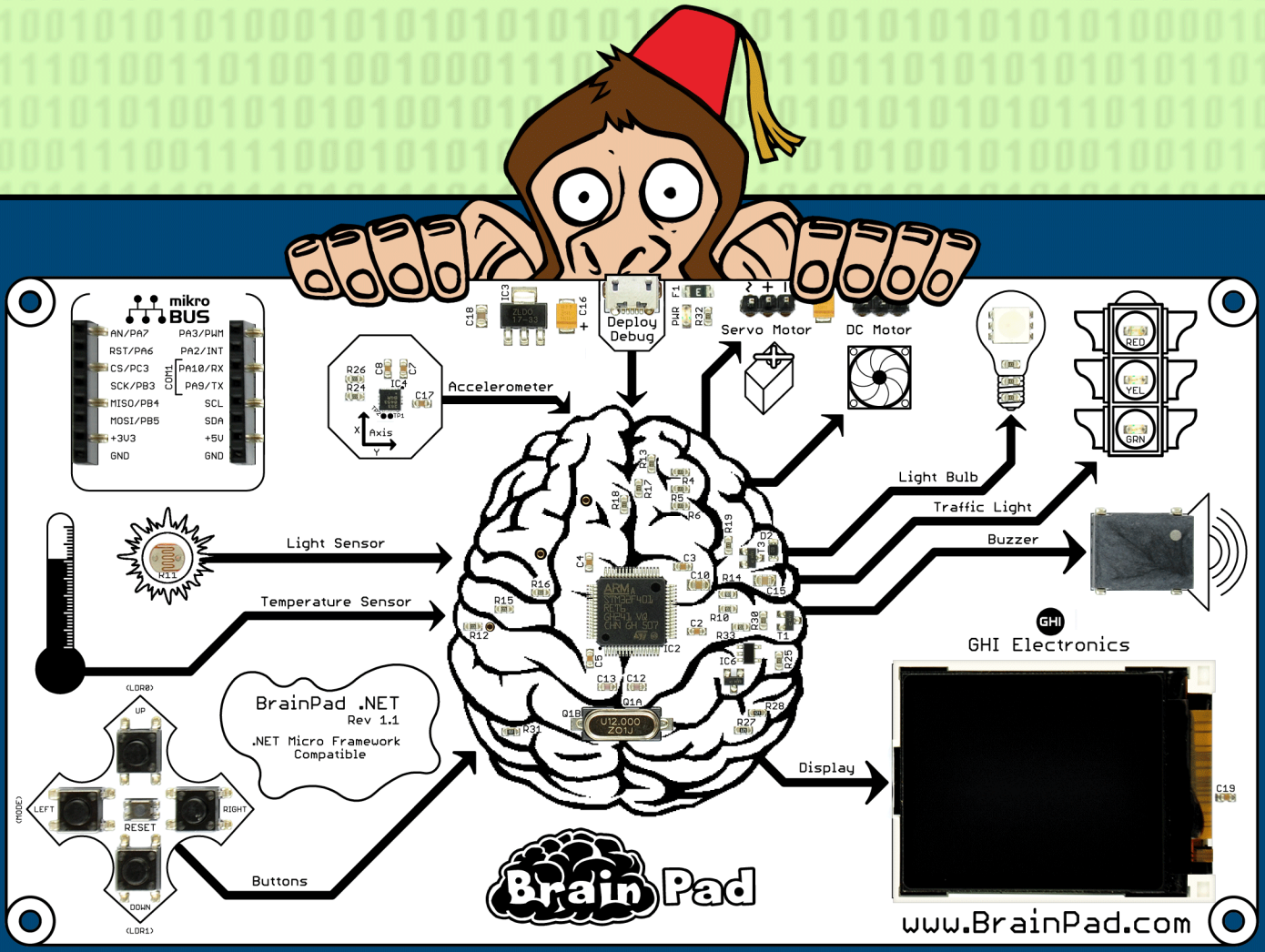
GHI electronics

Brain Pad

C#

INTRODUCTION



Contents

Introduction

The BrainPad circuit board is designed as a powerful educational tool that can be used to teach everyone from kids, to college students and professionals. Kids will start to learn programming using Visual Studio, one of the most widely used professional tools. College students and professionals that already know programming can use the BrainPad circuit board to learn about digital electronics and the connection between computing and the physical world.

Overview

Students will learn how to create projects in Visual Studio along with programming basics. Applications in this lesson will be limited to the `BrainPad` object, `if` statements, a `while` loop and the template functions.

Guidelines

- Prerequisites: None
- Ages 12 and up
- PC setup with Visual Studio, .NET Micro Framework and GHI Electronics' software.
- Supplies: BrainPad

Starting a New Project

Open Visual Studio and under Visual C# select .NET Micro Framework and choose the BrainPad .NET Application. Change the Location to `C:\Users\<YourUsername>\Desktop\` then name the project BrainPad_Project as shown in Figure 1.

