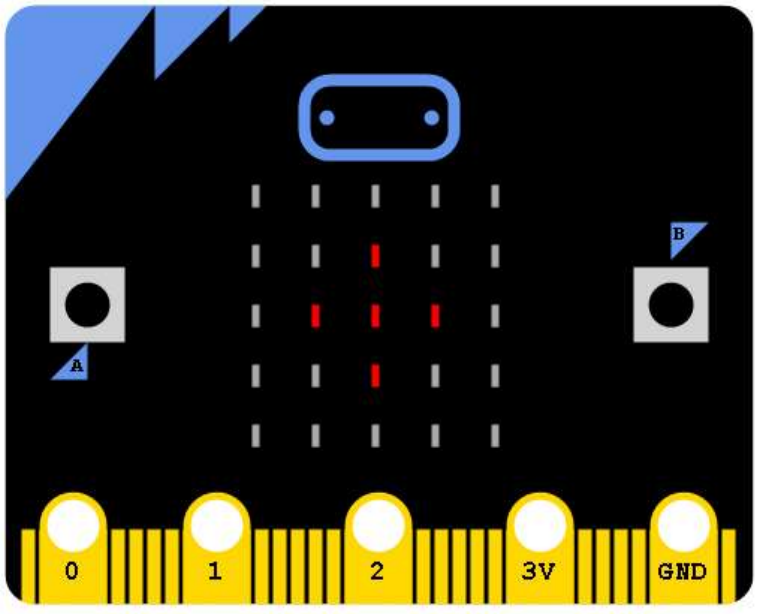
BBC micro:bit
Lights Out Game

Introduction

This example is based on the classic handheld electronic game Lights Out. Lights Out was played on a 5x5 grid of lights, just like our matrix.

In the classic game, the lights were buried beneath buttons. Pressing one of the buttons would change a number of lights from on to off or off to on. If all of the lights were out and we pressed the button in the dead centre of the grid, we would see this pattern. The light that was selected and the 4 lights around it are toggled (on becomes off, off becomes on).



Selecting the same button again would turn all of these lights out.

In our game, we don't have 25 buttons. If we want to avoid having to connect a load of components, we will need to develop a system for selecting the 'button' we want to press. In this example, the accelerometer will be read, allowing the player to move a blinking LED to the position they want to select. Pressing the A button at this point will do the move on that position.

Programming

Our first job is to develop the navigation method. Here is the test code for that part of the process. It's always a good idea to develop your programs with small milestones in mind. For this program, if there is no way to navigate and select an LED, there is no game to be made. This is a natural first step. Clearly, this is a technique you might use for another game.

```
from microbit import *

x = 2
y = 2
tick = -1

def DrawGame(t):
    img = Image('00000:'*5)
    img.set_pixel(x, y, (t % 2)*9)
    return img

while True:
    tick +=1
    if tick==2:
        tick = 0
    # check for movement
    dx = accelerometer.get_x()
    dy = accelerometer.get_y()
    if dx > 300:
        x += 1
        sleep(200)
    if dx < -300:
        x -= 1
        sleep(200)
    if dy > 300:
        y += 1
        sleep(200)
    if dy < -300:
        y -= 1
        sleep(200)
```

BBC Microbit

Collapse All Expand All

+ Block Editor - The Basics
+ Block Editor - Components
+ Kodu - micro:bit Worlds
+ JavaScript Blocks
+ JavaScript Blocks - Exercises
+ Blocks - Bit:Bot
+ Blocks - Bit:Commander
+ MicroPython - Starting Off
- MicroPython - Examples
  - Dice Rolling
  - Shut The Matrix
  - Encoding Morse Code
  - Encoding Ciphers
  - Drawing A Maze
  - Scrolling Race Track
  - Vertical Scroller
  - Concentration Game
  - Text Entry - Accelerometer
  - Charlie's Python Game
  - Lights Out Game
  - Sam's French Number Game
  - Bryn's Concentrated Clocks
  - Lattice Paths
  - Knight Moves
+ MicroPython - Components
+ MicroPython - Breakout Boards
+ MicroPython - Exercises
+ MicroPython - Pi Accessories
+ MicroPython - Bit:Bot
+ MicroPython - Bit:Commander
+ MicroPython - Projects
+ MicroPython - Visual Basic
+ Other - Odds & Ends

```
    # keep on grid
    x = max(0, min(x, 4))
    y = max(0, min(y, 4))
    # update screen
    i = DrawGame(tick)
    display.show(i)
    sleep(50)
```

