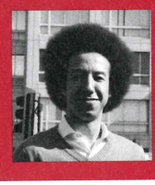
Sense HAT: Create a physical menu system

Take your Python code to the next level with some physical computing and the Raspberry Pi Sense HAT



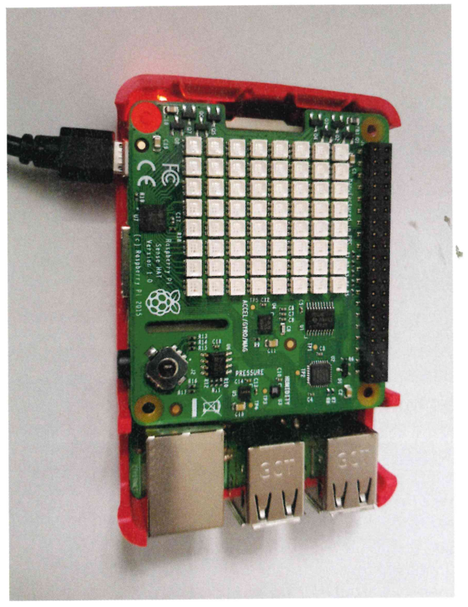
Calvin Robinson

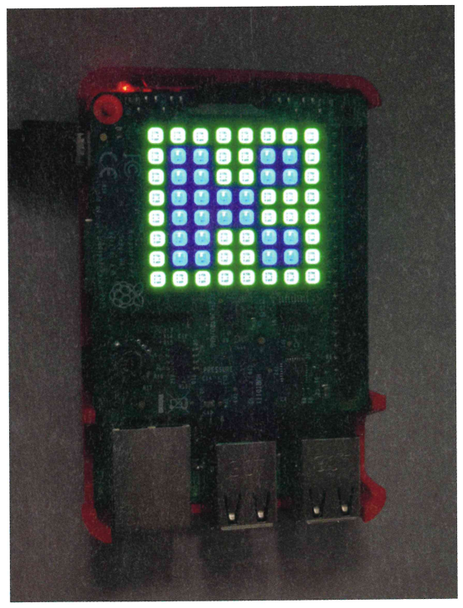
Calvin is an Associate Assistant Principal, Teaching & Learning, New Technologies at a three-campus all-through school in North West London.

Resources

- Raspberry Pi
- Sense HAT www.raspberrypi.org/products/sense-hat
- Python 3
- Sense HAT for Python 3 https://pypi.org/project/sense-hat

Tutorial files available: filesilo.co.uk





The Sense HAT is an official add-on board for the Raspberry Pi, offering a whole host of built-in sensors including a gyroscope, accelerometer and magnetometer, plus temperature, barometric pressure and humidity sensors. What we're most interested in, though, is the fancy 8x8 RGB LED matrix and five-button joystick on top. Using an array of 8x8 images we're going to create a menu system that is operable with the five-button joystick and select button.

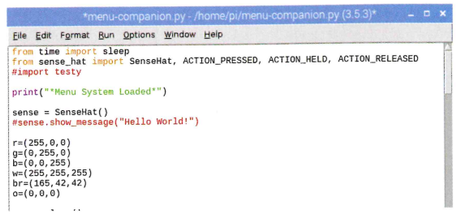
We'll start off with a simple four-option menu system where up, down, left and right analogue positions enable the user to choose from four different menu screens made up of 8x8 pixel art. Pushing in the analogue stick button will select the desired option. We'll then take a look at building on this to create potentially unlimited menu options.

This menu system can then interface with any Python program you're creating, providing a physical user interface which we think you'll agree proves to be a much more appealing user experience than entering a number in a regular Python shell window command prompt.

01 Install the HAT

When attaching the HAT to your Pi be sure to screw the standoffs in place before plugging the HAT onto the Raspberry Pi's GPIO pins; this will avoid it bending on one side and potentially breaking the pins. If for some reason you need to remove the Sense HAT at some point, be very careful as the pin header tends to come off with it. Once attached, we'll need to install the sense-hat Python module with:

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



02 Set up modules

We may want to use the sleep command for added effect when loading pixels onto the LED matrix, so we'll need to import the time module. We're also going to use a bunch of functions from the sense-hat module that we installed previously. These are mostly for recognising button-presses on the analogue stick:

```
from time import sleep
from sense_hat import SenseHat, ACTION_
PRESSED, ACTION_HELD, ACTION_RELEASED
```

This enables us to monitor whether a button is pressed (down), held or released.