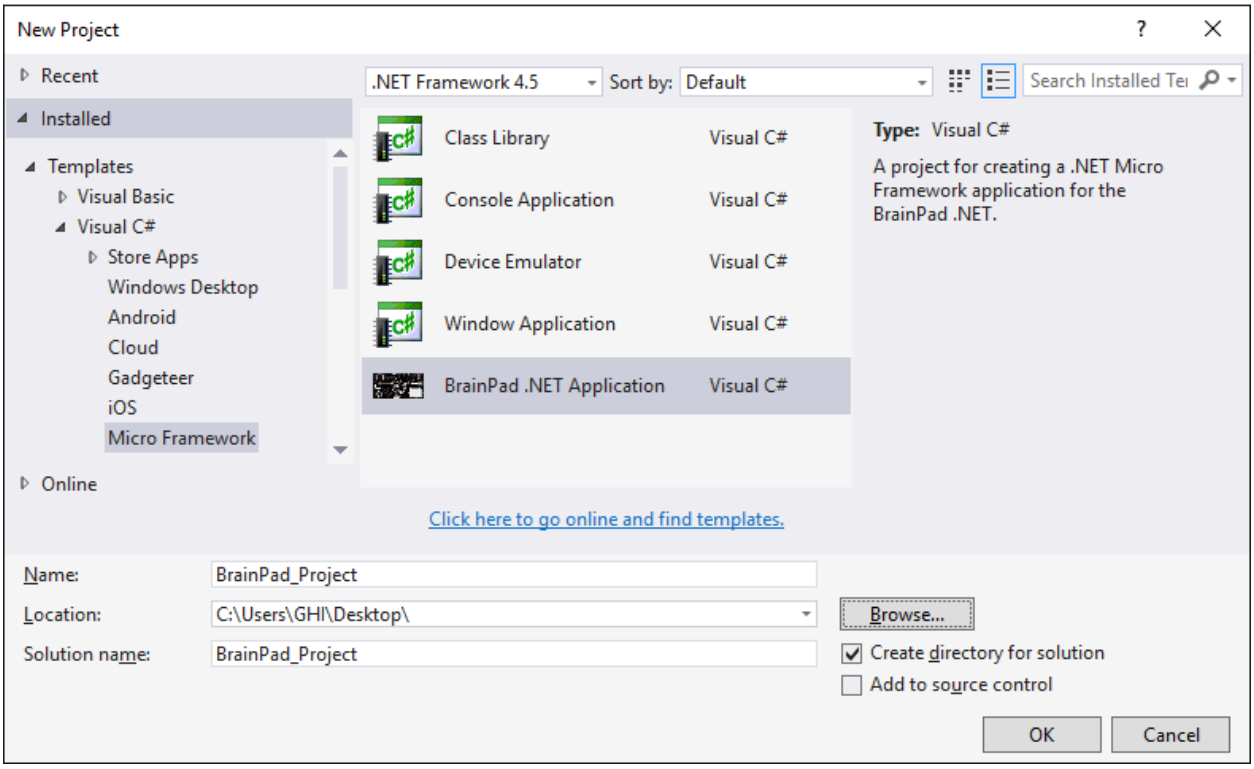


Figure 1 – Creating the BrainPad_Project BrainPad .NET Application.

Once created, you'll be presented with a Program.cs tab as shown in Figure 2.

