Sense HAT Marble Maze

Introduction

In this activity you will create a marble maze game that can be played on the LED matrix of your Sense HAT. You will use the in-built orientation sensors of the Sense HAT to control the movement of a simulated marble that moves the same way a real marble would roll around a maze.

What you will make

Here is an example of a marble maze game running on the Sense HAT emulator. Press the **Run** button to see it in action.

Click and drag the emulated Sense HAT to move it around. As you change the position of the Sense HAT, the marble on the display moves around.

What you will learn

By coding a marble maze game with your Raspberry Pi and Sense HAT, you will learn how to:

- Illuminate Sense HAT pixels
- Capture pitch, roll, and yaw data with the Sense HAT's orientation sensors
- Implement a basic form of collision detection

This resource covers elements from the following strands of the Raspberry Pi Digital Making Curriculum (https://www.raspberrypi.org/curriculum/):

- Design basic 2D and 3D assets (https://www.raspberrypi.org/curriculum/design/creator)
- Combine programming constructs to solve a problem (https://www.raspberrypi.org/curriculum/programming/builder)
- Process input data to monitor or react to the environment (https://www.raspberrypi.org/curriculum/physical-computing/developer)

What you will need

Hardware

- Sense HAT (If you don't have one, you can use the emulator)

Software

You will need the latest version of Raspbian (https://www.raspberrypi.org/downloads/), which already includes the following software packages:

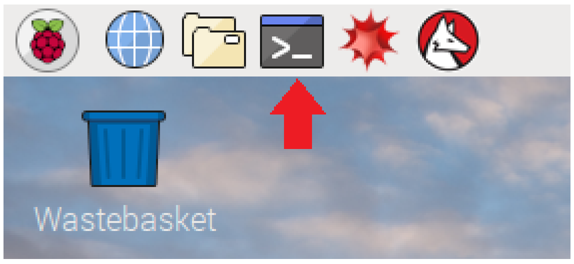
- Python 3
- Sense HAT for Python 3

If for any reason you need to install a package manually, follow these instructions:

Install a software package on the Raspberry Pi

Your Raspberry Pi will need to be online to install packages. Before installing a package, update and upgrade Raspbian, your Raspberry Pi's operating system.

- Open a terminal window and enter the following commands to do this:



```
sudo apt-get update
sudo apt-get upgrade
```

- Now you can install the packages you'll need by typing `install` commands into the terminal window. For example, here's how to install the Sense HAT software:

```
sudo apt-get install sense-hat
```

Type this command into the terminal to install the Sense HAT package:

```
sudo apt-get install sense-hat
```

What is a marble maze?

A marble maze is a game of skill and dexterity: marbles are placed inside a maze, and the player guides them to a specific point by tilting the maze in various directions to roll the marbles around.

By Annielogue (Own work) [CC BY-SA 3.0] (http://creativecommons.org/licenses/by-sa/3.0), via Wikimedia Commons

Given that the Sense HAT is capable of reporting its exact orientation and has an in-built 8×8 pixel LED display, it is the perfect device with which to create an electronic marble maze game.

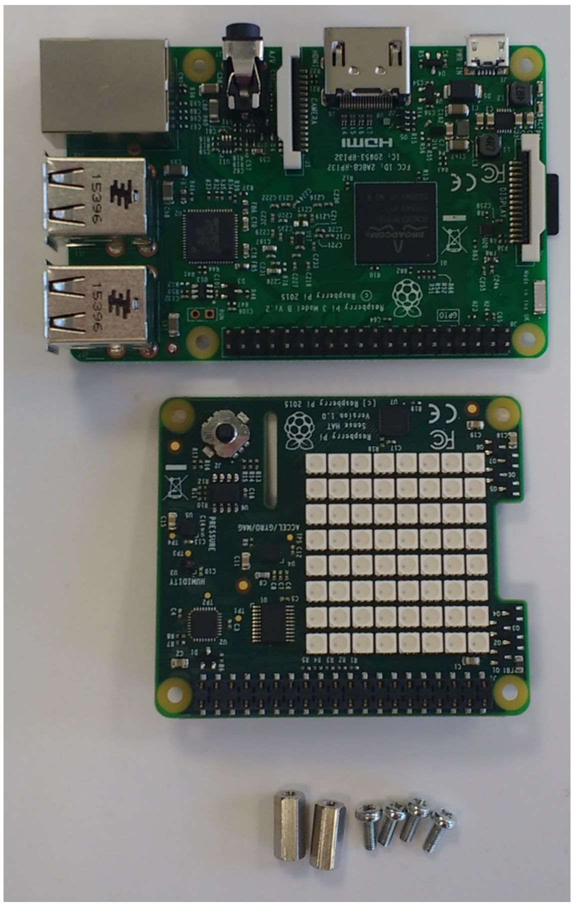
Set up the Sense HAT

- Start by attaching the Sense HAT to your Raspberry Pi.

Attaching a Sense HAT

Before attaching any HAT to your Raspberry Pi, ensure that the Pi is shut down.

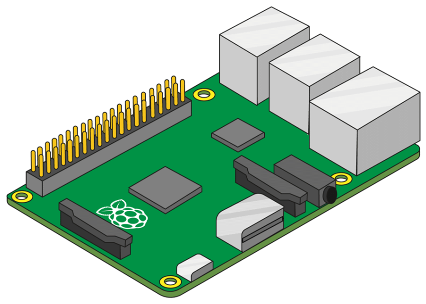
- Remove the Sense HAT and parts from their packaging.



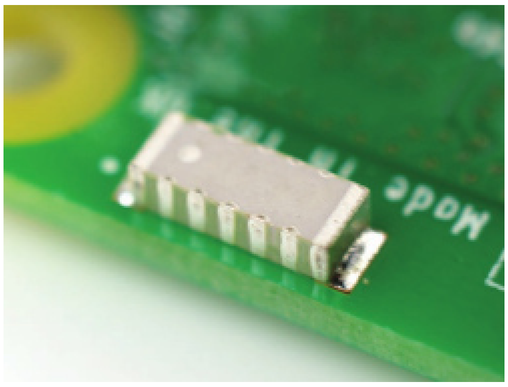
- Use two of the provided screws to attach the spacers to your Raspberry Pi, as shown below.

Note: the above step is optional — you do not have to attach the standoffs to the Sense HAT for it to work.

- Then push the Sense HAT carefully onto the pins of your Raspberry Pi, and secure it with the remaining screws.



Note: using a metal stand-off next to the Raspberry Pi 3's wireless antenna will degrade its performance and range. Either leave out this stand-off, or use nylon stand-offs and nylon screws instead.



Pro tip: be careful when taking off the Sense HAT, as the 40-pin black header tends to get stuck.