03 Set up the variables

We'll need a new instance of SenseHat, which we'll call sense for simplicity. We'll also need to set up variables for each colour of pixel we want to draw on the LED matrix; we've gone with red, green and blue here, but you can use any RGB colour values:

```
sense = SenseHat()
r=(255,0,0)
g=(0,255,0)
b=(0,0,255)
```

04 Menu images - pixel art

Creating images for your menu system based on pixel art can be a challenge in itself. Try looking up some ASCII art for inspiration. Being limited to an 8x8 grid can be a blessing and curse. You'll need a different 'frame' for each menu option, four to begin with.

```
frame1 = [g,g,g,g,g,g,g,g,g,b,b,g,g,b,b,g,g,b,b,g,
g,b,b,g,g,g,g,b,b,g,g,g,g,g,g,b,b,g,g,g,g,b,b,g,
g,b,b,g,g,g,g,b,b,g,g,g,g,g,g,b,b,g,g,g,g,b,b,g,
g,b,b,g,g,g,g,b,b,g,g,g,g,g,g,b,b,g,g,g,g,b,b,g,
g,g,g,g,g,g,g,g,g,b,b,g,g,b,b,g,g,g,g,g,g,g,g,g,]
```

```
frame1 = [
g,g,g,g,g,g,g,g,
g,b,b,g,g,b,b,g,
g,b,b,g,g,b,b,g,
g,g,g,b,b,g,g,g,
g,g,g,b,b,g,g,g,
g,b,b,b,b,b,b,g,
g,b,b,b,b,b,b,g,
g,g,g,g,g,g,g,g,
]

frame2 = [
g,g,g,g,g,g,g,g,
g,w,w,g,g,w,w,g,
g,w,w,g,g,w,w,g,
g,g,g,w,w,g,g,g,
g,g,g,w,w,g,g,g,
g,w,w,w,w,w,w,g,
g,w,w,w,w,w,w,g,
g,g,g,g,g,g,g,g,
]

diamond = [
w,w,w,b,w,w,w,o,
w,w,b,b,b,w,w,o,
w,b,b,b,b,b,w,o,
b,b,b,b,b,b,b,o,
w,b,b,b,b,b,w,o,
```

Be experimental with your frames. Rather than loading the whole thing at once, the following tip will demonstrate how to slowly load a frame line by line.

05 Animating a frame

Create a variable `i=0` and add the following:

```
for i in range(8):
    sense.set_pixel(0,i,g)
    sense.set_pixel(1,i,g)
    sense.set_pixel(7,i,g)
    sleep(0.2)
    i=i+1
```

Filling in the gaps above for rows 2-6 will fill the LED matrix one row at a time, incrementally.

06 Showing off with animations

Another neat trick is the flashing frames. By using the below technique and adding more frames you can create a flick-book style animation that looks quite impressive:

```
sense.clear()
i=0
#fill column one at a time
for i in range(8):
    sense.set_pixel(0,i,g)
    sense.set_pixel(1,i,g)
    sense.set_pixel(2,i,g)
    sense.set_pixel(3,i,g)
    sense.set_pixel(4,i,g)
    sense.set_pixel(5,i,g)
    sense.set_pixel(6,i,g)
    sense.set_pixel(7,i,g)
    sleep(0.2)
    i=i+1
```

```
sense.clear()
while True:
    sense.set_pixels(frame1)
    sleep(0.5)
    sense.set_pixels(frame2)
    sleep(0.5)
```

07 Set up buttons

Our buttons will be set up in three sections of code. We'll have a function for displaying the 8x8 pixel image so that we can look at different menu options before selecting one. We'll then have a separate function for actioning a button press; when pushed/released we'll call up a specific action depending on which image is currently highlighted.

An easy way to experiment with this code without connecting your Sense HAT to the Pi is by running it on a web emulator. Trinket.io has a fantastic emulator that makes the trial and error process far quicker, because you can run the code on your favourite Linux distro. See https://trinket.io/sense-hat.

08 Create menu highlighting

When a menu function is called (upMenu,

Sense Hat: Menu system

Displaying alphanumeric digits on the Sense HAT

To display a single-character on the LED matrix use `sense.show_letter("C")` where C is any number or letter that takes your fancy. This is far simpler than creating an 8x8 frame of pixels per menu – ideal for creating a quick concept menu.