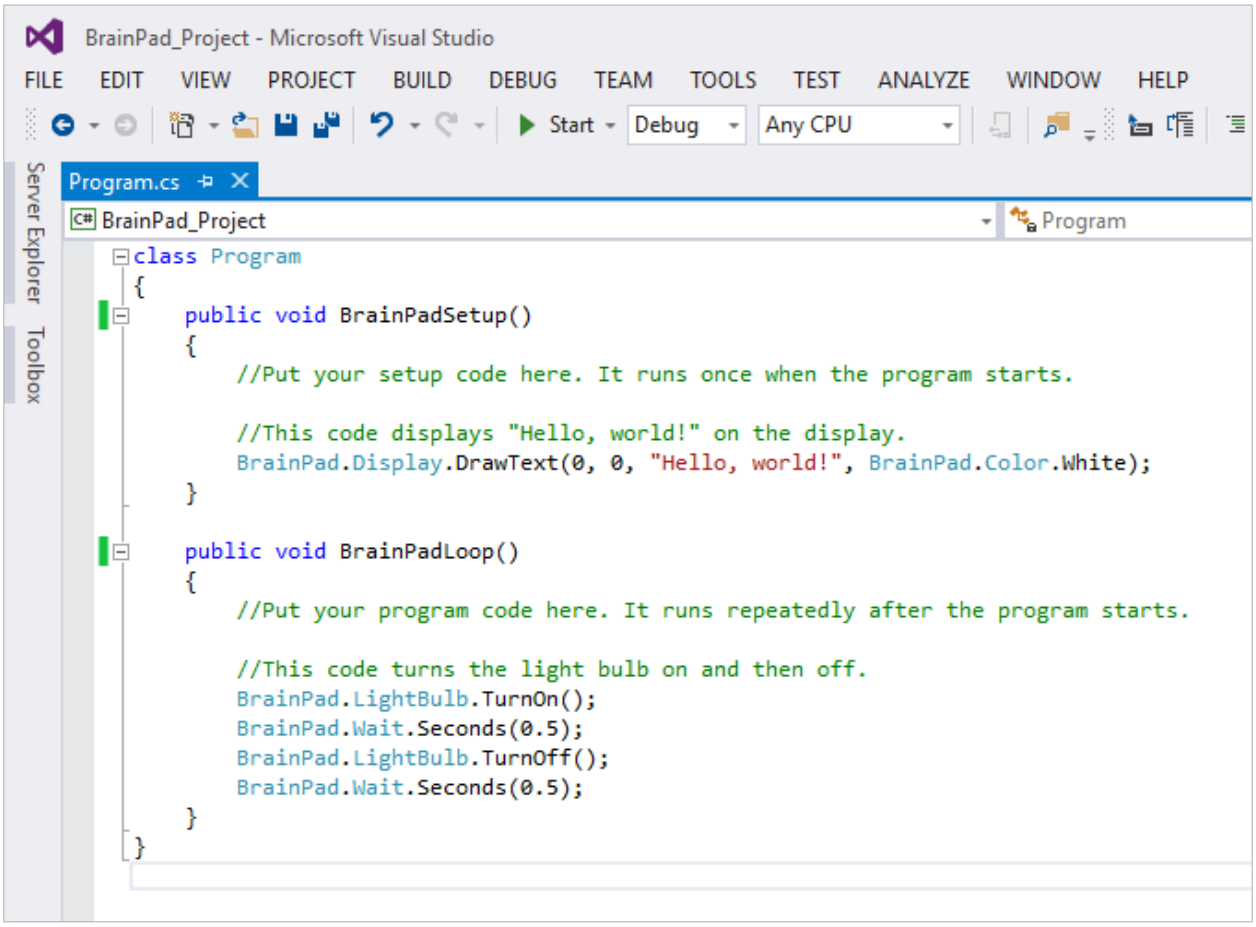


Figure 2 – The Program.cs file and its contents are shown.

This file contains the default code when you first create a project. The `lines in green` that begin with `//` are called comments, and they generally describe what the code does. Take a moment to review them.

Let's run the code to see these comments' descriptions come to life. Connect the BrainPad and press the F5 function key or the Start button (Figure 3).

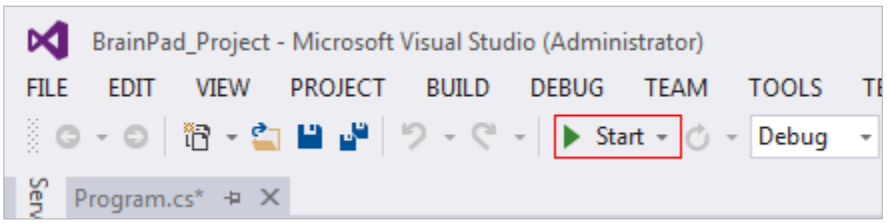


Figure 3 – Press the Start button to run the Program.

Visual Studio will now run the code, a few things will happen and the display will now have the text “Hello, world!” as shown in Figure 4.