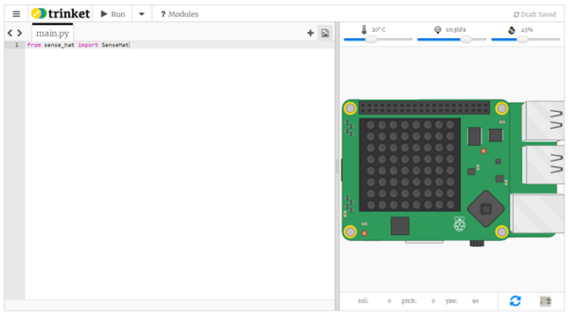
- If you do not have a Sense HAT, you can create the project offline or in a web browser by using the Sense HAT emulator.

Using the Sense HAT emulator

If you don't have access to a Sense HAT, you can use the emulator.

Online Sense HAT emulator

There is an online emulator you can use in your browser to write and test code for the Sense HAT.



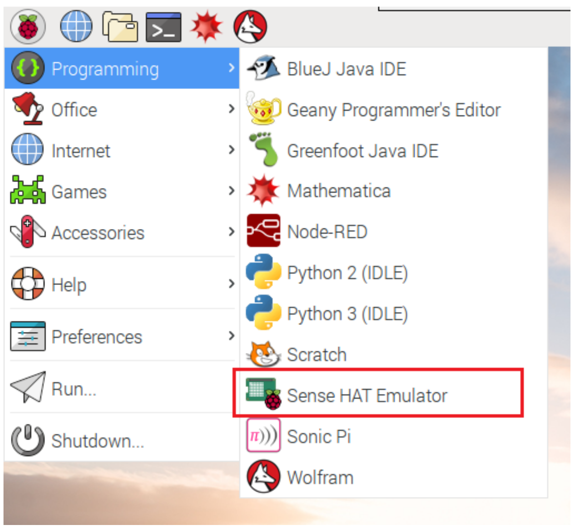
```
from sense_hat import SenseHat
```

- Open an internet browser and go to https://trinket.io/sense-hat (https://trinket.io/sense-hat).

- If you would like to save your work, you will need to create a free account (https://trinket.io/signup) on the Trinket website.

Sense HAT emulator on the Raspberry Pi

If you are using a Raspberry Pi, there is a Sense HAT emulator included in the Raspbian operating system.



- From the main menu, select **Programming** > **Sense HAT emulator** to open a window containing the emulator.

- If you are using this version of the emulator, your program must import from `sense_emu` instead of `sense_hat`:

```
from sense_emu import SenseHat
```

If you later want to run your code on a real Sense HAT, just change the import line as shown below. All other code can remain exactly the same.

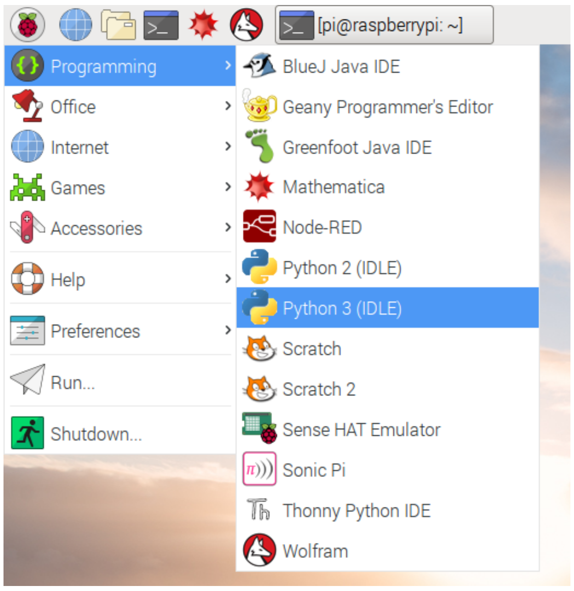
```
from sense_hat import SenseHat
```

- Open the IDLE editor and create a new file, or if you are using the online emulator, open a new trinket.

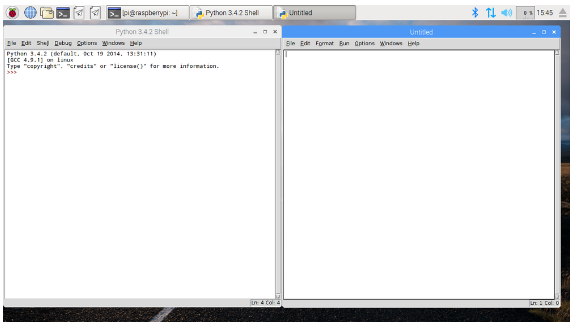
Opening IDLE3

IDLE is Python's **I**ntegrated **D**eve**l**opment **E**nvironment, which you can use to write and run code.

To open IDLE, go to the menu and choose `Programming`.
You should see two versions of IDLE - make sure you click on the one that says `Python 3 (IDLE)`.



To create a new file in IDLE, you can click on `File` and then `New File` in IDLE's menu bar.
This will open a second window in which you can write your code.


[pi@raspberrypi: ~]
Python 3.4.2 Shell
Untitled
File Edit Shell Debug Options Windows Help
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

- Save your IDLE file as **marble_maze.py**.
- In the new file, start by importing the Sense HAT module:

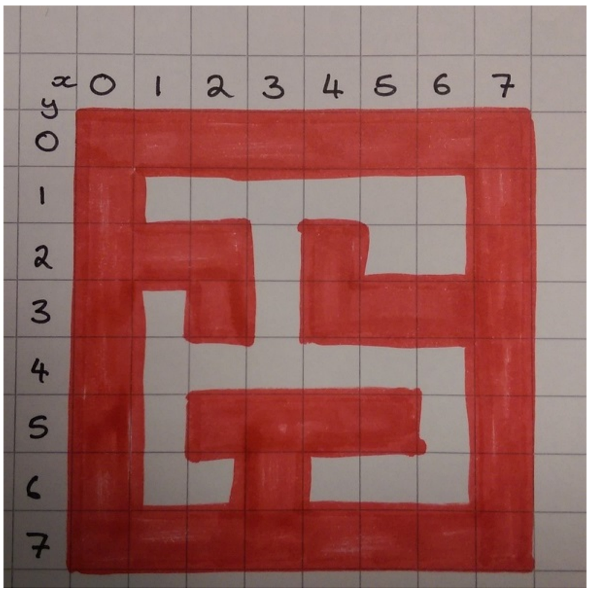
```
from sense_hat import SenseHat
```

- Next, create a connection to your Sense HAT and clear the screen by adding:

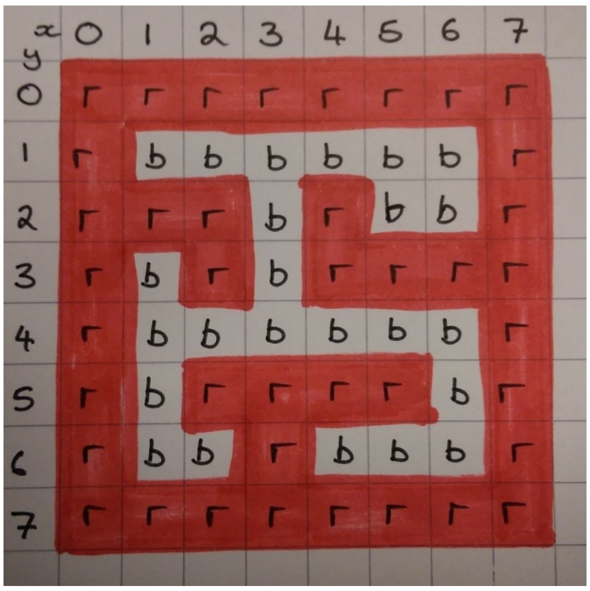
```
sense = SenseHat()
sense.clear()
```

Draw the maze

- First, design a maze that fits on the 8×8 LED matrix on squared paper. It is important that the maze is constructed from solid walls, and that there are no diagonal gaps.



- Once you have drawn your maze, write down the initial of the colour used in each square.



y \ x	0	1	2	3	4	5	6	7
0	r	r	r	r	r	r	r	r
1	r	b	b	b	b	b	b	r
2	r	r	r	b	r	b	b	r
3	r	b	r	b	r	r	r	r
4	r	b	b	b	b	b	b	r
5	r	b	r	r	r	r	b	r
6	r	b	b	r	b	b	b	r
7	r	r	r	r	r	r	r	r

Now you can recreate your maze on the Sense HAT's LED display.

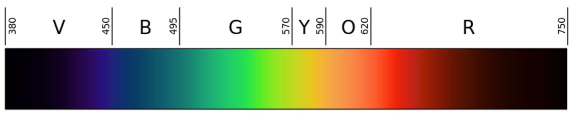
- Define the colour of the walls and the floor by specifying their RGB values. Add this code:

```
r = (255,0,0)
b = (0,0,0)
```

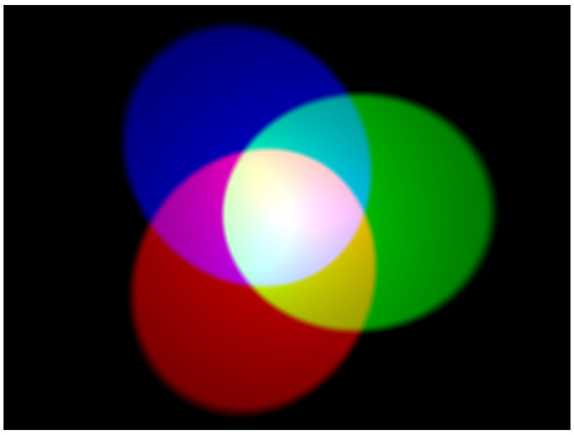
In this example, `r` represents red and `b` represents blank.

Representing colours with numbers

The colour of an object depends on the colour of the light that it reflects or emits. Light can have different wavelengths, and the colour of light depends on the wavelength it has. The colour of light according to its wavelength can be seen in the diagram below. You might recognise this as the colours of the rainbow.



Humans see colour because of special cells in our eyes. These cells are called *cones*. We have three types of cone cells, and each type detects either red, blue, or green light. Therefore all the colours that we see are just mixtures of the colours red, blue, and green.



In additive colour mixing, three colours (red, green, and blue) are used to make other colours. In the image above, there are three spotlights of equal brightness, one for each colour. In the absence of any colour the result is black. If all three colours are mixed, the result is white. When red and green combine, the result is yellow. When red and blue combine, the result is magenta. When blue and green combine, the result is cyan. It's possible to make even more colours than this by varying the brightness of the three original colours used.

Computers store everything as 1s and 0s. These 1s and 0s are often organised into sets of 8, called **bytes**.

A single byte (A set of 8 bits, for example 10011001) can represent any number from 0 up to 255.

When we want to represent a colour in a computer program, we can do this by defining the amounts of red, blue, and green that make up that colour. These amounts are usually stored as a single byte (A set of 8 bits, for example 10011001) and therefore as a number between 0 and 255.

Here's a table showing some colour values:

Red	Green	Blue	Colour
255	0	0	Red
0	255	0	Green
0	0	255	Blue
255	255	0	Yellow
255	0	255	Magenta
0	255	255	Cyan

You can find a nice colour picker to play with at w3schools (https://www.w3schools.com/colors/colors_rgb.asp).

- Add code to draw your maze by copying the letters you wrote on your plan into a two-dimensional list. Each row of LEDs is represented by a list, and the lists for the eight rows are grouped together in a larger list.

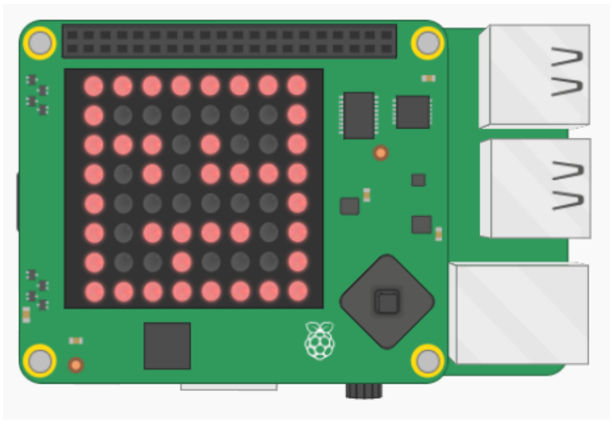
```
maze = [[r,r,r,r,r,r,r,r],
        [r,b,b,b,b,b,b,r],
        [r,r,r,b,r,b,b,r],
        [r,b,r,b,r,r,r,r],
        [r,b,b,b,b,b,b,r],
        [r,b,r,r,r,r,b,r],
        [r,b,b,r,b,b,b,r],
        [r,r,r,r,r,r,r,r]]
```

Note: it is possible to draw on the LED matrix using a single list of 64 items. We have deliberately set up the maze as a two-dimensional list, because we will need to access the rows and columns of the LED matrix separately for the game. This will be much easier in a two-dimensional list.

- Display your maze on the LED matrix. To do this, **flatten** the two-dimensional list into a single one-dimensional list and then display it, like this:

```
sense.set_pixels(sum(maze,[]))
```

- Save and run your code to see the maze displayed on the LED matrix.



