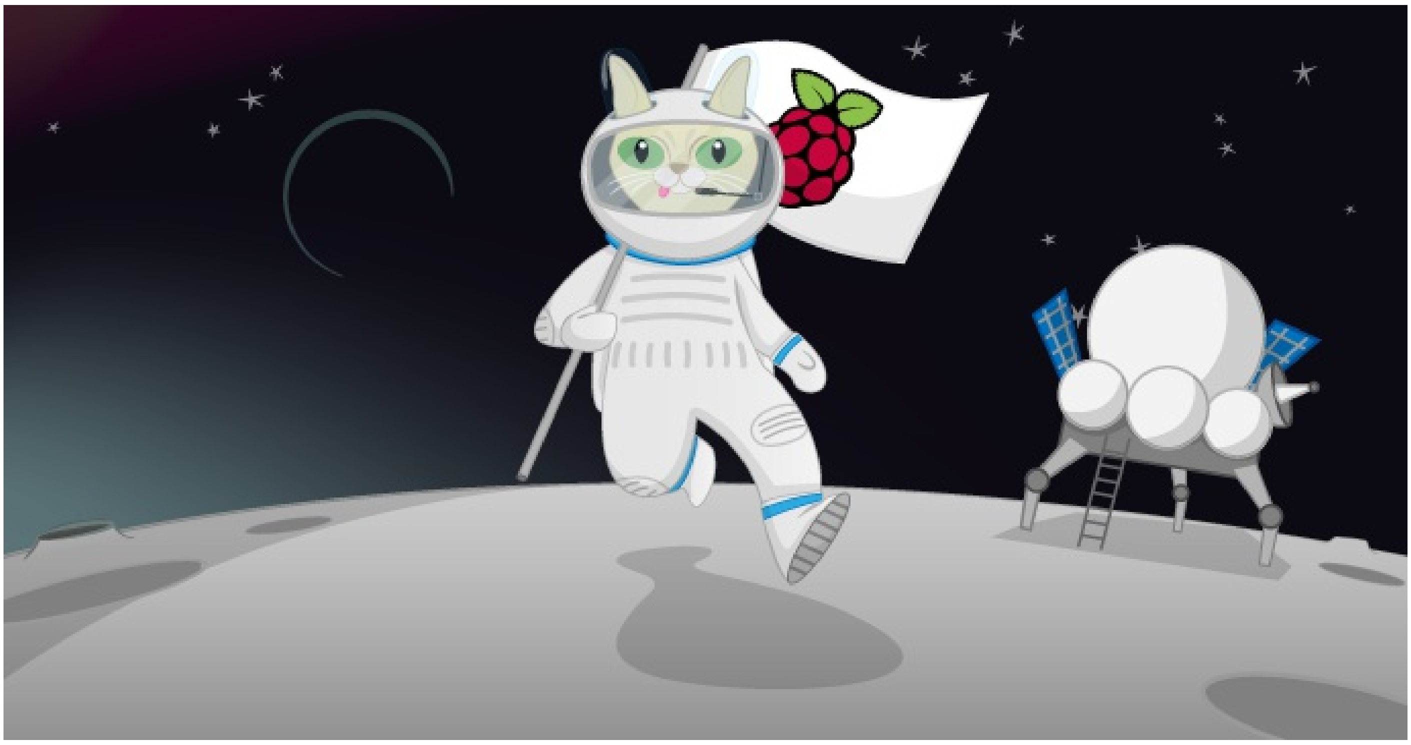
Flappy Astronaut



While astronauts are kept pretty busy while on the ISS, they still need to grab a few minutes of relaxation time every now and then. As there are two Raspberry Pis with Sense HATs up there with them, a little game of Flappy Astronaut would be the perfect way to unwind after a hard day's work in zero gravity.

In this activity you will create a Flappy Bird clone using your Raspberry Pi and a Sense HAT, along with some Python code. Flappy Astronaut uses the accelerometer in the Sense HAT to control the astronaut, by shaking the Raspberry Pi.

Setting up the Sense HAT

1. To begin with you'll need to open IDLE3 by clicking on the **Main Menu**, **Programming** and selecting **Python 3**.
2. Now create a new text file to write your code in `File>New File`.
3. You're going to need to import some modules from the `sense_hat` package to get going, so write the following three lines into your text file that will enable access to the Sense HAT and clear the LED matrix.

