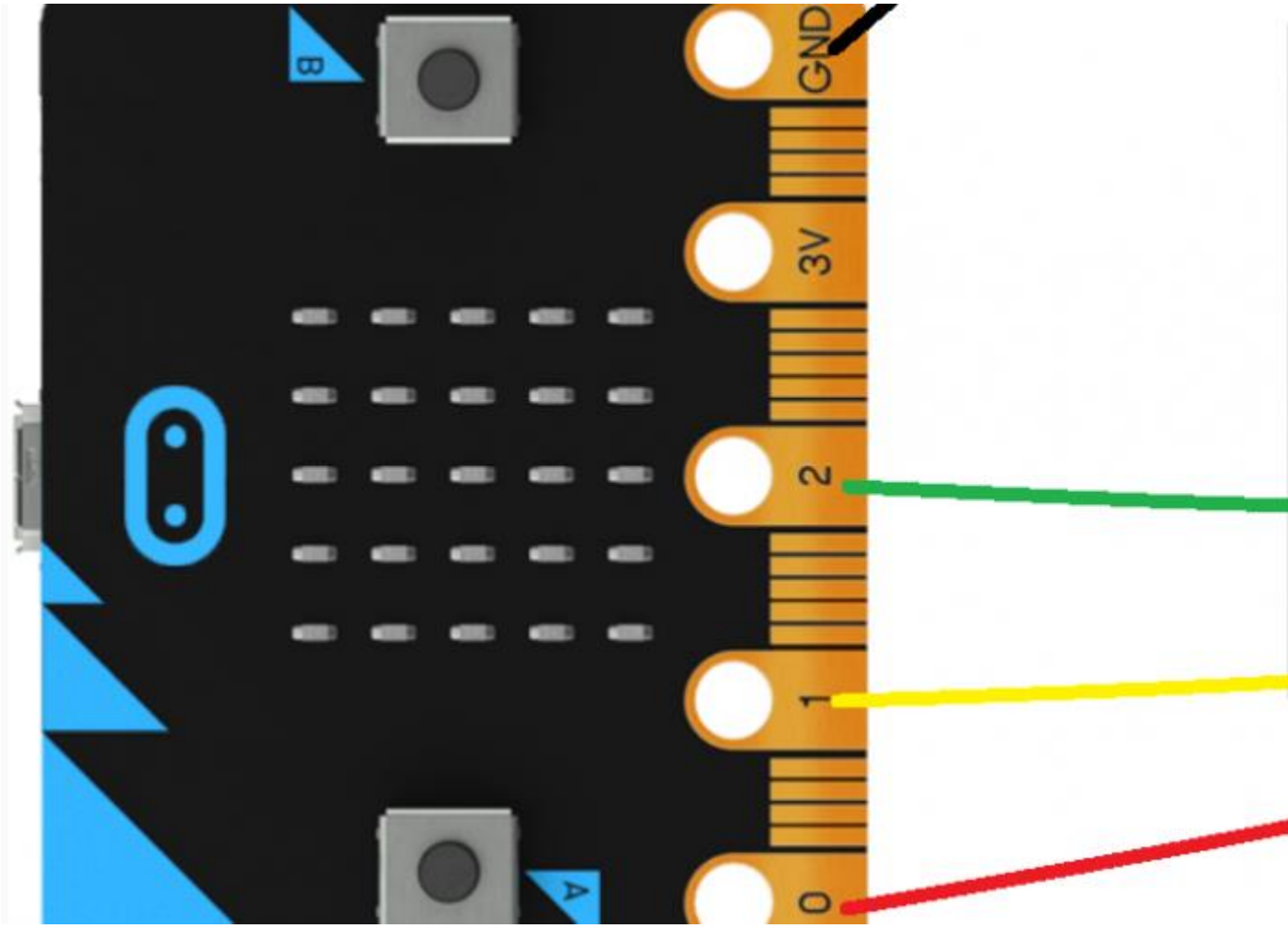


A Selection of micro:bit Projects

2 Aug, 2016 *in microbit* *tagged Accelerometer / BBC Microbit / LED* *by Ivan Holland*

Some time ago, we had a look at the BBC micro:bit (see http://www.makerspace-uk.co.uk/microkit-a-low-cost-breadboarding-solution/ and http://www.makerspace-uk.co.uk/bbc-microbit/). Since then, we've been spending a little more time exploring some of the things that you can do with it.