Add a marble

The maze needs a marble. Let's change one of the LEDs in the maze list to white to represent the marble.

- Create a variable to store the colour white. Add this code underneath the code where you defined the other colours.

Displaying a colour on the Sense HAT

- In a Python file, type in the following code:

```
from sense_hat import SenseHat

sense = SenseHat()

r = 255
g = 255
b = 255

sense.clear((r, g, b))
```

- Save and run your code. The LED matrix will then go bright white.

- The variables `r`, `g`, and `b` represent the colours red, green, and blue. Their values specify how bright each colour should be; each value can be between `0` and `255`. In the above code, the maximum value for each colour has been used, so the result is white.

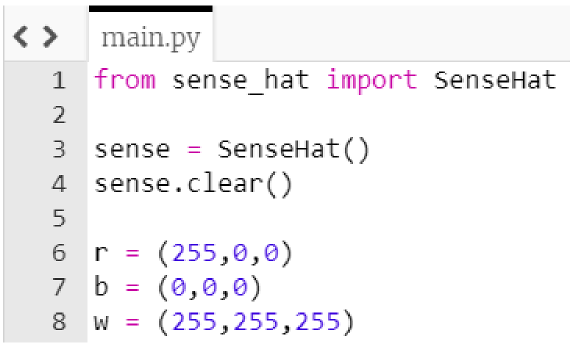
- You can also define all three RGB values of a colour using a single line of code:

```
red = (255,0,0)
```

- Change the value of one of the colours, then run the code again. What do you see?
- Which other colours can you make?

I need a hint

Here is how your code should look:



```
from sense_hat import SenseHat

sense = SenseHat()
sense.clear()

r = (255,0,0)
b = (0,0,0)
w = (255,255,255)
```

- Underneath the code for the colours, create two variables called `x` and `y` to represent the starting position of the marble. The variables `x` and `y` should both start with a value of `1`.

Creating a variable in Python

A variable allows you to store data within a program. Variables have a **name** and a **value**.

This variable has the name `animal` and the value `cat`:

```
animal = "cat"
```

This variable has the name `score` and the value `30`:

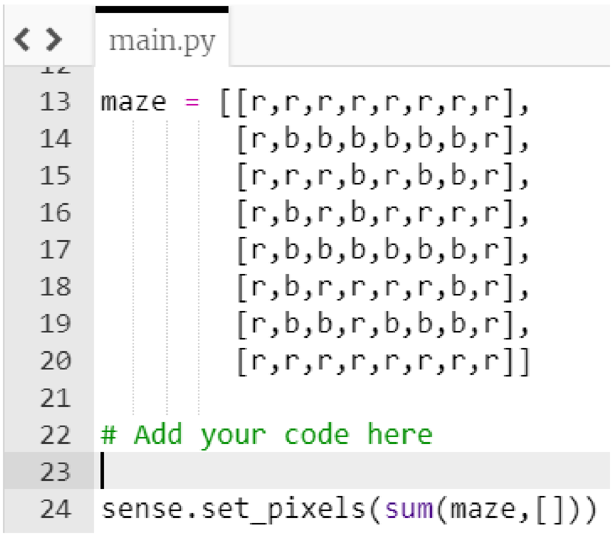
```
score = 30
```

To create a variable, give it a name and set it equal to a value. The name of the variable always goes on the left, so this code is wrong:

```
# This code is wrong
30 = score
```

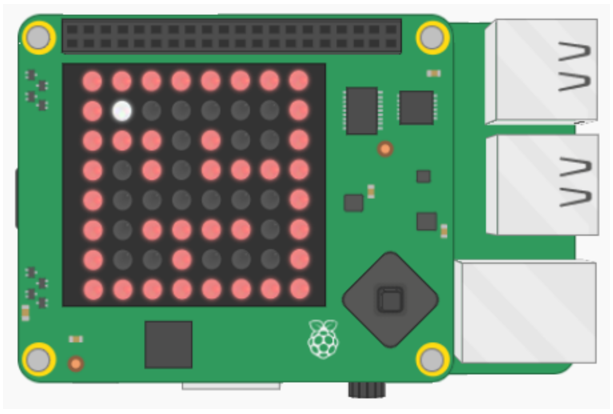
- Add some code to set the pixel in the maze at the coordinates `y, x` to the colour white you just created.

```
maze[y][x] = w
```



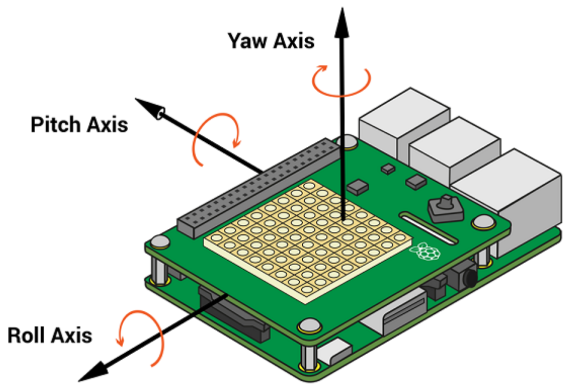
```
maze = [[r,r,r,r,r,r,r,r],
        [r,b,b,b,b,b,b,r],
        [r,r,r,b,r,b,b,r],
        [r,b,r,b,r,r,r,r],
        [r,b,b,b,b,b,b,r],
        [r,b,r,r,r,r,b,r],
        [r,b,b,r,b,b,b,r],
        [r,r,r,r,r,r,r,r]]

# Add your code here

sense.set_pixels(sum(maze,[]))
```

- Save your code and run it to see the maze and marble on the LED matrix.

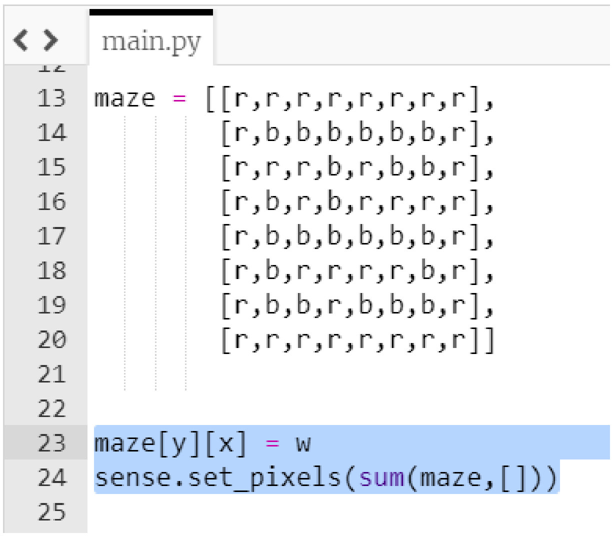


Detect pitch and roll

The marble's movement will be controlled by moving the Sense HAT. The Sense HAT library can detect pitch, roll, and yaw. You can see a picture illustrating this below.