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

Drawing the columns

1. You can start by drawing the columns that will scroll across the LED matrix. You're going to use some loops in this game, so you'll need a way to bring the loops to a close. To achieve this, you can use a **global variable** called `game_over` to keep track of whether the game is being played or has ended.

   ```
   ## GLOBALS
   game_over = False
   ```

2. To begin with, you will make a vertical line of LEDs that scroll across the screen. This can be produced using a function called `draw_column`. The function will need to be able to change the `game_over` variable, so within the function you need to set it as a global variable.

   ```
   def draw_column():
    global game_over
   ```

3. The column is going to start on the far right of the matrix. If you look a the diagram below, you can see that this means it will have a position on the `x` axis of 7.

4. Set the starting position of the column, in the `draw_column` function.

   ```
   def draw_column():
    global game_over
    x = 7
   ```

5. Now you need to illuminate the last column of LEDs, pause for a little bit, turn off the column of LEDs and then illuminate the next column along, by reducing the value of `x` by one. This can all be done within a `while` loop, which keeps looping until the value of x gets to 0 or the game is over.

   ```
   def draw_column():
    global game_over
    x = 7
    while x >= 0 and not game_over:
   ```

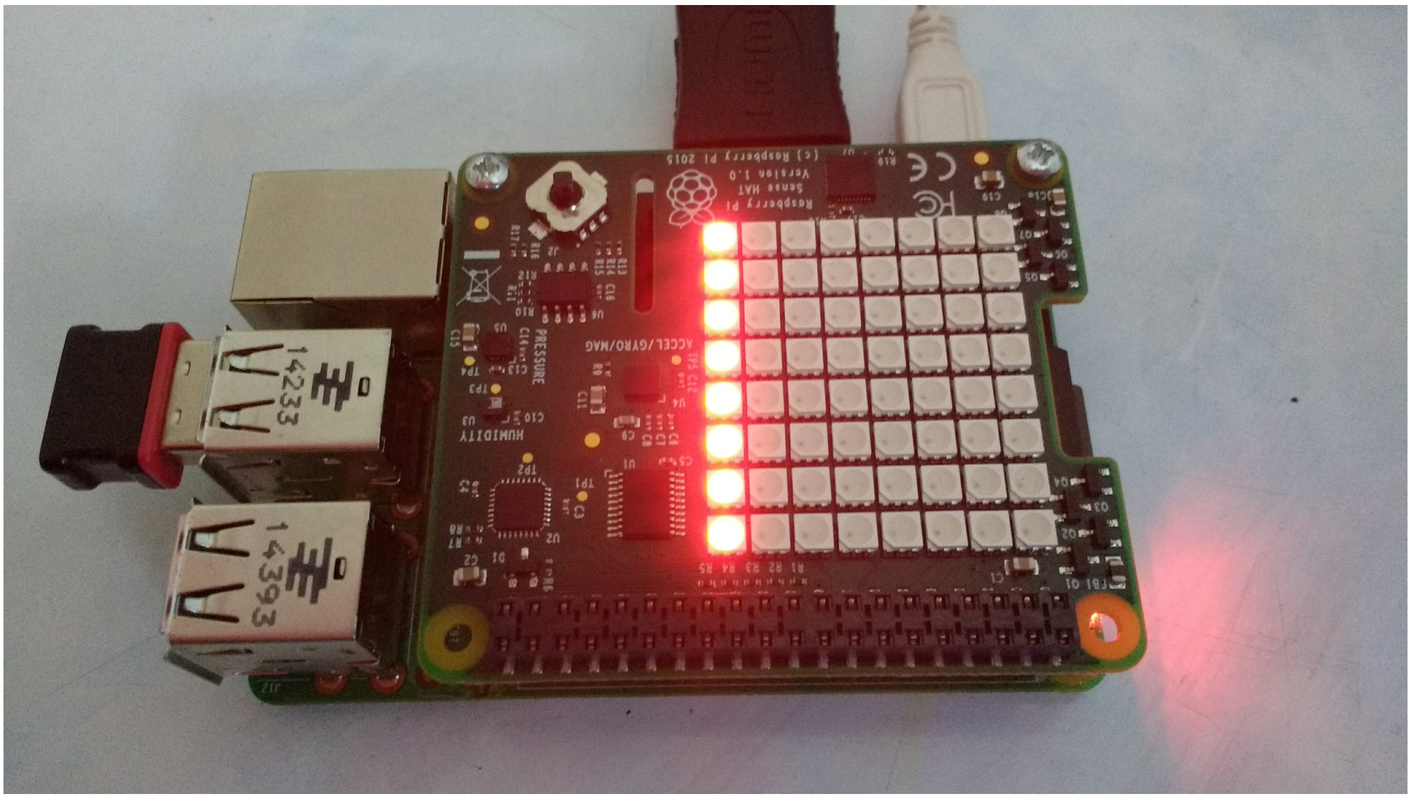
6. The column of LEDs is going to be red, and to switch them off, they're going to be changed to black. You need to specify these variables in your `##Globals` section.

```
##Globals
game_over = False
RED = (255,0,0)
BLACK = (0,0,0)
```