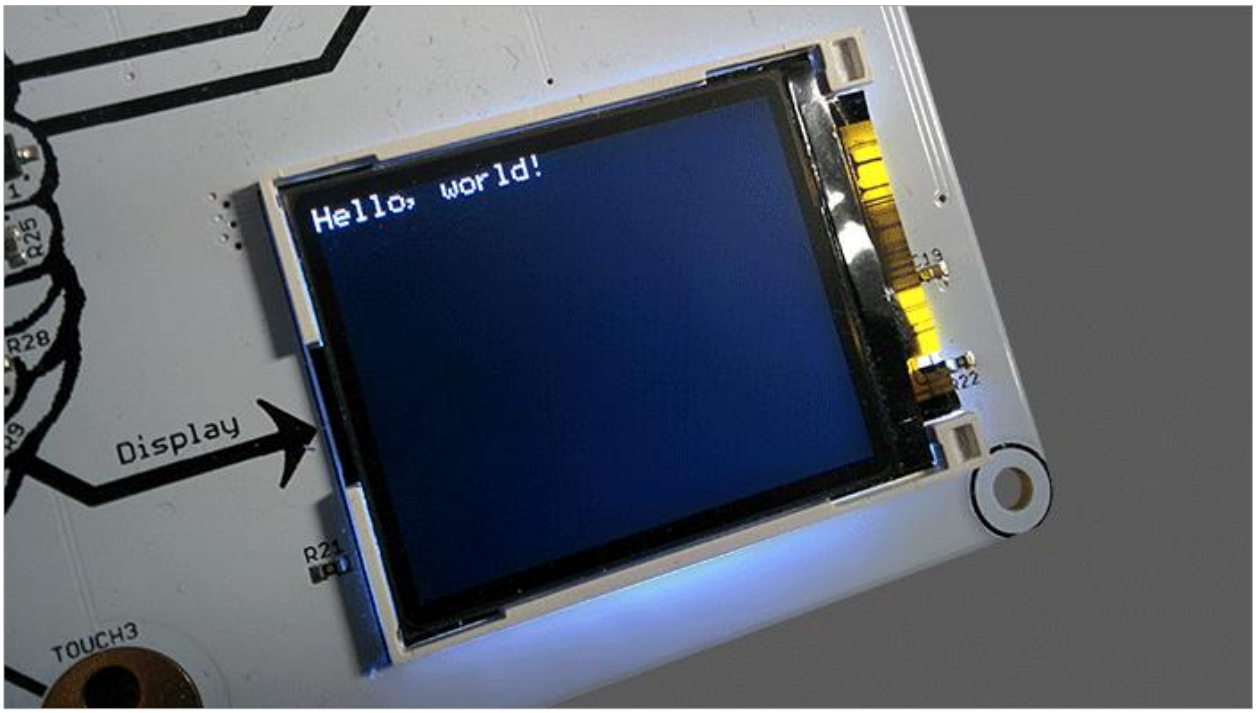


Figure 4 – The display shows “Hello, world!”

What happened exactly? Our application began by calling the `BrainPadSetup()` function. This function is called one time, when the application starts, and is generally used to set things up. In this case, it executed the `BrainPad.Display.DrawText` line to instruct the display to show the text “Hello, world!” After `BrainPadSetup()` was finished, the application called `BrainPadLoop()`. Code placed inside this function is executed in an infinite loop. This is why the code that turns the LED on and then off again never stops.

Exercise

Change the text to make the program print your name on the display.

The BrainPad Object

The world around us is full of objects. Every person, table or circuit board is an object. If we were to ask a person to say “hello,” we'd be asking an object to perform a task. The same concept can be applied to code. Like in the previous application we ran, the BrainPad was asked to control the display and light bulb through the `BrainPad` object.

The `BrainPad` object itself, is a piece of code that is contained in one file. This was developed by GHI Electronics to cover the internals of the BrainPad. This enables students like you, with an easy way to control the BrainPad circuit board.

After typing `BrainPad`, press the period key to get a list of all available options (Figure 5) for this object, like `Display`.

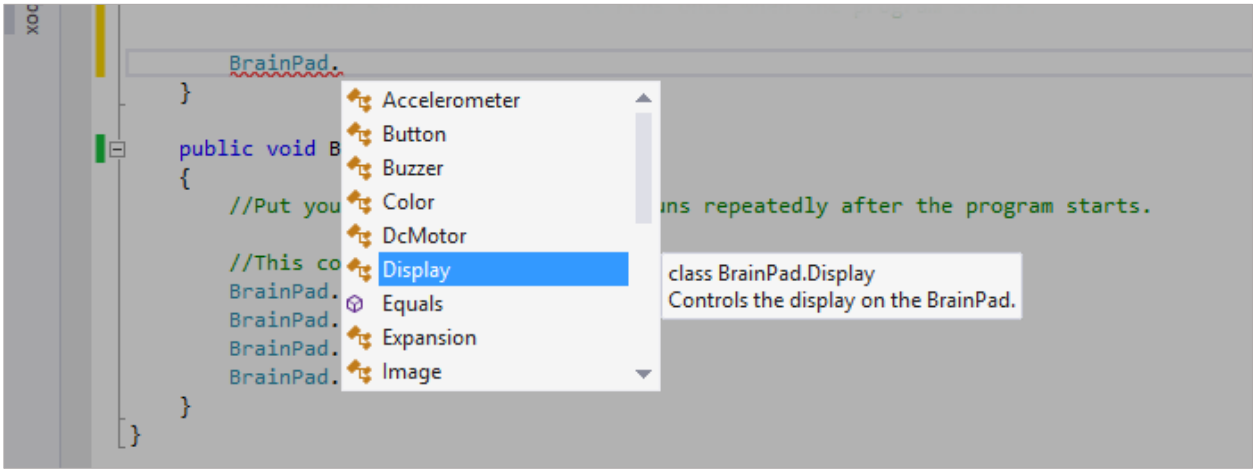


Figure 5 – Using IntelliSense we can see the options offered by the `BrainPad` object.

Now we can use the arrow keys to go up and down in the list to find the `Display`, and then press the period key again to see what options (Figure 6) are the available for the `Display`.