- Locate the code that sets the position of the marble and draws the maze on the display.


main.py

```
maze = [[r,r,r,r,r,r,r,r],
        [r,b,b,b,b,b,b,r],
        [r,r,r,b,r,b,b,r],
        [r,b,r,b,r,r,r,r],
        [r,b,b,b,b,b,b,r],
        [r,b,r,r,r,r,b,r],
        [r,b,b,r,b,b,b,r],
        [r,r,r,r,r,r,r,r]]


maze[y][x] = w
sense.set_pixels(sum(maze,[]))
```

- Above this code, create a Boolean variable called `game_over` with a starting value of `False`.
- Create a while loop which runs while the `game_over` variable is `False`. Put the two highlighted lines of code inside the loop.

While loop with a Boolean in Python

The purpose of a **while** loop is to repeat code over and over while a condition is `True`. This is why while loops are sometimes referred to as **condition-controlled** loops.

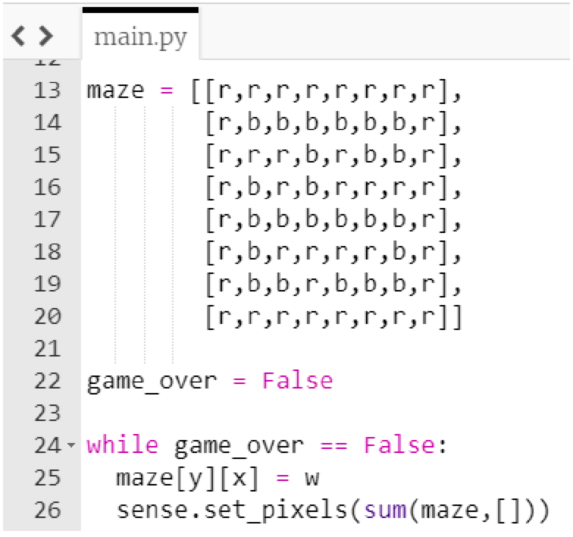
In this example, the condition is a boolean variable we gave the name `keep_looping`. Its value is set as `True` at the top. The value can become `False`, which will make the condition of the while loop `False`. When this happens, the loop stops running.

```
keep_looping = True

while keep_looping:
    print("I am in a loop")
    command = input("Shall I keep looping?")
    if command == "no":
        keep_looping = False
```

This kind of loop is useful in situations where you want to repeat code until a specific event happens. For example, you may want a program to continue running until someone types something to tell it to quit.

I need a hint

Here is how your code should look:



```python
maze = [[r,r,r,r,r,r,r,r],
        [r,b,b,b,b,b,b,r],
        [r,r,r,b,r,b,b,r],
        [r,b,r,b,r,r,r,r],
        [r,b,b,b,b,b,b,r],
        [r,b,r,r,r,r,b,r],
        [r,b,b,r,b,b,b,r],
        [r,r,r,r,r,r,r,r]]

game_over = False

while game_over == False:
  maze[y][x] = w
  sense.set_pixels(sum(maze,[]))
```

This will create a game loop allowing us to update the position of the marble when the Sense HAT is moved.

- Add some code inside the `while` loop to detect the current **pitch** and **roll** of the Sense HAT.

Detecting pitch, roll, and yaw with the Sense HAT

The Sense HAT has orientation sensors which detect pitch, roll, and yaw. Do the following to access these data.

- In a Python file, enter this code:

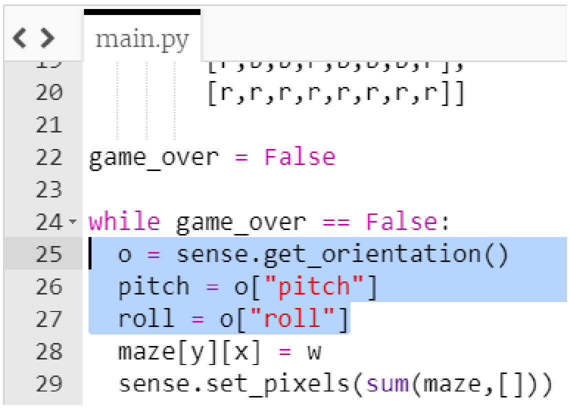
```python
from sense_hat import SenseHat
sense = SenseHat()
sense.clear()

o = sense.get_orientation()
pitch = o["pitch"]
roll = o["roll"]
yaw = o["yaw"]
print("pitch {0} roll {1} yaw {2}".format(pitch, roll, yaw))
```

- When you run the program, you should see something like this:

```
pitch 356.35723002363454 roll 303.4986602798494 yaw
339.19880231669873
```

I need a hint

Here is how your code might look:



```
              [r,r,r,r,r,r,r,r]]

game_over = False

while game_over == False:
    o = sense.get_orientation()
    pitch = o["pitch"]
    roll = o["roll"]
    maze[y][x] = w
    sense.set_pixels(sum(maze,[]))
```

If you run your program at this stage, nothing different will happen, as we need to put the pitch and roll data to work in the next step!

Move the marble

- **Above** the while loop, define a new function which will be used to move the marble.

```
def move_marble(pitch, roll, x, y):
```

The `pitch`, `roll`, `x`, and `y` variables are passed to the function as parameters.

Creating and calling functions in Python

When coding, sometimes you may want to use the same few lines of code multiple times within your script. Alternatively, you may want to have the same few lines of code run every time a certain event occurs, e.g. when a specific key is pressed, or a particular phrase is typed. For tasks like this, you might want to consider using a **function**.

Functions are **named** blocks of code that perform a defined task. Just about the simplest function you can create in Python looks like this:

```
def hello():
    print('Hello World!')
```

You tell Python that you're creating a new function by using the `def` keyword, followed by the name of the function. In this case it is called `hello`. The parentheses after the function name are important.

The colon at the end of the line indicates that the code inside the function will be indented on the next line, just like in a `for` or `while` loop or an `if/elif/else` conditional.

You can **call (run the lines of code within the function)** a function by typing its name with the parentheses included. So to run the example function, you would type `hello()`. Here's the complete program:

```
def hello():
    print('Hello World!')

hello()
```

- Inside the function, make a copy of the `x` and `y` values:

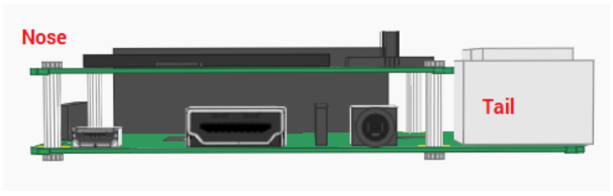
```
def move_marble(pitch, roll, x, y):
        new_x = x
        new_y = y
```

These will represent the new position of the marble after you've calculated whether it has moved based on the pitch and roll data.

