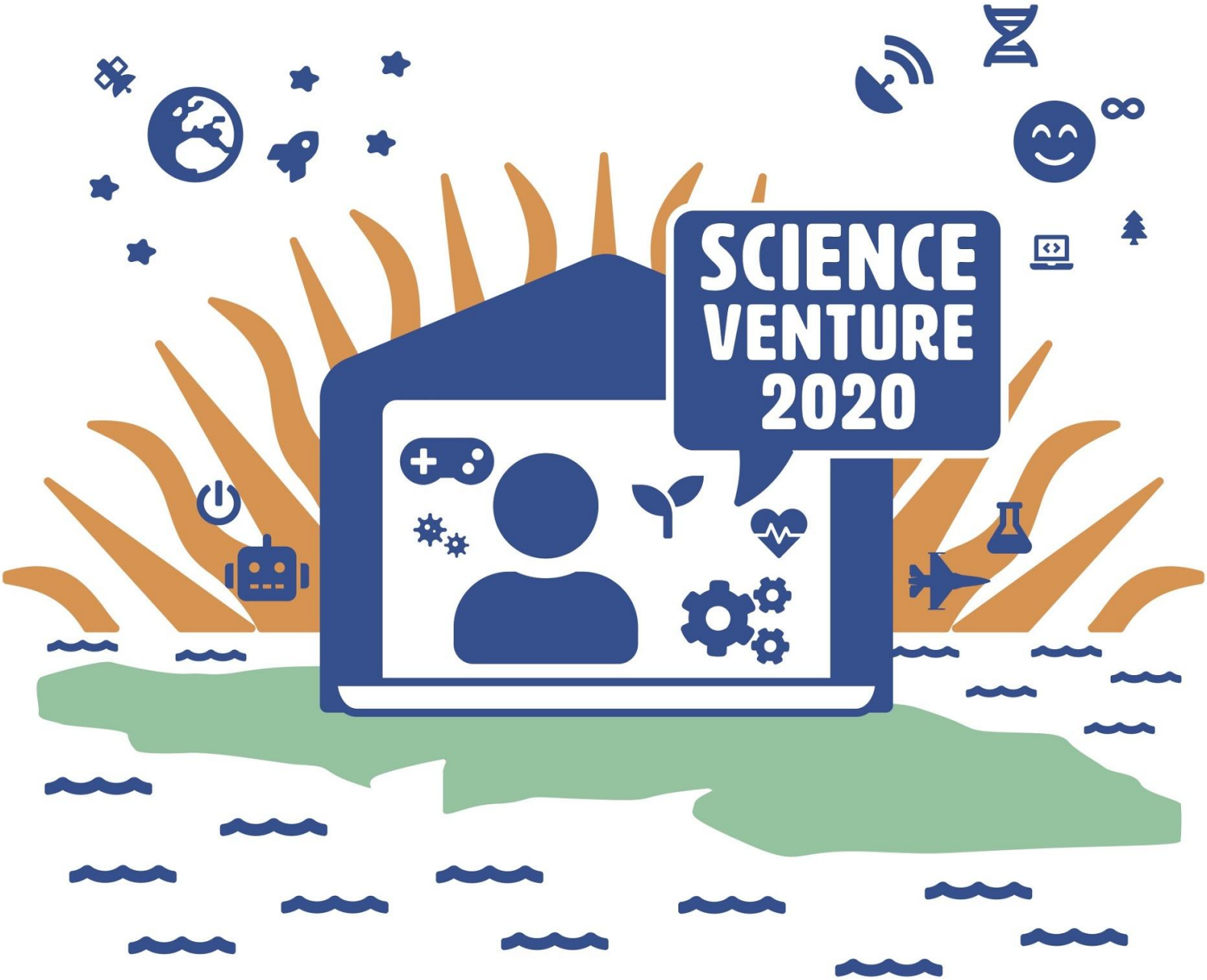


Micro:Bits

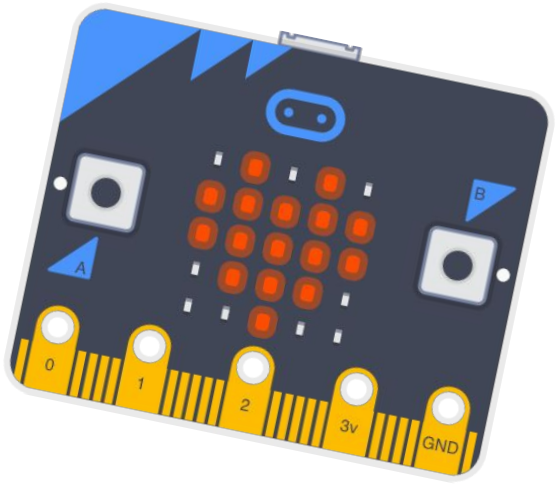
Micro:Bits Python Snake Game

A snake slithers from one corner of the Micro:bit screen to the other. Can you remember it's path and follow it?

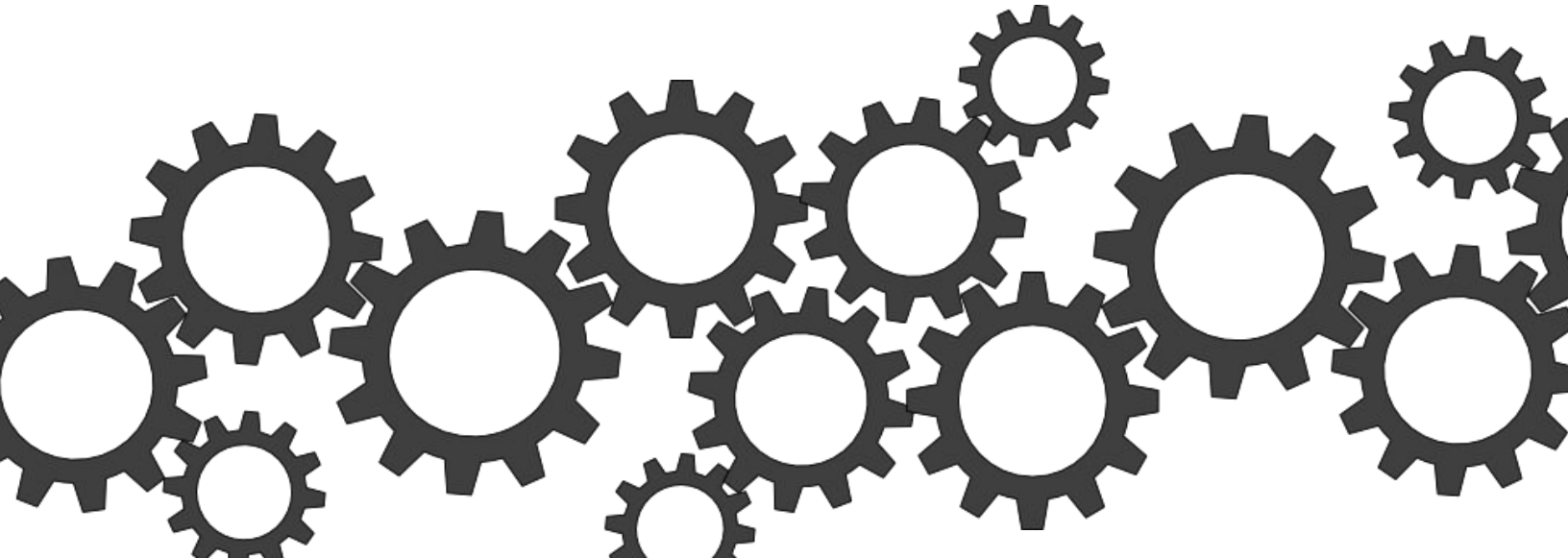
What is a Micro:Bit?

A Micro:bit is a miniature computer with a 5x5 LED display grid. We can use the MakeCode editor to code a program to execute on our Micro:bit. A program is simply a set of instructions for our Micro:bit to complete.

Materials



- Micro:bit kit
  - 1 Micro:bit
  - 1 Battery pack
  - 1 Micro USB cable
- Optional: 1 Dongle adapter if using Mac



Micro:Bits Python Snake Game

Procedure

1) Open make code editor and switch it to Python.

2) You will notice there is some code there already! Anything code we type under the “def on_forever():” statement will act the same as if it was in the “forever” frame in the block coding version. In block coding, you could see how the code fit inside of each other with the blocks. In Python, you use whitespace to see this. Notice how “pass” is indented below the on_forever() definition? That is four spaces in. When we are ready to add to the forever loop, we will replace pass with our first line of code.

