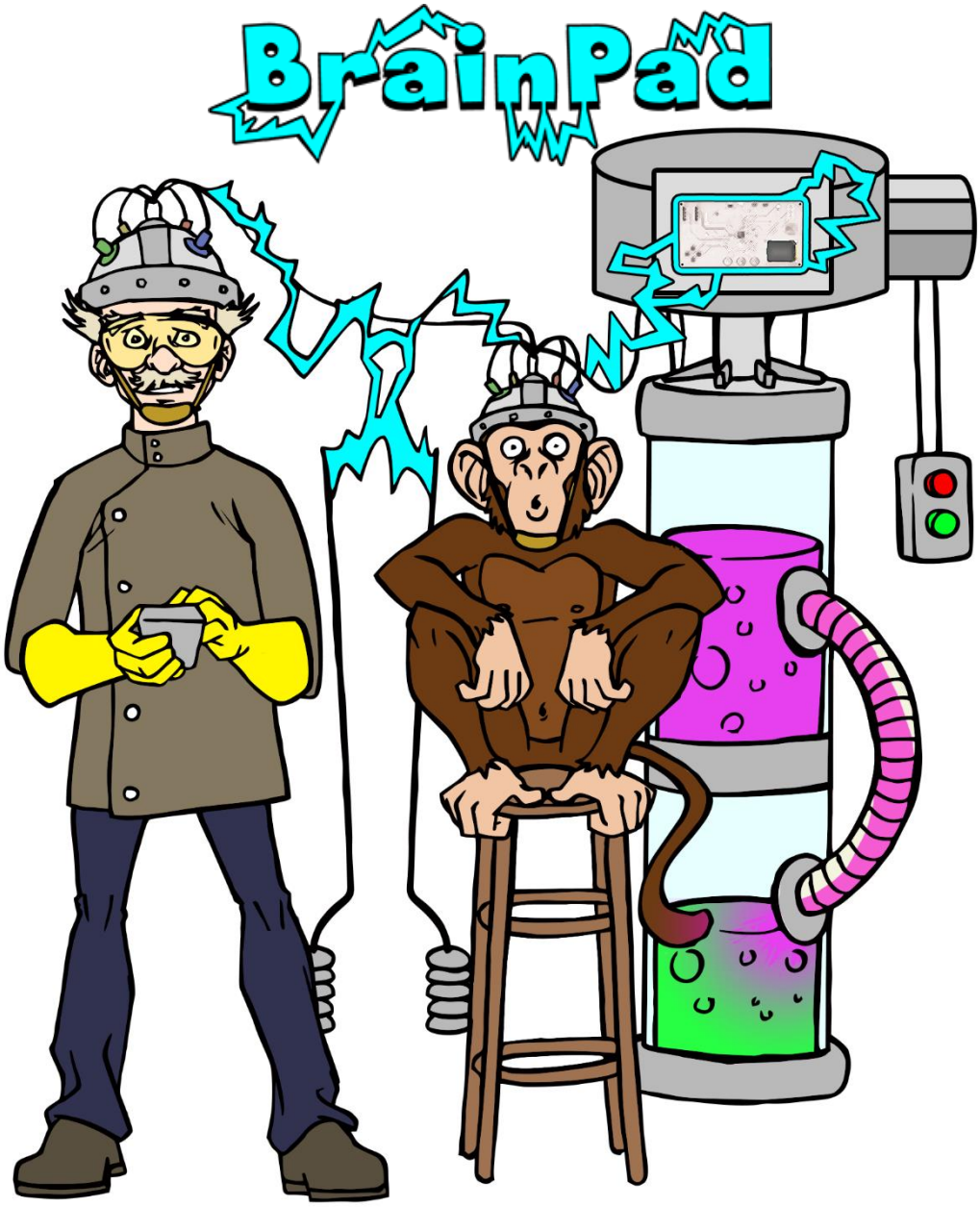
Самый простой способ начать создавать собственные схемы - использовать bread-boards макеты. Провода и компоненты просто вставляются в отверстия. Макет содержит компоненты и соединяет их вместе электрически. Кроме того, это самый быстрый способ протестировать новую схему, и исправить допущенные ошибки.



Если вы хотели бы использовать некий макет постоянно, есть несколько вариантов: от простых, предварительно изготовив прототип, до более сложных - проектирования и изготовления ваших собственных печатных плат. Чертеж печатных плат можно изготовить вручную или с помощью бесплатных компьютерных инструментов для проектирования. Это то, что делает BrainPad таким особенным – с ним можно легко начать, но вы можете продвигаться так далеко, насколько вам нравится и при этом работать в своем собственном темпе.

Вы также можете добавить к вашему проекту нужные вам датчики. Поищите в Интернете «sensor kit» и вы сможете найти множество разных вариантов. Вот один на пример:


Для использования этих модулей необходимы некоторые знания, однако у нас есть несколько простых проектов, которые помогут вам двигаться в правильном направлении. Как только вы познакомитесь с одним типом сенсора, становится намного легче понять, как работают другие датчики.




www.BrainPad.com