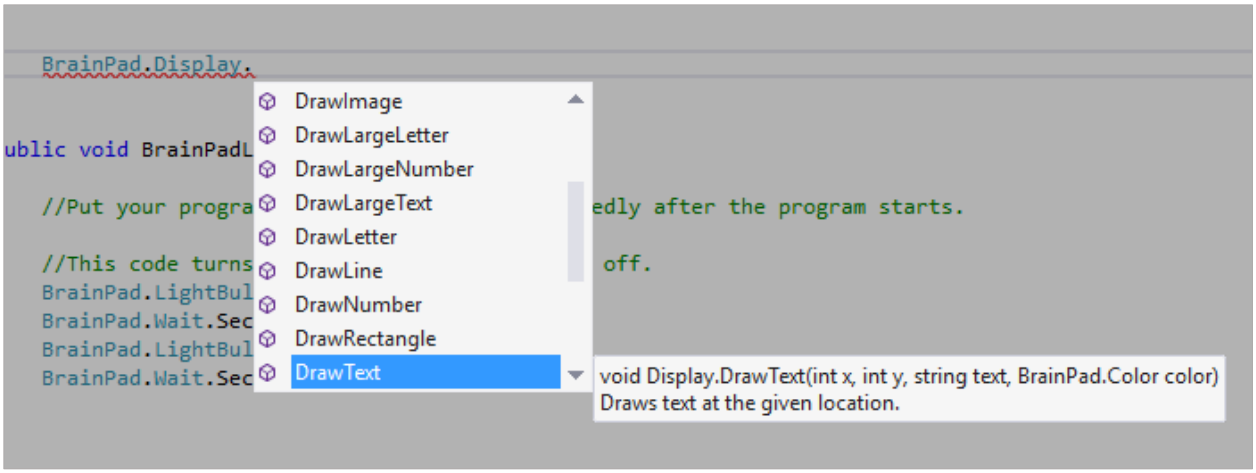


Figure 6 – Using IntelliSense we can see the options offered by the `Display` object.

Visual Studio does a great deal of simplifying programming by automatically listing the available options for each object.

Now that we know how to use the BrainPad object, let's learn about controlling a traffic light.

Traffic Light

Traffic lights are meant to guide traffic. We can test our green traffic light by turning it on and off. Copy and paste the code from Example 1 into your project's `BrainPadSetup()` function.

```
BrainPad.TrafficLight.TurnGreenLightOff();
BrainPad.TrafficLight.TurnGreenLightOn();
BrainPad.TrafficLight.TurnGreenLightOff();
BrainPad.TrafficLight.TurnGreenLightOn();
```

Example 1 – Put this code inside the `BrainPadSetup()` function.

Now, press F5 to run the code and you'll see the green light come on and stay on. This happens because our code is executed faster than we can see.

Problem Solving

Since applications execute extremely fast, we need to slow them down to figure out the problem. This way we can see what is happening or if the results are as expected. We will start by stepping in code to see what the application does at a slow speed.

STEPPING IN CODE

Add a breakpoint at the first line of code inside of `BrainPadSetup()` by moving the cursor to that line and pressing the F9 key as shown in Figure 7.

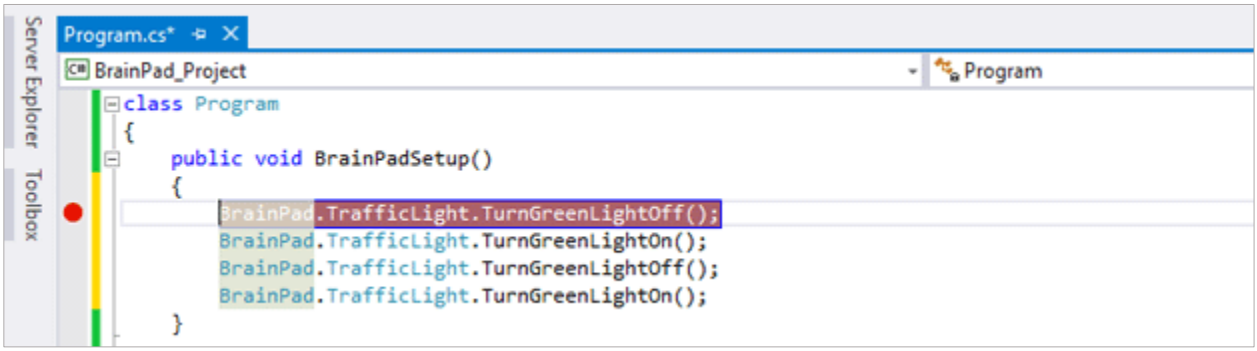


Figure 7 – We've added a breakpoint on line 5.

Press F5 to run the application. The project will be built and deployed but then the execution will stop at the breakpoint as shown in Figure 8.