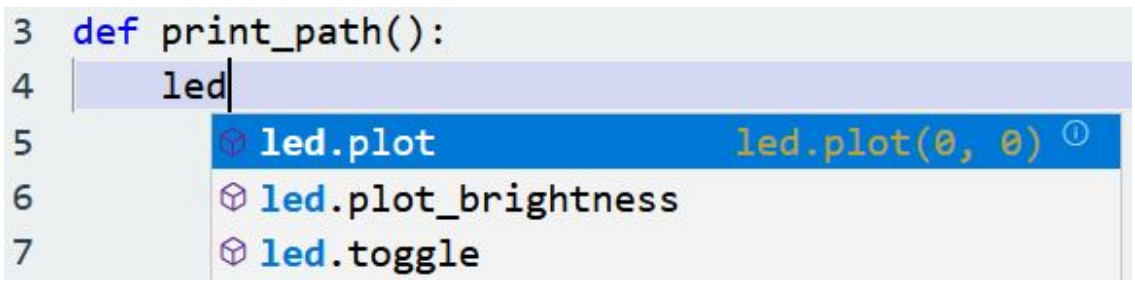
3) For our snake game, we want the program to randomly generate a path of LEDS from the top left [0,0] to the bottom right [4,4], then to display that path. After the path disappears, we will have to use the A and B buttons to try to trace the same path. Before we code a randomly generated path, let’s “hardcode” in an example path, then write a function which prints the path to the screen.

4) Put your cursor in front of the first line and press ‘enter’ to give yourself some space at the top of the program. I named our list egPath, and filled the list with smaller lists!

```
1  egPath = [ [0,0], [0,1], [0,2], [0,3], [0,4], [1,4], [2,4], [3,4], [4,4] ]
```

5) In Python, lists are a very useful structure, and you will often even see lists of lists like this. The outermost brackets are the main list, and each of the LED coordinates is a smaller list (with two elements, the position of the x coordinate, and then the position of the y coordinate).

6) Now, let’s write a function that prints this path to the LEDs! Like we have seen a few times now, function declarations always start with the ‘def’ keyword, followed by the function name, parentheses, and a colon. Make sure you have 4 spaces at the start of the next line. We want to light a specific LED, so start to type led and then you will see the autocomplete options. We want to use led.plot, so press tab to autocomplete.



Micro:Bits Python Snake Game

7) Let's run the function like this to see what happens. Below the function, but without any indentation, type "print_path()". Then, run the program (we can use the virtual Micro:bit for this still). The top left pixel of the Micro:bit should light up.

8) Now, let's add to our function so that it will light up all of the pixels in our example path. We need to add a for loop to do this. Click in front of our led.plot command and press enter to add a line above it. For this code, we will write "for step in egPath:"
   a) The basic for loop structure goes "for *value* in *list*:" where list is the list we want to iterate through, and value becomes the variable name keeping track of which item in the list we are currently using. The for loop always starts with value = list[0], then list[1], list[2], and so on until list runs out of elements.
   b) So, in this case, we are using "step" to keep track of the individual elements of our list (named egPath). Each "step" is a list of form [x,y], telling us the coordinates of the pixel. To access the x coordinate, we write step[0], and for the y coordinate, step[1].

9) We will write x = step[0] and y = step[1] inside of the loop. Each time through the loop, the x and y coordinates of that step will be saved in these two variables. Then, we can call led.plot(x,y) to plot those coordinates.

10) Also add a pause of half of a second so that we can see the path step by step.

```
1  egPath = [ [0,0], [0,1], [0,2], [0,3], [0,4], [1,4], [2,4], [3,4], [4,4] ]
2
3  def print_path():
4      for step in egPath:
5          x = step[0]
6          y = step[1]
7          led.plot(x, y)
8          basic.pause(500)
9
10 print_path()
```

11) This works now, but we are still missing something crucial -- comments! Use the # and write a line describing what our new function does. It also helps to make a few comments about any parts you found difficult so that you remember later.