A lot of these projects involve connecting the micro:bit to other bits and pieces; such as buzzers, or LEDs. However, we start off with a really simple project which just uses the micro:bit on its own.

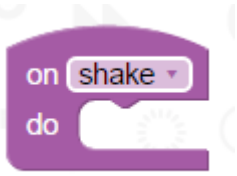
micro:bit Dice (or die)

Our micro:bit dice project uses the onboard LED matrix and accelerometer to simulate a die. Simply shake the micro:bit, and it will generate a random dice throw, and display it on the micro:bit LED matrix. To create the code for this project, head over to the micro:bit web site, and select the 'Create Code' link (https://www.microbit.co.uk/create-code). From this page

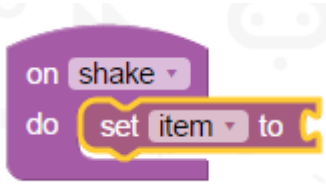
you'll need to create a new project using the Block Editor. If you've not used the micro:bit block editor before, there are tutorials available at the BBC micro:bit web site to help you on your way.

When you create your new project, you'll be presented with a blank canvas on which to create your program. Select the blocks you want to use by dragging them into place on the canvas.

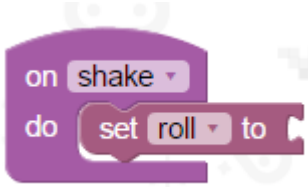
For the micro:bit dice project, the first block that you will need is the 'on shake' block, which can be found under the 'Input' tab. So, click on the 'Input' tab, and drag the 'on shake' block on to the canvas.



The next block you'll need is the 'set' block, to allow us to set up a variable called 'roll'. This block is under the 'Variables' tab; so click on 'Variables' and drag a 'set' block into place within the 'on shake' block.

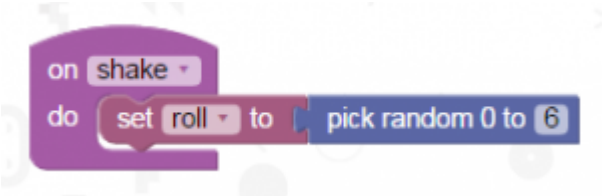


You're going to use a variable called 'roll', so to rename your variable (currently called 'item'), click on the word 'item' and select 'Rename variable…' from the menu that appears. In the prompt that appears, replace the word 'item' with 'roll', and press OK to rename the variable.



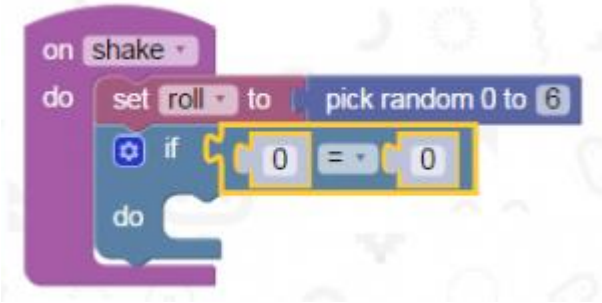
Next, you will need to set the value of the variable 'roll' to have a random value. For this, you'll need to select the 'Math' tab, and drag a 'pick random' block into place next to the 'set' block.

For a dice roll, the random number chosen should be between 0 and 6, so you will need to change the number in the 'pick random' block to have a value of 6 instead of 4.

