BBC micro:bit
Bit:Commander - Simon Game

Introduction

I wanted to try to write a Blocks version of the Simon game that I had previously written in MicroPython. I don't really enjoy programming with blocks very much but thought the challenge worthwhile.

I was keen to make sure that I didn't use any JavaScript that could not be represented as blocks.

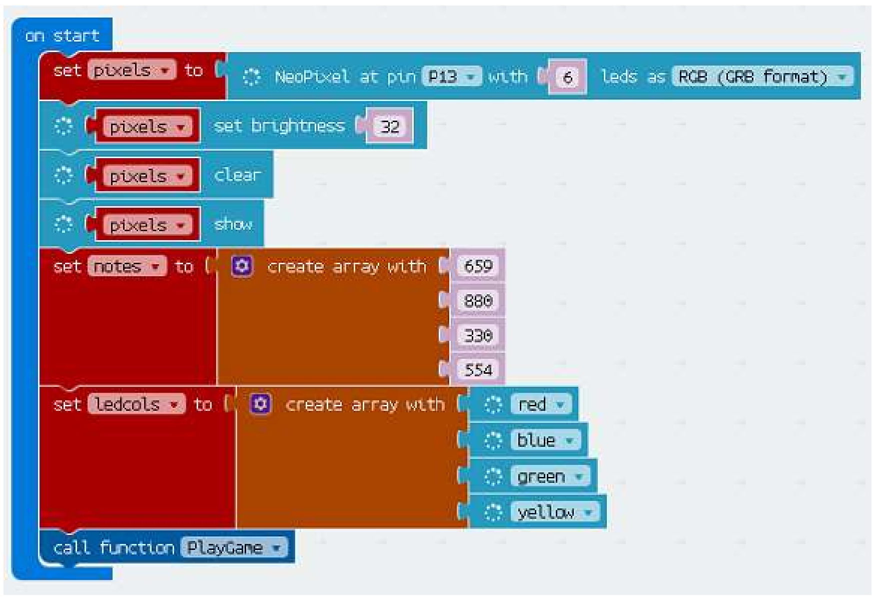
Programming

The Neopixel library is needed for this program.

I wrote this as a series of functions. Things can get out of hand very quickly with blocks.

Starting

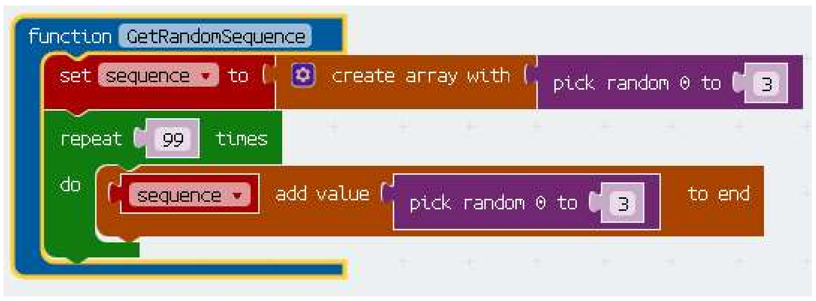
When the micro:bit first starts, we set up the Neopixels and make sure that they are switched off. We make two arrays, one to store the frequencies of the notes that we want to play, another to store the colours of the buttons. The PlayGame function is called here. In a more polished version of the game, this would be triggered by some user interaction.



```
on start
    set pixels to NeoPixel at pin P13 with 6 leds as RGB (GRB format)
    pixels set brightness 32
    pixels clear
    pixels show
    set notes to create array with 659
                                   880
                                   330
                                   554
    set ledcols to create array with red
                                     blue
                                     green
                                     yellow
    call function PlayGame
```

Generate A Random Sequence

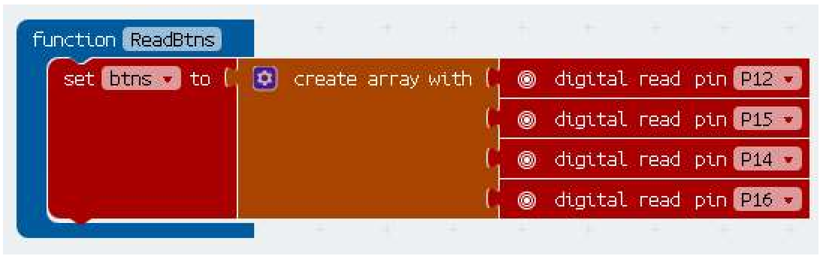
This function makes an array of random integers from 0 to 3. The sequence length is fixed at 100 items. The game will crash if that limit is reached. Code could be added to detect this situation and call it a win of some sort. People usually drop out somewhere before 20.



```
function GetRandomSequence
    set sequence to create array with pick random 0 to 3
    repeat 99 times
    do
        sequence add value pick random 0 to 3 to end
```

Reading The Buttons

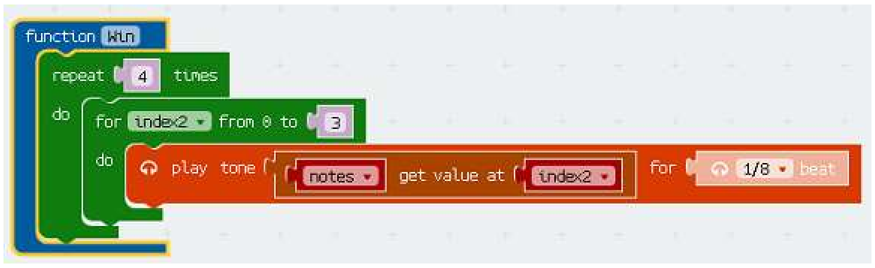
This function creates an array of the button states. This is a convenient way to read them all in one go.



```
function ReadBtns
    set btns to create array with digital read pin P12
                                  digital read pin P15
                                  digital read pin P14
                                  digital read pin P16
```