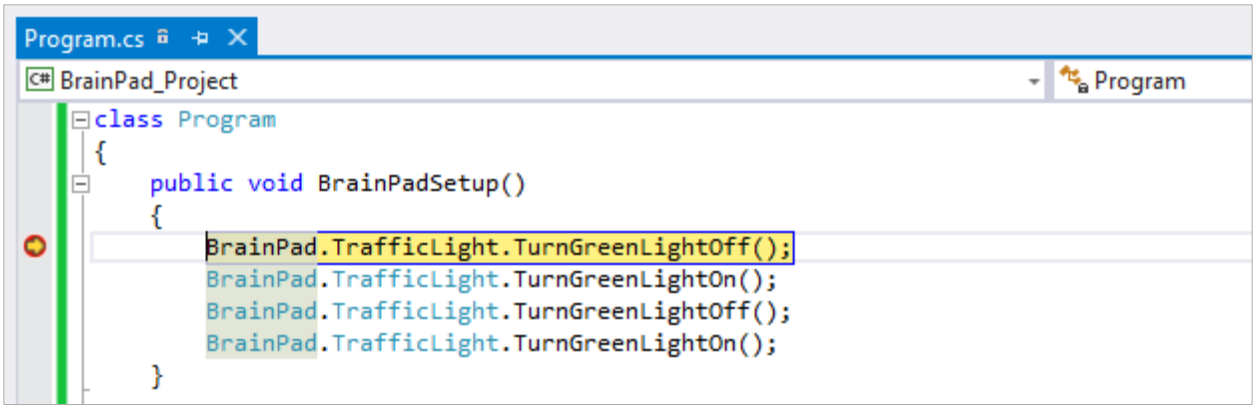


Figure 8 – The program has executed up to the breakpoint.

Press the F10 key to execute the current line, which turns the green light off. Now press the F10 key again, and the green light will turn on. The next steps will continue through the rest of the code. When you reach the closing curly bracket, press F5 to stop.

Now that we know our code works (it's just happening too fast) we can add some delays so we can see the light turn on and then off.

Delays in Code

In order to see what's going on we need to add some delays in the code. This is done by telling the BrainPad to wait between tasks.

```
BrainPad.TrafficLight.TurnGreenLightOff();
BrainPad.Wait.Seconds(0.5);
BrainPad.TrafficLight.TurnGreenLightOn();
BrainPad.Wait.Seconds(0.5);
BrainPad.TrafficLight.TurnGreenLightOff();
BrainPad.Wait.Seconds(0.5);
BrainPad.TrafficLight.TurnGreenLightOn();
```

Example 2 – This code waits half a second (0.5) between turning the green light on and off.

Copy and paste the code from Example 2 into your project's `BrainPadSetup()` function. Run the code and observe the green light again. You should now see the green light turning on and off every half a second.

Exercise

Create a real traffic light using the following logic:

1. Turn the red light off.
2. Turn the yellow light off.
3. Turn the green light on.
4. Wait 5 seconds.
5. Turn the green light off.
6. Turn the yellow light on.
7. Wait 1 seconds.
8. Turn the yellow light off.
9. Turn the red light on.
10. Wait 5 seconds.

While Loop

The previous exercise simulated a traffic light but it only did it once. We could repeat the code over and over to make the traffic light run a few more times but what if we want this to run indefinitely? This is where `while` loops come in handy.

```
while (BrainPad.Looping)
{
    // Your code
}
```

Example 3 – Code inside a while loop will be executed indefinitely.

Code inside a while loop will be executed indefinitely as long as its condition is met. In our case, we'll use a constant value from the BrainPad object called `Looping`. This constant always returns true, which causes the loop's condition to always be met.

```
while (BrainPad.Looping)
{
    BrainPad.TrafficLight.TurnGreenLightOn();
    BrainPad.Wait.Seconds(0.5);
    BrainPad.TrafficLight.TurnGreenLightOff();
    BrainPad.Wait.Seconds(0.5);
}
```

Example 4 – This code will blink a green light on and off for half a second indefinitely.

Let's keep things simple and blink the green traffic light every half a second. Copy and paste the code from Example 4 into your project's `BrainPadSetup()` function. Run the code and observe the green light.

Tip: The `BrainPadLoop()` function does the same thing as our `while` loop behind the scenes, so you can use this function in your application to simplify things.

Understanding The Loop

To further understand what's going on, add a breakpoint by moving the cursor to the first line where we turn the green light on and press F9. This will add a breakpoint and pause the application as soon as the it reaches that line as shown in Figure 9.

```
public void BrainPadSetup()
{
    while (BrainPad.Looping)
    {
        BrainPad.TrafficLight.TurnGreenLightOn();
        BrainPad.Wait.Seconds(0.5);
        BrainPad.TrafficLight.TurnGreenLightOff();
        BrainPad.Wait.Seconds(0.5);
    }
}
```

Figure 9 – Add a breakpoint while the application is running.

Now use F10 to step through the code and when the program reaches the end of the `while` loop it will go back to the beginning. Stop the program execution by pressing the stop button as shown in Figure 10.

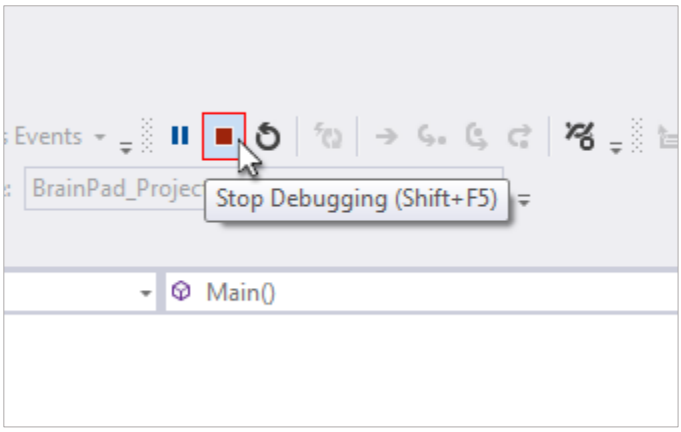


Figure 10 – The stop button will stop the application that's running.