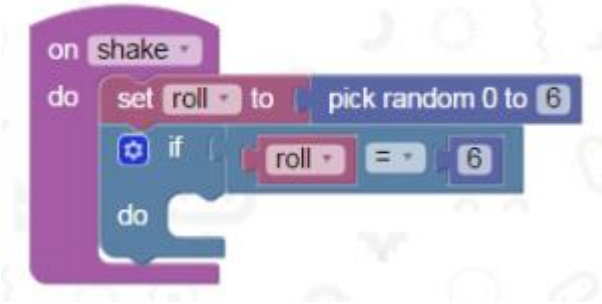


Having picked the random number, you will then need to display the number somehow. For this you will use the 5×5 LED matrix. However, the pattern displayed by the LEDs will depend on the value of the variable ‘roll’, so it will be necessary to introduce an ‘if’ block into our program. To do this, select the ‘Logic’ tab, and drag an ‘if’ block into place within the program as shown in the diagram below.

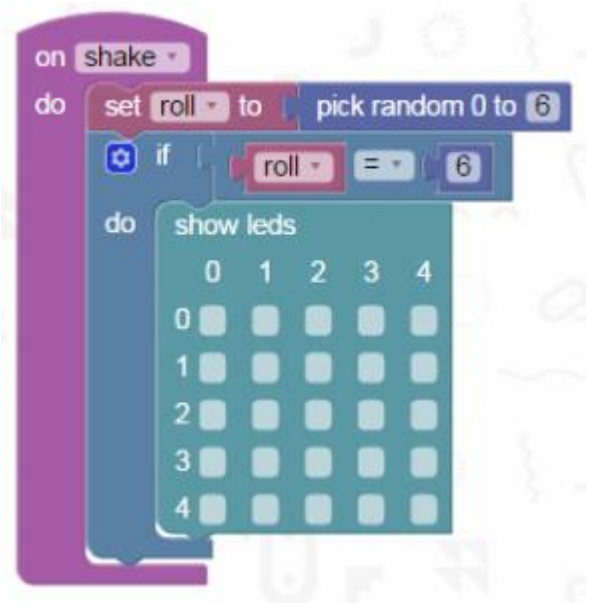
The ‘if’ statement will need to check the value of the variable ‘roll’, so the next step will be to drag a code block that will let you compare 2 numbers. This comparison (‘=’ or equality) code block is under the ‘Logic’ tab. Drag this ‘=’ code block into place as shown below.



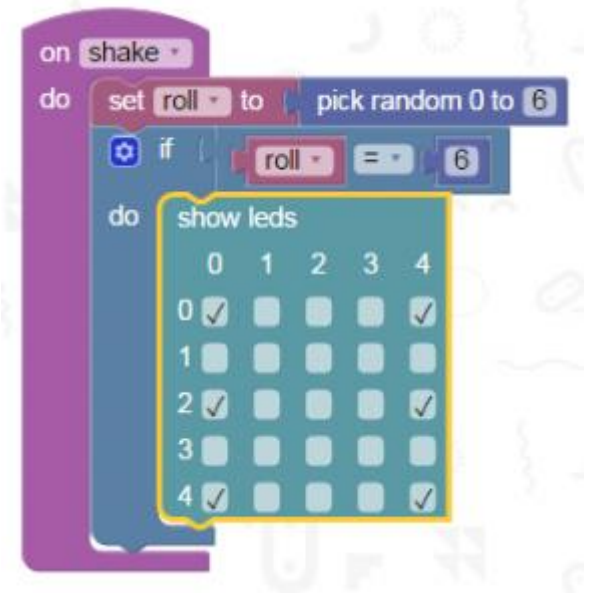
Having done this, you will need to replace the ‘0’ blocks with the correct values. You need to check to see whether the value of the variable ‘roll’ is equal to a number (for example 6), so replace the first ‘0’ with the variable ‘roll’ (you will find this under the ‘Variables’ tab), and replace the second ‘0’ with the number ‘6’; as below.



