# Project: Board for Snakes and Ladders

### Difficulty level: Medium/Hard

## Description

In this project you will create a *Board* for the classic board game snakes and ladders.

You can try out the complete version of the project here:


Use the up, left, right and left arrows to move the brush. Press the middle button (ENTER) to paint a pixel.

## Project Manual

This project guide will tell you step-by-step the main things you have to do in order to create a *countdown clock*. For some of the steps, you'll have to use your own creativity to proceed, good luck!

---

## Introducing the project

The first thing you should do is open the *skeleton code* for the project.
In programming, skeleton code means code that only has the basic elements of a program. It is up to you to fill in the rest!

You can find the skeleton code on Trinket, here:
https://goo.gl/rXr3vQ (https://goo.gl/rXr3vQ)

**If you can, you should also create and account and log in to Trinket. This will allow you to save the Trinket projects. Otherwise you have to copy the code on to your computer to save it.**

On Trinket, you'll be able to test your code on a *virtual* Sense HAT, before you try your code on the real thing.

As you might see, the skeleton code is split up into sections, divided by the headlines.
For example:

```
#### 2. Code section
```

This guide will go through the various sections (not necessarily in order), and help you write your code. **Very important note:** *You should add the code in the specified section in your skeleton code as you follow this guide.*
The next part of this guide will explain the stuff that's already in the skeleton code when you first open it.

#### Explanation of the skeleton code

Before we get on to the coding, it's worth looking over the *skeleton code* and make sure you are familiar with it.

The first few lines in the script are:

```
#### 1.1 Import libraries

import sys
sys.path.insert(1,'/home/pi/Go4Code/g4cSense/skeleton')
sys.path.insert(1,'/home/pi/Go4Code/g4cSense/graphics')
from sense_hat import SenseHat
import random
from senselib import *
```

Without going into detail, these lines are called *import statements*. They are used to *import* code from other Python files into your own file. This is useful because you can use other people's code to simplify your own.

The next part of the code (Sec. 1.2) creates some important *Objects* (don't worry if you're not sure what that means) that we'll use in the later on.

Sec. 1.3 is where the initial variables of the project are set up.
You'll need to edit these values later on in order to customize the it.

Sec. 2 is where all the main coding will take place.

Sec. 2.1 is where we'll write code that controls the *brush*, that places paint on the screen.

Sec. 2.2 is where we'll write code that changes the colour of the brush.

---

## Writing the code

#### (Sec. 1.3) Set up the variables

In this section we will define some variables which we will use later on.

The starting position of the paint brush is in the variables *cursorx* and *cursory*.

The possible paint brush colours are stored in the list *colours*. Your first job will be to add some colours to this list. As you might remember, in programming a colour is usually represented as *three* numbers *[r, g, b]*. Where the first number is how red the colour is, the second how green it is, the third how blue it is.
For example, to add the colours red, green and blue to the available brush colours:

```
colours = [
      [255, 0, 0],
      [0, 255, 0]
      [0, 0, 255]
    ]
```

The final variable is *currentColour*. This variable is a bit trickier to understand. currentColour is a number that decides which colour the paint brush is currently drawing. For example, if we set the *colours* list to what was shown above:

- *currentColour = 0* would mean the brush is red.
- *currentColour = 1* would mean the brush is green.
- *currentColour = 2* would mean the brush is blue.

Later on we'll let the user change the brush colour by shaking the Sense HAT, this will be achieved by randomly changing the value of *currentColour*.

#### (Sec. 2) Main program code

Before we start coding, we'll explain a bit what's going on in Sec. 2.

In this part of the code, we'll do all the main programming.
If we look at the skeleton code, we see that the section is within a *while*-loop. Remember that code inside a while loop runs again and again, until we tell it to stop. The reason why we want our program to be in a loop is because we want to continually check if the user has shaken the device.

Because the code is in the *while*-loop, remember that you have to add a *Tab* at every line, to make it indented. Like this:

```
while True:
    # write your code like this,
    # with a tab at the start of the line.
```

#### (Sec 2.1) Control the paint cursor and place colours on the screen

Inside the while-loop, you should write the following:

```
for event in sense.stick.get_events():
    if event.action == "pressed":

        if event.direction == "up":
            # Fill in with your own code

        elif event.direction == "down":
            # Fill in with your own code

        elif event.direction == "left":
            # Fill in with your own code

        elif event.direction == "right":
            # Fill in with your own code

        elif event.direction == "middle":
            # Fill in with your own code
```

This code will check if the user has pressed the *up*, *down*,*left*, *right* or the *middle* buttons on the joystick. Don't mind the complicated structure of the code above, try to figure out how to use the code by looking at it.