12) Let's randomly generate a path now! Let's define a new function, generate_path. This time, our function will create something new, and pass it along to the program afterwards.

```
11 def generate_path():
12     # randomly generates a path and then returns the list
13     path = []
14     path.append([0,0])
15     return path
```

Micro:Bits Python Snake Game

13) Line 13 initializes the list as empty. Line 14 appends the coordinates to the top left pixel to the list. Then, “return path” ends the function by passing the new path to wherever it was called. This will be the basic structure of our function: make a list, put some things in it, then hand it off to the next part of the program. We just need to add some more steps to the path now! Let’s use a different kind of loop this time.

```
11 def generate_path():
12     # randomly generates a path and then returns the list.
13     path = []
14     path.append([0,0])
15     #This for loop starts with i=1, goes up to i=8
16     for i in range(1,9):
17         #x and y are the coordinates of the last step
18         x = path[i-1][0]
19         y = path[i-1][1]
20         # Choose a random number, 0 or 1
21         # 0 means 'False' and 1 means 'True'
22         stepRight = randint(0,1)
23         if stepRight:
24             # stepping right means adding 1 to the x coordinate
25             path.append([x+1, y])
26         else:
27             # stepping down means adding 1 to the y coordinate
28             path.append([x, y+1])
29     return path
```

14) We use “stepRight” as a boolean variable, 0 means false and 1 means true. Our next step in the path will either be one step to the right or one step down, depending on what random number was chosen. There’s still a small issue in this code, but we will come back to fix it soon.

Micro:Bits Python Snake Game

15) Let's try to generate a path and then display it with our print function. There's a small issue here: we wrote the print_path function to only print egPath! Let's change line 5 so that it displays a variable named path instead. We will call generate_path, then save the result as path before printing it.

```
1  # egPath = [ [0,0], [0,1], [0,2], [0,3], [0,4], [1,4], [2,4], [3,4], [4,4] ]
2
3  def print_path():
4      # lights up the LEDS in the list "path", one at a time
5      for step in path:
6          x = step[0]
7          y = step[1]
8          led.plot(x, y)
9          basic.pause(500)
10
11 def generate_path():
12     # randomly generates a path and then returns the list.
13     path = []
14     path.append([0,0])
15     #This for loop starts with i=1, goes up to i=8
16     for i in range(1,9):
17         #x and y are the coordinates of the last step
18         x = path[i-1][0]
19         y = path[i-1][1]
20         # Choose a random number, 0 or 1
21         # 0 means 'False' and 1 means 'True'
22         stepRight = randint(0,1)
23         if stepRight:
24             # stepping right means adding 1 to the x coordinate
25             path.append([x+1, y])
26         else:
27             # stepping down means adding 1 to the y coordinate
28             path.append([x, y+1])
29     return path
30
31 #define and then print path
32 path = generate_path()
33 print_path()
```

Micro:Bits Python Snake Game

16) Keep restarting the program to see which paths get generated. Does the snake always get to the bottom right corner? Why not?
   a) The loop always adds 8 steps to the path.
   b) Sometimes the loop steps to the right too many times and goes off the right end of the screen. Sometimes it goes off the bottom end of the screen. The biggest x or y coordinate is 4, so any coordinates with a 5 or larger go off the screen.

17) Let's add a little more code to our generate path function to make sure we always end with the ninth coordinate as [4,4]. Each time through the loop, before we decide to step right or to step down, let's check the size of the x and y coordinates. If one of the coordinates is 4, we will decide the direction of the next step instead of letting it be randomly chosen. Add a few lines of blank space to the middle of your generate path function.

```
11  def generate_path():
12      # randomly generates a path and then returns the list.
13      path = []
14      path.append([0,0])
15      #This for loop starts with i=1, goes up to i=8
16      for i in range(1,9):
17          #x and y are the coordinates of the last step
18          x = path[i-1][0]
19          y = path[i-1][1]
20
21
22          # Choose a random number, 0 or 1
23          # 0 means 'False' and 1 means 'True'
24          stepRight = randint(0,1)
25          if stepRight:
26              # stepping right means adding 1 to the x coordinate
27              path.append([x+1, y])
28          else:
29              # stepping down means adding 1 to the y coordinate
30              path.append([x, y+1])
31      return path
```