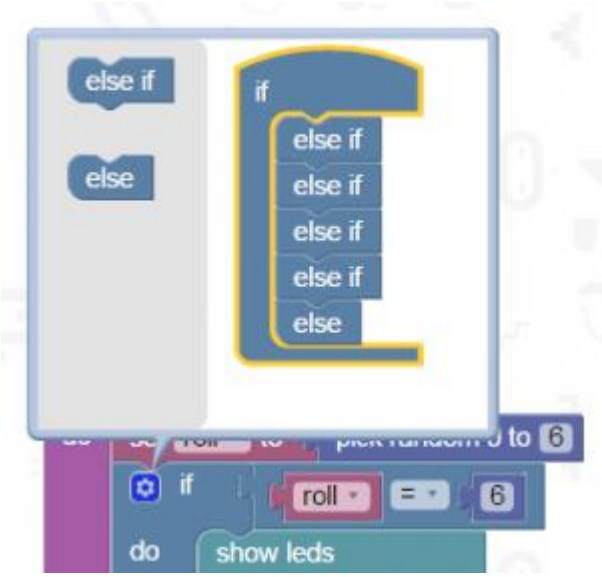
Next you need to define the LED pattern that will be displayed for a roll value of 6. LED patterns are set up using the ‘show LEDs’ block under the ‘Basic’ tab, so drag one of these blocks into place next to the word ‘do’ in the ‘if’ block.



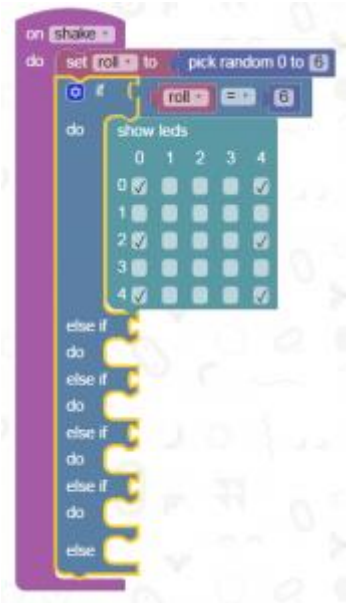
Then draw the pattern for a dice throw of 6 by selecting which of the LEDs should be lit up for this dice throw.



As it stands, the dice will only indicate a throw of 6, so the next step is to check for other values of the variable ‘roll’. This is done by means of ‘else if’ and ‘else’ statements. To create these statements, you will need to click on the settings icon in the ‘if’ block, then drag a number of ‘else if’ and ‘else’ blocks into place within the ‘if’ block. You will need to create 4 ‘else if’ blocks and a single ‘else’ block, as below.



Having done this, click on the settings icon again, and the ‘if’ block settings will disappear, leaving you with the gaps required for dragging other ‘=’ and ‘show LEDs’ blocks into place.



Finally, drag in the comparison (‘=’) blocks and ‘show LEDs’ blocks for the various other possible values of the variable ‘roll’. You only actually need to check 4 other values (5, 4, 3 and 2), as if it is none of these values, then it must have a value of 1 (you’ve already checked for the value of 6).

The diagram below shows what your completed code for the micro:bit dice (or die) should look like.

Dice Blocks

Having completed the code for the micro:bit dice, you can try running it in the on-screen emulator by pressing the ‘Run’ button at the top of the web page. This will allow you to check out your code before downloading it to your micro:bit.

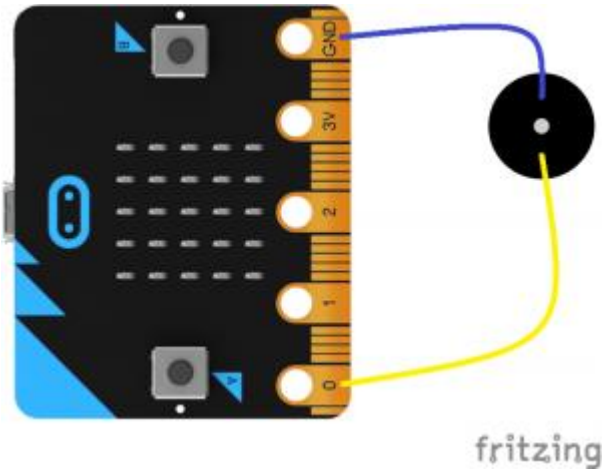
Once you’re happy that the code is working correctly, try downloading it to your micro:bit. To do this, you’ll need to ensure that your micro:bit is connected to your PC using the USB cable. The micro:bit should show up as a USB drive called ‘MICROBIT’. Having done this, select the ‘Compile’ option at the top of the web page to generate the program code which needs to be downloaded to the micro:bit. Depending on which browser you are using you may have the option to save the generated file (it should have a file extension of ‘.hex’) straight to the micro:bit. If your browser does not support this, then you’ll need to open the folder in which the .hex file was generated, and then copy the file to the micro:bit drive.