Now, reset the BrainPad and note how the green light is still blinking but this time without the need for Visual Studio. In fact, you can connect the BrainPad to any appropriate power source (such as a phone charger) and the green light will blink without the need for a PC.

Exercise

Make the traffic light simulate a real traffic light indefinitely.

If Statement

An `if` statement (or conditional statement) checks to see if a statement is true or false and then does one of two things depending on the result. Like if the down button is pressed, turn the green light on as shown in Example 5.

```
public void BrainPadSetup()
{
    BrainPad.TrafficLight.TurnGreenLightOff();

    if (BrainPad.Button.IsDownPressed())
    {
        BrainPad.TrafficLight.TurnGreenLightOn();
    }
}
```

Example 5 – Pressing and hold the down button when running this program will turn the green light on.

For the code above to work, you have to be holding the button down at the time the program is deployed. This is because once the `if` statement is reached, the program will continue executing until the program ends. To solve this hard to reach goal, we need to repeatedly check if the button is pressed (Example 6). This is where a `while` loop comes in handy. Let's use the `BrainPadLoop()` function to simplify things per the tip earlier.

```
public void BrainPadSetup()
{
    BrainPad.TrafficLight.TurnGreenLightOff();
}

public void BrainPadLoop()
{
    if (BrainPad.Button.IsDownPressed())
    {
        BrainPad.TrafficLight.TurnGreenLightOn();
    }
}
```

Example 6 – This code loops indefinitely checking if the down button is pressed. If it is pressed, the green light will turn on.

Example 6 first makes sure the green light is off and then it falls into an infinite loop. However, this program still has a bug. Run the program and test it. The green light will be off when the program runs and then once the down button is pressed the green light will turn on. That is all good so far, but when the button is released, the green light never

turns off. Can you guess why? Try stepping in the code to see what happens when the button is pressed and when it is not pressed.

Computers are strict on following orders. In previous examples, the light never turned off because we actually never told the program to turn the green light off. This means we have to tell the program to turn the light on when the button is pressed and we also have to tell it to turn the light off when the button is not pressed as shown in Example 7.

```
public void BrainPadSetup()
{
    BrainPad.TrafficLight.TurnGreenLightOff();
}

public void BrainPadLoop()
{
    if (BrainPad.Button.IsDownPressed())
    {
        BrainPad.TrafficLight.TurnGreenLightOn();
    }
    if (!BrainPad.Button.IsDownPressed())
    {
        BrainPad.TrafficLight.TurnGreenLightOff();
    }
}
```

Example 7 – This code will turn the green light on when the down button is pressed, and off when the down button is not pressed.

The `!` symbol represents “not” or false in an `if` statement. While the BrainPad doesn’t have a method called `BrainPad.Button.IsDownNotPressed()` we can continue to use `BrainPad.Button.IsDownPressed()` but with a `!` before it to represent if the button is not pressed. Now when you run it, not pressing the button will turn/keep the light off.

Exercise

Turn the green light on when the up button is pressed. Turn the green light off when the down button is pressed. Turn the red and the green lights on when the left button is pressed.

The Else Statement

The `else` statement is always used with the `if` statement and has a very useful purpose that would be perfect for the last example. Previously, we needed to check if the button is pressed and if the button is not pressed. We can simplify this by catching when an `if` statement isn't true using `else` as shown in Example 8.

```
public void BrainPadSetup()
{
    BrainPad.TrafficLight.TurnGreenLightOff();
}

public void BrainPadLoop()
{
    if (BrainPad.Button.IsDownPressed())
    {
        BrainPad.TrafficLight.TurnGreenLightOn();
    }
    else
    {
        BrainPad.TrafficLight.TurnGreenLightOff();
    }
}
```

Example 8 – This code uses an `else` statement to check if a button is pressed or not to turn a green light on and off.

Or & And Operators

The most used logical operators are the `||` (or) and `&&` (and) operators. These are typically used within the `if` statement. In Example 9, we want the buzzer to generate a 5,000 Hz sound when both the up and down buttons are pressed. This is accomplished by using an `&&` operator between `IsUpPressed()` and `IsDownPressed()`.

```
public void BrainPadLoop()

    if (BrainPad.Button.IsUpPressed() && BrainPad.Button.IsDownPressed())
    {
        BrainPad.Buzzer.PlayFrequency(5000);
    }
    else
    {
        BrainPad.Buzzer.Stop();
    }
}
```

Example 9 – This code checks if both up and down are pressed, if so it plays a high pitch sound.