7. To illuminate all the pixels in a given column, you could write something like `sense.set_pixel(x,0,RED)` eight times, changing the 0 to 1, then 2, then 3, etc. However, this is simpler to do in a `for` loop.

```
def draw_column():
 global game_over
 x = 7
 while x >= 0 and not game_over:
  for led in range(8):
   sense.set_pixel(x,led,RED)
```

8. You can test out your new function, to make sure a line of LEDs are being switched on. Save your file as **flappy.py** and then press **F5** to run it. Nothing will happen at first, because you haven't called the function, so just switch over into the Python Shell and type `draw_column()`



Moving the columns

1. Next we want to switch all those LEDs off and decrease the variable `x` by one, then let the loop carry on around. Another `for` loop can be easily used to turn off all the LEDs.

```
def draw_column():
 global game_over
 x = 7
 while x >= 0 and not game_over:
  for led in range(8):
   sense.set_pixel(x,led,RED)
  for led in range(8):
   sense.set_pixel(x,led,BLACK)
  x -= 1
```

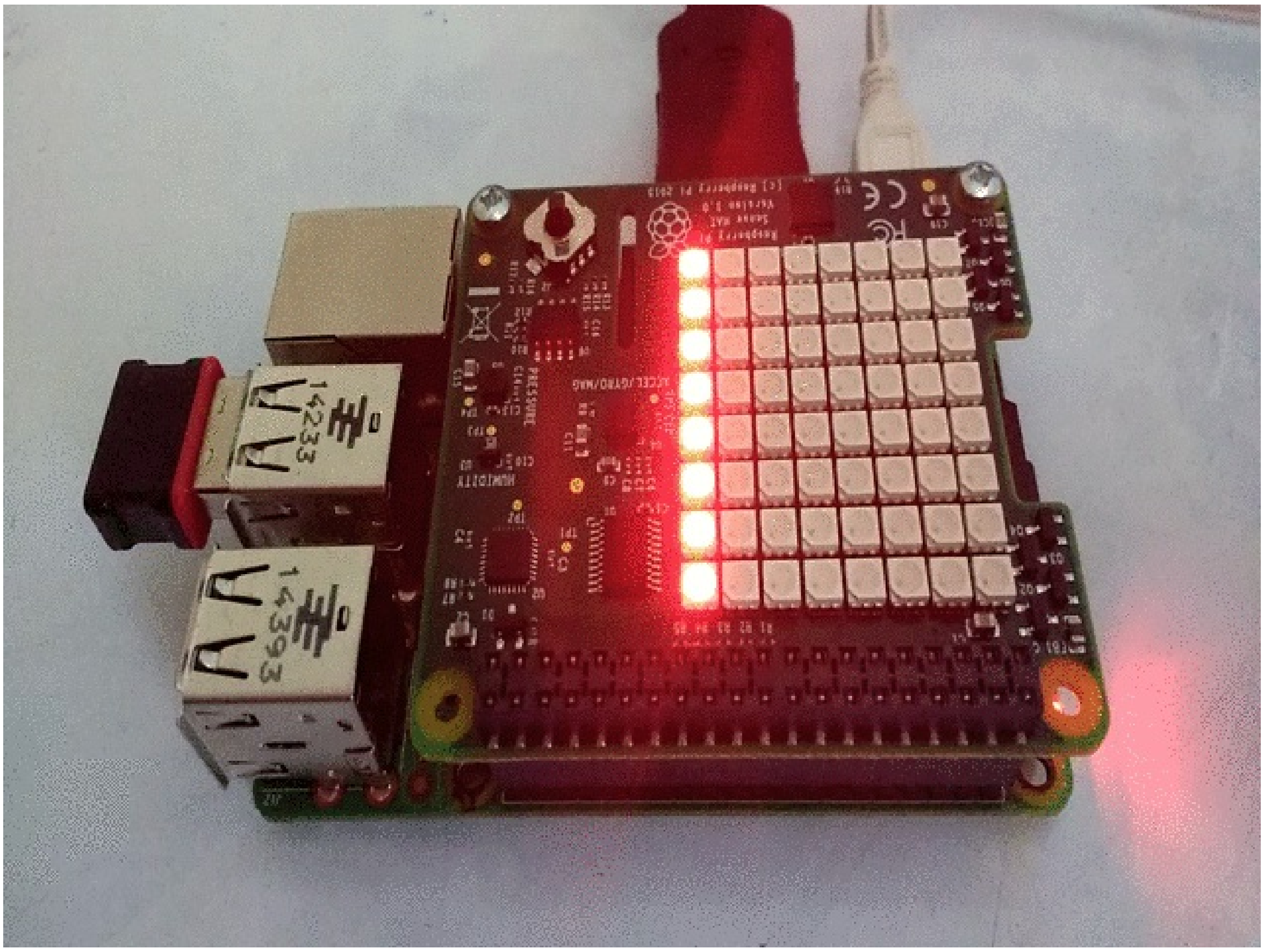
2. You can test this function out if you want, but you won't see much. The LEDs will switch on then off, so quickly that there'll be nothing to see. You need to ensure there is a pause between the LEDs lighting up and then turning off again. At the top of your file, import the `time` module

```
from time import sleep
```