Once the .hex file has been copied to the micro:bit it should start running; so try shaking your micro:bit, and see what happens. If the program runs as expected, you can try disconnecting the micro:bit from the USB cable, and connect a separate battery power supply to power your micro:bit die.

micro:bit Buzzer

Our next project gets you started with connecting some simple electronic components to your micro:bit. The first electronic component we’re going to try out is our buzzer. For this project you’ll need the micro:bit (and its USB cable or course), a buzzer, and 2 crocodile clip leads. It doesn’t matter what colour leads you use, and it doesn’t matter which terminals on the buzzer they are connected to. Connect one of the leads from the micro:bit terminal marked ‘GND’ (ground) to one of the terminals on the buzzer. Connect the second lead from the micro:bit terminal marked ‘0’ to the other terminal on the buzzer.

You should have a circuit looking something like the diagram below.



Buzzer Circuit

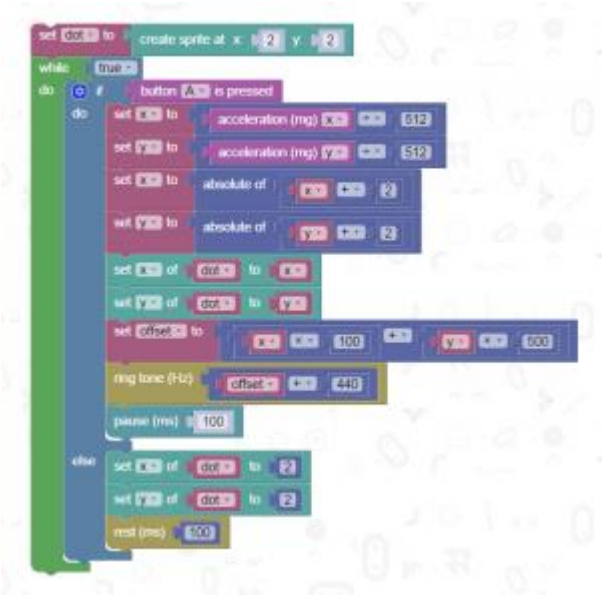
To check that your buzzer circuit works, you will need to create the following code in the micro:bit block editor, and download it to the micro:bit. This time, you won't be able to check the program out first using the on-screen emulator, but hopefully this shouldn't be necessary anyway, as the program is a fairly simple one. So, having created the code, download it to your micro:bit and try it out. Hopefully when you press the 'A' button on the micro:bit, you should hear a two tone buzz from your buzzer.

Buzzer Blocks

micro:bit Theremin

If you've got your buzzer working successfully, why not take things a stage further and have a little fun with it by making a simple and fun musical instrument. The instrument you're going to have a go at creating is something called a Theremin, and it doesn't require any modifications to the buzzer circuit that you've just created. If you've not come across the Theremin before, have a look at the following link which explains a little about what it is and how it works:-

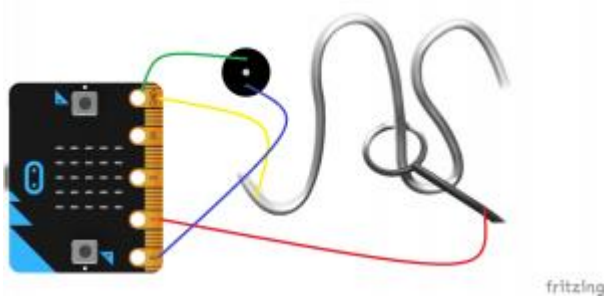
The code for the micro:bit Theremin was originally written by David Whale, but we've adapted it slightly to allow breaks (rests) in the music created by this somewhat bizarre musical instrument. In our modified version of the Theremin, the buzzer will only play a sound while the 'A' button is being pressed. Our modified Theremin code is shown below. Try creating this code and downloading it to your micro:bit (you may need to hunt around a little to find the blocks that you need, but they shouldn't be too difficult to find). Having done this, if you move the micro:bit around while pressing the 'A' button, it should play a bit of a tune.