Micro:Bits Python Snake Game

18) If x is already 4, we will set stepRight to be 0 (which means do not step right again). If y is already 4, we will set stepRight to be 1 (which means to step right, since we have no room to step down). If neither of those things are true, we can safely pick a direction at random.

```
11  def generate_path():
12      # randomly generates a path and then returns the list.
13      path = []
14      path.append([0,0])
15      # This for loop starts with i=1, goes up to i=8
16      for i in range(1,9):
17          # x and y are the coordinates of the last step
18          x = path[i-1][0]
19          y = path[i-1][1]
20
21          # decide whether to randomly choose the next step
22          if x == 4:
23              #if x is 4, do not step right
24              stepRight = 0
25          elif y==4:
26              #if y is 4, step right
27              stepRight = 1
28          else:
29              # Choose a random number, 0 or 1
30              # 0 means 'False' and 1 means 'True'
31              stepRight = randint(0,1)
32
33          # take the next step
34          if stepRight:
35              # stepping right means adding 1 to the x coordinate
36              path.append([x+1, y])
37          else:
38              # stepping down means adding 1 to the y coordinate
39              path.append([x, y+1])
40      return path
```

Micro:Bits Python Snake Game

19) Now the path should make it to the bottom right LED each time. To finish the game, we need to add the player's response. We will start with x coordinate 0 and y coordinate 0, and wait for the player's response. If the player presses A, we will increase the y coordinate by 1 (moving down). If the player presses B, we will increase the x coordinate by 1 (moving right). When each step is added, we will check if the player's new step matches with the same step in the snake's path. If the step does not match, it will be game over! If the player can get all the way to [4,4] by following the snake's path, they win the game.

20) Start by clearing the snake's path from the LED screen and initializing the player's coordinates. We can find the code to clear the screen in the Basic tab.
   a) Since we are adding more code here, this is a great time to add some comments to the existing code as well.
   b) Create variables 'current_x' and 'current_y' to keep track of the current x and y coordinates (both 0 for now!). Then, plot these coordinates to the LED screen.
   c) Now we will make a while loop. We want to keep looping while it is NOT the case that 'current_x' is 4 AND 'current_y' is 4. So we exit the loop when 'current_x' is 4 AND 'current_y' is 4.
   d) Inside the loop, we will check if either button has been pressed. We use an if statement, and we can find the "button is pressed" condition by looking in the Input tab. You will have to change one of them to say 'Button.B" instead of "Button.A". We will fill out the inside of these if statements in a moment, but for now leave a comment saying what we will do if the condition is true.

```
42  # Generate the snake's path
43  path = generate_path()
44  # Print the snake's path to the screen
45  print_path()
46  # Clear the screen
47  basic.clear_screen()
48  # Keep track of the current x and y values, then plot the first values
49  current_x = 0
50  current_y = 0
51  led.plot(current_x, current_y)
52
53  # Loop until the player gets to the bottom right LED
54  while not (current_x == 4 and current_y == 4):
55      if input.button_is_pressed(Button.A):
56          # move down
57      if input.button_is_pressed(Button.B):
58          # move right
```

Micro:Bits Python Snake Game

21) If we want to move down, we need to increase current_y by 1. Then, we need to compare our current coordinates to the coordinates of the same step in the snake's path. In order to do this, we actually need to add one more variable to keep track of which step number we are on. Let's call this variable "stepNum", and initialize it to 0 before our while loop. Each time we increment the x or the y value, we will increment "stepNum" also.

```
52  # Keep track of which step we are on
53  stepNum = 0
54  # Loop until the player gets to the bottom right LED
55  while not (current_x == 4 and current_y == 4):
56      if input.button_is_pressed(Button.A):
57          # move down
58          current_y = current_y + 1
59          stepNum = stepNum + 1
60      if input.button_is_pressed(Button.B):
61          # move right
62          current_x = current_x + 1
63          stepNum = stepNum + 1
```

22) Now we ot. If we do, let's display the current step on the LED display. If not, we will display a game over screen.

```
55  while not (current_x == 4 and current_y == 4):
56      if input.button_is_pressed(Button.A):
57          # move down
58          current_y = current_y + 1
59          stepNum = stepNum + 1
60      if input.button_is_pressed(Button.B):
61          # move right
62          current_x = current_x + 1
63          stepNum = stepNum + 1
64
65      if path[stepNum][0] == current_x and path[stepNum][1] == current_y:
66          # plot the new step if it is correct
67          led.plot(current_x, current_y)
68      else:
69          # game over and reset if the new step is wrong
70          basic.clear_screen()
71          basic.show_icon(IconNames.SAD)
72          basic.pause(1000)
73          control.reset()
```