3. Then add a sleep interval between switching the LEDs on an off again.

```
def draw_column():
   global game_over
   x = 7
   while x >= 0 and not game_over:
    for led in range(8):
     sense.set_pixel(x,led,RED)
    sleep(0.5)
    for led in range(8):
     sense.set_pixel(x,led,BLACK)
    x -= 1
```

4. Save your code and then press **F5** to run it. Type `draw_column()` in the Python shell to see it working.



Splitting the columns.

The game would be a little tricky if each column is a solid wall of LEDs, so you need to add a gap into the column. It would be a little too easy if the gap was always in the same place, so you'll need some randomness to its placement.

1. Add a line near the top of your file to get the `randint` function from `random`. This will generate random integers for you.

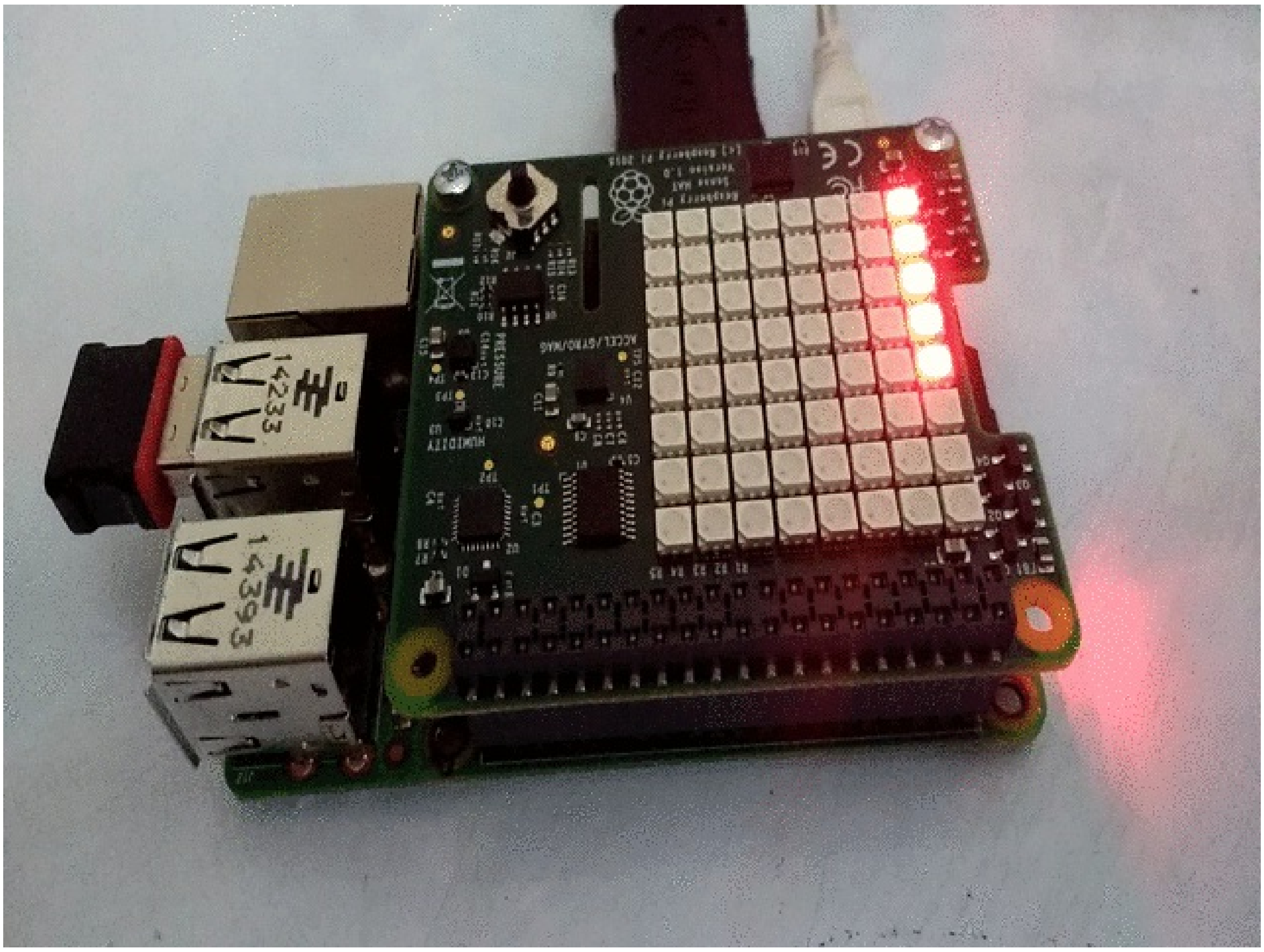
   ```
   from random import randint
   ```