Theremin Blocks

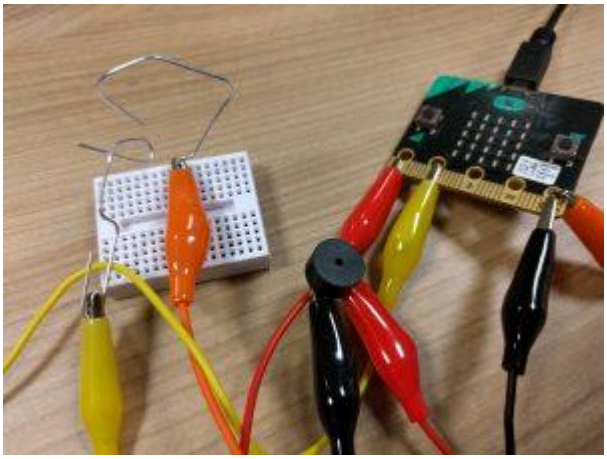
micro:bit Miniature Steady Hand Game

Our next little project still uses the buzzer circuit that you've previously used, but now requires you to connect a little more circuitry. You're going to have a go at creating a micro:bit miniature steady hand game, for which you'll need the buzzer circuit together with 2 more crocodile clip leads, a breadboard and a couple of paper clips. The paper clips will be used to create your steady hand game course and a wand to use with it; and you'll need to bend these into shape to create the course and wand that you want for your version of the game. The breadboard is used as a stand to hold the game course paper clip. Both the course and the wand will need to be connected to terminals on the micro:bit using the 2 crocodile clip leads, as shown in the circuit diagram below.

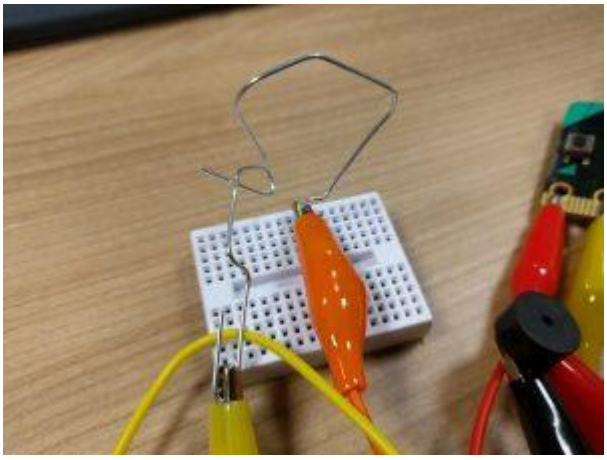


Steady Hand Game Circuit

The following pictures show our setup.

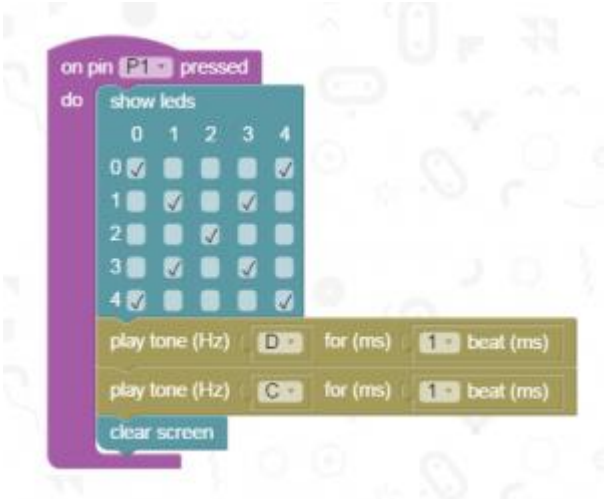


Steady Hand Game



Steady Hand Game Track and Wand

The code for this game is fairly straightforward, as all it needs to do is to get the buzzer to make a sound, and display a cross on the micro:bit's LED matrix when the wand comes into contact with the game track. The code blocks to do this are shown below.