`rightMenu`, `leftMenu` or `downMenu`), we're accepting a single parameter (event) which will let us know if the button has been released. We're using 'released' rather than 'pressed' simply because it feels more natural.

If the button (in this case, 'up') is released we're printing a message to the console, mainly for debug purposes so that we know the code is working, then we're clearing the LED matrix and displaying a new pixel image from `frame1`. If the middle button is pressed we call our

```
File  Edit  Format  Run  Options  Window  Help
def downMenu(event):
  sense.clear()
  if event.action == ACTION_RELEASED:
    sense.set_pixels(snowflake)
    print("*Down option highlighted.*")
    #menu item 2
    sense.stick.direction_middle = downCreate

def downCreate(event):
  if event.action == ACTION_RELEASED:
    print("*Down option selected.*")
    #Action code goes here

def rightMenu(event):
  if event.action == ACTION_RELEASED:
    sense.clear()
    sense.set_pixels(diamond)
    print("*Right option highlighted.*")
    sense.stick.direction_middle = rightCreate

def rightCreate(event):
  if event.action == ACTION_RELEASED:
    print("*Right option selected.*")
```

second function, `upCreate`; again, we'll have a create functions for down, left and right as well.

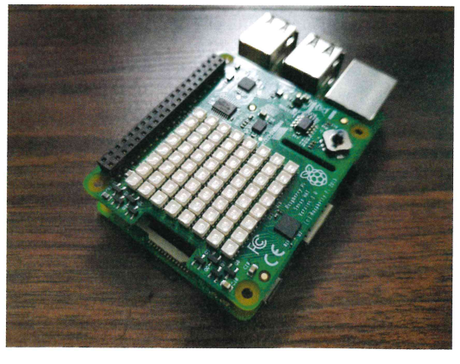
09 Create menu highlighting – the code

We'll have four instances of the following code, one for each menu option (up/down/left/right). Change the function names accordingly and insert the correct variable for the pixel art:

```
def upMenu(event):
    if event.action == ACTION_RELEASED:
        print("*Up option highlighted.*")
        sense.clear()
        sense.set_pixels(frame1)
        sense.stick.direction_middle = upCreate
```

10 Menu selection code

Very similar to the previous function, we have one parameter for a button release, we're printing to console for debugging, and then there's a space to insert whatever actionable code you require. You could



essentially put a `while` Python statement after that comment if you wanted to:

```
def upCreate(event):
    if event.action == ACTION_RELEASED:
        print("*Up option selected.*")
        #Action code goes here
```

11 Operating the buttons

Now that we've got our buttons set up to display menus and action code, we'll need to provide Python with a way of knowing if they've been pressed in the first place. Using our sense object we can listen for events `stick_direction_up`, `stick_direction_down`, `stick_direction_left` and `stick_direction_right`:

```
sense.stick.direction_up = upMenu
sense.stick.direction_down = downMenu
sense.stick.direction_right = rightMenu
sense.stick.direction_left = leftMenu
```

12 Keeping it running

In order for the above to work and to keep our programming running we'll use a loop:

```
while True:
    pass
```

Or to be more efficient and save CPU resources use:

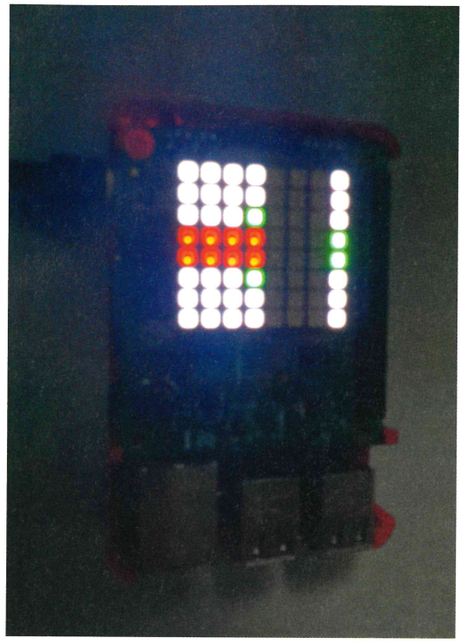
```
while True:
    time.sleep(1)
```

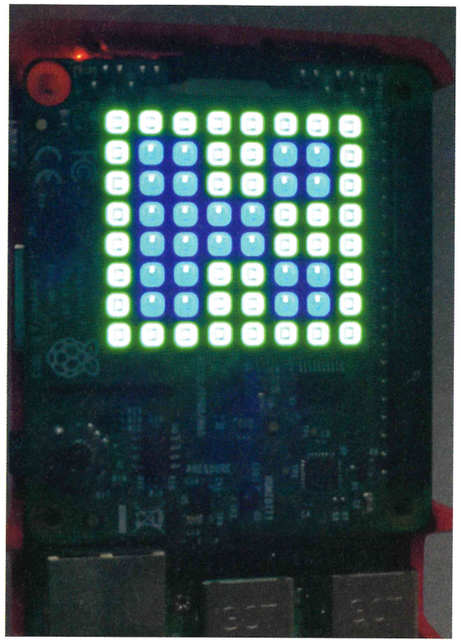
```
menu-companion-advanced.py - /h..nu-companion-advanced.py (3.5.3)
File  Edit  Format  Run  Options  Window  Help
def optionMenu(event):
  sense.clear()
  if event.action == ACTION_RELEASED:
    #sense.set_pixels(frame1)
    #menu item 1
    sense.stick.direction_right = optionMenu2
    sense.stick.direction_left = optionMenu4

def optionMenu2(event):
    if event.action == ACTION_RELEASED:
        sense.set_pixels(frame2)
        #menu item 2
        sense.stick.direction_right = optionMenu3
        sense.stick.direction_left = optionMenu

def optionMenu3(event):
    if event.action == ACTION_RELEASED:
        sense.set_pixels(frame2)
        #menu item 2
        sense.stick.direction_right = optionMenu4
        sense.stick.direction_left = optionMenu2

def optionMenu4(event):
    if event.action == ACTION_RELEASED:
        sense.set_pixels(frame2)
        #menu item 2
        sense.stick.direction_right = optionMenu
        sense.stick.direction_left = optionMenu3

sense.stick.direction_right = optionMenu
```

13 Advanced menu system

To improve this system we may want to increase the number of menu options available. Rather than using up, down, left and right to highlight menu options we could stick with left and right for back and forth by introducing scrolling. With this method our options are potentially unlimited.

We could do this by adding alternate functions to the left and right directional menus, that is by adding `sense.stick.direction_right = optionMenu3` to our





`rightMenu` function. We could add as many `optionMenu` functions as we need, progressing with the right stick and not forgetting to add `sense.stick.direction_left optionMenu2` to give users an option to go back using the left stick.

14 Call independent Python programs

We can implement any code into the actionable functions and it'll occur whenever we select that menu option. However, you might instead want to launch a completely independent Python program when certain menu options are selected.

Python's included `os` module takes care of this; if we add `import os` to the top of our code we can call upon independent Python programs in each menu option with the following:

```
os.system('testy.py')
```

where `test.py` is the name of the program.

15 Building from here

We have the option of tilting the Pi left and right to change menu options using the Sense HAT's accelerometer. The two axes we need to observe are Pitch, which is left/right, and Roll which is up/down, in-line with the analogue stick.

```
while True:
    acceleration = sense.get_accelerometer_raw()
    x = acceleration['x']
    y = acceleration['y']
```

Shake to clear

Introduce the Yaw axis to know when the Pi travels up or down vertically by reading the z acceleration with the `sense.get_accelerometer_raw` function. This would be a nice way of introducing a 'shake to clear' function, which could reset the menu screen.

Introduce `z = acceleration['z']` to the `while` loop in our penultimate tutorial step, and don't forget to round the values with `z=round(z, 0)`. Then we would need an `if` statement along the lines of `if z == 1: clearMenu`, provided `clearMenu` was a function set to clear the screen with `sense.clear()`. Then listen for further menu selections with either the analogue stick or the Roll/Pitch axis.

16 Shaking/rolling

We'll need to round off the accelerometer values into floats to make them useable; `x=round(x, 0)` and `y=round(y, 0)` will do the trick. We can now use an `if` statement for `x` or `y` being equal to 1 or -1 for left or right.

```
if x == 1:
elif x == -1:
```

Implement a call to an `optionMenu` function in these `if`/`elif` statements and we have a functioning tilt menu.