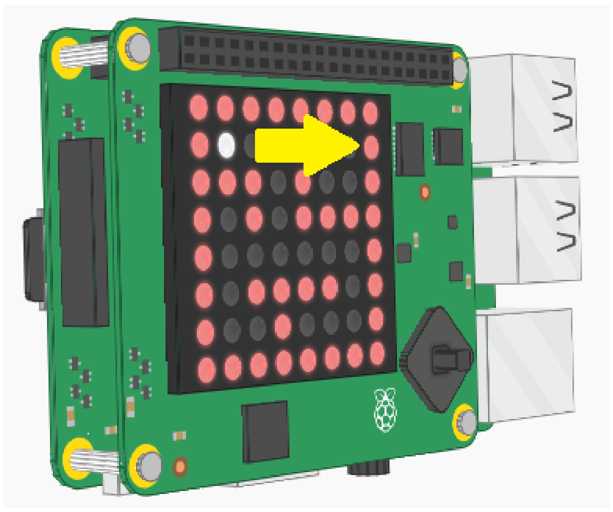
How does the pitch value change?

Imagine that the Raspberry Pi with the Sense HAT attached is an aeroplane, and the end with the USB ports is the tail of the plane.

When the Sense HAT is lying flat, the pitch should be approximately `0`.

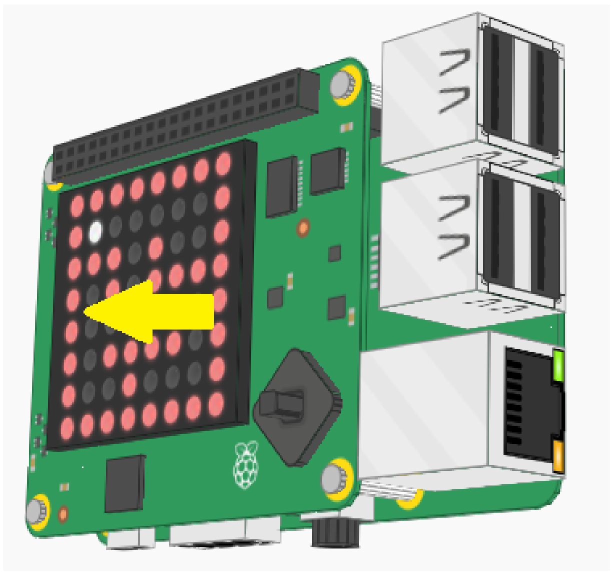


If the Sense HAT is rotated so that the nose of the 'plane' is pointing into the air (as it would at take-off), the pitch value will decrease (359,359,357,356...).



If the pitch is between `359` and `181`, then the value of `new_x` needs to increase to represent the marble moving towards the edge which is nearest the ground, marked on the diagram with a yellow arrow.

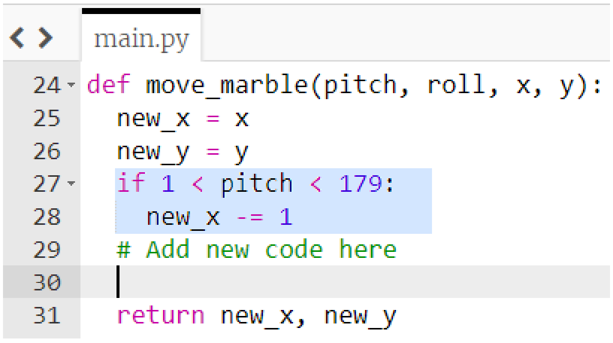
If the Sense HAT is rotated so that the nose of the 'plane' is pointing towards the ground (as it would during landing), the pitch value will increase (0,1,2,3,4...).



If the pitch is between 1 and 179, then the value of `new_x` needs to decrease to represent the marble moving the opposite way.

Coding the marble

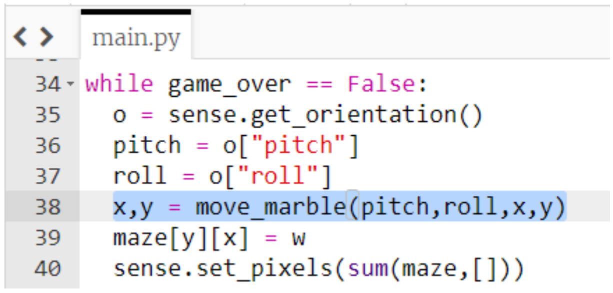
If the pitch is between 1 and 179, then the value of `new_x` should decrease by 1. Add some code within your `move_marble` function to implement this:



main.py

```python
def move_marble(pitch, roll, x, y):
  new_x = x
  new_y = y
  if 1 < pitch < 179:
    new_x -= 1
  # Add new code here

  return new_x, new_y
```

- Add more code at the location specified above so that, if the pitch is between 359 and 181, the value of `new_x` increases by 1. This will move your marble to the right.
- To test this code out, you'll need to **call the function**. Add this line of code inside the while loop.



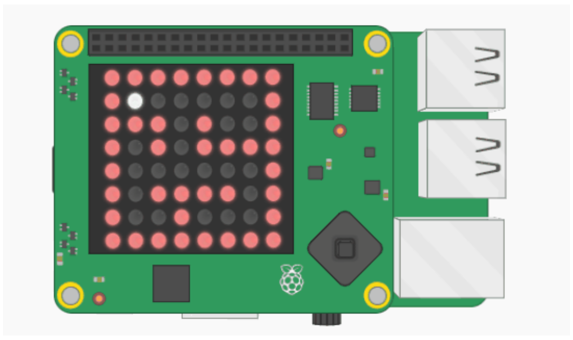
main.py

```python
while game_over == False:
  o = sense.get_orientation()
  pitch = o["pitch"]
  roll = o["roll"]
  x,y = move_marble(pitch,roll,x,y)
  maze[y][x] = w
  sense.set_pixels(sum(maze,[]))
```

- Save and run your code, then move the Sense HAT to change its pitch.

What do you see on the LED matrix?

The marble does not exactly behave as we would like!