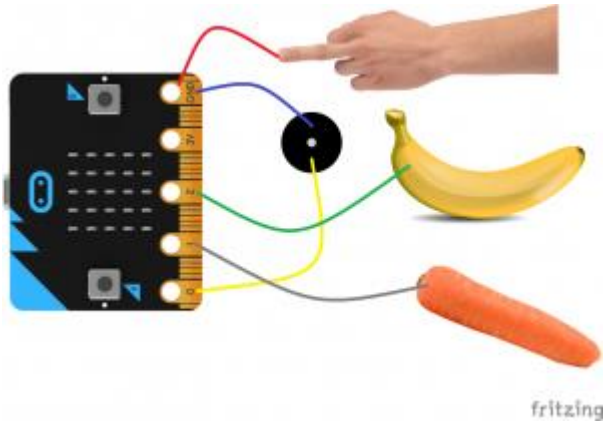
Steady Hand Game Blocks

You could try creating a larger version of the game using an old wire coat hanger and use a lump of plasticine as a stand for the course.

micro:bit Fruit Keyboard

Our next project also uses the buzzer circuit. In addition, you'll need the remaining crocodile clip wires, and a couple of pieces of fruit or vegetable, which need to be connected up as shown in the diagram below.

To make this project work, you need to be in contact with the wire that is connected to the 'GND' (ground) connector on the micro:bit. You can do this by simply touching the metal end of the crocodile clip. When you touch one of the pieces of fruit or vegetable, the micro:bit is able to detect this, and will play a tone through the buzzer. You don't have to use fruit or vegetable, as this will also work with things like jelly babies, wiggly worm sweets, or anything else which will conduct very small amounts of electric current which the micro:bit is able to detect.



Fruit Keyboard Circuit

The code block for our fruit keyboard is shown below. Although it's unlikely you'll be able to play much of a tune with just 2 notes, you could try experimenting with using different notes.

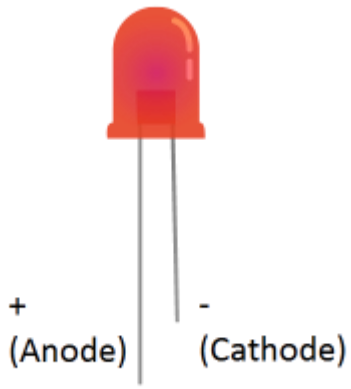
Fruit Keyboard Blocks

micro:bit Traffic Lights

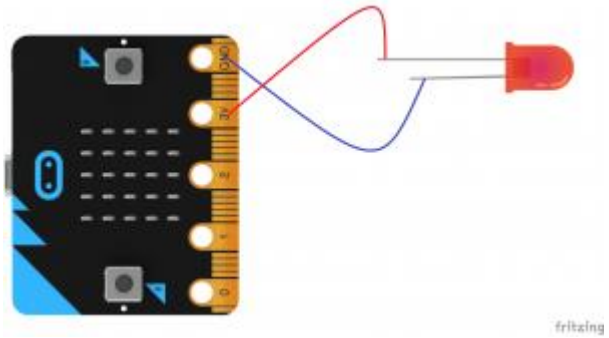
For our final project, we're going to make a simple set of traffic lights. This project makes use of the micro:bit, breadboard, 4 crocodile clip wires, and 3 LEDs.

Before putting the circuit together, it's worth taking a quick look at the LED. If you examine one of the LEDs closely, you should notice that one of the leads is slightly longer than the other. This longer lead is generally referred to as the anode, which is the positive terminal of

the LED. The other lead is the cathode, or negative terminal. The LED will only light up if it is connected the correct way round.