2. You'll need the program to get a random integer between 1 and 6 (inclusive) and then place a gap in the line of pixels centred about that value. Don't forget, adding a gap in the column just means that you are turning off a few pixels.

   ```
   def draw_column():
    global game_over
    x = 7
    gap = randint(1,6)
   ```

```
while x >= 0 and not game_over:
 for led in range(8):
  sense.set_pixel(x,led,RED)
 sense.set_pixel(x,gap,BLACK)
 sense.set_pixel(x,gap-1,BLACK)
 sense.set_pixel(x,gap+1,BLACK)
 sleep(0.5)
 for i in range(8):
  sense.set_pixel(x,i,BLACK)
 x -= 1
```

3. Save and run your code, then type `draw_column()` into the interpreter, to check that everything is working.



Multiple Columns

1. Now that you have a single column scrolling across the matrix, you'll want keep them coming. This can be achieved with a `while` loop. Add this code to the bottom of your script.

   ```
   while not game_over:
    draw_column()
    sleep(2)
   ```

Threading

Now we have a single column scrolling across the matrix, but what if we wanted to display more than one column at a time? It would be possible to alter the `draw_column` function to produce more than one column, but it is easier to have the same function called several times.

The problem is that, at the moment, the program has to wait for a function to finish before it can be called again. It is possible to overcome this problem by using **threading**.

Threading allows you to call a function in a way that doesn't block the rest of your program, meaning that the `draw_column()` function can be called several times in a row.

1. First you'll need the `Thread` function. At the top of your program, add in a line to import it.

   ```
   from sense_hat import SenseHat
   from time import sleep
   from random import randint
   from threading import Thread
   ```

2. Now you can turn the function call in the `while` loop into a threaded function call, that will be called every two seconds

