BBC micro:bit
Lighting The Way

Introduction

'Neopixels' is a term used by the company Adafruit to describe chainable RGB LEDs that come with a constant current driver. You can control a decent number of neopixels using a single GPIO pin and they tend to be wonderfully bright. Since we are running these from battery power, rather than power from the micro:bit, we can afford to run them a little brighter than we might normally do.

There are 12 neopixels on the Bit:Bot, 6 of them on each side. They are helpfully numbered on the PCB from 0 to 11, no excuses for not lighting the correct ones.

Programming

The following program demonstrates the basics of working with neopixels on the micro:bit.

```
from microbit import *
import neopixel

# Initialise neopixels
npix = neopixel.NeoPixel(pin13, 12)

# Define some colours
red = (255,0,0)
green = (0,255,0)
blue = (0,0,255)


# Make pixel 0 red
npix[0] = red
# Make pixel 6 green
npix[6] = green
# Make last pixel blue
npix[-1] = blue

# Cause the colour changes to be shown
npix.show()

sleep(5000)
# Clear the neopixels
npix.clear()
```

The line that initialises the strip of pixels states the pin and the number of neopixels in the chain. For the Bit:Bot, this is pin 13 and 12 neopixels. Once this has been done, the variable we used becomes a list of tuples that can be accessed in the normal way. When we define colours, we need to remember to call the show() method in order to see them on the pixels. We can also clear all of the pixels with a single statement.

If you want to turn off a single pixel, rather than all of them, set its colour to (0,0,0).

Helper Functions

Lighting Groups Of Pixels

The functions in this first set are pretty basic. I use these as indicators when trying to get the robot to respond to its sensors.

```
from microbit import *
import neopixel

# Initialise neopixels
npix = neopixel.NeoPixel(pin13, 12)

# Define some colours
red = (255,0,0)
green = (0,255,0)
blue = (0,0,255)
nocol = (0,0,0)

# light all neopixels with given colour
def LightAll(col):
    for pix in range(0, len(npix)):
        npix[pix] = col
    npix.show()
    return

# light neopixels on the left fin
def LightLeft(col):
    for pix in range(0, 6):
        npix[pix] = col
    npix.show()
    return

# light neopixels on the right fin
def LightRight(col):
    for pix in range(6, 12):
        npix[pix] = col
    npix.show()
    return
```


```
while True:
    # All red
    LightAll(red)
    sleep(2000)
    # Clear all
    npix.clear()
    sleep(1000)
    # Light left pixels green
    LightLeft(green)
    # Light right pixels blue
    LightRight(blue)
    sleep(2000)
    # Clear left
    LightLeft(nocol)
    sleep(1000)
    # Clear right
    LightRight(nocol)
    sleep(1000)
```

Colour Wipe

A colour wipe is when you light each pixel one at a time with a colour. A small delay is used between each lighting up neopixel so that the colour seems to be wiped across the chain. There are 3 functions to do this. The first wipes the colour according to the numerical order of the neopixels. The other two wipe the colour from the back to the front of the robot and vice versa.

```
from microbit import *
import neopixel

# Initialise neopixels
npix = neopixel.NeoPixel(pin13, 12)

# Define some colours
red = (255,0,0)
green = (0,255,0)
blue = (0,0,255)
nocol = (0,0,0)

# Colour wipe functions
def Wipe(col, delay):
    for pix in range(0, len(npix)):
        npix[pix] = col
        npix.show()
        sleep(delay)
    return

def WipeForward(col, delay):
    for pix in range(0, 6):
        npix[pix] = col
        npix[pix+6] = col
        npix.show()
        sleep(delay)
    return

def WipeBackward(col, delay):
    for pix in range(11, 5,-1):
        npix[pix] = col
        npix[pix-6] = col
        npix.show()
        sleep(delay)
    return

while True:
    Wipe(red, 100)
    sleep(1000)
    Wipe(green, 100)
    sleep(1000)
    Wipe(blue, 100)
    sleep(1000)
    WipeForward(red, 100)
    sleep(1000)
    WipeForward(green, 100)
    sleep(1000)
    WipeForward(blue, 100)
    sleep(1000)
    WipeBackward(red, 100)
    sleep(1000)
    WipeBackward(green, 100)
    sleep(1000)
    WipeBackward(blue, 100)
    sleep(1000)
```