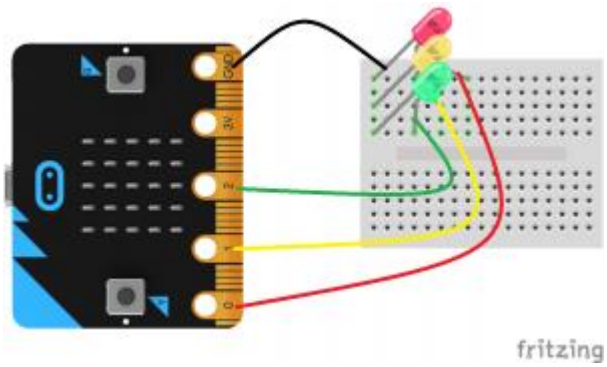


You can try this out by simply connecting the LED to your micro:bit as shown below, using 2 of the crocodile clip leads. The micro:bit will need to be connected to your PC or powered using the battery pack. Provided that the LED is connected the correct way round, it should light up.



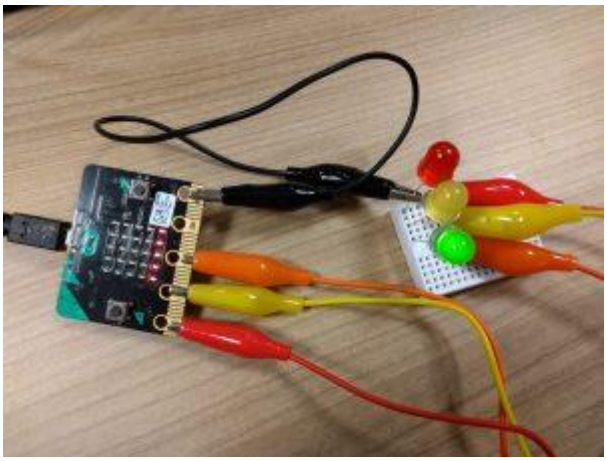
micro:bit and LED

For our traffic light circuit, you'll need to push the pins of the 3 LEDs into your breadboard, being careful to ensure that the shorter pins of each of the LEDs (the cathodes) are all in line down one of the strips on one side of the breadboard (as shown in the circuit diagram below). The anodes must all be in different breadboard strips. One of the cathode (short) LED pins needs to be connected to the 'GND' (ground) connector of the micro:bit, using one of the crocodile clip wires. Because the LED pins are all in the same strip of the breadboard, this means that they are effectively connected together. This arrangement is generally referred to as a 'common cathode' connection. The longer pins (the anodes) of the 3 LEDs should be connected to connectors '0', '1', and '2' using the other 3 crocodile clip leads. You'll need to be careful that the ends of the crocodile clips on the LEDs do not come into contact with each other.



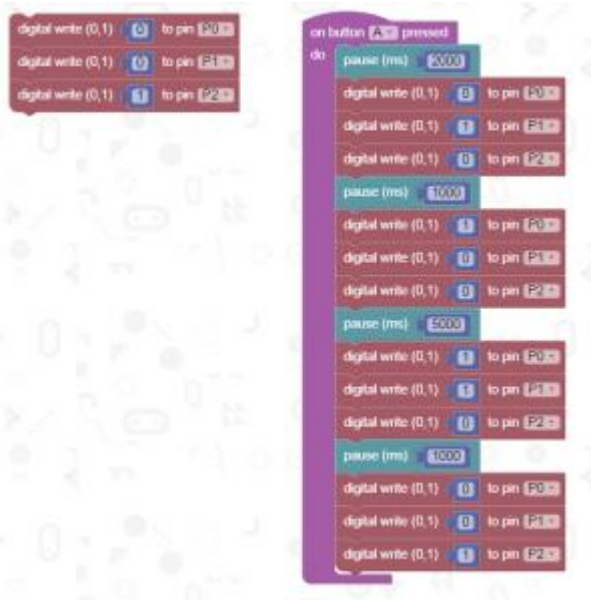
Traffic Lights Circuit

The image below shows our completed micro:bit traffic lights circuit.



Traffic Lights

To make the traffic lights work, you'll need to create the following arrangement of code blocks, and download the code to your micro:bit in the same way as you've done for your previous projects. After you download the code to the micro:bit, the green LED should light up. When you press the 'A' button, the traffic lights should change through a normal sequence, ending up with the green LED on.



Traffic Lights Blocks