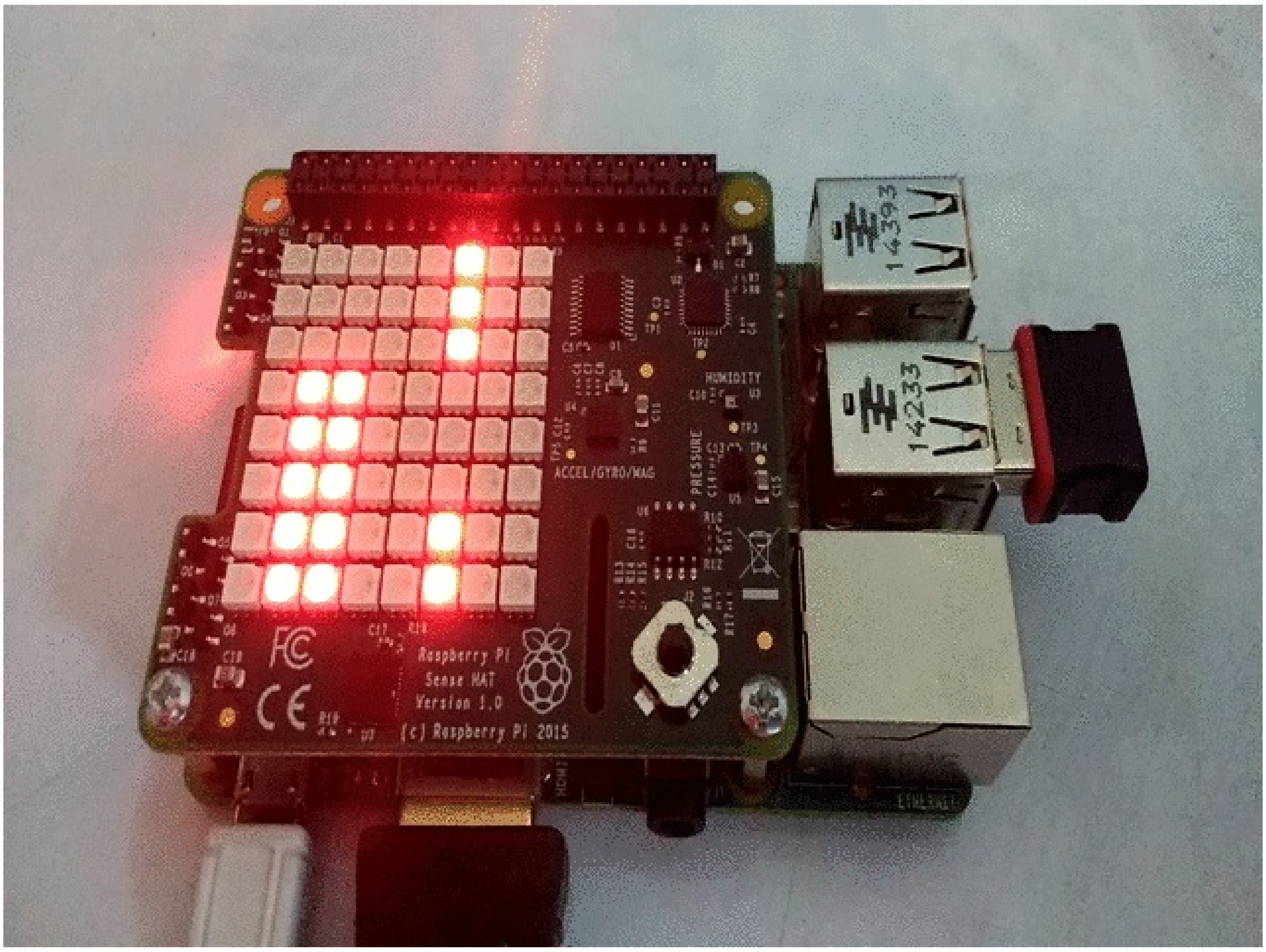
   ```
   while not game_over:
    column = Thread(target=draw_column)
    column.start()
    sleep(2)
   ```

3. The problem is that now you have this while loop blocking the program, and there is still a lot to do, such as getting some user input and moving the flappy astronaut itself up and down. The solution is to place the `while` loop into a function, and have it called as another thread. Add it to a function first:

   ```
   def draw_columns():
    while not game_over:
     column = Thread(target=draw_column)
     column.start()
     sleep(2)
   ```

4. Then call it as a threaded function.

   ```
   columns = Thread(target=draw_columns)
   columns.start()
   ```



5. Your entire code should so far look like this.

```
from sense_hat import SenseHat
from time import sleep
from random import randint
from threading import Thread

sense = SenseHat()
sense.clear()

##Globals
game_over = False
RED = (255,0,0)
BLACK = (0,0,0)

def draw_column():
 global game_over
 x = 7
 gap = randint(1,6)
 while x >= 0 and not game_over:
  for led in range(8):
   sense.set_pixel(x,led,RED)
  sense.set_pixel(x,gap,BLACK)
  sense.set_pixel(x,gap-1,BLACK)
  sense.set_pixel(x,gap+1,BLACK)
  sleep(0.5)
  for i in range(8):
   sense.set_pixel(x,i,BLACK)
  x -= 1

def draw_columns():
 while not game_over:
  column = Thread(target=draw_column)
  column.start()
  sleep(2)

columns = Thread(target=draw_columns)
columns.start()
```