With that done, our attention can turn to the rest of the game. Here is the full code,

```
from microbit import *
import random

x = 2
y = 2
tick = -1

grid = [
    [0,0,0,0,0],
    [0,0,0,0,0],
    [0,0,0,0,0],
    [0,0,0,0,0],
    [0,0,0,0,0]
    ]

# A 'move' in Lights Out
def Toggle(tx, ty):
    grid[tx][ty] ^= 1
    if tx>0:
        grid[tx-1][ty] ^= 1
    if tx<4:
        grid[tx+1][ty] ^= 1
    if ty>0:
        grid[tx][ty-1] ^= 1
    if ty<4:
        grid[tx][ty+1] ^= 1

def RandomGrid():
    for r in range(0,50):
        cx = random.randint(0,4)
        cy = random.randint(0,4)
        Toggle(cx, cy)

def DrawGame(t):
    img = Image('00000:'*5)
    for cy in range(0,5):
        for cx in range(0,5):
            img.set_pixel(cx, cy, grid[cx][cy]*5)
    img.set_pixel(x, y, (t % 2)*9)
    return img

def CheckWin():
    tot = 0
    for cy in range(0,5):
        for cx in range(0,5):
           tot += grid[cx][cy]
    if tot == 0:
       return True
    else:
       return False

# set the board up for a game
RandomGrid()

while True:
    tick +=1
    if tick==2:
        tick = 0
    # check for movement
    dx = accelerometer.get_x()
    dy = accelerometer.get_y()
    if dx > 300:
        x += 1
        sleep(200)
    if dx < -300:
        x -= 1
        sleep(200)
    if dy > 300:
        y += 1
        sleep(200)
    if dy < -300:
        y -= 1
        sleep(200)
    # keep on grid
    x = max(0, min(x, 4))
    y = max(0, min(y, 4))
    # check for button press
    if button_a.was_pressed():
        Toggle(x, y)
        sleep(200)
    # update screen
    i = DrawGame(tick)
    display.show(i)
    if CheckWin():
        sleep(1000)
        for w in range(0,6):
            display.show(Image.HAPPY)
            sleep(500)
            display.clear()
            sleep(500)
```

```
        RandomGrid()
        x = 2
        y = 2
        i = DrawGame(tick)
    sleep(50)
```

The board is randomised by making 50 random moves. This ensures that the pattern of lights is solvable. Not all patterns in this game can be solved.

In order to make the moves, the XOR operator is used. The on/off lights are stored as a 2D list of 1s and 0s. If we do XOR with 1, this will make a 1 into a 0 and a 0 into a 1. This saves us writing a handful of IF statements.

How To Beat The Game

Lights Out is a very interesting game. The mathematics behind the game is fascinating but also pretty tough to grasp for most people. The solution I use is the one developed by Jaap Scherphuis and explained on his site at **http://www.jaapsch.net/puzzles/lights.htm**. Jaap's site is very well known among the puzzle solving community and both the site and Jaap himself are brilliant sources of information for programmers, mathematicians and puzzle solvers.

Stage 1 - Chase The Lights

The start of the technique is called **chasing the lights**. This means that you start by looking at the top row. For every light that is on, select the light beneath it. Do this with the next 4 rows. If you are really lucky, this will solve the puzzle. If not, there will be a couple of lights on the bottom row.

Stage 2 - Set Up For The Win

Look at the first 3 LEDs on the bottom row.

- If LED(0,4) is on, press LED(3,0) and LED(4,0). These are the 4th and 5th lights of the top row.
- If LED(1,4) is on, press LED(1,0) and LED(4,0). These are the 2nd and 5th lights of the top row.
- If LED(2,4) is on, press LED(3,0). This is the 4th light on the top row.

Stage 3 - Chase The Lights Again

Repeat the chase the lights process again with the lights that are now on. By the time you get to the bottom row, the lights should all be out and you win.

Challenges

1. One obvious omission is a way of keeping score. Traditionally, the idea is to use the least number of moves that you can. You can add a variable for this and set it to 0 at the start of the game. Add 1 to it each time the player makes a move. Show the player the score when the game has been won. Remember to set the score back to 0 before the next game is played.
2. This would be nice with a different approach to the user interface. 5 externally connected pushbuttons would be one approach. You could also make an easy-to-control game if you used an analogue joystick and a pushbutton. Potentiometers, rotary encoders, keypads are all possible too.
3. There are variations on the classic game. A little bit of web searching and you will find out about them. You could adapt the program to make one of those.
4. You can use the user interface code of this program to make your own game. Anything which needs you to be able to select individual LEDs could make use of this approach.