White Space

Spaces and lines are used to make our code more readable. C# programs completely ignore white space as shown in Example 10.

```
class Program{public void
BrainPadSetup(){BrainPad.TrafficLight.TurnGreenLightOff();}public void
BrainPadLoop(){if (BrainPad.Button.IsUpPressed() &&
BrainPad.Button.IsDownPressed()){BrainPad.Buzzer.PlayFrequency(5000);}else{BrainPad.Bu
zzer.Stop();}}}
```

Example 10 – This code shows how white space is ignored when the program runs. White space simply makes code readable by humans.

The above program is perfectly valid.

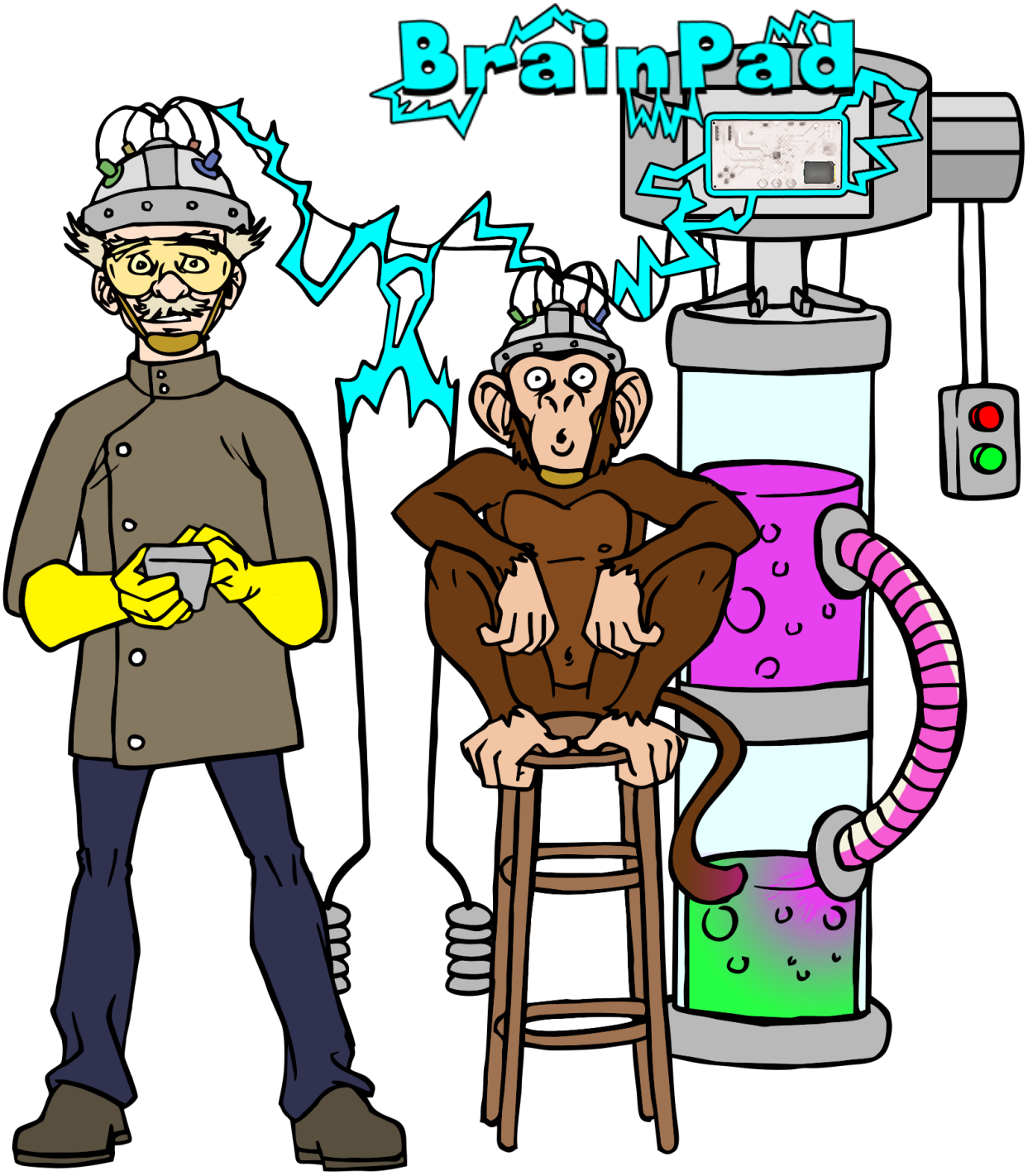
Exercise

Turn the green light on and play the buzzer at 5,000 Hz whenever the up or down button is pressed. Pressing both or either one will have the same effect.

Extra Credit

If the up and down buttons are pressed together, turn the red light on. If only the up or down button is pressed, then play the buzzer at 5,000 Hz. You must have two `if` statements in your code.

Tip: Use nested statements such as an `if` statement inside another `if` statement.



GHIElectronics.com/support/brainpad