6. Save **(Ctrl+s)** and run **(F5)** your program to make sure that it works.

Adding the flappy astronaut

1. The flappy astronaut will always sit horizontally on the 4th column of LEDs (position 3), but its vertical (y) values will have to change. This can be set as a global variable. As the astronaut can either be moving up or down, you can also set a global speed variable with `1` indicating that it is moving down and `-1` indicating that it is moving up. A nice blue colour would suit the astronaut as well.

   ```
   ##Globals
   game_over = False
   RED = (255,0,0)
   BLACK = (0,0,0)
   BLUE = (0,0,255)
   y = 4
   speed = +1
   ```

2. You can start just by illuminating a single LED, by adding a new `while` loop to the bottom of your script.

   ```
   while not game_over:
    sense.set_pixel(3,y,BLUE)
   ```

3. Then make it fall:

   ```
   while not game_over:
    sense.set_pixel(3,y,BLUE)
    sleep(0.1)
    sense.set_pixel(3,y,BLACK)
    y += speed
   ```

4. When you run this, your program will crash, because `y` eventually reaches a value of 8, and that is off the matrix. It's simple to fix this though.

```
while not game_over:
 sense.set_pixel(3,y,BLUE)
 sleep(0.1)
 sense.set_pixel(3,y,BLACK)
 y += speed
 if y > 7:
  y = 7
 elif y < 0:
  y = 0
```

Catching user input

1. The astronaut needs to move upwards when the Raspberry Pi and Sense HAT are shaken. To do this you'll need to catch the **accelerometer** readings from the Sense HAT. To do this you can make another threaded function. Add this after the `draw_columns` function.

```
def get_shake():
 global speed
 while not game_over:

```

2. The next step is to read the data from the accelerometer, and then round each of the values. The accelerometer detects changes in velocity (speed) in three directions: x, y, and z.

```
def get_shake():
 global speed
 while not game_over:
  accel = sense.get_accelerometer_raw()
  x = round(accel['x'])
  y = round(accel['y'])
  z = round(accel['z'])
```

3. If the Raspberry Pi and Sense HAT are motionless and sitting flat on a surface then the values should be:

   ```
   x will be 0
   y will be 0
   z will be 1
   ```

4. Note that z has a value of 1 because it is reading the gravitational pull of the Earth. If these values change (because the Pi is being shaken), then you want the speed of the astronaut to change. A simple conditional will do this.

   ```python
   def get_shake():
    global speed
    while not game_over:
     accel = sense.get_accelerometer_raw()
     x = round(accel['x'])
     y = round(accel['y'])
     z = round(accel['z'])
     if x != 0 or y != 0 or z != 1:
      speed = -1
     else:
      speed = +1
   ```

5. Then make this function threaded, by adding these two lines.

   ```python
   shake = Thread(target=get_shake)
   shake.start()
   ```

6. Save and run your code, and shake the Raspberry Pi (carefully) to see the astronaut move up and then down.

7. Your script should so far look like this:

```
from sense_hat import SenseHat
from time import sleep
from random import randint
from threading import Thread

sense = SenseHat()
sense.clear()

##Globals
game_over = False
RED = (255,0,0)
BLACK = (0,0,0)
BLUE = (0,0,255)
y = 4
speed = +1

def draw_column():
 global game_over
 x = 7
 gap = randint(2,6)
 while x >= 0 and not game_over:
  for led in range(8):
   sense.set_pixel(x,led,RED)
  sense.set_pixel(x,gap,BLACK)
  sense.set_pixel(x,gap-1,BLACK)
  sense.set_pixel(x,gap+1,BLACK)
  sleep(0.5)
  for i in range(8):
   sense.set_pixel(x,i,BLACK)
  x -= 1

def draw_columns():
 while not game_over:
  column = Thread(target=draw_column)
  column.start()
  sleep(2)

def get_shake():
 global speed
 while not game_over:
  accel = sense.get_accelerometer_raw()
```

```
   x = round(accel['x'])
   y = round(accel['y'])
   z = round(accel['z'])
   sleep(0.01)
   if x != 0 or y != 0 or z != 1:
    speed = -1
   else:
    speed = +1

columns = Thread(target=draw_columns)
columns.start()

shake = Thread(target=get_shake)
shake.start()

while not game_over:
 sense.set_pixel(3,y,BLUE)
 sleep(0.1)
 sense.set_pixel(3,y,BLACK)
 y += speed
 if y > 7:
  y = 7
 if y < 0:
  y = 0
```

