MultiWingSpan

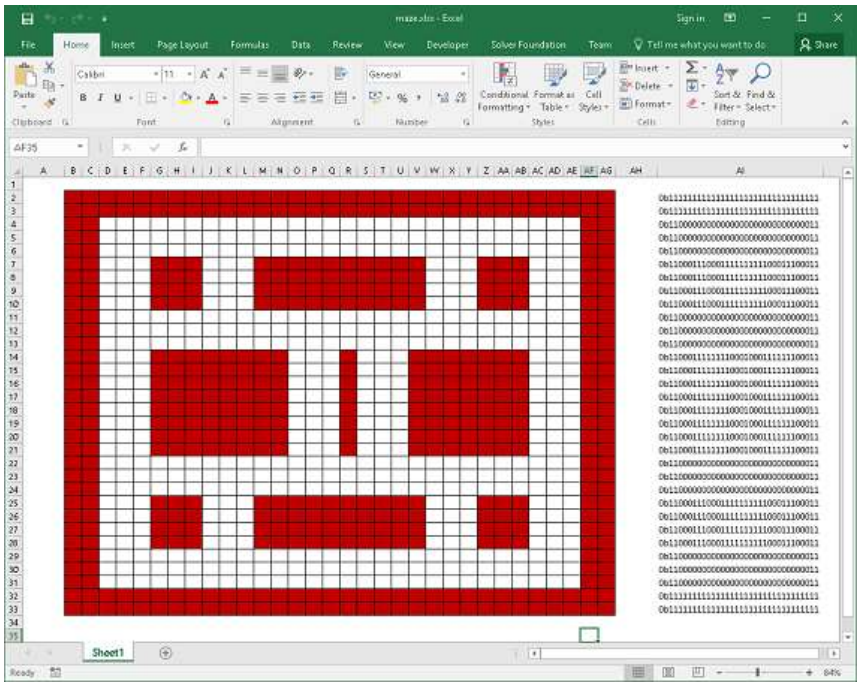
BBC micro:bit
Drawing A Maze

Introduction

If you hunt around in the MicroPython, you will find some code for a simple maze program. This page is a (very) slightly modified version of that program with some additional explanations of the process.

Making Your Maze

We define our maze using binary numbers. The maze is bigger than the matrix and only a 5x5 slice of it is going to be shown at any time. I used a spreadsheet with conditional formatting to help me to get a good sense of what the maze would look like.



That gave me this definition of the maze walls for Python.

```
maze = [
    0b11111111111111111111111111111111,
    0b11111111111111111111111111111111,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11000111000111111111100011100011,
    0b11000111000111111111100011100011,
    0b11000111000111111111100011100011,
    0b11000111000111111111100011100011,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11000111111111000100011111100011,
    0b11000111111111000100011111100011,
    0b11000111111111000100011111100011,
    0b11000111111111000100011111100011,
    0b11000111111111000100011111100011,
    0b11000111111111000100011111100011,
    0b11000111111111000100011111100011,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11000111000111111111100011100011,
    0b11000111000111111111100011100011,
    0b11000111000111111111100011100011,
    0b11000111000111111111100011100011,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11111111111111111111111111111111,
    0b11111111111111111111111111111111
]
```

The maze is described using 32bit binary numbers. Walls are 2 pixels thick around the edge of the maze.

Drawing The Maze

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

This function uses a logical AND operation to return the value of the pixel (1 or 0) at a particular set of coordinates. The right shift operator is used to move the bit to the rightmost place value ensuring a result of 1 if the pixel is a 1 and 0 if not.

```
def get_dot(x,y):
    return (maze[y] >> (31 - x)) & 1
```

The next function creates an image based on reading the correct parts of the integer list. It uses the get_dot function to read the pixel values.

```
def draw_slice(x,y):
    img = Image(5,5)
    for row in range(0,5):
        for col in range(0,5):
            img.set_pixel(col, row, get_dot( col + x - 2, row + y - 2)*5)
    img.set_pixel(2,2,9)
    return img
```

The loops read the pixels in a 5x5 grid with x and y as the centre of this square. After the loops, the centre pixel is set to full brightness to indicate the position of the character.

The full program is as follows,

```
from microbit import *

maze = [
    0b11111111111111111111111111111111,
    0b11111111111111111111111111111111,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11000111000111111111100011100011,
    0b11000111000111111111100011100011,
    0b11000111000111111111100011100011,
    0b11000111000111111111100011100011,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11000111111111000100011111100011,
    0b11000111111111000100011111100011,
    0b11000111111111000100011111100011,
    0b11000111111111000100011111100011,
    0b11000111111111000100011111100011,
    0b11000111111111000100011111100011,
    0b11000111111111000100011111100011,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11000111000111111111100011100011,
    0b11000111000111111111100011100011,
    0b11000111000111111111100011100011,
    0b11000111000111111111100011100011,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11000000000000000000000000000011,
    0b11111111111111111111111111111111,
    0b11111111111111111111111111111111
]
def get_dot(x,y):
    return (maze[y] >> (31 - x)) & 1

def draw_slice(x,y):
    img = Image(5,5)
    for row in range(0,5):
        for col in range(0,5):
            img.set_pixel(col, row, get_dot( col + x - 2, row + y - 2)*5)
    img.set_pixel(2,2,9)
    return img

# main program

x = 2
y = 2

# game loop

while True:
    if accelerometer.get_x() > 200 and get_dot(x+1,y)!=1:
        x += 1
    elif accelerometer.get_x() < -200 and get_dot(x-1,y)!=1:
        x -= 1
    elif accelerometer.get_y() > 200 and get_dot(x,y+1)!=1:
        y += 1
    elif accelerometer.get_y() < -200 and get_dot(x,y-1)!=1:
        y -= 1
    display.show(draw_slice(x,y))
    sleep(50)
```

In the last part, we set our starting co-ordinates.

The infinite loop checks the accelerometer readings and, if above 200 in either direction, the coordinates of the character are changed. The second part of each IF statement ensures that there is no wall in the position that you want to move to, only changing the position if there is no collision.

The image is drawn each time in the loop and a delay of 50 milliseconds takes place before the next iteraton. You need a decent pause because accelerometer control can be tricky for humans.

Challenges

1. Start by making your own maze.
2. You can make the character blink by adding a variable to the main program. Set the variable to 0 before the loop. Add one to the variable inside the while loop. Every time the variable reaches 9, make it 0 again. Use this variable as the brightness setting for your character pixel, drawn in the draw_slice function.
3. The game needs something for you to find in the maze. Write code to find a random, free, location on the grid. Store the coordinates and draw a faint dot at that location. In the while loop, check whether the player is in the same position and do something interesting, like stop the game.
4. Add a scoring system. Start by setting the score really high. Reduce it by an amount each time the game loop repeats. Change the loop stopping condition or break out of it when the player reaches the end or picks up the item. Report the score at the end.
5. This is the basis for a race track. Redesign the maze as a race track with some twists and turns. Write an IF statement to add to the game loop, checking if the player has reached the finish. Make it so that the start and finish are not joined up in a loop - you don't want to be able to go backwards. You can extend the binary array vertically, adding more rows if you need more space. Time the whole affair and you have a racing game.