Win Sound Effect

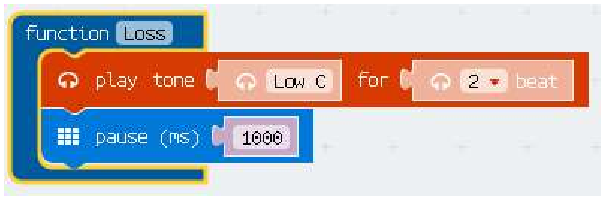
This is a little lazy. It plays the notes very quickly, one after the other, doing this 4 times. This could be improved.



```
function Win
    repeat 4 times
    do
        for index2 from 0 to 3
        do
            play tone notes get value at index2 for 1/8 beat
```

Loss Sound Effect

This is just a low note, played for a while.



```
function Loss
    play tone Low C for 2 beat
    pause (ms) 1000
```

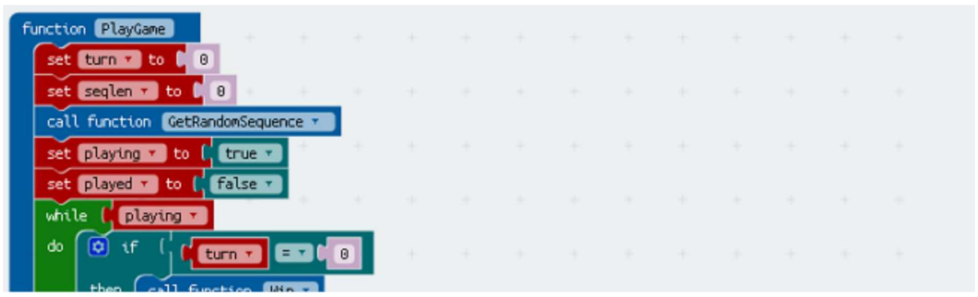
Playing A Part Of The Sequence

To make the game work, we need a way of playing the sequence for the player. This function does that. It depends on a varaiable called **turn**. That variable is set and changed in the PlayGame function.

A loop is used to play the sequence as far as we need to. The LED colours and notes are worked out using the arrays we set up earlier.

Playing The Game

This function took a few more blocks. The main playing loop checks for one of 4 conditions. The first situation is the start of the game. A win effect is played. The next condtion is that the user has not yet heard the sequence that they need to enter. The sequence is played. Then a check is made to see if the player has correctly entered the sequence. A win effect is played and the turn advanced by 1. In the last clause, the buttons are read. If a button is being held down, it is checked against the sequence. If it is incorrect, a loss effect is played and the game is done. If it is correct, a record of how far the player is into the sequence is kept and the game continues.



```
function PlayGame
    set turn to 0
    set seqlen to 0
    call function GetRandomSequence
    set playing to true
    set played to false
    while playing
    do  if turn = 0
        then call function Win
```

JavaScript

If you add the Neopixel library, you can copy and paste the JavaScript (listing at end of page) into the JavaScript window and then click to see the blocks.

```
let played = false
let notes: number[] = []
let ledcols: number[] = []
let playing = false
let seqlen = 0
let turn = 0
let sequence: number[] = []
let btns: number[] = []
let pixels: neopixel.Strip = null
function GetRandomSequence()  {
    sequence = [Math.random(4)]
    for (let i = 0; i < 99; i++) {
        sequence.push(Math.random(4))
    }
}
function Loss()  {
    music.playTone(131, music.beat(BeatFraction.Double))
    basic.pause(1000)
}
function Win()  {
    for (let i = 0; i < 4; i++) {
        for (let index2 = 0; index2 <= 3; index2++) {
            music.playTone(notes[index2], music.beat(BeatFraction.Eighth))
        }
    }
}
function PlaySequence()  {
    for (let index3 = 0; index3 <= turn - 1; index3++) {
        pixels.showColor(ledcols[sequence[index3]])
        pixels.show()
        music.playTone(notes[sequence[index3]], music.beat(BeatFraction.Whole))
        basic.pause(100)
        pixels.clear()
        pixels.show()
    }
}
function ReadBtns()  {
    btns = [pins.digitalReadPin(DigitalPin.P12), pins.digitalReadPin(DigitalPin.P15),
     pins.digitalReadPin(DigitalPin.P14), pins.digitalReadPin(DigitalPin.P16)]
}
function PlayGame()  {
    turn = 0
    seqlen = 0
    GetRandomSequence()
    playing = true
    played = false
    while (playing) {
        if (turn == 0) {
            Win()
            turn += 1
        }
        if (seqlen == 0 && played == false) {
            PlaySequence()
            played = true
        } else if (seqlen == turn) {
            Win()
            played = false
            turn += 1
            seqlen = 0
        } else {
            ReadBtns()
            for (let index = 0; index <= 4; index++) {
                if (btns[index] == 1) {
                    if (index != sequence[seqlen]) {
                        Loss()
                        playing = false
                    } else {
                        seqlen += 1
                        pixels.showColor(ledcols[index])
                        pixels.show()
                        music.playTone(notes[index], music.beat(BeatFraction.Whole))
                        basic.pause(250)
                        pixels.clear()
                        pixels.show()
                    }
                }
            }
        }
    }
}
pixels = neopixel.create(DigitalPin.P13, 6, NeoPixelMode.RGB)
pixels.setBrightness(32)
pixels.clear()
pixels.show()
notes = [659, 880, 330, 554]
ledcols = [neopixel.colors(NeoPixelColors.Red), neopixel.colors(NeoPixelColors.Blue),
neopixel.colors(NeoPixelColors.Green), neopixel.colors(NeoPixelColors.Yellow)]
PlayGame()
```

This needs some improvement, but there is enough for a start.