Detecting a collision

1. To finish off, you need the game to end if the flappy astronaut collides with the wall. Or, to put it another way, you want the game to continue playing, as long as the flappy astronaut makes it though the gap in the column.
2. A simple function can be provided the `x` position of the columns and the position of the gap, to determine if the astronaut makes it though.

```
def collision(x,gap):
```

3. If the `x` value of the column is 3, then the column and astronaut have the same horizontal position.

   ```
   def collision(x,gap):
    if x == 3:
   ```

4. Then if the `y` position of the astronaut is between `gap-1` and `gap+1`, the astronaut has made it through the gap.

   ```
   def collision(x,gap):
    if x == 3:
     if y < gap -1 or y > gap +1:
      return True
    return False
   ```

5. This function can be called inside the `draw_column` to see if the game needs to be ended or not.

   ```
   def draw_column():
    global game_over
    x = 7
    gap = randint(2,6)
    while x > 0 and not game_over:
     for led in range(8):
      sense.set_pixel(x,led,RED)
     sense.set_pixel(x,gap,BLACK)
     sense.set_pixel(x,gap-1,BLACK)
     sense.set_pixel(x,gap+1,BLACK)
     sleep(0.5)
     for i in range(8):
      sense.set_pixel(x,i,BLACK)
     if collision(x,gap):
      game_over = True
     x -= 1
   ```

6. Test your game to see if it's working.

Finishing off

1. To finish off you need to make sure that the two threads have actually ended. You can also leave a message for the player.

```
shake.join()
columns.join()

sense.show_message("You Lose", text_colour=(255,0,0))
```

2. Have a play with your game. Your full code should look like this:

```
from sense_hat import SenseHat
from time import sleep
from random import randint
from threading import Thread

sense = SenseHat()
sense.clear()

##Globals
game_over = False
RED = (255,0,0)
BLACK = (0,0,0)
BLUE = (0,0,255)
y = 4
speed = +1


def draw_column():
 global game_over
 x = 7
 gap = randint(2,6)
 while x >= 0 and not game_over:
  for led in range(8):
   sense.set_pixel(x,led,RED)
  sense.set_pixel(x,gap,BLACK)
  sense.set_pixel(x,gap-1,BLACK)
  sense.set_pixel(x,gap+1,BLACK)
  sleep(0.5)
  for i in range(8):
   sense.set_pixel(x,i,BLACK)
  if collision(x,gap):
   game_over = True
  x -= 1


def draw_columns():
 while not game_over:
  column = Thread(target=draw_column)
  column.start()
  sleep(2)

def get_shake():
 global speed
 while not game_over:
  accel = sense.get_accelerometer_raw()
  x = round(accel['x'])
  y = round(accel['y'])
  z = round(accel['z'])
  sleep(0.01)
  if x != 0 or y != 0 or z != 1:
   speed = -1
  else:
   speed = +1

def collision(x,gap):
 if x == 3:
  if y < gap -1 or y > gap +1:
   return True
 return False


columns = Thread(target=draw_columns)
columns.start()

shake = Thread(target=get_shake)
shake.start()

while not game_over:
 sense.set_pixel(3,y,BLUE)
 sleep(0.1)
 sense.set_pixel(3,y,BLACK)
 y += speed
 if y > 7:
  y = 7
 if y < 0:
  y = 0


shake.join()
columns.join()

sense.show_message("You Lose", text_colour=(255,0,0))
```

What Next?

1. Can you play around with the variable values to make the game easier or more difficult?
2. Can you keep a score so that each time a column is successfully negotiated, the score increases by one point?
3. Can you think of other ways of controlling the astronaut? Maybe you could use the joystick or the humidity sensor?

Credits

This resource was created by the Raspberry Pi Foundation, "Flappy Astronaut" https://www.raspberrypi.org/learning/flappy-astronaut/