Fix the marble

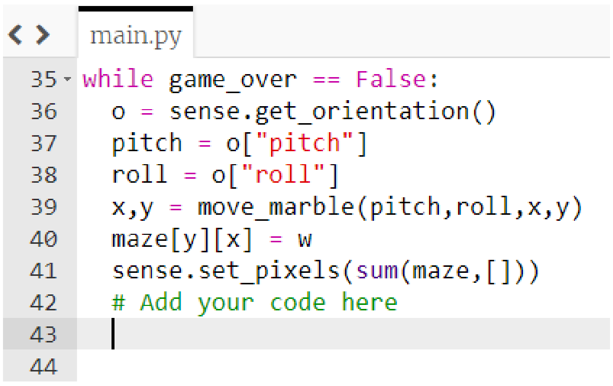
There are two problems:

- The marble is a whole line of illuminated LEDs, instead of just one LED
- The code breaks with a `IndexError: list assignment index out of range` message

Problem 1 — lots of LEDs are illuminated

The first problem occurs because, once the marble moves onto the next LED, the colour of the first LED does not change back to blank.

- Use the `sleep` function to add a pause of 0.05 seconds.



```
while game_over == False:
  o = sense.get_orientation()
  pitch = o["pitch"]
  roll = o["roll"]
  x,y = move_marble(pitch,roll,x,y)
  maze[y][x] = w
  sense.set_pixels(sum(maze,[]))
  # Add your code here

```

Using Python's sleep command

You can use the `sleep` function to temporarily pause your Python program.

- Add this line of code at the top of your program to import the `sleep` function.

```
from time import sleep
```

- Whenever you want a pause in your program, call the `sleep` function. The number in the brackets indicates how many seconds you would like the pause to be.

```
sleep(2)
```

You can pause for fractions of a second as well.

```
sleep(0.5)
```

- Then, on the line below the `sleep`, add code to tell the LED at the current `x, y` position to reset itself to blank.

I need a hint

Your code should look like this:

```
while not game_over:
    pitch = sense.get_orientation()['pitch']
    roll = sense.get_orientation()['roll']
    x,y = move_marble(pitch,roll,x,y)
    maze[y][x] = w
    sense.set_pixels(sum(maze,[]))
    sleep(0.05)
    maze[y][x] = b
```

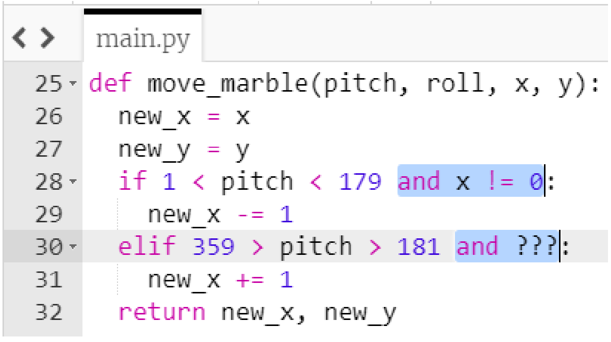
- Save and run your code. Move the Sense HAT to check that the marble looks as if it is moving.

Problem 2 — error message

This error occurs because, in the `move_marble` function, we always add or substract `1` from the value of the `new_x` variable, depending on which way the Sense HAT is tilted. Eventually this means that the value of `new_x` value will increase above `7` or decrease below `0`. As these values would be outside the boundaries of the LED matrix, we get an error.

This can be fixed by changing the code so that the values of `x` and `y` are only allowed to decrease or increase when they are **not** equal to 0 or 7.

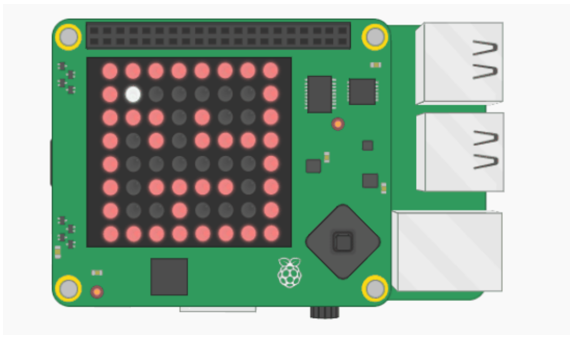
- Alter your `move_marble` function to check that adding or substracting`1` from the `new_x` value will not cause the marble to go off the edge. The first change looks like this:



```
def move_marble(pitch, roll, x, y):
  new_x = x
  new_y = y
  if 1 < pitch < 179 and x != 0:
    new_x -= 1
  elif 359 > pitch > 181 and ???:
    new_x += 1
  return new_x, new_y
```

- Save and run your code. Tilt the Sense HAT to check that the marble doesn't get stuck.

What will the result look like?



You might notice that your marble gobbles up the walls when it touches them — we'll fix that in the next step!

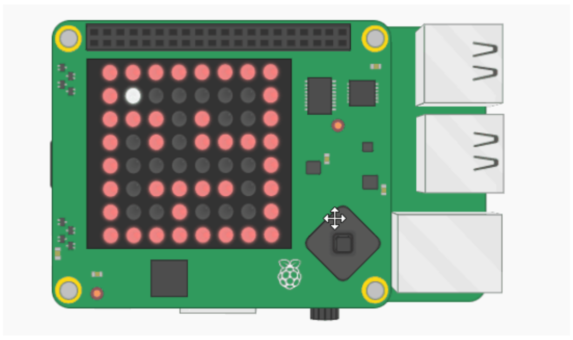
Now that you have the marble moving horizontally, you need to make it move vertically as well.

- Add some more code to the `move_marble` function so that it uses the `roll` value to move the marble up and down in the `y` direction.

I need a hint

-

What will the result look like?



Make the walls solid

You have probably noticed that when the marble moves around the maze, it deletes the walls as it goes. To prevent this from happening, you're going to need some basic **collision detection**. Write a new function for this.

- Above your while loop, add this line of code to begin a new function:

```
def check_wall(x,y,new_x,new_y):
```

- Add code inside the `check_wall` function to check whether the `new_x` or `new_y` coordinate will be inside a wall. If either of the coordinates is inside a wall, return the old `x` or `y` value for that coordinate.

I need a hint

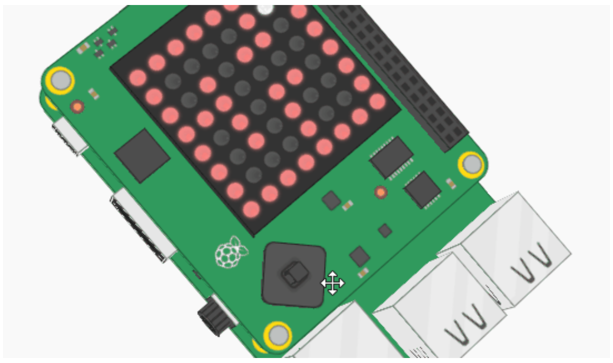
Here is how your code should look:

```
def check_wall(x,y,new_x,new_y):
    if maze[new_y][new_x] != r:
        return new_x, new_y
    elif maze[new_y][x] != r:
        return x, new_y
    elif maze[y][new_x] != r:
        return new_x, y
    else:
        return x,y
```

- Call the `check_wall` function within the `move_marble` function to decide what the `x` and `y` coordinates of the marble will be. Put this line of code immediately above the `return` line.

```
new_x, new_y = check_wall(x,y,new_x,new_y)
```

- Save and run your code. Move the Sense HAT and check that the marble now stops when it hits a wall in any direction.