Micro:Bits Python Snake Game

23) It seems like our game is finished, but there is one more tweak to make! As is, the game over screen displays regardless of which button is pressed. That is because the code runs so fast that it registers multiple button presses even if you only press the button once. To fix this, we need to introduce a short pause after each button press. How long do you normally have the button pressed for? Let's try a delay of half a second, or 500 milliseconds, after each button press.

```
while not (current_x == 4 and current_y == 4):
    if input.button_is_pressed(Button.A):
        # pause so the game only registers one button press
        basic.pause(500)
        # move down
        current_y = current_y + 1
        stepNum = stepNum + 1
    if input.button_is_pressed(Button.B):
        # pause so the game only registers one button press
        basic.pause(500)
        # move right
        current_x = current_x + 1
        stepNum = stepNum + 1

    if path[stepNum][0] == current_x and path[stepNum][1] == current_y:
        # plot the new step if it is correct
        led.plot(current_x, current_y)
    else:
        # game over and reset if the new step is wrong
        basic.clear_screen()
        basic.show_icon(IconNames.SAD)
        basic.pause(1000)
        control.reset()
```

24) Now the game should run fine! Here's a screenshot of the entire game code. There are a lot of lines!

Micro:Bits Python Snake Game

```
# egPath = [ [0,0], [0,1], [0,2], [0,3], [0,4], [1,4], [2,4], [3,4], [4,4] ]

def print_path():
    # lights up the LEDS in the list "path", one at a time
    for step in path:
        x = step[0]
        y = step[1]
        led.plot(x, y)
        basic.pause(500)

def generate_path():
    # randomly generates a path and then returns the list.
    path = []
    path.append([0,0])
    # This for loop starts with i=1, goes up to i=8
    for i in range(1,9):
        # x and y are the coordinates of the last step
        x = path[i-1][0]
        y = path[i-1][1]

        # decide whether to randomly choose the next step
        if x == 4:
            #if x is 4, do not step right
            stepRight = 0
        elif y==4:
            #if y is 4, step right
            stepRight = 1
        else:
            # Choose a random number, 0 or 1
            # 0 means 'False' and 1 means 'True'
            stepRight = randint(0,1)

        # take the next step
        if stepRight:
            # stepping right means adding 1 to the x coordinate
            path.append([x+1, y])
        else:
            # stepping down means adding 1 to the y coordinate
            path.append([x, y+1])
    return path
```

Micro:Bits Python Snake Game

```
41
42  # Generate the snake's path
43  path = generate_path()
44  # Print the snake's path to the screen
45  print_path()
46  # Clear the screen
47  basic.clear_screen()
48  # Keep track of the current x and y values, then plot the first values
49  current_x = 0
50  current_y = 0
51  led.plot(current_x, current_y)
52  # Keep track of which step we are on
53  stepNum = 0
54  # Loop until the player gets to the bottom right LED
55  while not (current_x == 4 and current_y == 4):
56      if input.button_is_pressed(Button.A):
57          # pause so the game only registers one button press
58          basic.pause(500)
59          # move down
60          current_y = current_y + 1
61          stepNum = stepNum + 1
62      if input.button_is_pressed(Button.B):
63          # pause so the game only registers one button press
64          basic.pause(500)
65          # move right
66          current_x = current_x + 1
67          stepNum = stepNum + 1
68
69      if path[stepNum][0] == current_x and path[stepNum][1] == current_y:
70          # plot the new step if it is correct
71          led.plot(current_x, current_y)
72      else:
73          # game over and reset if the new step is wrong
74          basic.clear_screen()
75          basic.show_icon(IconNames.SAD)
76          basic.pause(1000)
77          control.reset()
78
79  # If we take a wrong step then the program resets, so if we make it
80  # outside of the while loop, then we win the game!
81  basic.show_icon(IconNames.HAPPY)
82  basic.show_string("You win!")
83  control.reset()
```

#SVatHome

Want to share your project or results with us?

Email or tag us @ScienceVenture

Have a question?

Reach us at svcamp@engr.uvic.ca