Rainbows

Some functions to use colours from a colour wheel distributed across the neopixels.

```
from microbit import *
import math
import neopixel

# Initialise neopixels
npix = neopixel.NeoPixel(pin13, 12)

def LightAll(col):
    for pix in range(0, len(npix)):
        npix[pix] = col
    npix.show()
```

```
def hsv_to_rgb(h, s, v):
    if s == 0.0:
        return (v, v, v)
    i = int(h*6.0) # XXX assume int() truncates!
    f = (h*6.0) - i
    p = v*(1.0 - s)
    q = v*(1.0 - s*f)
    t = v*(1.0 - s*(1.0-f))
    i = i%6
    if i == 0:
        return (v, t, p)
    if i == 1:
        return (q, v, p)
    if i == 2:
        return (p, v, t)
    if i == 3:
        return (p, q, v)
    if i == 4:
        return (t, p, v)
    if i == 5:
        return (v, p, q)

def MakeColour(h):
    hsv = hsv_to_rgb(h,1,0.5)
    r,g,b = hsv
    return (math.floor(r*255),math.floor(g*255),math.floor(b*255))


def Rainbow(delay):
    for pix in range(0, len(npix)):
        npix[pix] = MakeColour(pix/(len(npix)-1))
        npix.show()
        sleep(delay)

def RainbowCycle(delay):
    l = len(npix)
    colours = [MakeColour(i/(l-1)) for i in range(0, l)]
    for i in range(0, len(npix)):
        x = [colours[(i + j) % l] for j in range(0,l)]
        for pix in range(0,l):
            npix[pix] = x[pix]
        npix.show()
        sleep(delay)



while True:
    LightAll((128,0,0))
    sleep(1000)
    Rainbow(1000)
    sleep(1000)
    LightAll((128,0,0))
    sleep(1000)
    Rainbow(50)
    for i in range(0,20):
        RainbowCycle(150)
    LightAll((128,0,0))
    sleep(1000)
    for i in range(0,1000):
        RainbowCycle(20)
    sleep(1000)
```

Challenges

There's a fair bit of code on this page and it only scratches the surface of the lovely and varied effects you can make with Neopixels. It is well worth experimenting and exploring this at length.

1. An obvious starting point is in synchronising lights and movement. Using the functions on this page, you can get different lights to indicate the direction in which you are driving the motors. You could light up a different number of lights on each fin depending on how quicky you are driving the motor on that side. You could use colour to indicate the direction of the motor too.
2. You can fade in and out of colours by using a loop to change the amount of red, green or blue in small steps. Work out how to write a function that fades from no colour to full brightness red. Work out how to fade back to nothing again. Then develop a function that fades from no colour to any colour you choose. This is harder since you have to go up/down in different amounts. Finally make a function that fades from one colour to another. Now you have a robot car that can perform some basic mood lamp functions.
3. Set all of the pixels to a single colour, not one that gives full brightness on any channel. Let's say you start with a light red. Now make one pixel at a time in the strip show full brightness red. Muck around with the intervals you use so that it looks like the colour is running around the strip of lights.
4. Work out how to make a rainbow pattern across all of the lights, starting at one end of the colour spectrum and ending at the other. It's quite tricky to work out a general rule for doing this. You might find it easier to work out the 12 colours you need first and store them in a list of tuples.
5. Similar to the last challenge. Make a single pixel twinkle. Quickly brighten one pixel in a brighter version of the same colour used on the other pixels. Work out how to vary which pixel you twinkle and create a reusable function.
6. Using different shades of red, you can make a heartbest effect. This looks just as good if you use a different colour. What you are looking for is a range of shades of the same colour with the edges of the strip being dimmer than the centre. Make the pixels glow brighter by differing amounts depending on their location in the strip.
7. When you have written a decent number of functions and code snippets for the neopixels, it's worth collating them all into a single example program or into a reference you make for yourself. As you explore more of the functionality of the Bit:Bot, you will be able to pick and choose from your list more easily when writing new programs.