Inside the *if*-statements ou'll find the comments *# Fill in with your own code*. This is where you should write your code.

If the user presses *up*, *down*,*left* or *right* on the joystick, the variables *cursorx* and *cursory* should be changed. We'll give you a hint to start off with: If the user presses the up button, cursory should *decrease* by 1.

*Bonus: One thing that you might want to add is that if the user pushes the brush off the screen, it should pop back up on the other side of the screen. You can do this using further if-statements.*

Finally, if the user presses the *middle* button, the brush should paint a pixel at the position of the brush. Remember that the available brush colours are in the variable *colours*, and the current brush colour is given by *currentColour*. So you can get the current colour, and then draw it on the screen, using:

```
c = colours[currentColour]
sense.set_pixel(cursorx, cursory, c[0], c[1], c[2])
```

### 2.2 Change colour by shaking

**This part is a bit complicated, so pay attention! Don't be afraid to ask a supervisor if you don't understand something.**

To see if the user has shaken the Sense HAT, we'll have to use the *accelerometer* inside the device. The accelerometer basically tells us how fast the device has been accelerating. To get the information from the accelerator, use the following code:

```
ac = sense.get_accelerometer_raw()
x = ac["x"]
y = ac["y"]
z = ac["z"]
```

Now, in the *x*, *y* and *z* variables, we have information recorded by the accelerometer. This is all a bit complicated, but the basic gist of it is this, calculate this:

```
shake = x*x + y*y + z*z
```

We have stored *x times x plus y times y plus z times z* into the variable *shake*. Now, the larger the variable *shake* is, the more has the Sense HAT been shaken.
We're going to say that if *shake* is *larger* than *5*, the Sense HAT has been shaken. To do this, you're going to have to use an *if*-statement.

If you have detected that the Sense HAT has been shaken (in other words, if *shake* is larger than 5) you want to randomly change the current brush colour.

Remember that the current brush colour is given by *currentColour*. We can generate random numbers using the function *random.randint*. You can read about in the *Function Reference* document. If we have 5 different colours to choose from, we should randomly assign *currentColour* with a number from 0 to 4 (in Python, we usually start counting from 0, rather than from 1).

To get the length of a list (like *colours*), you can use the *len* function. You can use this to generate a new brush colour every time the user has shaken the Sense HAT:

```
currentColour = random.randint(0, len(colours) - 1)
```

**Finished!**

If it's all done, correctly, the game should now work! Don't worry if it doesn't, things often go wrong in programming. Errors in code are usually called *bugs*. If you have a bug in your code, you'll have to *debug*it!

If it works, congratulations! You can either move on to another project or try to come up with new things to add to the current project. Use your creativity! You can discuss any ideas you have with a supervisor.

---

**Bonus: Add a visible cursor on the screen**

Currently, when you draw on the screen, you can't see the brush as you move it. If you want a real challenge, figure out how you can have the brush always be visible on the screen.

The main issue that complicates this is that you don't want the brush leaving a trail as you move it on the screen. You'll have to somehow *save* how the screen looks before you draw the brush, and then when you move the brush again, restore that picture. It's a bit confusing, so try reading it again or discuss it with a supervisor.

Two functions that you'll have to use is the *sense.get_pixels* and *sense.set_pixels* functions (note that the latter function is *sense.set_pixel*, but with an *s* at the end). The first function saves the pixels on the screen into a variable, and the second one can redraw those pixels at a later stage.

```
image = sense.get_pixels() # Save the pixels on the screen

sense.get_pixel(1, 1, 255, 255, 55) # Draw some stuff
sense.get_pixel(7, 3, 77, 34, 0) # Draw some stuff
sense.get_pixel(2, 5, 22, 222, 111) # Draw some stuff

sense.set_pixels(image) # The screen is as it was from the beginning!
```

---

**Author:** Lukas Kikuchi

**Date:** August 09, 2017