Goal!

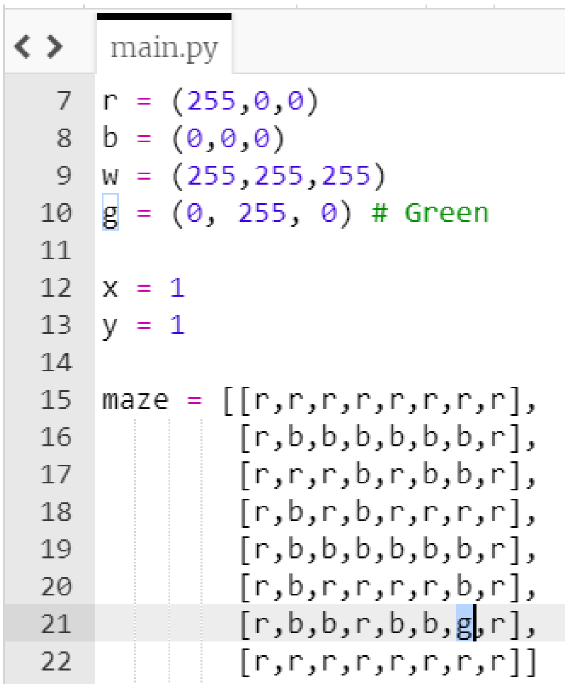
Lastly, you'll want a way for the player to win. You can pick any blank LED in the maze and set it as the target for the player to reach.

- Create a new variable to store your chosen colour for the winning marker. Put this code with the other colour variables you already created.

- Add the winning marker to the `maze` by changing a `b` in the grid to the letter representing your winning marker colour.

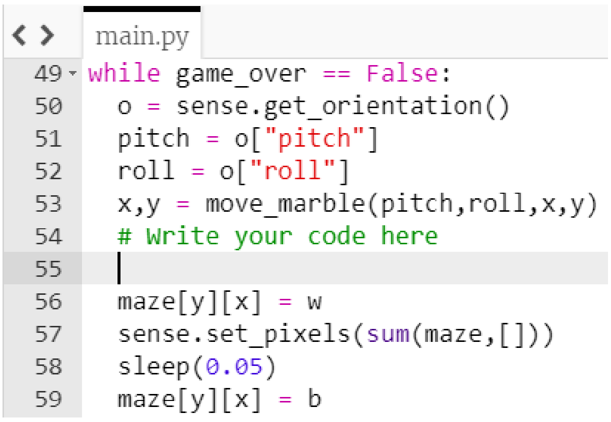
I need a hint

Here is how your code should look, with green as the winning marker colour:



```
r = (255,0,0)
b = (0,0,0)
w = (255,255,255)
g = (0, 255, 0) # Green

x = 1
y = 1

maze = [[r,r,r,r,r,r,r,r],
        [r,b,b,b,b,b,b,r],
        [r,r,r,b,r,b,b,r],
        [r,b,r,b,r,r,r,r],
        [r,b,b,b,b,b,b,r],
        [r,b,r,r,r,r,b,r],
        [r,b,b,r,b,b,g,r],
        [r,r,r,r,r,r,r,r]]
```

- Inside your while loop, once you have calculated the new `x` and `y` coordinates with the `move_marble` function, write an `if` statement to check whether the colour at this position is the winning colour.



```
while game_over == False:
  o = sense.get_orientation()
  pitch = o["pitch"]
  roll = o["roll"]
  x,y = move_marble(pitch,roll,x,y)
  # Write your code here

  maze[y][x] = w
  sense.set_pixels(sum(maze,[]))
  sleep(0.05)
  maze[y][x] = b
```

- If the colour is the winning colour, display a message saying “Win!” and set the `game_over` variable to `True` to end the game.

Show a message on the Sense HAT

Make sure you have the following lines of code in your Python program to set up your connection with the Sense HAT. There is no need to add them more than once.

```
from sense_hat import SenseHat
sense = SenseHat()
```

- Add this code to display a message on the Sense HAT’s LED matrix.

```
sense.show_message("Hello world")
```

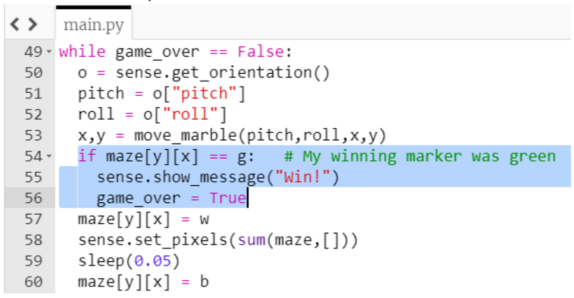
The message “Hello world” will now scroll across the LED screen.

- Change the words in the quotes (`""`) to see a different message.

You can try it out here:

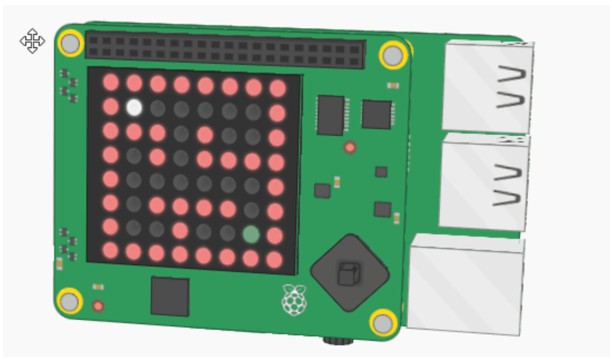
I need a hint

Here is what your code should look like:



```
main.py
while game_over == False:
  o = sense.get_orientation()
  pitch = o["pitch"]
  roll = o["roll"]
  x,y = move_marble(pitch,roll,x,y)
  if maze[y][x] == g:   # My winning marker was green
    sense.show_message("Win!")
    game_over = True
  maze[y][x] = w
  sense.set_pixels(sum(maze,[]))
  sleep(0.05)
  maze[y][x] = b
```

- Save and run your code. Move the Sense HAT around and check that when you reach the winning marker your “Win!” message is displayed.



Challenge: improve the maze

- Change the speed of the marble’s movement to make the game easier or more difficult.
- Alter the code so that the goal changes position each time the player wins a game.
- Create several mazes and choose one at random at the beginning of each round.
- Add in a second marble that starts in another part of the maze.